PROGRAMACION DE MICROCONTROLADORES PIC

EN LENGUAJE C

OPCIÓN II:

LIBROS DE TEXTO Y PROTOTIPO DIDÁCTICOS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELECTRÓNICO

PRESENTA:

CÉSAR PERALTA RUIZ

TEHUACAN, PUE. MARZO DE 2008.

AGRADECIMIENTOS

A Dios por sus bendiciones, permitirme dar este paso y a quien dedico este logro.

A la memoria de mi Padre:
Por haber sido un ejemplo de sabiduría, honradez y sencillez. y que espiritualmente siempre lo eh tenido presente en cada paso de mi vida

A mi madre:
Por traerme a esta vida, por su amor, entereza, dedicación y afrontar tantos obstáculos, por sacarme adelante, gracias por ser el pilar de mi existencia y estar ahí en esos momentos difíciles de mi vida.

A mi Hermana Hortensia.
Por ser mí amiga, por todos los momentos buenos y malos que hemos compartido, por brindarme su cariño y apoyo incondicional y estar ahí en los momentos más críticos de mi vida.

A los Señores del medio Aeronáutico:
Sr Ricardo Álvarez Mtz., Sr José Castillo Lara, Sr Javier Sobrado, Sr José Canaán Castro, Sr Salvador Gandarillas, a todos ellos gracias por brindarme su amistad, sus consejos y su apoyo durante toda mi carrera.

A la memoria de mis Seres queridos:
Abuelita Hortensia García, Abuelita Raquel Flores, Abuelo Germán Peralta, Tía Delfina García, Tío Álvaro García, gracias por su amor y por todos los consejos que en vida me brindaron.

A los tíos y tías:
Rita Ruiz, Elizabeth Ruiz, Hugo Ruiz, Juan Ruiz, Abel Ruiz, Rodrigo Ruiz, Rafael García, Fernando otáñez, Ángel Otáñez, Adán Otáñez, Sandra Otáñez, Graciela Otáñez, Alta Otáñez, José Otáñez y Antonio Otáñez, a todos gracias porque atinadamente han estado para bien en cada parte de mi vida.

Mis primos:
Yesenia, Amparo, Pilar, Aidé, Darío, Rabid, Ludivina, Elizabeth.

Al Ing. Alfredo Juárez García:

Por su paciencia, por asesorarme durante la elaboración de este trabajo.

A los revisores de este trabajo:
Ing Alberto Cortez, M.C. Gerardo Cortez Lozano, M.C. Angélica Granados.

Al Instituto Tecnológico de Tehuacán y en especial a los ingenieros del área de Ingeniería Electrónica por haberme permitido ser parte de esta gran familia. Sin el apoyo de ellos a lo largo de la carrera no habría llegado a consolidarme como un Ingeniero.

Y agradezco también a mis grandes amigos:

Erick Reyes Ruiz

Santiago Valdivia

Mario Cortes

Héctor Cervantes

Isai Figueroa

José Antonio Gil

Jesús Antonio Jiménez

Karla Salas

Ricardo Álvarez Jr.

Humberto Pérez

Abel Macías

Tobías Sánchez

Pedro Flores

Yaredi

Marco Morales

Felipe Gonzalez

Noé Gandarillas

Placido López

Pablo

Y a todos aquellos con los que compartí grandes momentos en el transcurso de mi carrera.

GRACIAS

TABLA DE CONTENIDO

PROLOGO

Los microcontroladores están invadiendo el mundo. Están presentes en el entorno de nuestra vida diaria. Se pueden encontrar controlando los hornos microondas y los televisores de nuestro hogar, en los teclados y ratones de los computadores y en los automóviles. En el bolsillo llevamos unos cuantos entre los del teléfono móvil, los que tienen las modernas llaves del coche y los mandos a distancia y la alarma doméstica. La conquista masiva de estos diminutos computadores ha comenzado y gobernarán la mayor parte de los aparatos que fabricamos y usamos los humanos.

Además de lo anterior esta obra se realiza con el objetivo primordial de ser una guía en el aprendizaje de la programación de microcontroladores PIC basados en lenguaje C, ya que las extensas áreas de aplicación de los microcontroladores que se pueden considerar ilimitadas, exigirán un gigantesco trabajo de diseño acorde a la función y enfoque particular de cada proyecto.

Aprender a manejar y aplicar los microcontroladores sólo se consigue desarrollando prácticamente en diseños reales. Sucede lo mismo con cualquier instrumento musical, cualquier deporte y muchas otras actividades.

El lenguaje C fue creado entre 1970 y 1972 por Brian Kernighan y Dennis Ritchie para escribir el código del sistema operativo UNIX.

Desde su nacimiento se fue implantando como el lenguaje de programación de sistemas favorito para muchos programadores, sobre todo por ser un lenguaje que conjugaba la abstracción de los lenguajes de alto nivel con la eficiencia del lenguaje máquina. Los programadores de sistemas que trabajaban sobre MS-DOS y Macintosh también utilizaban C, con lo cual la práctica totalidad de aplicaciones de sistema para microordenadores y para sistemas UNIX está escrito en este lenguaje.

A mediados de los ochenta el C se convierte en un estándar internacional ISO. Este estándar incluye tanto la definición del lenguaje como una enorme biblioteca de funciones para entrada/salida, tratamiento de textos, matemáticas, etc.

A mediados de los ochenta se crea el C++, extensión de C orientada a objetos. El C++ se convierte en estándar ISO en 1998. En el momento actual, el lenguaje C no va a modificarse más. Será el C++ el que incorporará nuevos cambios.

Es por lo cual que algunas compañías como es el caso de mikroElektronika se dieron a la tarea de adaptar el lenguaje C a una plataforma para la programación de microcontroladores PIC e inclusive para la programación de procesadores digitales de señales como lo son los DSPic's.

Este libro utiliza como entorno de programación al compilador mikroC de la compañía mikroElektronika y como base de proyectos al microcontrolador PIC16F877A. Se escogió al compilador mikroC como entorno de programación por su ambiente grafico y por la gran cantidad de librerías que trae incluidas para diversas aplicaciones, lo que lo hace muy versátil y provoca que la programación sea amena para el lector de este libro.

En esta versión de este libro se opto por utilizar al microcontrolador PIC16F877A por la gran cantidad de aplicaciones que se pueden realizar con él, además que su costo es accesible y no tiene limitaciones en el mercado de la región.

En el capítulo 1 de esta obra se realiza un breve esbozo histórico de los microcontroladores PIC, así como los tipos de arquitecturas y las memorias existentes para su diseño, el capitulo 2 trata las generalidades de los microcontroladores PIC y las características del microcontrolador PIC16F877A. El capitulo 3 abarca el ambiente de programación de compilador mikroC, es decir es espacio de trabajo, así como las diferentes ventanas y asistentes que se utilizan durante la elaboración de un programa o proyecto. El capitulo 4 se adentra en el desarrollo de aplicaciones que se pueden realizar con mikroC y además de los componentes sintácticos o tokens de los que se compone el mismo. Al pasar el capitulo 5 se trataran los temas "tipos" y las declaraciones, que se utilizan durante la realización de un programa. El capitulo 6 describe algunas de las funciones y librerías más utilizadas en los proyectos que se realizan en el capítulo 7. Y en dicho capitulo se realizan una serie de proyectos básicos, comenzando con un proyecto por demás sencillo pero que es esencial dado que en él se explica la forma de realizar un proyecto y que engloba la mayor parte de los temas expuestos desde el capitulo 3 hasta el capitulo 6. También se incluye un CD con los respectivos códigos de cada proyecto asi como los circuitos para la simulación.

Para finalizar con esta obra existen dos apéndices, los cuales engloban la instalación del programa mikroC y la utilización del Simulador PROTEUS en el modulo ISIS que es de gran ayuda durante la realización de proyectos.

Cabe mencionar que la gran mayoría de información integrada a este libro fue obtenida de las documentaciones técnicas de Microchip

(www.microchip.com), microElektronika (www.mikroe.com) y Labcenter Electronics (www.labcenter.co.uk) creador del software PROTEUS VSM. A quienes se agradece la información publicada en sus páginas electrónicas.

También se aclara que esta obra es realizada sin fines de lucro y para el uso de la enseñanza y el auxilio en materias que involucren microcontroladores, tales como adquisición de datos o señales y su tratamiento, controladores o filtros de tipo digital etc.,

CAPITULO 1
INTRODUCCIÓN A LOS MICROCONTROLADORES PIC

1.1 RESEÑA HISTÓRICA

Recibe el nombre de **controlador** el dispositivo que se emplea para el gobierno de uno o varios procesos. Por ejemplo, el controlador que regula el funcionamiento de un horno dispone de un sensor que mide constantemente su temperatura interna y, cuando traspasa los límites prefijados, genera las señales adecuadas que accionan los actuadores que intentan llevar el valor de la temperatura dentro del rango estipulado.

Aunque el concepto de controlador ha permanecido invariable a través del tiempo, su implementación física ha variado frecuentemente. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta, posteriormente se emplearon los microprocesadores, que se rodeaban con chips de memoria y de entradas y salidas (E/S) sobre una tarjeta de circuito impreso. En la actualidad, todos los elementos del controlador se han podido incluir en un chip, el cual recibe el nombre de **microcontrolador**. Realmente consiste en un sencillo pero completo computador contenido en el corazón (chip) de un circuito integrado.

1.1.1 BREVE ESBOZO HISTÓRICO

En 1971 Intel fabrica el primer microprocesador (el 4004) de tecnología pMOS. Este era un microprocesador de 4 bits y fue fabricado por Intel a petición de Datapoint Corporation con el objeto de sustituir la CPU de terminales inteligentes que eran fabricadas en esa fecha por Datapoint mediante circuitería discreta. El dispositivo fabricado por Intel resultó 10 veces más lento de lo requerido y Datapoint no lo compró, de esta manera Intel comenzó a comercializarlo. El 4004 era un **microprocesador de 4 bits**, contenía 2,300 transistores y corría a 108 KHz podía direccionar

sólo 4096 localidades de memoria de 4 bits, reconocía 45 instrucciones y podía ejecutar una instrucción en 20 μs en promedio. Este procesador se utilizó en las primeras calculadoras de escritorio.

1972. Las aplicaciones del 4004 estaban muy limitadas por su reducida capacidad y rápidamente Intel desarrolló una versión más poderosa (el 8008), el cual podía manipular bytes completos, y por lo tanto fue un microprocesador de 8 bits. La memoria que este podía manejar se incrementó a 16 Kbyte's, sin embargo, la velocidad de operación continuó igual.

1973. Intel lanza al mercado el 8080 el primer microprocesador de tecnología NMOS, lo cual permite superar la velocidad de su predecesor (el 8008) por un factor de diez, es decir, el 8080 puede realizar 500 000 operaciones por segundo, además se incrementó la capacidad de direccionamiento de memoria a 64 Kbyte's. A partir del 8080 de Intel se produjo una revolución en el diseño de microcomputadoras y varias compañías fabricantes de circuitos integrados comenzaron a producir microprocesadores. Algunos ejemplos de los primeros microprocesadores son: el IMP-4 y el SC/MP de National Semiconductors, el PPS-4 y PPS-8 de Rockwell International, el MC6800 de Motorola, el F-8 de Fairchild.

1975. Zilog lanza al mercado el Z80, uno de los microprocesadores de 8 bits más poderosos. En ese mismo año, Motorola abate dramáticamente los costos con sus microprocesadores 6501 y 6502 (este último adoptado por APPLE para su primera microcomputadora personal). Estos microprocesadores se comercializan en $20 y $25 (dls. USA) respectivamente. Esto provoca un auge en el mercado de microcomputadoras de uso doméstico y un caos en la proliferación de lenguajes, sistemas operativos y programas (ningún producto era compatible con el de otro fabricante).

En 1976 surgen las primeras microcomputadoras de un sólo chip, que más tarde se denominarán **microcontroladores**. Dos de los primeros microcontroladores, son el 8048 de Intel y el 6805R2 de Motorola.

198x. En la década de los 80's comienza la ruptura entre la evolución tecnológica de los microprocesadores y la de los microcontroladores, Ya que los primeros han ido incorporando cada vez más y mejores capacidades para las aplicaciones en donde se requiere el manejo de grandes volúmenes de información y por otro

lado, los segundos han incorporado más capacidades que les permiten la interacción con el mundo físico en tiempo real, además de mejores desempeños en ambientes de tipo industrial.

1.2 ¿QUE ES UN MICROCONTROLADOR?

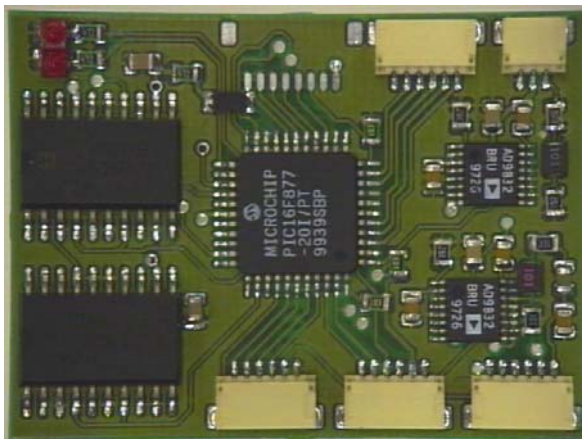


FIGURA 1.1

Microcontrolador PIC16f877 de Microchip y chips de soporte

Básicamente un microcontrolador se describe como un Circuito Integrado o chip que incluye en su interior las tres unidades funcionales de un ordenador o computadora: CPU, Memoria y Unidades de E/S, es decir, se trata de una computadora completa en un solo circuito integrado. Aunque sus prestaciones son limitadas, además de dicha integración, su característica principal es su alto nivel de especialización. Aunque los hay del tamaño de un sello de correos, lo normal es que sean incluso más pequeños, ya que, lógicamente, forman parte del dispositivo que controlan.

Un **microcontrolador** es un microprocesador optimizado para ser utilizado en el control de equipos electrónicos. Los microcontroladores representan la inmensa mayoría de los chips de ordenadores vendidos, sobre un 50% son controladores "simples" y el restante corresponde a DSP's más especializados. Mientras se pueden tener uno o dos microprocesadores de propósito general en casa, se podría probablemente tener distribuido entre los electrodomésticos del hogar una o dos docenas de microcontroladores. Pueden encontrarse en casi cualquier dispositivo eléctrico como automóviles, lavadoras, hornos microondas, teléfonos, etc.

1.3 DIFERENCIA ENTRE MICROCONTROLADOR Y MICROCROPROCESADOR.

El microcontrolador es un computador completo, aunque de limitadas prestaciones, que está contenido en el chip de un circuito integrado programable y se destina a gobernar una sola tarea con el programa que reside en su memoria, con un mínimo de chips externos de apoyo. La idea es que el chip se coloque en el dispositivo, conectado a la fuente de energía y de información que necesite, y eso es todo. Un microprocesador tradicional no le permitirá hacer esto, ya que espera que todas estas tareas sean manejadas por otros chips.

Por ejemplo, un microcontrolador típico tendrá un generador de reloj integrado y una pequeña cantidad de memoria RAM y ROM/EPROM/EEPROM, significando que para hacerlo funcionar, todo lo que se necesita son unos pocos programas de control y un cristal de sincronización. Los microcontroladores disponen generalmente también de una gran variedad de dispositivos de entrada/salida, como convertidores de analógico a digital, temporizadores, UARTs y buses de interfaz serie especializados, como I^2C y CAN. Frecuentemente, estos dispositivos integrados pueden ser controlados por instrucciones de procesadores especializados. Los modernos microcontroladores frecuentemente incluyen un lenguaje de programación integrado, como el BASIC o el C que se utiliza bastante con este propósito.

Los microcontroladores negocian la velocidad y la flexibilidad para facilitar su uso. Debido a que se utiliza bastante espacio en el chip para incluir funcionalidad, como los dispositivos de entrada/salida o la memoria que incluye el microcontrolador, se ha de prescindir de cualquier otra circuitería.

Los microcontroladores más comunes en uso se mencionan en la tabla 1.

Algunas arquitecturas de microcontrolador están disponibles por tal cantidad de vendedores y en tantas variedades, que podrían tener, con total corrección, su propia categoría. Entre ellos encontramos, principalmente, las variantes de 8051 y Z80.

Empresa	8 bits	12 bits	14 bits	16 bits	32 bits	64 bits
Atmel AVR						
Freescale (antes Motorola)	68HC05, 68HC08, 68HC11	x	x	68HC12, 68HC16	683xx	**x**
Hitachi	H8	x	x	x	x	**x**
Holtek	HT8					
Intel	MCS-48 (familia 8048) MCS51 (familia 8051) 8xC251	x	x	MCS96, MXS296	x	**x**
National Semiconductor	COP8	x	x	x	x	**x**
Microchip	Familia 10f2xx	Familia 12Cxx	Familia 12Fxx, 16Cxx y 16Fxxx	18Cxx y 18Fxx. PIC24 y DSPic	x	**x**
NEC	78K					
Parallax						
ST	ST 62,ST 7					
Texas Instruments	TMS370, MSP430					
Zilog	Z8, Z86E02					

Figura 1.2

Familias de microcontroladores comúnmente utilizados

1.4 TIPOS DE ARQUITECTURAS DE MICROCONTROLADORES

1.4.1 ARQUITECTURA VON NEUMAN

La **Arquitectura Von Neumann** se refiere a las arquitecturas de computadoras que utilizan el mismo dispositivo de almacenamiento tanto para las instrucciones como para los datos (a diferencia de la arquitectura Harvard). El término se acuñó en el documento *First Draft of a Report on the EDVAC* (1945), escrito por el conocido matemático John Von Neumann, que propuso el concepto de programa almacenado. Dicho documento fue redactado en vistas a la construcción del sucesor de la computadora ENIAC, y su contenido fue desarrollado por Presper Eckert, John Mauchly, Arthur Burks, y otros durante varios meses antes de que Von Neumann redactara el borrador del informe.

Los ordenadores con arquitectura Von Neumann constan de cinco partes: La unidad aritmético-lógica o ALU, la unidad de control, la

memoria, un dispositivo de entrada/salida y el bus de datos que proporciona un medio de transporte de los datos entre las distintas partes. Esto se puede observar en la figura 1.2.

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

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

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

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

1º. La limitación de la longitud de las instrucciones por el bus de datos, que hace que el microprocesador tenga que realizar varios accesos a memoria para buscar instrucciones complejas.
2º. La limitación de la velocidad de operación a causa del bus único para datos e instrucciones que no deja acceder simultáneamente a unos y otras, lo cual impide superponer ambos tiempos de acceso.

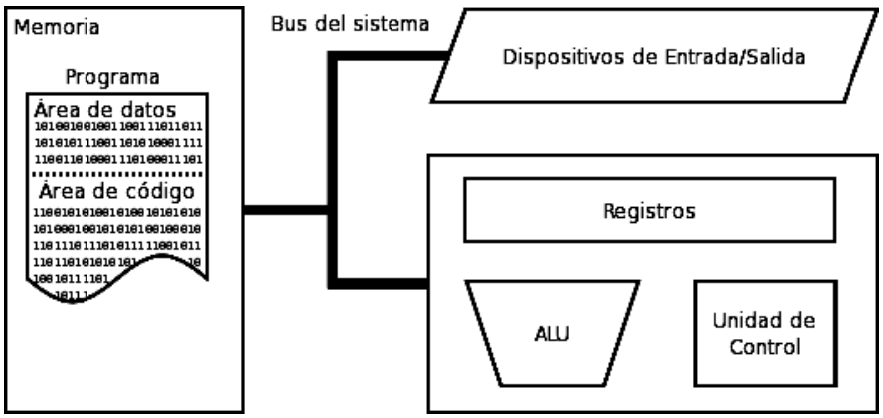


FIGURA 1.2
Arquitectura Von Neumann

1.4.2 ARQUITECTURA HARVARD

El término **Arquitectura Harvard** originalmente se refería a las arquitecturas de computadoras que utilizaban dispositivos de almacenamiento físicamente separados para las instrucciones y para los datos (en oposición a la Arquitectura Von Neumann). El término proviene de la computadora Harvard Mark I, que almacenaba las instrucciones en cintas perforadas y los datos en interruptores.

La arquitectura Harvard tiene la unidad central de proceso o CPU conectada a dos memorias (una con las instrucciones y otra con los datos) por medio de dos buses diferentes (ver figura 1.3).

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

Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC, el set de instrucciones y el bus de memoria de programa pueden diseñarse de tal manera que todas las instrucciones tengan una sola posición de memoria de programa de longitud.

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

Ventajas de esta arquitectura:

1º. 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.

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

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

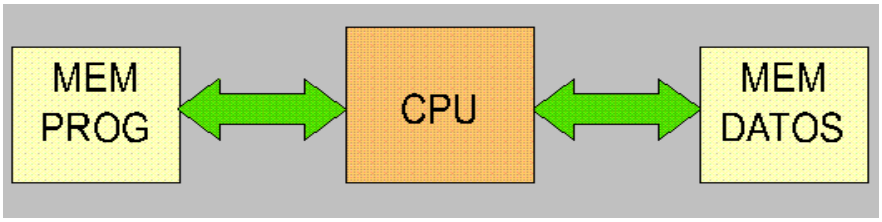


FIGURA 1.3

Arquitectura Harvard

1.5 RECURSOS COMUNES A TODOS LOS MICROCONTROLADORES.

Al estar todos los microcontroladores integrados en un chip, su estructura fundamental y sus características básicas son muy parecidas. Todos deben disponer de los bloques esenciales: Procesador, memoria de datos y de instrucciones, líneas de entrada y salida, oscilador de reloj y módulos controladores de periféricos. Sin embargo, cada fabricante intenta enfatizar los recursos más idóneos para las aplicaciones a las que se destinan preferentemente.

En este apartado se hace un recorrido de todos los recursos que se hallan en todos los microcontroladores describiendo las diversas alternativas y opciones que pueden encontrarse según el modelo seleccionado.

1.5.1 ARQUITECTURA BÁSICA

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en el momento presente se impone la Arquitectura Harvard

La arquitectura Harvard a la cual responden los microcontroladores PIC, como se mencionó anteriormente dispone de dos memorias independientes una, que contiene sólo instrucciones y otra, sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias. Figura 1.4.

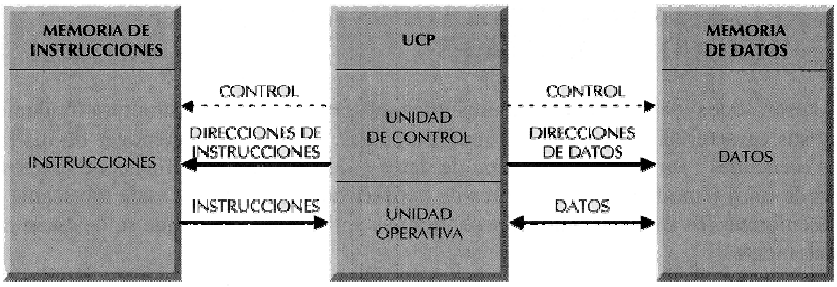


FIGURA 1.4

La arquitectura Harvard dispone de dos memorias independientes para datos y para instrucciones, permitiendo accesos simultáneos.

1.5.2 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 de operación de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

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

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

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

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

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

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

1.5.3 MEMORIA

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

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

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

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

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

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

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

- ROM con máscara

Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. El elevado coste del diseño de la máscara sólo hace aconsejable el empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varios miles de unidades.

- OTP

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

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

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

- EPROM

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

- EEPROM

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read OnIy Memory). Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado

de un PC. Es muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie.

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

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

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

Este tipo de memoria es relativamente lenta.

- FLASH

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

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

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

Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados "en circuito", es decir, sin tener que sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil permite que pueda modificarse el programa durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas, etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

1.5.4 PUERTOS DE ENTRADA Y SALIDA

La principal utilidad de las terminales que posee el encapsulado que contiene un microcontrolador es soportar las líneas de E/S que comunican al computador interno con los periféricos exteriores.

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

1.5.5 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 exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red R-C.

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

CUESTIONARIO.

1. ¿Cuál es el concepto de controlador?
2. ¿Quiénes son los pioneros en la elaboración de microcontroladores?
3. Investigue la correlación entre el contenido y los enunciados de la ley de Moore.
4. Defina el concepto de microcontrolador.
5. ¿Cuál es la diferencia entre microprocesador y microcontrolador?
6. ¿Cuáles son las filosofías de arquitectura que se utilizan en el diseño de microcontroladores?
7. ¿De qué partes está constituido un microprocesador con arquitectura Von Neumann?
8. ¿Básicamente como está constituido un microcontrolador con arquitectura Harvard?
9. Mencione las características comunes a todos los microcontroladores.
10. Mencione algunos tipos de memorias que se utilizan en la fabricación de microcontroladores

CAPÍTULO 2
MICROCONTROLADORES PIC

Para la realización de las prácticas de este libro se ha elegido la familia PIC de microchip por diversos motivos:

1. Por la cantidad de información disponible sobre estos microcontroladores, y es que para las aplicaciones más habituales la elección de una versión adecuada de PIC es la mejor solución.
2. Por su sencillez de manejo, tienen un juego de instrucciones reducido, de 35 en la gama media.
3. Por su precio, que es comparativamente inferior al de sus competidores
4. Por su velocidad y promedio de parámetros en consumo, tamaño, etc.
5. Porque posee gran variedad de herramientas, tanto de software como de hardware, baratas y fáciles de utilizar.

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.

2.1 CARACTERÍSTICAS RELEVANTES

*a) La arquitectura del procesador sigue el modelo **Harvard**:*

En esta arquitectura, la CPU se conecta de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos y así permitir su acceso simultaneo, ver figura 2.1

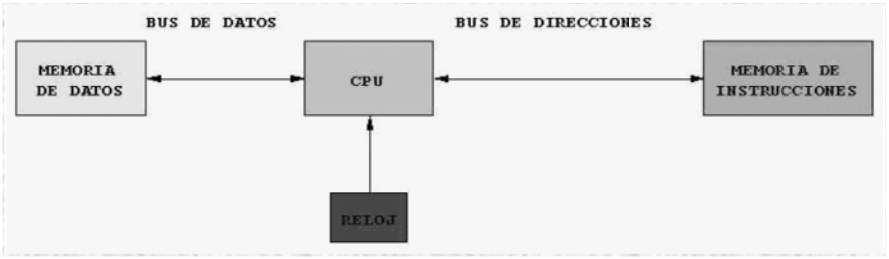


FIGURA 2.1
Arquitectura Harvard

b) Se aplica la técnica de segmentación ("pipe-line") en la ejecución de las instrucciones:

La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente. De esta forma se puede ejecutar cada instrucción en un ciclo (un ciclo de instrucción equivale a cuatro ciclos de reloj) 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. Figura 2.2.

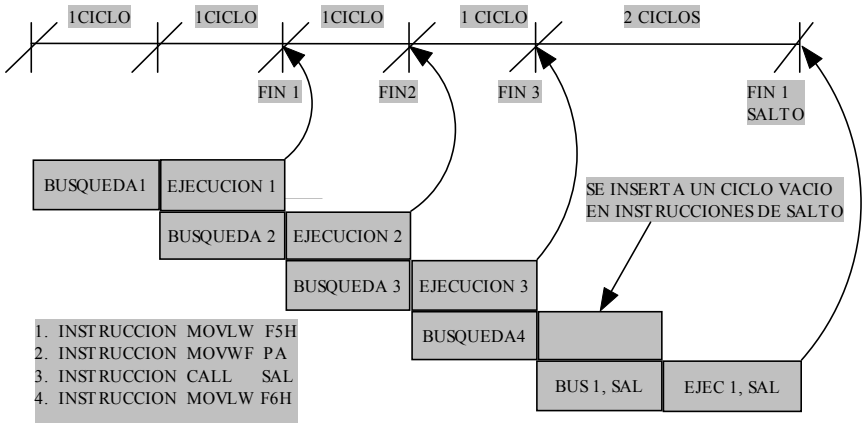


FIGURA 2.2

La segmentación permite solapar en el mismo ciclo la fase de una instrucción y la búsqueda de la siguiente, excepto en las instrucciones de salto

c) El formato de todas las instrucciones es de la misma longitud:

Las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits. Las de la gama media tienen 14 bits y más 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.

d) 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.

5º Todas las instrucciones son ortogonales:

Cualquier instrucción puede manejar cualquier elemento de la arquitectura como fuente o como destino.

e) Arquitectura basada en un banco de registros:

Esto significa que todos los objetos del sistema (puertas de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.

f) Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes:

La gran variedad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.

g) 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.

2.2 LAS GAMAS PIC

Para resolver aplicaciones sencillas se precisan pocos recursos; en cambio, las aplicaciones grandes requieren numerosos y potentes. Siguiendo esta filosofía, Microchip construye diversos modelos de microcontroladores orientados a cubrir, de forma óptima, las necesidades de cada proyecto. Así, hay disponibles microcontroladores sencillos y baratos para atender las aplicaciones simples y otros complejos y más costosos para las de mucha envergadura.

Con las cuatro 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.

2.2.1 GAMA BAJA

La gama baja de los PIC encuadra nueve modelos fundamentales en la actualidad. A muchos de estos microcontroladores de gama baja se les llama “enanos” porque solamente disponpen de 8 patillas.

La memoria de programa puede contener 512, 1 K. y 2 K palabras de 12 bits, y ser de tipo ROM, EPROM aunque también hay modelos con memoria OTP. La memoria de datos puede tener una capacidad

comprendida entre 25 y 73 bytes. Sólo disponen de un temporizador (TMR0), un repertorio de 33 instrucciones y un número de terminales para soportar las E/S comprendido entre 12 y 20. El voltaje de alimentación admite un valor muy flexible comprendido entre 2 y 6,25 V, lo cual posibilita el funcionamiento mediante pilas corrientes teniendo en cuenta su bajo consumo (menos de 2 mA a 5 V y 4 MHz).

Al igual que todos los miembros de la familia PIC16/17, los componentes de la gama baja se caracterizan por poseer los siguientes recursos.

1. Sistema POR (POWER ON RESET):

Todos los PIC tienen la facultad de generar un auto reset al conectarles la alimentación.

2. Perro guardián (Watchdog):

Existe un temporizador que produce un reset automáticamente si no es recargado antes de que pase un tiempo prefijado. Así se evita que el sistema quede ciclado dado que en esa situación el programa no recarga dicho temporizador y se genera un reset.

3. Código de protección:

Cuando se procede a realizar la grabación del programa, puede protegerse para evitar su lectura.

4. Líneas de E/S de alta corriente:

Las líneas de E/S de los PIC pueden proporcionar o absorber una corriente de salida comprendida entre 20 y 25 mA, capaz de excitar directamente ciertos periféricos.

5. Modo de reposo (bajo consumo o SLEEP):

Ejecutando una instrucción (SLEEP), la CPU y el oscilador principal se detienen y se reduce notablemente el consumo.

Y por otro lado, conviene nombrar dos restricciones importantes de la gama baja y es que la pila solo dispone de dos niveles, lo que supone no poder encadenar más de dos subrutinas y además no admiten interrupciones.

2.2.2 GAMA MEDIA

En esta gama sus componentes añaden nuevas prestaciones a las que poseían los de la gama baja, haciéndoles más adecuados en las aplicaciones complejas. Admiten interrupciones, poseen comparadores de magnitudes analógicas, convertidores A/D, puertos serie y diversos temporizadores.

Algunos modelos disponen de una memoria de instrucciones del tipo OTP que resulta mucho más económica en la implementación de prototipos y pequeñas series. Otros en cambio, disponen de una memoria de instrucciones tipo EEPROM, que, al ser borrables eléctricamente, son mucho más fáciles de reprogramar que las EPROM, que tienen que ser sometidas a rayos ultravioleta durante un tiempo determinado para realizar dicha operación.

El PIC elegido para realización de las prácticas, el 16F877, pertenece a esta gama y en los sucesivos capítulos, las explicaciones se centrarán en él.

2.2.3 GAMA ALTA Y GAMA MEJORADA

En la actualidad, esta gama está formada por tres modelos. Los dispositivos PIC17C4X responden a microcontroladores de arquitectura abierta pudiéndose expansionar en el exterior al poder sacar los buses de datos, direcciones y control. Así se pueden configurar sistemas similares a los que utilizan los microprocesadores convencionales, siendo capaces de ampliar la configuración interna del PIC añadiendo nuevos dispositivos de memoria y de E/S externas. Esta facultad obliga a estos componentes a tener un elevado número de terminales comprendido entre 40 y 44. Admiten interrupciones, poseen puerto serie, varios temporizadores y mayores capacidades de memoria, que alcanza el 8 k palabras en la memoria de instrucciones y 454 bytes en la memoria de datos.

En 2003, Microchip lanzaba varios modelos de microcontroladores de gran potencia y velocidad, y se destinan a aplicaciones muy avanzadas. Con terminales que llegan desde las 28 hasta las 84, la memoria de código alcanza las 64k palabras y una frecuencia de 40 MHz

2.3 PIC16F87X: CARACTERÍSTICAS GENERALES

2.3.1 ORGANIZACIÓN DE LA MEMORIA

Los microcontroladores PIC 16F87X poseen dos bloques de memoria separados, la memoria de programa y la memoria de datos. Las posiciones de la memoria de datos son de 1 byte cada una y las de la memoria de programa de 14 bits. Para direccionar los datos hacen falta 7 bits para elegir posición dentro de un determinado banco y 2 bits más para seleccionar el banco, ya que pueden existir cuatro bancos de registros.

2.3.1.1 MEMORIA DE PROGRAMA

La memoria de instrucciones puede tener una capacidad mínima de 4k palabras de 14 bits hasta una máxima de 8k palabras de la misma longitud. Durante la fase de búsqueda, la dirección de la instrucción la proporciona el **contador de programa**, el cual normalmente se auto incrementa en la mayoría de las instrucciones, excepto en las de salto.

Al tener cada posición de memoria un tamaño de 14 bits, en las instrucciones del programa están implícitas las direcciones de los registros o memoria de datos. En una sola palabra se agrupa el código de la instrucción y su dirección. De los 14 bits, 7 se utilizan para indicar su dirección, lo que da un máximo direccionable en la memoria de datos de 128 bytes por banco de registros.

2.3.1.2 DIRECCIONAMIENTO Y PAGINADO

En los PIC's 16F87X, el contador de programa (CP) del microcontrolador tiene un tamaño de 13 bits, con lo que es posible direccionar un tamaño total de memoria de programa hasta un máximo de 8k x 14 bits organizada en páginas de un tamaño de 2k x 14 bits.

Para manejar microcontroladores de 4 u 8k bytes de memoria de programa, ha de recurrirse a la paginación, cada página tiene una capacidad de 2k, y éstas a su vez, se dividen en 8 subpáginas de 256 bytes.

El contador del programa, al estar formado por 13 bits, está compuesto por dos secciones tal y como se muestra en la figura 2.3. El byte bajo viene del registro de **PCL** que puede ser leído y escrito. Los bits superiores (PC<12:8>), están alojados en el registro **PCH**, sobre el que no se puede leer ni escribir, pero se puede acceder a él indirectamente a través del registro **PCLATH.**

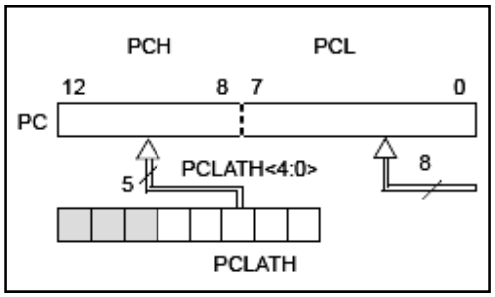


FIGURA 2.3

Contador de programa

2.3.2 BANCO DE REGISTROS Y MEMORIA DE DATOS

En las primeras posiciones de dichos bancos se encuentran los **registros de funciones especiales**, seguidos a continuación por los **registros de propósito general o memoria de datos.**

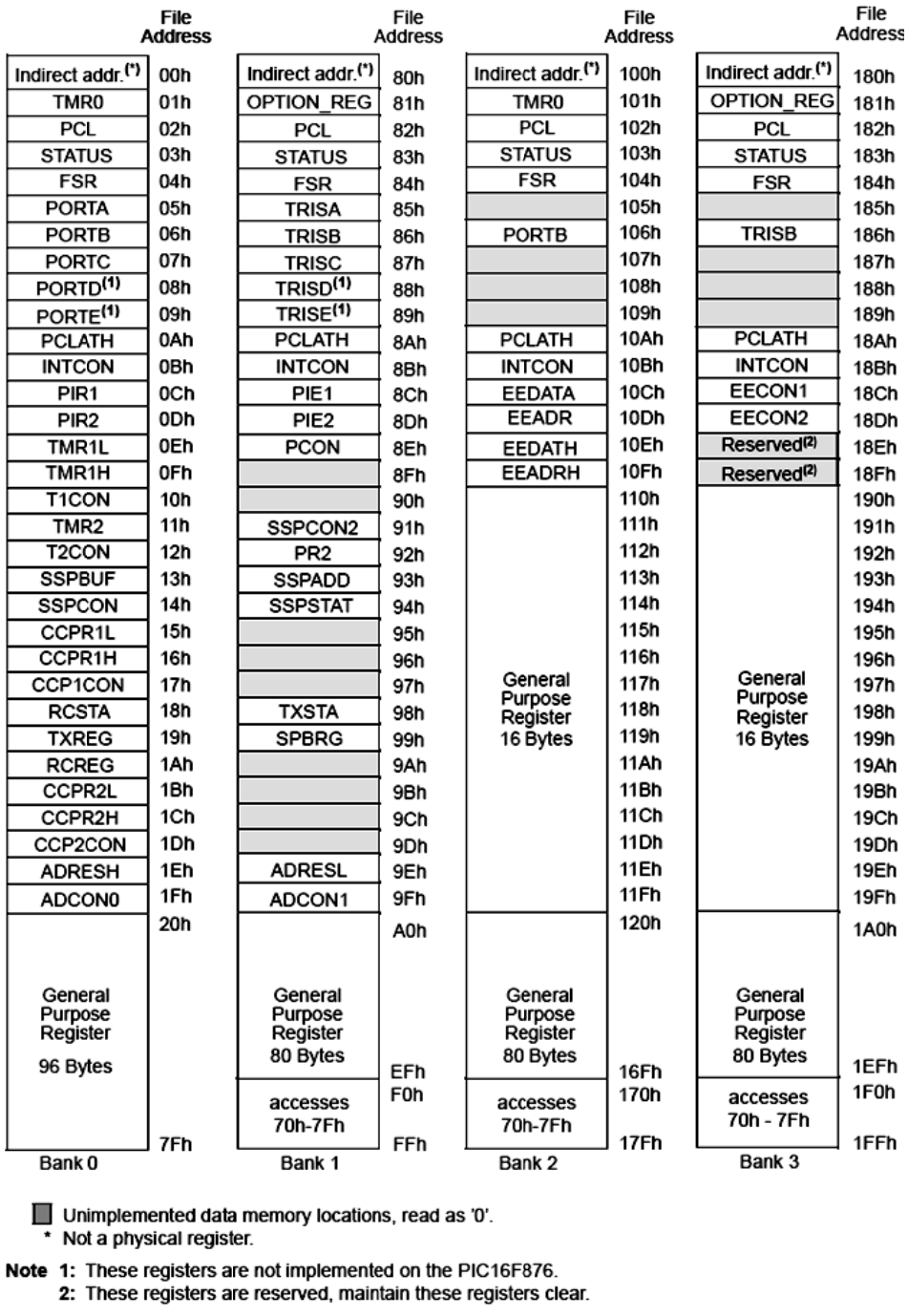


FIGURA 2.4

Banco de registros del PIC16F877

La **memoria de datos SRAM** en los PIC 16F877 es de 368 bytes. La memoria de trabajo o **acumulador (w)** de 1 byte en la SRAM es un registro de almacenamiento temporal. Este registro no puede ser accedido de forma directa, pero su contenido sí puede moverse a otro registro al que sí puede accederse directamente. Cada operación aritmética que se realiza utiliza el registro w.

La **memoria de datos EEPROM** de 256 bytes en el PIC16F877 puede almacenarse de forma indefinida cualquier dato que se desee retener cuando se apague la alimentación. Esta memoria es de 8 bits y no forma parte del espacio normal direccionable, y sólo es accesible en lectura y escritura a través de dos registros.

Los registros son de 8 bits y están formados por cuatro bancos como se puede ver en la figura 2.4.

2.3.2.1 DIRECCIONAMIENTO DE LOS DATOS

Para direccionar la memoria de datos se emplean tres modelos de direccionamiento, el Inmediato, el Directo y el Indirecto.

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.

El direccionamiento **Directo** consiste en codificar el nombre del o de los registros en cuestión directamente en la instrucción, pero primero hay que situarse en el banco adecuado. Si el dato no se halla en el banco 0, se emplean los bits 6 y 5 del registro FSR (Registro de Selección de Banco), que se denominan RAI y RA0, respectivamente.

