ÍNDICE GENERAL

INTRODUCCIÓN

Desde la invención del circuito integrado, el desarrollo constante de la electrónica digital ha dado lugar a dispositivos cada vez más complejos; entre ellos, los microprocesadores y los microcontroladores, los cuales son básicos e indispensables en nuestro mundo actual. El microcontrolador es uno de los logros más sobresalientes del siglo XX. Año tras año, se ha ido incorporando en más elementos de nuestra vida cotidiana y muchos aparatos y maquinaria del día a día no funcionarían sin estos pequeños microchips. Los hornos microondas, lavavajillas y otros aparatos electrodomésticos funcionan gracias a ellos. También se encuentran presentes en los televisores, videos, aviones, vehículos, etc.

Con este Proyecto Fin de Carrera se pone en práctica una aplicación de un microcontrolador: el desarrollo de un ***Mando a Distancia Universal por Infrarrojos Reprogramable***. Para ello, se ha empleado el microcontrolador PIC16F84A fabricado por Microchip Technology. La elección de un "PIC" no se debe a una razón en especial. Si hay que reseñar que esta familia de microcontroladores está "fascinando" a los diseñadores en la actualidad. Puede ser la velocidad, el precio, la facilidad de uso, la información, las herramientas de apoyo,... Lo que sí parece cierto es que, todas estas características en conjunto producen una imagen de sencillez y utilidad que hacen que los diseñadores se decanten hoy en día a emplear estos microcontroladores.

Para implementar este Mando a Distancia Universal se han utilizado los siguientes elementos:

- Un microcontrolador PIC16F84A, en el que reside el programa.
- Una memoria EEPROM de 2 Kbytes, en la que se almacena los datos necesarios para poder reproducir la trama muestreada.
- Un diodo emisor de infrarrojos con el que se emite la trama reproducida.
- Un fotodiodo receptor para poder capturar la trama.
- Un diodo led visible rojo.
- Otros elementos como son resistencias, condensadores y transistores.

La idea principal del funcionamiento de este mando es la siguiente: se enfrenta el prototipo con el mando del cual queremos muestrear la trama, y, después de pulsar una combinación de teclas determinada, comienza a recoger una serie de datos de la trama del mando emisor que graba en la memoria EEPROM y que le permitirán reproducir el conjunto de bits recibidos.

Es decir, este mando a distancia será capaz de:

- Muestrear y grabar una trama o conjunto de bits que emite un segundo mando a distancia. La trama podrá cumplir cualquiera de los protocolos estándar establecidos internacionalmente: *RC5* y *RECS80*.

- Reproducir fielmente la trama muestreada, de forma que sea capaz de manejar el mismo dispositivo que el mando original.

Durante el desarrollo de este Proyecto Fin de Carrera se han seguido varias etapas.

- En un primer momento, se hizo una toma de contacto y estudio del funcionamiento del microcontrolador PIC16F84A, aprendiendo a manejar el lenguaje de programación así como familiarizarse con las herramientas y aparatos de programación del mismo.

- A continuación, se desarrolló el programa que rige el funcionamiento del mando a distancia en leguaje ensamblador, dividido en tres etapas: lectura del teclado, muestreo/reproducción de la trama y grabación/lectura de los datos de la memoria EEPROM.

- Finalmente, se procedió al diseño e implementación del hardware del mando a distancia.

Los elementos empleados han sido:

- PC Pentium II con 64 Mbytes de memoria RAM
- Programa MPLAB V12.0 para la escritura y compilación del programa.
- Programa Orcad 9.1.
- Programador Universal Dataman 48.
- Analizador Lógico LA4000.
- Máquina fresadora.

La memoria se estructura en nueve capítulos a lo largo de los cuales se explica el trabajo desarrollado. En el primer capítulo se realiza una introducción sobre los microcontroladores y aplicaciones de los mismos. En el segundo se habla de la familia de los PIC, así como de las características y componentes del microcontrolador PIC16F84A, y será en el tercero donde se muestre un pequeño manual de usuario de la herramienta de programación MPLAB. Posteriormente, se abre una serie de dos capítulos en los que se desarrolla el protocolo de emisión de trama que emplean los mandos a distancia y el protocolo de comunicación entre la memoria y el microcontrolador (capítulo cuatro y cinco respectivamente). Será en el sexto y en el séptimo donde se mostrará la implementación hardware del mando a distancia reprogramable y se dará una explicación del programa desarrollado para dicha aplicación.

Por último, nos podemos encontrar las conclusiones y un apéndice que engloba el layout e imágenes del prototipo, unas estimaciones sobre el coste de diseño y de producción y las hojas de especificaciones de los principales componentes empleados, así como el programa en lenguaje ensamblador desarrollado en este Proyecto Fin de Carrera.

CAPITULO 1
CONCEPTOS GENERALES. EL MICROCONTROLADOR

1.1¿QUÉ ES UN MICROCONTROLADOR?

Recibe el nombre de ***controlador*** el dispositivo que se emplea para el gobierno de uno o varios procesos. Aunque el concepto de controlador ha permanecido invariable en el tiempo, su implementación física ha cambiado notablemente. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta; posteriormente, se emplearon los microprocesadores, que se rodeaban de un chip de memoria y de líneas E/S sobre una tarjeta de circuito impreso. Fue entonces cuando hacia el año 1980 los fabricantes de circuitos integrados iniciaron la difusión de un nuevo circuito para control, medición e instrumentación al que llamaron microcomputador en un sólo chip o de manera más exacta ***microcontrolador.***

Un **microcontrolador** es un circuito integrado programable de alta densidad de integración, capaz de ejecutar las órdenes o secuencias que se encuentran grabadas en su memoria. Contiene todos los componentes o bloques funcionales de un computador y se emplea para controlar el funcionamiento de una tarea determinada. Debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna.

El microcontrolador es un computador dedicado. En su memoria reside un único programa destinado a gobernar una aplicación determinada. Sus líneas de E/S soportan la conexión de los sensores y actuadores del dispositivo a controlar y todos los recursos complementarios disponibles tienen como única finalidad atender sus peticiones. Una vez programado y configurado el microcontrolador, solamente podrá emplearse para realizar la tarea asignada.

Los productos que para su regulación incorporan un microcontrolador disponen de ventajas como:

- ***Aumento de fiabilidad.-*** Al reemplazar el microcontrolador a un elevado número de elementos, disminuye el riesgo de averías y se precisan menos calibraciones.
- ***Reducción de tamaño en el producto acabado.-*** La integración del microcontrolador en un chip disminuye el volumen, la mano de obra y los stocks.
- ***Mayor flexibilidad.-*** Dado que las características de control están programadas, su modificación sólo precisa cambios en el programa de instrucciones. Esto supone una fácil adaptación a las circunstancias que rodean al producto final, junto a una gran rapidez en la implementación de posibles cambios.

Hoy en día, el número de productos que funcionan en base a uno o varios microcontroladores aumenta de forma exponencial y no parece descabellado pronosticar que en un futuro próximo habrá pocos elementos que carezcan de un microcontrolador. Como ejemplos de aplicación de microcontroladores podemos citar los siguientes:

- La industria informática acapara gran parte de los microcontroladores que se fabrican. Casi todos los periféricos del computador, desde el ratón o el teclado hasta la impresora, son regulados por el programa de un microcontrolador.

- Los electrodomésticos como lavadoras, hornos, televisores, vídeos,... incorporan numerosos microcontroladores. Igualmente, los sistemas de supervisión, vigilancia y alarma en los edificios utilizan estos chips para optimizar el rendimiento de ascensores, calefacción, aire acondicionado, alarmas de incendio, robo, etc.
- Las comunicaciones y sus sistemas de transferencia de información utilizan en gran número estos pequeños computadores incorporándolos en los grandes automatismos y en los teléfonos.
- La instrumentación y la electromedicina son dos campos idóneos para la implantación de estos circuitos integrados.
- Una importante industria consumidora de microcontroladores es la del sector automovilístico, que los aplica en el control de aspectos tan populares como la climatización, la seguridad y los frenos.

1.2 ARQUITECTURA INTERNA DE UN MICROCONTROLADOR

Todos los microcontroladores poseen una estructura fundamental y sus características básicas son parecidas. Los principales componentes de los que consta son:

- Procesador
- Memoria no volátil que contiene el programa
- Memoria de lectura y escritura para guardar los datos
- Líneas de E/S para controlar periféricos
  - Comunicación paralelo
  - Comunicación serie
  - Diversos puertos de comunicación (bus I^2C, USB, etc.)
- Reloj principal
- Recursos auxiliares
  - Temporizadores
  - Perro Guardián ("Watchdog")
  - Conversores A/D y D/A
  - Comparadores analógicos
  - Protección ante fallos de alimentación
  - Estado de reposo o de bajo consumo

Según el modelo de microcontrolador que se trate, tanto el tamaño como el tipo de memoria pueden diferir, así como el número de líneas de E/S y los módulos de control de

periféricos. La diversidad de modelos permite seleccionar el más adecuado según la aplicación de que se trate. Una estructura interna fija supone una limitación, que se convierte en una ventaja en el caso de que en un simple circuito integrado residan todos los componentes que necesita el controlador.

Entre los fabricantes de microcontroladores hay dos tendencias para resolver las demandas de los usuarios:

Microcontroladores de arquitectura cerrada

Cada modelo se construye con una CPU determinada, cierta capacidad de memoria de datos y de memoria de instrucciones, un número de líneas de E/S y un conjunto de recursos auxiliares muy concreto. El modelo no admite variaciones ni ampliaciones. La aplicación a la que se destina debe encontrar en su estructura todo lo que precisa y, en caso contrario, hay que desecharlo.

Microcontroladores de arquitectura abierta

Estos microcontroladores se caracterizan porque además de disponer de una estructura interna determinada, pueden emplear sus líneas de E/S para sacar al exterior los buses de datos, de direcciones y de control, con lo que se posibilita la ampliación de la memoria y las líneas de E/S con circuitos integrados externos.

A continuación examinaremos las características más representativas de cada uno de los componentes de un microcontrolador.

1.2.1 Arquitectura básica

La arquitectura tradicional de computadoras y microprocesadores se basaba en el esquema propuesto por ***John Von Neumann***, el cual se caracteriza por disponer de una única memoria principal donde se almacena datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de bus único (direcciones, datos y control).

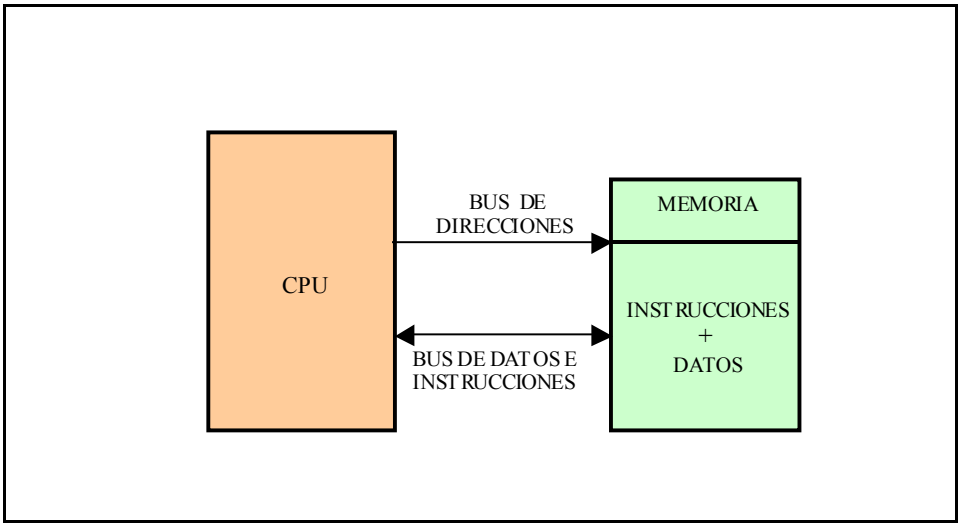


Figura 1.1. *Arquitectura Von Neumann*

Esto tiene como consecuencia dos hechos fundamentales, que a su vez suponen dos limitaciones:

- El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus de la memoria; es decir, un microprocesador de 8 bits sólo podrá manejar datos e instrucciones de 8 bits de longitud. Cuando necesite acceder a una instrucción o dato de más de un byte de longitud, deberá realizar más de un acceso a memoria.
- La velocidad de operación del microcontrolador está limitada por la existencia de un bus único para datos e instrucciones, ya que no se puede buscar en la memoria una nueva instrucción antes de que finalicen las transferencias de datos que pudieran resultar de la instrucción anterior.

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

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en la actualidad se impone la ***arquitectura Harvard***. Esta arquitectura dispone de dos memorias independientes: Una, que contiene sólo instrucciones (***Memoria de Programa***) y otra, sólo de datos (***Memoria de Datos***). Ambas disponen de sus respectivos buses y es posible realizar operaciones de acceso (lectura o escritura simultáneamente) en ambas memorias; es decir, la CPU puede estar accediendo a los datos para completar la ejecución de una instrucción y al mismo tiempo estar leyendo la próxima instrucción a ejecutar. Además, el tamaño del bus de acceso a la memoria de datos puede ser distinto al del bus de acceso a la memoria de programa.

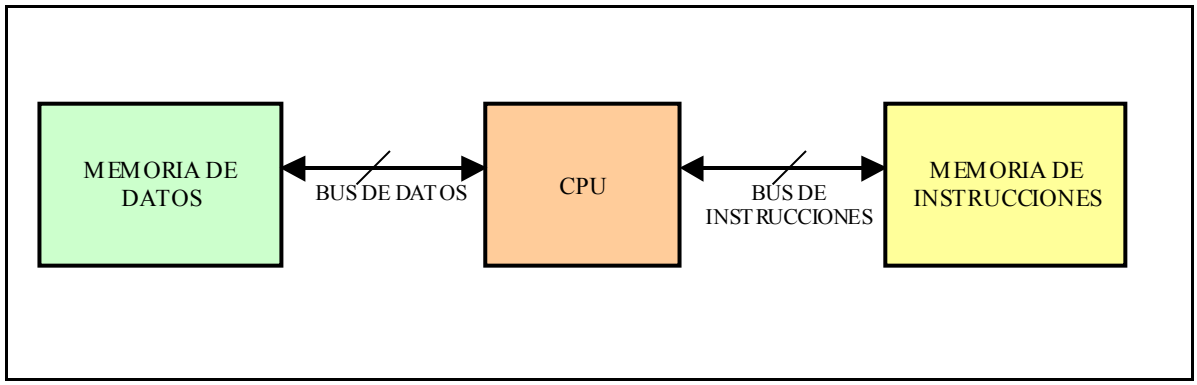


Figura 1.2. *Arquitectura Harvard*

Las principales ventajas que posee esta arquitectura son:

- El tamaño de las instrucciones no está 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.

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

1.2.2 El procesador o CPU

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.

Procesador 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.

Procesador 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 único ciclo.

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

Procesador SISC

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

1.2.3 La Memoria

En los microcontroladores, la memoria de instrucciones y de 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 de tipo RAM, volátil, y se destina a guardar las variables y los datos.

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 otro lado, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM, pues se ejecuta directamente desde la ROM.

1.2.3.1 Memoria de programa

Según el tipo de memoria ROM que disponga el microcontrolador, la aplicación y utilización del mismo son diferentes. Existen cinco tipos de memoria no volátil que se pueden encontrar en los microcontroladores del mercado:

ROM con máscara

En este tipo de memoria el programa se graba en el chip durante el proceso de su fabricación mediante el uso de "máscaras". Los altos costes 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.

EPROM

Los microcontroladores que disponen de memoria ***EPROM (Erasable Programmable Read Only Memory)*** pueden borrarse y grabarse repetidas veces. La grabación se realiza mediante un dispositivo físico gobernado desde un computador personal, que recibe el nombre de **grabador**. En la superficie de la cápsula del microcontrolador existe una ventana de cristal por la que se puede someter al chip de la memoria a rayos ultravioletas durante varios minutos para producir su borrado y emplearla nuevamente. Las cápsulas son de material cerámico y es interesante el empleo de este tipo de memoria en la fase de diseño y depuración de los programas, pero su coste unitario es elevado.

OTP (One Time Programmable)

Este modelo de memoria se puede grabar una vez por parte del usuario, utilizando el mismo procedimiento que con la memoria EPROM. Posteriormente, no se puede borrar. Su bajo precio y la sencillez de la grabación aconsejan este tipo de memoria para prototipos finales y series de producción corta.

EEPROM *(Electrical Erasable Programmable Read Only Memory)*

La grabación es similar a las memorias OTP y EPROM, pero el borrado es mucho más sencillo al poderse efectuar de la misma forma que el grabado, es decir, eléctricamente. Sobre el mismo zócalo del grabador puede ser programada y borrada

tantas veces como se quiera, lo cual la convierte en la memoria ideal para la enseñanza y para la creación de nuevos proyectos.

Los microcontroladores dotados de memoria EEPROM, una vez instalados en el circuito, pueden grabarse y borrarse sin ser retirados de dicho circuito empleando los llamados **"grabadores en circuito"**, que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

Aunque se garantizan 1.000.000 de ciclos de escritura / borrado en una EEPROM, todavía su tecnología de fabricación tiene obstáculos para alcanzar capacidades importantes y el tiempo de escritura de las mismas es relativamente grande y con elevado consumo de energía.

En los últimos tiempos, se va extendiendo entre 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.

FLASH

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar, al igual que las EEPROM, sin retirarla del circuito, pero suelen ser más rápidas y disponer de mayor capacidad que estas últimas. El borrado sólo es posible con bloques completos y no se puede realizar sobre posiciones concretas.

Este tipo de memorias es muy recomendable en aplicaciones en las que sea necesario modificar el programa a lo largo de la vida del producto, como consecuencia del desgaste o cambios de piezas, como sucede, por ejemplo, en los vehículos. También es recomendable emplear una memoria Flash en vez de una EEPROM cuando se precisa gran cantidad de memoria de programa no volátil, pues es más veloz y tolera más ciclos de escritura/borrado. Además, por sus mejores prestaciones este tipo de memoria está sustituyendo a la memoria EEPROM para contener instrucciones.

Como ejemplo podemos destacar el caso de Microchip, que comercializa dos microcontroladores prácticamente iguales, sólo se diferencian en que la memoria de programa de uno de ellos es tipo EEPROM y la del otro tipo Flash. Se trata del PIC16C84 y el PIC16F84, respectivamente.

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.

1.2.3.2 Memoria de datos

Los datos que manejan los programas varían continuamente, y esto exige que la memoria que les contiene deba ser de lectura y escritura, por lo que la memoria RAM estática (SRAM) es la más adecuada, aunque sea volátil.

Hay microcontroladores que disponen como memoria de datos una de lectura y escritura no volátil, del tipo EEPROM. De esta forma, un corte en el suministro de la alimentación no ocasiona la pérdida de la información, que está disponible al reiniciarse el programa.

1.2.4 Líneas de E/S

A excepción de dos pines destinados a recibir la alimentación, otros dos para el cristal de cuarzo que regula la frecuencia de trabajo, y uno más para provocar el reset, los restantes pines de un microcontrolador sirven para soportar su comunicación con los periféricos externos que controla.

Las líneas de E/S se agrupan en conjuntos de distintos tamaños llamados ***Puertos***. Hay modelo con líneas que soportan la comunicación en serie; otros disponen de conjuntos de líneas que implementan puertas de comunicación para diversos protocolos, como el I^2C, el USB, etc.

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 o control.

1.2.5 El 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.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes externos 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 asociado un aumento del consumo de energía.

Para saber en cuanto tiempo se ejecuta una instrucción se emplea la siguiente fórmula.

Dado que una instrucción se ejecuta en 4 ciclos de reloj

$$T_{ciclo} = (4 * (1/F_{osc}))$$

El tiempo total depende del tipo de instrucciones que el programa contenga, ya que las instrucciones de salto precisan de 2 ciclos de reloj.

$$T_{total} = (\text{Nº Instrucciones de un ciclo})*(T_{ciclo}) + (\text{Nº Instrucciones de dos ciclos})*(T_{ciclo})$$

1.2.6 Recursos auxiliares

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
- Conversor A/D
- Conversor D/A
- Comparador analógico
- Modulador de anchura de impulso o PWM
- Puertos de comunicación

Veamos ahora cada uno de estos recursos más detenidamente

1.2.6.1 Temporizadores o "Timers"

Se emplean para medir periodos de tiempo entre eventos (temporizadores), para generar temporizaciones o salidas con una frecuencia específica 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 desea contar acontecimientos que se materializan por cambios de nivel o flancos en alguno de los pines del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos.

1.2.6.2 Perro Guardián o "Watchdog"

El Perro Guardián es un temporizador que, cuando se desborda y pasa por 0, provoca automáticamente un reset del sistema. Por ello, es necesario diseñar un programa de trabajo que controle la tarea de forma que refresque o inicialice al Perro Guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescará al Perro Guardián y, al completar su temporización "ladrará" y provocará el reset.

1.2.6.3 Protección ante el fallo de alimentación o "Brownout"

Se trata de un circuito que resetea al microcontrolador cuando el voltaje de alimentación (V_{DD}) es inferior a un voltaje mínimo **("Brownout")**. Mientras el voltaje de alimentación sea inferior al umbral el dispositivo se mantiene reseteado, comenzando a funcionar normalmente cuando sobrepasa dicho valor.

1.2.6.4 Estado de reposo o de bajo consumo "Stand By"

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 ciertos equipos como, por ejemplo los portátiles, los microcontroladores disponen de una instrucción especial que les introduce en un estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal. Al activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador es despertado y reanuda su trabajo.

1.2.6.5 Conversor A/D

Los microcontroladores que incorporan un conversor A/D pueden procesar señales analógicas, como por ejemplo, temperatura, voltaje, luminosidad, etc. Suelen disponer de un multiplexor que permite aplicar a la entrada del conversor A/D diversas señales analógicas que llegan a través de los pines.

1.2.6.6 Conversor D/A

Transforma los datos digitales obtenidos del procesamiento del controlador en su correspondiente señal analógica que se saca al exterior por uno de los pines del encapsulado.

1.2.6.7 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 los pines del encapsulado. La salida del comparador proporciona un nivel lógico "1" ó "0" dependiendo de qué señal sea mayor o menor que la otra.

Por otra parte, en ciertos modelos de microcontroladores existe un módulo de tensión que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

1.2.6.8 Modulador de anchura de impulsos o PWM

Son circuitos que proporcionan en su salida impulsos de anchura variable, que se ofrecen al exterior a través de los puertos de salida.

1.2.6.9 Puertos 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 se lo permiten entre los que destacan:

- **UART**.- Adaptador de comunicación asíncrona.
- **USART**.- Adaptador de comunicación serie síncrona y asíncrona.
- Puerto paralelo esclavo para poder conectarse con los buses de otros microprocesadores.
- **USB (*Universal Serial Bus*)**.- Es un bus serie empleado para los PC.
- **Bus I^2C**.- Es un interfaz serie de dos hilos desarrollado por Philips

1.3 ALGUNOS TIPOS DE MICROCONTROLADORES

Actualmente existen en el mercado varias marcas reconocidas como las principales, dadas sus características, difusión y uso en productos de consumo masivo. Entre ellas están, Motorola, Intel, Philips, National y Microchip. A continuación, vamos a describir las características de algunas de ellas:

1.3.1 Altair

ALTAIR es el nombre genérico de una familia de microcontroladores de propósito general. Todos ellos son programables directamente desde un equipo PC mediante un lenguaje macroensamblador, o bien mediante otros lenguajes disponibles como Basic, C,...

Los microcontroladores ALTAIR disponen de una CPU de 8 bits, 256 bytes de memoria interna, 128 registros de funciones especiales (SFR), puertos de E/S de propósito general, 111 instrucciones y posibilidad de direccionar 128 Kbytes.

Las diferencias entre los distintos microcontroladores ALTAIR radican fundamentalmente en el número de E/S de sus puertos y en los periféricos que poseen (convertidores A/D, convertidores D/A, watchdog, PWM, velocidad de ejecución, etc.), por lo que la elección de un modelo u otro dependerá de las necesidades.

1.3.2 Intel

El 8051 es el primer microcontrolador de la familia introducida por Intel Corporation. La familia 8051 de microcontroladores está formada por controladores de 8 bits capaces de direccionar hasta 64 Kbytes de memoria de programa y 64 Kbytes de memoria de datos.

El 8031(la versión sin ROM interna del 8051) tiene 128 bytes de RAM interna (el 8032 tiene RAM interna de 256 bytes y un temporizador adicional). El 8031 tiene dos temporizadores/contadores, un puerto serie, cuatro puertos de E/S paralelas de propósito general (P0, P1, P2 y P3) y una lógica de control de interrupción con cinco fuentes de interrupciones. Junto a la RAM interna, el 8031 tiene varios registros de funciones especiales (SFR) que se emplean para control y como registros de datos.

La ROM interna del 8051 y el 8052 no puede ser programada por el usuario. Éste debe suministrar el programa al fabricante, y el fabricante programa los microcontroladores durante la producción. Debido al gran coste, la opción de la ROM programado por el fabricante no es económica para producción de pequeñas cantidades.

El 8751 y el 8752 son las versiones del 8051 y el 8052 que contienen memoria EPROM. Estos pueden ser programados por los usuarios.

Durante la década pasada muchos fabricantes introdujeron miembros mejorados del microcontrolador 8051. Las mejoras incluyen más memoria, más puertos, convertidores A/D, más temporizadores, más fuentes de interrupción, watchdog, y subsistemas de comunicación en red.

Todos los microcontroladores de la familia usan el mismo conjunto de instrucciones, el MCS-51.

1.3.3 Siemens

El Siemens SAB80C515 es un miembro mejorado de la familia 80C51 de microcontroladores. El 80C515 es de tecnología CMOS que típicamente reduce los requerimientos de energía comparado a los dispositivos no CMOS. Las características que le diferencian de los microcontroladores de la familia 8051 son:

- Un número mayor de puertos.
- Un convertidor A/D muy versátil.
- Un segundo temporizador.
- Un watchdog.
- Sistemas de ahorro de energía sofisticados.

El microcontrolador 80C515 es completamente compatible con el 8051 es decir, usa el mismo conjunto de instrucciones del lenguaje ensamblador MCS-51. Además, el 80C515 posee los mismos registros SFRs que el 8051, y de este modo puede correr cualquier programa escrito para el 8051 con la excepción del uso del registro prioridad de interrupción IP. Por ello, si un programa creado para el 8051 usa prioridades de interrupción, debe ser modificado antes de que se ejecute sobre el 80C515. Además, a la hora de modificar el código deberá tenerse en cuenta que el 80C515 dispone de más fuentes de interrupción y prioridades que el 8051.

1.3.4 Motorola

El 68HC11 de la familia Motorola es un potente microcontrolador que posee un bus de datos de 8 bits, un bus de direcciones de 16 bits y un conjunto de instrucciones que es similar a los más antiguos miembros de la familia 68XX (6801, 6805, 6809). Dependiendo del modelo, el 68HC11 contiene internamente los siguientes dispositivos: memoria EEPROM u OTPROM, memoria RAM, temporizadores, conversor A/D, generador PWM, y canales de comunicación sincrónica y asincrónica (RS232 y SPI).

La CPU tiene 2 acumuladores de 8 bits (A y B) que pueden ser concatenados para crear un acumulador 16 bits (D). Además, posee dos registros índices de 16 bits (X, Y) que se emplean para indexar el mapa de memoria.

Aunque es un microcontrolador de 8 bits, el 68HC11 tiene algunas instrucciones de 16 bits (add, subtract, etc.). También posee instrucciones y un puntero de 16 bits para la manipulación de la pila.

Como temporizador contiene un único contador de 16 bits y existe un preescalador programable para disminuirlo si es necesario. También viene provisto de un convertidor A/D que es típicamente de 8 canales y 8 bits de resolución. Además, posee un interfaz de comunicaciones serie (SCI) de parada y un interfaz periférico serie (SPI).

1.3.5 Atmel

Es una familia consolidad de microcontroladores entre cuyos miembros se puede destacar el AT89C51. Es un microcontrolador de 8 bits con una memoria de 4 Kbytes tipo Flash y una memoria interna RAM de 128 bytes. Además, posee 32 líneas de E/S y dos temporizadores/contadores.

El conjunto de instrucciones que emplea es compatible con el MCS-51. Por otro lado, gracias a sus 6 fuentes de interrupciones le convierte en un microcontrolador muy versátil.

1.3.6 Microchip

Los microcontroladores PIC de Microchip Technology Inc. combinan una alta calidad, bajo coste y excelente rendimiento. Estos microcontroladores son empleados en una gran cantidad de aplicaciones tan comunes como periféricos del ordenador, sistemas de seguridad y aplicaciones en el sector de telecomunicaciones. En el capítulo 2 se profundizará más en esta familia.

CAPITULO 2
MICROCONTROLADORES PIC.
EL PIC16F84A

2.1 INTRODUCCIÓN

Los ***PIC*** son unos microcontroladores desarrollados por Microchip Technology Inc. que emplean memorias EPROM ó FLASH. El fabricante los define como *una familia de microcontroladores de bajo costo, bajo consumo de potencia y alta velocidad de operación*.

En la actualidad, los PIC están teniendo una gran aceptación en la comunidad de técnicos y aficionados que trabajan con microcontroladores. Parece ser que esto es debido a detalles como:

- Su sencillez de manejo
- Una buena documentación
- Su coste es comparativamente inferior al de los competidores
- Su juego de instrucciones es reducido (35 en la gama media)
- Poseen una elevada velocidad de funcionamiento
- Los programas son compactos
- Bajo consumo unido a un amplio rango de voltajes de alimentación
- Permite realizar diseños rápidos
- Poseen herramientas software de desarrollo gratuitas que se pueden extraer de la página de la página web de la empresa Microchip
- Existe una gran variedad de herramientas hardware que permiten grabar, depurar, borrar y comprobar el comportamiento del PIC
- La gran variedad de modelos de PIC permite elegir el que mejor responde a los requerimientos de la aplicación

Lo cierto es que, actualmente, a principios del siglo XXI y en su corta vida, los PIC ocupan las posiciones de cabeza en el ranking mundial, compitiendo codo a codo con gigantes como Intel y Motorola. En 1990 ocupaba el puesto vigésimo y actualmente son más de 100 millones de PIC los que vende Microchip cada año.

2.2 RESEÑA HISTÓRICA

En 1965, la empresa GI creó una división de microelectrónica (*GI Microelectronics Division*), destinada a generar las primeras arquitecturas viables de memoria EPROM y EEPROM. De forma complementaria, GI Microelectronics Division fue también responsable de desarrollar una amplia variedad de funciones digitales y analógicas en las familias AY3-XXXX y AY5-XXXX.

A principios de los años 70 diseñó el microprocesador de 16 bits CP1600, razonablemente bueno pero que no manejaba eficazmente las funciones de entrada y salida. Para solventar este problema, en 1975 diseñó un chip destinado a controlar E/S: el **PIC (Peripheral Interface Controller)**. Se trataba de un controlador rápido pero limitado y con pocas instrucciones ya que iba a trabajar en combinación con el CP1600.

La arquitectura del PIC, que se comercializó en 1975, era sustancialmente la misma que la de los actuales modelos PIC16C5X. En aquel momento se fabricaba con tecnología NMOS y el producto sólo se ofrecía con memoria ROM y con un pequeño pero robusto microcódigo. El mercado, no obstante, no pensó así y el PIC quedó reducido a ser empleado por grandes fabricantes únicamente.

La década de los 80 no fue buena para GI, que tuvo que reestructurar sus negocios, concentrando sus actividades en los semiconductores de potencia. La GI Microelectronics División se convirtió en una empresa subsidiaria, llamada GI Microelectronics Inc. Finalmente, en 1985, la empresa fue vendida a un grupo de inversores de capital de riesgo (Venture Capital Investors), los cuales, tras analizar la situación, rebautizaron a la empresa con el nombre de Arizona Microchip Technology y orientaron su negocio a los PIC, las memorias EPROM paralelo y las EEPROM serie. Como parte de esta estrategia, se rediseñó la familia NMOS PIC165X y así emplear algo en lo que la empresa destacaba, la memoria EPROM. De esta forma, comenzaron a basarse en la tecnología CMOS, OTP y en la memoria de programación EPROM, surgiendo la familia de gama baja PIC16C5X, considerada como la "clásica". Actualmente, Microchip ha realizado un gran número de mejoras a la arquitectura original, adaptándola a las últimas tecnologías y al bajo costro de los semiconductores.

Microchip cuenta con su factoría principal en Chandler (Arizona), en donde se fabrican y prueban los chips con los más avanzados recursos técnicos. En 1993 construyó otra factoría de similares características en Tempe (Arizona). También cuenta con centros de ensamblaje y ensayos en Taiwán y Tailandia. Para hacerse una idea de la tasa de producción, es de destacar que supera el millón de unidades por semana en productos CMOS de la familia PIC16C5X.

Una de las razones del éxito de los PIC se basa en su utilización. Cuando se aprende a manejar uno de ellos, conociendo su arquitectura y su repertorio de instrucciones, es muy fácil emplear otro modelo.

2.3 PRINCIPALES CARACTERÍSTICAS DE LOS PIC

Las principales características que describen a un microcontrolador PIC son:

- **Son microcontroladores de arquitectura cerrada**
  Cada modelo se construye con una determinada CPU, cierta capacidad de memoria de datos, cierto tipo y capacidad de memoria de instrucciones, un número de E/S y un conjunto de recursos auxiliares muy concreto. El modelo no admite variaciones ni ampliaciones.
- **La arquitectura del procesador sigue el modelo Harvard**
- **Se emplea la técnica de segmentación ("pipe-line") en la ejecución de las instrucciones**
  La segmentación permite al procesador realizar simultáneamente la ejecución de una instrucción y la búsqueda del código de la siguiente. De esta forma se puede ejecutar cada instrucción en un ciclo, excepto las instrucciones de salto que ocupan dos ciclos al no conocer la dirección de la siguiente instrucción hasta que no se haya completado la de bifurcación. En el apartado 4.2 se explica detalladamente en qué consiste esta técnica.
- **El formato de todas las instrucciones tiene la misma longitud**
  Todas las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits. Las de la gama media tienen 14 bits y un número superior las de la gama alta. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.
- **Procesador RISC (Computador de Juego de Instrucciones Reducido)**
  Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, 35 los de la gama media y casi 60 los de la alta.
- **Todas las instrucciones son ortogonales**
  Cualquier instrucción puede manejar cualquier elemento de la arquitectura como fuente o como destino.
- **Posee una arquitectura basada en un banco de registros**
  Esto significa que todos los objetos del sistema (puertos de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.
- **Gran variedad de modelos de microcontroladores con prestaciones y recursos diferentes**
  La gran diversidad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.
- **Herramientas de soporte potentes y económicas**
  La empresa Microchip y otras que utilizan los PIC ponen a disposición de los usuarios numerosas herramientas para desarrollar hardware y software. Son muy abundantes los programadores, los simuladores software, los emuladores en tiempo real, ensambladores, Compiladores C, Intérpretes y Compiladores BASIC, etc.