En las instrucciones con direccionamiento **Indirecto** se usa como operando el registro INDF, que ocupa la posición 0 del banco 0. En tal caso, se accede a la posición que apunta el contenido del registro FSR ubicado en la posición 04 del área de datos. Sus 5 bits de menos peso apuntan la dirección del dato y los bits 6 y 5 seleccionan el banco. No tiene implementado el bit 7, que siempre se lee como 1.

El registro INDF no se halla implementado físicamente. Cada vez que se le referencia, se utiliza el contenido del registro FSR para direccionar al operando.

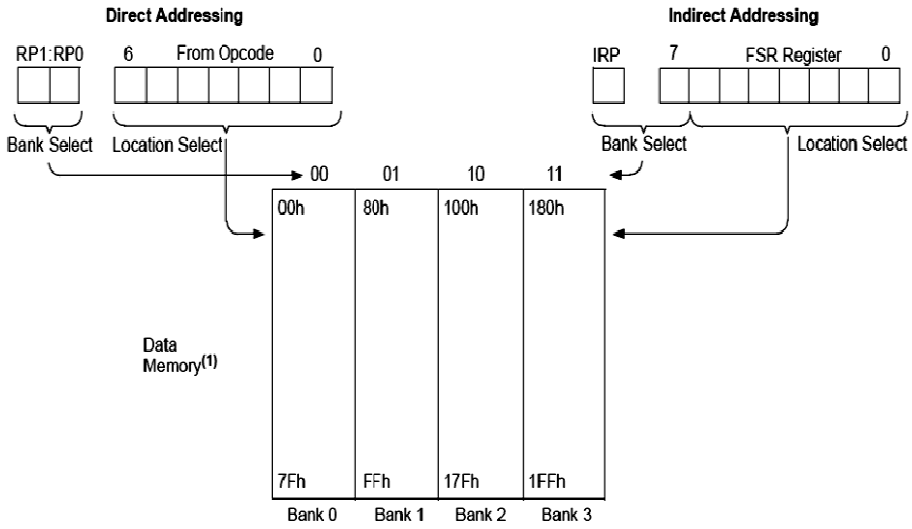


FIGURA 2.5

Direccionamiento directo e indirecto

2.3.3 REGISTROS CON FUNCIONES ESPECIALES

Los registros de funciones especiales o registros de control tienen como cometido gobernar el funcionamiento de los recursos del microcontrolador.

2.3.3.1 REGISTRO DE ESTADO (STATUS)

Ocupa las posiciones 03h, 83h, 130Bh y 183Bh de la memoria de datos del PIC. Es uno de los registros más importantes, cuyos bits controlan funciones vitales del microcontrolador. En la figura 2.6 se muestra la estructura y la misión de cada uno de sus bits.

El bit IRP hace la selección de bancos para el direccionamiento indirecto, tal y como se ha visto en el punto anterior y los bits RP1:RP0 lo hacen para el direccionamiento directo. Los bits TO y PD, no se pueden escribir, son banderas que indican la causa por la que se ha producido el reset del PIC y permiten actuar en consecuencia: Z es la bandera de cero, DC bandera de acarreo en el 4º bit de menor peso y C bandera de acarreo en el 8º bit.

STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**
Bit 7							bit 0

bit 7 IRP: Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

bit 6-5 RP1:RP0: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h-1FFh)
10 = Bank 2 (100h-17Fh)
01 = Bank 1 (80h-FFh)
00 = Bank 0 (00h-7Fh)
Each bank is 128 bytes.

bit 4 TO: Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 3 PD: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 Z: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 DC: Digit carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions) (for borrow, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 C: Carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For borrow, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.6

Registro de estado. Dirección 03H, 83H, 103H, 183H.

2.3.3.2 REGISTRO DE OPCIONES (OPTION_REG)

El registro de opciones se encuentra en las posiciones 81H y 181H, puede ser leído y escrito y contiene varios bits para la configuración de las asignaciones del pre divisor al TMR0 o al WDT, la interrupción externa, el TMR0 y las resistencias internas de polarización del puerto B.

OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
bit 7							bit 0

bit 7 **RBPU:** PORTB Pull-up Enable bit
1= PORTB pull-ups are disabled
0= PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1= Interrupt on rising edge of RB0/INT pin
0= Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1= Transition on RA4/T0CKI pin
0= Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1= Increment on high-to-low transition on RA4/T0CKI pin
0= Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1= Prescaler is assigned to the WDT
0= Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT value
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'

- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

Note: When using Low-Voltage ICSP Programming (LVP) and the pull-ups on PORTB are enabled, bit 3 in the TRISB register must be cleared to disable the pull-up on RB3 and ensure the proper operation of the device

FIGURA 2.7

Registro de opciones

(Dirección 81H, 181H)

RBPU es el bit de conexión de las resistencias de polarización del Puerto B; INTDEG selecciona el tipo de flanco para la interrupción por RB0/INT, según esté a 0 o a 1 será ascendente o descendente.

T0CS selecciona la fuente de reloj para el TMR0 y T0SE el tipo de flanco activo para el TMR0. PSA indicará la asignación del divisor de frecuencias al WDT o al TMR0.

Finalmente, los bits PS2:PS0 asignan la tasa del valor del divisor de frecuencias, y difiere dependiendo del predivisor que se haya asignado al TMR0 o al WDT.

Los detalles de todo lo comentado sobre este registro se encuentran en la figura 2.7.

2.3.3.3 REGISTRO DE INTERRUPCIONES (INTCON).

Puesto que los microcontroladores PIC de la gama media admiten interrupciones, requieren un registro encargado de su regulación. La funcionalidad de sus bits se entenderá mejor cuando se explique la operatividad de las interrupciones. En la figura 2.8 se ofrece la estructura y la misión de los bits del registro INTCON.

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	**PEIE**	**TMR0IE**	**INTE**	**RBIE**	**TMR0IF**	**INTF**	**RBIF**
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1= Enables all unmasked interrupts
0= Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1= Enables all unmasked peripheral interrupts
0= Disables all peripheral interrupts

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit
1= Enables the TMR0 interrupt
0= Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1= Enables the RB0/INT external interrupt
0= Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1= Enables the RB port change interrupt
0= Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit
1= TMR0 register has overflowed (must be cleared in software)
0= TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1= The RB0/INT external interrupt occurred (must be cleared in software)
0= The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1= At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0= None of the RB7:RB4 pins have changed state

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.8
Registro de interrupciones (direcciones 0bH, 8BH, 10BH, 18BH)

El bit GIE concede o cancela la activación global de las interrupciones. PEIE, TOIE, INTE y RBIE activan o desactivan las interrupciones en periféricos, TMR0, interrupciones externas y del puerto B respectivamente. El resto de bits son banderas de estado.

2.3.3.4 OTROS REGISTROS ESPECIALES

Los registros PIE1 y PIE2 contienen los bits que autorizan o prohíben las interrupciones producidas por los periféricos internos del PIC y que no están incluidos en el registro INTCON. Los registros PIR1 y PIR2 contienen una serie de bits que actúan como banderas de señalización de la causa que origina la interrupción, esté permitida o no. Para ver mayor detalle de estos registros ver las figuras 2.9, 2.10 y 2.11:

PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	**ADIE**	**RCIE**	**TXIE**	**SSPIE**	**CCP1IE**	**TMR2IE**	**TMR1IE**
bit 7							bit 0

bit 7	**PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit(1) 1= Enables the PSP read/write interrupt 0= Disables the PSP read/write interrupt Note 1: PSPIE is reserved on PIC16F873A/876A devices; always maintain this bit clear.
bit 6	**ADIE:** A/D Converter Interrupt Enable bit 1= Enables the A/D converter interrupt 0= Disables the A/D converter interrupt
bit 5	**RCIE:** USART Receive Interrupt Enable bit 1= Enables the USART receive interrupt 0= Disables the USART receive interrupt
bit 4	**TXIE:** USART Transmit Interrupt Enable bit 1= Enables the USART transmit interrupt 0= Disables the USART transmit interrupt
bit 3	**SSPIE:** Synchronous Serial Port Interrupt Enable bit 1= Enables the SSP interrupt 0= Disables the SSP interrupt
bit 2	**CCP1IE**: CCP1 Interrupt Enable bit 1= Enables the CCP1 interrupt 0= Disables the CCP1 interrupt
bit 1	**TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit 1= Enables the TMR2 to PR2 match interrupt 0= Disables the TMR2 to PR2 match interrupt
bit 0	**TMR1IE**: TMR1 Overflow Interrupt Enable bit 1= Enables the TMR1 overflow interrupt 0= Disables the TMR1 overflow interrupt

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

FIGURA 2.9

Registro PIE1 (dirección 8CH)

PIR1 REGISTER (ADDRESS 0Ch)

R/ W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	**ADIF**	**RCIF**	**TXIF**	**SSPIF**	**CCP1IF**	**TMR2IF**	**TMR1IF**
bit 7							bit 0

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1= A read or a write operation has taken place (must be cleared in software)
0= No read or write has occurred
Note 1: PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1= An A/D conversion completed
0= The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1= The USART receive buffer is full
0= The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1= The USART transmit buffer is empty
0= The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag bit
1 = The SSP interrupt condition has occurred and must be cleared in software before returning from the
Interrupt Service Routine. The conditions that will set this bit are:
• SPI – A transmission/reception has taken place.
• I^2C Slave – A transmission/reception has taken place.
• I^2C Master
- A transmission/reception has taken place.
- The initiated Start condition was completed by the SSP module.
- The initiated Stop condition was completed by the SSP module.
- The initiated Restart condition was completed by the SSP module.
- The initiated Acknowledge condition was completed by the SSP module.
- A Start condition occurred while the SSP module was Idle (multi-master system).
- A Stop condition occurred while the SSP module was Idle (multi-master system).
0= No SSP interrupt condition has occurred

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1= A TMR1 register capture occurred (must be cleared in software)
0= No TMR1 register capture occurred
Compare mode:
1= A TMR1 register compare match occurred (must be cleared in software)
0= No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1= TMR2 to PR2 match occurred (must be cleared in software)
0= No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1= TMR1 register overflowed (must be cleared in software)
0= TMR1 register did not overflow

Legend:

R = Readable bit	W = Writable bi	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

FIGURA 2.10

Registro PIR1 (dirección 0CH)

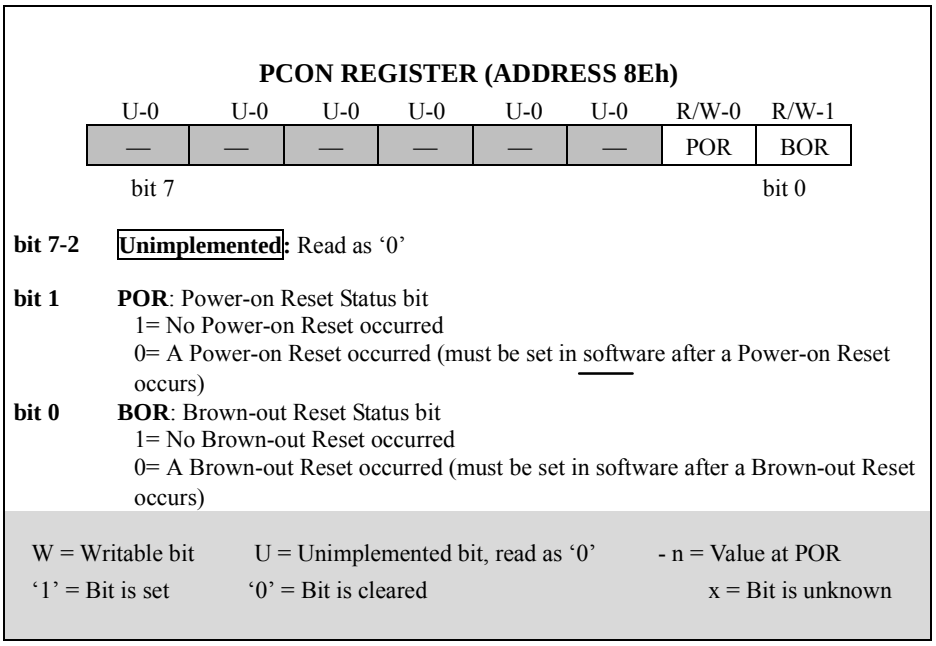


FIGURA 2.11

Registro PCON (dirección 8EH)

2.3.4 PALABRA DE CONFIGURACIÓN E IDENTIFICACIÓN

La palabra de configuración en los PIC de la gama media se compone de 14 bits que se escriben durante el proceso de grabación del dispositivo. Dichos bits ocupan la posición reservada de la memoria de programa 2007H.

Como se puede comprobar en la figura 2.12, el bit CP es el bit de protección de código, BODEN activa o desactiva el fallo en la alimentación, PWRTEN activa el temporizador, WDTEN activa el perro guardián, los bits FOSC1:FOSC0 seleccionan el tipo de oscilador.

También existen cuatro posiciones reservadas en la memoria de programa destinadas a contener las palabras de identificación (ID). En estas palabras solo se emplean los 4 bits de menos peso, en donde se almacena el número de serie, códigos de identificación, numeraciones secuénciales o aleatorias, etc. También se escribe su contenido en el proceso de grabación del dispositivo.

CONFIGURATION WORD (ADDRESS 2007h)(1)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BODEN	—	—	PWRTEN	WDTEN	FOSC1	FOSC0

bit13 bit0

bit 13 **CP:** Flash Program Memory Code Protection bit
1= Code protection off
0= All program memory code-protected

bit 12 **Unimplemented:** Read as '1'

bit 11 **DEBUG:** In-Circuit Debugger Mode bit
1= In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
0= In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger

bit 10-9 **WRT1:WRT0** Flash Program Memory Write Enable bits
For PIC16F876A/877A:
11= Write protection off; all program memory may be written to by EECON control
10= 0000h to 00FFh write-protected; 0100h to 1FFFh may be written to by EECON control
01= 0000h to 07FFh write-protected; 0800h to 1FFFh may be written to by EECON control
00= 0000h to 0FFFh write-protected; 1000h to 1FFFh may be written to by EECON control
For PIC16F873A/874A:
11= Write protection off; all program memory may be written to by EECON control
10= 0000h to 00FFh write-protected; 0100h to 0FFFh may be written to by EECON control
01= 0000h to 03FFh write-protected; 0400h to 0FFFh may be written to by EECON control
00= 0000h to 07FFh write-protected; 0800h to 0FFFh may be written to by EECON control

bit 8 **CPD:** Data EEPROM Memory Code Protection bit
1= Data EEPROM code protection off
0= Data EEPROM code-protected

bit 7 **LVP**: Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit
1= RB3/PGM pin has PGM function; low-voltage programming enabled
0= RB3 is digital I/O, HV on MCLR must be used for programming

bit 6 **BOREN**: Brown-out Reset Enable bit
1= BOR enabled
0= BOR disabled

bit 5-4 **Unimplemented:** Read as '1'

bit 3 **PWRTEN**: Power-up Timer Enable bit
1= PWRT disabled
0= PWRT enabled

bit 2 **WDTEN**: Watchdog Timer Enable bit
1= WDT enabled
0= WDT disabled

bit 1-0 **FOSC1:FOSC0**: Oscillator Selection bits
11= RC oscillator
10= HS oscillator
01= XT oscillator
00= LP oscillator

Legend:
R = Readable bit P = Programmable bit U = Unimplemented bit, read as '0'
- n = Value when device is unprogrammed u = Unchanged from programmed state

Note 1: The erased (unprogrammed) value of the Configuration Word is 3FFFh.

FIGURA 2.12

Palabra de configuración (dirección 2007H)(1)

En la 2.13 se muestran en una tabla las características más relevantes de los PIC 16F87X:

Características	16F873	16F874	16F876	16F877X
Frecuencia Máxima	DC-20Mhz	DC-20Mhz	DC-20Mhz	DC-20Mhz
Memoria de programa FLASH Palabra de 14 bits	4KB	4KB	8KB	8KB
Posiciones RAM de datos	192	192	368	368
Posiciones EEPROM de datos	128	128	256	256
Ports E/S	A, B y C	A, B, C, D y E	A, B y C	A, B, C, D y E
Nº de terminales	28	40	28	40
Interrupciones	13	14	13	14
Timers	3	3	3	3
Módulos CCP	2	2	2	2
Comunicaciones Serie	MSSP, USART	MSSP,USART	MSSP,USART	MSSP, USART
Comunicación Paralelo	-	PSP	-	PSP
Líneas de entrada en Convertidor A/D de 10 bits	5	8	5	8
Juego de Instrucciones	35 instrucciones	35 instrucciones	35 instrucciones	35 instrucciones
Longitud de la instrucción	14 bits	14 bits	14 bits	14 bits

Figura 2.13

Características relevantes de los PIC16F8TX

Todos estos elementos que se mencionan en la tabla serán analizados y explicados en los puntos sucesivos de este capítulo.

2.4 RECURSOS COMUNES E INTERRUPCIONES

2.4.1 DIAGRAMAS DE TERMINALES DEL PIC16F877A

En la figura 2.14 se muestra el diagrama de conexionado de un PIC 16F877A.

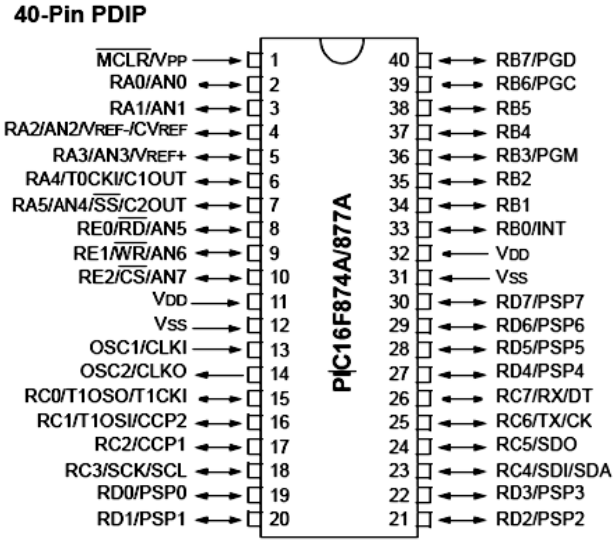


FIGURA 2.14

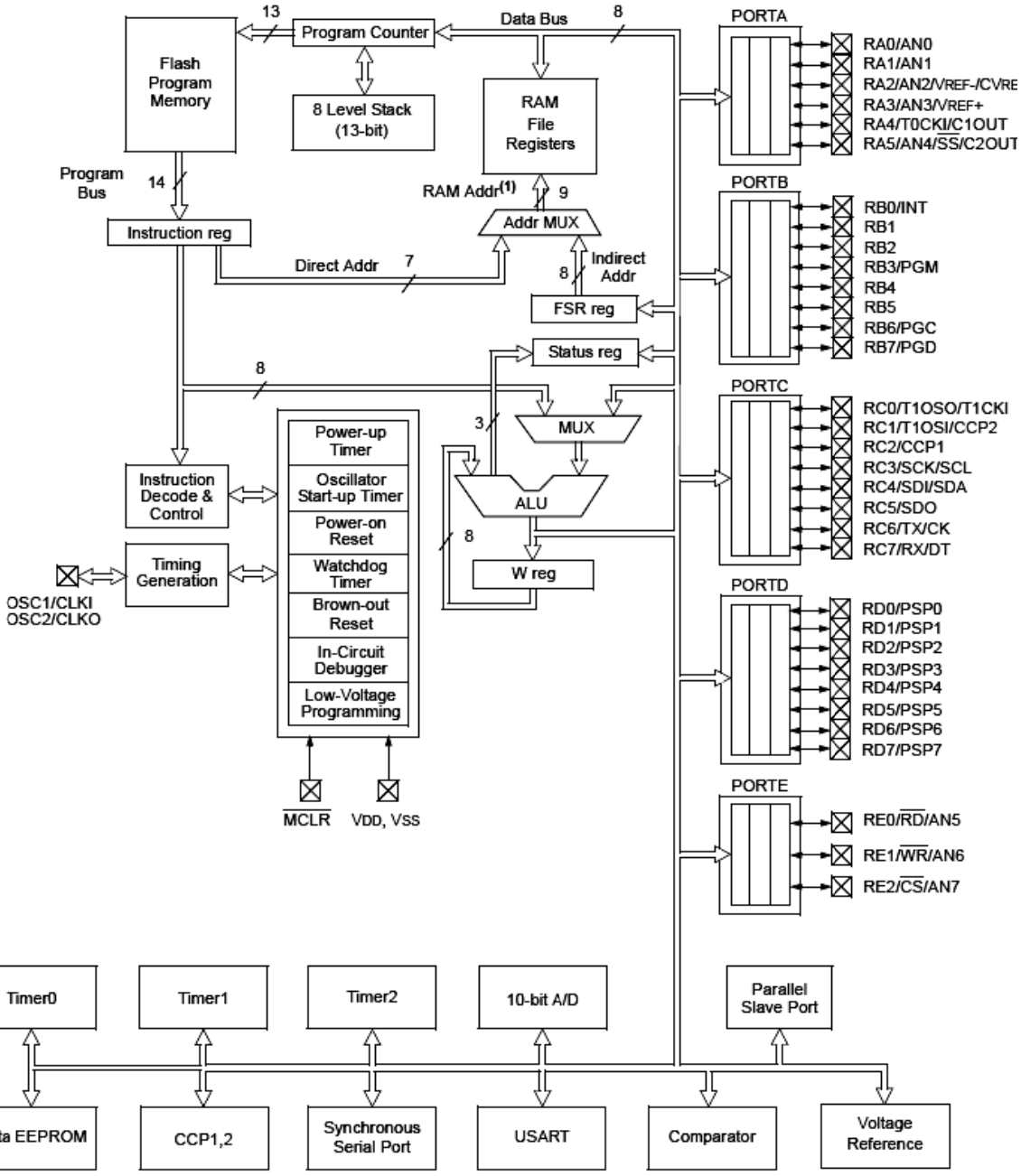
Diagrama de terminales del PIC16F877A

A continuación se nombran las funciones de todas las terminales:

- MCLR/V_{PP}: Reset externo. Por esta terminal se aplica también la tensión / V_{PP} usada en la grabación del programa.
- RA0/AN0: E/S digital o entrada analógica
- RA1/AN1: E/S digital o entrada analógica
- RA2/AN2/V_{REF}: E/S digital, entrada analógica o salida de la tensión de referencia
- RA3/AN3/ V_{REF}: E/S digital, analógica o entrada externa de V_{REF}
- RA4/TOCKI: E/S digital o entrada del reloj para TMR0
- RA5/AN4/SS: E/S digital, analógica o selección del puerto serie síncrono
- RB0/INT-RB7: E/S digitales del Puerto B. RB0/INT puede actuar como entrada de interrupción externa. RB4-RB7 pueden provocar interrupción cuando cambian de estado

- RE0/RD/AN5: E/S digital del Puerto E. Señal de lectura del Puerto paralelo esclavo. Entrada analógica.
- RE1/WR/AN6: E/S digital. Señal de escritura del Puerto paralelo esclavo. Entrada analógica.
- RE2/CS/AN7: E/S digital. Señal de activación del Puerto paralelo esclavo. Entrada analógica.
- V_{DD}: Entrada del positivo de la alimentación
- OSC1/CLKIN: Entrada al cristal cuarzo o reloj externo
- OSC2/CLKOUT: Salida del cristal. En modo R-C por este terminal sale ¼ F_{OSC1}
- RC0/T1OSO/T1CL1: E/S digital del Puerto C. Conexión del oscilador externo para el temporizador TMR1 o entrada de reloj para el TMR1
- RC1/T1OSI/CCP2: E/S digital del Puerto C. Conexión del oscilador externo para TMR1 o salida del modulo 2 de captura/comparación
- RC2/CCP1: E/S digital del Puerto C. Salida del modulo 1 de captura/comparación
- RC3/SCK/SCL: E/S digital. E/S de reloj para el Puerto serie síncrono (SSP) de los módulos SPI a I^2C
- RC4/SDI/SDA: E/S digital. Entrada de datos serie en el modo SPI. E/S de datos serie en modo I^2C
- RC5/SD0: E/S digital del Puerto C. Salida de datos serie en el modo SPI
- RC6/TX/CK: E/S digital. Transmisión serie asíncrona. Entrada de reloj para comunicación serie síncrona
- RC7/RX/DT: E/S digital. Recepción serie asíncrona. Línea de datos en la comunicación serie síncrona.
- RD0/PSP0-RD7/PSP7: E/S digitales del Puerto D. Este Puerto puede trabajar como Puerto paralelo esclavo para interconexión con un bus de datos de 8 bits de otro microprocesador

Una vez explicado el funcionamiento de cada terminal del PIC16F877, en la figura 2.15 se muestra su arquitectura interna, un diagrama de bloques donde se muestran los periféricos y las líneas de entrada y salida.



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

FIGURA 2.15 *Arquitectura interna del PIC16F877A*

2.4.2 RECURSOS COMUNES

En la gama media existen muchos recursos que son comunes a la mayoría de los modelos y que son conocidos por la gama baja.

2.4.2.1 OSCILADOR PRINCIPAL

Para la generación de los impulsos de reloj internos los PIC16F87X disponen de cuatro alternativas, debiendo el usuario seleccionar el más adecuado y programar adecuadamente los bits FOSC1 y FOSC0 de la palabra de configuración, que establecen la configuración entre las siguientes:

- - LP: Oscilador de cristal de cuarzo o resonador cerámico de baja potencia
- - XT: Cristal o resonador cerámico
- - HS: Oscilador de cristal o resonador de alta velocidad
- - RC: Oscilador formado por una red resistencia condensador

En la figura 2.16 se ofrece un circuito para las alternativas que usan cristal de cuarzo o resonador.

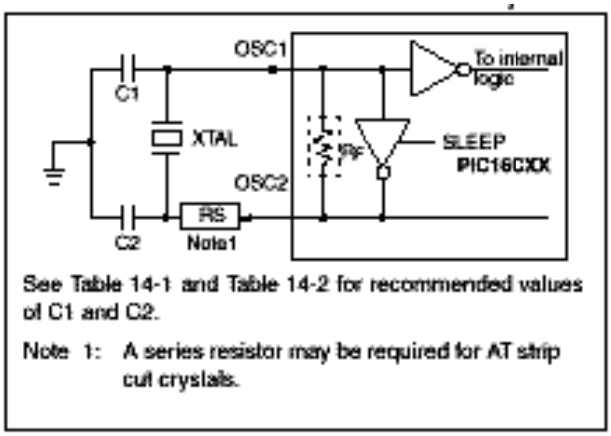


FIGURA 2.16

Diagrama para configuraciones LP, XT Y HS

Cuando no se requiere una gran precisión en la generación de impulsos de reloj, se puede emplear una red RC mucho más económica

como la que se muestra en la figura 2.17. En este caso, la frecuencia de oscilación viene determinada por los valores de resistencia y de los condensadores exteriores, así como de la temperatura de funcionamiento.

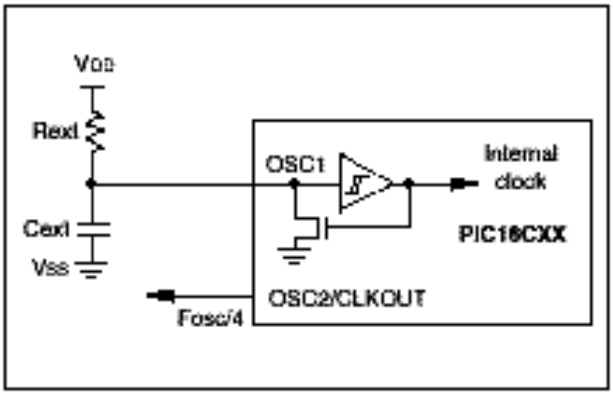


FIGURA 2.17

Diagrama para configuraciones RC.

2.4.2.2 PERRO GUARDIÁN (WDT)

El perro guardián (WDT) vigila que el programa no se "cicle" y dejen de ejecutarse las instrucciones secuénciales del mismo tal como lo ha previsto el diseñador. Para realizar esta labor de vigilancia, el perro guardián da un paseo por la CPU cada cierto tiempo y comprueba si el programa se ejecuta normalmente; en caso contrario, el perro provoca un reset, reinicializando todo el sistema.

Este temporizador está controlado por un oscilador interno independiente, con una temporización nominal de 18 ms, que puede aumentarse asignando el divisor de frecuencia al perro guardián, con el cual, trabajando en el rango mayor, puede alcanzar hasta 2,3 segundos.

Para evitar que se desborde el WDT y genere un reset, hay que recargar o refrescar su cuenta antes de que llegue el desbordamiento. Este refresco, que en realidad consiste en ponerle a 0 para iniciar la temporización, se consigue por software con las instrucciones CLRWDT y SLEEP.

2.4.2.3 TEMPORIZADOR TMR0

El TMR0 en los PIC16F87X es un contador ascendente de 8 bits, que puede funcionar con reloj interno o externo y ser sensible al flanco

ascendente o descendente. Se le puede asigna el divisor de frecuencia, y además posee la posibilidad de generar una interrupción cuando se desborda.

El TMR0 se comporta como un registro de propósito especial ubicado en la posición 1 del área de datos. Para trabajar con TMR0 se pueden utilizar las siguientes fórmulas en el caso que los impulsos de reloj provengan del oscilador interno con un periodo de Tosc.

$$\text{Tiempo} = 4 \cdot \text{Tosc} \cdot (\text{Valor cargado en TMR0}) \cdot (\text{Rango del Divisor})$$

$$\text{Valor a cargar en TMR0} = (\text{tiempo} / 4 \cdot \text{Tosc}) \cdot (\text{Rango del Divisor})$$

En la figura 2.18 se ofrece el diagrama de bloques del TMR0 y el preescaler que comparte con el WDT. 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 no se usa el Divisor de frecuencia, la entrada de la señal de reloj externa es la misma que la salida de dicho Divisor.

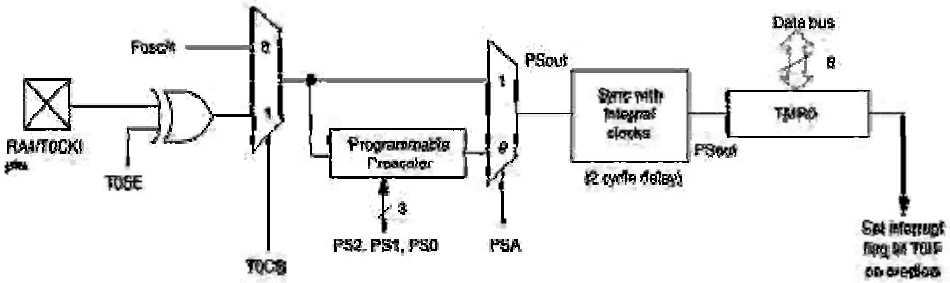


FIGURA 2.18

Diagrama a boques del TMR0.

2.4.2.4 RESET

El reset de los microcontroladores puede ser originado por las siguientes causas:

1. Conexión de la alimentación (POR: Power-On-Reset)
2. Activación de la terminal MCLR (Master Clear Reset) durante una operación normal.
3. Activación de MCLR en el estado de Reposo o SLEEP.
4. Desbordamiento del Perro guardián.

Como se aprecia en el esquema de la figura 2.19, cualquiera de estas posibilidades introduce un nivel bajo en la entrada S del flip-flop y pone en marcha un temporizador propio que, al cabo de 18 ms, origina un flanco ascendente en la salida Q que supone la generación del reset interno.

El bloque temporizador de la figura produce un retraso en la generación del reset para dar tiempo a que se estabilice la tensión V_{DD} de alimentación y la frecuencia del oscilador principal. Este temporizador está gobernado por un oscilador RC independiente.

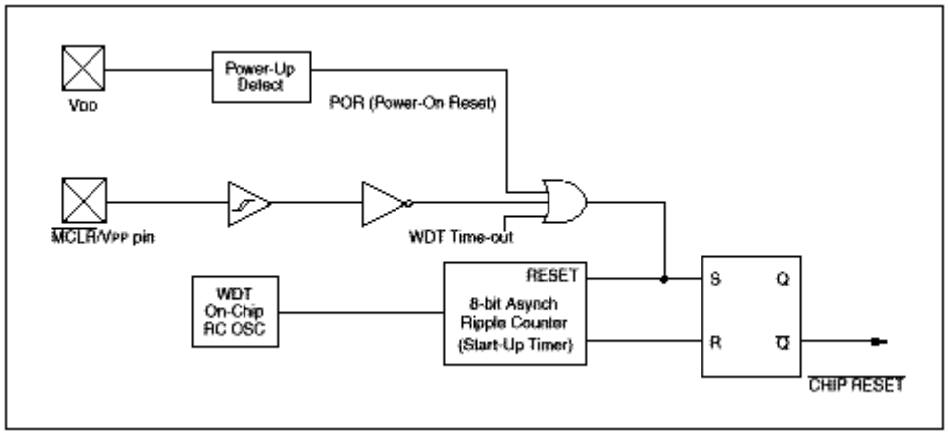


FIGURA 2.19

Diagrama a bloques del circuito de Reset

Los bits TO y PD del Registro de Estado toman el valor correspondiente según la causa que haya provocado el reset, tal como se muestra en la figura 2.20

TO	PD	Estado tras el RESET
0	0	WDT en el modo “SLEEP”
0	1	WDT en el modo normal
1	0	MCLR en el modo SLEEP
1	1	Conexión de la alimentación (POR)
u	u	MCLR en el modo normal

FIGURA 2.20

Valores para los bits TO y PD

2.4.2.5 MODO DE REPOSO (SLEEP)

En este modo especial de funcionamiento del microcontrolador se introduce cuando se ejecuta la instrucción SLEEP. Esta forma de trabajo se caracteriza por su bajo consumo. Las líneas de E/S digitales que se utilizaban mantienen su estado, las que no se empleaban reducen al mínimo su consumo, se detienen los temporizadores y tampoco opera el convertidor A/D.

Al entrar en el modo de reposo, si estaba funcionando el WDT se borra pero sigue trabajando. Para salir de ese estado ("despertar") y pasar a ejecutar la instrucción direccionada por PC+1 existen varias causas.

1. Activación externa de la terminal MCLR#
2. Desbordamiento del WDT que sigue trabajando en reposo
3. Generación de interrupción por la activación del terminal RB0/INT o por cambio de estado en los 4 terminales de más peso del puerto B
4. Interrupción originada por alguno de los nuevos periféricos como:
    a. Lectura o escritura en la puerta paralela (PSP)
    b. Interrupción del Timer1
    c. Interrupción del módulo CCP en modo captura
    d. Disparo especial del TMR1 funcionando en el modo asíncrono con reloj externo.
    e. Interrupción en el módulo de comunicación SSP (Start/Stop)
    f. Transmisión o recepción del MSSP en modo esclavo (SPI/I2C)
    g. Transmisión o recepción del USART
    h. Fin de la conversión A/D
    i. Fin de la operación de escritura sobre la EEPROM.

2.4.3 INTERRUPCIONES

Una interrupción consiste en una detención del programa en curso para realizar una determinada rutina que atienda la causa que ha provocado la interrupción. Es como una llamada a subrutina, que se origina por otra causa que por una instrucción del tipo CALL. Tras la terminación de la rutina de interrupción, se retorna al programa principal en el punto en que se abandono.

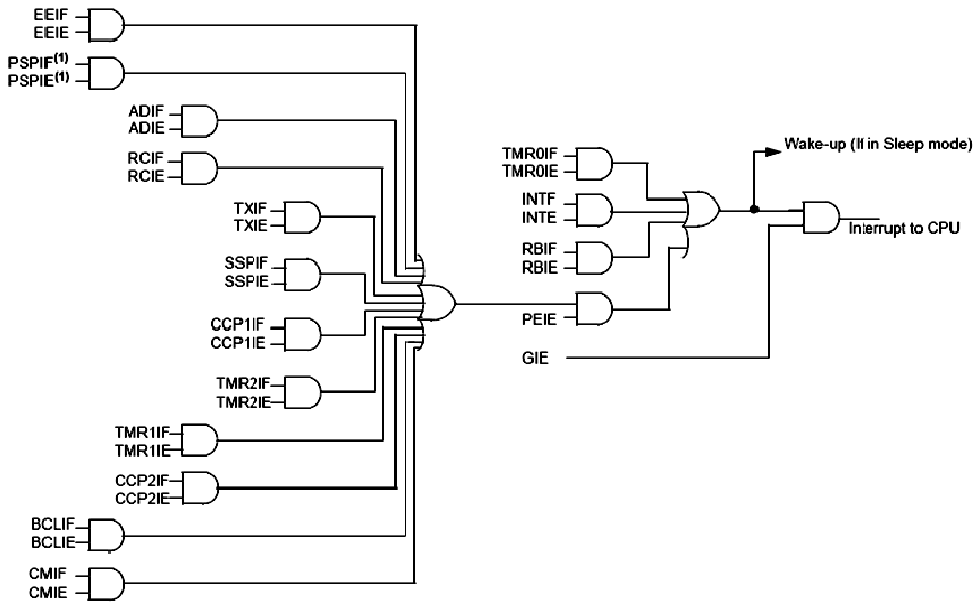
Las causas que originan una interrupción pueden ser externas, como la activación de una terminal con el nivel lógico apropiado, e internas, como las que pueden producirse al desbordarse un temporizador, como el TMR0.

En las aplicaciones industriales, las interrupciones son un producto muy potente para atender los acontecimientos físicos en tiempo real. Las interrupciones evitan que el CPU explore continuamente el nivel lógico de una terminal o el valor de un contador.

Los PIC16F87X tienen cuatro posibles fuentes de interrupción:

1. Interrupción externa provocada al activar la terminal RB0/INT
2. Desbordamiento del TMR0
3. Cambio de estado en las líneas del Puerto B
4. Cambio de estado en el comparador analógico

En la figura 2.21 se ofrece el esquema lógico que controla la generación de la interrupción, cuando aparece un nivel alto en su línea INT de salida.



Note 1: PSP interrupt is implemented only on PIC16F874A/877A devices.

FIGURA 2.21

Lógica de control para la generación de la interrupción

Cada causa de interrupción está controlada mediante dos líneas o señales. Una de ellas actúa como una bandera de señalización que indica si se ha producido o no el acontecimiento, mientras que la otra es el permiso o prohibición de la interrupción en sí.

El valor que se aplica a las señales de entrada del circuito de gobierno de las interrupciones proviene del que tengan los bits de los registros INTCON, PIR1 y PIE1.

El bit GIE de activación global del permiso de interrupción, situado en el registro INTCON, se borra automáticamente cuando se reconoce una interrupción para evitar que se produzca otra cuando se atiende a la inicial. Al retornar de la interrupción, el bit GIE se vuelve a activar.

2.5 PERIFÉRICOS

2.5.1 PUERTOS DE ENTRADA Y SALIDA

Las terminales de comunicación de los microcontroladores se agrupan en conjuntos llamados puertos porque dejan entrar y salir la información al procesador o terminales. Dichos puertos deben soportar las líneas que precisan los distintos periféricos que hay integrados en la cápsula. Cuantos más periféricos dispone el modelo, exige más líneas de comunicación y mayor numero de terminales, con más multiplexado de señales.

Los Puertos de E/S de los PIC16F87X disponen de versiones con 40 terminales y 5 puertos de E/S. Tienen Conversor A/D, 4 temporizadores, modulo CCP, Puerto Serie SSP, interfaz Serie SCI, Puerto Paralelo Esclavo y más capacidad en sus memorias.

Puerto A:

Consta de 6 terminales o líneas (RA0-RA5). Todas, menos RA4, pueden actuar como E/S digitales o como canales de entrada para el Conversor AD. La terminal RA4, además de E/S digital puede funcionar como entrada de reloj externo para el TMR0.

Puerto B:

Las 4 líneas de más peso del Puerto B (RB<3:0>) actúan como E/S digitales, según la programación del registro TRISB. Además pueden disponer de una carga pull-up interna si se programa la línea como entrada y el bit<7> (RBPO) del registro OPTION vale 0.

Las líneas RB<7:4> funcionan como las anteriores, pero además pueden provocar una interrupción si se programan como entradas y se produce el cambio de nivel lógico en alguna de ellas. En tal caso se activa el bit <0> (RBIF) de INTCON. La interrupción se anula al borrar el bit <3> (RBIE) de INTCON o al hacer una nueva lectura del Puerto B.

Puerto C:

Es un puerto bidireccional de 8 bits, cada terminal actúa como E/S digital, según la programación de TRISC. Además, también puede actuar como entrada o salida de diversos periféricos internos. Consultar el punto 2.4.1 donde se explican los diagramas de conexionado para ver la función de cada terminal en el puerto C.

Puerto D:

Cada terminal puede configurarse como E/S digital, según la programación del registro TRISD. También puede funcionar como Puerto Paralelo Esclavo para soportar la interconexión directa con el bus de datos de 8 bits de otro microprocesador. Para funcionar en este modo hay que poner a 1 el bit<4> (PSMODE) de TRISE. En tal caso, las líneas RE<2:0> del Puerto E pasan a soportar las señales de control CS, WR y RD, entre el Puerto D y el bus del microprocesador. Cada vez que el microprocesador realiza un ciclo de lectura o escritura sobre el Puerto D el bit <7> (PSPIF) del registro PIR1 se pone a 1.

Puerto E:

Este puerto que solo dispone de tres terminales esta disponible en el modelo, la nomenclatura de las líneas es:

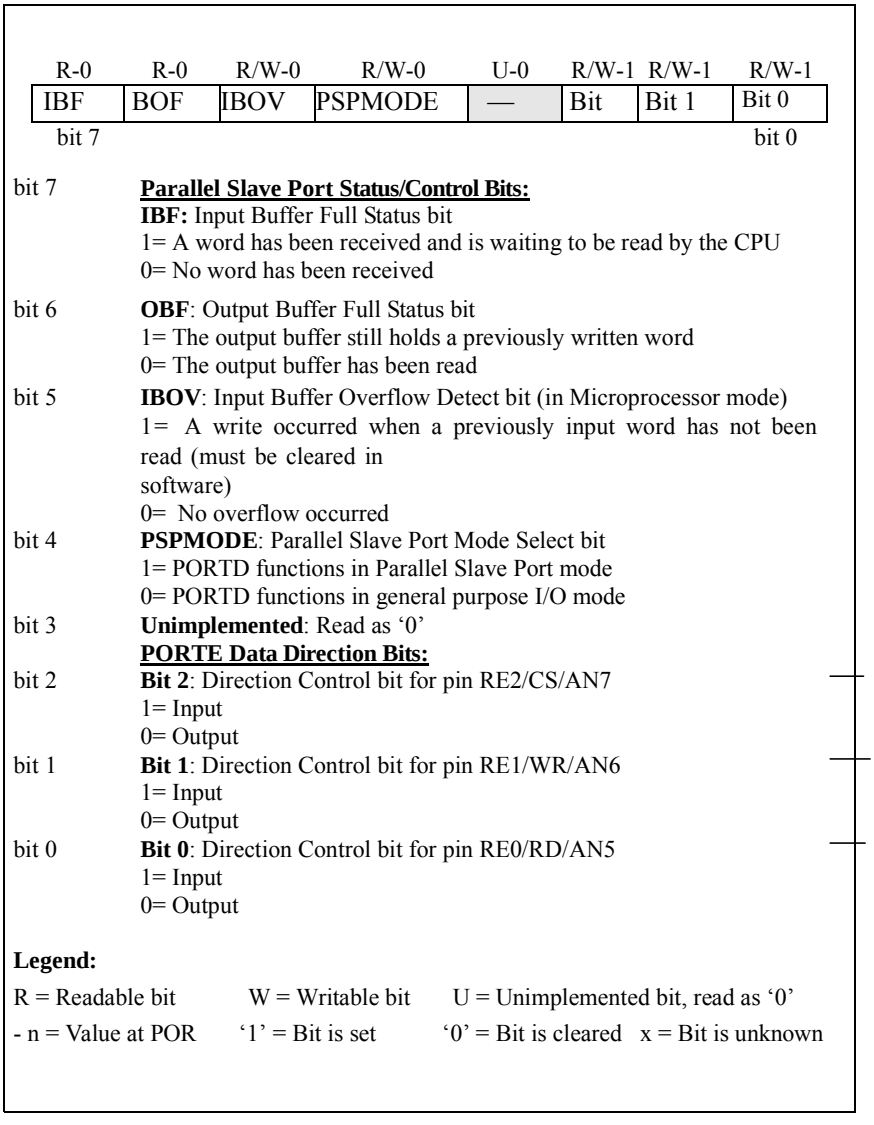
RE0/RD/AN5

RE1/WR/AN6

RE2/CS/AN7

Las 3 terminales pueden funcionar como E/S digitales, según la programación de los tres bits de menos peso del registro TRISE. También pueden actuar como señales de control (RD, WR y CS) para el flujo de datos entre un microprocesador y el Puerto D, cuando está programado en el modo Esclavo. Deben programarse como entradas. Finalmente, también pueden realizar estas 3 terminales la función de canales de entradas analógicas para el Conversor A/D, según la programación del registro ADCON1.

El registro TRISE, que se muestra en la figura 2.22 sirve para configurar las líneas de E/S digitales del Puerto E, o bien, para actuar como Registro de Estado cuando el Puerto D funciona como Puerto Paralelo Esclavo.



R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	BOF	IBOV	PSPMODE	—	Bit	Bit 1	Bit 0
bit 7							bit 0

bit 7 **Parallel Slave Port Status/Control Bits:**
IBF: Input Buffer Full Status bit
1= A word has been received and is waiting to be read by the CPU
0= No word has been received

bit 6 **OBF**: Output Buffer Full Status bit
1= The output buffer still holds a previously written word
0= The output buffer has been read

bit 5 **IBOV**: Input Buffer Overflow Detect bit (in Microprocessor mode)
1= A write occurred when a previously input word has not been read (must be cleared in software)
0= No overflow occurred

bit 4 **PSPMODE**: Parallel Slave Port Mode Select bit
1= PORTD functions in Parallel Slave Port mode
0= PORTD functions in general purpose I/O mode

bit 3 **Unimplemented**: Read as '0'

PORTE Data Direction Bits:

bit 2 **Bit 2**: Direction Control bit for pin RE2/CS/AN7
1= Input
0= Output

bit 1 **Bit 1**: Direction Control bit for pin RE1/WR/AN6
1= Input
0= Output

bit 0 **Bit 0**: Direction Control bit for pin RE0/RD/AN5
1= Input
0= Output

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.22

Registro TRISE
Dirección 89H

2.5.2 CONVERTIDOR ANALÓGICO DIGITAL

El módulo de conversión **Analógico/Digital** dispone de cinco entradas para los dispositivos de 28 terminales y ocho para los otros dispositivos de la familia.

Es un convertidor analógico a digital de 8 bits, como el que se muestra en la figura 23, con una tensión de referencia que puede ser interna (V_{DD}) o externa (entra por la terminal AN3/V_{ref}). En cada momento la conversión solo se realiza con la entrada de uno de sus canales, depositando el resultado de la misma en el registro ADRES y activándose la bandera ADIF, que provoca una interrupción si el bit de permiso correspondiente esta activado. Además, al terminar la conversión el bit GO/DONE se pone a 0.

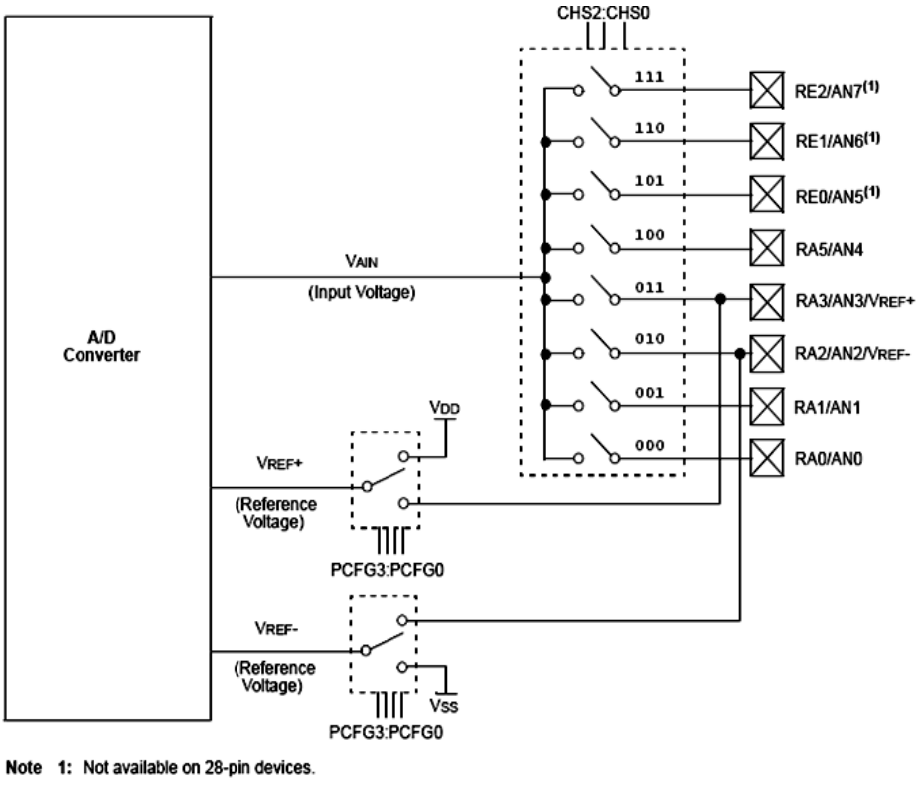


FIGURA 2.23

Conversor AD con 8 canales para entradas analógicas

ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	**ADCS0**	**CHS2**	**CHS1**	**CHS0**	**GO/DONE**	**—**	**ADON**
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	**FOSC/2**
0	**01**	**FOSC/8**
0	**10**	**FOSC/32**
0	**11**	**FRC (clock derived from the internal A/D RC oscillator)**
1	**00**	**FOSC/4**
1	**01**	**FOSC/16**
1	**10**	**FOSC/64**
1	**11**	**FRC (clock derived from the internal A/D RC oscillator)**

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000= Channel 0 (AN0)
001= Channel 1 (AN1)
010= Channel 2 (AN2)
011= Channel 3 (AN3)
100= Channel 4 (AN4)
101= Channel 5 (AN5)
110= Channel 6 (AN6)
111= Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/DONE:** A/D Conversion Status bit
When ADON = 1:
1= A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0= A/D conversion not in progress bit 1
Unimplemented: Read as '0'

bit 0 **ADON:** A/D On bit
1= A/D converter module is powered up
0= A/D converter module is shut-off and consumes no operating current

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

FIGURA 2.24

Registro ADCON0.

(Dirección 1FH)

El módulo de A/D tiene cuatro registros que son: ADRESH, ADRESL, ADCON0 y ADCON1.

Para gobernar el funcionamiento del CAD se utilizan el ADCON0 y el ADCON1. El primero, que se muestra en la figura 2.24, selecciona el canal a convertir con los bits CHS <2:0>, activa al conversor y contiene la bandera que avisa del fin de la conversión (ADIF) y el bit GO/DONE.

ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1= Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0= Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	**ADCON0 <ADCS1:ADCS0>**	**Clock Conversion**
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	**AN7**	**AN6**	**AN5**	**AN4**	**AN3**	**AN2**	**AN1**	**AN0**	**VREF+**	**VREF-**	**C/R**
0000	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**VDD**	**VSS**	**8/0**
0001	**A**	**A**	**A**	**A**	**VREF+**	**A**	**A**	**A**	**AN3**	**VSS**	**7/1**
0010	**D**	**D**	**D**	**A**	**A**	**A**	**A**	**A**	**VDD**	**VSS**	**5/0**
0011	**D**	**D**	**D**	**A**	**VREF+**	**A**	**A**	**A**	**AN3**	**VSS**	**4/1**
0100	**D**	**D**	**D**	**D**	**A**	**D**	**A**	**A**	**VDD**	**VSS**	**3/0**
0101	**D**	**D**	**D**	**D**	**VREF+**	**D**	**A**	**A**	**AN3**	**VSS**	**2/1**
011x	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**—**	**—**	**0/0**
1000	**A**	**A**	**A**	**A**	**VREF+**	**VREF-**	**A**	**A**	**AN3**	**AN2**	**6/2**
1001	**D**	**D**	**A**	**A**	**A**	**A**	**A**	**A**	**VDD**	**VSS**	**6/0**
1010	**D**	**D**	**A**	**A**	**VREF+**	**A**	**A**	**A**	**AN3**	**VSS**	**5/1**
1011	**D**	**D**	**A**	**A**	**VREF+**	**VREF-**	**A**	**A**	**AN3**	**AN2**	**4/2**
1100	**D**	**D**	**D**	**A**	**VREF+**	**VREF-**	**A**	**A**	**AN3**	**AN2**	**3/2**
1101	**D**	**D**	**D**	**D**	**VREF+**	**VREF-**	**A**	**A**	**AN3**	**AN2**	**2/2**
1110	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**A**	**VDD**	**VSS**	**1/0**
1111	**D**	**D**	**D**	**D**	**VREF+**	**VREF-**	**D**	**A**	**AN3**	**AN2**	**1/2**

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Legend:
R = Readable bit W = Writable b U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

Note: On any device Reset, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

FIGURA 2.25

Registro ADCON1

El registro ADCON1 establece las entradas que son digitales y analógicas, así como el tipo de tensión de referencia (interna o externa).

ADFM selecciona el formato del resultado de la conversión, con justificación izquierda o derecha. PCFG3:PCFG0 son los bits de configuración de los canales de entrada del conversor. Se utilizan para configurar las patillas como E/S digital o como entrada analógica de acuerdo con la tabla de la figura 2.25.

Finalmente, se describen de forma resumida los pasos para realizar una conversión en el CA/D:

1. Se configura correctamente el CA/D programando los bits de los registros de control.
2. Se autoriza o prohíbe la generación de interrupción al finalizar la conversión, cargando los bits del PIE1.
3. Para iniciar la conversión se pone el bit GO/DONE = 1. Hay que tener en cuenta el tiempo que durara la conversión.
4. Se detecta el final de la conversión bien porque se genera la interrupción, o bien porque se explora cuando el bit GO/DONE = 0.
5. Se lee el resultado de la conversión en el registro ADRES.

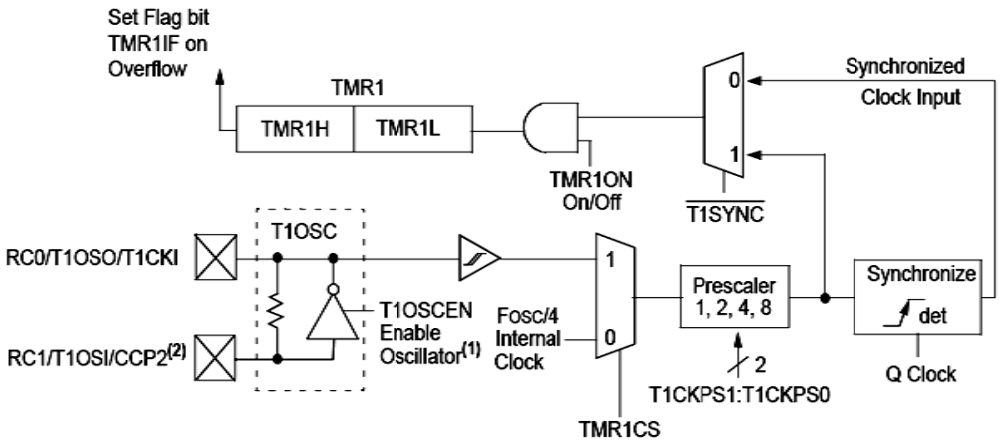
Los registros ADRESH:ADRESL contienen los 10 bits resultados de la conversión A/D. Cuando se completa la conversión A/D, el resultado se guarda en los registros y se pone a cero el bit GO/DONE y el flag de fin de conversión ADIF (PIR1<6>) se pone a 1. Después de que el convertidor A/D se ha configurado como se quiere, la selección del canal debe realizarse antes de hacer la adquisición. Los canales de entrada analógica deben tener los correspondientes bits del registro TRIS seleccionados como entradas.

2.5.3 TEMPORIZADOR TMR1

El TMR1 es un Temporizador/Contador ascendente de 16 bits, por lo que esta implementado mediante dos registros específicos TMR1H y TMR1L, que contienen el valor del conteo en cada momento. El valor de registro TMR1H-TMR1L evoluciona desde 0000h hasta FF FFH, en cuyo instante activa la bandera TMR1IF y vuelve a 0000h. Como fuente de los impulsos de reloj existen tres alternativas:

1. Generación interna (4 T_{OSC})
2. Generación mediante un oscilador externo controlado por cristal que se conecta a las terminales RC0/T1OSO/T1CKI y RC1/T1OSI/CCP2. El oscilador se activa poniendo a 1 el bit T1OSCEN del registro T1CON. El bit TMR1CS del registro T1CON selecciona entre el reloj interno o externo.
3. Trabaja en modo contador de eventos, cuando los impulsos externos a contar se aplican a la terminal RC0/T1OSO/T1CKI.

La fuente de los impulsos de reloj aplica a un Divisor de Frecuencias que los divide por 1, 2, 4 u 8, según el valor de los bits <1:0> (TICKPS) del registro T1CON. El reloj externo puede estar sincronizado o no con el interno, según el bit T1SYNC de T1CON. El interno siempre es síncrono. T1OSCEN habilita el oscilador, #TlSYNC es el bit de control de sincronización de la señal de entrada, TMR1CS selecciona la fuente de reloj y TMR1ON activa el temporizador TMR1. Ver Figuras 2.26 y 2.27.



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

FIGURA 2.26

Esquema del TMR1

El periodo en T1CKI es preciso que tenga una duración mínima de 4.T_{OSC}.

En el modo de Reposo cuando funciona en modo síncrono, el TMR1 deja de incrementarse pues se desconecta el circuito de sincronismo. En forma asíncrona el TMR1 sigue contando durante el modo de reposo, por eso se puede emplear como un reloj de tiempo real y para sacar del modo de Reposo al sistema. También en modo

asíncrono se puede usar como base de tiempos en operaciones de Captura y Comparación. El modulo CCP pone a 0 el TMR1 cuando se produce una Captura o una Comparación.

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6	**Unimplemented:** Read as '0'
bit 5-4	**T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits 11= 1:8 prescale value 10= 1:4 prescale value 01= 1:2 prescale value 00= 1:1 prescale value
bit 3	**T1OSCEN**: Timer1 Oscillator Enable Control bit 1= Oscillator is enabled 0= Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)
bit 2	**T1SYNC**: Timer1 External Clock Input Synchronization Control bit When TMR1CS = 1: 1= Do not synchronize external clock input 0= Synchronize external clock input When TMR1CS = 0: This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.
bit 1	**TMR1CS**: Timer1 Clock Source Select bit 1= External clock from pin RC0/T1OSO/T1CKI (on the rising edge) 0= Internal clock (FOSC/4)
bit 0	**TMR1ON**: Timer1 On bit 1= Enables Timer1 0= Stops Timer1
Legend: R = Readable bit W = Writable bit U = Unimplemented bit, read as '0' - n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown	

FIGURA 2.27

Registro T1CON.

(dirección 10H)

2.5.4 TEMPORIZADOR TMR2

El TMR2 solo está incorporado en unos pocos modelos de la gama media porque se trata de un temporizador de 8 bits diseñado para usarse conjuntamente con el circuito de Modulación de Anchura de Impulsos (PWM).

Se incrementa al ritmo de los impulsos que se le aplican ($4.T_{OSC}$), que pueden ser divididos por 1, por 4 o por 16 mediante un Predivisor. Cuando el valor de TMR2 coincide con el del PR2 (Registro de Periodo) se genera un impulso en la salida EQ y TMR2 pasa a 00H. PR2 es un registro específico de lectura y escritura que cuando hay un Reset se carga con el valor FFH. Los impulsos producidos por EQ se aplican a

un Post-divisor que puede dividirlos hasta 1:16, activando su salida la bandera TMR2IF. El registro T2CON regula los principales parámetros de este temporizador. Figuras 2.27 y 2.28.

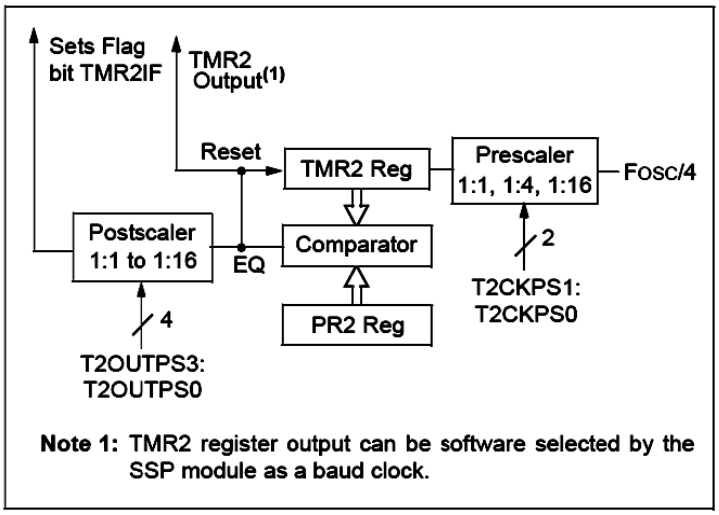


FIGURA 2.28

Diagrama a bloques del TMR2

El reset borra al Predivisor y al Post-divisor. También lo hace al TMR2 cuando se ha generado como consecuencia del WDT, POR o MCLR. Cada vez que se escribe sobre el TMR2 o el T2CON se borran el Predivisor y el Post-divisor. Ver figura 2.28.

La salida EQ se puede utilizar como señal de reloj para el modulo de interfaz serie SSP.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	**TOUTPS3**	**TOUTPS2**	**TOUTPS1**	**TOUTPS0**	**TMR2ON**	**T2CKPS1**	**T2CKPS0**
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6-3	**TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits 0000= 1:1 postscale 0001= 1:2 postscale 0010= 1:3 postscale • • • 1111= 1:16 postscale
bit 2	**TMR2ON**: Timer2 On bit 1= Timer2 is on 0= Timer2 is off
bit 1-0	**T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits 00= Prescaler is 1 01= Prescaler is 4 1x= Prescaler is 16

FIGURA 2.29

Registro T2CON

(Dirección 12H)

2.5.5 MÓDULOS DE CAPTURA/COMPARACIÓN / MODULACIÓN DE ANCHURA DE PULSOS CCP

Los microcontroladores PIC16F87X disponen de dos de estos módulos, llamados CCP1 y CCP2, que se controlan con los registros CCP1CON y CCP2CON, respectivamente. Realizan tres funciones:

1. Capturan información de 16 bits procedente del TMR1.
2. Comparan el valor de un registro con el del TMR1.
3. Modulan o controlan el intervalo de tiempo en el que bascula de 1 a 0 una terminal del microcontrolador.

Los módulos CCP1 y CCP2 son idénticos, excepto en ciertas funciones especiales de disparo. Ambos constan de dos registros CCPRxH y CCPRxL.

2.5.5.1 MODO DE CAPTURA

En este modo el registro CCPRxH/L de 16 bits captura el valor contenido en el TMR1, siempre que ocurra uno de los siguientes acontecimientos en la terminal Rcy/CCPx del Puerto C, que previamente ha tenido que configurarse como entrada en el registro TRISC:

a. Un flanco ascendente
b. Un flanco descendente
c. Cada 4 flancos ascendentes
d. Cada 16 flancos ascendentes

Al realizarse una captura se activa la bandera CCPxIF del registro PIR1 o PIR2 y si se programa adecuadamente el bit de permiso en PIE1 o PIE2, se genera una interrupción. Ya se puede leer el valor del registro CCPRx.

Cuando se produce una captura y no se ha leído el contenido de CCPRx se borra y pasa a contener el nuevo. Si se van a modificar las condiciones en las que se va a efectuar la Captura conviene detener o desactivar el modulo CCP para que no se produzcan falsas interrupciones durante la operación.

Una aplicación del modo de Captura puede ser la medición de los intervalos de tiempo que existen entre los impulsos que llegan a la terminal Rcy/CCPx configurada como entrada. En este modo de trabajo TMR1 debe usarse como entrada de reloj externo sincronizado.

2.5.5.2 MODO DE COMPARACIÓN

Cuando un modulo CCP trabaja de esta manera el contenido del registro CCPRxH/L se compara continuamente con el del TMR1, que debe trabajar en modo síncrono. Cuando coinciden ambos valores la terminal Rcy/CCPx, que previamente se habrá configurado como salida, puede bascular a 1, a 0, o bien no variar, pero se activara el señalizador CCPxIF. En tal caso, si el bit de permiso esta activado se provoca una interrupción.

Si se selecciona la función especial de disparo, el CCP1 pone a 0 al TMR1 con lo que CCPR1 funciona como un registro de periodos capaz de provocar interrupciones periódicamente. También el CCP2 pone a 0 el TMR1 e inicia una conversión en el Convertidor A/D, con lo que se pueden realizar periódicamente conversiones de analógico-digitales, sin el control del programa de instrucciones.

2.5.5.3 MODULO DE ANCHURA DE PULSOS (PWM)

En este modo la terminal Rcy/CCPx, que se ha programado como salida, bascula entre 0 y 1 a intervalos variables de tiempo. Cuando el valor de registro PR2 coincide con los 8 bits de más peso de TMR2 la terminal mencionada pasa a 1 y TMR2 toma el valor 00 y reanuda la cuenta. El contenido de CCPRxL pasa a CCPRxH y se compara con TMR2.

Cuando ambos coinciden la terminal Rcy/CCPx pasa a 0 y se repite la secuencia. Variando el valor de PR2 y CCPRxL se varia el intervalo de tiempo que la terminal esta a 1 y esta a 0, respectivamente.

Tiempo a 1 = (PR2 + 1) . 4 . TOSC . PREDIVISOR
Tiempo a 0 = DCI . TOSC . PREDIVISOR

DCI representa el valor de los 8 bits del registro CCPRxL concatenado con los bits <5:4> de CCPxCON. Los 8 bits de TMR2 se concatenan con dos bits Q del reloj interno haciendo que cuente cada T_{OSC}, en vez de cada 4. T_{OSC}. Todo ello sucede cuando se opera con una resolución de 10 bits, porque si se usa una de 8 bits, los dos bits de menos peso se ponen a 0.

La figura 2.30 representa cualquiera de los registros CCPICON o CCP2CON. Los bits 4 y 5 funcionan solamente en el modo PWM y son los dos bits de menos peso cuando se trabaja con una resolución de 10 bits. CCPxM3:CCPxM0 seleccionan el modo de trabajo.

CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
	bit 7						bit 0

bit 7-6	**Unimplemented:** Read as '0'
bit 5-4	**CCPxX:CCPxY**: PWM Least Significant bits Capture mode: Unused. Compare mode: Unused. PWM mode: These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.
bit 3-0	**CCPxM3:CCPxM0**: CCPx Mode Select bits 0000= Capture/Compare/PWM disabled (resets CCPx module) 0100= Capture mode, every falling edge 0101= Capture mode, every rising edge 0110= Capture mode, every 4th rising edge 0111= Capture mode, every 16th rising edge 1000= Compare mode, set output on match (CCPxIF bit is set) 1001= Compare mode, clear output on match (CCPxIF bit is set) 1010= Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected) 1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled) 11xx= PWM mode

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.30
Registros CPP1CON y CCP2CON. (Dirección 17H, 1DH)

2.5.6 PUERTO SERIE SÍNCRONO (SSP)

Se trata de un periférico diseñado para soportar una interfaz serie síncrono que resulta muy eficiente para la comunicación del microcontrolador con dispositivos tales como displays, EEPROM, ADC, etc. Tiene dos modos de trabajo:

1. Interfaz Serie de Periféricos (SPI)
2. Interfaz Inter-Circuitos (I^2C)

2.5.6.1 MODO SPI

Sirve para conectar varios microcontroladores de la misma o diferente familia, bajo el formato "maestro-esclavo", siempre que dispongan de un interfaz compatible.

En este modo se pueden emplear 3 o 4 señales de control: Salida de Datos (SDO), Entrada de Datos (SDI), Reloj (SCK) y Selección de Esclavo (SS). Dichas señales se corresponden con las terminales RC5, RC4, RC3 y RA5 respectivamente. Cada una de las señales debe programarse como entrada o salida según su condición, utilizando los bits de los registros TRIS. Cualquier función del modo SPI queda anulada poniendo con el valor opuesto a su condición el bit correspondiente de TRIS. Por ejemplo, si solo se quiere recibir datos, se programa la terminal que soporta a SOD como entrada y así se anula su función.

Con el registro de control SSPCON se eligen las diferentes opciones de trabajo: Modo Master (SCK es salida), Modo Esclavo (SCK es entrada), tipo de flanco de reloj, velocidad de SCK en Modo Master, etc.

Cuando se recibe un dato útil, este se introduce en serie en SSPSR y pasa a SSPBUF en paralelo. El dato a transmitirse deposita en SSPBUF y de aquí pasa a SSPSR. Se puede recibir y transmitir datos simultáneamente. SSPSR es un registro de desplazamiento que funciona serie/paralelo/serie.

Cuando se acaba de transmitir o recibir un dato completo se activa el bit BF (Buffer Lleno) del registro SSPSTAT. También lo hace la bandera SSPIF y si el bit de permiso esta activado se genera una interrupción.

Cuando se recibe un dato durante una transmisión se ignora y se activa el bit WCOL del registro SSPCON que indica que se ha producido una "colisión". (Para ver los detalles de funcionamiento de los registro SSPSTAT y SSPCON consultar el DATA SHEET del 16F87X que se puede encontrar en la página web de microchip.)

En el caso que se reciba un nuevo dato en SSPSBUF sin haber leído el anterior se genera un error de desbordamiento. Figura 2.31

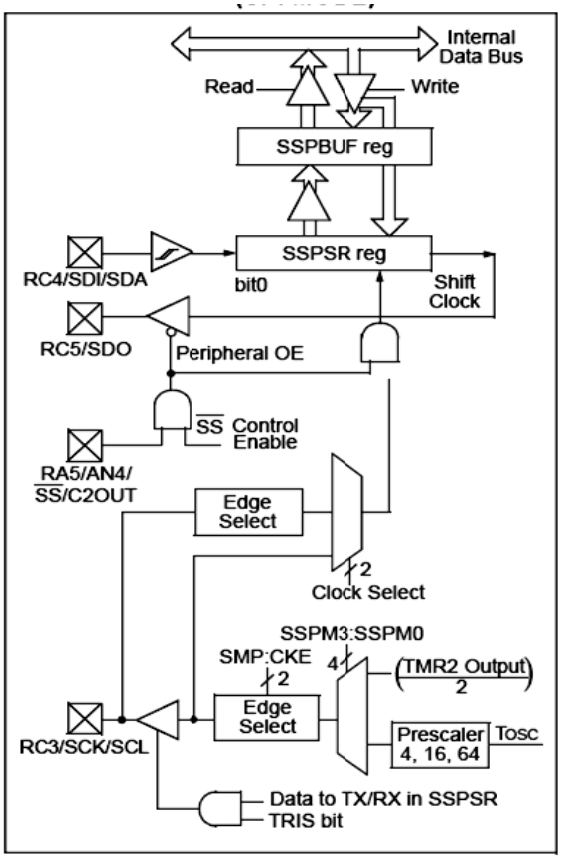


FIGURA 2.31

Modo SPI

2.5.6.2 MODO I²C

Este tipo de interfaz serie ha sido desarrollado por Philips y utiliza solo dos hilos trenzados y una masa común para la interconexión de los diversos dispositivos, que han tenido que ser diseñados para soportar este protocolo, asegurando una gran fiabilidad en la comunicación que llega a tolerar una velocidad máxima de 400 Kbps. Es capaz de interconectar hasta 128 dispositivos situados a gran distancia, por lo que resulta muy usado en edificios inteligentes, control de distribuciones de electricidad, agua y gas, piscifactorías, etc.

El master es el que inicia y termina la transferencia general y provee de la señal de reloj. El esclavo es el dispositivo direccionado por el master, mediante 7 bits, lo que limita el número de componentes a 128.

El inicio de la transmisión se determina con el bit de inicio (S) y el final con otro bit de stop (P). El bus serie de 2 hilos utiliza uno de ellos para transferir datos (SDA) y el otro para la señal de reloj (SCL).

En el protocolo I^2C cada dispositivo tiene asignada una dirección de 7 o de 10 bits que envía el master cuando comienza la transferencia con uno de ellos. Tras la dirección se añade el bit de recepción/transmisión o lectura/escritura (R/W). Los datos se transmiten con longitud byte y al finalizar cada uno se inserta un bit de reconocimiento ACK. Debe existir un modulo de arbitraje que gestione que solo hay un maestro en cada instante sobre el bus compartido.

La figura 2.32 muestra un esquema interno de funcionamiento del interfaz I^2C.

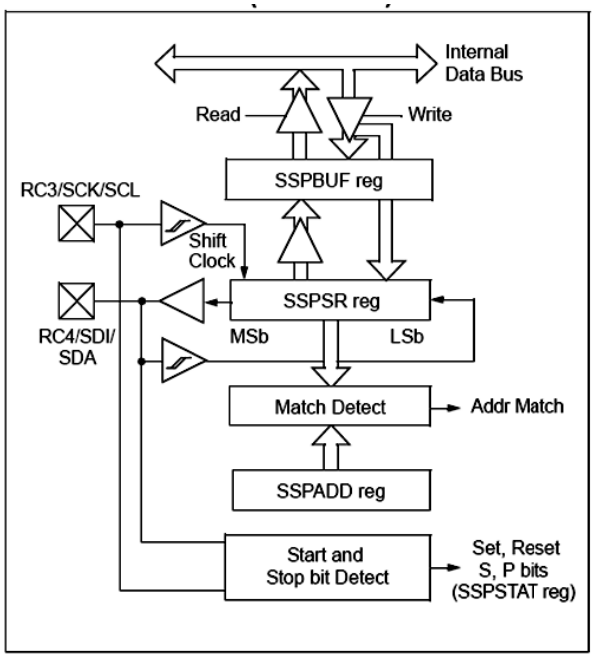


FIGURA 2.32

Modo I^2C

SSPBUF es el registro donde se almacena el byte a transmitir o el que se recibe. SSPSR es el registro desplazamiento serie de la línea E/S. SSPADD es el registro de direcciones que identifica el dispositivo (modo esclavo) o que lo direcciona (modo master). El registro de control SSPCON selecciona las diversas funciones del modo I^2C..

Cada vez que se detecta un bit de inicio o un bit de stop es posible que se active la bandera SSPIF y en el caso de estar también activado el bit de permiso correspondiente generar una interrupción.

2.5.7 INTERFAZ DE COMUNICACIONES SERIE (USART-SCI)

La interfaz de comunicaciones SCI proporciona las mismas prestaciones que una UART programable. Se puede configurar de dos modo diferentes:

Asíncrono (full-duplex)

La comunicación es bidireccional. la terminal RC6/Tx/CK actúa como línea de transmisión y la RC7/Rx/DT como línea de recepción. Cada dato lleva un bit de inicio y otro de stop.

Síncrono (semiduplex)

Comunicación unidireccional. Una sola línea para los datos que se implementan sobre la terminal RC7/Rx/DT. Existen dos modos, en el modo master la señal de reloj sale por la terminal RC6/Tx/CK y en el modo esclavo entra por ella.

En ambos modo los datos pueden ser de 8 o 9 bits, pudiendo emplear el noveno como bit de paridad, transmitiéndose o recibiéndose por el bit <0> de RXSTA y/o RCSTA.

El registro específico TXSTA actúa como registro de estado y control del transmisor y el RCSTA hace lo mismo para el receptor.

La velocidad en baudios se establecen por el valor cargado en el registro SPBRG y el bit BRGH del registro TXSTA, con el que se puede elegir la velocidad alta (1) o baja (0) en el modo asíncrono.

$$BAUDIOS = F_{OSC} / (n(x + 1))$$

n = 4 en el modo síncrono
n = 16 en el modo asíncrono de alta velocidad
n = 64 en el modo asíncrono de baja velocidad
x = valor cargado en el registro SPBRG

y siendo $x = (F_{OSC} / Baudios) / (n - 1)$

Mediante la programación de los bits del registro TXSTA y RCSTA se configura el modo de trabajo. Así, SPEN configura RC7/Rx y RC6/Tx como líneas de comunicación serie. El transmisor se activa con el bit TxEN. El dato a transmitir se carga en TxREG y luego pasa al registro transmisor TSR, cuando se haya transmitido el bit de stop del dato anterior. Entonces se activa la bandera TxIF y si el bit de permiso esta activado se produce una interrupción.

Activando Tx8/9 se inserta el noveno bit almacenado en el bit <0> (TxD8) de TXSTA. El bit TRMT indica si el transmisor esta vacío o no. El dato se recibe por RSR y cuando se completa se pasa al registro RCREG para su posterior lectura, activándose la bandera RCIF y si acaso la interrupción.

Si se activa el bit RC8/9del RCSTA el noveno bit se deposita en el bit <0> (RCD8) del RCSTA. Los bits OERR y FERR indican error de desbordamiento y de trama, respectivamente.

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1= Master mode (clock generated internally from BRG)
0= Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1= Selects 9-bit transmission
0= Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1= Transmit enabled
0= Transmit disabled
Note:SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1= Synchronous mode
0= Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1= High speed
0= Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit
1= TSR empty
0= TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURA 2.33
Registro TXSTA (Direccion 98H)

En la figura 2.33 se ofrece la asignación de funciones de los bits de los registros TXSTA y RCSTA que gobiernan al receptor y transmisor asíncronos, respectivamente.

En modo síncrono el SCI trabaja en half duplex, no pudiendo emitir y transmitir a la vez. La señal de reloj la envía el transmisor (maestro) conjuntamente con los datos. Los principios y el funcionamiento de la emisión y la recepción sincronas son similares al modo asíncrono y únicamente hay que seleccionar esta forma de trabajo cargando adecuadamente los registros TXSTA y RCSTA.

RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7	**SPEN:** Serial Port Enable bit 1= Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins) 0= Serial port disabled
bit 6	**RX9**: 9-bit Receive Enable bit 1= Selects 9-bit reception 0= Selects 8-bit reception
bit 5	**SREN**: Single Receive Enable bit Asynchronous mode: Don't care. Synchronous mode – Master: 1= Enables single receive 0= Disables single receive This bit is cleared after reception is complete. Synchronous mode – Slave: Don't care.
bit 4	**CREN**: Continuous Receive Enable bit Asynchronous mode: 1= Enables continuous receive 0= Disables continuous receive Synchronous mode: 1= Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN) 0= Disables continuous receive
bit 3	**ADDEN:** Address Detect Enable bit Asynchronous mode 9-bit (RX9 = 1): 1= Enables address detection, enables interrupt and load of the receive buffer when RSR<8>is set 0= Disables address detection, all bytes are received and ninth bit can be used as parity bit bit 2 **FERR**: Framing Error bit 1= Framing error (can be updated by reading RCREG register and receive next valid byte) 0= No framing error
bit 1	**OERR**: Overrun Error bit 1= Overrun error (can be cleared by clearing bit CREN) 0= No overrun error
bit 0	**RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.34

Registro RCSTA.

La figura 2.33 muestra el registro TXSTA donde el bit 7, CSRC, es el bit de selección de reloj siendo solo importante en el modo síncrono donde se elige entre maestro o esclavo. El bit 6 habilita la transmisión de 8 o 9 bits, en general, es TXEN, quien habilita la transmisión. El bit SYNC selecciona el modo USART entre síncrono y asíncrono. Finalmente BRGH selecciona si el rango de baudios será de baja o alta velocidad en el modo asíncrono, TRMT es el bit de estado del registro

de desplazamiento del transmisor y TX9D es el noveno bit de datos de transmisión. Se puede emplear como bit de paridad.

La figura 2.34 muestra el registro RCSTA donde el bit 7, SPEN, es el bit de habilitación del puerto serie. El bit 6 habilita la recepción de 8 o 9 bits y SREN, bit 5, habilita la recepción única y CREN, bit 4, la recepción continua. Los bits 1 y 2 son bits de error y finalmente, RX9D es el noveno bit de datos de recepción.

2.5.8 LECTURA Y ESCRITURA DE LA MEMORIA DE DATOS EEPROM

En la familia de microcontroladores 16F87X tanto la memoria EEPROM de datos como la memoria de programa FLASH puede ser modificada sin necesidad de utilizar un programador exterior.

Se dispone de seis registros de SFR para leer y escribir sobre la memoria no volátil, estos registros son: EECON1, EECON2, EEDATA, EEDATH, EEADR y EEADRH. Para direccionar las 256 posiciones de la memoria EEPROM del PIC16F876 y 16F877 basta con 8 bit, por ello para escribir o leer solo hacen falta el registro EEADR para direccionar la posición y el registro EEDATA para colocar el dato leído o escrito. Sin embargo para poder escribir o leer datos en la memoria FLASH que puede tener hasta 8K palabras de 14 bits hacen falta dos registros para direccionar la posición de memoria, por ello se utiliza el registro EEADR concatenado con el registro EEADRH que contiene la parte alta de la palabra de direccionamiento de memoria. De forma similar se utilizan los registros EEDATA concatenado con el registro EEADRH que contiene los 6 bit de mayor peso de las palabras de 14 bits.

Además para controlar el proceso de lectura y escritura de la memoria EEPROM y FLASH se dispone de dos registros: el EECON1 y el EECON2.

2.5.8.1 LECTURA DE LA MEMORIA DE DATOS

Para leer un dato de la EEPROM, el registro EEADR es cargado con la dirección de la EEPROM donde se encuentra el dato y luego el microcontrolador copia el dato de dicha posición a EEDATA. A continuación hay que poner a 0 el bit EEPGD (EECON1<7>), para apuntar a la memoria de datos EEPROM. Una vez que se ponga a 1 la bandera RD (EECON1<0>), el dato estará disponible en el registro EEDATA, donde permanecerá hasta la siguiente escritura o lectura.