2.4 LAS DISTINTAS GAMAS DE MICROCONTROLADORES PIC

Dentro de un proyecto, dependiendo de la función o aplicación que se quiera implementar, los recursos a emplear serán más o menos potentes y numerosos. Por ello, Microchip Technology oferta un amplio abanico de microcontroladores de 8 bits agrupados en tres gamas principales, diferenciándose por el número de líneas de E/S y por la dotación de dispositivos. Con las tres gamas de PIC se dispone de gran diversidad de modelos y encapsulados, pudiendo seleccionar el que mejor se acople a las necesidades de acuerdo con el tipo y capacidad de las memorias, el número de líneas de E/S y las funciones auxiliares precisas. Sin embargo, todas las versiones están construidas alrededor de una arquitectura común, un repertorio mínimo de instrucciones y un conjunto de opciones muy apreciadas, como el bajo consumo y el amplio margen del voltaje de alimentación. En la siguiente figura se muestra la distribución de los modelos de PIC en las tres gamas.

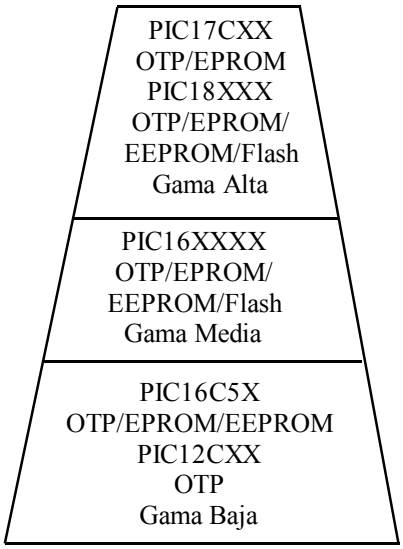


Figura 2.1. *Las distintas gamas de microcontroladores PIC*

- **Gama baja.-** La compone la familia de microcontroladores 16C5X, los cuales tienen un temporizador y pines de E/S.
- **Gama media.-** La componen las familias 16C6X/7X/8X, que incorporan conversores A/D, comparadores, interrupciones, etc.
- **Gama alta.-** La familia de rango superior la componen los 17CXX y los 18XXX, que tienen muchos más servicios y prestaciones.

Una descripción más detallada de todos los miembros de cada gama se encuentra disponible en la página web de Microchip http://www.microchip.com. En este capítulo únicamente se pretende profundizar en el microcontrolador PIC16F84A, que es el empleado en el de desarrollo de este proyecto.

2.5 CARACTERÍSTICAS DEL PIC16F84A

- Encapsulado de plástico DIP de 18 pines
- Repertorio de 35 instrucciones
- Memoria de programa Flash de 1K x 14 bits
- Memoria de datos RAM de 68 bytes
- Memoria de datos EEPROM de 64 bytes
- 15 registros de funciones especiales (SFR)
- Pila con 8 niveles de profundidad
- Modos de direccionamiento directo, indirecto y relativo
- 4 fuentes de interrupción
- Contiene 13 líneas de E/S divididas en dos puertos
    - Puerto A formado por 5 líneas
    - Puerto B formado por 8 líneas
- Contador/Temporizador TMR0 de 8 bits con divisor programable
- Watchdog Timer (WDT)
- Protección de código
- Modo de bajo consumo SLEEP
- 4 modos distintos de oscilador
- Programación en serie a través de dos pines
- Tecnología de baja potencia y alta velocidad CMOS Flash/EEPROM
- Rango de alimentación de 2 a 6 V

2.6 ARQUITECTURA EXTERNA

El PIC está fabricado con tecnología CMOS de altas prestaciones y encapsulado en plástico de 18 pines. En la Figura 2.2. se puede observar el encapsulado del PIC16F84A.

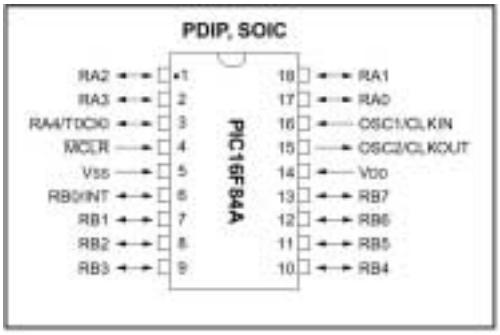


Figura 2.2. *Encapsulado del PIC16F84A*

Veamos cuáles son las señales disponibles en cada una de los pines.

- **Señales de alimentación**
  - **V_{SS}.-** Masa general o conexión a tierra.
  - **V_{DD}.-** Señal de alimentación (positiva), que debe estar comprendida entre 2 y 6 V. En modo SLEEP deberá tener una alimentación de cómo mínimo 1.5 V para que la RAM interna conserve sus contenidos.
- **OSC1/CLKIN**.- Es el pin por el que se aplica la entrada del circuito oscilador externo que proporciona la frecuencia de trabajo del microcontrolador.
- **OSC2/CLKOUT**.- Pin auxiliar del circuito oscilador.
- **MCLR#.-** Este pin es activa con nivel lógico bajo, lo que se representa en la Figura 2.2. con una barrita encima de la palabra MCLR. Su activación origina la reinicialización o reset del PIC. También se usa este pin durante la grabación de la memoria de programa para introducir por ella la tensión, V_{PP}, que está comprendida entre 12 y 14 V.
- **RA0 - RA4 .-** Son las 5 líneas de E/S digitales correspondientes al puerto A. La línea *RA4* multiplexa otra función expresada por *TOCKI*. En este segundo caso sirve para recibir una frecuencia externa para alimentar al temporizador interno TMR0.
- **RB0 - RB7.-** Estos 8 pines corresponden a las 8 líneas de E/S digitales del Puerto B. La línea *RB0* multiplexa otra función, que es la de servir como entrada a una petición externa de una interrupción, por eso la denomina *RB0/INT*.

2.7 ARQUITECTURA INTERNA

La arquitectura interna del PIC16F84A se representa en la Figura 2.3.

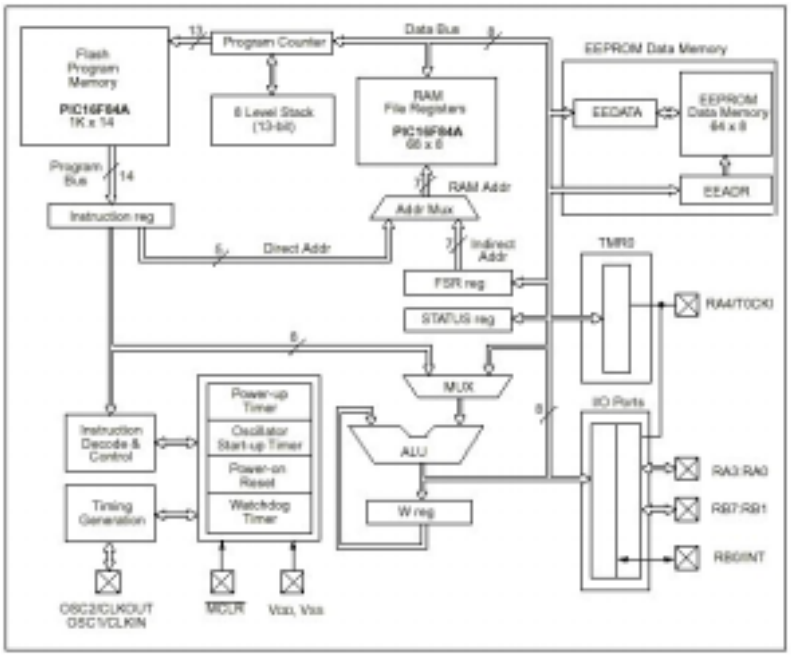


Figura 2.3. *Arquitectura interna del PIC16F84A*

Consta de siete bloques fundamentales:

- Una Memoria de Programa Flash de 1K x 14 bits.
- Una Memoria de Datos formada por dos áreas. Una RAM donde se alojan 22 Registros de Propósito Específico (SFR) y 36 de Propósito General (GPR), y otra de tipo EEPROM de 64 bytes.
- Un camino de datos que consta de una ALU de 8 bits y de un registro de trabajo W. Este registro se emplea para introducir en la ALU un operando; el otro operando puede provenir del bus de datos o del propio código de la instrucción (literal). W también se emplea para entregar el resultado después de la operación.
- Diversos recursos conectados al bus de datos, tales como líneas de E/S, el Temporizador TMR0, etc.
- Una base de tiempos y circuitos auxiliares.
- Un direccionamiento de la memoria de programa en base al Contador de Programa ligado a una pila de 8 niveles de profundidad.
- Direccionamiento directo e indirecto de la memoria RAM.

Para poder comprender y analizar el funcionamiento del microcontrolador nos vamos a centrar en la ejecución de una instrucción. Todo comienza con la fase de búsqueda, que la inicia el Contador de Programa facilitando la dirección de la memoria de instrucciones

donde se ubica. Su código binario de 14 bits se lee y se carga en el Registro de Instrucciones, desde donde se transfiere al decodificador y a la Unidad de Control. A veces, dentro del código de la instrucción existe el valor de un operando (literal) que se introduce como operando a la ALU, o bien una dirección de la memoria de datos donde reside otro operando.

La ALU es la encargada de realizar la operación lógico-aritmética que implica la instrucción decodificada. Uno de los operandos lo recibe desde el registro W y el otro desde un registro o de la propia instrucción.

Tanto el banco de registros específicos, en el que cada uno tiene una misión concreta, como el de registros de propósito general residen en la RAM. La EEPROM de datos puede contener datos que no se desee perder al quitar la alimentación, pero su acceso está controlado con unos registros especiales.

Las operaciones de E/S con los periféricos las soportan los Puertos A y B. Existe un Temporizador, TMR0, que se encarga de las funciones de control de tiempos. Finalmente, hay unos circuitos auxiliares que dotan al procesador de una interesantes posibilidades de seguridad, reducción del consumo y reinicialización.

2.8 MEMORIA DE PROGRAMA

La característica más relevante del PIC16F84A es que la **Memoria de Programa es de tipo Flash**, lo cual permite que este microcontrolador sea escrito y borrado eléctricamente.

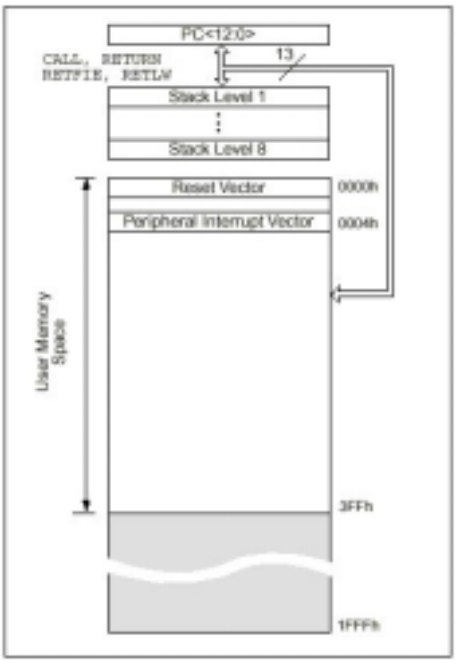


Figura 2.4. *Mapeo de la Memoria de Programa*

Por otro lado, la arquitectura del PIC16F84A admite un mapa de memoria de programa capaz de contener 8.192 instrucciones de 14 bits cada una (8K x 14 bits). Este mapa se divide en páginas de 2.048 posiciones (2K x 14 bits). Para direccionar 8 K posiciones se necesitan 13 bits, que es la longitud que tiene el **Contador de Programa (PC)**. Sin embargo, como se puede observar en la Figura 2.4., el PIC16F84A sólo tiene implementada 1 K posiciones (de la 0000h a la 03FFh), por lo que se ignora los 3 bits de más peso del PC. De esta forma, el apuntar a la dirección 33h es lo mismo que hacerlo a la 433h, 833h, C33h, 1033h, 1433h o a la 1C33h.

La dirección 00h está reservada para el vector de Reset y la 04h para el Vector de Interrupción.

2.8.1 El Contador de Programa

Al igual que todos los registros específicos que controlan la actividad del procesador, el Contador de Programa está implementado sobre un par de posiciones de la memoria RAM. Como ya se ha indicado anteriormente, consta de 13 bits, pero los 3 más significativos se ignoran.

Cuando se escribe el Contador de Programa como resultado de una operación de la ALU, los 8 bits de menos peso del PC residen en el ***registro PCL*** que ocupa, de forma repetida, la posición 2 de los dos bancos de la memoria de datos; este registro es accesible tanto para la escritura como para la lectura. Los bits de más peso, PC<12:8>, residen en los 5 bits de menos peso del ***registro PCLATH***, que ocupa la posición 0Ah de los dos bancos de la memoria RAM.

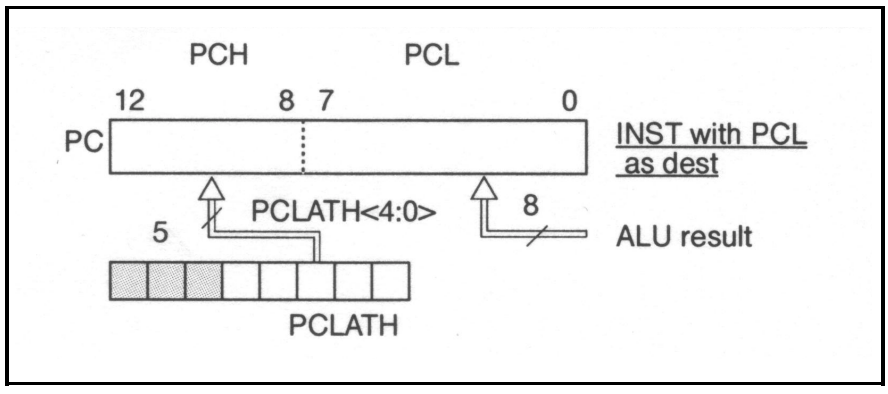


Figura 2.5. *Forma en la que se carga el PC cuando una instrucción deposita en él el resultado que se obtiene de la ALU.*

En las instrucciones de salto relativo, el resultado de las mismas sólo afecta a los 8 bits de menor peso. Los 5 bits de mayor peso se suministran desde PCLATH. En las instrucciones GOTO y CALL se efectúa la misma operación teniendo presente que el PC se codifica mediante 11 bits y se suministran en la propia instrucción desde el código OP. Los 2 bits de más peso del PC se cargan con los bits 4 y 3 del registro PCLATH.

Dado que la memoria de programa se organiza en páginas de 2 K, la posición de la memoria le indican los 11 bits de menor peso del PC y los 2 bits de mayor peso indican la página.

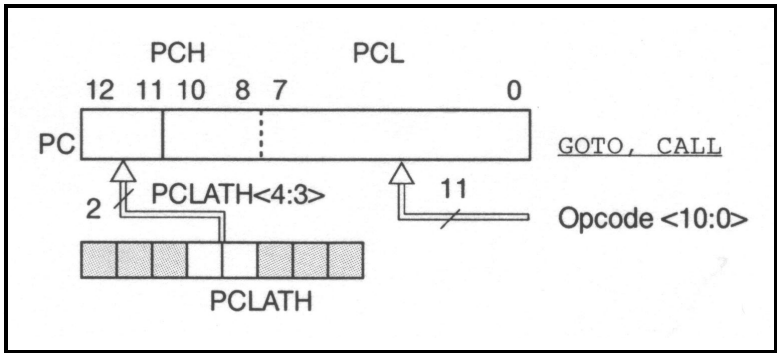


Figura 2.6. *Carga del PC en las instrucciones GOTO y CALL*

2.8.2 La Pila

La Pila es una zona aislada de las memorias de instrucciones y datos. Tiene una estructura LIFO, en la que el último valor guardado es el primero que sale. Tiene ocho niveles de profundidad, cada uno con 13 bits. Funciona como un "buffer" circular, de manera que el valor que se obtiene al realizar la novena extracción de la pila (*pop*) es igual al que se obtuvo en el primero.

La instrucción CALL y las interrupciones originan la carga del contenido del PC en la cima de la Pila. El contenido de este nivel se saca de la misma al ejecutar las instrucciones RETURN, RETLW y RETFIE. El contenido del registro PCLATH no resulta afectado por la entrada o salida de información de la Pila.

Los microcontroladores PIC no disponen de instrucciones específicas (*push* y *pop*) para manejar directamente la Pila y tampoco disponen de ningún señalizador que indique cuándo se produce el desbordamiento de la misma.

2.9 MEMORIA DE DATOS

El microcontrolador PIC16F84A, como todos los PIC, funcionan con datos de 8 bits, por lo que las posiciones de la memoria de datos tienen esa longitud.

La Memoria de Datos está dividida en dos zonas:

- **Área EEPROM** de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se quieran perder al desconectar la alimentación.
- **Área de RAM estática o SRAM**, donde residen los Registros de Funciones Especiales (SFR) y los Registros de Propósito General (GPR).

2.9.1 La Memoria de Datos RAM

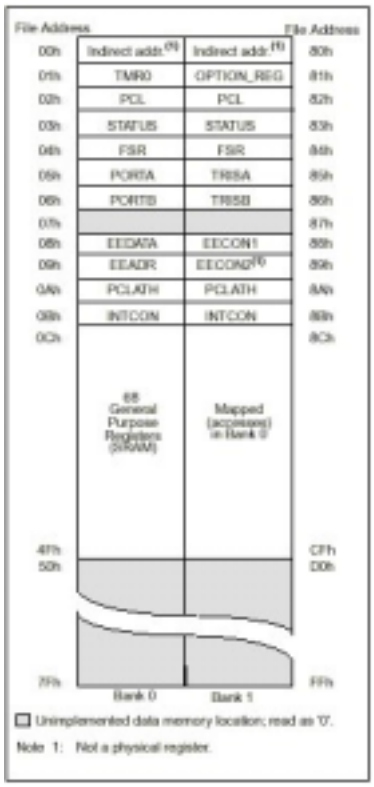


Figura 2.7. *Mapa de registros en la Memoria de Datos*

Está organizada en dos páginas o bancos de registro (banco 0 y banco 1) de 128 bytes cada uno, de los cuales se encuentran implementadas físicamente las 48 primeras posiciones de cada banco. Cada banco se divide a su vez en dos áreas (Véase Figura 2.7.):

- El primer área consta de 12 posiciones de un byte. En ella se encuentran los ***Registro de Funciones Especiales (SFR)***, que son los encargados del control del funcionamiento de la CPU y de los periféricos.

  Algunos de dichos registros están repetidos en la misma dirección de los dos bancos, para así simplificar su acceso *(INDF, STATUS, FSR, PCLATH e INTCON).*

La posición apuntada por la dirección 7h y la apuntada por la 87h no son operativas.

- El segundo área de cada banco está formado por 68 posiciones que se destinan para los ***Registros de Propósito General (GPR)***. En realidad, sólo son operativos los 36 registros del banco 0 porque los del banco 1 se mapean sobre el banco 0, es decir, cuando se apunta a un registro del banco 1, se accede al mismo de banco 0.

Para seleccionar el banco a acceder hay que manipular el bit 5 (***RP0***) del ***registro STATUS***. Si *RP0* = 1 se accede al banco 1 y si es = 0 se accede al banco 0. Tras un reset se accede automáticamente al banco 0.

Para seleccionar un Registro de Propósito General no es necesario tener en cuenta el estado del bit *RP0*, porque al estar mapeado el banco 1 sobre el banco 0, cualquier direccionamiento de un registro del banco 1 corresponde al homólogo del banco 0. En el direccionamiento directo a los registros GPR se ignora el bit de más peso, que identifica el banco y sus direcciones están comprendidas entre el valor 0Ch y 2Fh.

2.9.1.1 Direccionamiento de la Memoria de Datos

Las instrucciones pueden especificar los datos u operandos mediante tres modos de direccionamiento:

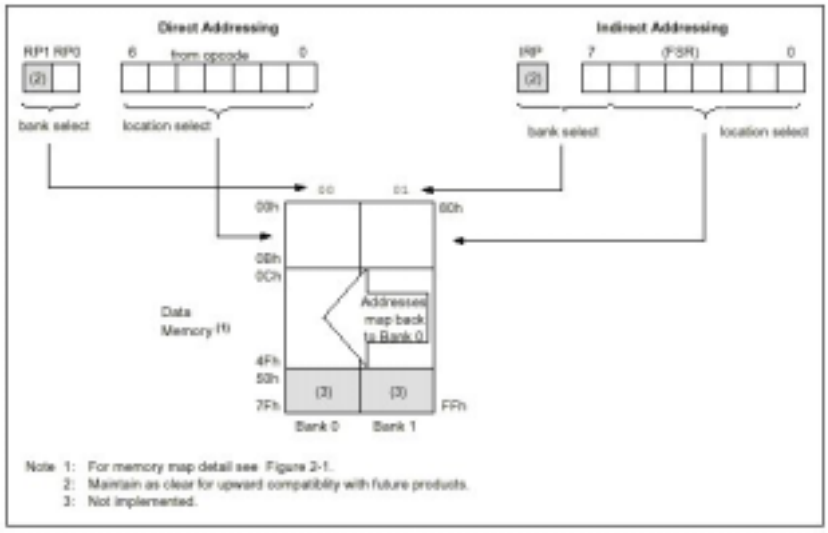


Figura 2.8. *Modos de direccionamiento de la Memoria de Datos*

- **Inmediato**
  Cuando una instrucción utiliza un dato inmediato, su valor (literal) lo contiene su código OP y en la ejecución se carga en el registro W para su posterior procesamiento.

- **Direccionamiento Directo**
  El operando que utiliza la instrucción en curso se referencia mediante su dirección, que viene incluida en el código OP de la misma, concretamente en los 7 bits de menos peso. El banco a acceder lo determina el bit *RP0* del registro STATUS.
- **Direccionamiento Indirecto**
  Este modo de direccionamiento se emplea cuando en una instrucción se utiliza como operando el ***registro INDF***, que ocupa la dirección 0 de ambos bancos. En realidad, el registro INDF no está implementado físicamente y cuando se le hace referencia, se accede a la dirección de un banco especificada con los 7 bits de menos peso del ***registro FSR***. El bit de más peso de FSR junto al bit *IRP* del registro STATUS se encargan de seleccionar el banco a acceder, mientras que los 7 bits de menos peso apuntan a la posición. Como sólo hay dos bancos en el PIC16F84A en este modo de direccionamiento, el bit *IRP* será siempre 0.

2.9.1.2 Los Registros de Funciones Especiales (SFR)

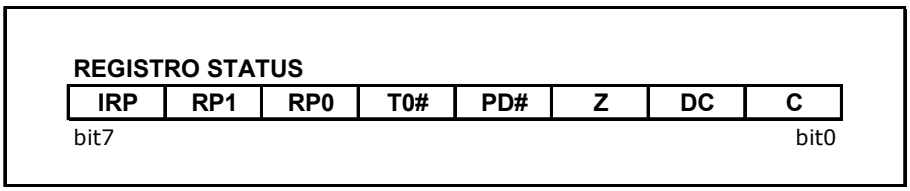
Como ya se ha indicado anteriormente, las 12 primeras posiciones de ambos bancos de la Memoria de Datos están ocupadas por aquellos registros que se encargan de controlar la CPU y los periféricos. En la tabla que se muestra a continuación se especifican cuáles son esos registros:

Direcc	**Nombre**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Banco 0									
00h	**INDF**	Dirección de FSR (no es físicamente un registro)							
01h	**TMR0**	Contador/Temporizador de 8 bits							
02h	**PCL**	8 bits lsb del PC							
03h	**STATUS**	IRP	RP1	RP0	TO#	PD#	Z	DC	C
04h	**FSR**	Puntero para el direccionamiento indirecto							
05h	**PORTA**				RA4/ T0CKI	RA3	RA2	RA1	RA0
06h	**PORTB**	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/ INT
08h	**EEDATA**	Registro de datos EEPROM							
09h	**EEADR**	Registro de direcciones EEPROM							
0A	**PCLATH**				5 bits msb del PC				
0B	**INTCON**	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Direcc	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Banco 1									
80h	**INDF**	Dirección de FSR (no es físicamente un registro)							
81h	**OPTION**	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	**PCL**	8 bits lsb del PC							
83h	**STATUS**	IRP	RP1	RP0	TO#	PD#	Z	DC	C
84h	**FSR**	Puntero para el direccionamiento indirecto							
85h	**TRISA**				Dirección de datos del Puerto A				
86h	**TRISB**	Dirección de los datos del Puerto B							
88h	**EECON1**				EEIF	WRERR	WREN	WR	RD
89h	**EECON2**	Segundo registro de control de la EEPROM							
0Ah	**PCLATH**				5 bits msb del PC				
0Bh	**INTCON**	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Las posiciones 07h y 87h no están implementadas. Veamos detenidamente cada uno de estos registros:

- **REGISTRO STATUS**



Es un registro que ocupa la posición 03h del banco 0 y la 83h del banco 1. Contiene varios bits que se encargan de avisar las incidencias del resultado de la ALU (*C, DC* y *Z*), indican el estado de Reset *(TO# Y PD#)* y seleccionan el banco a acceder en la memoria de datos*(IRP, RP0 y RP1)*.

Veamos los bits más detenidamente:

bit 0: **C: Acarreo/llevada (Carry/borrow bit)**

1 = Cuando este señalizador vale 1 indica que se ha producido acarreo en el bit de más peso del resultado al ejecutar las instrucciones *addwf* y *addlw*.

0 = No se ha producido acarreo

C también actúa como señalizador de "llevada" en el caso de la instrucción de resta como *subwf y sublw*. En este caso la correspondencia es inversa (si vale 1 no hay llevada y si vale 0 sí).

Para las instrucciones de rotación *rrf y rlf*, se carga a este registro con el bit más o menos significativo del registro fuente.

bit 1: **DC: Acarreo/llevada en el 4º bit (Digit Carry/Borrow)**
Tiene el mismo significado que C pero refiriéndose al 4º bit. De interés en operaciones en BCD.

bit 2: **Z: Cero**
1 = El resultado de una operación lógico-aritmética es 0.
0 = El resultado de la operación es distinto de 0.

bit 3: **PD#: Power Down**
1 = Automáticamente a 1 tras conectar la alimentación V_{dd} o ejecutar *clrwdt*.
0 = Se pone a 0 al ejecutar la instrucción *sleep*.

bit 4: **TO#: Time out**
1 = Tras conectar la alimentación V_{dd} o al ejecutar *clrwdt* o *sleep*.
0 = Desbordamiento del Perro Guardián.

bit 6-5: **RP1:RP0: Selección del banco en direccionamiento directo**
Como EL PIC16F84A sólo tiene dos bancos, se emplea solamente el bit *RP0*. *RP1* debe mantenerse a 0.
00 = Banco 0 (00h-7Fh).
01 = Banco 1 (80h-FFh).

bit 7: **IRP: Selección del banco en direccionamiento indirecto**
Este bit, junto con el de más peso del registro FSR sirven para determinar el banco de la memoria de datos seleccionado. Como el PIC16F84A sólo dispone de dos bancos, no se usa y debe programarse como 0.

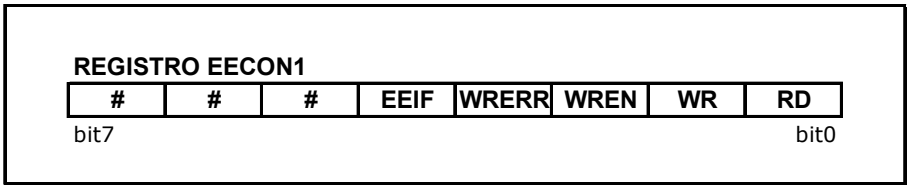
El resto de los registros se irán estudiando en distintos apartados.

2.9.2 La Memoria EEPROM de Datos

El PIC16F84A tiene 64 bytes de memoria EEPROM de datos, donde se pueden almacenar datos y variables que interesa que no se pierdan cuando se desconectan la alimentación al sistema. Soporta hasta un millón de ciclos de escritura/borrado. Su programación dura unos 10 ms y se controla mediante un temporizador interno.

La memoria EEPROM no está mapeada en la zona de la memoria de datos donde se ubican los registros SFRs y GPRs. Para poder leerla y escribirla durante el funcionamiento normal del microcontrolador hay que utilizar cuatro registros del banco SFR:

- **Registro EEDATA,** ubicado en la dirección 8h, se emplea para depositar en él los datos que se leen o escriben.
- **Registro EEADR,** ubicado en la dirección 9h, es el registro en el que se carga la dirección a acceder de la EEPROM de datos.
- **Registro EECON1,** que ocupa la dirección 88h, tiene misiones de control de las operaciones en la EEPROM. La distribución de sus bits es la siguiente:



bit 0: **RD: Lectura**
1 = Se pone a 1 cuando se va a realizar un ciclo de lectura de la EEPROM. Luego pasa a 0 automáticamente.

bit 1: **WR: Escritura**
1 = Se pone a 1 cuando se va a realizar un ciclo de escritura de la EEPROM. Luego pasa a 0 automáticamente.

bit 2: **WREN: Permiso de escritura**
1 = Permite la escritura de la EEPROM.
0 = Lo impide.

bit 3: **WRERR: Señalizador de error en escritura**
1 = La operación de escritura ha terminado prematuramente.
0 = La operación de escritura se ha realizado correctamente.

bit 4: **EEIF: Señalizador de final de operación de escritura**
1 = La operación de escritura se ha completado con éxito. Se pone a 0 por programa.
0 = La operación de escritura no se ha completado.

2.10 LA FRECUENCIA DE FUNCIONAMIENTO. EL RELOJ

La frecuencia de trabajo del microcontrolador es un parámetro fundamental a la hora de establecer la velocidad en la ejecución de instrucciones y el consumo de energía.

Cuando un PIC16F84A funciona a 20 MHz, que es su máxima frecuencia, le corresponde un ciclo de instrucción de 200 ns, puesto que cada instrucción tarda en ejecutarse cuatro períodos de reloj, es decir, 4 x 50 ns = 200 ns. Todas las instrucciones del PIC se realizan en un ciclo de instrucción, excepto las de salto, que tardan el doble.

Los impulsos de reloj entran por la patita *OSC1/CLKIN* y se dividen por 4 internamente, dando lugar a las señales Q1, Q2, Q3 y Q4, mostradas en la Figura 2.9.

Durante un ciclo de instrucción, que comprende las 4 señales mencionadas, se desarrollan las siguientes operaciones:

- ***Q1****.- Durante este impulso se incrementa el Contador de Programa y se busca la instrucción.*
- ***Q2-Q3-Q4****.- Durante la activación de estas tres señales se produce la decodificación y la ejecución de la instrucción.*

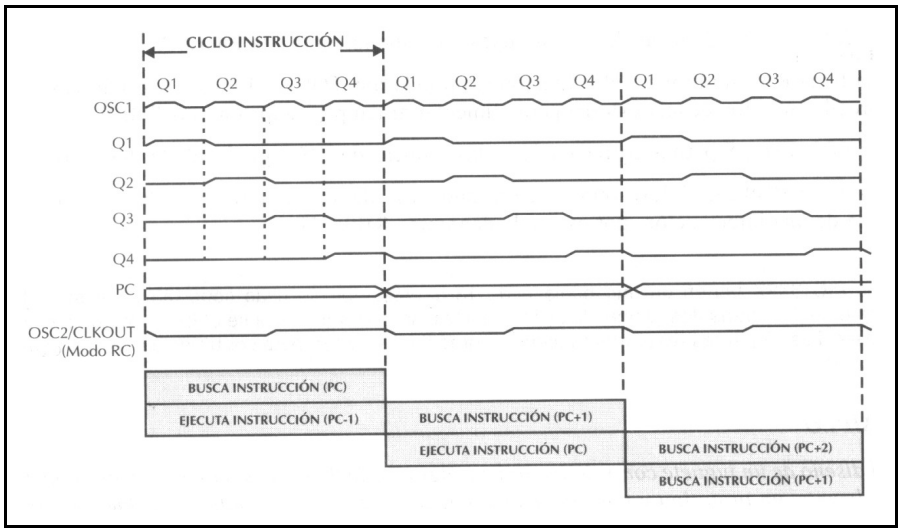


Figura 2.9. *Esquema del ciclo de instrucción*

Para poder ejecutar cada instrucción en un ciclo de instrucción (excepto las de salto, que tardan dos), se aplica la ***técnica de la segmentación o "pipe-line"****,* que consiste en realizar en paralelo las dos fases que comprende cada instrucción.

En realidad, cada instrucción se ejecuta en dos ciclos: en el primero se lleva a cabo la fase de búsqueda del código de la instrucción en la memoria del programa, y en el segundo se decodifica y se ejecuta (fase de ejecución). La estructura segmentada del procesador permite realizar al mismo tiempo la fase de ejecución de una instrucción y la de búsqueda

de la siguiente. Cuando la instrucción ejecutada corresponde a un salto no se conoce cuál será la siguiente instrucción por un ciclo "vacío", originando que las instrucciones de salto tarden en realizarse dos ciclos de instrucción. En la Figura 2.10. se muestra un ejemplo de ejecución de esta técnica

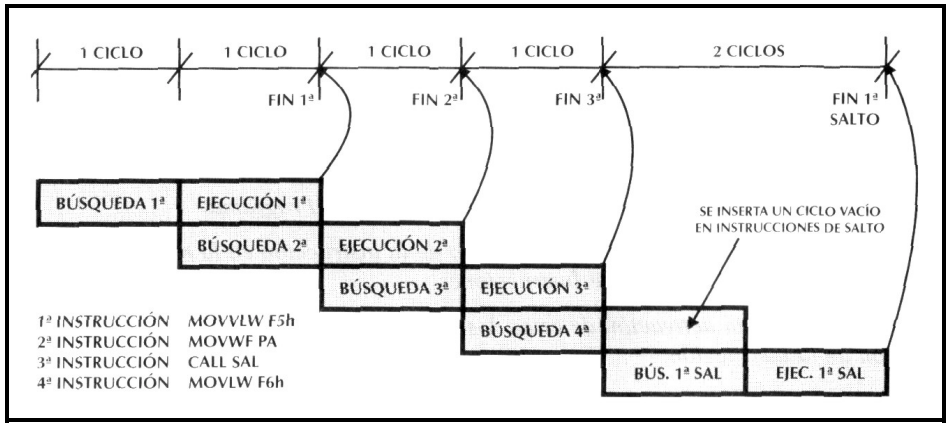


Figura 2.10. *Ejemplo de un ciclo de instrucción*

La técnica de la segmentación, unida a la arquitectura Harvard del procesador, permite al PIC16F84A superar la velocidad de sus competidores directos. Así, por ejemplo, es 1,54 veces más rápido que el microcontrolador de Motorola 68HC05 cuando ambos funcionan a la misma frecuencia de 4 MHz.

2.10.1 Tipos de osciladores

Todo microcontrolador requiere un circuito externo que le indique la velocidad a la que debe trabajar. Este circuito, que se conoce con el nombre de oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El PIC16F84A admite cuatro tipos de osciladores externos para aplicarles la frecuencia de funcionamiento. Estos cuatro tipos son:

Oscilador tipo "RC"

Se trata de un oscilador de bajo coste compuesto de un condensador y una resistencia. Los valores de estos dos elementos determinan el valor de la frecuencia. Proporciona una estabilidad mediocre de la frecuencia, por lo que no es aconsejable a la hora de programar temporizadores.

Oscilador tipo "HS"

Es un oscilador de alta velocidad basado en cristal de cuarzo o un resonador cerámico. Puede utilizarse con frecuencias en el rango de 4 MHz a 20 MHz.