EECON1 REGISTER (ADDRESS 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7	**EEPGD**: Program/Data EEPROM Select bit 1= Accesses program memory 0= Accesses data memory Reads '0' after a POR; this bit cannot be changed while a write operation is in progress.
bit 6-4	**Unimplemented:** Read as '0'
bit 3	**WRERR:** EEPROM Error Flag bit 1= A write operation is prematurely terminated (any MCLR or any WDT Reset during normal operation) 0= The write operation completed bit 2**WREN:** EEPROM Write Enable bit 1= Allows write cycles 0= Inhibits write to the EEPROM
bit 1	**WR:** Write Control bit 1= Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software. 0= Write cycle to the EEPROM is complete bit 0 **RD:** Read Control bit 1= Initiates an EEPROM read; RD is cleared in hardware.The RD bit can only be set (not cleared) in software. 0= Does not initiate an EEPROM read

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURA 2.35

Registro EECON1.

(Dirección 18CH)

2.5.8.2 ESCRITURA DE LA MEMORIA DE DATOS

La escritura, que es en realidad una programación, es más compleja por razones de seguridad. Antes de escribir un dato en la EEPROM, debe ponerse a 1 la bandera de activación de escritura WR (EECON1<1>). Para transferir el dato desde el registro EEDATA a la dirección de la EEPROM a la que apunta EEADR, debe ejecutarse una secuencia obligatoria indicada por el fabricante (Microchip). Posteriormente, cuando se ha realizado con éxito la operación de la bandera EEIF (PIR1<7>) se pone a 1. Si no lo hace, el almacenamiento ha sido incorrecto y no se ha realizado.

2.6 REPERTORIO DE INSTRUCCIONES DE LA GAMA MEDIA

Habiendo escogido los diseñadores de PIC la filosofía RISC, su juego de instrucciones es reducido, siendo éstas, además, sencillas y rápidas, puesto que casi todas se ejecutan en un único ciclo de máquina

(equivalente a 4 del reloj principal). Sus operandos son de gran flexibilidad, pudiendo actuar cualquier objeto como fuente y como destino, las cuales se muestran en la figura 2.36.

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

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

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

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

Note: Additional information on the mid-range instruction set is available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

FIGURA 2.36. *Conjunto de instrucciones del PIC16F877A*

Para comprender estas instrucciones, ante todo es conveniente tener clara la estructura interna del microcontrolador, puesto que las instrucciones la referencian, y puesto que en cualquier micro la comprensión de la nomenclatura de sus componentes es esencial. De este modo se expone la tabla de la figura 2.37 para ayudar a comprender las abreviaturas, y seguidamente las 35 instrucciones para la gama media.

Abreviatura	Descripción
PC	Contador de Programa que direcciona la memoria de instrucciones. Tiene un tamaño de 11 bits en la gama baja, de los cuales los 8 de menos peso configuran el registro PCL que ocupa el registro 0x02 del área de datos.
TOS	Cima de la pila, con 2 niveles en la gama baja y 8 en la media
WDT	Perro guardián (Watchdog)
W	Registro W, similar al acumulador
F	Suele ser un campo de 5 bits (fffff) que contiene la dirección del banco de registros, que ocupa el banco 0 del área de datos. Direcciona uno de esos registros.
D	Bit del código OP de la instrucción, que selecciona el destino. Si d=0, el destino es W, y si d=1 el destino es f.
Dest	Destino (registro W o f)
TO	Bit "Time Out" del registro de estado
PD	Bit "Power Down" del registro de estado
b	Suele ser un campo de 3 bits (bbb) que determinan la posición de un bit dentro de un registro de 8 bits
k	Se trata, normalmente, de un campo de 8 bits (kkkkkkkk) que representa un dato inmediato. También puede constar de 9 bits en las instrucciones de salto que cargan al PC
x	Valor indeterminado (puede ser un 0 o un 1). Para mantener la compatibilidad con las herramientas software de Microchip conviene hacer x = 0
label	Nombre de la etiqueta
[]	Opciones
()	Contenido
→	Se asigna a
<>	Campo de bits de un registro
∈	Pertenece al conjunto
Z	Señalizador de cero en W. Pertenece al registro de estado
C	Señalizador de acarreo en el octavo bit del W. Pertenece al registro de estado
DC	Señaliza el acarreo en el 4 bit del W. Pertenece al registro de estado
Itálicas	Términos definidos por el usuario

FIGURA 2.37

Tabla de abreviaturas

CUESTIONARIO.

1. ¿En qué filosofía se basan los microcontroladores PIC?
2. ¿En qué consiste la filosofía pipe-line?
3. ¿En qué consiste la filosofía RISC?
4. Mencione las características principales de los microcontroladores PIC.
5. ¿Cuáles son las familias de microcontroladores PIC y cuáles son sus características principales?
6. ¿Cuántos tipos de memoria utiliza un microcontrolador PIC de gama media?
7. ¿En qué consiste el registro STATUS?
8. ¿Cómo funciona el registro OPTION_REG?
9. Defina la función del registro INTCON
10. Analice y describa el funcionamiento del registro ADCON 0 y ADCON1.

CAPÍTULO 3
AMBIENTE DE TRABAJO DE MIKROC IDE

El compilador mikroC es una herramienta aplicada al desarrollo de sistemas distribuidos en donde se utilizan PIC's de las familias PIC12, PIC16 y PIC18 sin comprometer su performance o control.

Siendo los microcontroladores PIC's una familia mundialmente conocida por su alto desempeño en su tipo, y el lenguaje C apreciado por su eficiencia en variedad de sistemas, es natural que se elija para desarrollar sistemas distribuidos. Se elige al compilador mikroC , que cuenta con características avanzadas IDE y cumple con las normas que exige ANSI C.

Con mikroC se pueden desarrollar e implementar aplicaciones de una forma rápida y eficiente, ya que cuenta con un ambiente grafico, lo cual permite que el panorama de programación sea amigable para el programador.

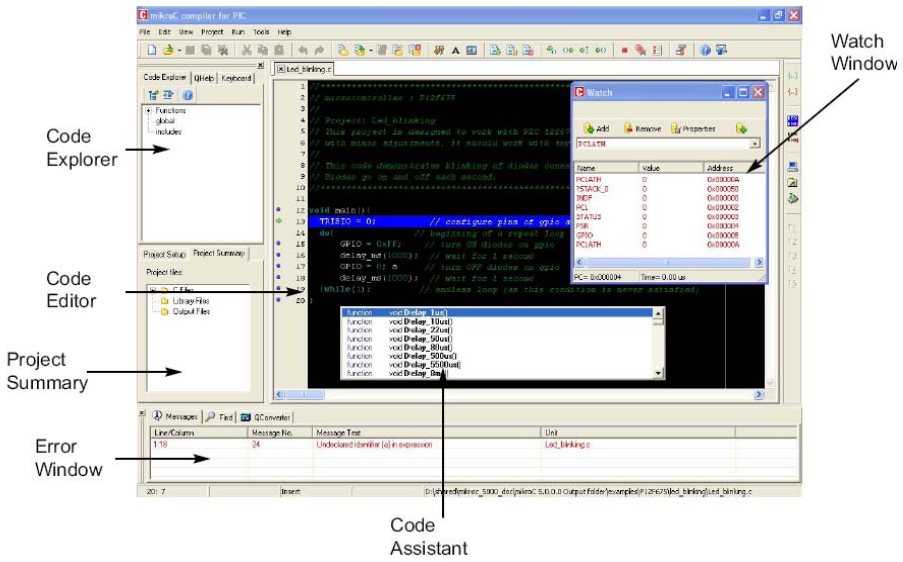


FIGURA 3.1

Ventana principal del Ambiente de mikroC

En el ambiente de mikroC se pueden desarrollar y desplegar rápidamente aplicaciones complejas permitiendo:

- Escribir el código fuente C utilizando su editor de código (Code Editor).

- Usar las librerías que vienen incluidas en mikroC, las cuales permiten acelerar el desarrollo de las aplicaciones tales como: adquisición de datos, memorias, displays, conversiones, comunicaciones…
- Monitorear la estructura del programa, variables y funciones en el explorador de código (code explorer). Genera comentarios, ensamblador legible para humanos, y estándar HEX compatible con todos los programadores.
- Examinar el flujo de programa y eliminar errores de la lógica ejecutable con el depurador (Debbuger) integrado.
- A continuación se describen las principales ventanas de dialogo q se utilizan para programar en mikroC.

3.1 CODE EDITOR (EDITOR DE CÓDIGO)

El code editor es un editor de texto, común a las características del entorno de Windows, dentro de sus características se incluye:

- Sintaxis resaltada
- Asistente de código
- Asistente de parámetros
- Código predefinido (autocompletar)
- Autocorregir para errores tipográficos comunes

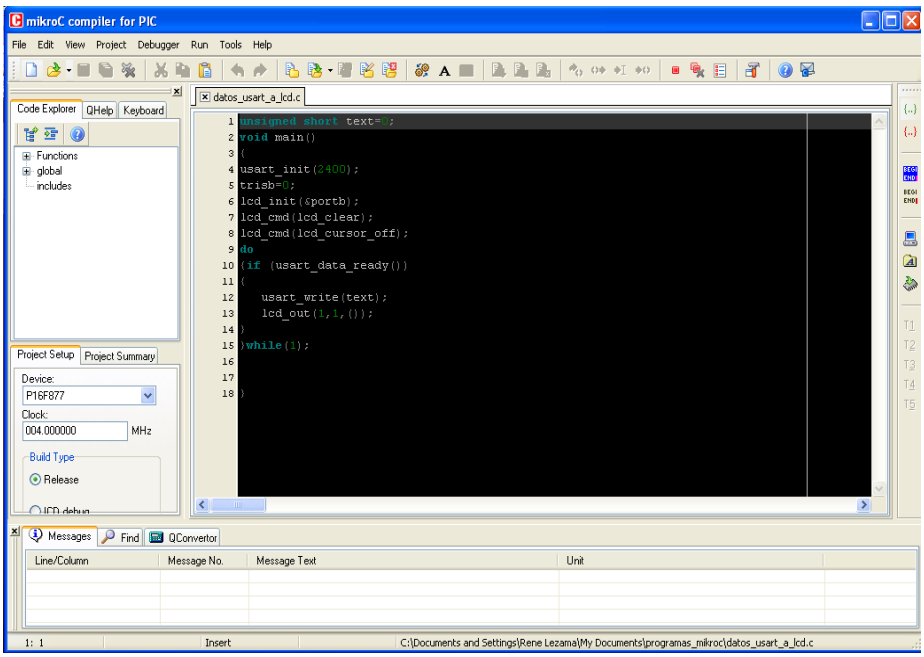


FIGURA 3.2

Code Editor

3.1.1 EDITOR SETTINGS (EDITORDE AJUSTES).

Icono tools

Se pueden ajustar estas opciones a partir del cuadro de dialogo Editor Settings. Para acceder a los ajustes, escoger Tools>Options del menú desplegable o dar "clic" en el icono Tools.

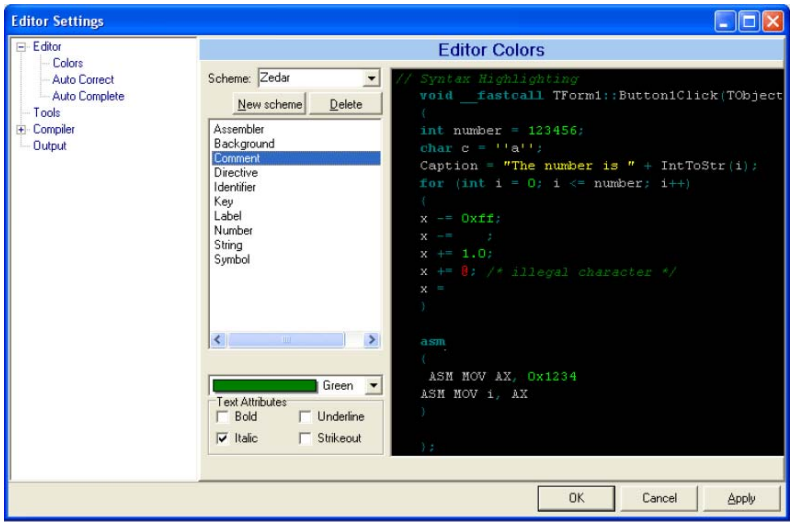


FIGURA 3.3

Ventana de dialogo del editor Settings

3.1.2 ASISTENTE DE CÓDIGO.

Al empezar a escribir una palabra se puede presionar CTRL+SPACE, con lo cual todos los identificadores validos que correspondan a las primeras letras que se escribieron se desplegaran en una ventana emergente. Entonces se puede escribir la opción más cercana a las necesidades propias del programa o elegir una con las flechas del teclado.

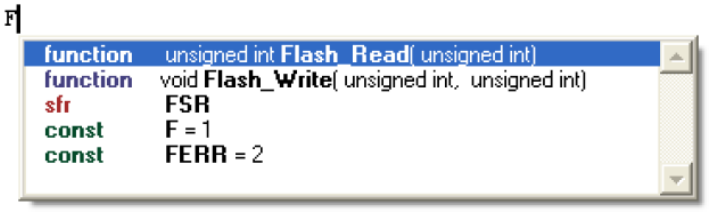


FIGURA 3.4

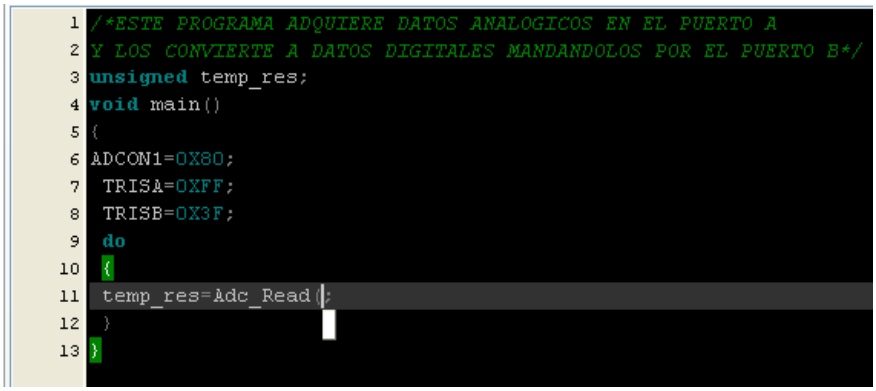
Ventana de Asistente de Código.

3.1.3 ASISTENTE DE PARÁMETROS [CTRL+SHIFT+SPACE]

El asistente de parámetros será invocado automáticamente cuando se abra un paréntesis "(“después de una función o presionando CTRL+SHFT+SPACE. Si el nombre de una función precede el paréntesis, entonces los parámetros previstos serán mostrados en un panel flotante

FIGURA 3.5

Ventana de Asistente de parámetros.

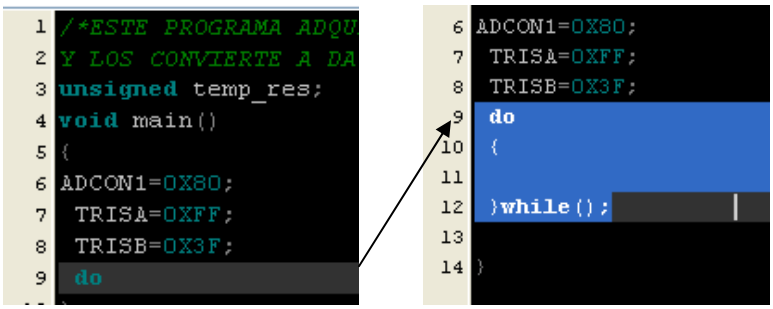


3.1.4 CODE TEMPLATE (PLANTILLA DE CÓDIGO)[CTRL+J].

Se refiere a estructuras predefinidas en código inherente al compilador. Al empezar a escribir un programa se inserta parte del código y se teclea CTRL+J, si se desea, y entonces automáticamente se generara el codigo restante a esa estructura. O se puede cliquear tool de la barra de herramientas y en opciones se elige autocompletar entonces se selecciona la plantilla deseada o simplemente seleccionando F12 aparecerá la misma ventana y se selecciona la misma plantilla.

FIGURA 3.6

Plantilla prediseñada de la estructura do-while.



Se pueden adicionar plantillas personales a la lista. Solo se selecciona Tools>Options de la barra de herramientas o presionando F12 y seleccionando autocompletar. Aquí introducirlas palabras reservadas apropiadas a utilizar, descripción y el código de la plantilla.

3.1.5 AUTOCORRECT (AUTOCORRECCIÓN).

La característica Auto correct corrige errores comunes de escritura. Para acceder a la lista de errores tipográficos, seleccionar Tools>Options de el menú emergente. De la misma forma que en las plantillas de codigo se pueden adicionar posibles errores comunes a la lista.

El editor código puede habilitar un bloque de código como comentario o como no comentario haciendo un "clic" en el mouse, o utilizando el icono comment/uncomment de la barra de herramientas de código.

3.1.6 BOOKMARKS (MARCAS DE TEXTO).

Los *bookmark* (marcas de texto) pueden hacer la navegación o exploración de un código más fácil. Esto se hace de la siguiente forma:

[CTRL+SHIFT+<number>]: crea el *bookmark*.

[CTRL+<numero>] va al *bookmark*

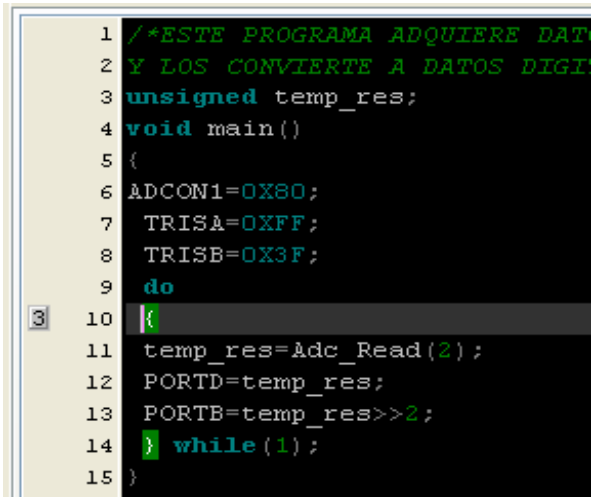


FIGURA 3.7

Bookmark asignado a línea 10.

3.1.7 GOTO LINE (NAVEGADOR ENTRE LINEAS).

Esta opción permite que la navegación a través de las líneas de programa extenso sea más fácil. Seleccionando *Search>goto line* del menú emergente o utilizando la combinación de teclas [CTRL+G]

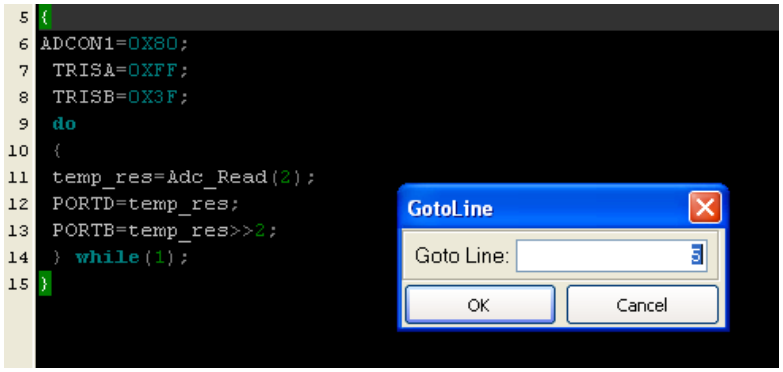


FIGURA 3.8

Ventana emergente Goto line (CTRL+G).

3.2 CODE EXPLORER (EXPLORADOR DE CÓDIGO).

El explorador de código está colocado a la izquierda de la ventana principal y da una vista detallada de cada ítem declarado en el código fuente. Se puede recurrir a alguna declaración de un ítem en específico dando doble "clic" en el mismo.

También están habilitadas dos tablas mas en el code explorer. QHelp muestra en un apéndice las funciones y estructuras built-in contenidas en la biblioteca, para una referencia rápida. Haciendo doble "clic" en una de los tópicos se abre una ventana de ayuda acerca de ellas. El apéndice Keyboard enlista todos los atajos que se pueden hacer en mikroC por medio del teclado.

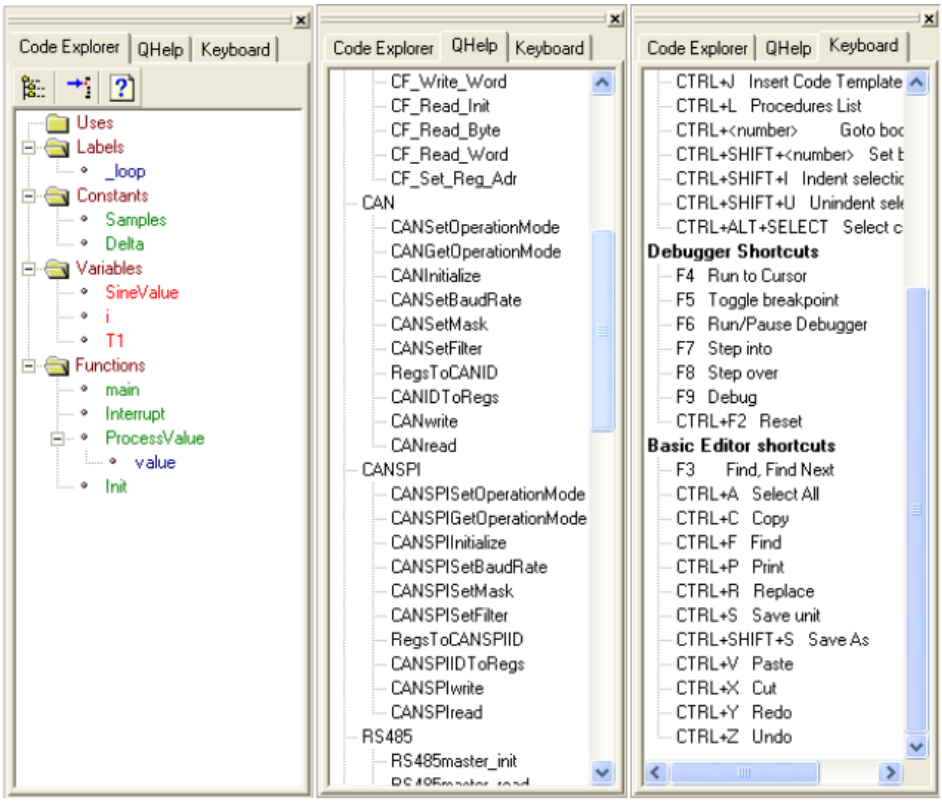


FIGURA 3.9

Ventanas del Code explorer,QHelp y del Keyboard.

3.3 WATCH WINDOW

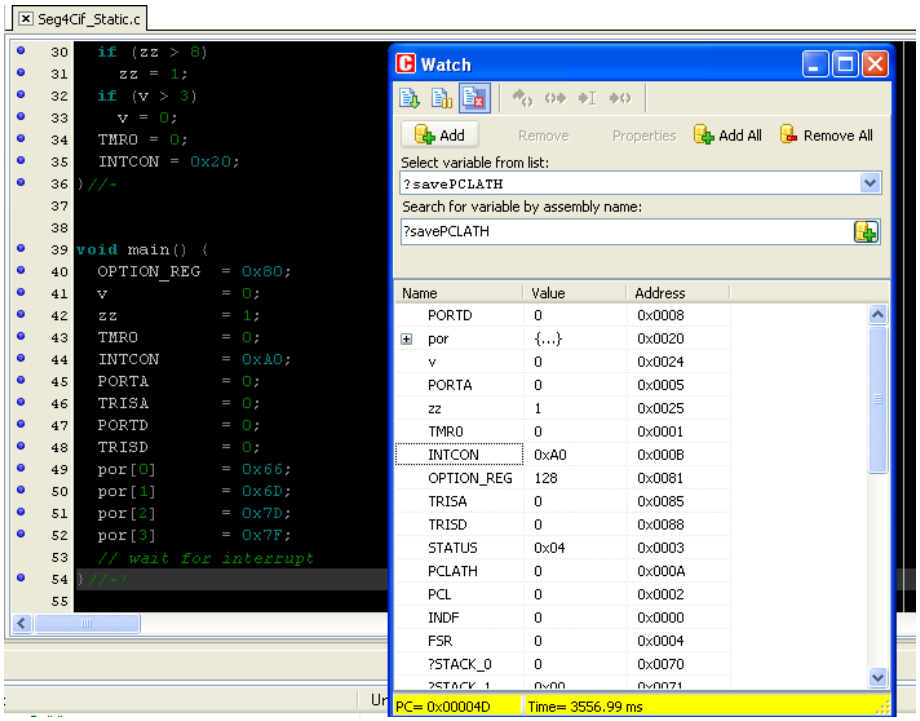


FIGURA 3.10

Watch Window

La ventana Watch Window permite monitorear elementos del programa mientras este se está corriendo. Esta muestra las variables y los registros especiales de funciones del PIC MCU, sus direcciones y valores. Los valores son actualizados conforme se avanza la simulación.

Realizando un doble "clic" en uno de los elementos abre una ventana la cual permite asignar un valor nuevo a la variable o registro y cambiar el formato del numero.

FIGURA 3.11

Ventanas para cambiar el formato del número.

3.4 STOPWATCH WINDOW

La ventana *Stopwatch* muestra la cuenta de ciclos desde la última acción del debbuger. Stopwatch mide la ejecución del del periodo (numero de ciclos) desde el momento en que el debbuger comenzó, y puede ser reseteado en cualquier momento. Delta representa el número de ciclos entre la línea de instrucción previa y la línea de instrucción activa.

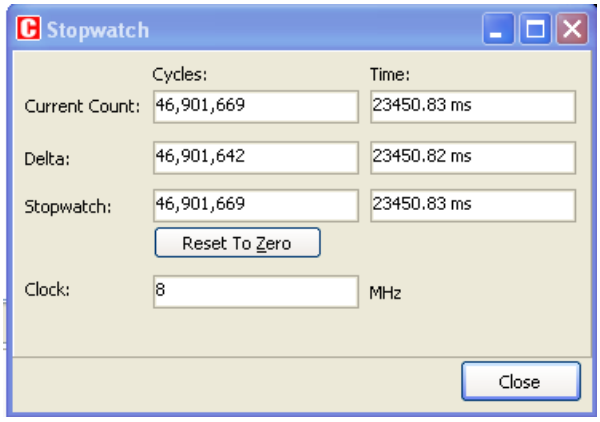


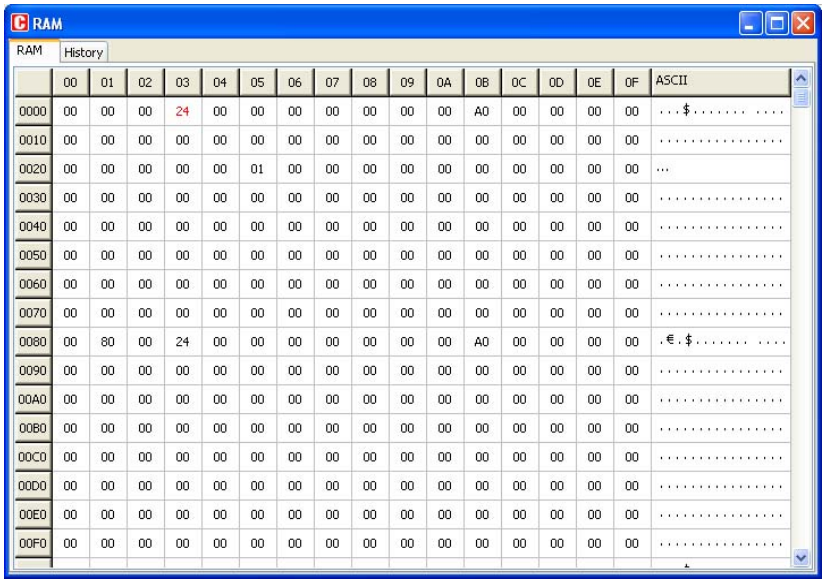
FIGURA 3.12

Ventana Stopwatch

***Nota**: Se puede cambiar el reloj la ventana Stopwatch; con lo cual se recalcularían valores, para una nueva frecuencia en especifico. Esto no afectaría el proyecto solo seria para mera simulación.*

3.5 VIEW RAM WINDOW.

Se accede a esta ventana desde la barra principal View>Debug Windows>View Ram.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	00	00	24	00	00	00	00	00	00	00	A0	00	00	00	00	...$..... ....
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0020	00	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	...
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0080	00	80	00	24	00	00	00	00	00	00	00	A0	00	00	00	00	.€.$..... ....
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

FIGURA 3.13

Ventana View RAM.

La ventana View RAM muestra el mapa de memoria RAM del Pic, con los cambios recientes que se hayan realizado coloreados en rojo.se puede realizar algún cambio en cualquier campo dando doble “clic” en el.

3.6 ERROR WINDOW.

En caso de que durante la compilación se encontraran algunos errores, el compilador los reportara y podría no generar el archivo *hex.* La ventana de error aparecerá por default en la parte inferior de la ventana principal.

La ventana de error está localizada debajo de la etiqueta *message* y muestra la ubicación y el tipo de error que el compilador a encontrado .

El compilador también reporta warnings, pero estos no afectan la salida; solo los errores pueden interferir con la generación del código *hex* .

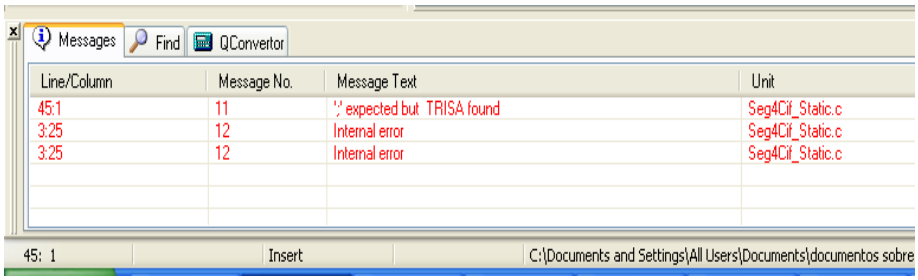


FIGURA 3.14

Ventana de Error

Realizando un doble "clic" sobre la línea con el mensaje de error, conducirá a la línea del código donde se localizo este, de una manera resaltada.

3.7 STATISTICS (ESTADÍSTICAS).

Después de realizar la compilación, se pueden revisar las estadísticas del código. Seleccionar View>*View Statistics* del menú principal, o seleccionando el icono Statistics. Existen 6 tipos de de ventanas para este elemento.

3.7.1 MEMORY USAGE WINDOW.

Esta ventana provee una vista general del uso de la memoria RAM y ROM en forma de histograma.

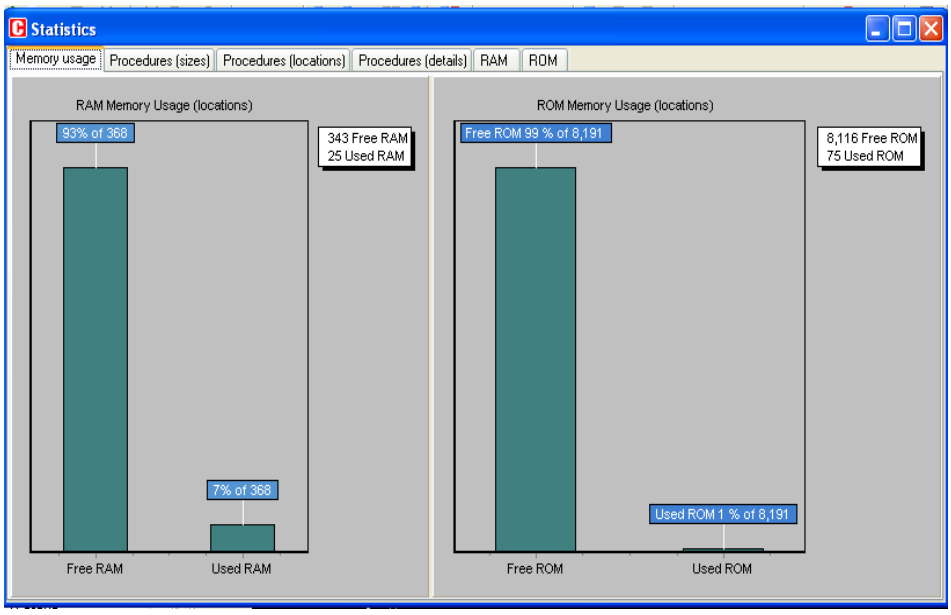


FIGURA 3.15

Ventana Memory Usage

3.7.2 PROCEDURES(SIZES) WINDOW

Muestra las funciones en forma de histograma de acuerdo al lote de memoria que usan.

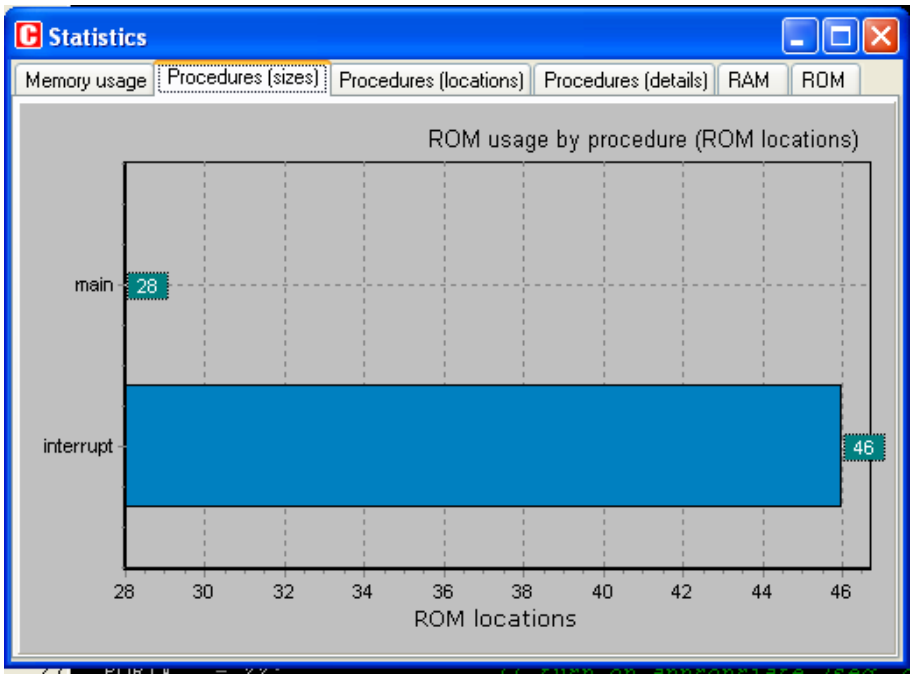


FIGURA 3.16

Ventana Procedure(sizes)

3.7.3 PROCEDURES (LOCATIONS) WINDOW

Muestra como están distribuidas las funciones en la memoria del microcontrolador.

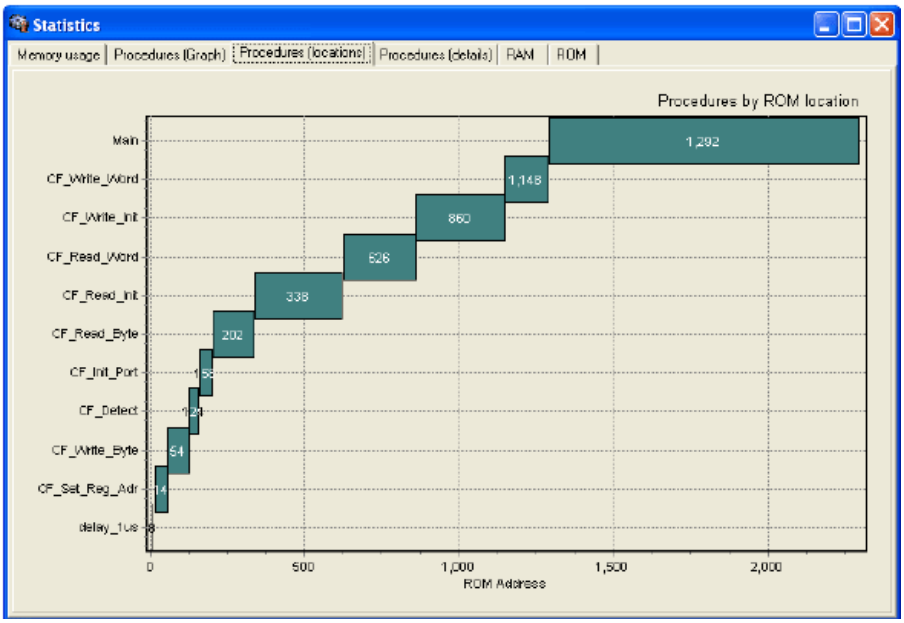


FIGURA 3.17

Ventana Procedure(location)

3.7.4 PROCEDURE (DETAIL) WINDOW

Exhibe el árbol completo de llamadas, junto con detalles para cada función: tamaño, comienzo y fin de la dirección, frecuencia dellamadas, tipo de retorno, etc.

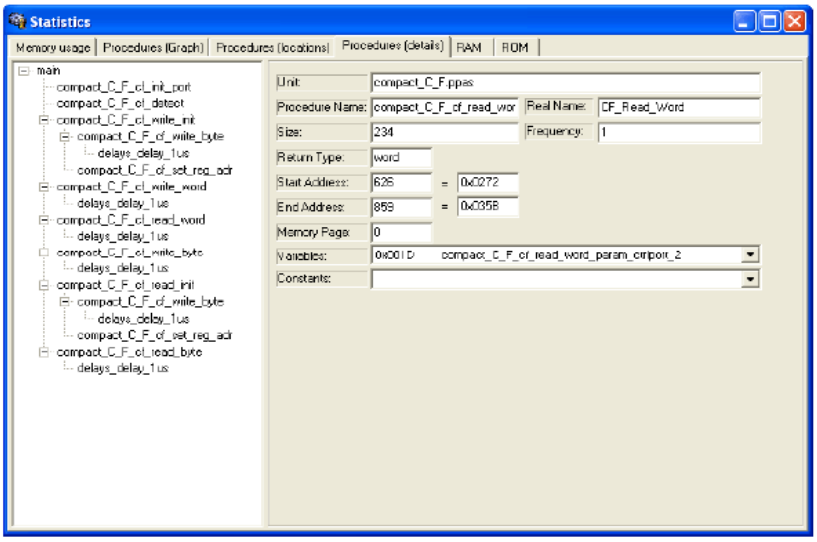


FIGURA 3.18

Ventana Procedure(details)

3.7.5 RAM WINDOW

Resume todos los registros GPR y SFR y sus direcciones. También exhibe nombres simbólicos de las variables y sus direcciones.

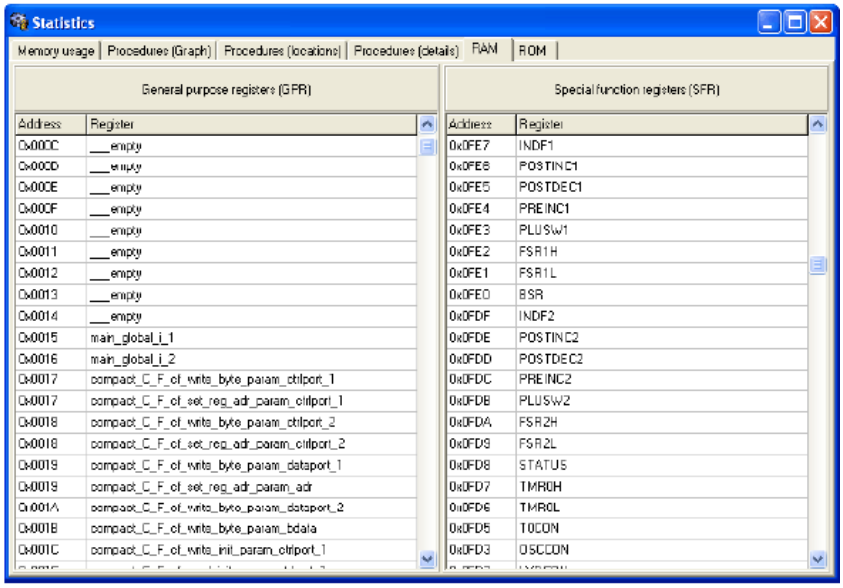
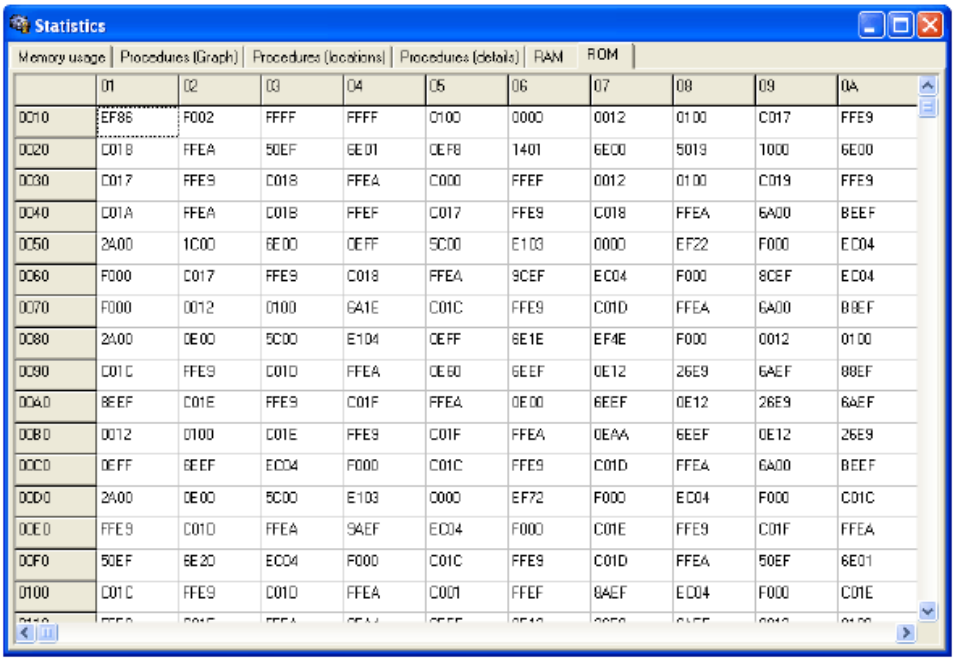


FIGURA 3.19

Ventana RAM

3.7.6 ROM WINDOW

Enlista los op-codes (códigos de operación) y sus direcciones en forma de código hexadecimal comprensible para humanos.