Oscilador tipo "XT"

Se trata de un oscilador de cristal o resonador para frecuencias estándar comprendidas entre 100 KHz y 4 MHz.

Oscilador tipo "LP"

Oscilador de bajo consumo con cristal o resonador diseñado para trabajar en un rango de frecuencias de 35 a 200 KHz. El cristal de cuarzo o el resonador cerámico se coloca entre las patitas *OSC1* y *OSC2*.

En el momento de programar el microcontrolador se debe especificar qué tipo de oscilador se usa. Esto se hace a través de dos bits (*FOSC1 y FOSC2*) pertenecientes a la "Palabra de Configuración", como se comenta más adelante.

En la Figura 2.11. se muestra el esquema de un oscilador RC conectado a un PIC.

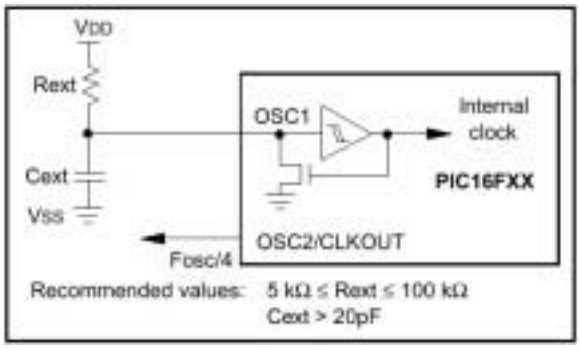


Figura 2.11. *Esquema del oscilador RC*

Los valores recomendados para este tipo de oscilador son:

$$5K < R_{ext} < 100\ K$$
$$C_{ext} > 20\ pF$$

En este tipo de configuración, en el pin *OSC2/CLKOUT* se encuentra disponible la frecuencia del oscilador dividida por cuatro. Además, cuando se programa el dispositivo en modo RC no se puede conectar un reloj externo en el pin *OSC1/CLKIN*, pues podría provocar daños al microcontrolador.

En la Figura 2.12. se muestra la configuración para los osciladores HS, XT y LP.

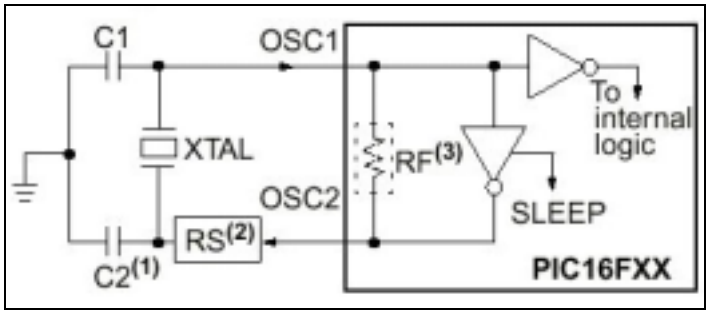


Figura 2.12. *Esquema del oscilador XT, LP y HS*

Según el tipo de oscilador y la frecuencia de trabajo se emplean diferentes valores en los condensadores C1 y C2 que acompañan al cristal de cuarzo. Para frecuencias comprendidas entre 4 MHz y 20 MHz, los condensadores tienen una capacidad de 15 pF a 33 pF. La resistencia RS sólo es necesaria en algunas versiones tipo HS.

2.11 TEMPORIZADORES

2.11.1 Control de Tiempos

Una de las labores más habituales en los programas de control de dispositivos suele ser determinar intervalos concretos de tiempo. El dispositivo típico destinado a realizar esta función recibe el nombre de ***temporizador o "timer"*** y, básicamente, consiste en un contador ascendente o descendente que determina un tiempo determinado entre el valor que se le carga y el momento en que se produce su desbordamiento o paso por 0.

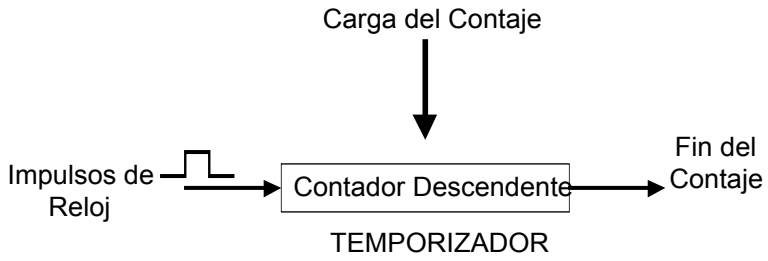


Figura 2.13. *Esquema simplificado de un temporizador.*

Existe un segundo temporizador denominado **Perro Guardián o "Watchdog" (WDT)**. Se encarga de vigilar que el programa no deje de ejecutarse. Para ello, el Perro Guardián comprueba cada cierto tiempo si el programa se ejecuta normalmente; en caso contrario, por ejemplo si el control está detenido en un bucle infinito o a la espera de algún acontecimiento que no se produce, el perro "ladra" y provoca un reset, reinicializando todo el sistema.

A menudo tanto el Temporizador Principal (TMR0) como el Perro Guardián, precisan controlar tiempos largos y aumentar la duración de los impulsos de reloj que les incrementan o decrementan. Para cubrir este requisito, se dispone de un circuito programable llamado ***Divisor de Frecuencia*** que divide la frecuencia utilizada por diversos rangos para poder realizar temporizaciones más largas.

Para regular el comportamiento del Temporizador Principal, el Perro Guardián y el Divisor de Frecuencia se emplean algunos bits de la ***Palabra de Configuración*** y ***del registro OPTION_REG***, que se explicarán oportunamente más adelante.

En la Figura 2.14. se representa un esquema simplificado de la arquitectura del circuito de control de tiempos empleado por el PIC16F84A. El Divisor de Frecuencia puede usarse

con el TMR0 o con el WDT. Con el TMR0 funciona como pre-divisor, es decir, los impulsos pasan primero por el Divisor de Frecuencia y, una vez aumentada su duración, se aplican a TMR0. Con el Perro Guardián actúa después, realizando la función de post-divisor.

Los impulsos, que divide por un rango el Divisor de Frecuencia, pueden provenir de la señal de reloj interna ($F_{osc}/4$) o de los que se aplican al pin *T0CKI*.

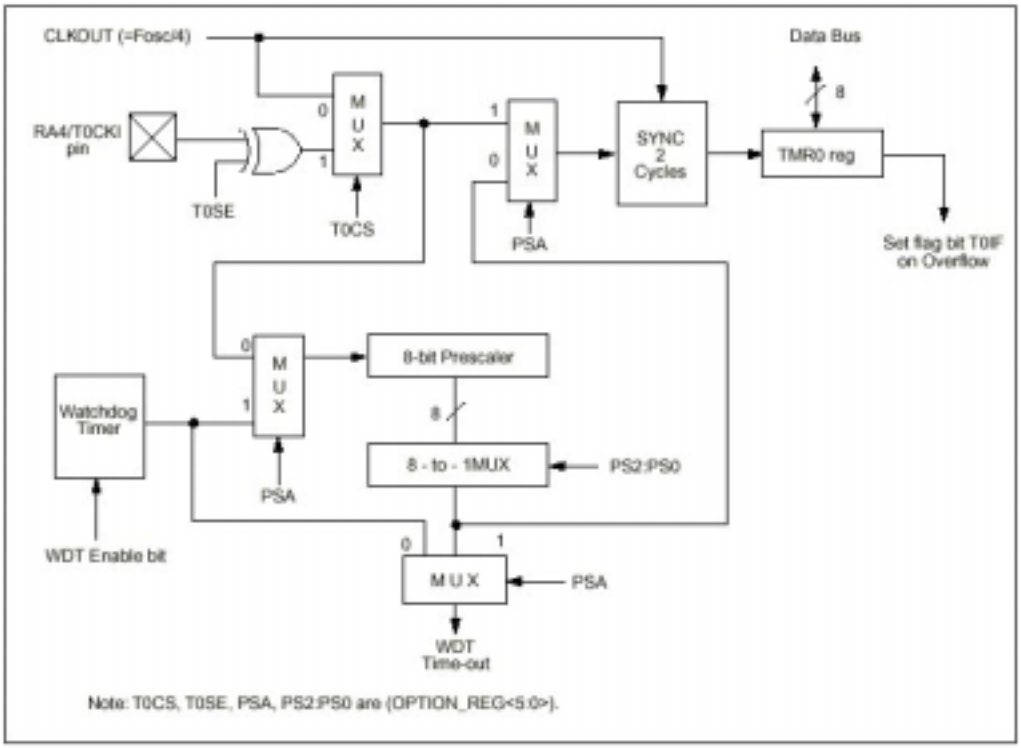


Figura 2.14. *Arquitectura del circuito de control de tiempos*

2.11.2 El registro OPTION_REG

La misión principal de este registro es gobernar el comportamiento del Temporizador Principal TMR0 y del Divisor de Frecuencia. Ocupa la posición 81h de la Memoria de Datos, que equivale a la dirección 1h del banco 1.

REGISTRO OPTION_REG

RBPU#	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

bit7 bit0

Veamos los bits que lo componen:

bit 7: **RBPU#: Resistencias pull-up Puerto B**

Existen unas resistencias de pull-up que pueden activarse o no en las líneas del puerto B.

1 = Desactivadas
0 = Activadas

bit 6: **INTEDG: Flanco activo interrupción externa (Interrupt Edge)**

Este bit permite seleccionar el flanco activo que provocará una interrupción externa al aplicarse al pin RB0/INT.
1 = Flanco ascendente
0 = Flanco descendente

bit 5: **TOCS: Tipo de flanco en T0CKI (Timer 0 Clock Source Select pin)**

El bit T0CS selecciona el multiplexor MPX1 la procedencia de los impulsos de reloj, que pueden ser los del oscilador interno ($F_{osc}/4$) o los que se aplican desde el exterior por la patita T0CKI.
1 = Pulsos introducidos a través de T0CKI (contador)
0 = Pulsos de reloj interno $F_{osc}/4$ (temporizador)

bit 4: **TOSE: Tipo de flanco en T0CKI (Timer 0 Source Edge Select)**
1 = Incremento de TMR0 cada flanco descendente.
0 = Incremento de TMR0 cada flanco ascendente.

bit 3: **PSA: Asignación del Divisor de frecuencia**
1 = El Divisor de frecuencia se le asigna al WDT.
0 = El Divisor de frecuencia se le asigna al TMR0.

bit 2-0: **PS2:PS0: Valor del Divisor de Frecuencia**

Determina el valor del rango del divisor de frecuencias que se aplica. Será distinto según se aplique al TMR0 o al WDT. Los valores son:

PS2	PS1	PS0	Divisor del TMR0	Divisor del 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

2.11.3 El Temporizador principal TMR0

Se trata de un temporizador/contador de 8 bits, que puede actuar de dos formas:

- Como contador de sucesos, que están representados por los impulsos que se aplican al pin *RA4/TOCKI*. Al llegar al valor FFh se desborda el contador y, con el siguiente impulso pasa a 00h, advirtiendo esta circunstancia activando un señalizador y/o provocando una interrupción.
- Como temporizador, se carga en el registro que implementa al recurso un valor inicial se incrementa con cada ciclo de instrucción ($F_{osc}/4$) hasta que se desborda y avisa poniendo a 1 un bit señalizador y/o provocando una interrupción.

El TMR0 se comporta como un registro de propósito especial (SFR) ubicado en la posición 1h de la memoria de datos. En la misma dirección, pero en el banco 1, se encuentra el registro OPTION_REG, con cuyos bits se configura el TMR0.

TMR0 puede ser leído y escrito al estar conectado directamente al bus de datos. Funciona como un contador ascendente de 8 bits. Cuando funciona como temporizador, conviene cargarle con el valor de los impulsos que se desean contar pero expresados en complemento a 2. De esta manera, al llegar el número de impulsos deseado se desborda y al pasar por 00h se activa el señalizador *T0IF* y/o se produce una interrupción.

Para calcular los tiempos a controlar con TMR0 se utilizan las siguientes fórmulas prácticas:

$$\textbf{\textit{Temporización} = 4 \cdot T_{osc} \cdot (\textit{Valor cargado en TMR0}) \cdot (\textit{Rango del Divisor})}$$
$$\textbf{\textit{Valor a cargar en TMR0} = (\textit{Temporización} / 4 \cdot T_{osc}) \cdot (\textit{Rango del Divisor})}$$

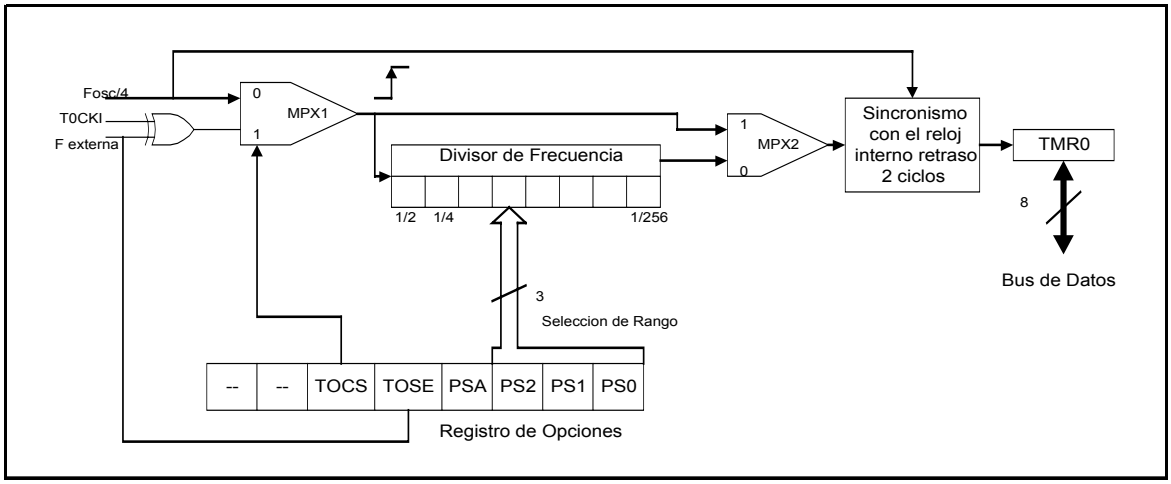


Figura 2.15. *Esquema de funcionamiento del Temporizador Principal TMR0*

En la Figura 2.15. se ofrece el esquema de funcionamiento del Temporizador Principal. Obsérvese que existe un bloque que retrasa dos ciclos y cuya misión consiste en sincronizar el momento del incremento producido por la señal *T0CKI* con el que producen los impulsos del reloj interno. Cuando se escribe TMR0 se retrasan 2 ciclos su reincremento y se pone a 0 el Divisor de Frecuencia. Si no se emplea el Divisor de Frecuencia, la entrada de la señal de reloj externa es la misma que la salida de dicho Divisor.

2.11.4 El Perro Guardián o Watchdog

También se trata de un contador de 8 bits que actúa como temporizador y tiene como objeto el generar un reset a todo el sistema cuando se desborda su valor. Su control de tiempos es independiente del oscilador principal y se basa en una red RC.

La temporización nominal con la que se halla programado el Perro Guardián es de 18 ms, pero utilizando el Divisor de Frecuencia llegar a alcanzar 2,3 segundos.

Para evitar que se desborde el Perro Guardián y genere un reset, hay que recargar o refrescar su cuenta antes de que llegue el desbordamiento. En realidad este refresco consiste en ponerle a 0 por software con las instrucciones *clrwdt* y *sleep*. El programador debe analizar las instrucciones de la tarea y situar alguna de estas dos instrucciones en sitios estratégicos por los que pasa el flujo de control antes de que transcurra el tiempo que controla el Perro Guardián. De esta manera, si el programa se “cuelga” (bucle infinito, espera de acontecimiento que no se produce, etc.), no se refresca el Perro Guardián y se produce una reinicialización del sistema.

La instrucción *clrwdt* simplemente borra el valor de Watchdog y reinicia la cuenta. Sin embargo, la instrucción *sleep*, además de borrar el WDT, detiene el sistema y lo mete en un estado de "reposo" o "de bajo consumo". Si no se desactiva al Perro Guardián cuando se entra en el modo de reposo, al acabar su conteo provocará un reset y sacará al microcontrolador del modo de bajo consumo Para desactivar al Perro Guardián, hay que poner un 0 en el bit 2 (*WDTE*) del registro de la Palabra de configuración.

En el registro STATUS existe un bit denominado *TO#* que pasa a valer 0 después del desbordamiento del WDT. Este flag será el indicador de que la causa del reset fue el desbordamiento del Perro Guardián.

2.12 LOS PUERTOS DE E/S

El PIC16F84A dispone de dos puertos de E/S.

- **Puerto A.-** Posee 5 líneas *RA0:RA4*, y una de ellas soporta dos funciones multiplexadas. Se trata de la *RA4/T0CKI*, que puede actuar como línea de E/S o como pin por la que se reciben los impulsos que debe contar el TMR0.
- **Puerto B.-** Tiene 8 líneas, *RB0:RB7*, y también una con funciones multiplexadas, la *RB0/INT*, que, además de línea típica de E/S, también sirve como pin por la que se reciben los impulsos externos que provocan una interrupción.

Cada puerto dispone de dos registros:

- **Registro de Datos denominados PORTA ó PORTB**.- Se pueden leer o escribir según que el puerto correspondiente se utilice como entrada o como salida. Ocupan las direcciones 5 y 6 del banco 0 de la Memoria de Datos.
- **Registro de Control denominado TRISA ó TRISB**.- En los registros de Control se programa el sentido de funcionamiento de cada una de las líneas de E/S. Colocando un "0" en el correspondiente bit del registro TRISA ó TRISB, la línea queda programada como salida, mientras que colocando un "1" la línea queda programada como entrada. Ocupan las posiciones 85h y 86h (banco 1) de la memoria de datos.

Cualquier línea puede funcionar como entrada o como salida. Sin embargo, si actúa como entrada la información que se introduce no se memoriza, por lo que la información debe ser mantenida hasta que sea leída. Si la línea actúa como salida, el bit que procede del bus de datos se guarda en el latch, con lo que la información que ofrece este pin permanece invariable hasta que se rescriba este bit.

La máxima corriente que puede absorber y suministrar cada línea individual es 25 y 20 mA respectivamente. La máxima corriente que puede absorber el puerto A es de 80 mA y suministrar 50 mA, mientras que para el puerto B son respectivamente 150 mA y 100 mA.

Cuando se produce un reset todas las líneas se programan automáticamente como entradas. Todos los pines de E/S que no se empleen deben ser llevadas a Vcc (regla de las entradas CMOS).

2.12.1 El Puerto A

Como ya se ha indicado anteriormente, está formado por 5 líneas. Las líneas RA3:RA0 admiten niveles de entrada TTL y de salida CMOS. La línea RA4/T0CKI está conectada a un Schmitt Trigger que le permite multiplexar su función de E/S con la entrada de impulsos externos para el TMR0. Cuando se configura como salida es de colector abierto, por lo que debe colocarse una resistencia de polarización.

En la Figura 2.16. se muestra el diagrama de bloques de las líneas *RA3:RA0*. Cuando se lee una línea del Puerto A (instrucción movf porta,w) se recoge el nivel lógico que tiene en ese momento. Cada línea posee un latch, de forma que cuando actúan como salidas el pin saca el nivel lógico que se haya cargado por última vez en el registro PORTA. La escritura de una línea de un puerto implica la operación "lectura/modificación/escritura" Primero se lee el puerto, luego se modifica el valor y finalmente se escribe en el latch de salida.

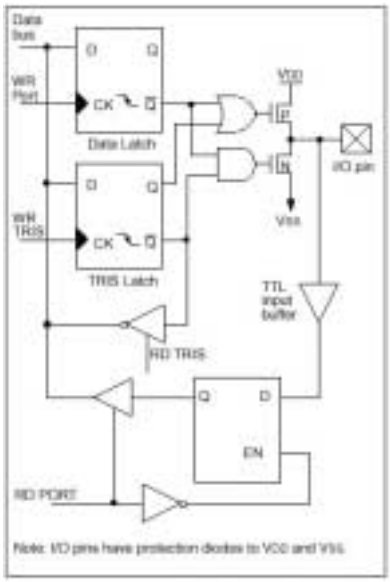


Figura 2.16. *Diagrama de bloques de RA3:RA0*

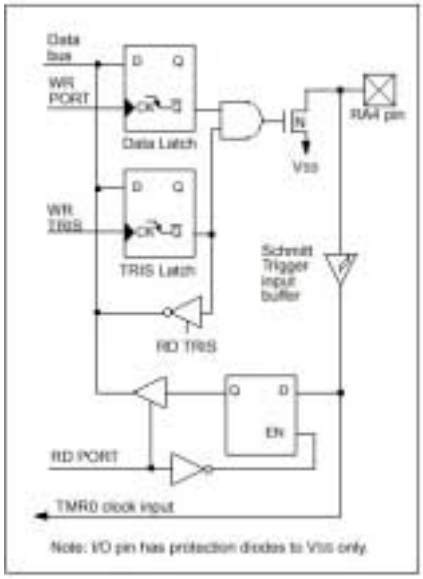


Figura 2.17. *Diagrama de bloques de RA4*

Para sacar un determinado nivel lógico por una línea de un puerto de E/S, se deposita el mismo en la línea correspondiente del bus de datos interno y se activa la señal WRITE, lo que origina el almacenamiento de dicho bit en el latch de datos. En esta situación, el latch de control de E/S debería contener un 0 (salida). Con estos valores iniciales, la puerta OR tendría un 0 en su salida, al igual que la puerta AND. Estas salidas producen la conducción del transistor PMOS superior y el bloqueo del NMOS. Así, el pin de E/S queda conectada a V_{DD} y soporta un nivel alto. Además, la línea, conserva su valor hasta que se desee sacar otro y se rescriba el latch D.

Cuando una línea de E/S actúa como entrada, al activar la señal *READ*, el nivel lógico depositado desde el exterior en el pin pasará a la línea correspondiente del bus de datos interno y empezará a conducir el dispositivo triestado que les une. Al programarse como línea de entrada, los dos transistores MOS de salida quedan bloqueados y la línea queda en

alta impedancia. Hay que destacar que, en este caso en que la línea está configurada como entrada, el estado que se obtiene es el hay en el pin correspondiente y no el valor que haya almacenado en el latch.

2.12.2 El Puerto B

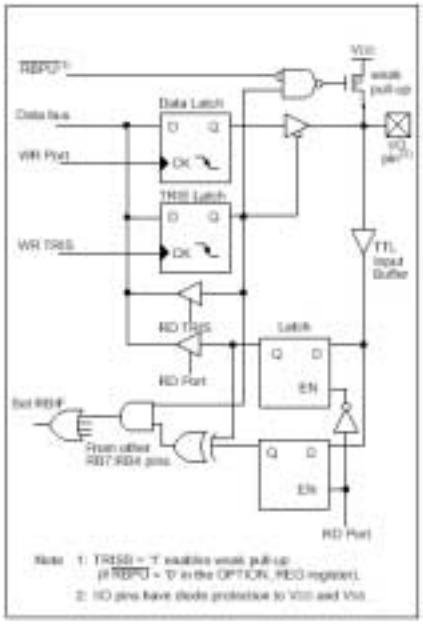


Figura 2.18. *Diagrama de bloques de RB7:RB4*

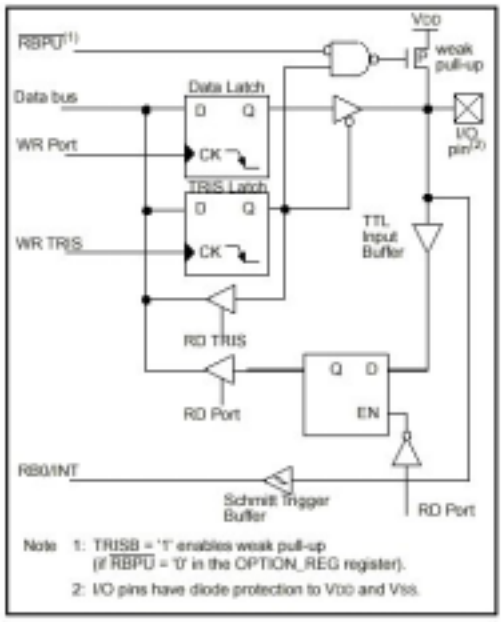


Figura 2.19. *Diagrama de bloques de RB3:RB0*

Consta de 8 líneas bidireccionales de E/S, *RB7:RB0*. La línea *RB0/INT* tiene dos funciones multiplexadas. Además de línea de E/S, puede actuar como entrada de interrupción externa; para ello, es necesario autorizar esta función mediante la adecuada programación del registro INTCON, del que se hablará en el apartado 2.15.1.

A todas las líneas de este puerto se las permite conectar una resistencia pull-up de elevado valor con el positivo de la alimentación. Para este fin, hay que programar en el registro OPTION_REG el bit *RBPU#* a 0, afectando la conexión de la resistencia a todas las líneas. Con el reset todas las líneas quedan configuradas como entradas y se desactivan las resistencias pull-up.

Las líneas *RB7:RB4*, cuando actúan como entradas se las puede programar para generar una interrupción si alguna de ellas cambia de estado lógico. Para ello, se compara el valor con el antiguo que tenían y que se había grabado en el latch durante la última lectura del Puerto B. El cambio de estado en alguna de las líneas origina una interrupción y la activación del señalizador *RBIF*.

La línea *RB6* también se utiliza para la grabación serie de la memoria de programa y sirve para soportar la señal de reloj. La línea *RB7* constituye la entrada de los datos en serie.

2.13 LA PALABRA DE CONFIGURACIÓN

Se trata de una posición reservada de la memoria de programa situada en la dirección 2007h y accesible únicamente durante el proceso de grabación. Al escribirse el programa de la aplicación es necesario grabar el contenido de esta posición de acuerdo con las características del sistema. Está formada por 14 bits distribuidos de la siguiente manera:

PALABRA DE CONFIGURACION

CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE#	WDTE	FOSC1	FOSC0
bit13													bit0

Veamos en detalle cada uno de los bits:

bit 1-0: **FOSC1:FOSC0: Selección del oscilador utilizado**

FOSC1	FOSC0	Tipo Oscilador
1	1	Oscilador RC
1	0	Oscilador HS
0	1	Oscilador XT
0	0	Oscilador LP

bit 2: **WDTE: Activación del Perro Guardián**
1 = Activado el WDT.
0 = Desactivado.

bit 3: **PWRTE: Activación del Temporizador "Power-up"**
1 = Desactivado.
0 = Activado.
El temporizador "power-up" retrasa 72 ms la puesta en marcha o Reset que se produce al conectar la alimentación al PIC para garantizar la estabilidad de la tensión aplicada.

bit 13:4 **CP: Bits de protección de la memoria de código**
1 = No protegida
0 = Protegida. El programa no se puede leer ni sobrescribir. Además evita que pueda ser accedida la EEPROM de datos y, finalmente, si se modifica el bit CP de 0 a 1, se borra completamente la EEPROM

2.14 LAS PALABRAS DE IDENTIFICACIÓN (ID)

Son 4 posiciones reservadas de la memoria de programa ubicadas en las direcciones 2000h-2003h que no son accesibles en el funcionamiento normal del microcontrolador y sólo pueden ser leídas y escritas durante el proceso de grabación.

Sólo se utilizan los cuatro bits menos significativos de cada palabra de identificación, en donde se almacena un valor que puede consistir en un número de serie, códigos de identificación, numeraciones secuenciales o aleatorias, etc.

2.15 LAS INTERRUPCIONES

Las interrupciones son desviaciones del flujo de control del programa originadas asíncronamente por diversos sucesos que no se hallan bajo la supervisión de las instrucciones. Dichos sucesos pueden ser externos al sistema, como la generación de un flanco o nivel activo en un pin del microcontrolador, o bien internos, como el desbordamiento de un contador.

El comportamiento del microcontrolador ante la interrupción es similar al de la instrucción *CALL* de llamada a subrutina. En ambos casos se detiene la ejecución del programa en curso, se salva la dirección actual del PC en la pila y se carga el PC con una dirección, que en el caso de *CALL* viene acompañando a la propia instrucción, y en el caso de una interrupción es una dirección reservada de la memoria de código, llamada **Vector de interrupción**.

En el PIC16F84A el Vector de Interrupción se encuentra situado en la dirección 04h, en donde comienza la **Rutina de Servicio a la Interrupción**. En general, en este vector se suele colocar una instrucción de salto incondicional (*GOTO*), que traslada el flujo de control a la zona de la memoria de código destinada a contener la rutina de atención a la interrupción.

El PIC16F84A puede ser interrumpido por cuatro causas diferentes, pero todas ellas desvían el flujo de control a la dirección 04h, por lo que la primera operación que debe realizarse en la subrutina de atención a la interrupción es averiguar qué causa fue la que la produjo.

Otro detalle es que este microcontrolador posee un bit llamado ***GIE (Global Interrupt Enable)*** en el registro INTCON que cuando vale 0 prohíbe todas las interrupciones. Así, al comenzar una rutina de atención a la interrupción *GIE* pasa automáticamente a 0, con el objeto de no atender nuevas interrupciones hasta que se termine la que ha comenzado. En el retorno de la interrupción, *GIE* pasa a valer nuevamente 1.

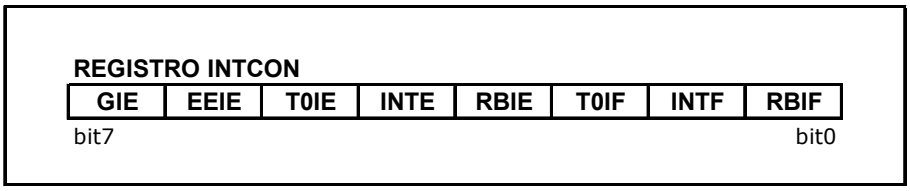
Las cuatro causas o fuentes de interrupción posibles son:

- Activación del pin *RB0/INT*.
- Desbordamiento del temporizador TMR0.
- Cambio de estado en uno de los 4 pines *RB7:RB4* del Puerto B.
- Finalización de la escritura en la EEPROM de datos.

Cuando ocurre cualquiera de los 4 sucesos indicados se origina una petición de interrupción, que si se acepta y se atiende comienza depositando el valor del PC actual en la pila.

2.15.1 El Registro INTCON

La mayor parte de los flags y bits de permiso de las fuentes de interrupción del PIC16F84A están implementados sobre los bits del registro INTCON, que ocupa la dirección 0Bh del banco 0 y la 8Bh del banco 1.



Veamos los bits que forman este registro más detenidamente:

bit 0: **RBIF: Flag de cambio de estado en los pines *RB7:RB4***
1 = Ha cambiado el estado de alguna de estas líneas.
0 = No ha cambiado.

bit 1: **INTF: Flag de activación del pin *RB0/INT***
1 = Se ha activado.
0 = No se ha activado.

bit 2: **T0IF: Flag de desbordamiento de TMR0**
1 = Ha habido desbordamiento.
0 = No se ha producido desbordamiento

bit 3: **RBIE: Activación de la interrupción por cambio de estado en *RB7:RB4***
1 = Activada.
0 = Desactivada.

bit 4: **INTE: Activación de la interrupción en el pin *RB0/INT***
1 = Activada.
0 = Desactivada.

bit 5: **T0IE: Activación de la interrupción TMR0**
1 = Activada.
0 = Desactivada.

bit 6: **EEIE: Activación de la interrupción por fin de escritura en la EEPROM**
1 = Activada.
0 = Desactivada.

bit 7: **GIE: Permiso global de las interrupciones**
1 = Activada.
0 = Desactivada.

Cuando *GIE*=0 no se acepta ninguna de las interrupciones; solamente si *GIE*=1 se aceptan aquellas fuentes de interrupción cuyo bit de permiso lo autorice.

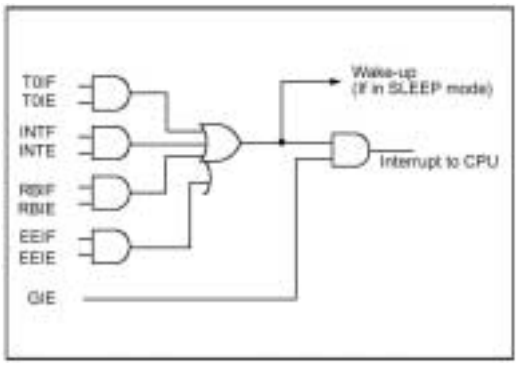


Figura 2.20. *Lógica de control para la generación de una interrupción*

Para conocer qué causa ha provocado la interrupción se exploran los flags y deben ponerse a 0 por programa antes del retorno de la interrupción y son operativos aunque la interrupción correspondiente se encuentre desactivada. En la Figura 2.20 se muestra el esquema de la lógica de control que origina la interrupción.

2.15.2 Interrupción Externa por el pin *RB0/INT*

Esta fuente de interrupción permite atender acontecimientos externos en tiempo real. Cuando ocurre alguno de ellos llega la señal al pin *RB0/INT* y se hace una petición de interrupción. Entonces, de forma automática, el bit *INTF* pasa a 1, y si el bit *INTE* también es 1, se autoriza el desarrollo de la interrupción.

Mediante el bit *INTEDG* (bit 6) del registro OPTION_REG se puede seleccionar cuál será el flanco activo en *RB0/INT*. Si se desea el ascendente se escribe un 1 en dicho bit y un 0 en caso contrario.

2.15.3 Interrupción por desbordamiento en TMR0

Cuando TMR0 se desborda, es decir, pasar del valor FFh al valor 00h, el flag *T0IF* se pone automáticamente a 1. Si además, tanto el bit *T0IE* como *GIE* valen 1, se produce una interrupción.

Si no se recarga el temporizador TMR0 cuando se desborda, sigue contando desde 00h a FFh. En cualquier momento se puede leer y escribir este registro, pero cada vez que se escribe se pierden dos ciclos de reloj para la sincronización.

Cuando se carga inicialmente TMR0 con un valor N, cuenta 256-N impulsos, siendo el tiempo que tarda en hacerlo el que expresa la siguiente ecuación:

Temporización = 4 Tosc (256-N) Rango del Divisor de Frecuencia

2.15.4 Interrupción por cambio de estado en las líneas *RB7:RB4*

Cuando se produce un cambio en el valor lógico de alguna de las líneas *RB7:RB4*, el flag *RBIF* pasa a 1, y se producirá la interrupción si *RBIE* y *GIE* están a 1. Esta interrupción es muy empleada para el manejo de teclados.

2.15.5 Interrupción por finalización de la escritura en la EEPROM de datos

El tiempo típico que tarda en desarrollarse una operación de escritura en la EEPROM de datos es de 10 ms, que es considerable comparado con la velocidad a la que el procesador ejecuta instrucciones. Para asegurarse que se ha completado la escritura y puede continuarse con el flujo de control del programa es aconsejable manejar la interrupción que se origina al finalizar la escritura, que pone automáticamente el señalizador *EEIF* a 1, y se autoriza siempre que los bits de permiso *EEIE* y GIE estén a 1.

2.16 REINICIALIZACIÓN O RESET

El PIC16F84A tiene cinco causas que provocan la reinicialización del sistema, que consiste en cargar al PC con el valor 0h (Vector de Reset) y poner el estado de los bits de los registros SFRs con un valor conocido. Las cinco causas son:

- Conexión de la alimentación (*Power on Reset*).
- Activación del pin MCLR# en funcionamiento normal (*Master Clear Reset*).
- Activación del pin MCLR# en modo reposo.
- Desbordamiento del perro guardián en funcionamiento normal.
- Desbordamiento del perro guardián en estado de reposo.

En la Figura 2.21 se presenta el estado lógico que adquieren los bits de los registros SFR cuando se provoca un Reset por cualquiera de las causas anteriormente especificadas.

Register	Address	Power-on Reset	MCLR Reset during: – normal operation – SLEEP WDT Reset during normal operation	Wake-up from SLEEP: – through interrupt – through WDT Time-out
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000h	0000h	PC + 1[2]
STATUS	03h	0001 1xxx	000q quuu[3]	uuuq quuu[3]
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA[4]	05h	---x xxxx	---u uuuu	---u uuuu
PORTB[5]	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu[1]
INDF	80h	---- ----	---- ----	---- ----
OPTION_REG	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000h	0000h	PC + 1
STATUS	83h	0001 1xxx	000q quuu[3]	uuuq quuu[3]
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	---1 1111	---u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 x000	---0 q000	---0 uuuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu[1]

u = No cambia x = indeterminado --- = No implementado

Figura 2.21. *Valor de los registros SFR tras el reset*

En el registro STATUS hay dos bits que indican las condiciones en las que se ha originado el Reset. Se trata de *TO#* (*Timer Out*) y *PD#* (*Power Down*)

En la Figura 2.22. se muestra el circuito de generación del reset en el PIC16F84A.

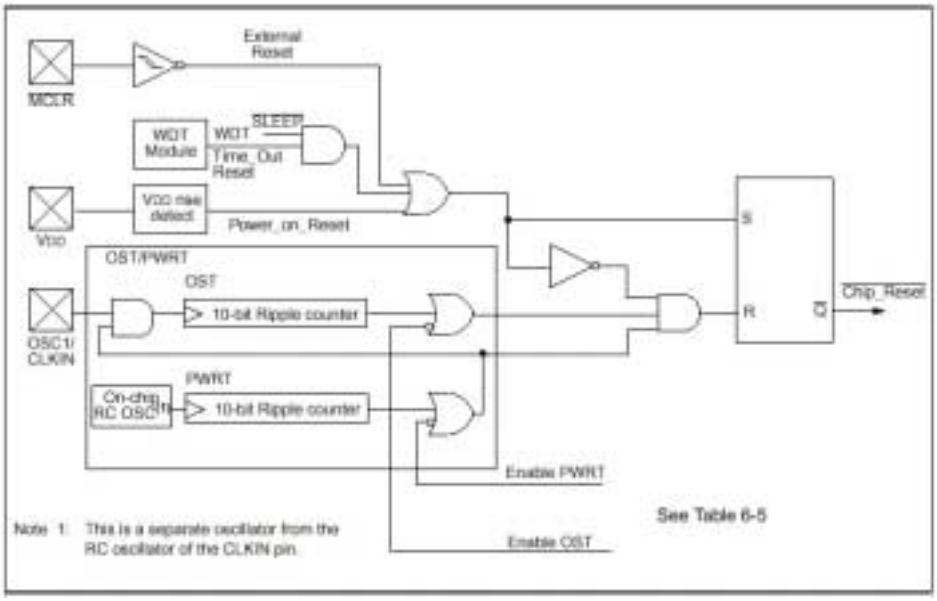


Figura 2.22. *Circuito para la generación interna del reset*

2.17 EL MODO DE REPOSO O DE BAJO CONSUMO

Este modo de funcionamientos de los PIC está caracterizado por el reducido consumo de energía que requiere y es recomendable en aquellas aplicaciones en las que hay largos períodos de espera hasta que se produzca algún suceso asíncrono como, por ejemplo, la pulsación de una tecla. En dichos períodos el procesador está inactivo.

Para entrar en el estado de reposo hay que ejecutar la instrucción **SLEEP**, tras lo cual el microcontrolador entra en un estado de letargo en el que el requerimiento de energía que hay que suministrarle es mínimo. Así, si el consumo típico del PIC es de 2 mA aproximadamente en el estado de Reposo se reduce a menos de 10 µA.

Para salir del estado de reposo existen tres alternativas:

- Activación externa de *MCLR#* para provocar un Reset.
- Desbordamiento del perro guardián si quedó operativo en el modo de sleep.
- Generación de una interrupción.

Para finalizar este capítulo, les recomendamos que para una información más detallada sobre el funcionamiento del PIC16F84A consulte la hoja de especificaciones de este microcontrolador que viene incluida en este Proyecto Fin de Carrera en el apartado "Apéndice D".

CAPITULO 3
EL PROGRAMA MPLAB

3.1 EL PROGRAMA MPLAB

El MPLAB es un software que, junto con un emulador y un programador de los múltiples que existen en el mercado, forman un conjunto de herramientas de desarrollo muy completo para el trabajo y/o el diseño con los microcontroladores PIC.

El MPLAB incorpora todas las utilidades necesarias para la realización de cualquier proyecto y, para los que no dispongan de un emulador, el programa permite editar el archivo fuente de nuestro proyecto en leguaje ensamblador, además de ensamblarlo y simularlo en pantalla, pudiendo ejecutarlo posteriormente paso a paso y ver cómo evolucionarían de forma real tanto sus registros internos, la memoria RAM y/o EEPROM de usuario como la memoria de programa, según se fueran ejecutando las instrucciones. Además, el entorno que se utiliza es el mismo que si se estuviera utilizando un emulador.

La versión que ha sido empleada en el desarrollo de este proyecto es la V.12.00, y se puede obtener junto a un gran número de herramientas gratuitas en la dirección de Internet de la empresa Microchip: http://www.microchip.com.

3.2 EXPLICACIÓN DE LA PANTALLA DE TRABAJO

La pantalla de trabajo está dividida en varias zonas, cumpliendo cada una de ellas una función asignada. En la siguiente figura se muestra cada una de dichas zonas:

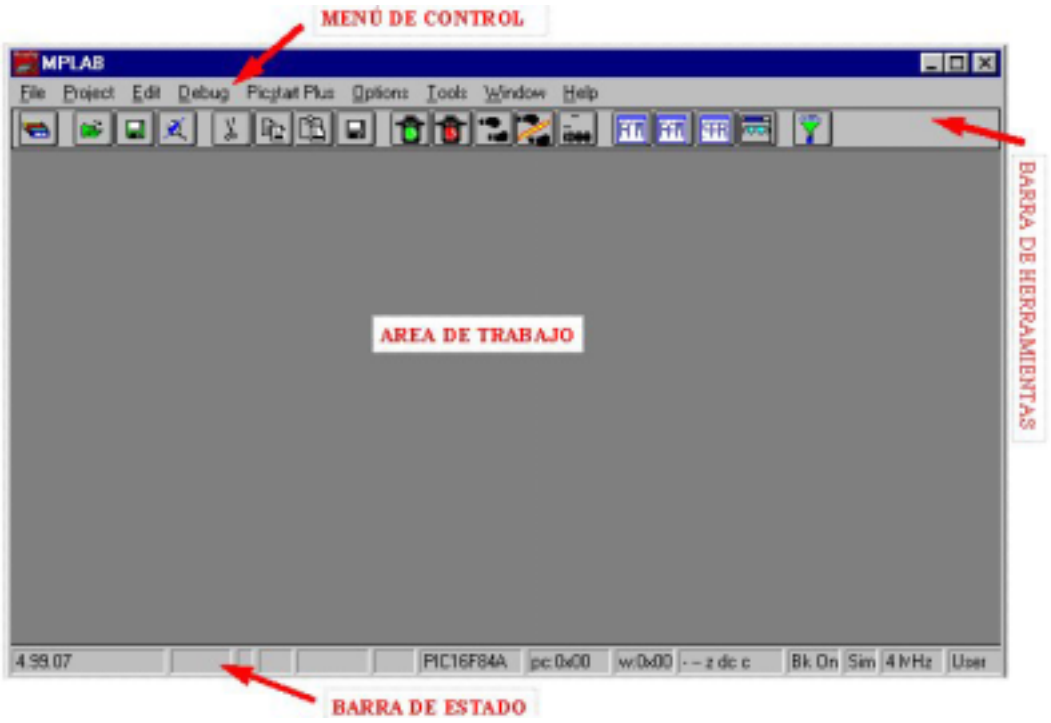


Figura 3.1. *Escritorio del MPLAB*

MENÚ DE CONTROL

Pulsando sobre las palabras contenidas en él se despliega un menú con una serie de funciones. Aparecen nueve posibilidades: *File, Project, Edit, Debug, Picstart Plus, Options, Tools, Windows, Help*. Por ejemplo, *File* contiene *New, Open, View, Save*,...

BARRA DE HERRAMIENTAS

Los iconos que aparecen en la barra de herramientas, son funciones que se encuentran incluidas en el menú de control, pero están situadas en esta barra para acceder directamente a ellas sin tener que buscarlas en el menú desplegable, facilitando la tarea y ahorrando tiempo.

El MPLAB contiene cuatro paletas distintas entre las que se puede conmutar simplemente pulsando en el icono  de cualquiera de las barras herramientas:

- Barra de edición
- Barra de depuración
- Barra de proyecto
- Barra pesonalizable por cada usuario

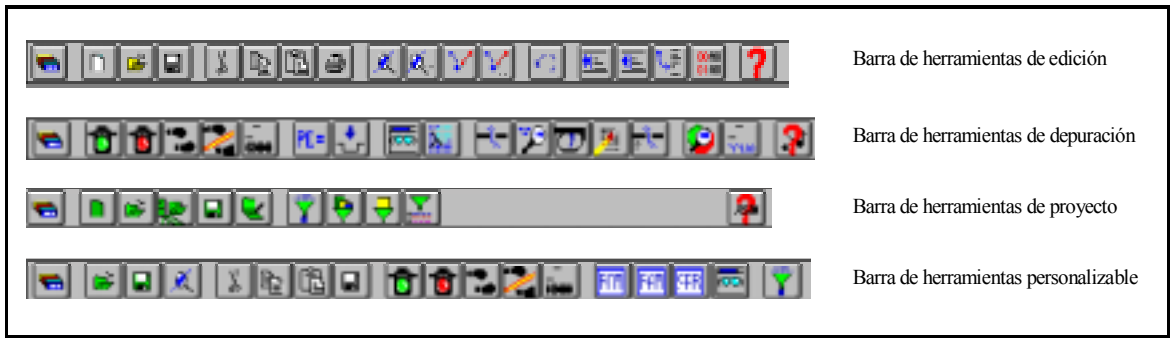


Figura 3.2. *Barras de herramientas*

ÁREA DE TRABAJO

Como su nombre indica es la zona de la pantalla en la que se realizan los programas. Puede haber varias ventanas abiertas, correspondiendo al programa principal o a cualquier ventana que se abra desde el menú *Windows*.

BARRA DE ESTADO

Se sitúa en la parte inferior y muestra información sobre los distintos menús cuando se sitúa el ratón sobre cada uno de ellos, el tipo de microcontrolador, la frecuencia,...

3.3 MÉTODO DE TRABAJO

Lo primero que debe realizarse cuando vayamos a trabajar con este programa es seleccionar el modo de trabajo como simulador y el tipo de microcontrolador con el que queremos trabajar. Para ello, se selecciona el botón de ***Options -> Development Mode*** del menú de control y en la ventana que aparece (véase Figura 3.3) se activa el modo ***MPLAB-SIM Simulator*** y el microcontrolador que se va emplear (en este caso, el PIC16F84A). Después, se pulsa ***OK***.

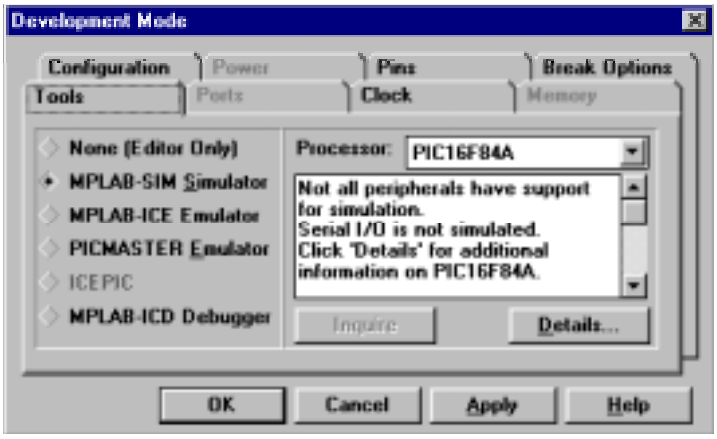


Figura 3.3. *Ventana Development Mode*

Una vez realizada esta operación, podemos iniciar el desarrollo de nuestro programa. Es de destacar que para trabajar en el MPLAB es necesario generar un proyecto, que simplemente es un conjunto de ficheros que se compilan y linkan con ciertas herramientas para poder implementar una aplicación determinada.

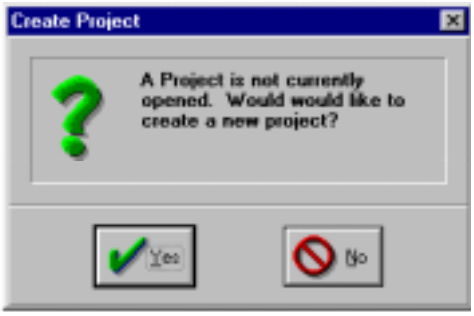


Figura 3.4. *Mensaje Proyecto no abierto*

Para comenzar a escribir nuestro programa seleccionamos en el menú de control ***File -> New*** o bien activamos el icono de ***Crear nuevo documento***  en la barra de herramientas. El programa contestará con un cuadro de diálogo como el de la Figura 7.4, con el que nos indica que no existe ningún proyecto abierto.

Activamos el botón ***Yes*** y automáticamente se abre una ventana de diálogo (Figura 3.5.) en el que se pregunta el nombre del proyecto y donde queremos salvarlo. Así, el programa devuelve un cuadro de diálogo similar al de la Figura 3.6. Presionamos ***OK*** y estamos en condiciones de

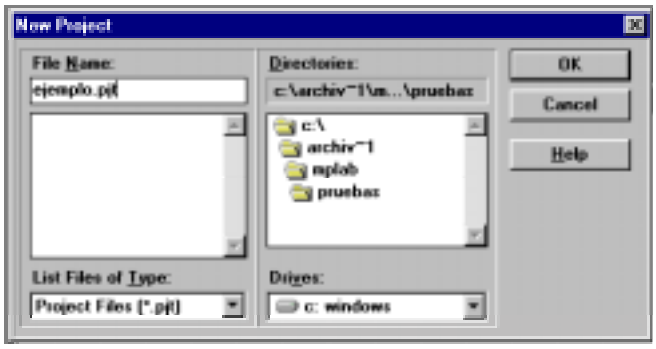


Figura 3.5 *Creación Proyecto*

empezar a escribir nuestro programa. Para ello, empleamos el editor de textos. Hay que tener en cuenta que todos los programas escritos en ensamblador llevan la extensión ****.asm.***

Figura 3.6. *Propiedades de edición del proyecto*

Una vez que terminemos de escribir nuestro programa seleccionamos ***File-> Save***, con lo que aparece un cuadro de diálogo como el de la Figura 3.7, donde le damos el nombre que deseemos guardar el programa.

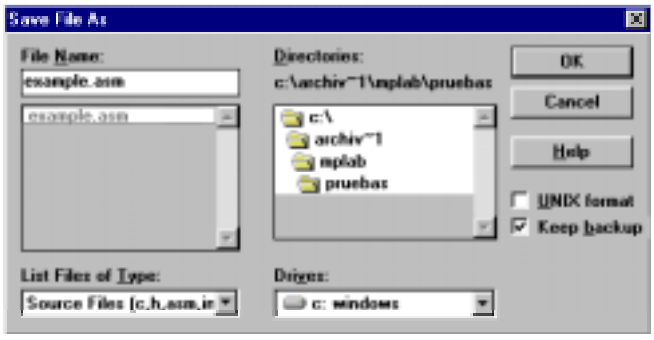


Figura 3.7. *Salvado del fichero del programa*

El siguiente paso será seleccionar ***Project -> Edit Project*** en el menú de control y aparecerá una ventana como la de la Figura 3.6. Entonces, seleccionamos el botón ***Add Node***, lo que provoca que aparezca un nuevo cuadro de diálogo en el que seleccionamos nuestro archivo. Pulsamos ***Aceptar*** y luego ***OK***.

Para ensamblar el programa seleccionamos en el menú de control la opción ***Project -> Build All***, y si no se han cometido errores al introducir el código, aparece una pantalla que nos indica que el programa se ha ensamblado con éxito y que ya estamos en condiciones de iniciar la simulación de programa. Si por el contrario, se han detectado errores, en dicha pantalla será mostrado el error; si se hace doble clic sobre la línea que muestra el error, el cursor saltará directamente a la línea de código donde se encuentra. Una vez subsanados los errores habrá que volver a compilar el programa.

Cuando el ensamblado se ha realizado con éxito en el directorio donde hemos guardado nuestro fichero *.asm se generan otros con extensión ****.cod***, ****.err***, ****.hex*** y ****.lst***.

- El fichero *.cod contiene código objeto.

- El *.err recoge los errores que se hayan podido generar durante la compilación.

- El *.lst muestra el código fuente en modo absoluto con el código objeto generado.

- El fichero *.hex contiene el código en formato hexadecimal y será con el que programaremos el microcontrolador para que realice la aplicación deseada.

3.4 EL EDITOR MPLAB

El editor permite a los programadores escribir y editar código fuente, así como otros archivos de texto.

Debemos tener en cuenta que la primera columna del editor está reservada para las **etiquetas** que son expresiones alfanuméricas escogidas por el usuario que definen los valores de posiciones de memoria. Éstas deben empezar siempre por una letra.

En las siguientes columnas se puede comenzar a escribir el nemónico de la instrucción o las directivas de ensamblador.

Por último hay que decir que se pueden añadir comentarios al texto, pero deben estar precedidos por un "**;**" (punto y coma).

3.5 SIMULACIÓN DE UN PROGRAMA

Una vez que hemos compilado un proyecto en MPLAB podemos emplear un programador, programar un microcontrolador y verificar que la aplicación funciona como se esperaba. Normalmente, una aplicación no funciona correctamente la primera vez y será necesario depurar el código. Para asistir al usuario en la depuración del software, Microchip ha diseñado la herramienta ***MPLAB-SIM*** y su empleo se selecciona, como ya indicamos en el apartado 3.3, en la opción del menú de control ***Options -> Development Mode***.

Cuando se dice que el sistema corre en ***tiempo real*** en el modo simulador, las instrucciones se ejecutan tan rápido como al software le resulta posible. Esta velocidad normalmente es más lenta que lo que en realidad podrían correr los microcontroladores PIC en su ciclo de reloj.

La velocidad a la cual un simulador corre depende de la velocidad de la computadora en la que se está realizando la simulación y de las demás tareas que se estén ejecutando al mismo tiempo.

El software simulador debe de actualizar todos los registros simulados en memoria RAM, monitorizar las E/S, ajustar y limpiar los flags o banderas y simular las instrucciones de los microcontroladores.

Para realizar la simulación de un programa emplearemos las distintas funciones de la opción del menú de control ***Debug***. Por otro lado, para poder monitorizar el funcionamiento de nuestro programa, el MPLAB pone a nuestra disposición una serie de ventanas que se encuentran englobadas en el menú ***Windows***. A través de estas ventanas podremos tener una visión de todos los registros del microcontrolador en cada momento.

3.5.1 Menú Windows

Al activar esta opción del menú de control, aparece el siguiente menú desplegable:

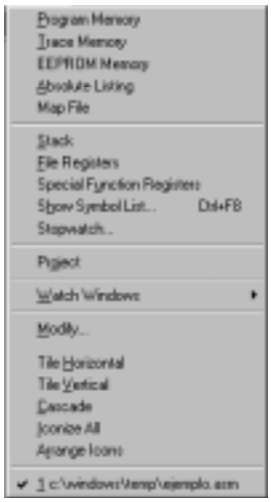


Figura 3.8. *Menú desplegable Windows*

Veamos cada una de las opciones de este menú:

Program Memory

Al seleccionar esta opción se abre una ventana en la que se puede apreciar las posiciones de memoria que ocupa cada una de las instrucciones, el código de operación de cada instrucción y la posición de memoria que se le ha dado a cada etiqueta.

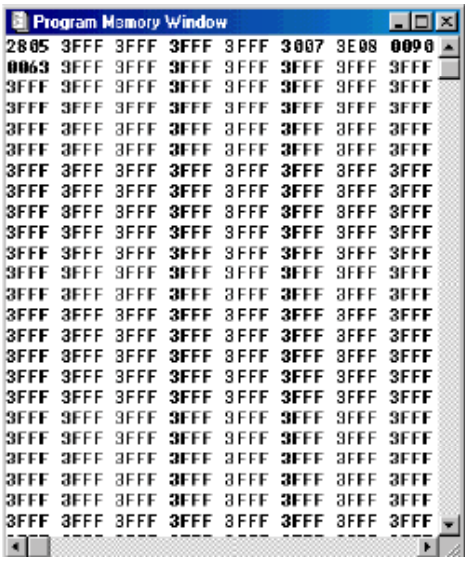


Figura 3.9. *Ventana de Memoria de Programa*

Trace Memory

La ventana de memoria de traza toma "una instantánea" de la ejecución del programa cuando se está conectando en tiempo real.

Antes de activar la opción Trace Memory, para poder obtener los datos en la memoria de traza en el simulador, es necesario marcar con el ratón las líneas de código de programa de las cuales queremos obtener los datos al ejecutarse el programa. Seguidamente, se pulsa el botón de la derecha del ratón, de manera que aparece un menú desplegable en el que elegimos la opción ***Trace Point(s).*** Así, las líneas seleccionadas anteriormente aparecen resaltadas en color verde.

A continuación, seleccionamos ***Debug -> Run -> Run*** (o el icono  de la barra de herramientas), lo que hará ejecutar la simulación en "tiempo real" (no olvidemos que en el simulador emula el funcionamiento del microcontrolador y es mucho más lento que éste). Después de unos segundos, seleccionamos ***Debug -> Run -> Halt the processor*** (o el icono ) para detener la selección. Entonces, si activamos ***Windows -> Trace Memory***, podemos observar las trazas obtenidas. Como ejemplo, en la Figura 3.10 se pueden observar el tiempo que se tarda en ejecutar esas líneas de programa y cualquier variación sobre los registros al ejecutarse el código de instrucción.

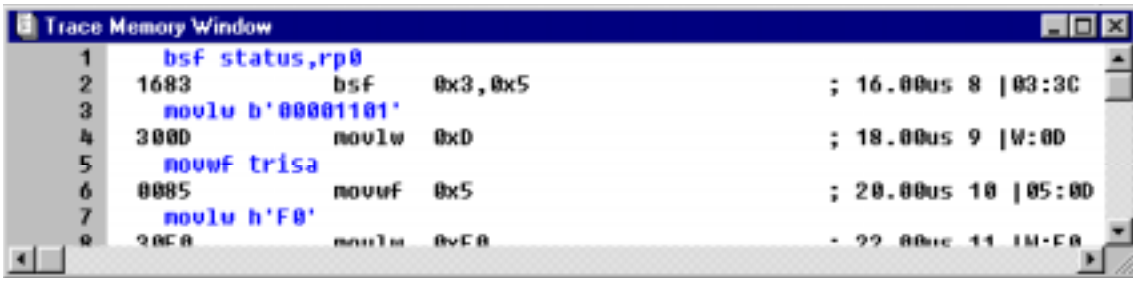


Figura 3.10. *Ventana Traza de memoria*

EEPROM Memory

Si el dispositivo emulado tiene EEPROM o memoria Flash, el contenido de la EEPROM puede visualizarse seleccionando ***Window -> EEPROM***.

Absolute Listing

La ventana de "Listado de Programa" realmente nos presenta el archivo de extensión *.lst de nuestro proyecto. El listado muestra el código fuente en modo absoluto. Además, al final de este archivo aparece la información de las etiquetas utilizadas en el programa, en qué línea se encuentran, la memoria utilizada, la memoria libre y los errores y warnings reportados por el ensamblador.

Stack

Permite visualizar el contenido de la pila. Éste puede mostrarse con o sin número de línea. El formato de presentación se elige pulsando el botón de la esquina superior izquierda de la ventana (llamado menú del sistema), opción ***Toggle Line Numbers***

Si la pila se desborda, el MPLAB lo indica con el mensaje ***underflow***.

File Registers

Con esta ventana podemos visualizar la lista de todos los Registros de Propósito General (GPR) del microcontrolador.

El listado de registros puede presentarse de tres maneras distintas. El formato deseado se elige a través del menú del sistema.

- **ASCII Display.-** Presenta el listado en código ASCII

- **Hex Display**.- Presenta el listado en formato hexadecimal

- **Symbolic Display**.- Este formato presenta un archivo con los Registros de Propósito General con sus etiquetas si las tienen y su contenido en hexadecimal, decimal, binario y formato carácter (Véase Figura 3.11.)

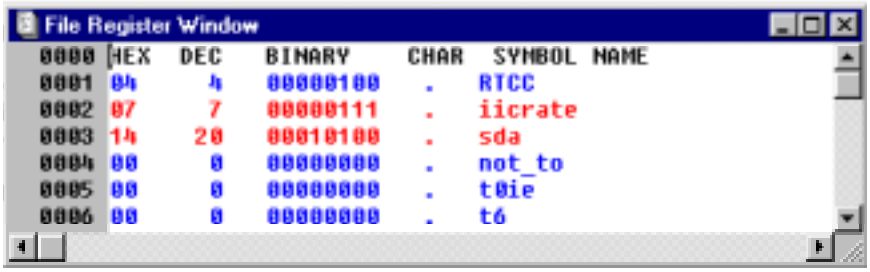


Figura 3.11. *Ventana en formato Symbolic Display de los registros de propósito general*

Special Function Registers (SFRs)

Presenta el contenido de los registros de funciones(SFRs). En esta ventana se muestra cada uno de los registros con el nombre que tiene asignado además de su contenido en distintos códigos: hexadecimal, decimal, binario y formato ASCII.

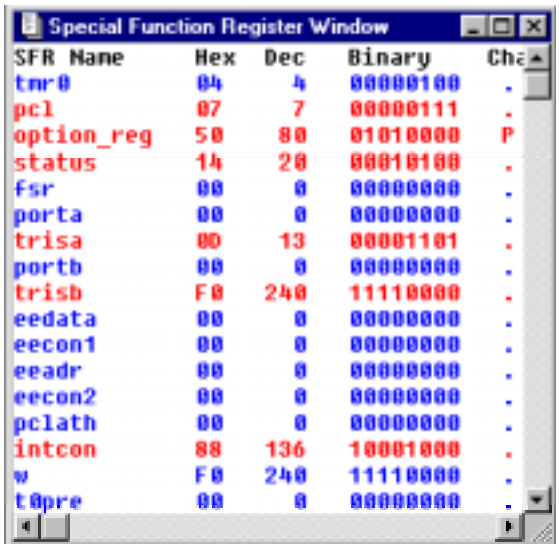


Figura 3.12. *Ventana de los registros especiales*

Show Symbol List

Esta ventana muestra un listado de los símbolos, es decir, variables y etiquetas utilizadas en el código fuente del programa. Estos símbolos están en el archivo *.cod de nuestro proyecto.

Stopwatch

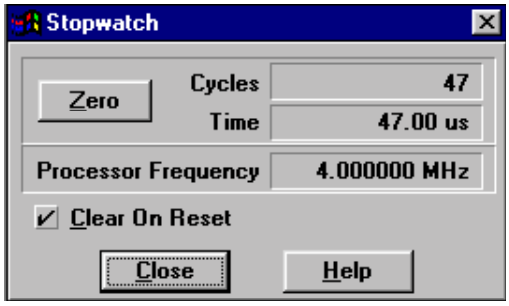


Figura 3.13. *Ventana StopWatch*

Para calcular el tiempo de ejecución de nuestro programa o de una subrutina, podemos contar el número de instrucciones que se realizan y multiplicarlo por 4 veces la frecuencia de la señal de reloj o por 8 en el caso de que sea una instrucción de salto. El Stopwatch o cronómetro calcula el tiempo basándose en la frecuencia con la que trabaje el microcontrolador que estamos simulando.

Para fijar la frecuencia de reloj seleccionamos ***Options -> Processor Setup -> Clock Frecuency***.

Después abrimos la ventana Stopwatch donde, como se puede observar en la Figura 3.13, se nos muestra el tiempo transcurrido y los ciclos de máquina empleados en la ejecución de cada instrucción.

Project

Esta ventana de proyecto sólo está disponible cuando un proyecto se encuentra abierto y presenta la lista de los archivos que le pertenecen. Por otra parte, aparecen resaltados en un azul claro los archivos asociados al proyecto como nodos del mismo. Así, al hacer un doble clic sobre ellos se abren automáticamente.

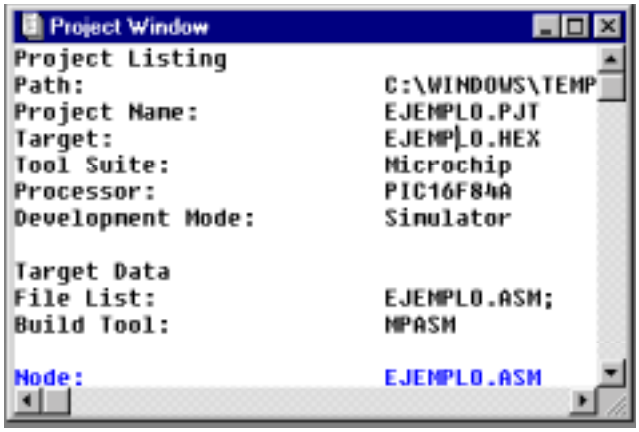


Figura 3.14. *Project Window*

Watch Windows

Es una ventana temporal desde la cual se pueden supervisar el contenido de los registros del archivo. Para crear una ventana ***Windows -> Watch Windows -> New Watch Window***. Aparece una ventana como la de la Figura 3.15. .Para añadir los registros a visualizar, se selecciona cada uno de ellos y se da al botón ***Add***. El mismo procedimiento es para retirarlo de la ventana, pero pulsando a continuación ***Delete***. Cuando ya tenemos en la ventana todos los registros deseado pulsamos ***Close*** y aparece una ventana como la de la Figura 3.16.

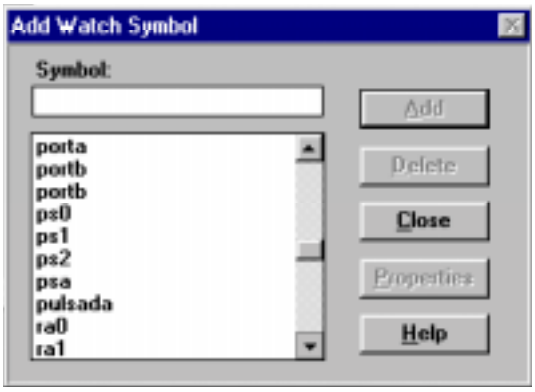


Figura 3.15. *Cuadro de diálogo de los símbolos*

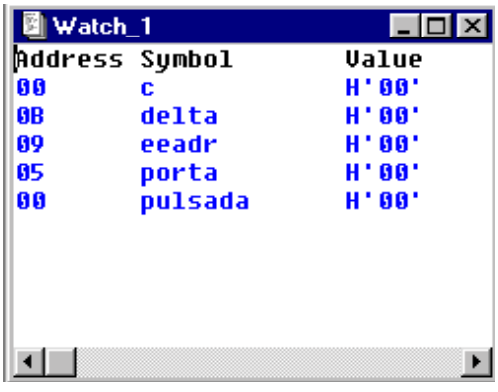


Figura 3.16. *Ventana Watch_1*

Modify

En este cuadro se permite leer y escribir una posición o el rango de una posición de memoria. Modify puede trabajar en las siguientes áreas de memoria:

- Data
- Stack
- Program
- EEPROM (si tiene)

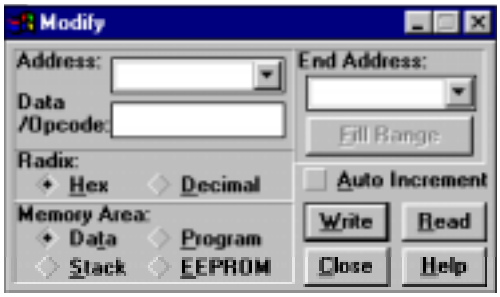


Figura 3.17. *Ventana Modify*

3.5.2 Menú Debug

Cuando seleccionamos el menú de control Debug aparece el siguiente desplegable:

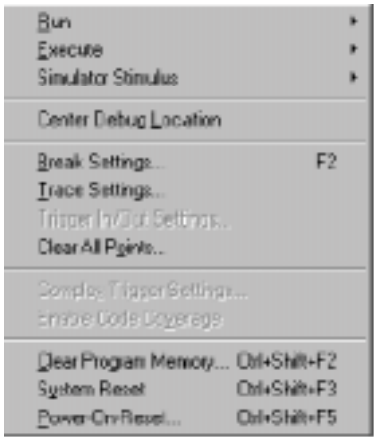


Figura 3.18. *Menú Debug*

3.5.2.1 Run

Está formado por las siguientes opciones:

- Run
- Reset
- Halt
- Animate
- Step

- Step Over
- Update All Registers
- Change Program Counter

Run

Esta opción ejecuta el programa en tiempo real; no obstante, no hay que olvidar que se trata de la simulación de la ejecución de un programa a través de un ordenador y que, por lo tanto, es mucho más lenta. Solamente se parará el programa si encuentra un *Break point* o se activa la opción *Halt*. Por otro lado, mientras el programa está corriendo, el fondo de la barra de estado se pone de color amarillo y no se actualizan los registros.

Otra forma de activar la opción Run es con la tecla F9 o con el icono  .

Reset

Esta opción inicializa el sistema. El Contador de Programa (PC) se pone a cero (a esta dirección de memoria se le denomina vector Reset). La línea del código fuente de esta dirección queda resaltada con una banda negra.

Para activar esta opción se puede pulsar F6 o el icono  de la barra de herramientas.

Halt

Esta opción detiene el programa que se está ejecutando en este momento. También se puede activar pulsando la tecla F5 o el icono  . Al hacerlo, la barra de estado vuelve a pasar al color blanco y se actualizan los registros con los valores correspondientes a la última instrucción ejecutada.

Halt Trace

Permite al emulador cargar el buffer de memoria con los datos de los registros en los punto marcados con un ***Trace Point***.

Step

Esta acción ejecuta la instrucción cuya dirección de memoria coincida con el valor al que apunta el PC antes de activarla. También se puede ejecutar pulsando F7 o el icono  .

Esta opción permite comprobar paso a paso como se ejecuta nuestro programa, y de esta forma, ver si hace lo que nosotros queremos o por el contrario si nos hemos equivocado al escribir el programa y donde no está haciendo lo deseado. *Step* es

especialmente interesante cuando se comienza a estudiar el ensamblador del microcontrolador y se quiere comprobar el repertorio de instrucciones del mismo o bien para depurar programas.

Animate

Esta opción es similar a la anterior con la salvedad de que en este caso en lugar de ejecutar de forma controlada el programa, éste se ejecuta de forma automática desde la dirección que indique el PC en el instante de activarla hasta que finaliza.