	01	02	03	04	05	06	07	08	09	0A
0010	EF86	F002	FFFF	FFFF	0100	0000	0012	0100	C017	FFE9
0020	C018	FFEA	50EF	6E01	0EF8	1401	6EC0	5019	1000	6E00
0030	C017	FFE9	C018	FFEA	C000	FFEF	0012	0100	C019	FFE9
0040	C01A	FFEA	C018	FFEF	C017	FFE9	C018	FFEA	6A00	BEEF
0050	2A00	1C00	6E00	0EFF	5C00	E103	0000	EF22	F000	EC04
0060	F000	C017	FFE9	C018	FFEA	9CEF	EC04	F000	8CEF	EC04
0070	F000	0012	0100	6A1E	C01C	FFE9	C01D	FFEA	6A00	B8EF
0080	2A00	0E00	5C00	E104	0EFF	6E1E	EF4E	F000	0012	0100
0090	C01C	FFE9	C01D	FFEA	0E60	6EEF	0E12	26E9	6AEF	88EF
00A0	BEEF	C01E	FFE9	C01F	FFEA	0E00	6EEF	0E12	26E9	6AEF
00B0	0012	0100	C01E	FFE9	C01F	FFEA	0EAA	6EEF	0E12	26E9
00C0	0EFF	6EEF	EC04	F000	C01C	FFE9	C01D	FFEA	6A00	BEEF
00D0	2A00	0E00	5C00	E103	0000	EF72	F000	EC04	F000	C01C
00E0	FFE9	C01D	FFEA	9AEF	EC04	F000	C01E	FFE9	C01F	FFEA
00F0	50EF	6E20	EC04	F000	C01C	FFE9	C01D	FFEA	50EF	6E01
0100	C01C	FFE9	C01D	FFEA	C001	FFEF	8AEF	EC04	F000	C01E

FIGURA 3.20

Ventana ROM

CAPITULO 4
DESARROLLO DE APLICACIONES Y TOKENS

En el capítulo 3 se explico el funcionamiento del ambiente de trabajo del compilador mikroC. En este capítulo se explicara la forma para desarrollar programas o proyectos de programación para PIC's en mikroC. En el capitulo siguiente se procederá a la explicación de las librerías y funciones más utilizadas por este compilador, para así llevar a cabo la realización de proyectos para la comprensión de la programación de PIC's bajo este ambiente,

4.1 CREANDO PROYECTOS

El compilador mikroC organiza las aplicaciones de un proyecto en una sola carpeta llamada proyectos, consistiendo de un archivo con extensión .ppc y uno o más archivos fuente (extensión .c). Los archivos fuente se pueden compilar sólo si son parte del proyecto.

El archivo proyecto lleva la siguiente información:

- Nombre del proyecto y descripción opcional.
- Dispositivo a utilizar.
- Banderas del dispositivo y reloj del mismo.
- Lista de códigos fuente del proyecto con sus rutas.

 **Nuevo proyecto.**

En la ventana principal que se muestra en la figura 3.1 (Capitulo 3) se observa el icono  , dando "clic" en él se comienza con la creación de un proyecto por medio del '*asistente de nuevo proyecto*'.

Sólo se debe llenar el cuadro de diálogo con los valores deseados (nombre del proyecto y descripción, ubicación, dispositivo, palabra de configuración) y mikroC creara el archivo apropiado del proyecto.

 **Editando el proyecto.**

Posteriormente se pueden hacer ajustes al proyecto dando "clic" en el menú *Proyect> Edit Proyect*. Se puede dar otro nombre al proyecto, modificar su descripción, cambiar el chip, cambiar el reloj o frecuencia, palabra de configuración etc. Para borrar el proyecto, simplemente se debe borrar el folder en el cual se encuentra alojado el proyecto.

 **Adicionando o removiendo archivos de un proyecto.**

Un proyecto puede contener cualquier número de archivos fuentes (extensión .c). La lista de archivos fuentes pertinentes se guarda en el archivo de proyecto (extensión .ppc). Para adicionar un archivo fuente a un proyecto, seleccionar *Proyect > Add to Project* del menú desplegable. Cada archivo fuente añadido debe ser autónomo, o sea debe tener todas las definiciones necesarias después de el preprocesado.

 Para remover archivo (s) de un proyecto, seleccionar *Proyect > Remove from Proyect* del menú desplegable.

4.2 SOURCE PATHS (RUTAS FUENTE)

4.2.1 SOURCE FILES (ARCHIVOS FUENTE)

El archivo fuenteque contenga el código en C deberá tener la extensión .c. la lista del código fuente relevante en la aplicación es almacenada en el archivo proyecto con la extensión .ppc, junto con otra información del proyecto. Los códigos fuente sólo se compilarán si son parte del mismo proyecto.

Usar la directiva *#include* del procesador para incluir encabezados. No confiar en el procesador para incluir otros códigos fuente.

4.2.2 SEARCH PATHS. RUTAS PARA CÓDIGO FUENTE (.C).

Se pueden especificar manualmente las rutas de búsqueda para los archivos fuente a utilizar en el proyecto. Esto puede ser configurado seleccionando Tools>Options del menu emergente, y en la ventana que se despliega dar "clic" en la raíz Project>Search Path.

En la configuración del proyecto, se pueden especificar ambas rutas absolutas o relativas para el código fuente. Si se especifica una ruta relativa mikroC buscará el archivo en la siguiente ubicación, en este orden particular:

1. El folder que contiene el proyecto (folder en el cual se aloja el archivo con extencion .ppc)
2. La ruta dada por el usuario.
3. El folder "uses" que está dentro del folder de instalación de mikroC.

4.2.3 HEADERS FILES (.H) (RUTAS PARA ARCHIVOS DE CABECERA).

Los archivos de cabecera son incluidos por medio del preprocesador con la directiva *#include*. Si se coloca una ruta explicita para el archivo de cabecera en la directiva del preprocesador, solo esa ubicación será buscada.

Se pueden especificar rutas hechas por el usuario para archivos con extensión .h de la misma manera que para los archivos con extensión *.c*

En la configuración del proyecto, se pueden especificar ambas rutas absolutas o relativas para el código fuente. Si se especifica una ruta relativa mikroC buscará el archivo en la siguiente ubicación, en este orden particular:

1. El folder que contiene el proyecto (folder en el cual se aloja el archivo con extensión *.ppc*)
2. El folder "include" que está dentro del folder de instalación de mikroC.
3. La ruta dada por el usuario

4.2.4 CREANDO UN ARCHIVO FUENTE NUEVO.

Para crear un nuevo archivo fuente, hacer lo siguiente:

Seleccionar File>New del menú emergente o presione [CTRLN+N], o de "clic" en el icono New File, una nueva ventana se abrirá con una pestaña llamada "Untitled1". Aquí se podrá crear el nuevo archivo fuente. Seleccionar File>Save As del menú principal para darle nombre adecuado.

Si se ha usado New Proyect Wizard, aparecerá automáticamente una ventana con el nombre que se le ha dado el proyecto y con la extensión "*.c*". El compilador no requiere que el código fuente con extensión "*.c*" lleve el mismo nombre que él que se le ha dado al proyecto, esto es sólo a manera de conveniencia.

4.3 MANIPULANDO ARCHIVOS

4.3.1|ABRIENDO UN ARCHIVO EXISTENTE.

Para abrir un archivo existente seleccionar File>Open del menú principal, o presionar CTRL+O, o dar "clic" en el icono Open File. Hecho lo anterior se abrirá una ventana de selección de archivo, se busca la ubicación del archivo que se quiere abrir, se selecciona y se da "clic" en el botón de abrir.

4.3.2 IMPRIMIENDO UN ARCHIVO.

Para imprimir un archivo solo debe cerciorarse que la ventana contenga al archivo que se desea imprimir, para esto seleccionar File>Print del menú, o presionar [CTRL+p], o dar "clic" en el icono Print.

4.3.3 GUARDANDO UN ARCHIVO.

Para guardar un archivo se debe asegurar que la ventana activa contenga al archivo en cuestión. Seleccionar File>Save del menú, o presionar CTRL+S, o dar "clic" en el icono Save. El archivo será guardado con el nombre activo en la ventana.

4.3.4 GUARDAR UN ARCHIVO CON NOMBRE DIFERENTE.

Para guardar un archivo con nombre diferente seleccione del menú Select File>Save As, o presione SHIFT+CTRL+S. aparecerá una ventana que hará alusión al nuevo nombre que se le quiere dar al archivo. Sólo hay que buscar la carpeta donde se desea guardar el archivo, dar el nuevo nombre del archivo y dar "clic" en el botón Save.

4.3.5 CERRANDO UN ARCHIVO.

Antes de cerrar el archivo se debe verificar que la ventana que se encuentra activa lo contenga. Para cerrar el archivo seleccionar del

menú File>Close, o dar “clic” derecho en la pestaña de la ventana que se desea cerrar en el editor de código. Si el archivo sufrió cambios previos una ventana emergerá preguntando si se desea guardar los cambios.

4.4 COMPILACION.

Una vez creado el proyecto y escrito el código fuente, se procede a su compilación, para esto seleccionar Proyect>Build del menu emergente, o dar “clic” en el icono Build, o simplemente teclear CTRL+9.

Una barra progresiva aparecerá para informar acerca del progreso de la compilación. Si hay errores estos se notificaran en la parte inferior del área de trabajo de mikroC. En caso contrario, mikroC generara los archivos de salida pertinentes.

4.4.1 OUTPUT FILES. (ARCHIVOS DE SALIDA).

Si la compilación es la correcta, mikroC generara archivos de salida en el folder del proyecto general (folder que contiene el archivo con extensión .ppc). Los archivos de salida que mikroC genera se resumen a continuación:

- **Intel HEX file(.hex)**

Registros hex del estilo Intel. Usar estos archivos para programar al PIC MCU.

- **Binary mikro Compiled Library (.mcl)**

Distribución binaria de aplicaciones que pueden ser incluidas en otros proyectos.

- **List file (.lst)**

Forma general de alojar datos en la memoria del PIC: dirección de instrucciones, registros, rutinas, etc.

- **Assembler file (.asm).**

Lenguaje ensamblador comprensible para humanos con nombres simbólicos, extraídos del archivo List.

4.5 TOKENS (COMPONENTES SINTÁCTICOS)

Existen seis clases de *componentes sintácticos* o *tokens* en el vocabulario del lenguaje C: ***palabras clave***, ***identificadores***, ***constantes***, ***cadenas de caracteres***, ***operadores*** y ***separadores***. Los *separadores* –uno o varios espacios en blanco, tabuladores, caracteres de nueva línea (denominados "espacios en blanco" en conjunto), y también los *comentarios* escritos por el programador– se emplean para separar los demás *tokens*; por lo demás son ignorados por el compilador. El compilador descompone el texto fuente o programa en cada uno de sus ***tokens***, y a partir de esta descomposición genera el código objeto correspondiente.

El compilador ignora también los sangrados al comienzo de las líneas.

4.5.1 KEYWORDS (PALABRAS CLAVE DE MIKROC).

En C, como en cualquier otro lenguaje, existen una serie de palabras clave (*keywords*) que el usuario no puede utilizar como identificadores (nombres de variables y/o de funciones). Estas palabras indican al procesador que realice una tarea determinada (desde evaluar una comparación, hasta definir el tipo de una variable) y tienen un especial significado para el compilador. El C es un lenguaje conciso, con muchas menos palabras clave que otros lenguajes. En la figura 4.1 se presenta la lista de las 33 palabras clave del mikroC. También las palabras relevantes que se refieren al SFR están definidas como variables locales y representan palabras reservadas que no pueden ser redefinidas (por ejemplo TMRO, PCL, etc)

asm	**double**	**long**	**typedef**
auto	**else**	**register**	**union**
break	**enum**	**return**	**unsigned**
case	**extern**	**short**	**void**
char	**float**	**signed**	**volatile**
const	**for**	**sizeof**	**while**
continue	**goto**	**static**	
default	**if**	**struct**	
do	**int**	**switch**	

Figura 4.1

Tabla con 33 palabras clave utilizadas por mikroC

4.5.2 IDENTIFICADORES

Identificador: es un nombre con el que se hace referencia a una función o al contenido de una zona de la memoria (variable). En ANSI C se siguen las siguientes reglas para determinar un identificador:

1. Un ***identificador*** se forma con una secuencia de ***letras*** (minúsculas de la ***a*** a la ***z***; mayúsculas de la ***A*** a la ***Z***; y ***dígitos*** del ***0*** al ***9***).

2. El carácter ***subrayado*** o ***underscore*** (_) se considera como una letra más.

3. Un identificador no puede contener espacios en blanco, ni otros caracteres distintos de los citados, como por ejemplo (*,;.:-+, etc.).

4. El primer carácter de un identificador debe ser siempre una letra o un (_), es decir, no puede ser un dígito.

5. El compilador mikroC no hace distinción entre letras mayúsculas y minúsculas. Así, Masa **es** considerado como un identificador equivalente de **masa** y de **MASA**.

6. ANSI C permite definir identificadores de hasta 31 caracteres de longitud, se considera la misma regla para mikroC. Ejemplos de identificadores válidos son los siguientes:

Tiempo, distancia1, caso_A, PI, velocidad_de_la_luz

Por el contrario, los siguientes nombres no son válidos.

1_valor, tiempo-total, dolares$, %final

En general es muy aconsejable ***elegir los nombres*** de las funciones y las variables de forma que permitan conocer a simple vista qué tipo de variable o función representan, utilizando para ello tantos caracteres como sean necesarios.

.

4.5.3 CONSTANTES

En C existen distintos tipos de constantes:

Constante entera.

Son valores numéricos enteros. Se permiten constantes decimales (base 10), *octales* (números enteros en base 8) y *hexadecimales* (base 16), binarios (base 2). En ausencia de cualquier sufijo primordial, el tipo de dato de un entero constante es derivado de su valor.

Sufijos Long y unsigned

El sufijo *L* (o *l*) anexado a cualquier constante forza a la constante a ser representada como un *long*. En forma similar, el sufijo *U* (o *u*) forza a la constante a ser *unsigned* se pueden utilizar ambos sufijos *L* y *U* en la misma constante en cualquier orden o caso: *ul*, *Lu*,*UL*, etc.

En la ausencia de algún sufijo (*U*, u, *L* o *l*), la constante es asignada al más pequeño de los siguientes tipos que pueden acomodar su valor: *short, usigned short, int, unsigned int, long int, unsigned long int*.

De otra manera:

Si la constante tiene el sufijo *U* o *u*, el tipo de dato será alguno de los siguientes en que pueda acomodar su valor: *unsigned short*, *unsigned int*, *unsigned long int*.

Si la constante tiene los sufijos *U* o *L*, su datos será del tipo *unsigned long int*.

Constante decimal

Las constantes decimales van desde -2147483648 hasta 429 496 7295. Las constantes decimales no deben iniciar con un valor cero. Una constante entera que inicia con cero es interpretada como una constante octal.

En la ausencia de algún sufijo el tipo de dato de la constante decimal es derivada de su valor actual como se muestra a continuacion:

< -2147483648	**Error: Out of range!**
-2147483648 .. -32769	long
-32768 .. -129	int
-128 .. 127	short
128 .. 255	unsigned short
256 .. 32767	int
32768 .. 65535	unsigned int
65536 .. 2147483647	long
2147483648 .. 4294967295	unsigned long
> 4294967295	Error: Out of range!

Constante hexadecimal.

Estas constantes empiezan con 0x(o 0X) de la misma manera en la ausencia de un sufijo el tipo de dato de una constante hexadecimal se

deriva de su valor de acuerdo a las reglas presentadas anteriormente. Por ejemplo, 0xc367 será tratada como un unsigned int.

Constante binaria.

Todas las contantes que empiecen con 0b o 0B son tomadas como binarias. En ausencia de un sufijo primordial, el tipo de dato una constante binaria es derivada de su valor, de acuerdo a las reglas presentadas anteriormente, por ejemplo el número 0b11101 será tratado como short.

Constante Octal.

Todas las constantes que inicien con cero serán tomadas como tal. Si una constante Octal contiene los dígitos no permitidos ocho o nueve, será reportado un error. En la ausencia de algún sufijo, el tipo de dato de una constante Octal es derivada de su valor refiriéndose a las reglas presentadas anteriormente. Por ejemplo, el número 0777 será tratado como *int*.

Constante de punto flotante.

Una constante de punto flotante consiste de:

- Entero decimal.
- Punto decimal
- Fracción decimal
- E o e y un entero con signo para un exponente(opcional)
- Un sufijo tipo: *F* o *f* o *l* o *L* (opcional).

Las constantes de tipo flotante negativas se toman como positivas con el operador unario menos (-) como prefijo. Las constantes de punto flotante tienen el siguiente rango:

±1.17549435082E38…± 6.80564774407e38.

Constante carácter.

Cualquier carácter individual encerrado entre apóstrofos (tal como 'a', 'Y', ')', '+', etc.) es considerado por mikroC como una constante carácter, o en realidad como un *número entero pequeño* (entre 0 y 255, o entre -128 y 127, según los sistemas). Existe un código, llamado ***código ASCII***, que establece una equivalencia entre cada carácter y un valor numérico correspondiente.

Escape sequences

El carácter *backslash* (\) es utilizado para introducir un *escape sequence*, la cual habilita una representación visual de ciertos caracteres no gráficos. Uno de estos caracteres o escape es el carácter nueva línea (\n).

Un *backslash* es usado con un numero octal o hexadecimal para representar el símbolo ASCII o código de control correspondiente a ese valor; por ejemplo *'\x3F'* para el símbolo de interrogación, a continuación se enuncian los *escape sequences* utilizados por mikroC

Sequence	Value	Char	What it does
\a	0x07	BEL	Audible bell
\b	0x08	BS	Backspace
\f	0x0C	FF	Formfeed
\n	0x0A	LF	Newline (Linefeed)
\r	0x0D	CR	Carriage Return
\t	0x09	HT	Tab (horizontal)
\v	0x0B	VT	Vertical Tab
\\	0x5C	\	Backslash
\'	0x27	'	Single quote (Apostrophe)
\"	0x22	"	Double quote
\?	0x3F	?	Question mark
\O		any	O = string of up to 3 octal digits
\xH		any	H = string of hex dig- its
\XH		any	H = string of hex dig- its

Figura 4.2

Lista de los "escape sequences" utilizados por mikroC

Cadenas de caracteres.

Un conjunto de caracteres alfanuméricos encerrados entre comillas es también un tipo de constante del compilador mikroC, como por ejemplo: *"espacio", "Esto es una cadena de caracteres"*, etc.

También se puede utilizar el *blackslash* (\) para continuar una cadena de constantes al final de una línea

Ejemplo:

*"esta es una cadena *

completa de caracteres"

Constante de enumeración.

Las constantes de enumeración son identificadores definidos como

declaraciones de tipo enumeración. Los identificadores son usualmente escogidos como hegemónicos para asistir legibilidad las constantes de numeración son del tipo int. Éstas pueden ser usadas en cualquier expresión donde la constante entero es válida.

Por ejemplo:

Enum weekdays { SUN=0,MON,TUE,WED,THU,FRI,SAT};

Los identificadores (enumeradores) usados deben ser únicos dentro del alcance de la declaración del enum. Los inicializadores negativos están permitidos. Vea Enumeraciones del manual del compilador para los detalles de declaraciones del *enum*.

Constante de puntero.

Un puntero o el objeto apuntado puede ser declarado con el modificador *const.* Cualquier cosa declarada como un const, posteriormente no podrá cambiar su valor.

Que está en el rango de valores que representa de acuerdo a su tipo. Las expresiones constantes son evaluadas de la misma forma que una expresión regular.

Las Expresiones constantes.

Una expresión constante es una constante que siempre evalúa a una constante y consiste sólo de constantes (literales) o constantes simbólicas. Esta expresión es evaluada en el tiempo de compilación y deberá ser evaluada como una constante expresiones constantes pueden consistir sólo de los siguientes: literales constantes de numeración constante simples (no arreglos de constantes o estructuras),tamaño de operadores.

Las expresiones constantes no pueden contener cualquiera de los siguientes operadores, a reserva de que el operador este contenido con el operando de un operador *sizeof* : asignación, decrement, llamada función, incremento.

Una expresión constante se puede usar de cualquier forma como una constante sea legal.

4.5.4 OPERADORES

Los *operadores* son signos especiales –a veces, conjuntos de dos caracteres– que indican determinadas operaciones a realizar con las variables y/o constantes sobre las que actúan en el programa. El lenguaje C es particularmente rico en distintos tipos de operadores:

aritméticos (+, -, *, /, %),

de asignación (=, +=, -=, *=, /=),

relacionales (==, <, >,<=, >=, !=),

lógicos (&&, ||, !) y otros.

Por ejemplo, en la sentencia:

```
espacio = espacio_inicial + 0.5 * aceleracion * tiempo * tiempo;
```

Aparece un operador de asignación (=) y dos operadores aritméticos (+ y *). También los ***operadores*** serán vistos con mucho más detalle en apartados posteriores.

4.5.5 SEPARADORES

Los *separadores* están constituidos por uno o varios espacios en blanco, tabuladores, y caracteres de nueva línea. Su papel es ayudar al compilador a descomponer el programa fuente en cada uno de sus ***tokens***. Es conveniente introducir espacios en blanco incluso cuando no son estrictamente necesarios, con objeto de mejorar la legibilidad de los programas. mikroC incluye en sus separadores a los corchetes([ ]), paréntesis(()), llaves({ }), coma(,), punto y coma(;), dos puntos(:), asterisco(*), signo igual(=) y el símbolo de numero(#).

4.5.6 COMENTARIOS

El lenguaje C permite que el programador introduzca *comentarios* en los ficheros fuente que contienen el código de su programa. La misión de los comentarios es servir de explicación o aclaración sobre cómo está hecho el programa, de forma que pueda ser entendido por una persona diferente (o por el propio programador algún tiempo después). El compilador[2] ignora por completo los comentarios.

Los caracteres (**/***) se emplean para iniciar un comentario introducido entre el código del programa; el comentario termina con los caracteres (***/**). No se puede introducir un comentario dentro de otro. Todo texto introducido entre los símbolos de comienzo (**/***) y final (***/**) de comentario son siempre ignorados por el compilador. Por ejemplo:

```
variable_1 = variable_2;        */ En esta línea se asigna a
                                 variable_1 el valor
                                 contenido en variable_2/*
variable_1;
```

Los comentarios pueden actuar también como *separadores* de otros tokens propios del lenguaje C. Una fuente frecuente de errores –no especialmente difíciles de detectar– al programar en C, es el olvidarse de cerrar un comentario que se ha abierto previamente.

El lenguaje ANSI C y por ño tanto mikroC permite también otro tipo de comentarios, tomado del C++. Todo lo que va en cualquier línea del código detrás de la doble barra (//) y hasta el final de la línea, se considera como un comentario y es ignorado por el compilador. Para comentarios cortos, esta forma es más cómoda que la anterior, pues no hay que preocuparse de cerrar el comentario (el fin de línea actúa como cierre). Como contrapartida, si un comentario ocupa varias líneas hay que repetir la doble barra (//) en cada una de las líneas. Con este segundo procedimiento de introducir comentarios, el último ejemplo podría ponerse en la forma:

```
variable_1 = variable_2;        // En esta línea se asigna a
                                // variable_1 el valor
                                // contenido en variable_2
```

CUESTIONARIO.

1. ¿Cuál es la información que debe llevar cada proyecto a ejecutar en mikroC?
2. Ejecute el compilador mikroC e intente comenzar con la edición de un proyecto.
3. ¿Cuáles son los comandos para compilar un proyecto en mikroC?
4. Mencione los archivos que se generan al compilar correctamente un proyecto en mikroC.
5. ¿Cuál es el archivo que se utiliza para programar un microcontrolador?
6. ¿Qué es un tokens?
7. ¿Cúantos tipos de tokens existen en C?
8. ¿Qué es una keyword y cuáles son sus restriciones?
9. ¿Cuáles son las keywords utilizadas por mikroC?
10. ¿Qué es un identificador?
11. ¿Cuáles son las constantes utilizadas por C?
12. ¿Qué es un operador?
13. ¿Qué es un separador en lenguaje C?
14. ¿Qué es un comentario, cuál es su uso y mencione las diferentes formas de introducirlos en un código dentro del ambiente de mikroC?

CAPITULO 5
TIPOS, DECLARACIONES, FUNCIONES, OPERADORES Y SENTENCIAS

La "tipología" es uno de los pilares fundamentales en que se asienta el lenguaje C y por lo tanto mikroC sigue la misma filosofía. Se supone que existe una especie de jerarquía de tipos, desde los más simples a los más complejos, así como de las operaciones que se pueden realizar con ellos. Por ejemplo, un **int** (entero) es un tipo de dato simple; del mismo modo, las operaciones aritméticas que se pueden realizar con ellos son simples también.

Básicamente un tipo sirve para:

Determinar la correcta asignación de memoria requerida por el programa inicialmente.

Interpretar el paterno encontrado en el objeto durante los subsecuentes accesos.

En muchas situaciones los tipos sirven para asegurarse que asignaciones ilegales no se lleven a cabo mandando mensajes de error a la hora de compilar el programa.

5.1 CATEGORÍA DE TIPOS.

Existen dos categorías de tipos que se explicarán a continuación:

- Los *tipos fundamentales* representan tipos que no pueden ser separados en partes más pequeñas. Algunas veces se le refiere como tipos no estructurados. Los tipos fundamentales son void, char,int, float y doublé, junto con short, long, signed y unsigned que son variantes de los mismos.
- Los *tipos derivados* son también conocidos como tipos estructurados. Los tipos derivados incluyen punteros a otros tipos, arreglos de otros tipos, tipos de funciones, estructuras y uniones.

5.1.1 TIPOS FUNDAMENTALES.

5.1.1.1 TIPOS ARITMÉTICOS.

El especificador de tipo aritmético está elaborado para los siguientes keywords: void, char,int, float y doublé, junto con short, long, signed y unsigned.

5.1.1.2 TIPOS INTEGRALES

Los tipos char e int, junto con sus variantes, son considerados tipos de datos integrales. Las variantes son creadas usando los prefijos modificador es short, long signed y unsigned.

5.1.1.3 TIPOS DE PUNTO FLOTANTE.

Los tipos float y double , conjuntamente con las variantes long double, son considerados tipos de punto flotante. El compilador mikroC considera los tres del mismo tipo.

El punto flotante en mikroC es implementado utilizando el formato de 32 bits de Microchip AN 575 (norma IEEE 754)

En la figura 5.1 se especifican los tipos aritméticos:

Tipo	Tamaño	Rango
(unsigned) char	8-bit	0 .. 255
signed char	8-bit	- 128 .. 127
(signed) short (int)	8-bit	- 128 .. 127
unsigned short (int)	8-bit	0 .. 255
(signed) int	16-bit	-32768 .. 32767
unsigned (int)	16-bit	0 .. 65535
(signed) long (int)	32-bit	-2147483648 .. 2147483647
unsigned long (int)	32-bit	0 .. 4294967295
float	32-bit	±1.17549435082E-38 ..±6.80564774407E38
double	32-bit	±1.17549435082E-38 ..±6.80564774407E38
long double	32-bit	±1.17549435082E-38 ..±6.80564774407E38

Figura 5.1

Tipos aritméticos utilizados por mikroC

5.1.1.4 ENUMERATIONS (ENUMERACIONES).

Un tipo de enumeración de datos es usado para representación de un valor abstracto o de un conjunto discreto de valores con nombres simbólicos apropiados.

```
Enumeration
```

Sintaxis:

```
enum
etiqueta(enumeración_lista);
```

Ejemplo:

```
enum colores{rojo, verde, azul, amarillo, negro, naranja,
            blanco} c;
```

Este ejemplo muestra un único tipo integral, colores; una variable "*c*" y un conjunto de enumeradores con valores constantes enteros (rojo=1, verde=2…etc).

5.1.1.5 VOID FUNCTION (FUNCIÓN VOID)

Use la palabra clave void como un tipo de retorno de función si la función no devuelve un valor. Por ejemplo:

```
void
print_temp(char
temp)          {
Lcd_Out_Cp("Tem
perature:");
Lcd_Out_Cp(temp
);
    Lcd_Chr_Cp(223);   // degree character
    Lcd_Chr_Cp('C');
    }
```

5.1.2 TIPOS DERIVADOS

5.1.2.1 ARRAYS (ARREGLOS).

Los arreglos son la estructura tipo más comúnmente utilizada. Son una colección de datos accesados por medio de un índice.

Declaración:

Arreglo unidimensional o vector:

```
<tipo var.> <nombre var.> [T];
```

Ejemplo:

```
int days[12];
/* un arreglo que contenga el numero de dias de cada mes */
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

/* esta declaracion es identica a la anterior */
   int days[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

En caso de tener arreglos con datos tipo char, se puede utilizar la notación string corta.

Ejemplo:

```
/* las dos declaraciones son identicas: */
const char msg1[] = {'T', 'e', 's', 't', '\0'};
const char msg2[] = "Test";
```

Arreglo bidimensional o matriz:

```
<tipo var.> <nombre var.> [T1] [T2];
```

Ejemplo:

```
int matriz [5] [5];
```

Arreglo n-dimensional:

```
<tipo var.> <nombre var.> [T1] [T2]...[Tn];
```

El rango de acceso al vector siempre va desde 0 (cero) hasta (Tn - 1).

Ejemplo:

```
int a[3][2][4]; /* arreglo de tres dimenciones tamaño 3x2x4
*/
```

También se puede inicializar una matriz multidimensional con el conjunto apropiado de valores. Ejemplo:

```
int a[3][2] = {{1,2}, {2,6}, {3,7}};
```

5.1.2.2 PUNTEROS

Declaración y uso de punteros simples:

Un puntero es una variable que contiene una dirección de memoria, usualmente la dirección de otra variable. Considérese el siguiente ejemplo:

```
void fn( )
{
int i;
int *pI;
pI = &i; /* pI ahora apunta a i */
*pI = 10; /* asignamos 10 a i */
}
```

Las variables tipo puntero son normalmente declaradas como cualquier otra variable, excepto por la adición del caracter unario * (asterisco). En una expresión, el unario **&** significa "La dirección de" y el unario ***** significa "La dirección que apunta ó Contenido de". Entonces se pudiese leer la primera instrucción como "*asigne la dirección de i en pI*". La segunda asignación es "*asigne 10 en la dirección que apunta pI".*

Observese lo que sucede en memoria en las figuras 5.2 y 5.3:

Asignando la dirección de i en pI:

pI = &i;

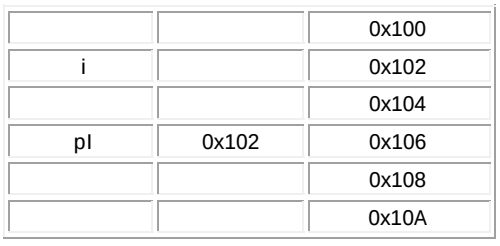
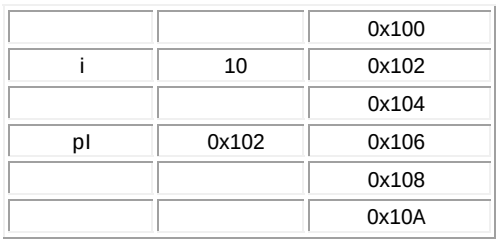


Figura 5.2

Asignacion de dirección i en PI a nivel memoria.

*pI = 10;



		0x100
i	10	0x102
		0x104
pI	0x102	0x106
		0x108
		0x10A

Figura 5.3

Asignación de un valor en la dirección que apunta el apuntador

5.1.2.3 PUNTERO NULO

Un puntero nulo asegura tener un valor diferente a cualquier otro puntero valido usado en un programa.

Ejemplo:

```
int *pn = 0; /* este es un punter nulo */
int *pn = NULL; /* esta es una declaracion equivalente */
```

5.1.2.4 ESTRUCTURAS

Una estructura es un tipo de dato definido por el usuario, al utilizar una estructura se tiene la habilidad para definir un nuevo tipo de dato considerablemente más complejo que los tipos comúnmente utilizados en C. Una estructura es una combinación de varios tipos de datos previamente definidos, incluyendo otras estructuras que hayamos definido previamente. Una definición simple es, "una estructura es un grupo de datos relacionados en una forma conveniente al programador y/o al usuario del programa".

Una estructura es definida y declarada, de la siguiente manera:

```
struct etiqueta{lista de variables de la estructura};
```

Ejemplo:

```
/* estructura definiendo un punto: */
struct Dot {float x, y;};

/* estructura definiendo un circulo: */
struct Circle {
double r;
```

```
struct Dot center;
} o1, o2; /* declara variables o1 y o2 of tipo circle */
```

Para tener acceso a los miembros de estructuras se utilizan los siguientes operadores.

```
. (period)
-> (right arrow)
```

Ejemplo:

```
Struct Dot *ptr = p;
p.x = 4;        //declara un apuntador a la estructura p
ptr-> x = 4;          // acceso directo a miembro x
ptr->  es  identico  a  (*ptr).x.  //acceso  indirecto  a
                                    //miembro x
```

5.1.2.5 UNIONS (UNIONES)

Una unión tiene las mismas características que una estructura, la diferencia es que solo uno de sus miembros esta activo en alguna de la ejecuciones de un programa.

Una unión se declara de la misma forma que una estructura, solo que la keyword utilizada es union en vez de struct.

```
union etiqueta{lista de variables de la estructura};
```

A diferencia de los miembros de estructuras, el valor de sólo uno de los miembros de unión puede ser almacenado a la vez.

Ejemplo:

```
union myunion {     // union tag is 'myunion'
int i;
double d;
char ch;
} mu, *pm = &mu;
```

El identificador mu, del tipo union myunion, puede tomar el valor de 2 bytes de un int, 4 bytes de un double, o un byte de un char, pero solo uno de ellos a la vez.

Para tener acceso a los miembros de uniones se utilizan los siguientes operadores.

. (period)

-> (right arrow)

El siguiente ejemplo hace referencia al ejemplo anterior:

```
mu.d = 4.016;
Lcd_Out_Cp(FloatToStr(mu.d));// OK: displays mu.d = 4.016
Lcd_Out_Cp(IntToStr(mu.i));  // peculiar result

pm->i = 3;
Lcd_Out_Cp(IntToStr(mu.i));  // OK: displays mu.i = 3
```

Ejemplo 2

```
Union spot (int x,y;) p;
p.x = 4;
display(p.x);//  esto  es  válido.  Muestra  el  valor  del
             //miembro x
display(p.y);// esto es invalido.
p.y = 7;
display(p.y);// esto es valido. Muestra el valor del
             //miembro y
```

En este ejemplo se hace hincapié en una llamada no valida en una union para desplegar un numero ya que mikroC no acepta una unión con un valor desconocido.

5.1.2.6 BIT FIELDS (CAMPO DE BITS).

Los campos de Bits pueden ser declarados sólo en estructuras. Se declara una estructura normalmente, y se asignan campos individuales como esto (los campos precisan ser unsigned):

```
struct etiqueta { unsigned bitfield-declarator-list; }
```

Aquí, *la etiqueta* es un nombre opcional de la estructura; *Bitfield-declarator-list* es una lista de campos de bits. Cada componente identificador requiere de dos puntos y su ancho de bits sean explícitamente especificados. El ancho total de todos los componentes no puede exceder dos bytes (16 bits).

Como un objeto, la estructura de campos de bits toma dos bytes. Los campos individuales están empaquetados dentro de dos bytes de derecha a izquierda. En *bitfield-declarator-list*, se puede omitir el identificador (s) para crear un "relleno" artificial, así pasando por alto bits irrelevantes.

Por ejemplo, si se necesitan manipular sólo los bits 2–4 de un registro como un bloque, se puede crear una estructura:

```
struct {
unsigned     : 2,     // brincar bits 0 y 1, sin identificador
mybits       : 3;     // bits relevantes 2, 3, and 4
// Bits 5, 6, and 7 are implicitly left out
} myreg;
```

Aquí se muestra un ejemplo:

```
typedef struct {
                    prescaler    : 2;
                    timeronoff : 1;
                    postscaler : 4;
                    }
                    mybitfield;
```

El cuál declara un tipo estructurado llamado `mybitfield` conteniendo tres componentes: `Prescaler` (los bits 0 y 1), `timeronoff` (el bit 2), y `postscaler` (los bits 3, 4, 5, y 6).

5.1.2.7 ACCESO AL BIT FIELDS(CAMPO DE BITS).

Los campos de bits pueden ser accesados en la misma forma como los miembros de una estructura. Se puede utilizar el operador directo e indirecto (". " y " –>"). Por ejemplo, se podría trabajar con el tipo `mybitfield` previamente declarado como se muestra en la siguiente página:

```
// Declare a bit field TimerControl:
mybitfield TimerControl;


void main()
{
    TimerControl.prescaler= 0;
    TimerControl.timeronoff = 1;
    TimerControl.postscaler = 3;
    T2CON = TimerControl;
}
```

A continuación se ejemplifica un ejercicio manejando un puerto.

```
Struct Port {led_0:1;
             other leds:7;
            }PortA;
```

Accesando al campo de bits.

```
PortA.led0=1;
PortA.other leds=0;
```

El código del último ejemplo enciende el LED0 y apaga el resto de los demás.

5.1.2.8 DECLARATIONS (DECLARACIONES).

La declaración introduce uno o varios nombres a un programa da a saber al compilador qué nombre le representa, cuál es su tipo, cuales son las operaciones permitidas con él, etc.

El rango de objetos que pueden estar declarados incluye:

- ✓ macros del preprocesador
- ✓ símbolos de sentencia
- ✓ arreglos de otros tipos
- ✓ miembros de unión
- ✓ miembros de estructura
- ✓ estructuras, uniones, y etiquetas de enumeración
- ✓ tipos
- ✓ funciones
- ✓ constantes
- ✓ variables

5.1.2.9 TYPEDEF SPECIFIER(ESPECIFICADOR TYPEDEF)

Specifier *typedef* introduce un sinónimo para un tipo especificado. se pueden usar declaraciones del *typedef* para construir nombres más cortos o más significativos para los tipos ya definido por el lenguaje o para los tipos que previamente se han declarado. no se puede utilizar al *typedef specifier* dentro de una definición de función.

En una declaración el typedef empieza en la primera posición:

```
typedef <type-definition> synonym;
```

La palabra clave `typedef` asigna el sinonimo a <type-definition> el sinonimo necesita ser un identificador valido.

La declaración de especificador `typedef` no introduce un objeto o función de un tipo dado, sino que más bien daun nombre nuevo a un tipo dado. Es decir, la declaración del `typedef` es idéntica para la declaración "normal", pero en lugar de objetos, declara tipos. Es común nombrar identificadores personalizados de tipo que inicien con letra mayúscula – ésto es estrictamente requerido por C.

Por ejemplo:

```
// declarar un sinonimo para "unsigned long int":
typedef unsigned long int Distance;

// ahora, el sinonimo "Distance" puede ser usado
//como un identificador tipo:
Distance i;   //declara la variable i como unsigned long int
```

5.2 ASM DECLARATION (DECLARACION ASM)

En mikroC se permite código ensamblador en el código fuente por medio de declaración asm. No se pueden destinar números direcciones absolutas para variables SFR o GPR en las instrucciones del ensamblador.

Se puede ingresar el bloque de código ensamblador por medio de la palabra clave *asm* (o _asm, o __asm).

```
asm {
     block of assembly instructions
     }
```

El siguiente ejemplo encienderia el dos leds en las terminales B0 y B1 del PORTB.

```
asm {
  MOVLW 3
  MOVWF PORTB
}
```

Si se quieren utilizar variables de C en el bloque asm, se debe asegurar de inicializarlas correctamente dentro del cuerpo del código de C. Es decir, por ejemplo en el siguiente código el enlazador no reconocerá la variable myvar:

```
unsigned myvar;
void main()
{
  asm {
       MOVLW 10    // just a test
       MOVLW test_main_global_myvar_1
       }
}
```

Para corregir el error solo se debe adicionar arriba del bloque asm la variable que está siendo usada y así el enlazador sabrá a qué variable se hace referencia.