Durante la ejecución de *Animate* se puede observar como cambia el valor de los SFRs en la ventana *Special Function Registers* o en una ventana *Watch*. Además, la barra de estado que está en la parte baja de la pantalla pasa a tener un fondo amarillo.

Otra forma de iniciar el modo *Animate* es pulsando simultáneamente las teclas Control+F. Para pararlo debemos activar la opción ***Halt***.

Cuando lo que se quiere es ejecutar el programa y que se detenga en una determinada instrucción para así analizar lo que ha ocurrido hasta ese momento, debemos utilizar un ***Break Point***. La forma más sencilla de introducir uno es ponerse con el ratón en la línea de programa donde queremos colocarlo, pulsar el botón derecho del ratón para que se despliegue un cuadro de acciones y seleccionar *Break Point*. Para poder identificar fácilmente en qué punto se ha introducido un Break Point la línea de programa queda resaltada en rojo.

Si lo que nos interesa es ver cómo evoluciona el microcontrolador al ejecutar una determinada instrucción debemos introducir un ***Trace Point*** en la instrucción deseada antes de iniciar la simulación; el método a emplear es el mismo que para un *Break Point* y en este caso la instrucción pasa a tener un color verde. Para ver la evolución del contenido del registro debemos abrir la ventana ***Window -> Memory Trace***.

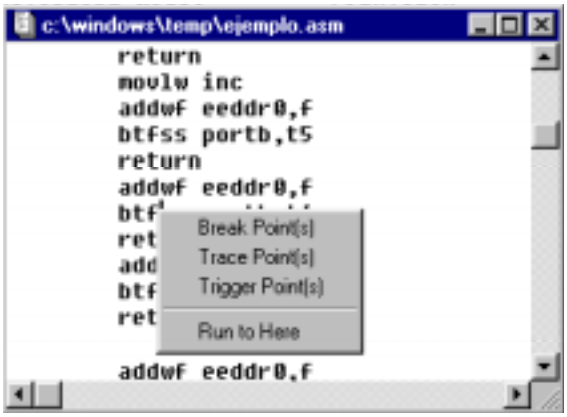


Figura 3.19. *Activación de un Break Point o un Trace Point*

Step Over

Esta opción ejecuta pasa a paso las instrucciones de igual forma que lo hace *Step*, pero cuando llega a la ejecución de una subrutina (instrucción call) ésta se ejecuta de igual forma que si fuera una sola instrucción.

Para activar esta opción, también se puede pulsar la tecla F8 o el icono 

Change Program Counter

Cuando se activa esta opción aparece la siguiente ventana de diálogo:

Figura 3.20. *Cuadro de diálogo para modificar el contador de programa*

En el campo PC se puede poner cualquier zona de memoria o bien, si se activa la flecha, se despliega un menú con todas las etiquetas del programa. Seguidamente se pulsa Change.

Con esta opción podemos realizar saltos y ser empleada en la depuración de un programa. También se puede activar con el icono  .

Update all Registers

Actualiza el contenido de los registros después de haber ejecutado una instrucción.

3.5.2.2 Execute

Está formado por las siguientes opciones:

- Execute an Opcode
- Conditional Break

Execute an Opcode

Activando esta opción se ejecuta una sola instrucción o una serie de instrucciones sin modificar el programa código objeto de la memoria. Después de ejecutar la instrucción, se puede seguir ejecutando el programa desde el estado actual de la memoria de programa.

Conditional Break

Con esta opción se fija un punto de ruptura condicional, el MPLAB se detiene cuando el valor de un registro específico alcanza un valor o una condición prefijada de antemano.

3.5.2.3 Simulator Stimulus

La función de simulación de estímulos simula la generación de señales de entrada en las patillas del microcontrolador. Hay cuatro modos de generar los estímulos y cuyas acciones de forma abreviada son las siguientes:

- ***Asynchronous Stimulus***: Abre un cuadro de diálogo interactivo para controlar las señales de entrada en los pines de entrada del microcontrolador.
- ***Stimulus Pin File***: Mediante un archivo de texto se describen las señales de entrada en los pines.
- ***Stimulus Register File***: Se usa el contenido de un archivo de texto para poner directamente a uno o cero cada uno de los 8 bits de un registro.
- ***Clock Stimulus***: Genera una señal cuadrada programable.

3.5.2.4 Center Debug Location

Esta opción pone en el centro de la pantalla de depuración del programa la posición de memoria que indica el PC.

3.5.2.5 Break Settings

Permite definir hasta 16 Break Points distintos. En el cuadro de diálogo que aparece, después de darle un nombre, introducimos la dirección de inicio y la de final, pulsamos el botón ***Add***. Así, los Break Points se salvarán como parte de nuestro proyecto.

3.5.2.6 Trace Settings

Esta opción es similar a la anterior pero en este caso, en lugar de Break Point, se activan las direcciones de memoria de Traza que se quiere rastrear posteriormente.

3.5.2.7 Clear All Points

Con esta opción se eliminan todos los Break Points, trazas y puntos de disparo seleccionados.

3.5.2.8 Clear Program Memory (Ctrl+Shift+F2)

Esta opción limpia la memoria de programa poniéndola a 0x3FF, es decir, pone todos los bits de memoria de programa a unos.

3.5.2.9 System Reset (Ctrl+Shift+F3)

Si seleccionamos esta función, se inicializa el sistema emulador incluido el hardware del emulador MPLAB-ICE (si está conectado), el software y el microcontrolador.

3.5.2.10 Power On Reset (Ctrl+Shift+F2)

Si seleccionamos Power On Reset en el cuadro de diálogo que aparece al pulsar esta opción, la memoria RAM y los registros SFR se cargan con valores aleatorios, tal y como haría el microcontrolador al realizar un Reset por aplicarle alimentación.

CAPITULO 4
TRANSMISIÓN DE SEÑALES INFRARROJAS

4.1 INTRODUCCIÓN

En este capítulo se detalla el protocolo de transmisión de señales infrarrojas empleados por los mandos a distancia.

Las señales infrarrojas son ondas electromagnéticas con frecuencias inferiores a las de la luz visible. Este tipo de radiaciones electromagnéticas son las empleadas en las llamadas comunicaciones locales, es decir, en aquellas en las que el emisor y el receptor se encuentran situados a distancias cortas. Como ejemplo de este tipo de comunicaciones se pueden citar los vídeos caseros, en los que el mando a distancia será el emisor y el vídeo el receptor. Así, la luz infrarroja es transmitida desde el mando a través del dispositivo denominado led emisor, el cual emite luz cuando le atraviesa una corriente eléctrica. En el vídeo receptor se encuentra otro diodo por el que circula corriente cuando absorbe luz infrarroja.

4.2 MODULACIÓN Y CODIFICACIÓN DE LA SEÑAL

Todos los mandos a distancia que usan señales infrarrojas emplean el mismo método de codificación de la señal. La señal es binaria y varía su longitud, tanto en tiempo como en longitud de bit.

El estándar empleado por los mandos a distancia consiste en enviar pulsos de luz modulados en amplitud con una portadora cuyo valor de frecuencia pertenece al rango 20-40 KHz. Este valor dependerá del fabricante del mando a distancia. De esta forma, el receptor puede cortar las frecuencias relacionadas con otras fuentes, como son las lámparas de luz visible.

Por otro lado, existen tres métodos distintos para la codificación de señales:

- **Codificación del pulso de la señal en el que se varía la longitud de los pulsos**

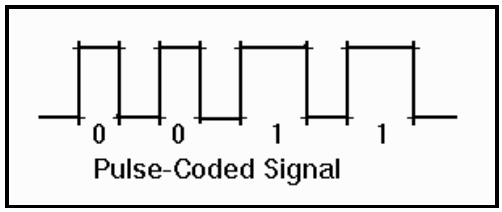


Figura 4.1. *Codificación por variación de la anchura de los pulsos*

Los antiguos mandos a distancia de Sony empleaban este tipo de codificación.

- **Codificación de los espacios de la señal que consiste en la variación de la longitud de los espacios**

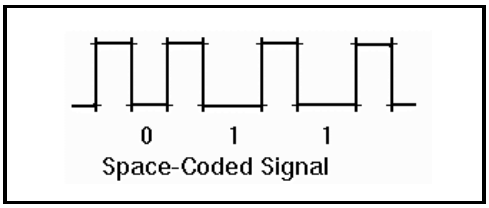


Figura 4.2. *Codificación por variación de la longitud de los espacios*

Es el empleado por un gran abanico de fabricante como, por ejemplo, Panasonic.

- **Código bifásico en el que se varía el orden de los espacios de los pulsos**

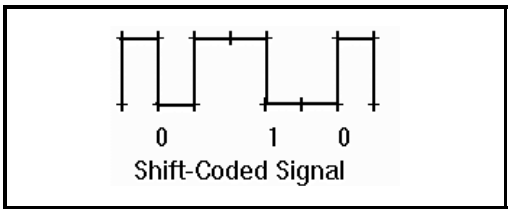


Figura 4.3. *Codificación por variación en el orden de los espacios de los pulsos*

Es el empleado por el fabricante Philips.

Además, es bastante habitual colocar una **cabecera** para iniciar la transmisión y consiste en un pulso de longitud superior al bit "1" y al "0" que se envía al comienzo de la ristra de bits.

También es bastante común, tras enviar el conjunto de bits correspondiente, emitir el llamado **bit de stop**, que está formado por una portadora modulada en amplitud.

4.3 PROTOCOLOS DE TRANSMISIÓN

Existen al menos dos protocolos internacionales que son los usados por los fabricantes para codificar los comandos de sus mandos a distancia. Estos protocolos son:

- ***RECS80***
- ***RC5***

4.3.1 Protocolo RECS80

Utilizado por la mayor parte de los mandos a distancia, emplea codificación por variación de la longitud del pulso. Así, cada bit es codificado por un nivel alto de duración T seguido de un nivel bajo de duración 2T cuando queremos representar el valor lógico "0" ó 3T cuando se trata del "1".

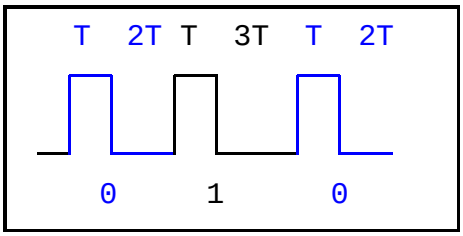


Figura 4.4. *Protocolo RECS80*

Como se observa en la Figura 4.4., el tiempo empleado para transmitir un "1" es superior al del "0".

4.3.2 Protocolo RC5

En este protocolo, la duración de los bits es la misma para todos ellos. Se diferencian en que en la mitad del intervalo se produce una transición de nivel. Así, el valor lógico "0" se codifica como una transición de alto a bajo, y el "1" como una transición de bajo a alto.

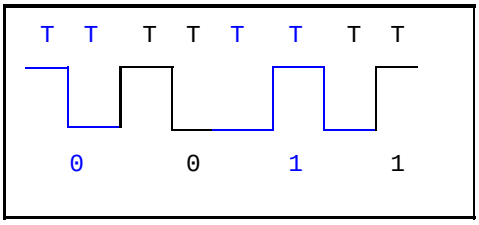


Figura 4.5. *Protocolo RC5*

Por otro lado, cuando se emite una ristra de bits pueden ser necesarias algunas transiciones al inicio del bit para establecer el correcto nivel de comienzo del mismo.

4.4 FORMATO DE TRAMA

4.4.1 Formato de Trama en el RECS80

La trama enviada por los mandos a distancia que cumplen este protocolo está formada por las siguientes partes:

- **Cabecera**
  Es un elemento que no presentan todos los fabricantes. Su función será la de permitir al receptor adaptarse al nivel de señal recibida. La mayor parte de los receptores de señales infrarrojas llevan integrados un circuito de control automático de ganancia (CAG) que permite recibir señales perfectamente desde distintas distancias. Pero, para ello, necesita un tiempo que es el que le brinda la cabecera.
- **Dirección**
  Identifica, dentro del fabricante, el modelo del equipo.
- **Comando**
  Especifica la función que se le envía al equipo.
- **Bit de stop**
  Su función será la de indicar al receptor que ha finalizado la emisión de la trama.

Este conjunto de bits se repite, después de transcurrir un intervalo de tiempo determinado, en lo que se tenga pulsada la tecla. Además, la longitud de esta trama puede variar desde 1 a 48 bits, y, algunos modelos añaden al conjunto una copia invertida del mismo.

4.4.2 Formato de trama en el RC5

La trama que cumple el protocolo RC5 está formada por 14 bits:

Start	Nature	Repetición	G4	G3	G2	G1	G0	A5	A4	A3	A2	A1	A0

El primer bit de la cadena se le denomina **bit start o bit de sincronización**. Su función es la misma que la de la cabecera en el protocolo RECS80: permitir al receptor sincronizarse y adecuarse al nivel de señal recibida. Además, este bit toma siempre el valor lógico "1".

El segundo bit se le denomina **bit nature** del código RC5, cuyo valor también es "1".

A continuación, viene el llamado **bit de repetición**. Es un bit que cambia su valor cada vez que se envía un comando nuevo. Su papel consiste en informar al receptor de que la

trama que va recibir pertenece a una señal distinta y, así, distinguir si se recibe un comando repetido o uno nuevo. Esto puede parecer evidente cuando se pulsa la tecla "stop" y luego "play". Pero, cuando se está, por ejemplo, introduciendo una hora en un vídeo y se le quiere indicar 33, al pulsar la tecla para la segunda cifra el bit de repetición cambia y así distingue que se pulsó dos veces la tecla 3. De todas formas, los receptores conocen perfectamente cuál es la periodicidad de una trama y, si no llega a tiempo, lo interpretará como que el comando ha finalizado y que la siguiente trama es un comando nuevo, aunque no cambie el bit de repetición.

Después, se envían cinco bits que contienen la **dirección del sistema**, es decir, indica cuál es el dispositivo a dirigir.

Para finalizar, se envía el **comando** deseado codificado en 6 bits más el bit nature.

4.5 ALGUNOS FABRICANTES DE MANDOS A DISTANCIA

4.5.1 Philips

Sus mandos son compatibles con el protocolo RC5. La frecuencia de la portadora es aproximadamente 37 KHz.

Sus tramas están formadas por 14 bits, cuya disposición corresponde a la de figura estudiada en el apartado 4.4.2. La longitud de sus bits es la misma para el valor "0" que para el valor "1", sólo se diferencian en la transición producida en la mitad del período. El aspecto que presentan se muestra en la figura siguiente:

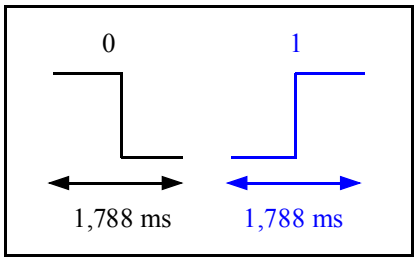


Figura 4.6. *Forma de los bits "0" y "1" para el fabricante Philips*

El espacio que existe entre la emisión de una trama y otra es de 25 ms.

4.5.2 Thomson

Los productos de este fabricante son compatibles con el protocolo RECS80. La frecuencia de la portadora empleada es aproximadamente 34 KHz y el tipo de codificación por variación en la longitud de los espacios.

Las tramas están formadas por 12 bits con la siguiente distribución:

T1	**T0**	**1**	**1**	**1**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**

donde podemos visualizar:

- 2 bits de repetición, T1 y T0
- 3 bits de dirección que siempre toman el valor "1"
- 7 bits de datos representados por A6:A0

El aspecto temporal de los bits "0" y "1" se presenta en la figura siguiente:

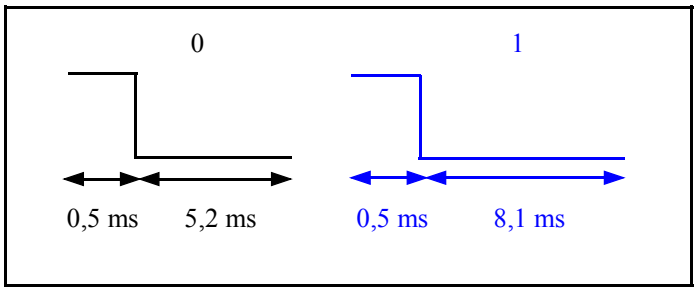


Figura 4.7. *Forma de los bits "0" y "1" para el fabricante Thomson*

Existen otros fabricantes como Saba o Telefunken que emplean el mismo formato de trama y la misma codificación.

4.5.3 Sony

Los mandos a distancia de Sony están basados en una codificación por variación de la anchura de los pulsos y cumplen el protocolo RECS80. El número de bits enviado por trama se reduce a 12 y la frecuencia de la portadora empleada es de unos 40 KHz.

La señal comienza con una cabecera en la que el nivel alto dura 4T y el nivel bajo T, donde T toma el valor de 550 μs. A continuación, se envía el comando que consiste en un conjunto de "0" y "1" cuya representación temporal es, como puede observarse en la Figura 4.8., de un pulso de valor 2T seguido por un espacio T para el "1" y de un pulso de valor T seguido por un espacio T para el "0"

El tiempo que transcurre entre la emisión de una trama y la siguiente es de 25 ms. Además, mientras se tenga pulsada una tecla del mando a distancia la trama será emitida una y otra vez.

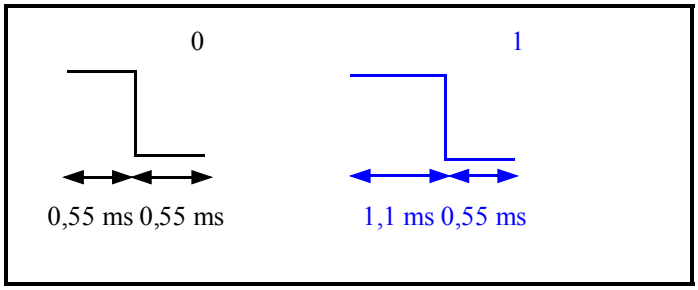


Figura 4.8. *Forma de los bits "0" y "1" para el fabricante Sony*

4.5.4 Panasonic

Este fabricante emplea la codificación por variación de la longitud de los espacios con una frecuencia de la portadora de 37 KHz. El número de bits emitidos es en total 48, donde los 32 primeros forman la dirección y los 16 siguientes el comando.

La señal emitida es compatible con el protocolo RECS80. Comienza con una cabecera formada por un pulso de valor 10T y por un espacio de valor 4T, donde T toma el valor 400 µs. A continuación de la ristra de bits enviará un bit de stop que consta de un pulso y de un espacio de valor T. El aspecto temporal de los bits se muestra en la figura siguiente:

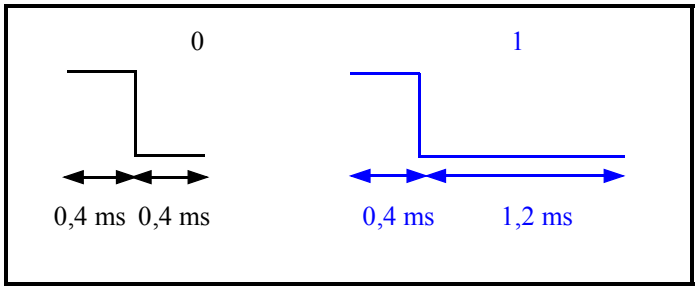


Figura 4.9. *Forma de los bits "0" y "1" para el fabricante Panasonic*

Como puede observarse ambos bits tienen un pulso de valor T y se diferencian en que el bit "0" tiene un espacio T y el "1" uno 3T.

El tiempo que transcurre entre la emisión de una trama y la siguiente es de 74 ms. Además, en lo que se tenga pulsada la tecla del mando a distancia se producirá la emisión de la trama.

CAPITULO 5
EL BUS SERIE I^2C

5.1 INTRODUCCIÓN

En este capítulo se describe el protocolo de comunicación del bus I^2C, que es el empleado en la comunicación entre el microcontrolador y la memoria EEPROM.

Los diseños electrónicos incluyen una serie de componentes que son comunes a todos ellos. Contienen:

- Un sistema de control, que normalmente es un microcontrolador.

- Circuitos de propósito general como son LCDs, memoria RAM, EEPROM, convertidores de dato,...

- Circuitos de propósito específico como pueden ser circuitos procesadores de señal para sistemas de radio y video,...

Para explotar estas similitudes en beneficios de los diseñadores de sistemas y equipos, así como para maximizar la eficiencia del hardware y la simplicidad de los circuitos, Philips desarrolló un bus bidireccional de dos hilos llamado ***Bus Inter. Integrated Circuit o I^2C***.

5.2 EL CONCEPTO DEL BUS I^2C

El **I^2C** es un bus de comunicaciones serie que emplea simplemente dos hilos. Los dos hilos, ***serial data (SDA)*** y ***serial clock (SCL)***, transmiten la información entre los dispositivos que se encuentran conectados al bus. Además, ambas líneas son bidireccionales pero en cada momento la dirección del flujo de datos debe ser única.

Cada dispositivo se le reconoce en la implementación de este protocolo con una única dirección codificada en 8 bits y cada uno de ellos, dependiendo de la función que realicen, pueden operar como emisores o como receptores, pero nunca realizar ambas funciones simultáneamente. Obviamente, existe alguno de ellos, como por ejemplo un display, que sólo pueden funcionar como receptor.

Por otro lado, los dispositivos que están conectados en el bus se denominan ***maestro*** o ***esclavo***:

- El **maestro** es el dispositivo que comienza la transmisión de datos, genera la señal de reloj y las señales de control y termina la transmisión.

- El **esclavo** es el dispositivo controlado por el maestro.

Realmente, sólo cuando funcionan como esclavo deben ser identificados por una dirección que se la denomina ***dirección del esclavo***.

Cuando el maestro quiere iniciar la comunicación con un dispositivo esclavo transmite la dirección del esclavo y éste responde con una *señal de acknowledge* que veremos más adelante. Solamente el maestro y el esclavo del que se ha transmitido la dirección se encontrarán activos en el bus y el esclavo permanecerá en este estado hasta que el maestro dé la orden de desactivación. Además, esta comunicación sólo será posible cuando el bus esté libre.

El bus I^2C es un **bus multimaestro**, es decir, en el bus I^2C se pueden conectar distintos dispositivos maestros capaces de controlar el bus. Sin embargo, en un instante cualquiera solamente uno de ellos puede funcionar como maestro y al resto se les considerará como esclavos. En el caso particular de este proyecto vamos a tener un único maestro, que será el microcontrolador, y un solo esclavo, la memoria EEPROM.

5.3 CARACTERÍSTICAS GENERALES

Tanto la línea SDA como la SCL son líneas bidireccionales y están conectadas a tensión mediante una resistencia de pull-up (véase Figura 5.1). Esto significa que cuando el bus está libre ambas líneas se encuentran a nivel lógico alto.

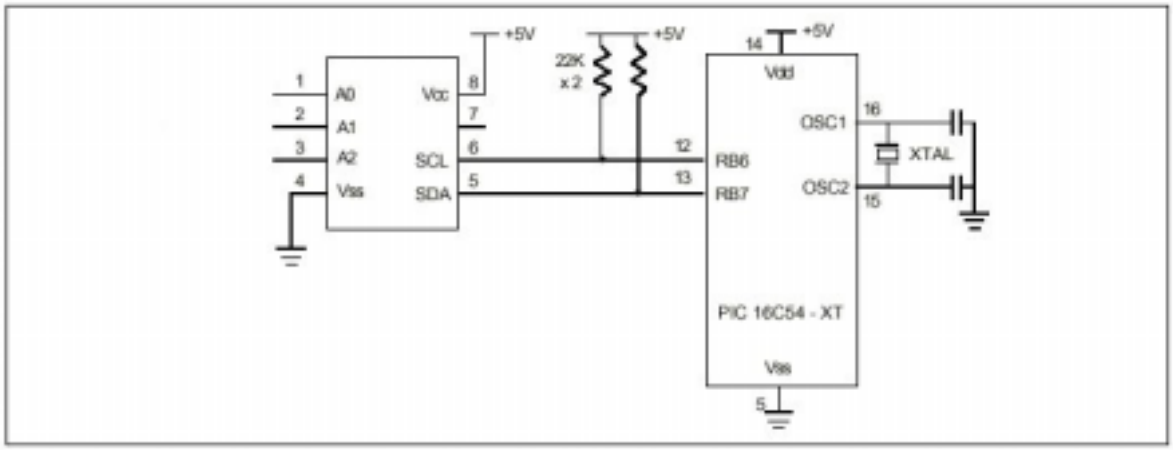


Figura 5.1. *Esquema de las conexiones del I^2C*

El hecho de que exista la posibilidad de tener más de un maestro conectado al bus significa que en un momento dado varios de ellos intenten iniciar una transmisión al mismo tiempo. Para evitar este conflicto es necesario implementar una puerta AND cableada, lo cual obliga a que, además de emplear las resistencias de pull-up anteriormente mencionadas, los dispositivos conectados al bus tengan como etapa de salida una configuración en drenador abierto o colector abierto. Así, cuando un maestro quiera iniciar una comunicación comprobará el estado de las líneas SCL. Si el bus está ocupado SCL está al nivel lógico bajo, lo que le indicará que debe esperar para establecer la comunicación.

La velocidad de transmisión de los datos en el bus I^2C puede ser de 100 kbits/s en el modo estándar o de hasta 400 kbits/s en el modo rápido. Además, el número de dispositivos conectados al bus está determinado por la capacidad del bus, cuyo valor máximo es de 400pF.

5.4 TRANSMISIÓN DE DATOS

La transmisión de datos en un bus serie I^2C, se realiza de la siguiente manera:

- La comunicación se inicializa con el envío por parte del maestro de la **condición de START**; inmediatamente después, se considera que el bus está ocupado.

- A continuación, el maestro envía la dirección del esclavo.

- Posteriormente, el emisor envía los datos en tamaño byte.

- Por último, el maestro envía la **condición de STOP**. Unos instantes después de la emisión de esta condición se considerará que el bus está libre.

El tamaño de la información que se envía a través del bus es de 8 bits y siempre emitiendo como primer bit el más significativo. Además, después de cada byte enviado es necesario que el receptor de la información emita la **condición de ACK (*Acknowledge*)** para indicar que ha recibido el byte correctamente. Si un receptor no puede recibir otro byte hasta que se dé otro evento, por ejemplo, la atención a una interrupción interna, puede mantener la línea de reloj SCL a un nivel lógico bajo para obligar al transmisor a que entre en un estado de espera. Así, se reiniciará la transferencia de datos cuando el receptor pase SCL a nivel lógico "1" (Véase Figura 5.2.)

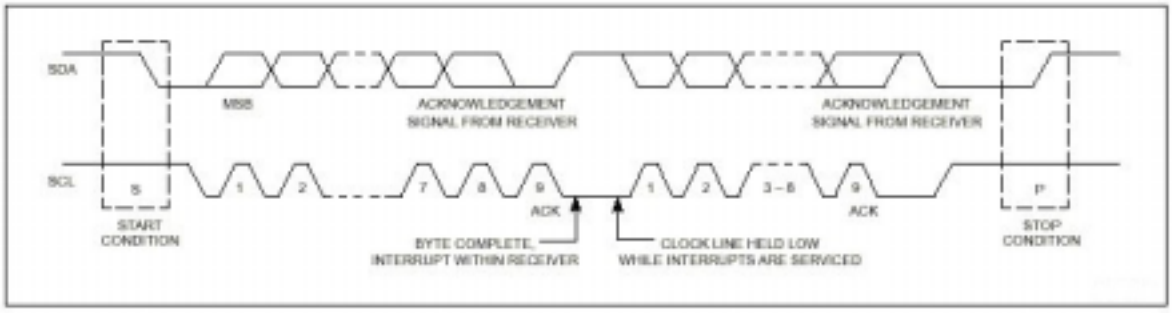


Figura 5.2. *Transferencia de datos en el bus I^2C*

5.4.1 Condiciones de START y de STOP

La **condición de START** viene determinada por una transición del nivel alto al nivel bajo en la línea SDA mientras se mantiene en nivel alto la línea SCL. Todas las operaciones vienen precedidas por esta señal.

Por el contrario, en la **condición de STOP** se produce una transición del nivel bajo al nivel alto en la línea SDA mientras SCL se mantiene en nivel alto. Todas las operaciones deben finalizar con esta condición.

Las condiciones de START y STOP son señales de control que sólo pueden ser generadas por el maestro.

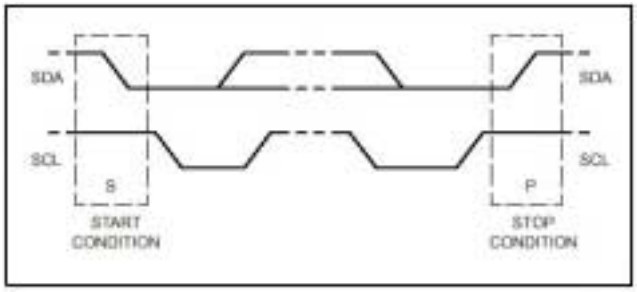


Figura 5.3. *Condiciones de start y stop*

5.4.2 Dato válido

Durante la transferencia de los datos, la línea SDA debe permanecer estable siempre que SCL esté situada en el nivel lógico alto. El dato sólo podrá cambiar en los períodos bajos de la señal de reloj y se dará un único pulso de reloj por cada uno de los bits del dato. Cualquier cambio de la señal SDA que se dé mientras SCL esté en baja se interpretará como una condición de START o STOP.

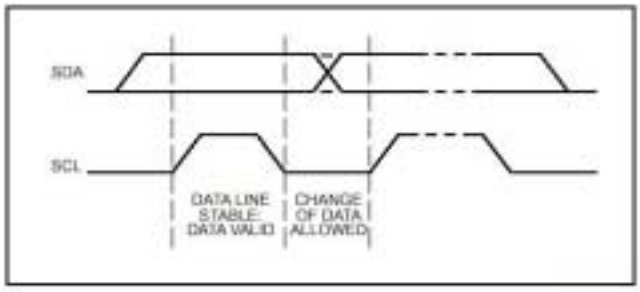


Figura 5.4. *Condición de estado válido*

5.4.3 Acknowledge

Cada vez que el receptor reciba un byte debe generar una **señal de acknowledge *(ACK)***. Para ello, el dispositivo maestro será el encargado de crear el pulso de la señal de reloj asociado a la señal ACK. Durante ese período el transmisor pone a nivel lógico alto la línea SDA y será el receptor el que la pase a nivel bajo, permaneciendo el valor de SDA estable en baja durante la parte alta del período de reloj del ACK. Además, debe cumplirse los tiempos de setup y hold establecidos en las hojas de especificaciones de las memorias. Cuando el dispositivo transmisor reciba el ACK continuará con la emisión del siguiente byte.

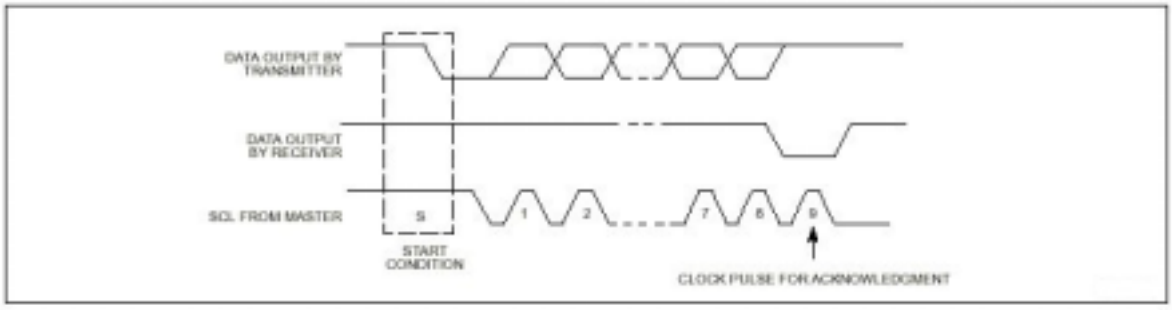


Figura 5.5. *Condición de Acknowledge*

Si un dispositivo esclavo-receptor no puede enviar la señal de ACK después de recibir el byte "dirección esclavo", la línea SDA permanecerá a nivel alto. Entonces, el maestro generará la condición de STOP para finalizar con la transferencia de datos.

Por otro lado, si el dispositivo esclavo-receptor envía la señal de ACK pero momentos después no puede recibir ningún otro byte de información, el dispositivo maestro debe abortar la transferencia. Así, el esclavo indiciará al maestro este hecho generando la **señal no ACK** en el siguiente byte de información, es decir, mantiene en alta SDA y el maestro envía la señal de STOP.

Si un maestro funciona como receptor, señalará el final de un dato al dispositivo esclavo transmisor no generando la señal de acknowledge durante el último byte enviado por el esclavo. En este caso, el esclavo pasará la línea de datos a nivel alto para permitir al maestro generar la condición de STOP.

5.4.4 La dirección del esclavo

Después de la condición de START, el primer byte que emite el maestro es la **dirección del esclavo**. Está compuesta por 8 bits, de los cuales los 7 más significativos indican la dirección propia del dispositivo y el menos significativo si se va a escribir o a leer del mismo. Un 0 en este bit indica que el maestro escribirá la información en el esclavo; un 1 indica que el maestro va a leer del esclavo.

En el caso de una memoria EEPROM, como puede verse en la Figura 5.6., los 4 bits más significativos de la dirección toman el valor fijo 1010; los tres bits siguientes (A2,A1,A0) definen el dispositivo o la parte del dispositivo al que accede el maestro. El bit menos significativo indica si se lee (valor 1) o se escribe (valor 0) en el dispositivo esclavo.

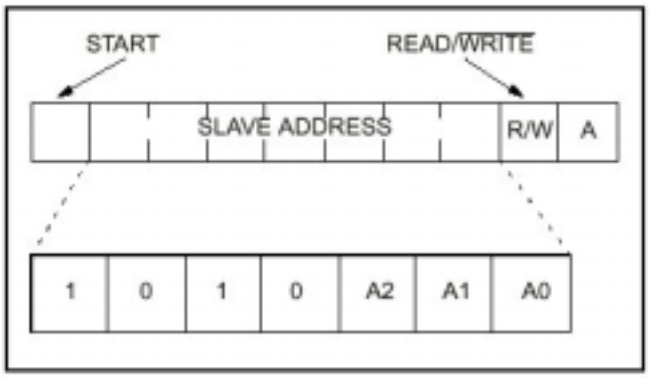


Figura 5.6. *Dirección del esclavo*

Como ya se ha indicado anteriormente, una vez enviada la dirección del esclavo éste contesta con un ACK. Entonces, el transmisor comienza a enviar los bytes de información, y cada uno de ellos vendrá sucedido por un ACK generado por el receptor.