```
myvar := 0;
```

nota: el compilador mikroC no verifica si se está utilizando el banco apropiado para la variable, esto se hace manualmente en código ensamblador.

5.3 FUNCIONES.

Se puede decir que una función es una parte de un programa (subrutina) con un nombre, que puede ser invocada (llamada a ejecución) desde otras partes tantas veces como se desee. Un bloque de código que puede ser ejecutado como una unidad funcional. Opcionalmente puede recibir valores; se ejecuta y puede devolver un valor. Desde el punto de vista de la organización, podemos decir que una función es algo que permite un cierto orden en una maraña de algoritmos. Como resumen de lo anterior podemos concluir que el uso de funciones se justifica en dos palabras: organización y reutilización del código.

5.3.1 DECLARACIÓN DE UNA FUNCIÓN.

```
type function_name(lista de declaración de parametros);
```

Ejemplo:

```
int max(int x, int y);
```

El *function_name* debe ser un identificador válido. Este nombre se usa para llamar a la función dentro del cuerpo del programa. El *type (tipo)* representa el tipo de resultado de función, y puede ser cualquier tipo estándar o creado por el usuario. Para funciones que no devuelven valor, se debe usar el *tipo* void. El *tipo* puede omitirse en declaraciones globales de función, y la función asumirá el tipo int por defecto.

5.3.2 DEFINICIÓN DE LA FUNCIÓN.

La definición de función consta de su declaración y un cuerpo de función. El cuerpo de función es técnicamente un bloque una secuencia de definiciones locales y declaraciones adjuntas dentro de llaves { }. Todas las variables declaradas dentro de cuerpo de función son locales a la función, o sea tienen extensión de función.

La función misma debe estar definida sólo dentro de la extensión del archivo. Esto quiere decir que las declaraciones de función no pueden estar anidadas.

Para devolver el resultado de la función, se debe usar la declaración `return.` Las declaraciónes en funciones de tipo `void` no no retornan un parámetro como resultado, por lo tanto se puede omitir la declaración `return` por completo si es la última declaración en el cuerpo de función.

Ejemplo:

```
/* la funcion max retorna el argumento de mayor valor : */

int max(int x, int y)
{
  return (x>=y) ? x : y;
}
```

En el programa principal se llama a la función de la sig forma:

```
…
 Int c;
 c = max(9,3);
```

c = 9

5.4 OPERADORES.

El compilador mikroC reconoce a los siguientes operadores:

- Operadores Aritméticos
- Operadores de Asignación
- Operadores Bitwise
- Operadores Lógicos
- Operadores de Referencia / Indirectos
- Operadores Relacionales
- Selectores de Miembro de Estructura
- Operador Coma ,
- Operador Condicional ?:
- Operador del Subíndice del Arreglo[]
- Operador de Llamada de Función ()
- Operador Sizeof
- Operador del Preprocesador # y ##

Hay 15 categorías de precedencia, algunos de cuales contiene sólo a un operador. Los operadores en la misma categoría tienen precedencia igual con cada uno de dicha categoría.

En la figura 5.4 se suman todos los operadores del compilador mikroC.

En donde para los operadores duplicados que aparecen en figura, el primer caso es unario, el segundo binario. Cada categoría tiene una regla de asociatividad: de izquierda a derecha o de derecha a izquierda.

A falta de paréntesis, estas reglas determinan el agrupamiento de expresiones con operadores de igual precedencia.

Precedencia	Operando	Operadores	Asociatividad
15	2	`()  []  .  ->`	izquierda a derecha
14	1	`!  ~  ++  -- + - *` `&  (type)  sizeof`	Derecha a izquierda
13	2	`*  /  %`	izquierda a derecha
12	2	`+  -`	izquierda a derecha
11	2	`<<  >>`	izquierda a derecha
10	2	`<  <=  >  >=`	izquierda a derecha
9	2	`==  !=`	izquierda a derecha
8	2	`&`	izquierda a derecha
7	2	`^`	izquierda a derecha
6	2	`|`	izquierda a derecha
5	2	`&&`	izquierda a derecha
4	2	`||`	izquierda a derecha
3	3	`?:`	izquierda a derecha
2	2	`= *= /= %= += -=` `&= ^= |= <<= >>=`	Derecha a izquierda
1	2	`,`	izquierda a derecha

Figura 5.4

Niveles de precedencia utilizados por mikroC

5.4.1 OPERADORES ARITMÉTICOS

Los operadores aritméticos se asocian de izquierda a derecha como se muestra en la figura 5.5.

operador	operacion	Precedence
+	adicion	12
-	Substraccion	12
*	multiplicacion	13
/	Division	13
%	Retorna el residuo de una division entera (can- not be used with floating points)	13
+ (unary)	el operador suma unario no afecta el operando	14
- (unary)	El operador unario menos cambia el signo del operando	14
++	Incrementa sumando uno al valor del operando	14
--	Decrementa substrayendo por uno el valor del operando	14

Figura 5.5

Precedencia de operadores aritméticos.

5.4.2 OPERACIONES ARITMÉTICAS BINARIAS

La division de dos enteros retorna un entero, mientras que el resuido es omitido.

Por ejemplo:

```
7 / 4;  // igual a 1
7 * 3 / 4;     // igual a 5


/* pero: */
7. * 3./ 4.;   // es igual a 5.25 por se trabaja con floats
```

El operando residuo % trabaja solo con enteros; el signo del resultado es igual al signo del primer operando:

```
/* por ejemplo: */
9 % 3;   // equals 0
7 % 3;   // equals 1
-7 % 3;  // equals -1
```

Se pueden utilizar los operandos aritméticos para manipular caracteres :

```
'A' + 32;              // igual 'a'(solo ASCII)

'G' - 'A' +   'a';      // igual 'g'(ambos ASCII y EBCDIC)
```

5.4.3 OPERADORES ARITMÉTICOS UNARIOS

Los operadores unarios ++ y -- son los únicos operadores que en C pueden tener los prefijos (e.g. ++k, --k) o postfijos (e.g. k++, k--).

Estando usando como prefijo, los operadores ++ y -- (el preincremento y el predecremento) sumar o sustraer uno, del valor de operando *antes de la evaluación*. Yal usar como el sufijo, los operadores ++ y -- , sumar o sustraier uno del valor del operando *después de la evaluación.* Por ejemplo:

```
int j = 5; j = ++k;
/* k = k + 1, j = k, lo cual nos da j = 6, k = 6 */

int j = 5; j = k++;
/* j = k, k = k + 1, lo cual nos da j = 5, k = 6 */
```

5.4.4 OPERADORES RELACIONALES

Los operadores relacionales se utilizan para para hacer pruebas de igualdad o inigualdad de expresiones. Si la expresión es verdadera retorna un 1; de otra manera retorna 0.

Todos los operadores relacionales se asocian de izquierda a derecha como se muestra en la figura 5.6.

Figura 5.6

Operadores relacionales y su precedencia

Operador	Operación	Precedencia
==	igual	9
!=	no igual	9
>	Mayor que	10
<	Menor que	10
>=	Mayor que o igual	10
<=	Menor que o igual	10

Siempre tener presente que los operadores relacionales devuelven ya sea 0 o 1. Considérese a los siguientes ejemplos:

```
8==13>5     //return 0: 8==(13>5), 8==1, 0
14>5<3// return 1: (14>5)<3, 1<3, 1
a<b<3 // return 1: (a>b)<5,(0 o 1)<5, 1
```

5.4.5 BITWISE OPERATORS (OPERADOR A BITS)

Use el operador Bitwise para modificar los bits individuales de operandos numéricos. Los operadores Bitwise se asocian de izquierda a derecha como lo muestra la figura 5.7. La única excepción es el operador complemento del bitwise ~ que se asocia de derecha a izquierda.

Operator	Operation	Precedence
&	bitwise AND; Devuelve 1 si ambos bits son 1, de otra manera regresa 0	9
|	bitwise (inclusive) OR; Devuelve 1 si uno o ambos bits son 1, de otra manera regresa 0	9
^	bitwise exclusive OR (XOR); retorna 1 si los bits son complementarios, de otra manera retorna 0	10
~	bitwise complement (unary); invierte cada bit	10
>>	bitwise shift left; recorre los bits a la izquierda.	10
<<	Bitwise shift right; recorre los bits a la derecha.	10

Figura 5.7

Operadores bitwise y su pecedencia.

Los operadores Bitwise &, |, y ^ realizan operaciones lógicas en pares correctos de bits de sus operandos. Por ejemplo:

```
0x1234 & 0x5678;  /* igual 0x1230 */

/* por que ..
        0x1234 : 0001 0010 0011 0100
        0x5678 : 0101 0110 0111 1000
        --------------------------------
        & : 0001 0010 0011 0000
```

```
.. esto es, 0x1230 */

/* similarmente: */

            0x1234 | 0x5678;     /* igual 0x567C */
            0x1234 ^ 0x5678;     /* igual 0x444C */
            ~ 0x1234; /* igual 0xEDCB */
```

Los operadores binarios << y >> recorren los bits del operando izquierdo a un número de posiciones especificadas por el operando derecho, hacia la izquierda o la derecha, respectivamente. El operando derecho tiene que ser positivo.

Con corrimiento a la izquierda (<<), los bits que se localicen mas a la izquierda son descartados, y los bits "nuevos" de la derecha le son asignados ceros.

```
000001 << 5;       /* igual a 000040 */
0x3081  << 4;      /* igual a 0x8010, sobre flujo*/
```

Con corrimiento a la derecha (>>), los bits que se localicen mas a la derecha son descartados, y los bits " nuevos " de la izquierda le son asignados ceros (en el caso de operadores unsigned), o el valor del bit de signo (en el caso de un operando signado).

```
0xFF56    >> 4;   /*igual a 0xFFF5*/
0Xff56u   >> 4;   /*igual a 0x0FF5*/
```

5.4.6 OPERADORES LÓGICOS

Figura 5.8

Operadores lógicos y sus niveles de precedencia

Operador	Operación	Precedencia
&&	logical AND	5
||	logical OR	4
!	logical negation	14

El operador booleano AND lógico (&&) devuelve un 1 sólo si ambas expresiones evaluadas son verdaderas, de otra manera devuelve 0. Si la primera expresión es falsa, la segunda expresión no es evaluada. Por ejemplo:

```
a > b && c < d;  // se lee como: (a>b) && (c<d)
// if (a>b) es falsa (0), (c<d) no será evaluada
```

El operador lógico OR (||) devuelve 1 si cualquiera de las expresiones evaluadas son verdaderas, de otra manera devuelve 0. Si la primera expresión evaluada es verdadera, la segunda expresión no será evaluada. Por ejemplo:

```
a && b || c && d;  // se lee como: (a && b) || (c && d)
// Si(a&&b) verdadero (1), (c&&d) no será evaluado.
```

5.4.7 CONDITIONAL OPERATOR ? :

El operador condicional ? : es el único operador ternario en C. La sintaxis del operador condicional es:

```
expression1? expression2: expression3
```

Expression1 se evalúa primero. Si su valor es verdadero, entonces *expression2* se evalúa y *expression3* es ignorada. Si *expression1* es evalúada como falso, entonces *expression3* se evalúa y *expression2* es ignorado. El resultado será el valor ya sea de *expression2* o *expression3*.

Ejemplo:

```
(i>0) ? led1 = 1: led2 = 1;
```

En este ejemplo si la variable i es mayor que cero entonces el led 1 encendera , también el led2 sera colocado a alto.

5.4.8 SIZEOF OPERATOR

El operador de prefijo unario `sizeoff` retorna una constante entera que da el valor del tamaño en bytes que ocupa la memoria de acuerdo al tipo de operando.

```
sizeof(char) /*     retorna     1     */
sizeof(int)  /*     retorna     2     */
sizeof(unsigned long) /* retorna 4 */
```

5.4.9 COMMA EXPRESSIONS(OPERADOR DE SECUENCIA)

Una de las características específicas de C es que permite usar a la *coma* como un operador de secuencias para crear al denominado comma expressions o secuencias. La Comma expression es una lista de expresiones delimitada por comas que es formalmente tratada como una expresión sola así es que puede ser usado en lugares donde una expresión es esperada.

Se declara de la siguiente forma

```
expresion1, expresion2, …expresión_n;
```

En la expresión anterior el resultado se evalúa de izquierda a derecha en cada expresión, dando el valor y tipo de resultado de la expresion_n. el resultado de las otras expresiones es descartado.

Ejemplo:

```
result = (a = 5, b /= 2, c++);
/*  retorna  el  valor  del  preincremento  la
variable c, pero tambien inicializa a, divide
b por 2, e incrementa c */


result = (x = 10, y = x + 3, x--, z -= x * 3 - --y);
/* retorna el calculo del valor de la variable z,
y también calcula x y y */
```

5.5 STATEMENTS (SENTENCIAS)

Las sentencias especifican el flujo de control de como un programa se ejecuta. En ausencia de saltos específicos y declaraciones de selección, las sentencias son ejecutadas secuencialmente en el orden de aparición en el código fuente.

Las sentencias generalmente son divididas en:

- Sentencias Etiquetadas
- Sentencias de Expresión
- Sentencias de Selección
- Sentencias de Iteración (loops)
- Sentencias de Salto
- Sentencias Compuestas (Bloques)

5.5.1 SENTENCIAS ETIQUETADAS

Cada Sentencia en un programa debe ser etiquetada. La etiqueta es un identificador agregado antes de la sentencia como se ejemplifica a continuación:

```
eqtiqueta_identificador : sentencia;
```

Una sentencia debe ser etiquetada por dos razones:

1. El identificador de la etiqueta sirve de un objetivo para el comando GOTO incondicional,

```
Loop: Display (message); //este es un loop infinnito que
Goto loop;               //llama a la funcion Display
```

2. El identificador de la etiqueta sirve como objetivo para la sentencia `switch.` Con este propósito, sólo las sentencias etiquetadas `case` y `default` son usadas:

```
case constant-expression : statement
default : statement
```

5.5.2 SENTENCIAS DE EXPRESIÓN

Cualquier expresión seguida por un punto y coma forma una sentencia de expresión:

```
expression;
```

5.5.3 SENTENCIAS DE SELECCIÓN

Existen dos tipos de sentencias de selección en C: la sentencia if y la sentencia switch

5.5.3.1 SENTENCIA IF ... ELSE

En su forma abreviada, cuando no existe la cláusula **else**, esta sentencia permite escoger entre ejecutar o no una sentencia, en función del resultado de una expresión lógica. En su forma ampliada, cuando la cláusula **else** está presente, permite escoger entre dos opciones alternativas.

Sintaxis

```
if ( <condición> )  <sentencia1>;
[ else  <sentencia2>; ]
```

La forma generalmente más utilizada es:

```
if (<condicion>) {
   <sentencia1>;
}
else  {
   <sentencia2>;
}
```

Ejemplos

```
if (salida == 'S') break;
```

Otro ejemplo:

```
if (a > b)
   z = a;
```

```
else
   z = b;
```

5.5.3.2 SENTENCIA SWITCH

Se trata de una sentencia condicional multi-salida en la que las decisiones se toman en función de un valor numérico entero de entre una serie de opciones posibles. Puede existir una cláusula por defecto o bien no adoptarse ninguna acción.

Sintaxis

```
switch ( <expresion> ) {
  case <const1> : <sentencia1>; [break;]
  case <const2> : <sentencia2>; [break;]
    .
    .
    .
  case <constN> : <sentenciaN>; [break;]
 [default : <sentenciaD>; ]
}
```

Descripción

La sentencia **switch** comprueba cuando una expresión <expresion> entre parentesis (que se traduce en un valor numérico) coincide con alguno de una serie de valores enteros constantes y diferentes (<constX>). En cuyo caso, se ejecuta un bloque de código específico <sentencia>. En caso de estar presente la cláusula opcional **default** y no existir concordancia con ninguno de los valores anteriores, se ejecuta una sentencia por defecto (<sentenciaD>).

Ejemplo:

```
Switch (input) {
   Case 1 : led1 = 1;
   Case 2 : led2 = 2
   Case 3 : led3 = 3
   Default : led7 = 1
}
```

El código anterior enciende un led dependiendo de los valores de entrada. Si el valor es diferente a alguno de los de la lista entonces se ejecuta la sentencia establecida por la sentencia default.

5.5.4 SENTENCIAS DE ITERACIÓN

Las **sentencias de iteración** permiten repetir una sentencia o conjunto de ellas. Es lo que se denomina ejecutar un *ciclo*. En C existen tres formas de iteraciones: los *ciclo*s **while**; **do…while** y **for**.

5.5.4.1 *CICLO* WHILE

La sentencia **while** permite ejecutar repetidamente un bloque de código mientras se cumpla una determinada condición que es chequeada antes de cada iteración.

Sintaxis

```
while ( <condicion> ) <sentencia> ;
```

Descripción

La sentencia **while** ejecuta iterativamente el *ciclo* definido por el bloque de código <sentencia> siempre que el valor devuelto por la expresión <condición> (que debe estar entre parentesis) sea **cierto**.

Ejemplo:

```
int s, i;
s = i = 0;
while (i < 6){
s = s + 2;
i = i + 1;
}
```

Este código adiciona el numero 2 a la variable s 6 veces. Al término de ejecución del código la variable s tendrá un valor de 12.

5.5.4.2 *CICLO* DO...WHILE

La sentencia **do ... while** permite ejecutar repetidamente un bloque de código mientras se cumpla una determinada condición que es chequeada después de cada iteración.

Sintaxis

```
do <sentencia> while ( <condición> );
```

Descripción

La sentencia **do** ejecuta repetidamente el *ciclo* definido por el bloque de código <sentencia> hasta que la sentencia de control <condición> devuelve el valor **falso**.

Puesto que el control se evalúa después de cada ejecución del *ciclo*, resulta que este se ejecuta al menos una vez, aunque <condición> devuelva el valor **falso** desde el principio (si requiere que el *ciclo* no se ejecute ninguna vez, es mejor utilizar **while**).

La forma más genérica de la expresión suele ser:

```
do {
    <sentencia> ;
   }  while ( <condición> );
```

Ejemplo:

```
int s, i;
s = i = 0;
do {
s = s + 2;
i = i + 1;
```

} **while** (i < 7);

Este código adiciona a la variable s el numero 2 hasta q la variable i es igual a 7, es decir al final la variable s tendrá un valr de 14.

5.5.4.3 SENTENCIA FOR

Esta sentencia permite realizar un *ciclo* repetidamente en base a una condición, la cual suele estar basada en el valor de un contador que se actualiza después de cada ejecución del *ciclo*.

Sintaxis

```
for ( [<inicio>] ; [<condicion>] ; [<incremento>] ) <sentencia>
```

Descripción

La utilización más común de cuerpo de la sentencia suele adoptar la forma:

```
for ( <inicio> ; <condición> ; <incremento> )
{
  <sentencia>;
}
```

Ejemplo

```
for(i=0; i<n; i++) { /* sentencias del ciclo */ }
```

- <inicio> inicia las variables para el *ciclo* antes de la primera iteración. Puede ser una expresión o una declaración.

- <condición> Debe ser una expresión relacional (devuelva un valor lógico). Es comprobada antes de la primera ejecución del bloque <sentencia>, que es ejecutado repetidamente hasta que <condición> sea **falso**, por lo que si <condición> devuelve falso desde el principio, el *ciclo* **no** se ejecuta nunca.

- <incremento>. Después de cada iteración del *ciclo*, <incremento> incrementa un contador de *ciclo*; en consecuencia j++ es funcionalmente equivalente a ++j.

- <sentencia> es el *ciclo* de **for**; sentencia o bloque de código que se ejecuta iterativamente. El ámbito de cualquier identificador declarado dentro él se limita al final de la sentencia de control.

Ejemplo:

```
for (s = 0, i = 0; i < 5; i++) {
                                    s += 2;
                                }
```

Este codigo adiciona el numero 2 a la variable *s* al termino de la ejecución *s* tendra un valor de 20.

5.5.5 SENTENCIAS DE SALTO

Las sentencias de salto permiten transferir el control del programa de forma incondicional. Existen cuatro de estas sentencias: **break** , **continue** , **goto** y **return** .

5.5.5.1 SENTENCIA BREAK

La sentencia **break** se usa para salir de forma incondicional de los *ciclo*s **do**, **for** y **while**, así como de las sentencias **switch** de multidecisión. Hay que tener en cuenta que en el caso de *ciclo*s anidados, **break** hace salir del *ciclo* interior (ver nota a continuación).

sintaxis

```
break;
```

Ejemplo:

```
switch (c)
{
case 0:
cout << "Falso" << endl;
break;
default: cout << "Cierto" << endl;
}
```

5.5.5.2 SENTENCIA CONTINUE

La sentencia **continue** se utiliza en los *ciclo*s **for**, **while** y **do...while**. En los primeros el control salta al final del *ciclo*, con lo que el contador se incrementa y comienza otra comprobación. En los **while**, el control pasa inmediatamente a la condición de control.

Sintaxis

```
continue;
```

Descripción

Continue suele utilizarse cuando la lógica del *ciclo* es complicada, de forma que establecer una nueva condición y un nuevo nivel de indentación puede volver está demasiado profunda.

Ejemplo

```
void main ()
{
 for (i = 0; i < 20; i++)
{
   if (array[i] == 0)
      continue;
      array[i] = 1/array[i];
  }
}
```

5.5.5.3 SENTENCIA GOTO

Goto es una sentencia de salto incondicional dentro del ámbito de una función.

Sintaxis

```
goto <etiqueta> ;
```

Descripción

La sentencia **goto** permite transferir el control de ejecución a la etiqueta especificada por el identificador <etiqueta> (las etiquetas terminan siempre en dos puntos :). Recordar que la etiqueta debe estar en la misma función que el **goto** (el ámbito de las etiquetas se limita a la función en que son declaradas).

5.5.5.4 SENTENCIA RETURN

La sentencia **return** devuelve el control de ejecución desde la función que contiene el **return** a la rutina que la invocó; opcionalmente puede devolver un valor.

Sintaxis

```
return [ <expresion> ] ;
```

Descripción

La sentencia **return** devuelve el control de ejecución desde la función que contiene el **return** a la rutina que la invocó. Además, opcionalmente puede devolver un valor (contenido en <expresion>).

El campo <expresion> es opcional, así como el paréntesis que se suele colocar, aunque no es necesario. Si no se indica <expresion> la función no devuelve nada.

Ejemplo:

```
double sqr(double x)
{
 return (x*x);
}
```

CUESTIONARIO.

1. ¿Para qué sirve los “tipos” en mikroC?
2. ¿Cuántas categorías de tipos se definen en mikroC?
3. ¿mencione los tipos fundamentales?
4. ¿defina los tipos derivados?
5. ¿Qué es un puntero?
6. ¿Cómo se declara un puntero?
7. ¿Qué es una declaración?
8. ¿Cómo se declara un bloque de código ensamblador en mikroC?
9. ¿Qué es una función?
10. ¿Cómo se declara una función?
11. ¿Cuáles son los operadores que utiliza mikroC?
12. ¿Cómo funciona una declaración?
13. ¿Cómo se dividen las declaraciones?
14. ¿Cuál es la estructura de la sentencia if …else?
15. ¿Cuál es la estructura de la sentencia switch?
16. ¿Cuál es la estructura de la sentencia while?
17. ¿Cuál es la estructura de la sentencia do… while?
18. ¿Cuál es la estructura de la sentencia for?
19. ¿Cuál es la estructura de la sentencia break?
20. ¿Cuál es la estructura de la sentencia continue?
21. ¿Cuál es la estructura de la sentencia goto?
22. ¿Cuál es la estructura de la sentencia return?

CAPITULO 6
LIBRERÍAS DE RUTINAS Y FUNCIONES BUILT-IN.

El compilador mikroC provee de funciones built-in y librerías de rutinas, las cuales ayudan a desarrollar un proyecto en forma fácil y rapida. Las librerías con que cuenta mikroC por mencionar algunas son ADC, CAN, USART, SPI, I2C, 1-wire, LCD, PWM, RS485, Serial Ethernet, Toshiba GLCD, Port expander, serial GLCD, numeric formatting, bit manipulation,y algunas otras que viene incluidas y explicadas en el manual de mikroC.

En este capítulo se hará referencia a las librerías utilizadas en el desarrollo de proyectos básicos para la utilización del compilador en estudio.

6.1 RUTINAS DE FUNCIONES BUILT-IN

El compilador mikroC provee de un conjunto de funciones previamente incorporadas y útiles en la mayoría de proyectos. Las funciones built-in no requieren de algún encabezado de archivo para que puedan ser incluidas en el proyecto a desarrollar; estas funciones se pueden usar en cualquier parte del proyecto.

Las funciones built-in generan el código en el lugar donde se hace la llamada, la llamada no cuenta como llamada anidada. Las únicas excepciones son *Vdelay_ms* y *Delay_Cyc*.

Nota: las funciones Lo, Hi, Higuer, Highest no son implementadas automáticamente por el compilador, para hacer uso de ellas se debe incluir built_in.h dentro del proyecto.

Las funciones built-in se enuncian a continuación:

- Lo
- Hi
- Higher
- Highest
- Delay_us
- Delay_ms
- Vdelay_ms
- Delay_Cyc
- Clock_Khz
- Clock_Mhz

Descripción de funciones:

FUNCIÓN LO

Prototipo	**Unsigned short** Lo (**long** number)
Retorno	Retorna los 8 bits de mas bajo peso de un número , bits 0… 7
Descripción	La función retorna el el byte de menor peso de un número. La función no interpreta un bit patrón de un número-meramente reconoce y retorna 8 bits encontrados en el registro. Esta es una rutina inline, el código es generado en el lugar de la llamada, la llamada no cuenta como llamada anidada.
Requerimientos	Los argumentos deben ser de tipo escalar
Ejemplo	d=0x1AC30F4 tmp=Lo(d); // igual a 0xF4

FUNCIÓN HI

Prototipo	**Unsigned short** Hi (**long** number)
Retorno	Retorna los 8 bits siguientes bits de menor peso de un número, bits 8… 15
Descripción	La función retorna el siguiente byte de menor peso de un número. La función no interpreta un bit patrón de un número-meramente reconoce y retorna 8 bits encontrados en el registro. Esta es una rutina inline, el código es generado en el lugar de la llamada, la llamada no cuenta como llamada anidada.
Requerimientos	Los argumentos deben ser de tipo escalar
Ejemplo	d=0x1AC30F4 tmp=Hi(d); // igual a 0x30

FUNCIÓN HIGHER

Prototipo	**Unsigned short** Higher (**long** number)
Retorno	Retorna los 8 bits siguientes bits de mayor peso de un número, bits 16… 23
Descripción	La función retorna el siguiente byte de mayor peso de un número. La función no interpreta un bit patrón de un número-meramente reconoce y retorna 8 bits encontrados en el registro. Esta es una rutina inline, el código es generado en el lugar de la llamada, la llamada no cuenta como llamada anidada.
Requerimientos	Los argumentos deben ser de tipo escalar
Ejemplo	d=0x1AC30F4 tmp=Higher(d); // igual a 0xAC

FUNCIÓN HIGHEST

Prototipo	**Unsigned short** Hi (**long** number)
Retorno	Retorna los 8 bits siguientes bits de mayor peso de un número, bits 24… 31
Descripción	La función retorna el siguiente byte de mayor peso de un número. La función no interpreta un bit patrón de un número-meramente reconoce y retorna 8 bits encontrados en el registro. Esta es una rutina inline, el código es generado en el lugar de la llamada, la llamada no cuenta como llamada anidada.
Requerimientos	Los argumentos deben ser de tipo escalar
Ejemplo	d=0x1AC30F4 tmp=Highest(d); // igual a 0x01

FUNCIÓN DELAY_US

Prototipo	`void Delay us(const time in us);`
Descripcion	Crea un retardo por software, consiste en una constante de tiempo en microsegundos.
ejemplo	`Delay_us(10); /* Ten microseconds pause */`

FUNCIÓN DELAY_MS

Prototipo	`void Delay ms(const time in ms);`
Descripcion	Crea un retardo por software. El retardo es una constante de tiempo en milisegundos.
ejemplo	`Delay_ms(1000); /* One second pause */`

FUNCIÓN VDELAY_MS

Prototipo	`void Vdelay ms(unsigned time in ms);`
Descripcion	Crea un retardo por software, consiste de una variable de tiempo en microsegundos
ejemplo	`pause = 1000;// ...` `Vdelay_ms(pause);// ~ one second pause`

FUNCIÓN DELAY_CYC

Prototipo	`void Delay Cyc(char Cycles div by 10);`
Descripcion	Crea un retardo basado en la frecuencia del reloj del microcontrolador. El retardo es un múltiplo de diez dependiendo del parámetro de entrada que se especifique
ejemplo	`Delay_Cyc(10);/* el retardo dura 100 cyclos de trabajo del microcontrolador */`

FUNCIÓN CLOCK_KHZ

Prototipo	`unsigned Clock Khz(void);`
Returns	Retorna la frecuencia de reloj en KHz redondeada al entero más cercano.
Descripcion	Frecuencia de reloj del dispositivo en KHz, redondeado al entero más cercano.
Ejemplo	`clk = Clock_Khz();`

FUNCIÓN CLOCK_MHZ

Prototipo	`unsigned Clock Mhz(void);`
Returns	Retorna la frecuencia de reloj en MHz redondeada al entero más cercano.
Descripcion	Frecuencia de reloj del dispositivo en MHz, redondeado al entero más cercano.
Ejemplo	`clk = Clock_Mhz();`

6.2 LIBRERÍA DE RUTINAS.

El compilador mikroC trae implícitamente un conjunto de librerías de funciones, con lo cual simplifica la iniciación y uso de los microcontroladores PIC y sus módulos. Las funciones de cada librería no requiere de algún archivo *header* (principal) para ser incluido en el programa; por lo cual dichas funciones pueden ser utilizadas de cualquier manera en los proyectos a realizar.

En este capítulo se hará referencia a librerías que se utilizan con más frecuencia en la mayoría de proyectos. Para una programación más

avanzada o utilización de módulos específicos, dependiendo del modelo de micro controlador PIC se debe consultar el manual del compilador mikroC en la sección “mikroC Libraries”.

6.2.1 LIBRERÍA ADC

El modulo ADC (convertidor analógico digital) está habilitado en un gran número de modelos de micro controladores PIC. La librería de la función Adc_Read está incluida para proveer al usuario un trabajo confortable con el módulo.

Prototipo	`Unsigned Adc_Read (char channel);`
Retorno	Retorna un valor de 10 bits de tipo unsigned de un canal específico del convertidor ADC
Descripción	Inicializa el módulo ADC interno del micro controlador PIC para trabajar con la frecuencia del reloj. El reloj determina el período de tiempo necesario para realizar la conversión analógico digital (mínimo 12 tiempos de adquisición de datos). El parámetro channel representa el canal del cual el valor analógico será adquirido. Para escoger el canal apropiado para la adquisición referirse a la documentación apropiada del PIC en uso
Requerimientos	Se requiere de un microcontrolador PIC que tenga integrado el módulo ADC. Se debe consultar la hoja de datos del dispositivo a utilizar. Antes de usar la función, se debe asegurar de configurar apropiadamente los bits TRISA para designar las terminales a utilizar como entradas. También, configurar la terminal deseada como entrada analógica y configurar el Vref (voltaje de referencia).

Ejemplo	`unsigned tmp;`
	`...`
	`tmp = Adc Read(1); /* lee el valor analógico del canal uno*/`

6.2.2 LIBRERÍA LCD (INTERFACE DE 4 BITS)

El compilador mikroC trae consigo esta librería para trabajar con displays LCD que se comunican con una interfaz de 4 bits. A continuación se enuncian las funciones que acompañan esta librería y la explicación pertinenente.

Nota: el puerto que controlara al LCD debe ser configurado como salida, antes de usar cualquiera de las siguientes funciones.

```
Lcd_Config
```

```
Lcd_Init
Lcd_Out
Lcd_Out_Cp
Lcd_Chr
Lcd_Chr_Cp
Lcd_Cmd
```

LCD _CONFIG

Prototipo	`void Lcd Config(char *port, char RS, char EN, char WR, char D7, char D6, char D5, char D4);`
Descripción	Inicializa el LCD en el puerto y terminales previamente configuradas por el usuario : parámetros RS, EN, WR, D7 ,, D4 Se necesitan tener una combinación de valores 0–7 (ejemplo 3,6,0,7,2,1,4).
Ejemplo	`void Lcd_Config(char *port, char RS, char EN, char WR, char D7, char D6, char D5, char D4);`

LCD_ INIT

Prototipo	`void Lcd Init(char *port);`
Descripción	Iinicializa el LCD en el Puerto especificado por el usuario y en las terminales configuradas por default (ver diagrama de conexiones en la figura 6.1)D7 -> PORT.7, D6 -> PORT.6, D5 -> PORT.5, D4 -> PORT.4, E -> PORT.3, RS -> PORT.2.
Ejemplo	`Lcd_Init(&PORTB);`

LCD_OUT

Prototipo	`void Lcd Out(char row, char col, char *text);`
Descripción	imprime el texto en la pantalla LCD en la fila y columna especificada(parámetros `row` y `col`)
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_Init`
Ejemplo	`Lcd_Out(1,3,"Hello!");` // imprime “hello!” en la línea 1, carácter 3.

LCD_OUT_CP

Prototipo	`void Lcd Out Cp(char *text);`
Descripción	Imprime el texto en la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las cadenas de variables y literales, pueden ser pasadas como texto
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_Init`
Ejemplo	`Lcd_Out_Cp("Here!"); // imprime "Here!" en // la posicion actual // del cursor`

LCD_CHR

Prototipo	`void Lcd Chr(char row, char col, char character);`
Descripción	Imprime caracteres en la pantalla de LCD en la fila y columna especificada (parámetros `row` y `col`). Las variables y literales, pueden ser pasadas como carácter.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_Init.`
Ejemplo	`Lcd_Out(2, 3, 'i'); // imprime 'i' en la // linea2, caracter 3`

LCD_CHR_CP

Prototipo	`void Lcd Chr Cp(char character);`
Descripción	Imprime caracteres sobre la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las variables y literales, pueden ser pasadas como carácter
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_Init`
Ejemplo	`Lcd_Out_Cp('e'); // Print 'e' at current // cursor position`

LCD_CMD

Prototipo	`void Lcd Cmd(char command);`
Descripción	Envia comandos al LCD. La lista de comandos validos se muestra en la tabla 6.1.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_Init`
Ejemplo	`Lcd_Cmd(Lcd_Clear); // limpia eldisplay LCD`

LISTA DE COMMANDOS LCD

LCD Command	Propósito
LCD_FIRST_ROW	Mueve el cursor a la primera fila
LCD_SECOND_ROW	Mueve el cursor a la segunda fila
LCD_THIRD_ROW	Mueve el cursor a la tercera fila
LCD_FOURTH_ROW	Mueve el cursor a la cuarta fila
LCD_CLEAR	Limpia el display
LCD_RETURN_HOME	Retorna el cursor a la posición inicial. Un dato desplegado que este recorrido lo retorna a su posición original. La RAM de datos de exhibición es inafectada.
LCD_CURSOR_OFF	Apaga el cursor
LCD_UNDERLINE_ON	Enciende el cursor de subrayado
LCD_BLINK_CURSOR_ON	Enciende el cursor intermitente
LCD_MOVE_CURSOR_LEFT	Mueve el cursor a la derecha sin alterar los datos exhibidos de la RAM.
LCD_MOVE_CURSOR_RIGHT	Mueve el cursor a la derecha sin cambiar los datos exhibidos de la RAM.
LCD_TURN_ON	Enciende el display LCD.
LCD_TURN_OFF	Apaga el display LCD.
LCD_SHIFT_LEFT	Recorre los datos desplegados a la izquierda del display sin afectar los datos de exhibición de la RAM
LCD_SHIFT_RIGHT	Recorre los datos desplegados a la derecha del display sin afectar los datos de exhibición de la RAM

6.2.3 LIBRERÍA LCD CUSTOM.

El compilador mikroC también trae consigo esta librería llamada librería LCD Custom, a diferencia de la librería LCD el usuario puede configurar las terminales del puerto que funcionaran como puerto de datos y cuales como puerto de control.

Nota: el puerto que controlara al LCD debe ser configurado como salida, antes de usar cualquiera de las siguientes funciones.

```
Lcd_Custom_Config
Lcd_Custom_Out
Lcd_Custom_Out_Cp
Lcd_Custom_Chr
Lcd_Custom_Chr_Cp
Lcd_Custom_Cmd
```

LCD _CUSTOM_CONFIG

Prototipo	`void Lcd Custom Config(char *data port, char RS, char EN, char WR, char D7, char D6, char D5, char D4);`
Descripción	Inicializa el LCD con puerto de datos y puerto de control con las terminales configuradas por el usuario.
Ejemplo	`Lcd_Custom_Config(&PORTD,3,2,1,0,&PORTB,2,3,4);`

LCD_CUSTOM_OUT

Prototipo	`void Lcd Custom Out(char row, char col, char *text);`
Descripción	imprime el texto sobre la pantalla LCD en la fila y columna especificada(parámetros `row` y `col`) . Ambas variables y literales pueden ser pasadas como texto.
Requerimientos	El puerto con el LCD debe ser inicializado. Ver `Lcd_config.`
Ejemplo	`Lcd_Out(1,3,"Hello!"); //imprime "hello!" en la línea //1, carácter 3.`

LCD_CUSTOM_OUT_CP

Prototipo	`void Lcd Custom Out Cp(char *text);`
Descripción	Imprime el texto en la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las cadenas de variables y literales, pueden ser pasadas como texto
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config.`
Ejemplo	`Lcd_Custom_Out_Cp("Here!"); // imprime "Here!" en // la posicion actual // del cursor`

LCD_CUSTOM_CHR

Prototipo	`void Lcd Custom Chr(char row, char col, char character);`
Descripción	Imprime caracteres en la pantalla de LCD en la fila y columna especificada (parámetros `row` y `col`). Las variables y literales, pueden ser pasadas como carácter.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config.`
Ejemplo	`Lcd_Custom_Chr(2, 3,'i'); // imprime 'i' en la // linea2, caracter 3`

LCD_CUSTOM_CHR_CP

Prototipo	`void Lcd Custom Chr Cp(char character);`
Descripción	Imprime caracter sobre la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las variables y literales, pueden ser pasadas como carácter
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config`.
Ejemplo	`Lcd_Custom_Chr_Cp('e'); // Print 'e' at . // cursor position . // current.`

LCD_CUSTOM_CMD

Prototipo	`void Lcd Custom Cmd(char command);`
Descripción	Envia comandos al LCD. La lista de comandos validos se muestra en la tabla 6.1.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_Custom_config`.
Ejemplo	`Lcd_Custom_Cmd(Lcd_Clear); //limpia el display`

6.2.4 LIBRERÍA LCD8 (INTERFACE DE 8 BITS).

El compilador mikroC provee una librería de funciones para comunicar al microcontrolador PIC con pantallas LCD que comúnmente uitlizan 8 bits como interface (controlador Hitachi HD44780).

Nota: los puertos que se asignaran como puertos de control y de datos para el LCD deben de configurarse como salidas, antes de utilizar alguna de las siguientes funciones.

```
Lcd8_Config
Lcd8_Out
Lcd8_Out_Cp
Lcd8_Chr
Lcd8_Chr_Cp
Lcd8_Cmd
```

LCD 8_CONFIG

Prototipo	`void Lcd8_Config(char *ctrlport, char*dataport, char RS, char EN, char WR, char D7, char D6, char D5, char D4, char D3, char D2, char D1, char D0);`

Descripción	Inicializa el LCD con puerto de datos y puerto de control con las terminales configuradas por el usuario. Los parámetros RS,EN, WR necesitan estar en el rango 0-7, los parámetros D0…D7 necesitan tener una combinación de valores de 0-7 (ejemplo: 3,6,5,0,7,2,1,4).
Ejemplo	`Lcd8_Config(&PORTC,&PORTD,0,1,2,6,5,4,3,7,1,2,0);`

LCD8_INIT

Prototipo	`void Lcd8 Init(char *ctrlport, char *dataport);`
Descripción	Iinicializa el LCD en el Puerto especificado y en las terminals configuradas por default (ver diagrama de conexiones en la figura)D7 -> PORT.7, D6 -> PORT.6, D5 -> PORT.5, D4 -> PORT.4, E -> PORT.3, RS -> PORT.2.
Ejemplo	`Lcd8_Init(&PORTB, &PORTC);`

LCD8_OUT

Prototipo	`void Lcd8 Out(char row, char col, char *text);`
Descripción	imprime el texto sobre la pantalla LCD en la fila y columna especificada(parámetros `row` y `col`) . Ambas variables y literales pueden ser pasadas como texto.
Requerimientos	El puerto con el LCD debe ser inicializado. Ver `Lcd8_config o Lcd8_init.`
Ejemplo	`Lcd8_Out(1,3,"Hello!"); //imprime “hello!” en la . //línea 1, carácter 3.`

LCD8_OUT_CP

Prototipo	`void Lcd8 Out Cp(char *text);`
Descripción	Imprime el texto en la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las cadenas de variables y literales, pueden ser pasadas como texto
Requerimientos	los puertos que contengan al LCD debe ser inicializados. Ver `Lcd_config o Lcd_init.`
Ejemplo	`Lcd8_Out_Cp("Here!"); // imprime "Here!" en la . // posicion actual del cursor`

LCD8_CHR

Prototipo	`Void Lcd8 Chr(char row, char col, char character);`

Descripción	Imprime un caracter en la pantalla de LCD en la fila y columna especificada (parámetros `row` y `col`). Las variables y literales, pueden ser pasadas como carácter.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd_config.`
Ejemplo	`Lcd8_Chr(2, 3,'i'); // imprime 'i' en la linea2 //, caracter 3`

LCD8_CHR_CP

Prototipo	`void Lcd8 Chr Cp(char character);`
Descripción	Imprime caracter sobre la pantalla LCD en la posición en que se encuentre ubicado el cursor. Las variables y literales, pueden ser pasadas como carácter
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd_config o Lcd8_Init.`
Ejemplo	`Lcd8_Out_Cp('e');// imprime "e"en la . // posicicion que se . . // encuentre el cursor`

LCD8_CMD

Prototipo	`void Lcd8 Cmd(char command);`
Descripción	Envia comandos al LCD. La lista de comandos validos se muestra en la tabla 6.1.
Requerimientos	El puerto que contenga al LCD debe ser inicializado. Ver `Lcd8_config o .`
Ejemplo	`Lcd8_Cmd(Lcd_Clear); //limpia el display`

6.2.5 LIBRERÍA PWM

La mayoría de microcontroladores PIC cuentan con el módulo CPP. El compilador mikroC brinda una librería con funciones que simplifica el uso del módulo PWM.

Nota: Algunos microcontroladores cuentan con 2 o más modulos CPP. En tal caso solo se tiene que especificar el modulo a utilizar, esto se hace adicionando el numero del modulo a utilizar despues de la palabra Pwm. Por ejemplo, Pwm2_Start().

PWM_INIT

Prototipo	`void Pwm Init(long freq);`
Descripción	Inicia al modulo PWM con rango útil 0. El parámetro freq es la frecuencia deseada para el PWM em Hz. La función `Pwm_init` debe ser llamado antes de usar cualquier otra función de la biblioteca PWM.
Requerimiento	El microcontrolador debe contar con el modulo CPP para poder utilizar esta librería, de lo contrario se debe consultar el manual del compilador para una solución alternativa.
Ejemplo	`Pwm_Init(5000);  // inicia al modulo PWM a 5KHz`

PWM_CHANGE_DUTY

Prototipo	`void Pwm Change Duty(char duty ratio);`
Descripción	Cambia el rango útil del PWM. El parámetro `duty_ratio` toma valores de 0 a 255 , donde 0 es 0%, 127 es 50% y 255 es 100% del rango útil. Otros valores especifico spueden ser calculados como (Percent*255)/100.
Requerimiento	Se necesita tener habilitado el modulo CPP en el puerto C para usar esta librería. Para usar esta función el modulo necesita ser inicializado- ver `Pwm_init`
Ejemplo	`Pwm_Change_Duty(192);  // configura el ciclo .  //  util a 75%`

PWM_START

Prototipo	`void Pwm Start(void);`
Descripción	Inicia el PWM.
Requerimiento	Se necesita tener habilitado el modulo CPP en el puerto C para usar esta librería. Para usar esta función el modulo necesita ser inicializado- ver `Pwm_init`
Ejemplo	`Pwm_Start();  //  iniciar el PWM`

PWM_STOP

Prototipo	`void Pwm Stop(void);`
Descripción	Para el PWM
Requerimiento	Se necesita tener habilitado el modulo CPP en el puerto C para usar esta librería. Para usar esta función el modulo necesita ser inicializado- ver `Pwm_init`
Ejemplo	`Pwm_Stop();  // parar el PWM`

6.2.6 LIBRERÍA USART

Un gran número de microcontroladores PIC tienen integrado el módulo de hardware USART. El compilador mikroC provee de una librería para trabajar en forma confortable y hacer una comunicación en modo asíncrono, esto es se puede comunicar el microcontrolador con otros dispositivos utilizando el protocolo RS232.

Nota: las funciones de la librería USART soportan al módulo en los puertos PORTB, PORTC o PORTG. Para trabajar con el módulo en otros puertos se necesita manipular a dicho puerto por software.

Rutinas de la librería

Usart_Init

Usart_Data_Ready

Usart_Read

Usart_Write

Nota: Algunos microcontroladores cuentan con 2 modulos USART. En tal caso solo se tiene que especificar el modulo a utilizar, simplemente se añade el número 1o 2 al nombre de la función. Por ejemplo, Usart_write2();.

USART_INIT

Prototipo	`void Usart_Init(const long baud_rate);`
Descripcion	Inicializa el modulo del hardward USART con el rango de baudios requeridos. Referirse a la hoja de datos del dispositivo en cuestión para el rango de baudios en que puede trabajar dicho dispositivo. Si se especifica un rango de baudios no soportado, el compilador reportará un error. La función Usart_Init sus necesita ser llamada antes de usar cualquier otra función de la librería USART
Requerimiento	se necesita un micro controlador con hardware USART
Ejemplo	`Usart Init(2400); // establece comunicación a 2400 bps`

USART_DATA_READY

Prototipo	`char Usart_Data_Ready(void);`

Retorna	la function retorna un 1 si el dato está listo o cero si no hay dato
Descripcion	Se usa la function para probar si el dato está listo para transmisión
Requerimiento	Requiere tener instalado el hardware para comunicación vía RS232 . El módulo USART debe ser inicializado y establecer comunicación antes de usar esta función. Ver Usart_Init.

Ejemplo	`int receive;`
	`...`
	`// If data is ready, read it:`
	`if (Usart Data Ready()) receive =Usart_Read;`

USART_READ

prototipo	`char Usart_Read(void);`
retorna	Retorna el byte recibido. Si el valle no se recibe, retorna 0.
Descripcion	La función recibe un byte vía USART. Antes Usar la función `Usart_Data_Ready()` para probar si el dato está listo
Requerimiento	

ejemplo	`int receive;`
	`...`
	`// If data is ready, read it:`
	`if (Usart Data Ready()) receive = Usart_Read;`

USART_WRITE

prototipo	`char Usart Write(char data);`
Descripcion	
Requerimiento	

ejemplo	`int chunk;`
	`...`
	`Usart_Write(chunk); /* send data chunk via .  USART */`

CAPÍTULO 7
PROGRAMACIÓN EN AMBIENTE MIKROC

7.1 ESTRUCTURA DE UN PROGRAMA EN C

Para crear un programa en C es necesario seguir los siguientes pasos:

1. Especificaciones del programa (qué se tiene que hacer)
2. Desarrollar diagrama de flujo
3. Escribir el código fuente
4. Compilar
5. Simular.
6. Grabar el programa en el microcontrolador

Como etapas previas a la escritura del código fuente, es importante tener muy claro que es lo que se pretende hacer, por eso, será conveniente realizar un listado con las órdenes necesarias para que el programa se lleve a cabo, y realizar un organigrama o diagrama de flujo con el orden de las mismas, las condiciones, etc.

De forma generalizada, la estructura de un programa en C tiene el siguiente aspecto:

```
declaraciones globales
prototipos de funciones
main()
 {
variables locales;
bloque de sentencias;
llamadas a las funciones;
}
funcion_1()
 {
variables locales a funcion_1;
bloque de sentencias;
llamada a otra/s funciones;
}
funcion_n()
 { …
 }
```

7.1.1 ESTRUCTURAS BÁSICAS EN C.

Existen tres estructuras básicas de para llevar a cabo un programa en mikroC, estas estructuras básicas constan de tres ciclos: ciclo *while* (figura 7.1), ciclo *for* (figura 7.2) y ciclo *do-while* (figura 7.3).

```
// Definición de variables globales
// Definición de funciones
void main(void)
{
  // Definición de variables locales
  // Configuración de registros (recursos y puertos)
  // ciclo infinito
   while ( 1 )
            {
               // Programa de usuario
            }
}
```

Figura 7.1

ciclo While (1)

```
// Definición de variables globales
// Definición de funciones
void main(void)
{
 // Definición de variables locales
// Configuración de registros (recursos y puertos)
 // ciclo infinito
for ( ; ; )
{
 // Programa de usuario
}
}
```

Figura 7.2

Ciclo For(;;)

```
// Definición de variables globales
// Definición de funciones
void main(void)
{
// Definición de variables locales
// Configuración de registros (recursos y puertos)
 // ciclo infinito
  do
{
   // Programa de usuario
 } while ( 1 ) ;
 }
```

Figura 7.3

Ciclo Do-While

7.2 PRIMER PROGRAMA EN MIKROC

Hasta este punto ya se conocen las características principales y el ambiente de trabajo del compilador mikroC, asi como los tipos de datos que se manejan en general para el este compilador, y también se conocen cuáles son los pasos a seguir para realizar un programa en lenguaje C, por lo tanto se expondrá un primer programa para explicar cómo se realiza un programa en compilador mikroC.

En este programa se explicara cómo se inicia un proyecto hasta antes de comenzar a escribir el código fuente, puesto que esto se efectúa siempre que se desee realizar un nuevo proyecto. Después de terminado el proyecto se procede a compilarlo, si no se genera un solo error este se compilara con éxito y se procede a simularlo o en su caso lo podemos descargar al microcontrolador y ver cómo se comporta físicamente. Esto último es a consideración del programador. Aunque siempre es aconsejable una simulación previa a la aplicación en tiempo real.

ENCENDIDO DE UN LED.

El objetivo de este primer proyecto es el de familiarizarse con el entorno del software que mikroC dispone. Para llevar a cabo lo propuesto se realizara un programa que encienda y apague un led del puerto D con un lapso de un segundo, cabe mencionar que se sugiere modificar el periodo entre encendido y apagado y verificar en el microcontrolador que efectivamente el tiempo de parpadeo cambia.

A continuación se muestra en la figura 7.4 el diagrama del circuito con el PIC16f877A sobre el que se probara el funcionamiento del programa a elaborar. Cabe mencionar que el circuito está elaborado en la plataforma del programa llamado Proteus, también se hará uso de este software para la simulación de cada proyecto.

Los componentes específicos son:

1 resistencia de 330Ω.
2 capacitores de 27 ρF
1 Diodo 1N4148
1 Microcontrolador PIC16F877A
1 fuente de Voltaje de 5 V.
1 resistencia de 100Ω.
1 Cristal de 4 MHz
1 Diodo Led
1 Pulsador
1Protoboard

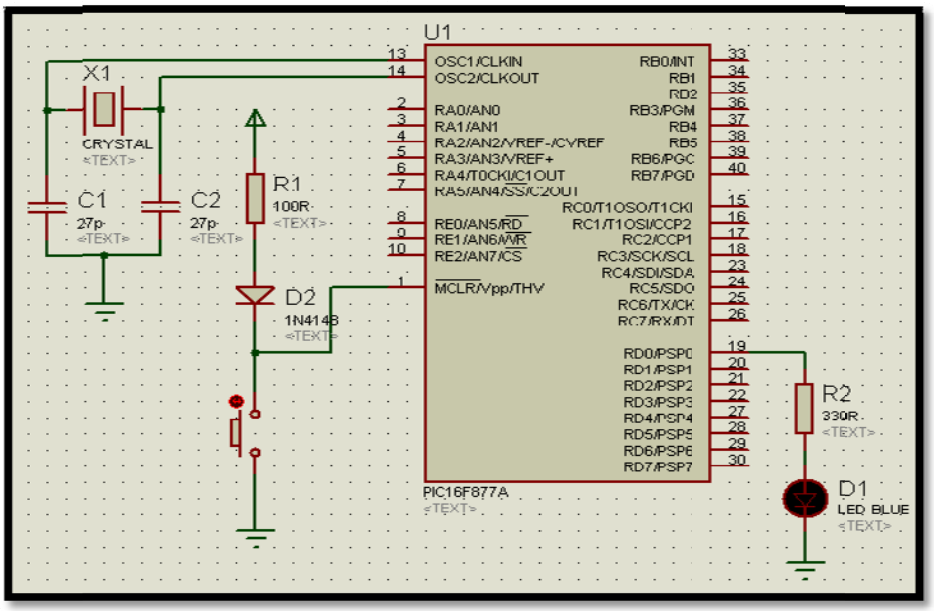


Figura 7.4

Circuito a Implementar con el microcontrolador PIC16F877A

DIAGRAMA DE FLUJO.

Se elabora el diagrama de flujo, el cual no manejara variables, ni datos complejos, ya que el programa trata de la manipulación de un solo bit del puerto D, dicho diagrama se muestra en la figura 7.5.

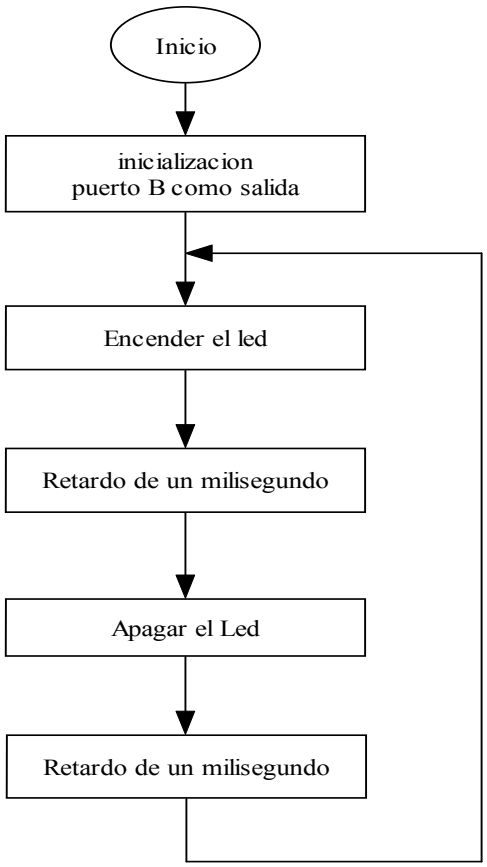


Figura 7.5

Diagrama a Bloques

Se Observa en el diagrama de flujo de la figura 7.5 que se necesita inicializar el programa, esto se logra con la instrucción *void main(void)* o simplemente *void main()*. Se coninúa con la declaración del puerto D como salida. Enseguida se envia el dato cargado con el numero 1, el cual se mantiene por un segundo haciendo uso de la función delay_ms() y se procede a enviar otro dato cargado con el numero 0 y se conserva ese estado por un segundo nuevamente y para finalizar se provoca que el ciclo sea infinito haciendo uso de la sentencia *while(1)*. De esta forma se logra que un led conectado al puerto D en la terminal D0 se encienda y apague en forma intermitente. El programa a escribir en mikroC es el que se muestra en la figura 7.6

Figura 7.6

Programa escrito en lenguaje C.

```
void main(void)
{
   TRISD = 0;     // CONFIGURACION COMO PUERTO DE SALIDA
   while ( 1 )    // CICLO INFINITO
        {
             PORTD = 0b00000001;      // ENVIA PRIMER DATO
             Delay_ms(1000);
             PORTD = 0b00000000;     // ENVIA SEGUNDO DATO
             Delay_ms(1000);
         }
}
```

SOLUCIÓN EN COMPILADOR MIKROC.

El diseño del programa en lenguaje C, es sencillo, la plataforma que mikroC ofrece la hace más versátil, a diferencia de otros compiladores como se explica a continuación.

Pasos a seguir.

- Abrir el compilador mikroC. Aparecerá una pantalla similar a la que se muestra en la figura 7.7. Dar "clic" en la pestaña *Project*, este desplegará un menú que se muestra en la figura 7.8

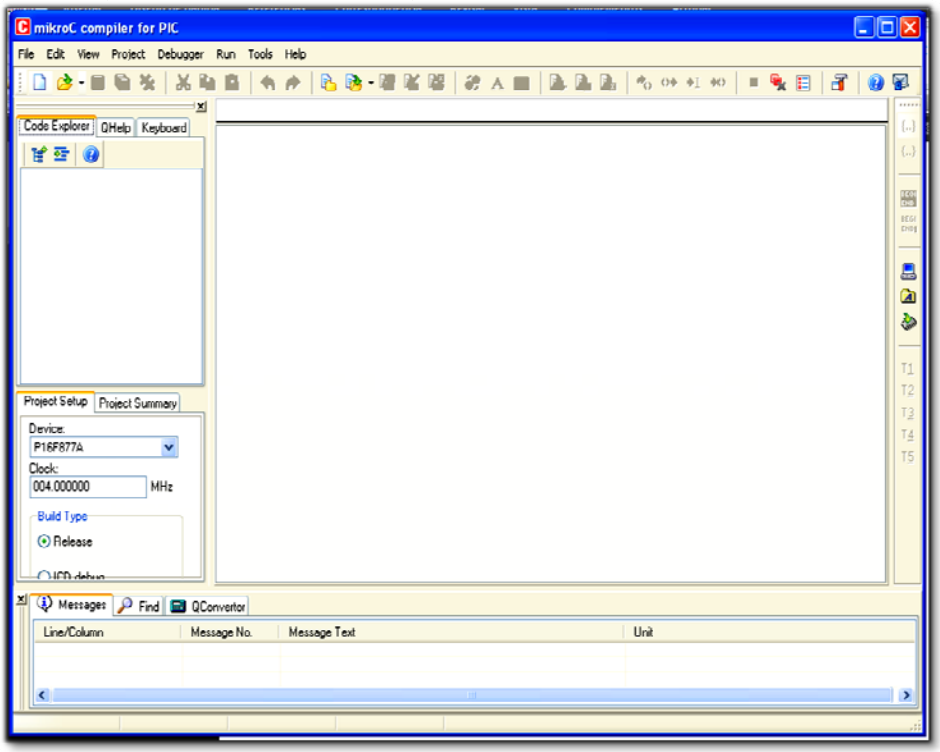


Figura 7.7

Ambiente del compilador mikroC

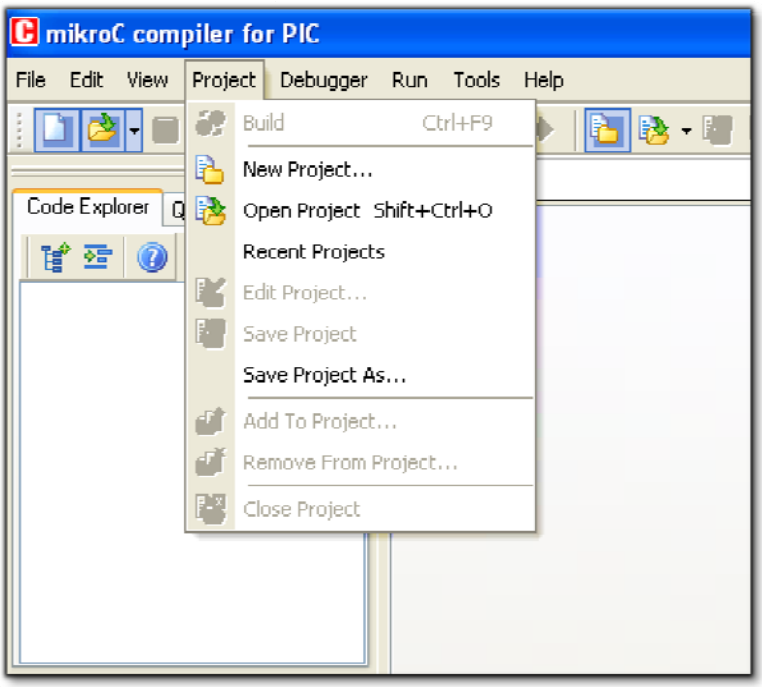


Figura 7.8

Ventana Menú Project

- En el menú de la figura 7.8 escoger la opción *New Project*, a continuación se desplegara una ventana titulada como *New Project* (ver figura 7.9).

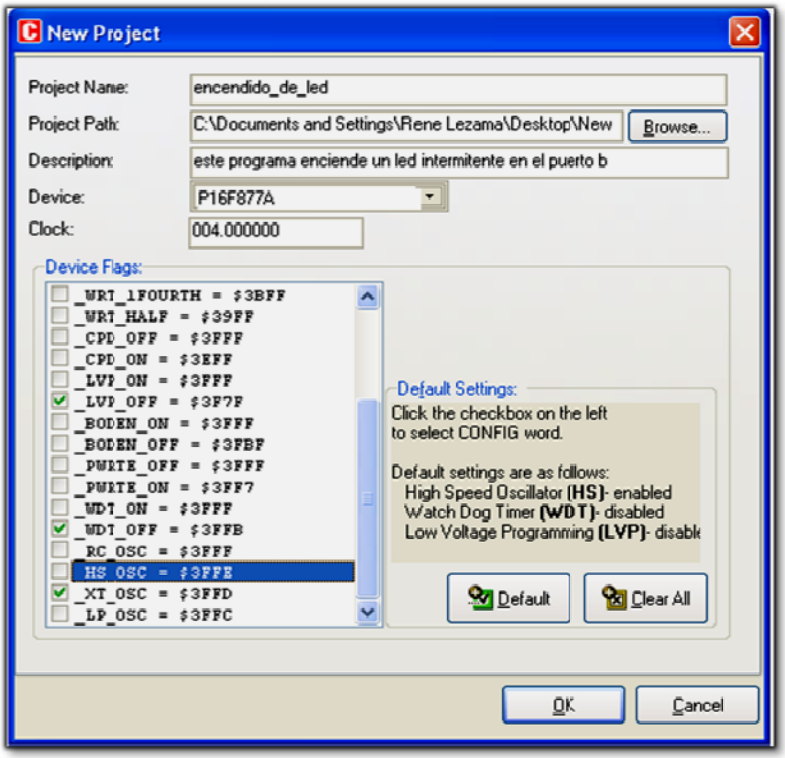


Figura 7.9

Ventana New Project

En la ventana *New Project* aparecen campos que se deben complementar con los datos ahí requeridos y que se explican a continuación:

- **Project name**: llenar este campo con el nombre del proyecto, no se permiten espacios, para distinguir entre palabras adicionar guion bajo
- **Project path**: aquí se debe especificar la ruta donde se guardar el código conjuntamente con los archivos generados por el compilador
- **Description**: este campo es opcional, en el se explica brevemente la función del programa
- **Device:** en el campo device se especifica el microcontrolador PIC a utilizar
- **Clock Device:** en este punto se especifica la frecuencia del cristal a utilizar.
- **Device Flags:** finalmente esta opción es para seleccionar o modificar las distintas modalidades de la palabra de configuración del PIC: proteger o no el código interno, activar o no el temporizador *Watchdog* (WDT), activar o no el temporizador de arranque (power timer), así como seleccionar el tipo de oscilador, etc. La configuración depende del modelo del PIC. Se puede seleccionar Default y verificar que la palabra de configuración tiene seleccionadas las banderas adecuadas, o seleccionarlas manualmente.

En el caso particular el proyecto lleva por nombre "encender_un_led". La dirección donde se encuentra alojado el proyecto es "\Documentos and settings\.....\proyectos mikroC", el modelo del dispositivo es PIC16F877A y la frecuencia de trabajo declarada es de 4Mhz. Para la configuración del proyecto aquí citado dar "clic" en *default settings*, deshabilitar HS_OSC y habilitar XT_OSC.

Hecho lo anterior se escribe el código fuente en el editor de código del compilador mikroC.

En la figura 7.10 aparece el editor de código con el código del programa realizado anteriormente.

encender_un_led.c

```
1  void main(void)
2  {
3     TRISD = 0;                              // CONFIGURACION COMO PUERTO DE SALIDA
4
5     while ( 1 )                             // CICLO INFINITO
6           {
7                    PORTD = 0b00000001;      // ENVIA PRIMER DATO
8                    Delay_ms(1000);
9                    PORTD = 0b00000000;      // ENVIA SEGUNDO DATO
10                   Delay_ms(1000);
11          }
12 }
```

Figura 7.10

Editor de texto del compilador mikroC conteniendo al programa encender_un_led

Una vez escrito el programa fuente se compila presionando la tecla [Ctrl+F9], dando "clic" en la opción Project y seleccionando la opción built o dando "clic" directamente en el icono  . Si la compilación es exitosa se generaran los archivos de salida pertinentes explicados en el capítulo 5. Si se encuentra un error, aparecerá un reporte en la parte inferior del compilador.

Los archivos que se generan se pueden observar en la parte intermedia izquierda dando un "clic" en la pestaña titulada <project summary> y dando "clic" en la raíz de la carpeta <Output Files>. Además se genera un archivo *hex* que es el que utilizaremos para programar al microcontrolador a utilizar dependiendo del proyecto en particular, dicho archivo lo encontraremos en la carpeta que se designo previamente para alojar los proyectos a diseñar. En la figura 7.11 observamos la ventana que aparece al escoger el archivo de salida con extensión *.asm*. El código *asm* generado es el correspondiente al proyecto *encendido_de_led*.

encender_un_led.c | encender_un_led.asm

```
;encender_un_led.c,1 ::                 void main(void)
;encender_un_led.c,3 ::                 TRISD = 0;                      // CONFIGURACION COMO PUERTO DE SAL
$0004   $1303                   BCF     STATUS, RP1
$0005   $1683                   BSF     STATUS, RP0
$0006   $0188                   CLRF    TRISD, 1
;encender_un_led.c,5 ::                 while ( 1 )                     // CICLO INFINITO
$0007   $       L_main_0:
;encender_un_led.c,7 ::                 PORTD = 0b00000001;             // ENVIA PRIMER DATO
$0007   $3001                   MOVLW   1
$0008   $1283                   BCF     STATUS, RP0
$0009   $0088                   MOVWF   PORTD
;encender_un_led.c,8 ::                 Delay_ms(1000);
$000A   $3006                   MOVLW   6
$000B   $00FC                   MOVWF   STACK_12
$000C   $30FF                   MOVLW   255
$000D   $00FB                   MOVWF   STACK_11
$000E   $30FF                   MOVLW   255
$000F   $00FA                   MOVWF   STACK_10
$0010   $0BFC                   DECFSZ  STACK_12, F
$0011   $2813                   GOTO    $+2
$0012   $281A                   GOTO    $+8
$0013   $0BFB                   DECFSZ  STACK_11, F
$0014   $2816                   GOTO    $+2
$0015   $2819                   GOTO    $+4
$0016   $0BFA                   DECFSZ  STACK_10, F
$0017   $2816                   GOTO    $-1
$0018   $2813                   GOTO    $-5
$0019   $2810                   GOTO    $-9
$001A   $301A                   MOVLW   26
```

Figura 7.11

Archivo con extensión asm

A continuación en la figura 7.12 se observa es archivo con extensión *.lst* se menciona solo como mera referencia, puesto que contiene la misma información que el archivo con extensión *asm*. El archivo con extensión *lst* es para el control interno del compilador y el *asm* nos puede servir para hacer una comparación al hacer un programa en ensamblador o simplemente como información.

```
;  ASM code generated by mikroVirtualMachine for PIC - V. 6.0.0.0
;  Date/Time: 25/09/2007 10:01:27 p.m.Cesar
;  Info: http://www.mikroelektronika.co.yu


; ADDRESS       OPCODE  ASM
; ------------------------------------------------
$0000   $2804                   GOTO    _main
$0004   $       _main:
;encender_un_led.c,1 ::                 void main(void)
;encender_un_led.c,3 ::                 TRISD = 0;                      // CONFIGURACION COMO PUERTO DE SAL
$0004   $1303                   BCF     STATUS, RP1
$0005   $1683                   BSF     STATUS, RP0
$0006   $0188                   CLRF    TRISD, 1
;encender_un_led.c,5 ::                 while ( 1 )                     // CICLO INFINITO
$0007   $       L_main_0:
;encender_un_led.c,7 ::                 PORTD = 0b00000001;             // ENVIA PRIMER DATO
$0007   $3001                   MOVLW   1
$0008   $1283                   BCF     STATUS, RP0
$0009   $0088                   MOVWF   PORTD
;encender_un_led.c,8 ::                 Delay_ms(1000);
$000A   $3006                   MOVLW   6
$000B   $00FC                   MOVWF   STACK_12
$000C   $30FF                   MOVLW   255
$000D   $00FB                   MOVWF   STACK_11
$000E   $30FF                   MOVLW   255
$000F   $00FA                   MOVWF   STACK_10
$0010   $0BFC                   DECFSZ  STACK_12, F
$0011   $2813                   GOTO    $+2
```

Figura 7.12

Archivo con extensión lst.

Hasta este punto se concluye la elaboración de el primer programa en lenguaje C, el siguiente paso es pasar a la simulación, para después cargar al microcontrolador con el programa previamente simulado.

El archivo a cargar se aloja en la carpeta que previamente se ha designado para los proyectos a realizar en mikroC, y este lleva la extensión "*.hex*". en la figura 7.13 se presenta el código de maquina (extensión *.hex*) que se cargara en el microcontrolador tanto del simulador como el que se dispone como hardware.

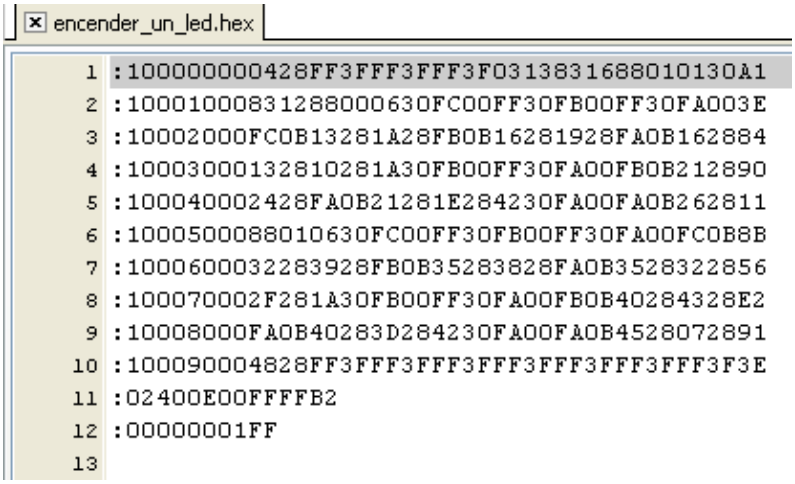


Figura 7.13

Código con extensión .hex generado por la compilación del proyecto encender_un_led

A continuación se explicara brevemente como cargar el programa en el simulador del Programa *Proteus*.

Primeramente se ejecuta el software *Proteus* en la sección *Isis*, una vez abierto se procede a realizar el diagrama de la figura 7.4. Ya que se cuenta con el diagrama plasmado en el ambiente del programa damos doble "clic" sobre el microcontrolador y nos mostrara una ventana como la que se muestra en la figura 7.14 titulada *Edit Component*.

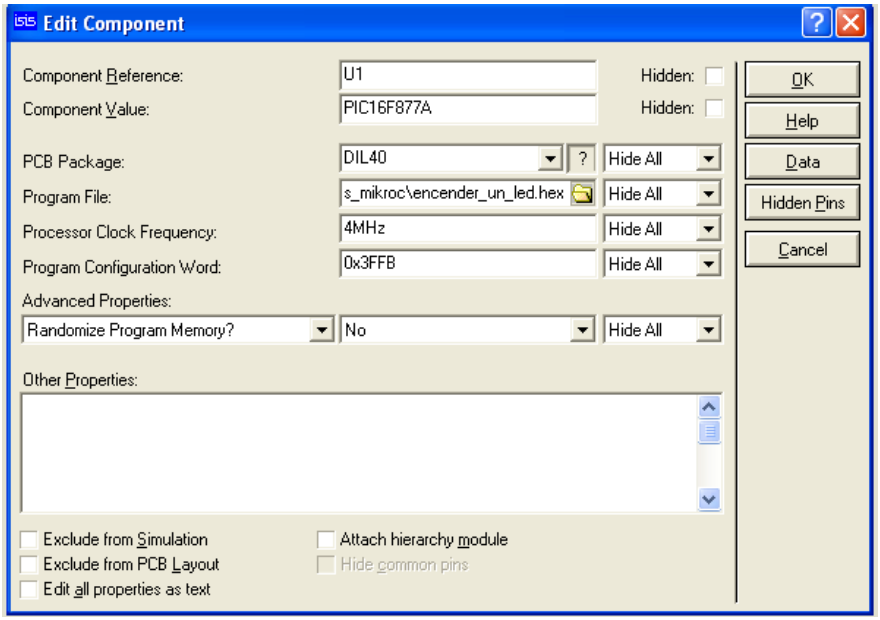


Figura 7.14

Ventana Edit Component

En la ventana *Edit Component* se editan los siguientes campos

- Program File: en este campo se da la ruta donde se encuentra el proyecto realizado en mikroC y se da doble "clic" en el archivo que contiene la extensión *.hex* y que contiene el nombre del archivo a simular.
- Processor Clock Frequency: Aquí es preciso especificar la frecuencia del reloj con la cual estará trabajando el microcontrolador.

Para finalizar dar "clic" en ok y continuar con la simulación dando "clic" en el icono play  que se encuentra en la parte inferior izquierda de la ventana del simulador.

Se verifica que la simulación arroje los resultados esperados y se procede con la programación del microcontrolador.

PROGRAMACIÓN DEL MICROCONTROLADOR

Una vez verificados los resultados de la simulación y realizadas las modificaciones pertinentes al programa en edición se procede a la programación del microcontrolador. En esta última parte es preciso mencionar que se puede utilizar cualquier programador de PIC's que sea útil a nuestras necesidades.

En este proyecto y todos los demás se utiliza un programador PIC-PROG USB protocolo USB 2.0 (ver figura 7.16), pero esto no quiere decir que no se puedan utilizar programadores que utilicen el protocolo RS 232. Esto es a consideración del programador.

En la figura 7.15 se muestra la ventana del entorno del programador PIC-PROG USB

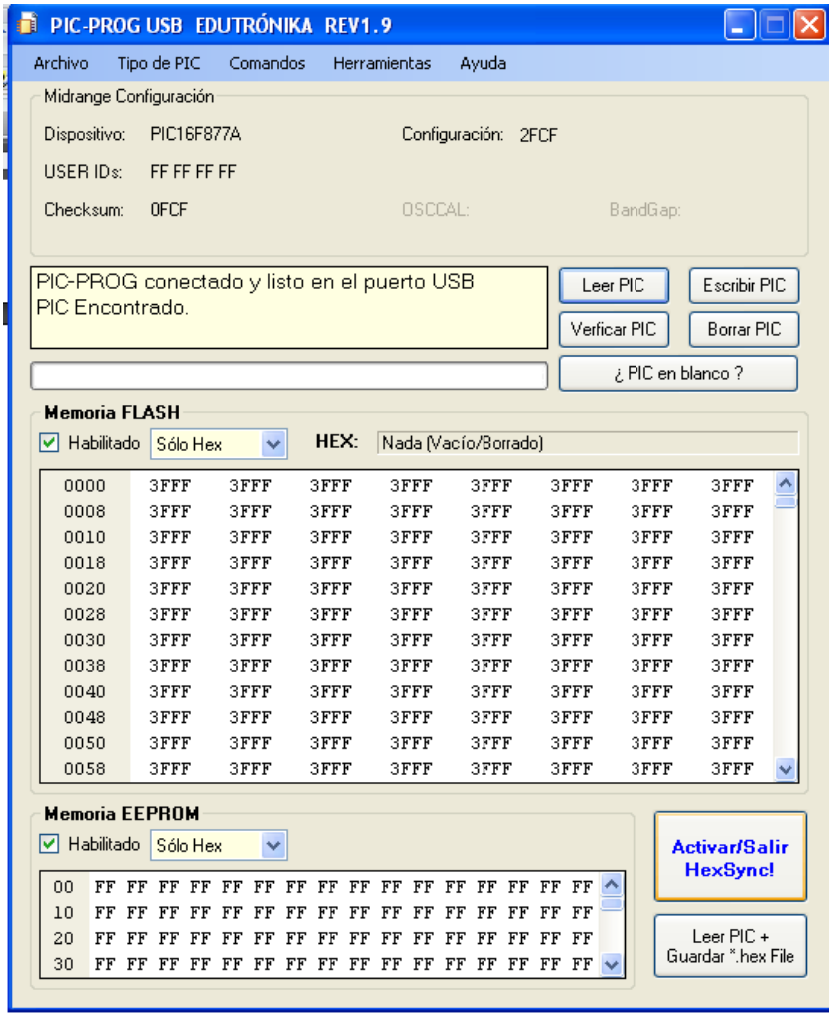


Figura 7.15

Entorno del programador PIC-PROG USB

En el entorno del programador PIC-PROG USB solo se necesita abrir el archivo *hex* mismo que se utilizo en la simulación, una vez abierto solo damos “clic” en el icono que tiene la etiqueta Escribir PIC .

En la figura 7.15 también se observa que el entorno del programador cuenta con las opciones de borrado, lectura y verificación del estado del PIC.

FOTOGRAFIA DEL PROGRAMADOR

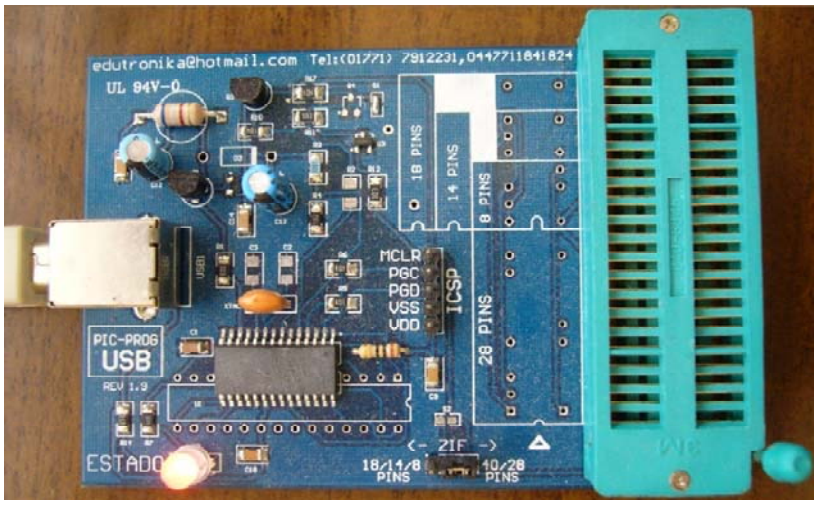


Figura 7.16

Imagen del programador de PIC's utilizando el protocolo USB.

Una vez programado el PIC se procede a montarlo en el protoboard y verificar que efectivamente el software funciona según lo previsto.

FOTOGRAFIA DEL MONTAJE.

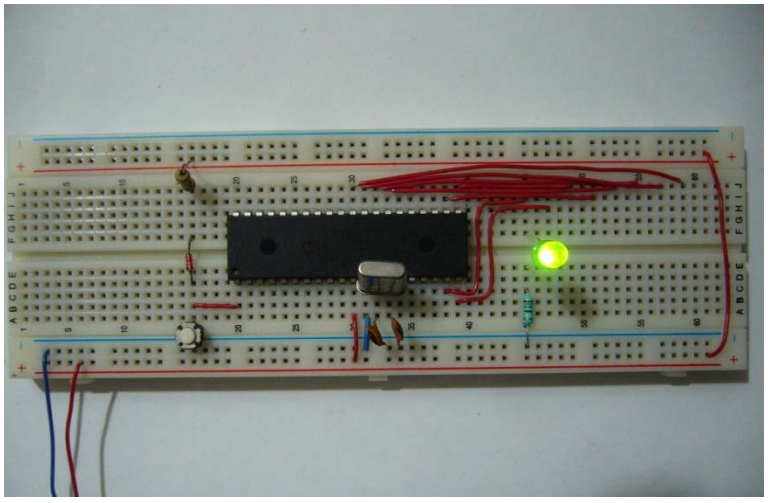


Figura 7.17

Fotografía del montaje sobre protoboard para manipular un led

Para concluir, en este primer proyecto se observa que la programación en lenguaje C ahorra tiempo al momento de desarrollar

un programa, sobre todo se observara su versatilidad al realizar proyectos donde se implica el tratamiento de operaciones matemáticas avanzadas y también si se quiere incurrir en el ámbito de adquisición de datos o el tratamiento de señales, conjuntamente con la implementación de filtros digitales que se pueden realizar con la utilización de otros modelos de PIC's o inclusive con el modelo PIC16F877A que este documento utiliza en la realización de todos los proyectos.

7.3 PROYECTOS

ENCENDIDO DE UN LED 2.

Este programa básicamente realiza lo mismo que el programa ejecutado anteriormente con la diferencia que se manipularan solo bits específicos de un puerto que se desee controlar por ejemplo manejar la terminal RB0 o tal vez la terminal RD1 etc.

DIAGRAMA ELÉCTRICO.

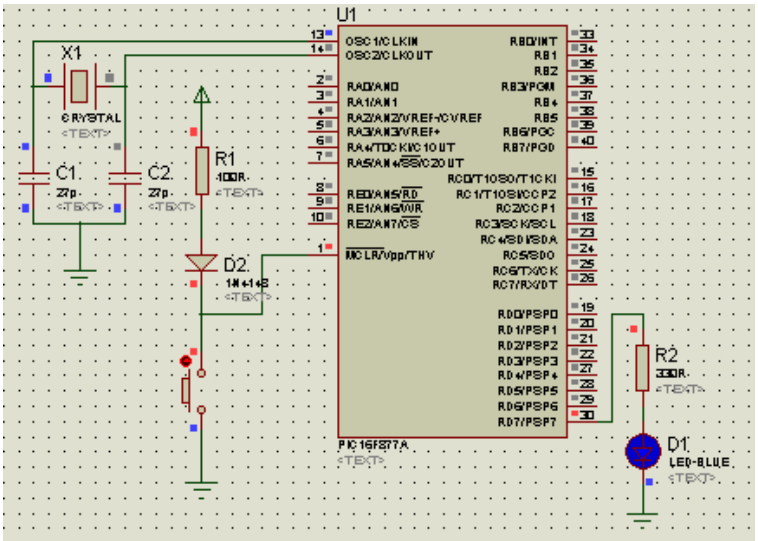


Figura 7.18

Diagrama eléctrico para el programa encender_un_led2

En el diagrama eléctrico de la figura 7.18 solo existe una modificación pues el led estará conectado a la terminal RD7. Este diagrama es el utilizado en la simulación y en la implementación en tiempo real.

DIAGRAMA DE FLUJO.

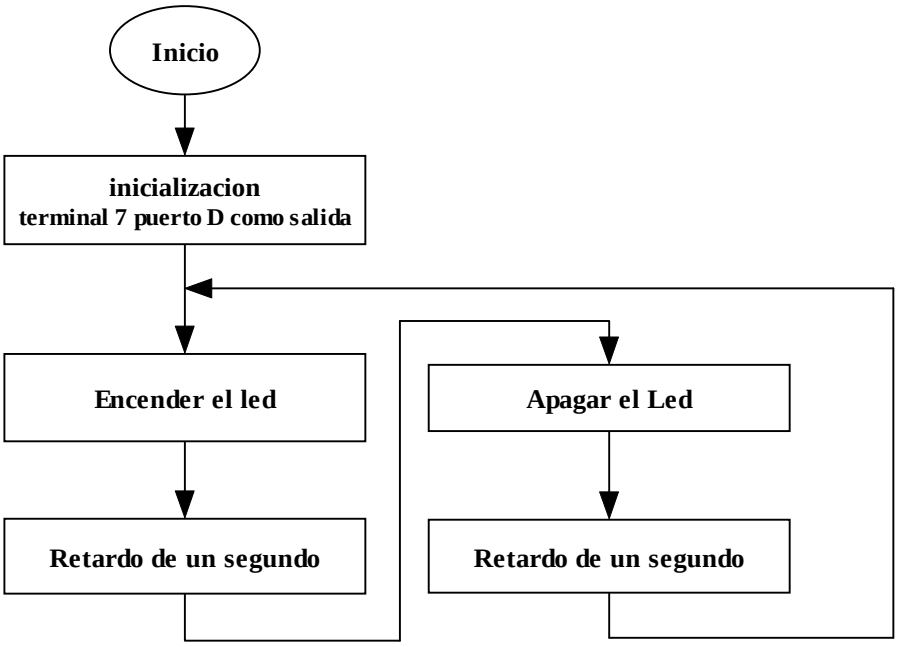


Figura 7.19

Diagrama de flujo para el programa encender un led2

En el diagrama a flujo de la figura 7.19 solo se nota un cambio que es declarar los bits a usar en forma individual. A continuación se procede con la solución en lenguaje C.

SOLUCIÓN EN LENGUAJE C.

El programa a realizar de acuerdo al diagrama de flujo es el código que se muestra en la figura 7.20

```
    void main (          //inicializa el programa
void )
    {
TRISD.F7 = 0;            //se declara el bit F7 del puerto D como salida

while( 1 )               //se inicializa un ciclo infinito
    {
PORTD.F7 = 1;            //se coloca en alto el bit F7 del puerto D
Delay_ms( 1000 );        //inicializa un retardo por software
                         //de 1000ms para mantener en alto al bit F7
    PORTD.F7 = 0;        //se coloca en 0 el bit F7 portd
Delay_ms( 1000 );        //se provoca un retardo de 1000 ms para
                       mantener
                         //en bajo al bit F7 del puerto D
    }
    }
```

Figura 7.20

Código fuente para el programa: encender_un_led2

En el programa que se observa en la figura 7.20 se declara a la terminal RD7 (bit PORTD.F7) como salida en vez de declarar al puerto completo como salida, esta es una de las características con que cuenta mikroC la instrucción para este caso es *TRISD.F7* = 0 y puede funcionar para cualquier puerto como se explica al principio de este capítulo e inclusive para configurar bits de registros especiales.

ENCENDIDO DE UN LED 3.

En este ejercicio se plantea una opción más para manipular datos por alguno de los puertos que se deseen manejar, el diagrama de flujo a seguir es el mismo que el que se muestra en la figura 7.19 así como el diagrama eléctrico de la figura 7.18.

Entonces se ejecuta el código de la figura 7. 21 donde se utiliza el operador "~", El cual complementa el dato anterior en cada ciclo en que se ejecuta el programa.