5.5 ESCRITURA EN UNA MEMORIA EEPROM

Existen dos métodos para escribir los datos en una memoria:

- Modo de escritura byte
- Modo de escritura página

5.5.1 Modo de escritura byte

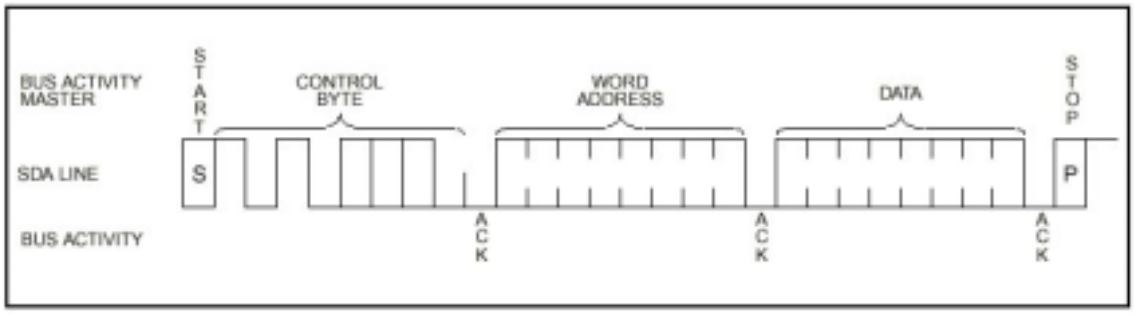


Figura 5.7. *Modo de escritura byte*

En este modo de escritura, el dispositivo maestro envía al esclavo la condición de START y la dirección del esclavo, donde el bit menos significativo toma el valor 0. Una vez que el esclavo genera el acknowledge, el maestro envía el byte que indica la posición de memoria en la que quiero escribir. Tras recibir de nuevo un ACK del esclavo, el maestro envía el byte de datos que tiene que escribirse en la posición de la memoria indicada. Entonces, el esclavo emite un ACK una vez más y el dispositivo maestro genera la condición de STOP. Es en este momento, la memoria entra en un ciclo interno en el que se encarga de grabar el dato recibido en la posición de memoria señalada. Durante este período, no responde a ninguna petición realizada por parte del maestro.

5.5.2 Modo de escritura página

Este tipo de operación permite enviar a la memoria, usando un solo ciclo de escritura, un número de bytes determinados que dependerá del modelo de la memoria.

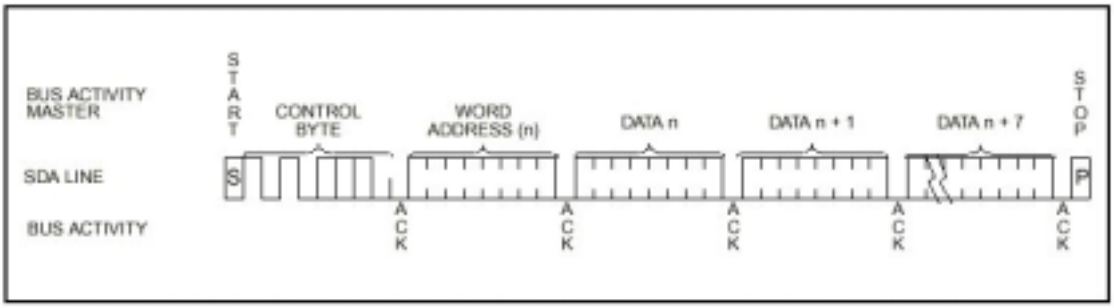


Figura 5.8. *Modo de escritura página*

El procedimiento empleado para comenzar la emisión es el mismo que en el modo de escritura anterior; el dispositivo maestro emite la condición de START y la dirección del esclavo, así como que la operación es de escritura (bit menos significativo a 0). A esta información el esclavo responde en el noveno período de reloj con un Acknowledge; entonces, el maestro continúa con la emisión de la dirección de memoria, que será almacenada en los 8 bits más bajos del puntero de dirección que indica la posición de la memoria en la que se va a escribir. A continuación, después de enviar el ACK correspondiente, la memoria comienza a recibir los datos en conjunto de 8 bits y los irá almacenando de forma consecutiva en un buffer. Tras recibir cada uno de los bytes deberá, naturalmente, responder con un ACK. Al finalizar la emisión de los datos se generará la condición de STOP y comenzará el ciclo interno de escritura de la memoria.

Si el dispositivo maestro emite un número de bytes mayor que el que soporta el buffer de la memoria, el contador de direcciones da una vuelta completa y se sobrescriben los primeros bytes recibidos.

5.5.3 Muestreo de la condición de Acknowledge

Una vez enviada la condición de STOP por parte del dispositivo maestro en un ciclo de escritura, la memoria comienza el ciclo interno de escritura. Para maximizar y optimizar los accesos a la memoria es necesario conocer en qué momento finaliza su ciclo de escritura interno. Para ello, se aprovecha el hecho de que la memoria no genera ninguna condición de ACK durante ese período de actividad interna. El proceso de muestreo del Acknowledge se resume en el siguiente diagrama de flujo:

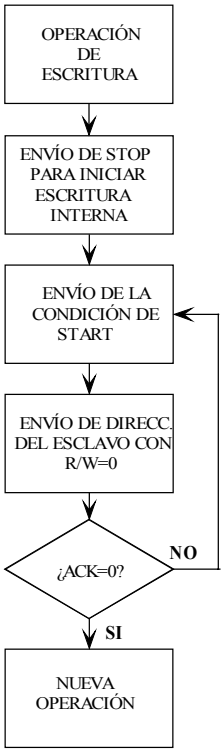


Figura 5.9. *Diagrama de flujo del muestreo del Acknowledge*

El proceso se inicia enviando la condición de START seguida de la dirección del esclavo con el bit R/W=0. Si el dispositivo está ocupado no responderá con un ACK; si la operación ya ha finalizado emitirá un ACK que indica al maestro que ya puede iniciar la siguiente operación de lectura o escritura.

5.6 LECTURA DE UNA MEMORIA EEPROM

Existen tres métodos de lectura distintos:

- Lectura de una dirección (*Random Read*)
- Lectura de la dirección actual (*Current Address Read*)
- Lectura secuencial (*Sequential Read*)

5.6.1 Lectura de una dirección

Con este método el dispositivo maestro puede acceder a cualquier posición de la memoria para realizar una operación de lectura. Para ello, el maestro comienza emitiendo la condición de START y la dirección del esclavo con el bit menos significativo a 0, es decir, habilitada la escritura. Después de recibir el ACK de la memoria vuelve a emitir la condición de START y la dirección del esclavo, pero esta vez con el bit a 1 para indicar que es una lectura. Entonces, la memoria responde enviando un acknowledge y enviando el dato de 8 bits; a esta información el maestro responde, no enviando un acknowledge, sino esperando un ciclo de la señal de reloj antes de enviar la señal de STOP.

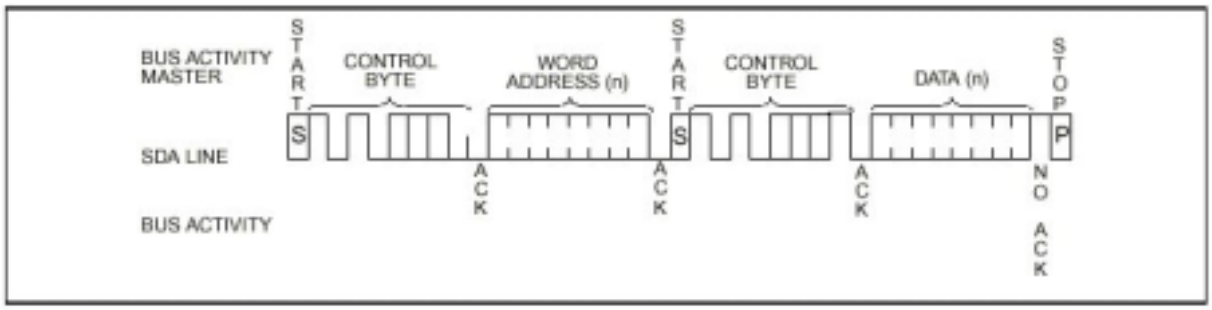


Figura 5.10. *Lectura de una dirección*

5.6.2 Lectura de la dirección actual

Con mucha frecuencia en una aplicación es necesario recuperar un dato, operar sobre él, recuperar el siguiente,... Esta secuencia de operaciones puede realizarse sin la necesidad de mantener grabada la última dirección que fue leída, ya que el puntero de direcciones de una memoria siempre apunta a la dirección del byte siguiente al último que se accedió.

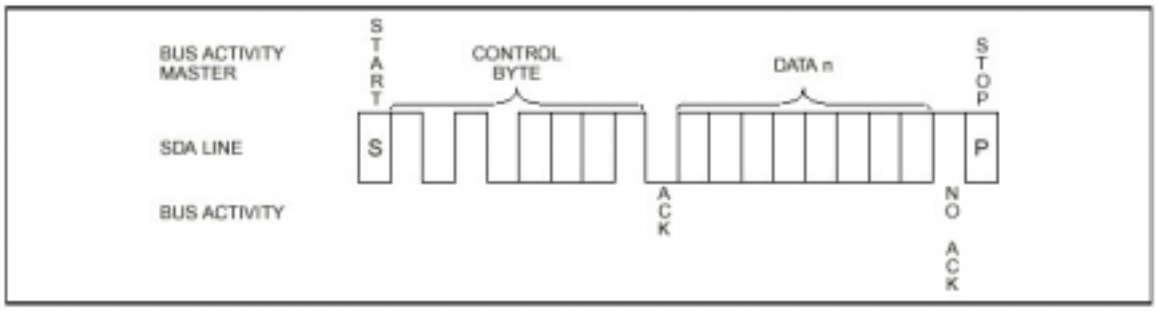


Figura 5.11. *Lectura de la dirección actual*

Es un método de lectura muy sencillo. Tras generar la condición de START el dispositivo maestro emite la dirección del esclavo con el bit menos significativo a 1 (lectura); entonces lee el dato (un byte) y espera un período de reloj (no ACK) antes de emitir la condición de STOP. Es decir, no emite la dirección de la posición de memoria que queremos leer.

5.6.3 Lectura secuencial

Este método de lectura es idéntico al de *lectura de una dirección* con la salvedad de que el dispositivo maestro, tras recibir el primer byte de información, transmite una ACK en vez de la condición de STOP, la memoria irá avanzando el contador de posición de la memoria y continuará enviando bytes de información hasta que reciba la condición de STOP. Con este método podemos llegar a leer todo el array de memoria. Es más, si el contador de direcciones llega a su valor máximo pasará a cero y comenzará a emitir los primeros valores de la memoria.

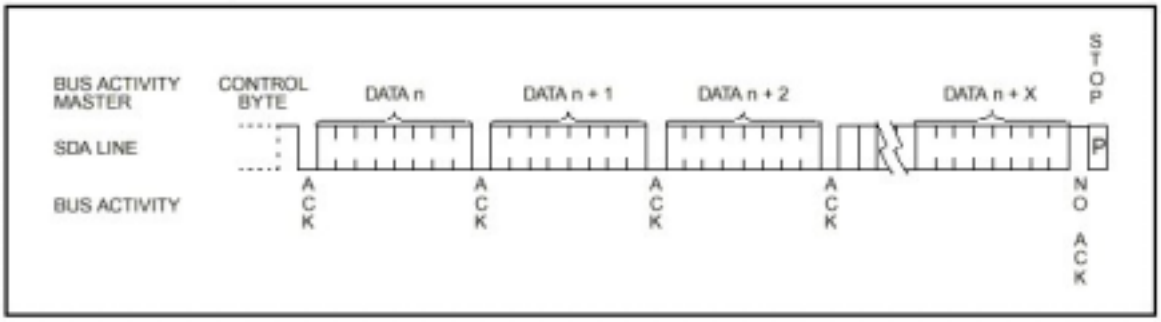


Figura 5.12. *Lectura secuencial*

CAPITULO 6
IMPLEMENTACIÓN HARDWARE

6.1 INTRODUCCIÓN

En este capítulo podemos ver detalladamente la implementación hardware desarrollado en este Proyecto. Tanto el diseño hardware como el layout para crear posteriormente la placa de circuito impreso del mando a distancia se han realizado con la herramienta de diseño Orcad 9.1.

6.2 ESQUEMA ELÉCTRICO

En la Figura 6.1 se puede observar el **esquema eléctrico del Mando a Distancia Reprogramable** presentado en este proyecto.

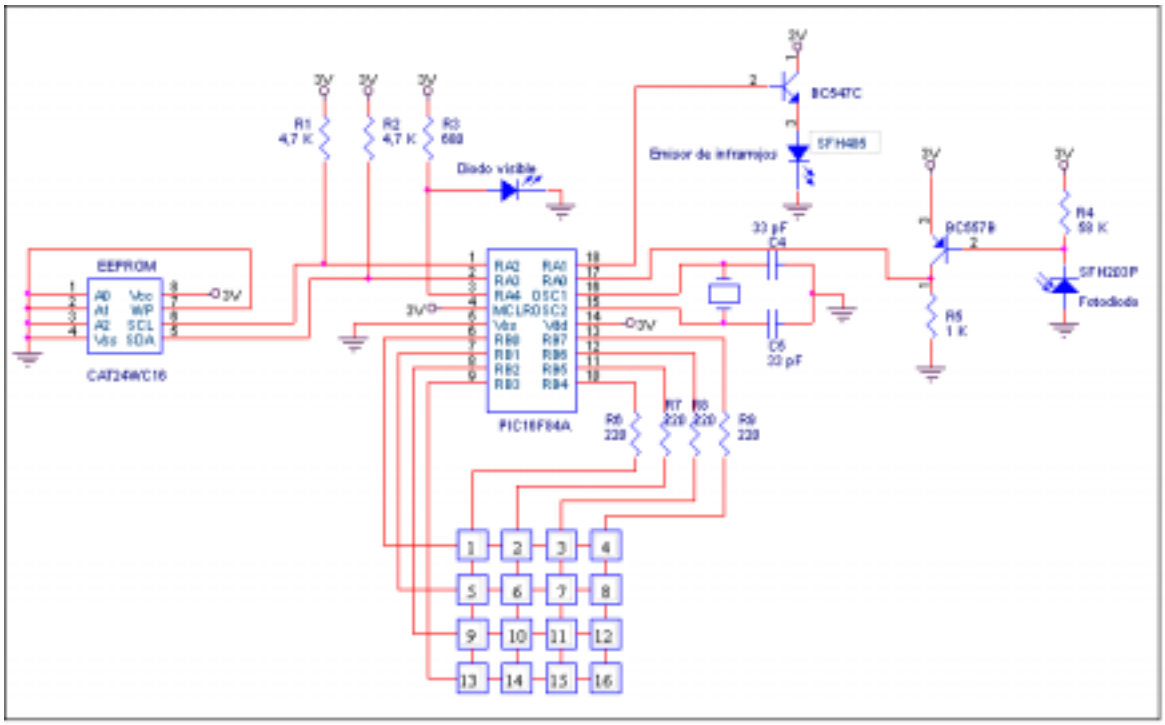


Figura 6.1 *Esquema eléctrico*

Como puede observarse consta de los siguientes elementos:

- Microcontrolador PIC16F84A de Microchip Technology Inc.
- Memoria Serie CMOS EEPROM de 2 Kbytes modelo CAT24WC16 del fabricante CATALYST.
- Teclado matricial de 16 teclas.
- Transistor NPN modelo BC547C de Philips.

- Transistor PNP modelo BC557B de Philips.
- Emisor de infrarrojos de AlGaAs SFH485 de longitud de onda en el pico de emisión de 880 nm del fabricante OSRAM.
- Fotodiodo PIN de Silicio modelo SFH203P de OSRAM.
- Diodo led rojo de luz visible.
- Resistencias de valores 4,7KΩ, 680Ω, 220Ω, 1KΩ y 58KΩ
- Condensadores de valor 33pF.

6.3 MICROCONTROLADOR PIC16F84A

Como ya se ha explicado anteriormente el PIC16F84A es un microcontrolador de 8 bits con una Memoria de Programa Flash de 1K x 14 bits, una Memoria de Datos RAM de 68 bytes, una Memoria de Datos EEPROM de 64 bytes.

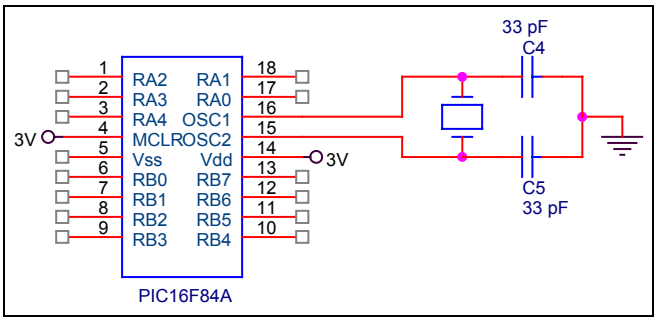


Figura 6.2. *Esquema del microcontrolador PIC16F84A*

Para su funcionamiento, se le ha colocado en los pines *OSC1* y *OSC2* un cristal oscilador de 4 MHz. Para estabilizar el valor de frecuencia del mismo se han situado en cada uno de sus extremos un condensador de 33 pF que a su vez están conectados a tierra.

La alimentación del PIC se realiza a través del pin 14 con dos pilas que suministran 3V de tensión. Los pines 4, correspondiente al reset *MCLR#*, y 5, correspondiente a *GND*, se encuentran conectados a 3V y a tierra respectivamente.

Debido a sus características internas y a las necesidades impuestas por el programa desarrollado, ha sido necesario incluir en el diseño una memoria EEPROM externa que pasa a describirse a continuación.

6.4 MEMORIA EEPROM

Es una memoria serie EEPROM CMOS de 2 Kbytes compatible con el protocolo de transmisión I^2C. Su encapsulado es de 8 pines, que pasamos a describir brevemente:

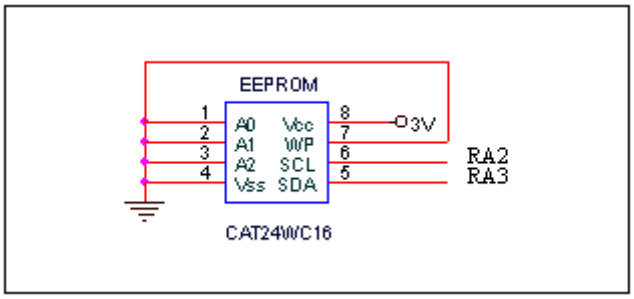


Figura 6.3. *Memoria EEPROM*

- **Vcc.-** Por este pin se introduce la alimentación. En este diseño se le alimenta a 3V.
- **Vss.-** Tierra.
- **A0, A1, A2.-** Estos pines se emplean para indicar la dirección correspondiente a la memoria cuando se colocan varias en cascada. En el caso del modelo CAT24WC16 sólo se puede emplear una, por lo que esos pines se conectan a Vss.
- **WP (Write Protect).-** Cuando se conecta este pin a tensión (Vcc) la memoria queda protegida contra la escritura, sólo se permite la acción de lectura.
- **SCL (Serial Clock).-** Es un pin de entrada y por él se introduce la señal de reloj.
- **SDA (Serial Data/Address).-** Es un pin bidireccional y a través de él se escriben o se leen los datos de la memoria. Cuando está configurado como salida este pin es una salida en drenador abierto.

Como se puede observar en las Figuras 6.1. y 6.3. los pines *SCL* y *SDA* de la memoria EEPROM se han conectado a las líneas del Puerto A del PIC *RA2* y *RA3* respectivamente. A través de estos dos pines se procederá a la escritura o lectura de los datos de la memoria.

Además, se han colocado dos resistencias de valor 4.7KΩ que harán las veces de resistencias de pull-up, ya que, al contrario que el Puerto B, las líneas del Puerto A no poseen resistencias de pull-up interna.

Por último es necesario comentar que la elección de una memoria en serie no ha sido aleatoria, si no que se ha debido a dos motivos principales. Primero, porque su número de pines es reducido, y segundo, porque permite implementar el protocolo I^2C.

6.5 TECLADO MATRICIAL

El teclado matricial consta de 16 teclas. Desde el punto de vista eléctrico, cada una de ellas es un mecanismo idéntico a un pulsador. Para disminuir las líneas necesarias para detectar la tecla pulsada, éstas se agrupan de forma matricial en filas y columnas. Con esta configuración sólo son necesarias 8 líneas del PIC para su gestión; si cada tecla actuase como un pulsador individual se precisarían 16 líneas de E/S.

Para controlar el funcionamiento del teclado, las cuatro líneas de menor peso del Puerto B (*RB0:RB3*) se configuran como salidas y se conectan a las filas del teclado. Del mismo modo, las cuatro líneas de mayor peso (*RB4:RB7*) han sido configuradas como entradas y recibe los niveles lógicos que tienen las columnas del teclado.

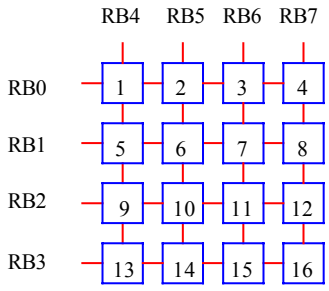


Figura 6.4. *Teclado matricial*

Por otro lado, se han colocado unas resistencias de protección de 2.2KΩ en las líneas *RB4:RB7* del PIC. Además, se han activado por software las resistencias de pull-up internas que posee cada línea del Puerto B.

La subrutina del programa que gestiona el teclado saca, secuencialmente, un nivel bajo por una de las 4 líneas de salida que se aplica a las filas. Al mismo tiempo, va leyendo el nivel lógico introducido por las columnas por las líneas de entrada. Si ninguna de las teclas de la fila por la que se introduce el nivel bajo está pulsada, se leerá un nivel alto en las cuatro columnas y se pasará a examinar la siguiente fila.

Si se aplica en una fila un nivel bajo y al leer las columnas una de ellas se encuentra a nivel bajo, se deduce que la tecla asociada a dicha fila y dicha columna está pulsada.

6.6 DIODO LED VISIBLE

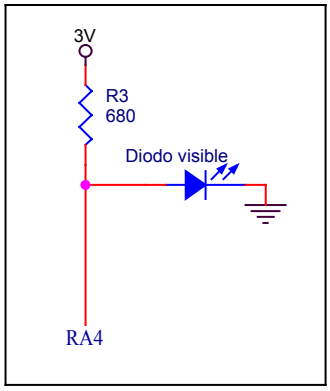


Figura 6.5. *Diodo visible*

Como puede observarse en las Figuras 5.1. y 5.5., el diodo led está conectado al pin *RA4* del microcontrolador PIC16F84A. Además, como este pin es una salida en drenador abierto, debemos colocar una resistencia de 680Ω.

La funcionalidad de este led es meramente informativa, es decir, se emplea para observar que el funcionamiento del Mando a Distancia es correcto.

Se emplea en dos casos distintos:

- **Etapa de muestreo.-** Cuando se inicializa el mando para grabar la señal que reciba, el led empieza a lucir y permanecerá encendido hasta que el microcontrolador haya inscrito los datos muestreados en la memoria, o hasta que pasen 10 segundos que es el tiempo que el mando espera para recibir alguna señal antes de salir del ciclo de muestreo.
- **Etapa de emisión.-** Durante la emisión de la señal grabada, el led permanecerá encendido mientras tengamos pulsada la tecla.

6.7 CIRCUITO EMISOR DE SEÑALES INFRARROJAS

El circuito empleado para la emisión de las tramas de bits previamente muestreadas y grabadas, como se puede observar en la Figura 6.6., está formado por el transistor NPN BC547C y por el emisor de infrarrojos de AlGaAs SFH485 de la marca OSRAM.

El emisor de infrarrojos tiene por longitud de onda en el pico de emisión 880 nm y un ángulo de emisión de ± 20 grados.

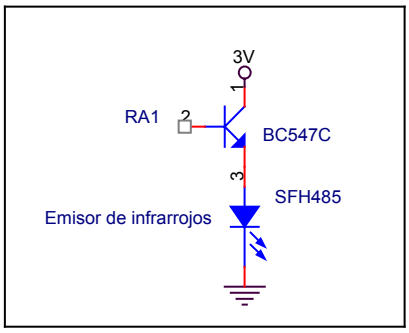


Figura 6.6. *Circuito emisor*

Para aumentar la potencia de emisión hemos colocado al SFH485 en el emisor de un transistor NPN, es decir, el transistor tiene una configuración en emisor común. De esta forma, aumentamos la corriente de las señales que salen a través del pin *RA1* y que llegan

al emisor de infrarrojos; así, la distancia a la cual el funcionamiento del mando a distancia es correcto aumenta.

6.8 CIRCUITO RECEPTOR DE SEÑALES INFRARROJAS

El circuito receptor de señales infrarrojas está formado por el fotodiodo de Silicio SFH203P de la marca OSRAM, un transistor PNP y dos resistencias de 1KΩ y 58KΩ.

Su misión consiste en transformar las señales infrarrojas emitidas por el mando a distancia cuya señal queremos grabar en pulsos cuadrados de tensión que se introducen al microcontrolador por el pin *RA0*.

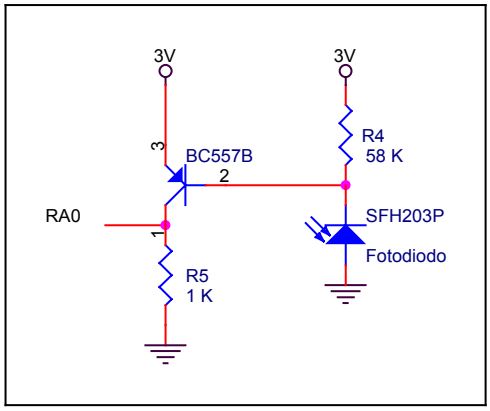


Figura 6.7. *Circuito receptor*

El fotodiodo receptor tiene una longitud de onda de máxima sensibilidad de 850 nm y un ángulo de recepción de ± 75 grados.

El transistor PNP está colocado en configuración de colector común. Su base está unida al cátodo del fotodiodo y el colector al pin *RA0* del microcontrolador. La función de este transistor será amplificar la señal recibida por el fotodiodo antes de introducirla por el pin *RA0* para así conseguir un muestreo adecuado de la misma.

CAPITULO 7
SOFTWARE DEL MANDO A DISTANCIA

7.1 INTRODUCCIÓN

En este capítulo se presenta el funcionamiento y el programa diseñado para este mando a distancia. Este último, ha sido escrito en lenguaje ensamblador para microcontroladores PIC.

7.2 FUNCIONAMIENTO DEL MANDO A DISTANCIA

El mando a distancia tiene dos modos de trabajo:

- **Modo grabación**.- En este modo de funcionamiento se realiza el muestreo y grabación de la señal emitida por otro mando a distancia. En total se pueden grabar 16 señales distintas, una por cada tecla del teclado matricial. Los datos muestreados se van a guardar en una zona de la memoria EEPROM asociada a una tecla determinada. El programa será capaz de distinguir si el código recibido cumple el protocolo RECS80 o el RC5.

- **Modo emisión**.- En este modo se procede a la emisión de una señal grabada anteriormente.

7.2.1 Modo grabación

Cuando queremos memorizar la señal que emite la tecla de un determinado mando a distancia debemos inicializar nuestro mando programable en modo grabación. Para ello, hay que pulsar las teclas * y # que se encuentran situadas en la cuarta fila, primera y tercera columna respectivamente del teclado matricial. A continuación, procedemos a pulsar la tecla a la cual queremos asociar la señal que vamos a recibir. Entonces, se ilumina el led visible y esto nos indica que el mando ya está preparado para recibir la trama. El siguiente paso será enfrentarlo al mando a distancia original y pulsar en éste la tecla cuya trama queremos grabar. En el momento en que el programa haya muestreado y grabado todos los datos necesarios de la trama el led visible se apagará y el microcontrolador volverá a entrar en un estado de bajo consumo o de sleep.

Por motivos de seguridad y para evitar que el microcontrolador quede bloqueado, una vez que se inicializa el mando en modo grabación (y se enciende el led), si en diez segundos no recibe ninguna señal, el microcontrolador sale automáticamente de este modo, apaga el led visible y vuelve a entrar en el estado de reposo.

7.2.2 Modo emisión

En este modo de funcionamiento se emite una trama que fue muestreada y grabada anteriormente. Para ello, debemos enfrentar el mando al dispositivo que queremos manejar y pulsar la tecla a la que asociamos la señal durante la grabación.

La emisión de la señal se producirá en lo que tengamos la tecla pulsada y, además, para visualizar el funcionamiento del mando, el led permanecerá encendido durante toda la emisión.

7.3 EL PROGRAMA DISEÑADO

Consta de un programa principal y una subrutina de atención a la interrupción situados en las direcciones 0h y 4h respectivamente.

En el programa principal se realizan dos funciones:

- Inicialización del microcontrolador, en la que se dan los valores necesarios a los registros SFR.

- Se sumerge al microcontrolador en un estado de bajo consumo o sleep.

Por otro lado, cuando el microcontrolador detecte que se ha producido la fuente de interrupción con la que ha sido programado salta a la dirección 4h del programa, en la cual se encuentra situada la parte de código correspondiente al servicio de atención a la interrupción. En nuestro caso, se programa al microcontrolador para que detecte cualquier cambio en los valores de las líneas *RB4:RB7*.

7.3.1 Programa principal

Una vez conectado el microcontrolador a la alimentación se procede desde el programa principal a la inicialización de ciertos registros. Las acciones que tienen lugar son:

- Inicialización a cero del Puerto A.
- Configuración de las líneas *RA0*, *RA2* y *RA3* como entradas y *RA1* y *RA4* como salidas. Para ello se introduce en el registro *TRISA* el valor 0Dh.
- Inicialización del registro *TRISB* con el valor F0h. De esta forma, configuramos *RB0:RB3* como salidas y *RB4:RB7* como entradas.
- Damos el valor 0 a los bits 7, 3 y 0-2 del registro *OPTION_REG*. Así, habilitamos las resistencias del pull-up del puerto B, determinamos un prescalado de 8 y lo aplicamos al contador TMR0 respectivamente.

- Establecemos el bit *RBIE* del registro *INTCON* a 1 para habilitar la interrupciones por cambio en el estado de algunas de las líneas *RB4:RB7*. Además, limpio el flag asociado a esta interrupción (bit *RBIF*).
- Introducimos el valor F0h al puerto B, para que en el momento en que se pulse una tecla se detecte el valor 0 en una de las líneas *RB4:RB7* y provoque la interrupción programada.
- A continuación, y ya como último paso del programa principal, se introduce al microcontrolador en el estado de bajo consumo

7.3.2 Subrutina de atención a la interrupción

Está situada en la dirección 4h de la memoria de programa, que es la reservada en el microcontrolador PIC16F84A para ubicar el código de atención a la interrupción.

En esta subrutina se realizan las siguientes operaciones:

- **Escaneado de las entradas del teclado**.- Se comprueba si se ha pulsado alguna tecla y, en caso afirmativo, se identifica cuál.- Se comprueba si se ha pulsado alguna tecla y, en caso afirmativo, se identifica cuál de todas ellas ha provocado la interrupción.

- **Muestreo o emisión de una trama determinada**.- Una vez detectada la tecla o conjunto de teclas pulsadas se procede al muestreo y grabación de datos en la memoria o a la lectura y emisión de la trama según corresponda.

Veamos cada una de las subrutinas más detenidamente.

A. Subrutina de escaneo del teclado

En esta parte de código, se comprueba la tecla o teclas que se han pulsado y el tipo de operación que se desea realizar, es decir, si vamos a seleccionar emisión o grabación de una señal.

En primer lugar, se verifica si realmente se ha pulsado una tecla. En caso de no detectar ninguna, se realiza una segunda comprobación que, de dar negativo, hará que el programa salga de esta subrutina, prepare el puerto B para un nuevo escaneo y meta al microcontrolador en estado de bajo consumo.

En el caso de que se detecte una tecla, el primer paso es comprobar si la tecla pulsada es el ***** (tercera fila, primera columna). Para ello, introduzco un 0 por la línea *RB3* y compruebo si se propaga a *RB4*. Pueden darse dos opciones distintas:

a. En *RB4* no se dé el valor 0. Entonces, eso significa que se ha pulsado cualquier otra tecla y que se pretende emitir la señal asociada a la misma. El siguiente paso a realizar es escanear todo el teclado y detectar la tecla pulsada. El método a emplear es el mismo que para detectar la tecla *, es decir, vamos introduciendo secuencialmente un 0 por cada fila (comenzando por la fila 1) y se comprueba en qué columna se propaga.

Además, al mismo tiempo que detectamos qué tecla fue pulsada vamos incrementando la variable ***eeddr0*** que será el puntero a la zona de memoria donde se encuentran los datos necesarios para emitir la señal.

b. En *RB4* se dé el valor 0. En este caso, nos encontramos con otras dos posibilidades, que quiera grabar o emitir la señal asociada a la tecla *. Para averiguarlo, escaneamos la tecla # (cuarta fila, tercera columna) durante diez segundos. Si detectamos que es pulsada, es que quiero entrar en modo grabación. En caso contrario, querré emitir.

En el caso de que queramos grabar una señal debemos escanear el teclado aplicando el método que anteriormente habíamos comentado para saber a qué tecla queremos asociar la trama y en qué zona de memoria debemos guardar los datos muestreados.

En la Figura 7.1. se puede observar el diagrama de flujo de esta subrutina.

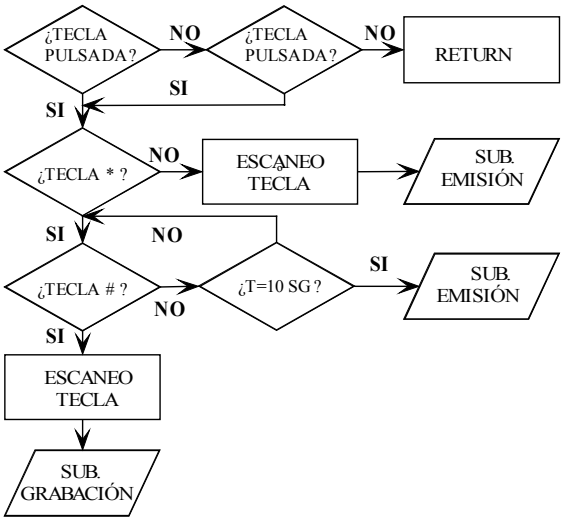


Figura 7.1. *Subrutina de escaneo del teclado*

B. Subrutina de muestreo de la señal

En esta subrutina se procede al muestreo de la señal o trama que llega a través de *RA0* y al almacenamiento de los datos obtenidos en la zona de memoria que comienza en el valor determinados por la variable *eeddr0*. Vamos a establecer una serie de conceptos para poder comprender explicaciones sucesivas.

Tanto la cabecera como los bits están formados por dos partes, una parte de señal que es una **portadora modulada** y otra parte en la que no se emite nada (valor lógico 0) y que la llamaremos **nivel bajo de la señal**.

Por otro lado, dentro de portadora modulada, definimos **nivel alto del pulso** cuando la portadora está en valor lógico 1 y **nivel bajo del pulso** cuando se emite un valor lógico 0.

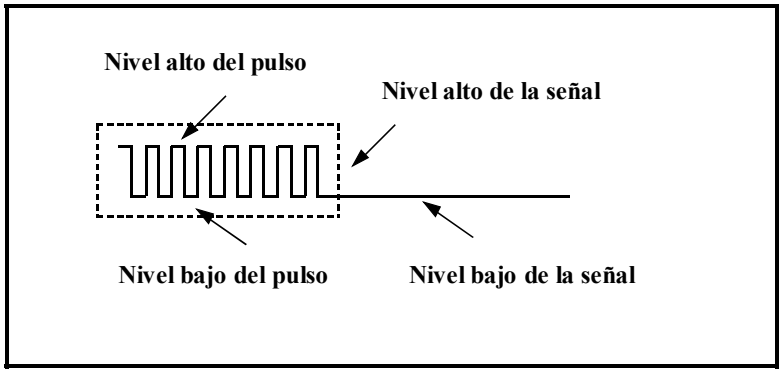


Figura 7.2. *Esquema de un pulso de una trama*

En el desarrollo del programa se emplean una serie de variables que especificamos a continuación:

Num_pulsos	Número de pulsos que forman el nivel alto de la cabecera
Cont_pulsos/1	Número de pulsos que forman el nivel alto de un bit
T_pulsoal	Duración del nivel alto de un pulso
T_pulsoba	Duración del nivel bajo de un pulso
T_cabba	Duración del nivel bajo de la cabecera
T_valle0	Duración del nivel bajo del bit codificado como "0"
T_valle1	Duración del nivel bajo del bit codificado como "1" (para RECS80)
Bit	Número de bits de los que está formado la trama
Vueltas1	Variable auxiliar empleada en la medición del tiempo que transcurre entre dos tramas
Trama	Variable de 6 bytes en la que se graba la codificación de los bits recibidos
T_espera	Tiempo que transcurre entre dos tramas

Como ya se comentó en el capítulo 3, dependiendo del tipo de formato empleado una trama puede tener cabecera o no. En este programa, si la señal no posee cabecera, el primer bit lo tomamos como tal a efectos de grabación de la información muestreada.

Una vez que hemos entrado en el modo de grabación y hemos elegido la tecla a la que queremos asociar la señal, se enciende el led visible para indicar este hecho y el mando a distancia queda en espera de recibir la señal, es decir, queda a la espera de detectar un 1 lógico. En este momento, debemos enfrentar nuestro mando al mando a original y pulsar la tecla del segundo cuya señal queremos grabar. Si a los diez segundos no ha recibido un nivel alto apagamos el led, salimos de esta subrutina, configuramos el puerto B con el valor F0h

Muestreo de la cabecera

Cuando comenzamos a recibir el nivel alto muestreamos hasta recibir un nivel bajo; entonces, guardaremos el valor que ha tomado el temporizador TMR0 en la variable ***alta1***. Inicializamos TMR0 a 0 y hacemos lo mismo con el nivel bajo, es decir, hacer un muestreo hasta que la señal pase a nivel alto y guardaremos el nuevo valor adquirido por TMR0 en la variable ***baja1***. Después, mostraremos los dos pulsos siguientes y haremos la media entre los valores obtenidos; así, obtenemos la frecuencia de la portadora, definida por dos tiempos, ***t_pulsoal*** y ***t_pulsoba***.

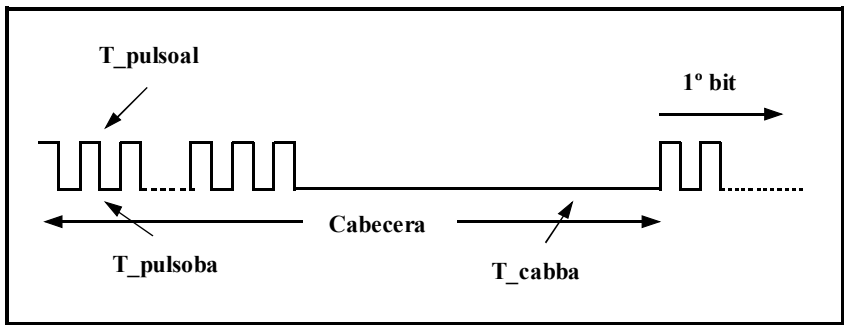


Figura 7.3. *Cabecera de una trama*

Por otro lado, es necesario ir contando los pulsos que forman el nivel alto de la cabecera. Para ello, cada vez que detectamos una transición entre baja y alta incrementamos la variable ***num_pulsos***.

El paso siguiente consiste en detectar el tiempo que la señal permanece en el valor lógico 0 hasta llegar el primer bit. Tras modificar el prescalado a un valor 32 para que RTCC no se desborde vamos comprobando en qué momento el valor del contador supera el de *t_pulsoba*; si la señal pasa de baja a alta y el valor no fue mayor, contamos un nuevo pulso y seguimos comprobando. Si es mayor, guardamos el valor de RTCC en ***t_cabba***. Por último, volvemos a asociar el valor 8 al prescalado de RTCC.

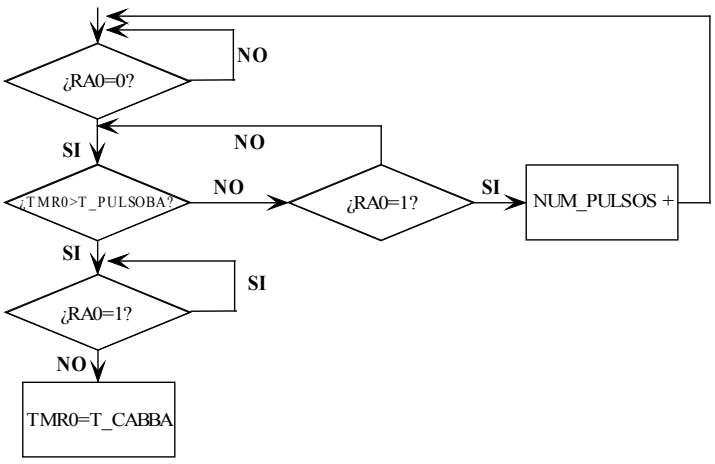


Figura 7.4. *Diagrama de flujo del muestreo del tiempo t_cabba*

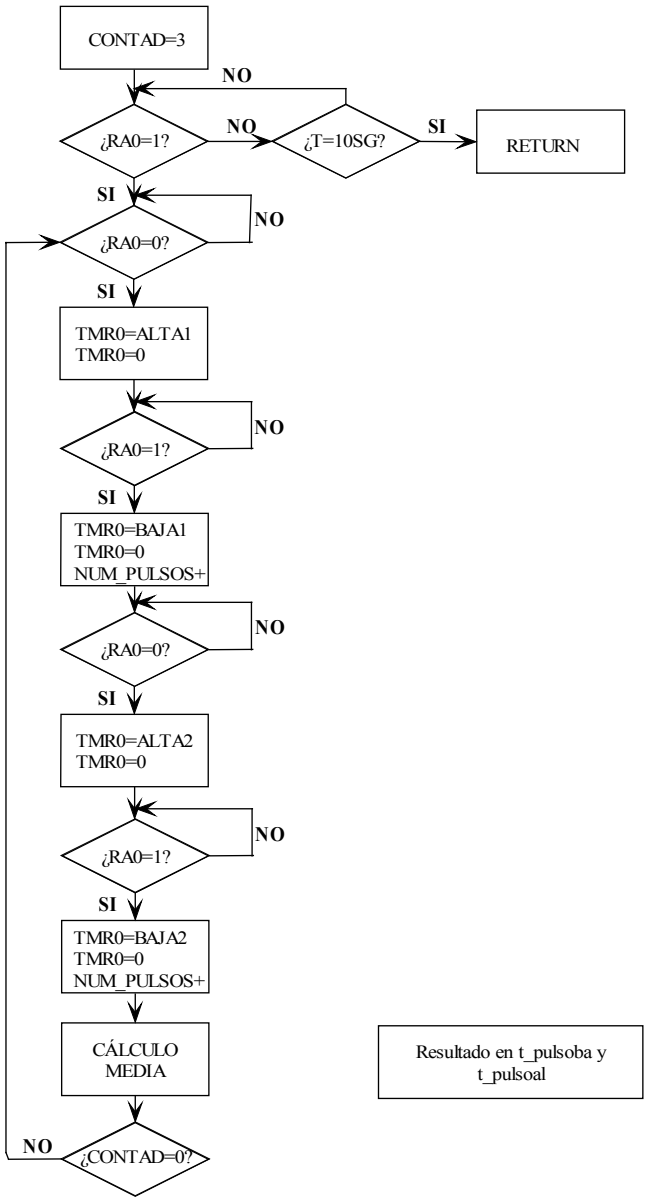


Figura 7.5. *Diagrama de flujo del muestreo de la parte alta de la cabecera*

Muestreo del primer bit

Una vez que hemos adquirido los valores de la cabecera, debemos obtener los del primer bit. A nivel del programa, el primero lo codificamos como si fuera un "0" y en comparación con los datos de éste obtendremos los del bit "1".

El proceso que se sigue es similar al empleado en la cabecera. La frecuencia de la portadora será la misma y el número de pulsos que forman la parte alta los averiguamos contando las transiciones de la señal de baja a alta y los guardamos en la variable ***cont_pulsos***.

Además, debemos averiguar cuál es el valor temporal del nivel bajo del primer bit). Para ello, como hicimos anteriormente, vamos comprobando si el valor de TMR0 es superior al de *t_pulsoba*. Si no es mayor incrementamos ***cont_pulsos***; si lo es lo guardamos en la variable ***t_valle0***, incrementamos la variable ***bit*** e introducimos un "0" en la variable ***trama***.

Una vez que hemos caracterizado el primer bit, nos preparamos para recibir los siguientes. Lo primero que hacemos es calcular un pequeño rango definido por ***t_valle0-delta*** y ***t_valle0+delta***, que nos va permitir determinar si el bit muestreado corresponde al bit "0", si es distinto, es decir, corresponde al bit "1", o por el contrario, se ha dado algún error. El valor que se ha aplicado a delta es 3.

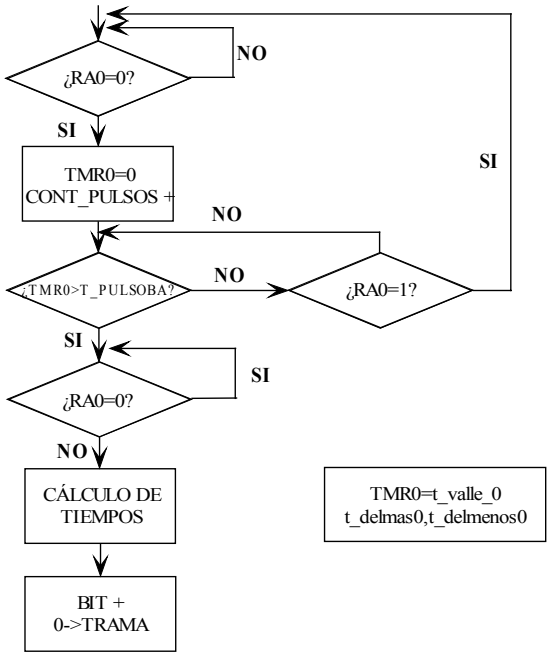


Figura 7.6. *Diagrama de flujo del muestreo del primer bit*

Muestreo de los bits restantes

Una vez que hemos recibido el primer bit, nos preparamos a recibir el resto. Por defecto, supondremos que el protocolo que cumple la trama que estamos recibiendo es el RECS80. Para comprobar si la trama es compatible con el protocolo RC5, nos centraremos en el número de pulsos que se reciben en el nivel alto del bit (variable ***pulsos***). En el momento en que el número de pulsos difiera con el de la variable *cont_pulsos*, supondremos que estamos recibiendo una trama que cumple el protocolo RC5. Entonces, almacenaremos el contenido de la variable *pulsos* en *cont_pulsos1* y consideraremos que el bit recibido es un "1".

- **Protocolo RECS80**

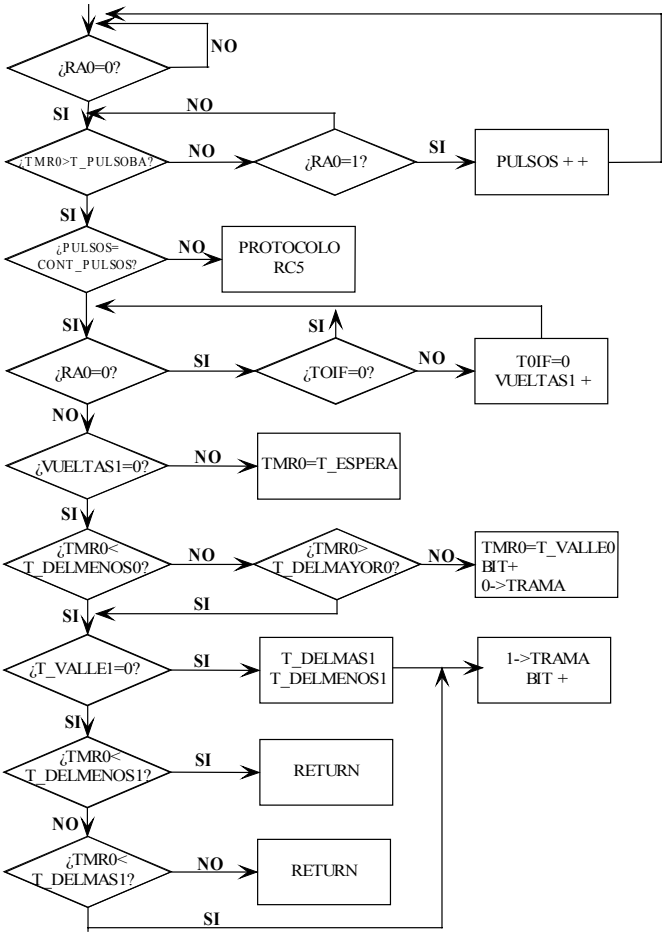


Figura 7.7. *Diagrama de flujo del muestreo de los bits restantes para el protocolo RECS80*

Cada vez que vayamos recibiendo cada uno de los bits, nos centraremos en los períodos de tiempo en que recibimos valores lógicos 0:

- Si el valor es menor o igual a *t_pulsoba*, sabemos que estamos recibiendo la portadora y dejamos pasar los pulsos.

- En el momento en que comprobamos que el valor es superior a *t_pulsoba* pueden darse otros dos casos:

a. Estamos midiendo el tiempo en baja de un bit.

b. Estamos midiendo el tiempo que transcurre entre la emisión de una trama y la repetición de la misma.

Para distinguir en cuál de los casos nos encontramos comprobamos si el valor de la variable ***vueltas1***, es igual o distinto de 0, cuyo valor se va incrementando cada vez que la cuenta del temporizador TMR0 llegue a 255.

Si estamos en el caso a., comprobamos si el valor obtenido pertenece al rango *(t_valle0-delta, t_valle0+delta*). Si no es así es que hemos recibido el tiempo en baja de un bit 1. Este valor característico lo guardaremos en ***t_valle1***, y solamente,

la primera vez que recibamos un bit 1; igualmente, se calcula los valores ***t_valle1-delta*** y ***t_valle1+delta***. Si en cualquier momento se detecta un tiempo en baja que no se encuentra dentro de ninguno de los dos rangos, se considerará que se ha producido un error y se saldrá de la subrutina de muestreo.

Cada vez que se detecta un bit debemos introducir el valor "0" ó "1" en la variable ***trama***, así como incrementar la variable ***bit***. Ambas dos, nos permitirá a la hora de reproducir la trama saber cuántos bits se habían recibido y de qué tipo eran.

Si estamos en el caso b y detectamos el tiempo que existe entre dos tramas, guardamos el valor de RTCC en ***t_espera***.

- **Protocolo RC5**

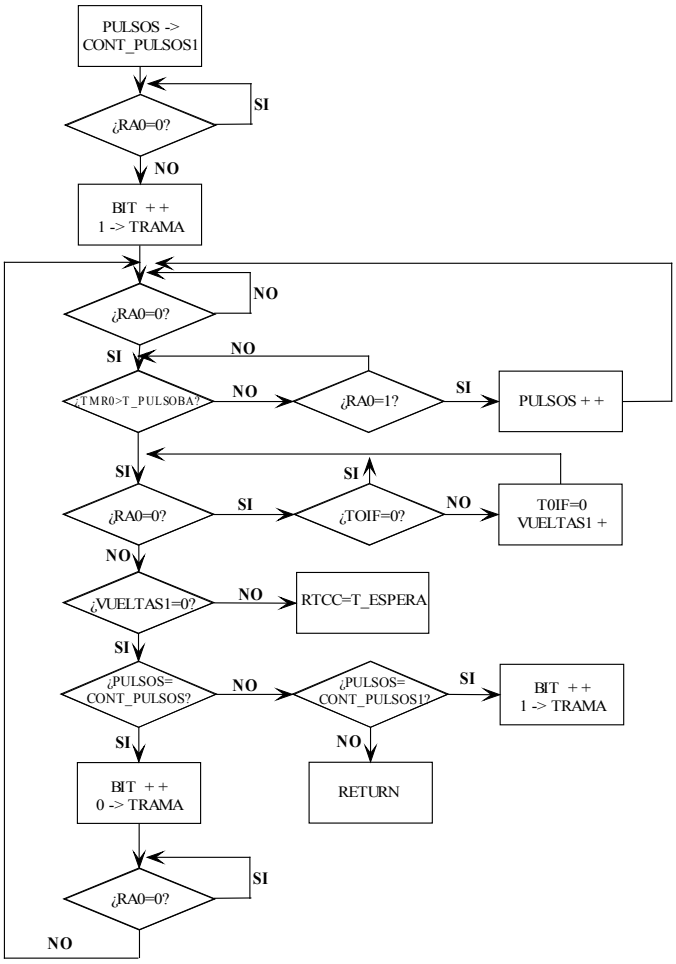


Figura 7.8. *Diagrama de flujo del muestreo de los bits restantes para el protocolo RC5*

Dentro del protocolo RC5, es necesario realizar la comparación del número de pulsos que recibe el nivel alto del bit con el que se contiene en las variables *cont_pulsos* y *cont_pulsos1*. Así, para cada uno de los bits, iremos almacenando el número de pulsos de su nivel alto en *pulsos*. Entonces, una vez que hemos

detectado que estamos en el nivel bajo de un bit comprobaremos si el número de pulsos obtenidos es igual al de la variable *cont_pulsos* o *cont_pulsos1*. De esta forma, detecto si es un bit "1" o un bit "0".

Para comprobar si estamos en el nivel bajo de un bit o en el tiempo que transcurre entre trama y trama, el procedimiento que se sigue es idéntico al del protocolo RECS80.

En cualquiera de ambos casos, una vez finalizado el muestreo de toda la trama, apago el led visible y espero a que no exista ninguna tecla pulsada. Una vez que se dé esta condición pasaré a grabar los datos en la memoria EEPROM.

C. Subrutinas de implementación del protocolo I^2C

Para poder leer o escribir en la memoria EEPROM es necesario implementar el protocolo I^2C. Para ello, se hace uso de distintas subrutinas que se comparten en las subrutinas de lectura y escritura de la memoria. A continuación veamos cuáles son los módulos o subrutinas empleados:

- **Módulo IISTRT**.- Generación de la condición de START.
- **Módulo IISTOP.-** Generación la condición de STOP.
- **Módulo IIOUT**.- Envío un dato tamaño byte a la memoria EEPROM. El dato a enviar se encuentra en la variable ***iidata***.
- **Módulo IIIN**.- Recepción un dato tamaño byte de la memoria EEPROM. La información recibida se introduce en la variable ***iidata***.
- **Módulo NACK**.- Testeo de la condición de ACK por parte del dispositivo receptor.
- **Módulo SACK**.- Envío de la condición de ACK al dispositivo esclavo.
- **Módulo FILLER**.- Se emplea para crear un retardo de 8 μsg.

A partir de estos módulos se han creado otras dos subrutinas:

- **Módulo EEPRE**.- Realiza el envío de la dirección del esclavo y la localización del dato en la EEPROM La variable en la que se introduce la dirección del esclavo es ***eeslv*** y la de la localización en la memoria EEPROM es ***eeddr0***.
- **Módulo EEPOLL**.- Se encarga de muestrear a la memoria EEPROM para determinar si ha finalizado el ciclo de escritura interno. Para ello, envía la dirección del esclavo a la EEPROM y comprueba si recibe la condición de ACK. Cuando lo reciba el ciclo de escritura habrá finalizado. Si después de 256 intentos no ha terminado sale de la subrutina y prepara al bus para la recepción o envío de otro dato.

C.1. Subrutina de grabación de los datos en la memoria EEPROM (subrutina eewrit)

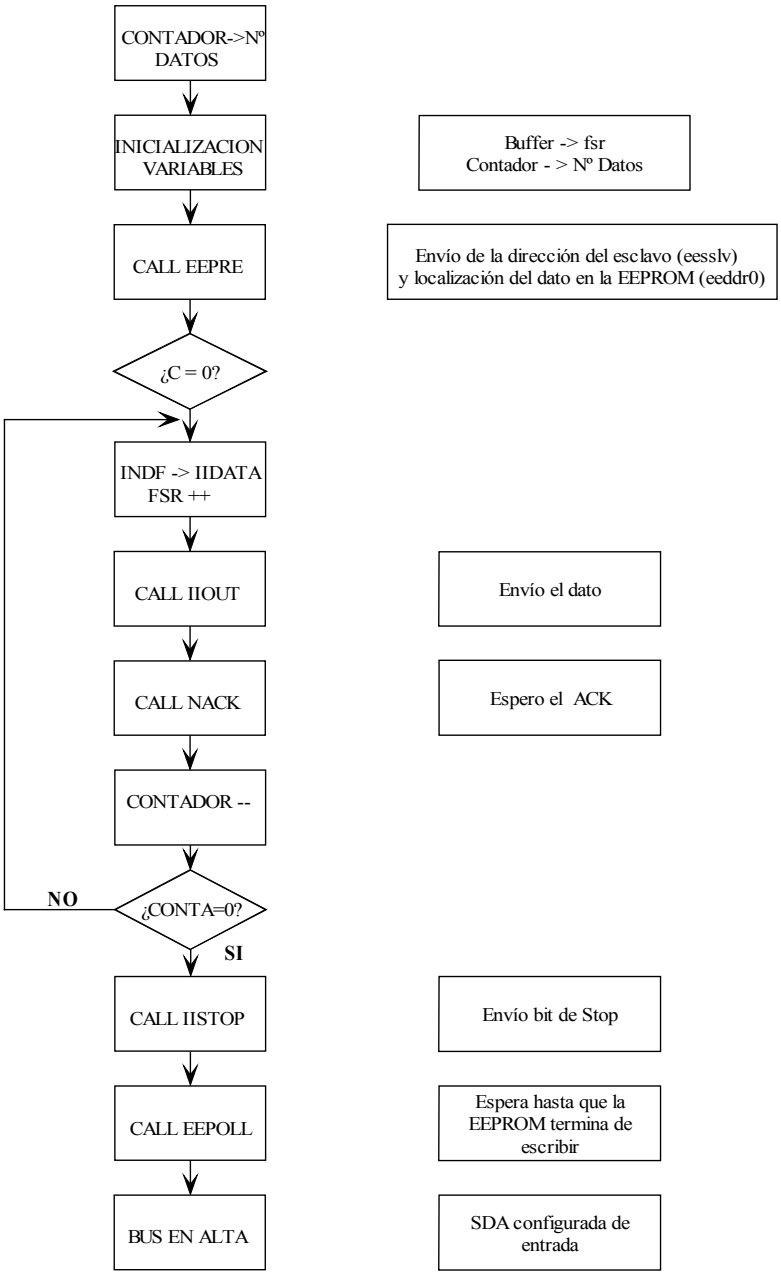


Figura 7.9. *Diagrama de flujo de la Subrutina de escritura*

Una vez que hemos obtenido los datos que caracterizan una trama debemos grabarlos en la memoria EEPROM. En total, debemos almacenar 11 variables, que son*: t_pulsoal, t_pulsoba, t_cabba, cont_pulsos, num_pulsos, cont_pulsos1 ó t_valle1 (dependiendo del protocolo), t_valle0, t_espera. trama, bit y vueltas1*. La dirección que indica en qué zona de la memoria debo colocar los datos viene dada por la variable ***eeddr0***, cuyo valor se ha calculado durante el muestreo del teclado.

Para enviar los datos a la memoria empleamos una variable de almacenamiento de ocho bytes llamado ***buffer***. Así, el primer paso es introducir las primeras ocho variables en *buffer*. Posteriormente, procedemos a enviar los datos a la memoria, implementando el protocolo I^2C.

Por cada una de las teclas, como se ha indicado anteriormente, se almacenan un total de 11 variables que ocupan 16 bytes de memoria. Como nuestro teclado matricial posee 16 teclas, el número de bytes ocupados de la memoria EEPROM externa asciende a 256 bytes. Es decir, el 88 % de la memoria queda libre. Esto podría permitir ser ampliado el programa con futuras aplicaciones.

C.2. Subrutina de lectura de los datos en la memoria EEPROM (subrutina eeread)

El procedimiento es similar al de la subrutina de grabación. La variable *eeddr0* indica la dirección a partir de la cual queremos leer la memoria y los datos obtenidos se almacenarían en la variable temporal *buffer*.

D. Subrutina de emisión de la señal

En esta parte de código, se procede a la emisión de la señal grabada a través de *RA1* hacia el circuito emisor implementado.

Una vez detectada la tecla pulsada, se procede a acceder a la zona de la memoria EEPROM reservada a la misma para recuperar los datos grabados e introducirlos en las variables correspondientes. Para ello, volvemos a emplear como variable intermedia *buffer*, que como se indicó anteriormente, consta de 8 bytes.

A continuación, se procede a calcular el valor con el que se debe inicializar TMR0 para cada una de las variables temporales.

Por último, se procede a actuar sobre la variable trama. Ésta ha recogido el valor de los bits de que consta la trama (hasta un número máximo de 48). Por ello, para poder acceder al primero de ellos, se deben realizar los giros necesarios a trama. El número a ejecutar se calcula restando 48 a la variable bit.

Una vez realizados todos estos preliminares, se enciende el led visible e inicia la emisión de la trama, comenzando por los datos de la cabecera, pasando por los bits y emitiendo, por último, el bit de stop. Después, se emite el tiempo que existe entre trama y trama. A continuación, se comprueba si la tecla sigue pulsada. En caso afirmativo, se sigue con el proceso de emisión de la trama. Si no es así, se apaga el led, se sale de la subrutina, se prepara el microcontrolador para la detección de una nueva tecla y se le introduce en el estado de reposo.

CAPITULO 8
CONCLUSIONES

A lo largo de los anteriores capítulo se ha tratado de explicar el funcionamiento de un "Mando a Distancia por Infrarrojos Universal", objetivo del presente proyecto. De esta forma, se ha construido un prototipo que:

- Pone en práctica la aplicación de un microcontrolador en un elemento tan común en la vida diaria como es un mando a distancia.

- Es capaz de muestrear y grabar una trama que emite una segundo mando a distancia para posteriormente reproducirla y, así, ser capaz de manejar el mismo dispositivo que el mando original.

Se ha podido comprobar que se ha desarrollado un prototipo que cumple con los objetivos planteados en la introducción a este Proyecto Fin de Carrera.

Olvidando los aspectos técnicos y centrándome en los conocimientos adquiridos, he de señalar la importancia que ha tenido este Proyecto Fin de Carrera en mi formación. Aunque son diversos los campos en los que se ha trabajado, querría apuntar fundamentalmente dos que, a mi parecer, son los más importantes.

Primero, el desarrollo del programa software empleado para implementar la función requerida, me ha permitido aprender cómo funciona un microcontrolador genérico y cómo programarlo para la realización de una actividad concreta.

Segundo, el diseño y elaboración de placas de circuito impreso. A mi parecer, éste es un campo de gran importancia en la formación de un estudiante de Ingeniería Electrónica y este Proyecto Fin de Carrera me ha permitido conocer el proceso completo de elaboración de una placa para poder implementar el prototipo de mando a distancia.

Así, este Proyecto Fin de Carrera ha permitido complementar los conocimientos adquiridos en las materias de la titulación Ingeniería en Electrónica.

APÉNDICE A
IMÁGENES DEL DISEÑO HARDWARE

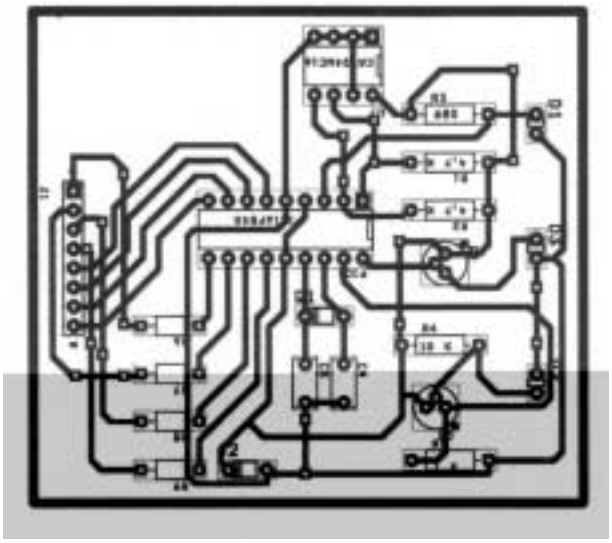


Figura A.1. *Layout de la placa de circuito impreso del mando a distancia*

Figura A.2. *Imagen de la placa de circuito impreso del mando a distancia*

Figura A.3. *Imagen del mando a distancia*

APÉNDICE B
COSTE ECONÓMICO DEL PROTOTIPO DISEÑADO

A. COSTE DEL PROTOTIPO

A.1 Coste del material

COMPONENTE	CANTIDAD	PRECIO UNIDAD (PTAS)
Microcontrolador PIC16F84A	1	1.418
EEPROM CAT24WC16	1	278
Resistencias	9	5
Transistor PNP	1	27
Transistor NPN	1	17
Condensadores	2	20
Cristal	1	135
Emisor de infrarrojos	1	69
Fotodiodo	1	245
Led visible	1	22
Porta pilas	1	20
Teclado	1	1.800
Placa	1	200
Tornillos	4	40
	TOTAL	4.356

A.2 Coste de Diseño

Estimando que el diseño completo de este prototipo de mando a distancia se realiza en un tiempo aproximado de 7 meses empleando una persona a jornada de trabajo completa y que el precio hora es de 1.600 ptas, obtenemos que:

$$(7\ meses)\ (20\ \frac{dias}{mes})\ (8\ \frac{horas}{dia})\ (1600\ \frac{ptas}{hora}) = 1.792.000\ ptas.\ de\ Coste\ de\ Diseño$$

B. COSTE DE PRODUCCIÓN

Supongamos que queremos fabricar una producción de 1.000 unidades. Debemos considerar los siguientes aspectos:

- **Material.-** Suponiendo que la compra de grandes volúmenes de material proporciona un descuento del 35%, el coste total del material necesario para crear las 1.000 unidades será de *2.831.400 ptas*.

- **Mano de obra.-** Considerando que para montar este mando a distancia un obrero no cualificado necesitaría 30 minutos y que cobra 1.000 ptas/hora, entonces el coste por mano de obra sería de *500.000 ptas*.

De esta forma un solo mando a distancia costaría:

$$\frac{1.792.000 + 2.831400 + 500.000}{1.000} = 5.123 \text{ ptas.}$$

Si añadimos 500 ptas. a cada mando por la carcasa y otros elementos y un 15% de beneficios, obtenemos un precio final de ***6.392 ptas.***

APÉNDICE C
PROGRAMA SOFTWARE

En este Apéndice se presenta el programa software diseñado en lenguaje ensamblador para implementar el Mando a Distancia por Infrarrojos Reprogramable desarrollado en este Proyecto Fin de Carrera.

```
Title "Desarrollo de un Mando a Distancia por Infrarrojos Reprogramable"
   List p=16F84A                    ; tipo de procesador


;*****************************************************************
;CONSTANTES
;*****************************************************************
w               equ     h'0'
f               equ     h'1'
RAM             equ     h'0C'
delta           equ     d'11'
pulsada         equ     d'0'
alta            equ     d'1'
inc             equ     d'16'
suma            equ     d'7'            ;Constantes para el cálculo de valores
suma1           equ     d'8'            ;temporales
suma2           equ     d'6'
suma3           equ     d'3'
rc5             equ     d'1'

;---------------------------- constantes identificación tecla ----------------------------------
t4              equ     h'4'
t5              equ     h'5'
t6              equ     h'6'
t7              equ     h'7'

;---------------------------- constantes para grabación memoria EEPROM -----------------
eeslv           equ     h'a0'           ;dirección eeprom
sda             equ     h'3'            ;A2 es I2C data
scl             equ     h'2'            ;A3 es I2C clock
iicrate         equ     h'2'

;---------------------------- constantes puerto A ------------------------------------------
ra0             equ     h'0'
ra1             equ     h'1'
ra2             equ     h'2'
ra3             equ     h'3'
ra4             equ     h'4'


;*****************************************************************
; REGISTROS
;*****************************************************************
indf    equ     h'0'
RTCC            equ     h'1'
pcl             equ     h'2'
status          equ     h'3'
```

```
fsr             equ     h'4'
porta           equ     h'5'
portb           equ     h'6'
eedata          equ     h'8'
eeadr           equ     h'9'
pclath          equ     h'a'
intcon          equ     h'b'
option_reg      equ     h'81'
trisa           equ     h'85'
trisb           equ     h'86'
eecon1          equ     h'88'
eecon2          equ     h'89'

;---------------------------------- bits del registro status ---------------------------------------
irp             equ     h'7'
rp1             equ     h'6'
rp0             equ     h'5'
not_to          equ     h'4'
not_pd          equ     h'3'
z               equ     h'2'
dc              equ     h'1'
c               equ     h'0'

;---------------------------------- bits del registro intcon ---------------------------------------
g               equ     h'7'
eeie            equ     h'6'
t0ie            equ     h'5'
inte            equ     h'4'
rbie            equ     h'3'
t0if            equ     h'2'
intf            equ     h'1'
rbif            equ     h'0'

;---------------------------------- bits del registro options ---------------------------------------
not_rbpu        equ     h'7'
intedg          equ     h'6'
t0cs            equ     h'5'
t0se            equ     h'4'
psa             equ     h'3'
ps2             equ     h'2'
ps1             equ     h'1'
ps0             equ     h'0'
```

```
;*****************************************************************
; VARIABLES
;*****************************************************************
    org RAM

contador        res     1
temp            res     1
t_cabba         res     1
t_pulsoba       res     1
t_pulsoal       res     1
cont_pulsos     res     1       ; pulsos de bit
bit             res     1
trama           res     6
t_valle1        res     1
t_bitba         res     1
t_valle0        res     1
num_pulsos      res     1       ; pulsos de cabecera
t_delmas0       res     1
t_delmenos0     res     1
t_delmas1       res     1
t_delmenos1     res     1
trama1          res     6
vueltas1        res     1
t_espera        res     1
iidata          res     1       ; byte para ser escrito o leido del bus I2C
bitctr          res     1       ; contador bit del bus I2C
data0           res     1       ; byte para la eeprom del bus I2C
eeddr0          res     1       ; byte bajo del bus I2C
timctr          res     1       ; contador de timeout
fillctr         res     1       ; contador de períodos de reloj del bus I2C
flag            res     1
contador1       res     1
buffer          res     8
alta1           res     1
alta2           res     1
baja1           res     1
baja2           res     1
pulsos          res     1
cont_pulsos1    res     1
```

```
;*****************************************************************
; PROGRAMA
;*****************************************************************
    org 0h

    goto Start

    org 4h

    goto Interrupc

;=================================================================
;Start.-Inicializamos el PIC y le metemos a dormir
;=================================================================
Start
    call Inicializac

Loop
    sleep
    goto Loop

;=================================================================
;Interrupc.- Servicio de Atención a la Interrupción
;=================================================================
Interrupc
    btfsc intcon,rbif
    goto Wakeup
    retfie


Wakeup
    movf portb,w            ; Compruebo si hay tecla pulsada
    movwf temp
    comf temp,w
    andlw b'11110000'
    btfsc status,z
    goto Verificac          ; Si no identifico tecla pulsada, vuelvo a comprobarlo
    goto Pulsada

Verificac
    movf portb,w
    movwf temp
    comf temp,w
    andlw b'11110000'
    btfss status,z
    goto Pulsada
    goto Volvemos           ; Si no está pulsada, volvemos
```

```
Pulsada
   movlw b'11110111'
   movwf portb
   nop
   movf portb,w
   btfsc portb,t4          ; Compruebo si pulso la tecla es "*"
   goto No_As
   goto Tecla_As

No_As                      ; Si no es el * es que no quiero grabar sino reproducir

   call Teclado            ; Compruebo la tecla pulsada

   btfss flag,pulsada      ; Compruebo si se pulsó alguna hasta ese momento
   goto Enviar
   goto Volvemos


Tecla_As                   ; En Tecla_As se espera que se pulse la tecla "#"
Ll movlw b'11110111'
   movwf portb
   movlw d'100'            ;Si no se ha pulsado, espero unos segundos y vuelvo a
   movwf contador1         ;intentarlo. Si aún así no se pulsa, es que quiero enviar
                           ; lo de "*"
L2

   call Tiempo

   movf portb,w
   btfss portb,t6
   goto Tecla_Al
   decfsz contador1,f
   goto L2
   movlw h'C0'
   movwf eeddr0
   goto Enviar


Tecla_Al
   movlw d'5'              ;Espero un tiempo antes de identificar en cuál quiero
   movwf contador1         ;grabar.

L3

   call Tiempo

   decfsz contador1,f
```

```
    goto L3

;Si llego aquí quiero grabar, con lo cual voy a identificar en cual.

    movlw d'120'            ; Durante un tiempo identifico la tecla pulsada. Si
    movwf contador1         ; pasado ese tiempo no se pulsó ninguna, salgo de
                            ; la subrutina.
L4

    call Tiempo

    call Teclado

    btfss flag,pulsada
    goto Seguimos
    decfsz contador1,f
    goto L4
    goto Volvemos


Seguimos

    call Muestrear

    goto Volvemos


Enviar                      ; Tengo la dirección de la que voy a transmitir.

    call Transmitir


Volvemos
    movlw b'11110000'
    movwf portb
    bcf intcon,rbif
    retfie


;-------------------------------------------------------------------------------------------------
; Teclado.- En esta subrutina,identifico la tecla grabada.Para ello, introduzco un 0 por
la ; columna y veo por qué fila se propaga.
;-------------------------------------------------------------------------------------------------
Teclado
    clrf eeddr0
    bcf flag,pulsada
    bcf flag,alta
```

```
movlw b'11111110'       ;escaneo la primera fila
movwf portb
nop
nop
btfss portb,t4
return
movlw inc
addwf eeddr0,f
btfss portb,t5
return
addwf eeddr0,f
btfss portb,t6
return
addwf eeddr0,f
btfss portb,t7
return

addwf eeddr0,f
movlw b'11111101'       ;la segunda fila
movwf portb
nop
nop
btfss portb,t4
return
movlw inc
addwf eeddr0,f
btfss portb,t5
return
addwf eeddr0,f
btfss portb,t6
return
addwf eeddr0,f
btfss portb,t7
return

addwf eeddr0,f
movlw b'11111011'       ;la tercera fila
movwf portb
nop
nop
btfss portb,t4
return
movlw inc
addwf eeddr0,f
btfss portb,t5
return
addwf eeddr0,f
```

```
    btfss portb,t6
    return
    addwf eeddr0,f
    btfss portb,t7
    return

    addwf eeddr0,f
    movlw b'11110111'           ;la cuarta fila
    movwf portb
    nop
    nop
    btfss portb,t4
    return
    movlw inc
    addwf eeddr0,f
    btfss portb,t5
    return
    addwf eeddr0,f
    btfss portb,t6
    return
    addwf eeddr0,f
    btfss portb,t7
    return
    bsf flag,pulsada
    return


;-------------------------------------------------------------------------------------------------
;Muestrear.- Muestreo la trama que entra por a0 y la guardo en la ;dirección de
; memoria correspondiente.
;-------------------------------------------------------------------------------------------------
Muestrear
    bsf porta,ra4
    clrf num_pulsos
    clrf bit
    clrf cont_pulsos
    clrf vueltas1
    clrf trama
    clrf trama+1
    clrf trama+2
    clrf trama+3
    clrf trama+4
    clrf trama+5
    clrf trama1
    clrf trama1+1
    clrf trama1+2
    clrf trama1+3
```

```
    clrf trama1+4
    clrf trama1+5
    clrf t_valle1
    clrf RTCC

    movlw d'3'
    movwf contador

;Cuando recibimos un 1, empieza la recepción. Si en 10 sg no he recibido nada,
;volvemos
    movlw d'50'
    movwf contador1
    movlw d'255'
    movwf temp
Ppio
    btfsc porta,0
    goto Salto

    btfss intcon,t0if
    goto Ppio
    bcf intcon,t0if
    decfsz temp,f
    goto Ppio
    movlw d'255'
    movwf temp
    decfsz contador1,f
    goto Ppio
    bcf porta,ra4
    return

    bcf intcon,t0if

Salto
    clrf RTCC

Cresta
    btfsc porta,0                ; Compruebo cuando pasamos de 1 a 0.
    goto Cresta

    movf RTCC,w
    clrf RTCC
    movwf alta1
```

```
Valle
    btfss porta,0                ; Espero a que llegue otra parte alta del pulso
    goto Valle
    movf RTCC,w
    clrf RTCC
    movwf baja1

    incf num_pulsos,f

;Siguiente pulso
Alta1
    btfsc porta,0
    goto Alta1

    movf RTCC,w
    clrf RTCC
    movwf alta2

Baja1
    btfss porta,0
    goto Baja1

    movf RTCC,w
    clrf RTCC
    movwf baja2

    incf num_pulsos,f

    movf alta2,w
    addwf alta1,f
    clrc
    rrf alta1,f

    movf baja2,w
    addwf baja1,f
    clrc
    rrf baja1,f

    decfsz contador,f
    goto Alta1

    movf alta1,w
    movwf t_pulsoal

    movf baja1,w
    movwf t_pulsoba
```

```
;Pasamos a contar pulsos hasta que reciba el t_baja cabecera.

    bsf status,rp0
    movlw b'01010100'          ; Prescalado de 32
    movwf option_reg
    bcf status,rp0

Cresta1
    btfsc porta,0              ; Espero a una parte baja
    goto Cresta1

    clrf RTCC

Comprob
    movf RTCC,w                    ; Compruebo si es una parte baja de pulso o
    movwf temp                 ; La parte baja de cabecera.
    btfss temp,3
    goto Valle1
    goto Mas31

Valle1
    btfss porta,0
    goto Comprob

    incf num_pulsos,f          ; Es parte baja de pulso e incremento nº de pulsos.
    goto Cresta1

Mas31
    btfsc porta,0              ; Es la parte baja de cabecera.
    goto Fin_cab
    goto Mas31

Fin_cab
    movf RTCC,w
    movwf t_cabba

    bsf status,rp0
    movlw b'01010010'          ; Prescalado de 8
    movwf option_reg
    bcf status,rp0
```

```
;Aquí empiezan a llegar los bits (empieza a llegar una parte alta)
;Analizamos el primer bit

Cresbit1
    btfsc porta,0
    goto Cresbit1

    clrf RTCC
    incf cont_pulsos,f

Comprbit1
    movf RTCC,w
    movwf temp
    btfss temp,5
    goto Valbit1            ; Compruebo si es un pulso o parte baja del bit
    goto Masbit1

Valbit1
    btfss porta,0
    goto Comprbit1
    goto Cresbit1           ; Es parte baja de pulso

Masbit1                     ; Es parte baja de bit
    btfss porta,0           ; Cuando pase a 1 es que ha pasado todo el t de baja
    goto Masbit1
    movf RTCC,w
    movwf  t_valle0         ; Guardo T1 y calculo T1-delta y T1+delta
    movlw delta
    addwf t_valle0,w
    movwf t_delmas0
    movlw delta
    subwf t_valle0,w
    movwf t_delmenos0
    incf bit,f
    clrc                    ; Guardo el primer bit como un cero y su tiempo en
                            ; baja
    rlf trama,f
    rlf trama+1,f
    rlf trama+2,f
    rlf trama+3,f
    rlf trama+4,f
    rlf trama+5,f
    clrf pulsos
```

```
; Leo los bits siguientes
Cresta2
    btfsc porta,0
    goto Cresta2

    clrf RTCC
    incf pulsos,f

Comprob1
    movf RTCC,w                 ; Compruebo si es parte baja de pulso o de bit.
    movwf temp
    btfss temp,5
    goto Valle2
    goto Mas311

Valle2
    btfss porta,0
    goto Comprob1
    goto Cresta2

Mas311
    movf pulsos,w
    xorwf cont_pulsos,w
    btfsc status,z
    goto Es_RC5             ; Si pulsos es distintos cont_pulsos, entonces es trama
                            ; de protocolo RC5.

    btfsc porta,0           ; Es parte baja de bit.
    goto Fin0               ; Ha terminado la parte baja.
    btfsc intcon,t0if
    goto Vuelt1
    goto Mas311

Vuelt1
    bcf intcon,t0if
    incf vueltas1,f
    goto Mas311

Fin0
    clrw                    ; Debo comprobar si es parte baja de bit o t entre
    xorwf vueltas1,w        ; trama y trama.
    btfss status,z          ; Si salta la sigte instrucción es que vueltas1 no es 0
    goto Fin_trama          ; Si hay vueltas es que he medido el tiempo entre
                            ; trama y trama.
```

```
    movf RTCC,w
    movwf temp

    clrf RTCC
    movf t_delmenos0,w
    subwf temp,w
    btfsc status,c              ;c=1 nuestro tiempo(temp) > t_delmenos
    goto Esmayor0               ; El tiempo es mayor que t_delmenos
    goto Outrange


Esmayor0
    movf t_delmas0,w            ; Compruebo si el tiempo es mayor que t_delmas
    subwf temp,w
    btfsc status,c              ; c=1 nuestro tiempo (temp) >t_delmas
    goto Outrange

;Si llego aqui es que nuestro tiempo es igual a t_valle0
    clrc
    goto Sigo

Outrange
; El tiempo no es t_valle0. Compruebo si he grabado anteriormente t_valle1.
; Si no, lo guardo como t_valle1. Si ya tenia t_valle1, compruebo que esta
; en el rango. Si no esta en el rango, error. Si esta en el rango lo guardo
; como un 1.

    clrw
    xorwf t_valle1,w
    btfss status,z              ; Si z=0 es que no estaba guardado t_valle1
    goto Yaguardado

    movf temp,w
    movwf t_valle1
    movlw delta
    addwf t_valle1,w            ; Calculo los márgenes para t_valle1
    movwf t_delmas1
    movlw delta
    subwf t_valle1,w
    movwf t_delmenos1
    setc
    goto Sigo

Yaguardado
    movf t_delmenos1,w
    subwf temp,w
    btfsc status,c              ; c=1 t_valle1>t_delmenos1
```

```
    goto Esmayor1           ; Es mayor que t_delmenos1
    goto Fallo              ; No entra dentro de ningun rango

Esmayor1
    movf t_delmas1,w        ;Compruebo si es mayor que t_delmas
    subwf temp,w
    btfsc status,c          ; c=1 t_valle1>t_delmas1
    goto Fallo
;Entra en rango=> Es un 1
    setc
    goto Sigo

Fallo
    bsf status,rp0
    movlw b'01010000'       ; Pull up habilitado. Prescalado de 2
    movwf option_reg
    bcf status,rp0

    return


Sigo
    rlf trama,f             ; Guardo el dato
    rlf trama+1,f
    rlf trama+2,f
    rlf trama+3,f
    rlf trama+4,f
    rlf trama+5,f
    incf bit,f
    clrf RTCC

    goto Comprob1

Es_RC5                      ; La trama detectada es del protocolo RC5
    bsf flag,rc5
    movf pulsos,w
    movwf cont_pulsos1

RC_51
    btfss porta,0
    goto RC_51
    incf bit,f
    setc
    rlf trama,f
    rlf trama+1,f
    rlf trama+2,f
    rlf trama+3,f
```

```
    rlf trama+4,f
    rlf trama+5,f

RC_52
    btfsc porta,0
    goto RC_52
    clrf RTCC

Comprob2
    movf RTCC,w
    movwf temp
    btfss temp,5
    goto Valle3
    goto Mas312

Valle3
    btfss porta,0
    goto Comprob2
    incf pulsos,f
    goto RC_52

Mas312
    btfsc porta,0          ; Es parte baja de bit.
    goto Fin1              ; Ha terminado la parte baja.
    btfsc intcon,t0if
    goto Vuelt2
    goto Mas312

Vuelt2
    bcf intcon,t0if
    incf vueltas1,f
    goto Mas312

Fin1
    clrw                   ; Debo comprobar si es parte baja de bit o t entre
    xorwf vueltas1,w       ; trama y trama.
    btfss status,z         ; Si salta la sigte instrucción es que vueltas1 no es 0
    goto Fin_trama         ; Si hay vueltas es que he medido el tiempo entre
                           ; trama y trama.

    movf pulsos,w                  ; Compruebo si es bit 0 o 1
    xorwf cont_pulsos,w
    btfsc status,z
    goto RC5Bit1
    incf bit,f
    clrc
    goto Tramas
```

```
RC5Bit1
    movf pulsos,w
    xorwf cont_pulsos1,w
    btfss status,z
    return

    incf bit,f
    setc
    goto Tramas

Tramas
    rlf trama,f
    rlf trama+1,f
    rlf trama+2,f
    rlf trama+3,f
    rlf trama+4,f
    rlf trama+5,f

RC_53
    btfss porta,0
    goto RC_53
    goto RC_52

Fin_trama
    movf RTCC,w
    movwf t_espera

    bcf porta,ra4

;Vuelvo a comprobar si hay una tecla pulsada
Comp
    movlw b'11110000'
    movwf portb
    bcf intcon,rbif
    movf portb,w
    movwf temp
    comf temp,w
    andlw b'11110000'
    btfsc status,z
    goto Salvado
    goto Comp                   ; Si hay tecla pulsada, espero a que deje de
                                ; presionarla.
```

```
;-------------------------------------------------------------------------------------------------
;Salvado Almacena los datos muestreados en la memoria EEPROM.
;-------------------------------------------------------------------------------------------------

Salvado

;Guardo los tiempos
    movf t_pulsoal,w
    movwf buffer
    movf t_cabba,w
    movwf buffer+1

    btfsc flag,rc5
    movf cont_pulsos1,w

    movf t_valle1,w
    movwf buffer+2
    movf t_valle0,w
    movwf buffer+3
    movf t_espera,w
    movwf buffer+4
    movf num_pulsos,w
    movwf buffer+5
    movf cont_pulsos,w
    movwf buffer+6
    movf t_pulsoba,w
    movwf buffer+7

    call eewrit

    movf vueltas1,w
    movwf buffer
    movf bit,w
    movwf buffer+1
    movf trama,w
    movwf buffer+2
    movf trama+1,w
    movwf buffer+3
    movf trama+2,w
    movwf buffer+4
    movf trama+3,w
    movwf buffer+5
    movf trama+4,w
    movwf buffer+6
    movf trama+5,w
    movwf buffer+7
```

```
    movlw d'8'
    addwf eeddr0,f

    movlw d'8'
    movwf temp
    call eewrit

    bsf status,rp0
    movlw b'01010000'          ; Pull up habilitado. Prescalado de 2
    movwf option_reg
    bcf status,rp0

    return


;-------------------------------------------------------------------------------------------------
; Transmitir.- Transmitimos la trama muestreada previamente empleando los datos
grabados en la memoria EEPROM.
;-------------------------------------------------------------------------------------------------

Transmitir

    call eeread

;Saco todo lo del buffer
    movf buffer,w
    movwf t_pulsoal
    movf buffer+1,w
    movwf t_cabba
    movf buffer+2,w
    movwf t_valle1
    movf buffer+3,w
    movwf t_valle0
    movf buffer+4,w
    movwf t_espera
    movf buffer+5,w
    movwf num_pulsos
    movf buffer+6,w
    movwf cont_pulsos
    movf buffer+7,w
    movwf t_pulsoba
    movlw d'8'
    addwf eeddr0,f



    call eeread
```

```
    movf buffer,w
    movwf vueltas1
    movf buffer+1,w
    movwf bit
    movf buffer+2,w
    movwf trama
    movf buffer+3,w
    movwf trama+1
    movf buffer+4,w
    movwf trama+2
    movf buffer+5,w
    movwf trama+3
    movf buffer+6,w
    movwf trama+4
    movf buffer+7,w
    movwf trama+5

; Para evitar que emita una señal bloqueante si no existe nada grabado en esa tecla,
; compruebo que t_pulsoal es distinto de FF.
    movf t_pulsoal,w
    btfsc t_pulsoal,7
    return

    movlw d'13'
    xorwf bit,w
    btfsc status,z
    bsf flag,rc5
    goto Tiempos

    movf t_valle1,w          ;Si es el RC5 el dato de t_valle1 pasa a cont_pulsos1
    movwf cont_pulsos1


Tiempos                      ; Calculo los tiempos con los que debo inicializar las
    movlw d'255'             ; variables temporales.
    movwf temp
    movf t_espera,w
    subwf temp,w
    movwf t_espera

    movlw d'255'
    movwf temp
    movf t_pulsoal,w
    subwf temp,w
    movwf t_pulsoal
```

```
movlw d'255'
movwf temp
movf t_pulsoba,w
subwf temp,w
movwf t_pulsoba

    movlw d'255'
    movwf temp
    movf t_valle0,w
    subwf temp,w
    movwf t_valle0

    movlw d'255'
    movwf temp
    movf t_valle1,w
    subwf temp,w
    movwf t_valle1

    movlw d'255'
    movwf temp
    movf t_cabba,w
    subwf temp,w

    movwf t_cabba


;Calculo los tiempos válidos
    movlw suma
    addwf t_espera,f
    movlw suma3
    addwf t_cabba,f
    movlw suma2
    addwf t_valle1,f
    addwf t_valle0,f
    movlw suma3
    addwf t_pulsoal,f
    incf t_pulsoal,f


;-------------------------------------------------------------------------------------------------
;Em_trama.- Se procede a la emisión de la trama
;-------------------------------------------------------------------------------------------------

Em_trama
    bsf porta,ra4                   ; Encendemos el LED
    bcf intcon,t0if
; Nivel alto de la cabecera
```

```
    movf num_pulsos,w
    movwf temp

    call Partealta

; Nivel bajo de la cabecera
;Compruebo si vueltas=0

    clrf RTCC
    bsf status,rp0
    movlw b'01010100'          ; Prescalado de 32
    movwf option_reg
    bcf status,rp0

    bcf intcon,t0if
    movf t_cabba,w
    movwf RTCC

Ciclo3
    btfss intcon,t0if
    goto Ciclo3

    bsf status,rp0
    movlw b'01010010'
    movwf option_reg
    bcf status,rp0


; Comienza la emisión de bits
    bcf intcon,t0if
    movf bit,w
    movwf contador
    movf trama,w
    movwf trama1
    movf trama+1,w
    movwf trama1+1
    movf trama+2,w
    movwf trama1+2
    movf trama+3,w
    movwf trama1+3
    movf trama+4,w
    movwf trama1+4
    movf trama+5,w
    movwf trama1+5
```

```
; Compruebo el nº de bits que se han capturado y hago los giros suficientes para que en
; el bit + significativo se sitúe el primer bit de la trama.
    movlw d'48'
    movwf temp
    movf bit,w
    subwf temp,f
    clrw
    xorwf temp,w
    btfsc status,z
    goto Trama

Inicial

    call RLFS

    decfsz temp,f
    goto Inicial

; Ahora ya empiezo a coger cada bit de la trama y a emitirlos

Trama
    bsf status,rp0
    movlw b'01010000'          ; Pull up habilitado. Prescalado de 2
    movwf option_reg
    bcf status,rp0

    call RLFS

    movf cont_pulsos,w
    movwf pulsos

    btfss status,c
    goto Txbit0
    goto Txbit1

Txbit0

    call Bits

    bsf status,rp0
    movlw b'01010010'          ; Pull up habilitado. Prescalado de 8
    movwf option_reg
    bcf status,rp0

    movf t_valle0,w
    movwf RTCC
    goto Partebaja
```

```
Txbit1
    btfsc flag,rc5
    goto Em_RC5

    call Bits

    bsf status,rp0
    movlw b'01010010'          ; Pull up habilitado. Prescalado de 8
    movwf option_reg
    bcf status,rp0

    movf t_valle1,w
    movwf RTCC
    goto Partebaja

Bits
    movf pulsos,w
    movwf temp

    call Partealta

    return

RLFS
    rlf trama1,f
    rlf trama1+1,f
    rlf trama1+2,f
    rlf trama1+3,f
    rlf trama1+4,f
    rlf trama1+5,f

    return

Partebaja
    bcf intcon,t0if
Ciclo4
    btfss intcon,t0if
    goto Ciclo4
    decfsz contador,f
    goto Trama

    btfsc flag,rc5
    goto Tespera

    bsf status,rp0
    movlw b'01010000'          ; Pull up habilitado. Prescalado de 2
```

```
    movwf option_reg
    bcf status,rp0

;Emito bit de stop
    movf pulsos,w
    movwf temp

    call Partealta


;Emito tiempo de espera hasta la siguiente trama
Tespera
    bsf status,rp0
    movlw b'01010010'          ; Pull up habilitado.Prescalado de 8
    movwf option_reg
    bcf status,rp0

    bcf porta,ra1
    clrw
    xorwf vueltas1,w
    btfsc status,z
    goto Sinvuelt
    movf vueltas1,w
    movwf temp

    clrf RTCC
    bcf intcon,t0if

Ciclob
    btfss intcon,t0if
    goto Ciclob
    bcf intcon,t0if
    decfsz temp,f
    goto Ciclob

Sinvuelt
    bcf intcon,t0if
    movf t_espera,w
    movwf RTCC

Ciclob1
    btfss intcon,t0if
    goto Ciclob1


    bsf status,rp0
    movlw b'01010000'          ; Pull up habilitado. Prescalado de 2
```

```
    movwf option_reg
    bcf status,rp0


; Ahora se observa si la tecla sigue pulsada
    movlw b'11110000'
    movwf portb
    bcf intcon,t0if
    movf portb,w
    movwf temp
    comf temp,w
    andlw b'11110000'
    btfss status,z
    goto Em_trama

    bcf porta,ra4               ; Apagamos el LED.
    return


Em_RC5                          ; Emisión para protocolo RC5
    movf cont_pulsos1,w
    movwf pulsos
    bsf status,rp0
    movlw b'01010010'
    movwf option_reg
    bcf status,rp0

    movf t_valle0,w
    movwf RTCC
    goto Partebaja



Partealta                       ; Emisión parte alta de un bit
Paso
    movf t_pulsoal,w
    movwf RTCC

    bsf porta,ra1
    bcf intcon,t0if

Ciclo
    btfss intcon,t0if
    goto Ciclo
    movf t_pulsoba,w
    movwf RTCC
```

```
    bcf intcon,t0if
    bcf porta,ra1

Ciclo1
    btfss intcon,t0if
    goto Ciclo1
    decfsz temp,f
    goto Paso
    return


;-----------------------------------------------------------------------------------------------
; Ciclos para crear retardos temporales
-----------------------------------------------------------------------------------------------
Tiempo
    movlw d'255'
    movwf temp

Tiempo1

    call delay_400

    decfsz temp,f
    goto Tiempo1
    return

delay_400
    movlw d'132'
    movwf contador
delay
    decfsz contador,f
    goto delay
    return


;===============================================================
; Subrutinas para implementar el protocolo I2C
;===============================================================
;-----------------------------------------------------------------------------------------------
;MÓDULO IISTRT
;Genera condición de START
;La línea de datos pasa a baja mientras la línea de reloj está en alta
;-----------------------------------------------------------------------------------------------
iistrt
    bsf status,rp0
    movlw b'00001101'           ;a3 y a2 de entrada y el resto de salida
    movwf trisa
```

```
    bcf status,rp0

    btfss porta,scl
    goto iistrtx

    btfss porta,sda
    goto iistrtx

    bsf status,rp0 ; RP 1
    bcf trisa,sda                ;sda de salida
    bcf status,rp0
    bcf porta,sda                ;sda pasa a baja (START)

    bsf status,rp0
    bcf trisa,scl                ;scl y sda de salida
    bcf status,rp0
    bcf porta,scl

    call filler

iistrtx
    retlw 0


;-------------------------------------------------------------------------------------------------
;Módulo IISTOP
; Genera la condición de STOP
; SDA pasa a 1 mientras SCL está en alta
;-------------------------------------------------------------------------------------------------
iistop
    bsf status,rp0
    movlw b'00000001'            ;sda y scl de salida
    movwf trisa
    bcf status,rp0

iistopa bcf porta,sda                    ;sda en baja

    bsf status,rp0
    bsf trisa,scl
    bcf status,rp0
    call filler

    bsf status,rp0
    bsf trisa,sda
    bcf status,rp0
    call filler
```

```
    retlw 0
;-------------------------------------------------------------------------------------------------
;Módulo IIOUT
; Envío un byte al esclavo.
; El dato está en iidata
;-------------------------------------------------------------------------------------------------
iiout
    movlw d'8'
    movwf bitctr
    bsf status,rp0
    movlw b'00000001'          ;sda y scl de salida
    movwf trisa
    bcf status,rp0

iiout1
    bsf status,rp0
    bcf trisa,scl              ;scl como salida
    bcf status,rp0
    bcf porta,scl
    rlf iidata,f
    btfss status,c
    goto iiout2
    bsf status,rp0
    bsf trisa,sda
    bcf status,rp0
    goto iiout3

iiout2
    bsf status,rp0
    bcf trisa,sda              ;sda como salida
    bcf status,rp0
    bcf porta,sda              ;la pongo en baja
iiout3      nop
    nop
    bsf status,rp0
    bsf trisa,scl              ;scl a 1
    bcf status,rp0
    nop
    btfss porta,scl
    goto iioute
    call filler                ;mantengo scl en alta
    decfsz bitctr,f            ;decremento nº de bits
    goto iiout1

    bsf status,rp0
    bcf trisa,scl              ;scl como salida
    bcf status,rp0
```

```
    bcf porta,scl           ;lo pongo para recibir después el ACK.

    call filler
    call filler

iioute
    retlw 0

;-----------------------------------------------------------------------------------------
;Módulo IIIN: Recibe 8 bits desde el esclavo
;Dato recibido en iidata
;-----------------------------------------------------------------------------------------
iiin
    movlw d'8'
    movwf bitctr            ;contador

    bsf status,rp0
    movlw b'00001001'       ;sda como entrada y scl de salida
    movwf trisa
    bcf status,rp0
    bcf porta,scl           ;scl en baja
    call filler

    clrf iidata             ;limpio el buffer

iiin1
    bsf status,rp0
    bsf trisa,scl           ;scl en alta
    bcf status,rp0
    nop

Etiqu
    btfss porta,scl
    goto Etiqu
    btfsc porta,sda
    bsf status,c
    nop
    goto Giro
    bcf status,c
    nop

Giro
    bsf status,rp0
    bcf trisa,scl
    bcf status,rp0
    bcf porta,scl
```

```
    call filler
    rlf iidata,f
    decfsz bitctr,f
    goto iiin1
    goto iiinx

iiinx
    retlw 0

;-----------------------------------------------------------------------------------------------
;Módulo NACK: Testea el ACK desde el receptor
;Salida:    Si se recibe el ACK-> c=0
;-----------------------------------------------------------------------------------------------
nack
    bsf status,rp0
    bsf trisa,sda                ;sda de entrada y scl de salida
    bcf status,rp0

    bsf status,rp0
    bsf trisa,scl                ;scl en alta
    bcf status,rp0

    btfsc porta,sda
    goto nack2
    bcf status,c
    goto nack1

nack2
    bsf status,c

nack1
    retlw 0

;-----------------------------------------------------------------------------------------------
;Módulo SACK: Envío del ACK al esclavo
;-----------------------------------------------------------------------------------------------
sack
    bsf status,rp0
    bcf trisa,sda                ;sda de salida
    bcf status,rp0
    bcf porta,sda
    nop

    bsf status,rp0
    bsf trisa,scl                ;scl a 1
    bcf status,rp0
    btfsc porta,sda
```

```
    goto sack2
    bcf status,c
    goto sack1

sack2
    bsf status,c

sack1
    bsf status,rp0
    bsf trisa,sda                ;sda de entrada
    bcf status,rp0
    retlw 0

;-------------------------------------------------------------------------------------------------
;Módulo FILLER.- Crea el retardo
;-------------------------------------------------------------------------------------------------
filler
    movlw iicrate
    movwf fillctr

filler1
    decfsz fillctr,f
    goto filler1
    retlw 0


;-------------------------------------------------------------------------------------------------
;Módulo EEWRIT.- Graba un byte en la EEPROM
;Dato a grabar en data0
;eeddr0 dirección de la EEPROM
;-------------------------------------------------------------------------------------------------
eewrit
    movlw d'8'
    movwf temp
    movlw buffer
    movwf fsr

    call eepre                   ;envío la dirección

    btfsc status,c
    goto ewerr

Emisión
    movf indf,w
    movwf iidata
    incf fsr,f
```

```
    call iiout              ;hago el envío del dato
    call nack

    decfsz temp,f
    goto Emisión

ewerr
    call iistop             ;envío bit de stop
    call eepoll             ;espera hasta que la EEPROM termina de escribir
    bsf status,rp0
    bsf trisa,sda           ;bus en alta
    bcf status,rp0

    retlw 0

;-----------------------------------------------------------------------------------------------
;Módulo EEPRE
;Envía la dirección del esclavo y la localización del dato en la EEPROM
; eeslv dirección del esclavo
;eeddr0 es la localización del dato en la EEPROM
;-----------------------------------------------------------------------------------------------
eepre
    call iistrt             ;envío start
    movlw eeslv             ;dirección del esclavo
    movwf iidata
    bcf iidata,0            ;lo pongo de escritura

    call iiout              ;hago el envío

    call nack               ;testeo el ACK del esclavo

    btfsc status,c          ;compruebo si ACK recibido
    goto eeprex

    movf eeddr0,w
    movwf iidata

    call iiout

    call nack

eeprex
    retlw 0
```

```
;-------------------------------------------------------------------------------------------------
;Modulo EEREAD.- Leo un dato de la memoria
;Entrada: Dirección de la EEPROM en eeddr0
;Salida: Dato leido en data0
;-----------------------------------------------------------------
hereda
    movlw d'8'
    movwf temp
    movlw buffer
    movwf fsr

    call eepre                  ;envío la dirección

    btfsc status,c              ;si c=1, no hubo ACK
    goto erderr
    bsf status,rp0
    bcf trisa,scl
    bcf status,rp0
    bcf porta,scl

    call filler                 ;introducimos retardo

    call iistrt                 ;condición de start

    movlw eeslv                 ;dirección del esclavo
    movwf iidata
    bsf iidata,0                ;como lectura

    call iiout                  ;envío dirección del esclavo

    call nack

    btfsc status,c              ;ACK recibido
    goto erderr

eeread2
    call iiin                           ;recibo dato de la memoria

    movf iidata,w               ;guardo el dato en buffer
    movwf indf
    incf fsr,f
    decfsz temp,f
    goto eeread1

    bsf status,rp0
    bsf trisa,sda
    bcf status,rp0
```

```
    bsf status,rp0
    bsf trisa,scl
    bcf status,rp0

    bsf status,rp0
    bcf trisa,scl
    bcf status,rp0
    bcf porta,scl

;tiempo de no ACK
    bcf status,c                ;limpio error flag
erderr
    call iistop             ;envio stop

    retlw 0

eeread1
    bsf status,rp0
    bcf trisa,sda
    bcf status,rp0
    bcf porta,sda
    bsf status,rp0
    bsf trisa,scl
    bcf status,rp0

    bsf status,rp0
    bcf trisa,scl
    bcf status,rp0
    bcf porta,scl

    goto eeread2

;-----------------------------------------------------------------------------------------------
;Módulo EEPOLL.- Muestrea la EEPROM para determinar si ha finalizado el ciclo de
;escritura. Envía la dirección del esclavo a la EEPROM. Si el ACK es recibido, el ciclo
;de escritura ha finalizado. Si después de 256 intentos (30 ms) no lo ha recibido,sale.
;-----------------------------------------------------------------------------------------------
eepoll
    clrf timctr                 ;comienzo el contador de timeout

eepoll0
    call iistrt                 ;envío start

    movlw eeslv
    movwf iidata
    bcf iidata,0                ;bit0=0 (para escritura)
```

```
    call iiout                  ;envío a la EEPROM

eepoll1
    call nack                   ;testeo ACK del esclavo
    btfss status,c              ;c=0 -> ACK recibido
    goto eepollx
    decfsz timctr,f
    goto eepoll0

eepollx
    bsf status,rp0
    bcf trisa,scl
    bcf status,rp0
    bcf porta,scl

    bsf status,rp0
    bsf trisa,sda               ;bus en alta
    bcf status,rp0

    bsf status,rp0
    bsf trisa,scl
    bsf status,rp0
    retlw 0


;-------------------------------------------------------------------------------------------------
; Inicialización
;-------------------------------------------------------------------------------------------------
Inicializac
    clrf RTCC
    clrf porta
    clrf portb                  ;Inicializo a 0 porta y portb

    bsf status,rp0              ;Paso al banco 1
    movlw b'00001101'
    movwf trisa                 ;a0 recibe la señal de receptor
                                ;a2 y a3 son sda y scl
                                ;a1 emite la señal
    movlw h'F0'
    movwf trisb                 ; rb7-rb4 de entrada
                                ; rb3-rb0 de salida

    movlw b'01010000'           ;pull up habilitado.Prescalado de 8
    movwf option_reg
    bcf status,rp0

    bcf intcon,rbif             ;limpio el flag
```

```
bcf intcon,t0if
bsf intcon,rbie          ;habilito la mascara
bcf intcon,t0ie
movlw b'11110000'
movwf portb              ;escaneo

retfie

end
```

APÉNDICE D
HOJAS DE ESPECIFICACIONES

En este Apéndice se puede encontrar las hojas de especificaciones de los siguientes componentes:

- Microcontrolador PIC16F84A
- Memoria EEPROM CAT24WC16
- Fotodiodo SFH203P
- Emisor de infrarrojos SFH485

BIBLIOGRAFÍA

- "Microcontroladores PIC. La solución en un chip"
  E. Martín Cuenca, J. M. Angulo Usategui, I. Angulo Martínez
  Editorial Paraninfo, 2ª edición, 1998
- "Microcontroladores PIC. Diseño práctico de aplicaciones"
  J. M. Angulo Usategui, I. Angulo Martínez
  Editorial Mc Graw Hill, 2ª edición, 1998.
- Hoja de especificaciones del microcontrolador PIC16F84A.
- Hoja de especificaciones de la memoria CAT24WC16.
- Hoja de especificaciones de la memoria 24C04A.
- Hoja de especificaciones del fotodiodo SFH203P.
- Hoja de especificaciones del emisor de infrarrojos SFH485
- Internet