```
void main()
{
 TRISD.F7 = 0;          // declara el bit F7 como salida
 PORTD.F7 = 0;          //colocar el estado inicial como cero
while(1)                // se inicializa un ciclo infinito
{
 PORTD.F7= ~PORTD.F7;   // se complementa el dato actual en el puerto D
                        //bit 7
 Delay_ms(1000);        // se mantiene el dato por un segundo
 }                      //fin del ciclo y empieza nuevamente
}
```

Figura 7.21

Programa para encender un led utilizando el operador "~"

La instrucción delay_ms() puede tomar distintos valores de acuerdo a las necesidades del programador, esto es válido en los dos programas iniciales, donde el tiempo de encendido y el tiempo de apagado no llevan una relación directa, a diferencia del tercer programa donde se hace una sola recurrencia a la función delay por cada ciclo de ejecución del programa.

FOTO DEL MONTAJE

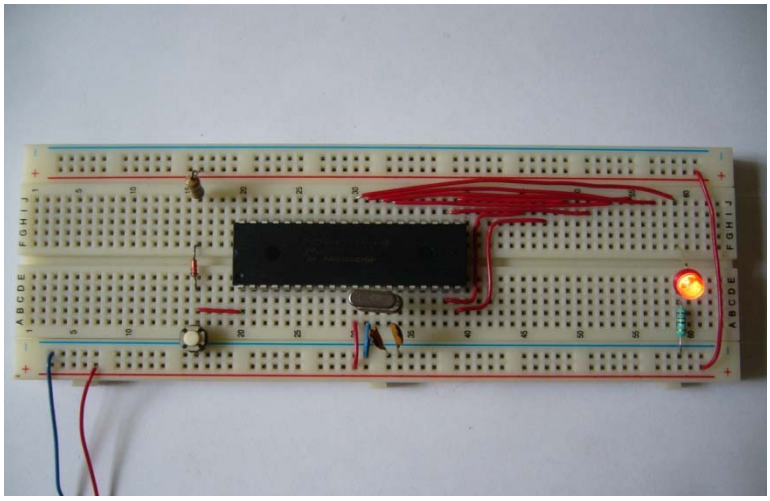


Figura 7.22

Montaje del circuito sobre protoboard para probar los programas encendido de un led 1 y 2

LUCES SECUENCIALES

El programa a desarrollar tiene básicamente como objetivo controlar un de arreglo de leds conectados al puerto D (figura 7.23) en forma secuencial. En la figura 23 se muestra el esquema eléctrico del circuito a simular o montar sobre el cual se deben hacer las pruebas y verificar el funcionamiento del programa.

Materiales específicos para el proyecto luces secuenciales

8 resistencia de 330Ω.	1 resistencia de 100Ω.
2 capacitores de 27 ρF	1 Cristal de 4 MHz
1 Diodo 1N4148	8 Diodos Led
1 Microcontrolador PIC16F877A	1 Pulsador
1 fuente de Voltaje de 5 V.	1Protoboard

DIAGRAMA ELÉCTRICO

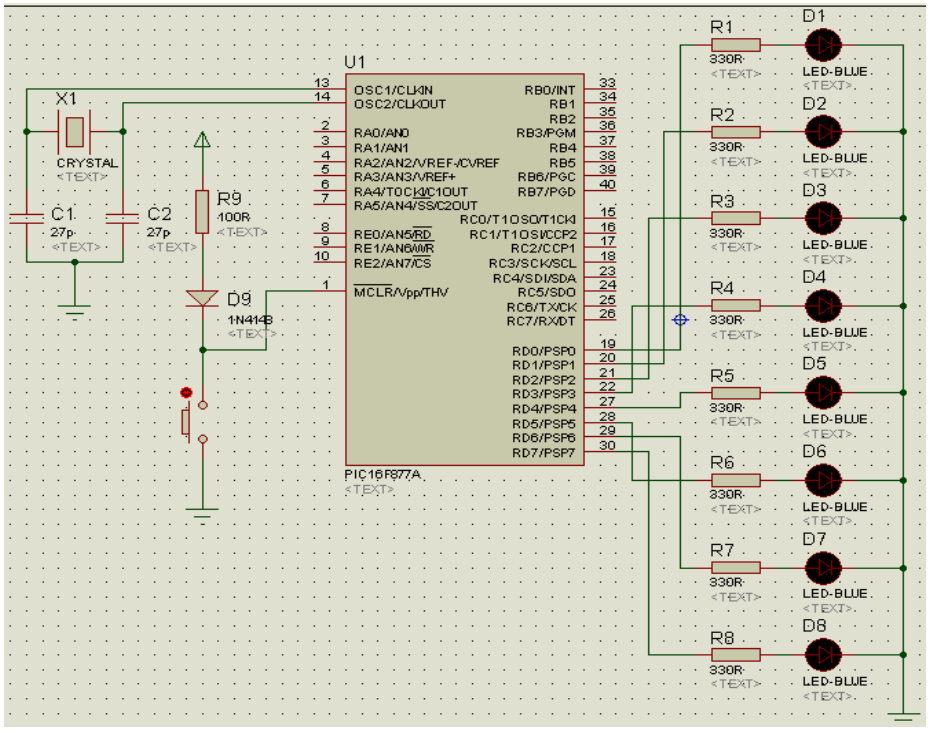


Figura 7.23

Esquema eléctrico para ejecutar el programa luces_secuenciales

Se debe crear un programa que envíe la siguiente secuencia de datos al puerto de salida D.

Secuencia :

- 00000001
- 00000010
- 00000100
- 00001000
- 00010000
- 00100000
- 01000000
- 10000000

ALGORITMO

1. Configuración de puerto como salida de datos.
2. Envío de primer dato al puerto de salida
3. Envío de segundo dato al puerto de salida
4. Envío de tercer dato al puerto de salida.

.

9. Envío de último dato al puerto de salida

10. Regresa.

DIAGRAMA DE FLUJO.

En base al algoritmo se diseña el diagrama de flujo a seguir mostrado en la figura 7.24 para así diseñar el programa propuesto.

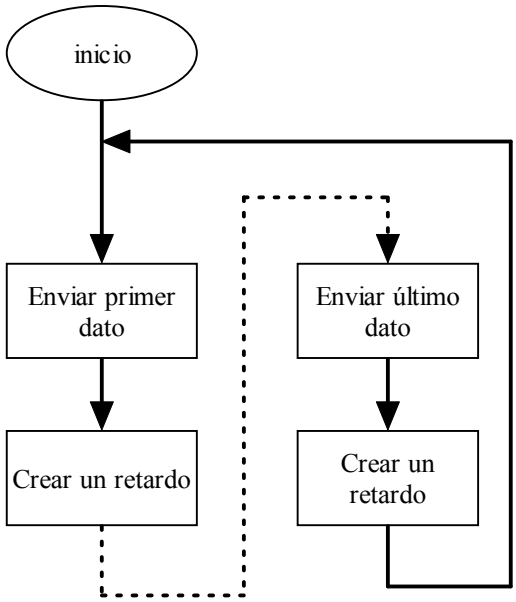


Figura 7.24

Diagrama de flujo para el programa luces secuenciales

Enseguida en la figura 7.25 se muestra el código a ejecutar en mikroC y cargarlo al microcontrolador para su puesta en marcha.

```
void main(void)
{
TRISD = 0                       //configuración como puerto de salida
while ( 1 )                     // ciclo infinito
      {
      PORTD = 0b00000001;       // envía primer dato
      Delay_ms(500);
      PORTD = 0b00000010;       // envía segundo dato
      Delay_ms(500);
      PORTD = 0b00000100;       // envía tercer dato
      Delay_ms(500);
      PORTD = 0b00001000;
      Delay_ms(500);
      PORTD = 0b00010000;
      Delay_ms(500);
      PORTD = 0b00100000;
      Delay_ms(500);
      PORTD = 0b01000000;
      Delay_ms(500);
      PORTD = 0b10000000;
      Delay_ms(500);
      }
}
```

Figura7. 25

Código a ejecutar en mikroC.

Una vez compilado el programa en el compilador mikroC, se procede a su simulación en el Simulador de Proteus, y verificado su funcionamiento se realiza el montaje en el protoboard como el mostrado en la figura 7.26.

FOTOGRAFIA DEL MONTAJE

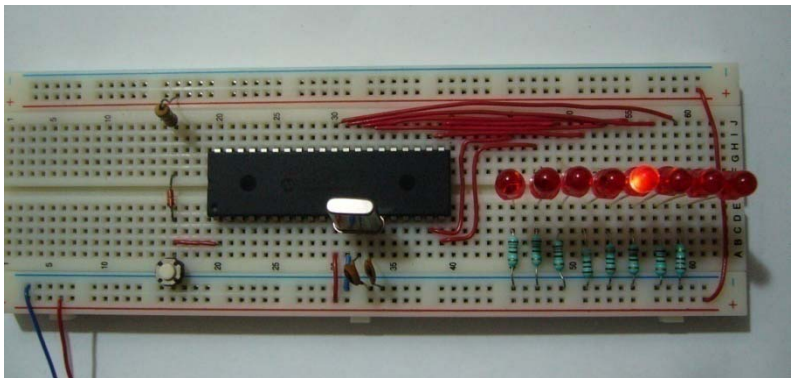


Figura 7.26

Fotografía del circuito en protoboard para verificar el funcionamiento del programa encendido de un LED .

LUCES SECUENCIALES 2

En esta versión del programa de luces secuenciales se hace uso de cadenas y apuntadores, los cuales son un recurso del lenguaje C. en este proyecto se utilizan los mismos materiales del proyecto "Luces secuenciales" así como la misma configuración del circuito a implementar y simular. El programa exhibirá una serie de datos almacenados en un arreglo predefinido y una variable utilizada como apuntador exhibirá el dato pertinente a donde este apuntando

A continuación se expone un breve resumen sobre el concepto de arreglos, condicionante *if* y símbolos de condición.

Definición de arreglos.

```
#define     MAX   50
int         vector_one[10];          /* arreglo de 10 enteros */
float       vector_two[MAX];         /* arreglo 50 flotantes */
float       vector_three[MAX - 20];  /* arreglo 30 flotantes */
char        numero[5];
Short       dato [8];
Long        temperature [15];
Unsigned    peso[7];
Unsigned    short d[3];
```

Inicializando arreglos.

Arreglo el cual contiene el número de días de cada mes:

```
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

La siguiente declaración es idéntica a la anterior:

```
int *days = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Las dos declaraciones siguientes son equivalentes:

```
const char msg1[ ] = {'T', 'e', 's', 't', '\0'};
const char msg2[ ] = "Test";
```

Condicionante if.

if (*expresión*) conjunto *1* [else conjunto *2*]

Cuando la *expresión* evaluada es verdadera, Las instrucciones del conjunto *1* son ejecutadas. Si la *expresión* es falsa, las instrucciones del conjunto *2* son ejecutadas. La *expresión* debe ser evaluada a un valor entero. Los paréntesis que encierran la *expresión* son obligatorios.

La palabra especial "else conjunto 2" es opcional.

Símbolos de condición

Operador	Operación
==	igual
!=	no igual
>	mayor que
<	menor que
>=	mayor que o igual a
<=	menor que o igual a

ALGORITMO

Se expone el siguiente algoritmo para desarrollar el proyecto

1. Configuración de puerto como salida.
2. Inicializa apuntador.
3. Envío de dato apuntado.
4. Incrementa apuntador.
5. Si apuntador es mayor que o igual a 8 inicia el apuntador.
6. Regresa a 3.

DIAGRAMA DE FLUJO

A partir del algoritmo anterior se desarrolla el diagrama de flujo (figura 27) que se utilizará para generar el código del programa "luces secuenciales2".

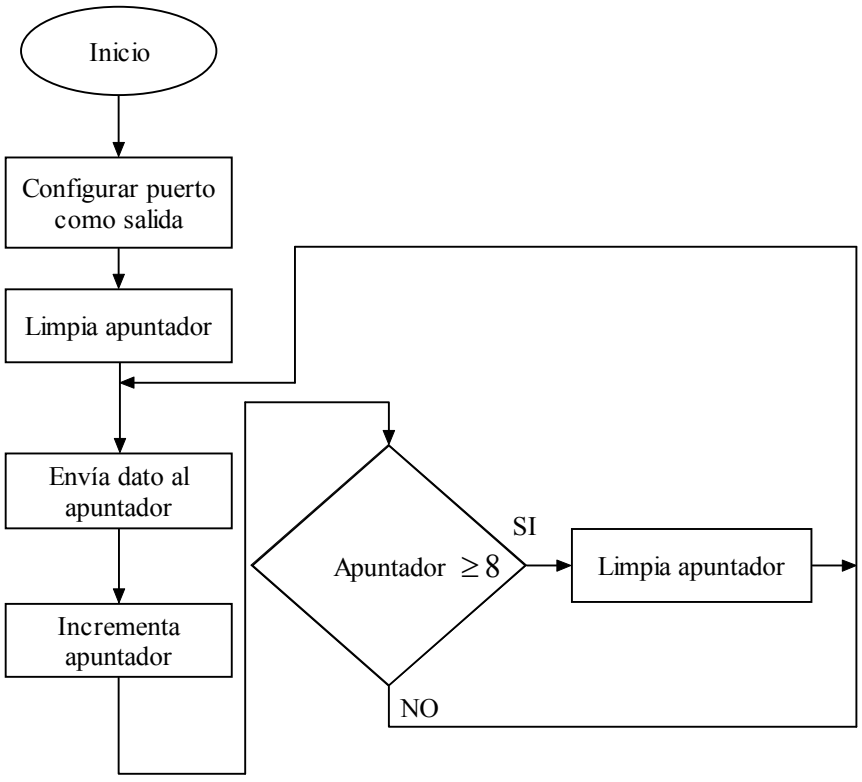


Figura 7.27

Diagrama de flujo para el proyecto luces secuenciales2

En base al diagrama de flujo de la figura 7.27 se dispone a realizar el código del programa para controlar una serie de leds conectadas al puerto D del PIC (ver figura 7.23). En la figura 7.28 se expone el código a ejecutar en el compilador mikroC.

CÓDIGO DEL PROGRAMA

Figura 7.28

Código del programa luces secuenciales 2 utilizando arreglos y apuntadores

```
Short dato [8]= {1,2,4,8,16,32,64,128};
short apunta;
void main(void)
{
      TRISD = 0;                          //Configura puerto
       apunta = 0;                        //Limpia apuntador
       while(1)                           // Ciclo infinito
              {
              PORTD = dato [ apunta ];    // Envía dato
              Delay_ms(1000);
              apunta ++;                  //Incrementa      el
                                          //apuntador
              if ( apunta > =  8 )        //Si apuntador ≥ 8
              apunta = 0;                 //Limpia apuntador
               }
}
```

EXPLICACIÓN DEL PROGRAMA

Como se menciono al principio el programa hace uso de arreglos y apuntadores. Analizando el código de la figura 7.28 obsérvese que se inicia con un arreglo de tipo *short* llamado "dato" y que contiene 8 datos, estos datos están declarados en formato decimal, pero en su equivalente binario permiten visualizar cada bit del puerto D. Nótese también que existe otro elemento declarado como "apunta", el cual realiza la función de apuntador, es decir apuntara a la localidad de memoria del arreglo "dato". Una vez que se tienen declarados los arreglos y los apuntadores se procede a iniciar el programa en forma normal, es decir declaramos el puerto D como salida y al apuntador se inicia con valor 0, realizado lo anterior se inicia un ciclo infinito. En primera instancia "apunta" hace referencia a la posición 0 del arreglo "dato", por lo tanto el dato mostrado en D es el contenido en dicha posición, se continua con la secuencia del programa y se realiza un retardo de 1 segundo, a continuación se incrementa "apunta" y se prosigue con un condicionante; si el valor de apunta es menor que 8, se brinca la siguiente instrucción y continua con el ciclo infinito, es decir, ahora "apunta" tiene el valor 1 y apunta a la posición 1 del arreglo "dato", por lo tanto, el puerto D mostrara el valor contenido en "dato" al cual hace referencia "apunta" . Lo anterior se realizara sucesivamente hasta que "apunta" llegue al valor 8, entonces la instrucción siguiente es

tomada en cuenta y se limpia el apuntador y se sigue con la secuencia del programa.

Para verificar lo anterior se recomienda ejecutar el software en el simulador del Software Proteus y verificar su funcionamiento antes de su montaje en forma física.

La fotografía del circuito montado es la misma a la de la figura 7.26.

LUCES SECUENCIALES 3

EJEMPLO A NIVEL DE BITS.

El siguiente programa realiza las mismas funciones de encender una serie de leds en forma secuencial conectados al puerto D, solo que el objetivo es utilizar operaciones a nivel de bits para realizar el desplazamiento sobre el puerto D.

El material a utilizar es el mismo utilizado en la el proyecto "Luces secuenciales" y el esquema eléctrico para su simulación y montaje es el mostrado en la figura 7.23.

ALGORITMO

1. Configuración de puerto como salida.
2. Inicializa variable.
3. Envía valor de la variable al puerto.
4. Modifica la variable.
5. Si variable es cero, Inicializa la variable.
6. Regresa a 3.

DIAGRAMA DE FLUJO

De nueva forma basándose en el algoritmo anterior se lleva a cabo la elaboración del diagrama de flujo, para que posteriormente se elabore el código del programa

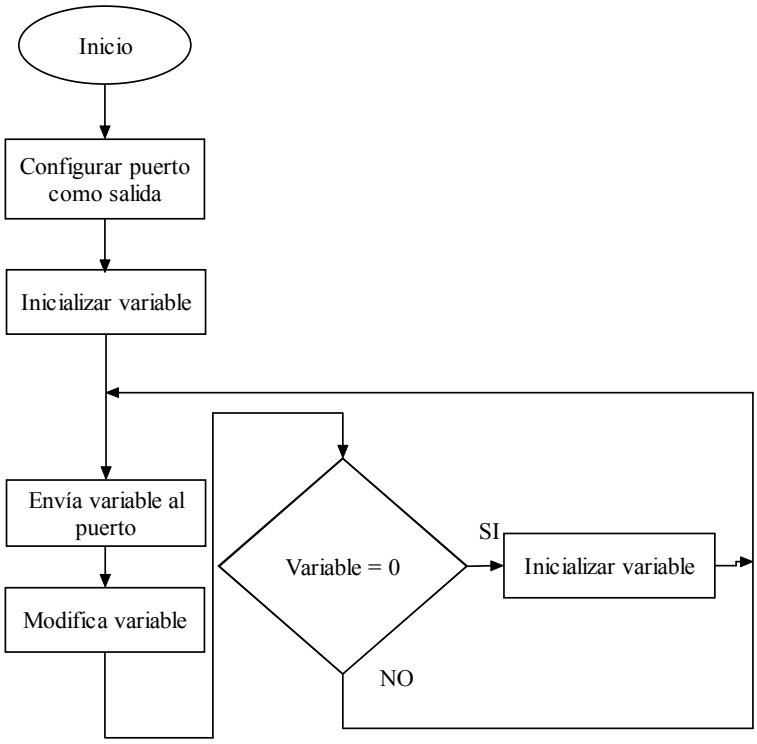


Figura7. 29

Diagrama de flujo para el programa luces secuenciales3

PROGRAMA

En base al diagrama de flujo de la figura 7.29 se desarrolla el código mostrado en la figura 7.30.

```
void main ( void )
{
 unsigned short dato;                  //declarar variable
TRISD = 0;                             //declarar Puerto como salida
dato = 0x01;                           //inicializar variable con 0x01
while ( 1 )                            //declara ciclo infinito
          {
                PORTD = dato;          //enviar dato a PORTD
                Delay_ms ( 300 );      //realizar un retardo de 300 ms
                dato = dato << 1;      //recorrer dato a la izquierda
                if ( dato == 0 )       //si el dato es igual a cero colocar en
                dato = 0x01;           //dato el valor 0x01
           }
}
```

Figura 7.30

Código a ejecutar para el programa luces secuenciales3

FUNCIONAMIENTO DEL PROGRAMA.

Básicamente el programa hace uso del recurso de operaciones a nivel de bits. Primero el programa inicia con la declaración de una variable llamada "dato" que cambia constantemente de valor , pero que se inicializa con valor 0x01 en hexadecimal o lo que es lo mismo 0b00000001 en binario. Entonces de acuerdo al código de la figura 7.30 el primer dato observado en el puerto D es 0b0000001, se crea un retardo de 300ms, para que enseguida la variable cambie de valor es decir el dato es 0b0000001 y con el operador "<<" cambia a 0b0000010, y nuevamente se repite el ciclo hasta que dato sea igual con 0x00, y es cuando se asigna nuevamente el valor 0x01.

Se procede a la compilación del programa en el compilador mikroC, se simula para ver los resultados. Finalmente se prueba sobre el mismo circuito montado para el programa "luces secuenciales" .

LUCES SECUENCIALES4

(OPERACIONES MATEMÁTICAS)

Este programa tiene por objetivo generar un código que controle una serie de leds de la misma forma a los últimos tres programas editados en este capítulo, la diferencia es la utilización de operaciones aritméticas.

Para poder realizar un corrimiento a la izquierda equivalente a el que se logra con el operador << se utiliza el producto 2^n y el corrimiento a la derecha (operador >>) se logra haciendo la división entre 2^n. El corrimiento a la izquierda se observa en las siguientes operaciones:

decimal	binario
2*1 =1+1=2	0b00000001+0b00000001=0b00000010
2*2=2+2=4	0b00000010+0b00000010=0b00000100
2*4=4+4=8	0b00000100+0b00000100=0b00001000
2*8=8+8=16	0b00001000+0b00001000=0b00010000
2*16=16+16=32	0b00010000+0b00010000=0b00100000
2*32=32+32=64	0b00100000+0b00100000=0b01000000

2*64=64+64=128 0b01000000+0b01000000=0b10000000

Para la realización del programa se hace uso del algoritmo del proyecto “luces secuenciales3” así como el diagrama de flujo para el mismo.

La única variante dados los argumentos anteriores será el operador matemático “*” y el multiplicando 2.

PROGRAMA

```
void main ( void )
   {
   unsigned short dato;
   TRISD = 0;
   dato = 1;
   while ( 1 )
         {
           PORTD = dato;
           Delay_ms (250);
           dato = dato * 2;
           if ( dato == 0 )
           dato = 0x01;
          }
   }
```

Figura7.31

Código del programa luces secuenciales 4 utilizando operaciones matemáticas.

Una vez que se ha escrito el código mostrado en la figura 7.31 se lleva a la compilación, hecho esto y si no se generan errores proseguir con la simulación, verificando que el resultado es similar a los resultados obtenidos en los programas que se refieren a luces secuenciales. Para finalizar se prueba el código sobre el circuito como el que se muestra en la figura 7.26.

SECUENCIAS CONDICIONADAS.

Hasta este punto se han elaborado programas utilizando los recursos básicos que ofrece el lenguaje C pero que son vastos para involucrarse con programas no complejos, pero sí involucran otras instrucciones, con lo cual se hace más amigable la programación en C, aunando a esto la plataforma que posee mikroC permite que la programación sea todavía más versátil.

El siguiente programa controla el puerto D enviando una secuencia de datos que empezarán a recorrerse ya sea derecha a izquierda o viceversa, la dirección depende de un interruptor que está conectado a una terminal del puerto A. Además se hace uso del control a nivel de bits para el caso del puerto A por ejemplo:

```
ADCON1 = 6;
TRISA = 0x7F;
{…;
  …;
if ( PORTA.F0 == 0 )
   PORTD.F0 =1;
```

Para este proyecto se necesitarán algunos elementos más para el montaje del circuito en el protoboard y que se deben conectar de acuerdo a la figura 7.32.

8 Resistencias de 330Ω.	2 Resistencias de 100Ω.
2 Capacitores de 27 ρF	1 Cristal de 4 MHz
2 Diodo 1N4148	8 Diodos Led
1 Microcontrolador PIC16F877A	2 Pulsador
1 fuente de Voltaje de 5 V.	1Protoboard

ESQUEMA ELÉCTRICO.

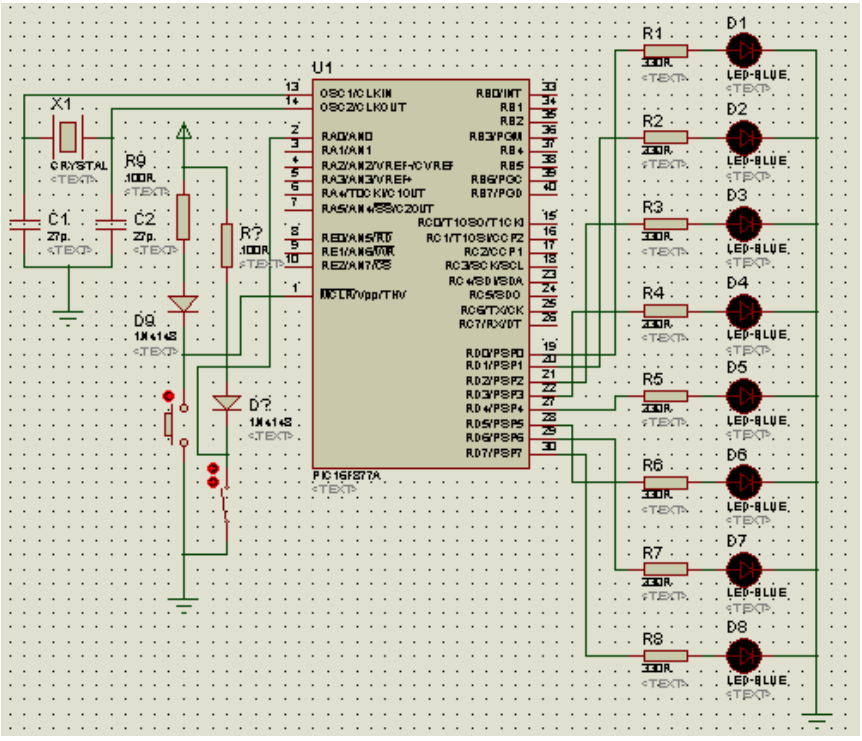


Figura 7.32

Esquema eléctrico para simulación y montaje del proyecto secuencias condicionadas

ALGORITMO

1. Configuración de puertos
2. Inicia contador
3. Si RA0 es igual a 0 entonces envía secuencia_izquierda
4. De lo contrario envía secuencia_derecha
5. Incrementa contador
6. Si contador es igual a 8 entonces contador igual a 0
7. Regresa al punto 3

DIAGRAMA DE FLUJO

Una vez que se tiene el algoritmo se obtiene una idea clara de lo que debe hacer el programa, por lo tanto se continua con la elaboración del diagrama de flujo de la figura 7.33.

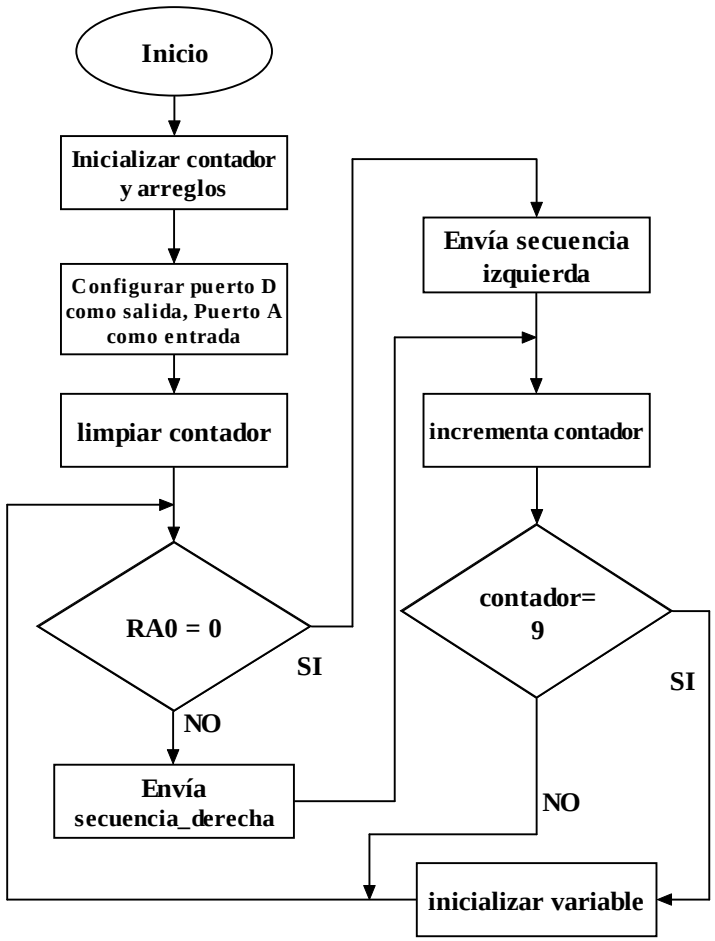


Figura 7.33

Diagrama de flujo del proyecto

CÓDIGO DEL PROGRAMA

Siguiendo la secuencia del diagrama de flujo se lleva a cabo la elaboración del código a ejecutar, el cual aparece en la figura 7.34.

```
short izquierda[9] = { 0, 1, 3, 7, 15, 31, 63, 127, 255 };   //declaración de
short derecha[9] = { 255, 127,63 , 31, 15, 7, 3, 1,0 };      //arreglos
int contador;                          //declaración de variable
void main ( void )
{
TRISD = 0;                             //declara Puerto D como salida
ADCON1 = 6;                            //habilitar el puerto A
TRISA = 0x7F;                          //se configura puerto A0 como entrada
Contador = 0;                          //se inicia contador con 0
for ( ; ; )                            //se inicia ciclo For
{
Delay_ms ( 250 );                      //iniciar un retardo de 250 ms
if ( PORTA.F0 == 0 )                   //verificar valor PORT A0
PORTD = izquierda [ contador];         //si A0=0 inicia secuencia izquierda
else                                   //de lo contrario A0=1 entonces
PORTD = derecha [ contador];           //inicia secuencia derecha
contador ++ ;                          //se incrementa contador por cada ciclo
if ( contador == 9 )                   //si el contador = 9 se reinicia
contador = 0;                          //el contador con 0 y reinicia el ciclo
}                                      // si contador es < 9 continua el ciclo
}
```

Figura 7.34

Código a implementar en mikroC para controlar luces en forma condicionada .

El programa anterior se compila en mikroC con la configuración pertinente explicada al principio de este capítulo. Una vez que se a compilado el código, se simula el mismo en Proteus para verificar su funcionamiento, logradas las expectativas el siguiente paso es la grabación del microcontrolador con el código, y asi montarlo al circuito armado en el protoboard como el que se muestra en la figura 7.35.

Montaje del circuito

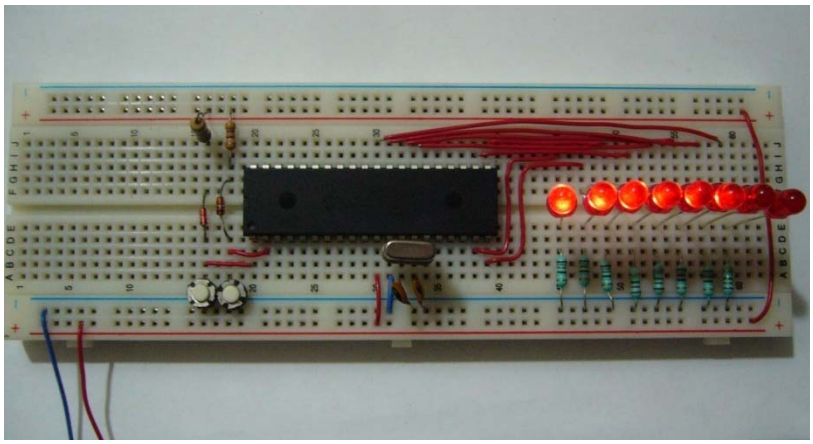


Figura 7.35

Fotografía del circuito sobre un protoboard para verificar el fucionamiento del programa luces condicionadas .

CONTADOR 0 A 9

En base a los proyectos básicos hasta aquí expuestos, se propone comenzar con proyectos de aplicación y en los que se puede verificar la comodidad de programar en lenguaje C.

El objetivo de este proyecto, como su nombre lo indica, es realizar un programa que realice la cuenta de 0 a 9, dicho evento visualiza cada digito en un display de 7 segmentos con un periodo de 1 segundo por evento.

El material a utilizar es el siguiente:

8 resistencia de 330Ω.
2 capacitores de 27 ρF
2 Diodo 1N4148
1 Microcontrolador PIC16F877A
1 fuente de Voltaje de 5 V.
2 resistencia de 100Ω.
1 Cristal de 4 MHz
1 Display de catodo comun
2 Pulsador
1Protoboard

Diagrama eléctrico.

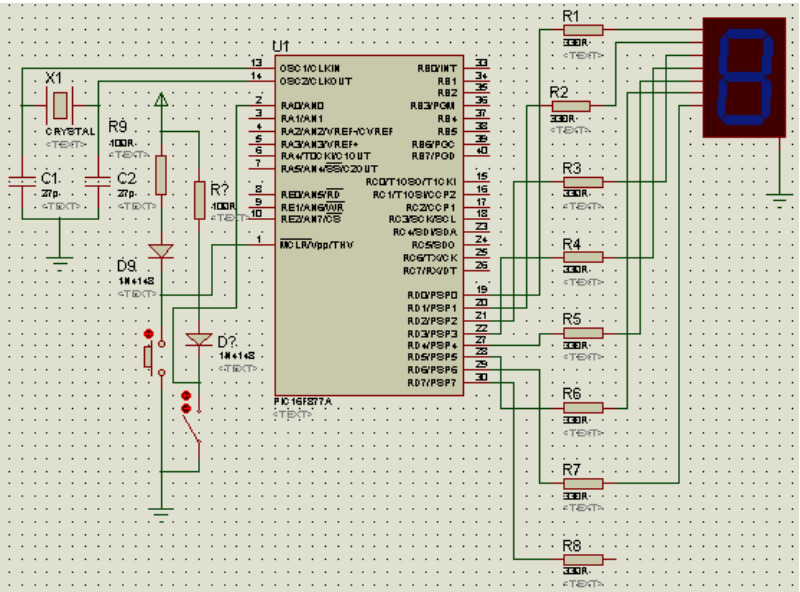


Figura 7.36

Diagrama eléctrico para el contador 0 a 9

Algoritmo

El proyecto se basa en el proyecto “luces secuenciales 2”, entonces se propone el siguiente algoritmo.

1. Configuración de puerto como salida.
2. Inicializa apuntador.
3. Envío de dato apuntado.
4. Incrementa apuntador.
5. Si apuntador es mayor que o igual a 10 reinicia el apuntador.
6. Regresa a 3.

Cada dato del arreglo contendrá código decimal o hexadecimal equivalente a un digito del exhibidor de 7 segmentos de catado común. El compilador mikroC cuenta con una herramienta para facilitar el código deseado para el exhibidor ya sea en ánodo común o cátodo común. En el menú principal del compilador mikroC escoger la opción *Tools* enseguida se desplegara una ventana en donde se debe escoger la opción “Seven Segment Convert”. A continuación mostrará una pantalla con un display de 7 segmentos en blanco, dar “clic” en cada segmento según sea el numero que se desea exhibir. En la misma ventana se observan campos que se llenan automáticamente conforme se escoge cada segmento, y es aquí aparece el código de 7 segmentos decodificado

a decimal o hexadecimal según se puede elegir en la parte inferior a dichos campos. Ver figura 7.37.

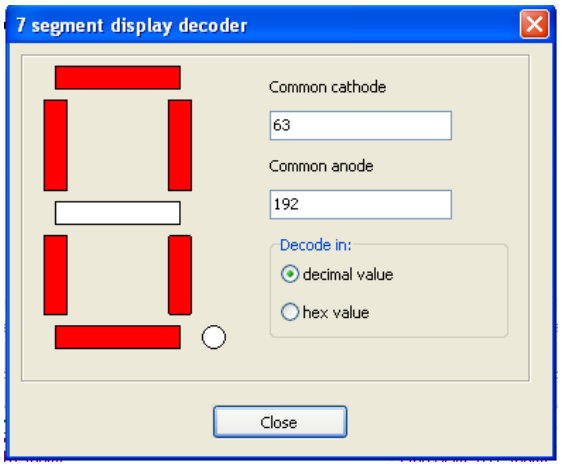


Figura 7.37

Pantalla del decodificar 7 segmentos a código decimal o hexadecimal

Diagrama de flujo

El diagrama de flujo es el mostrado en la figura 7.38 según el algoritmo propuesto.

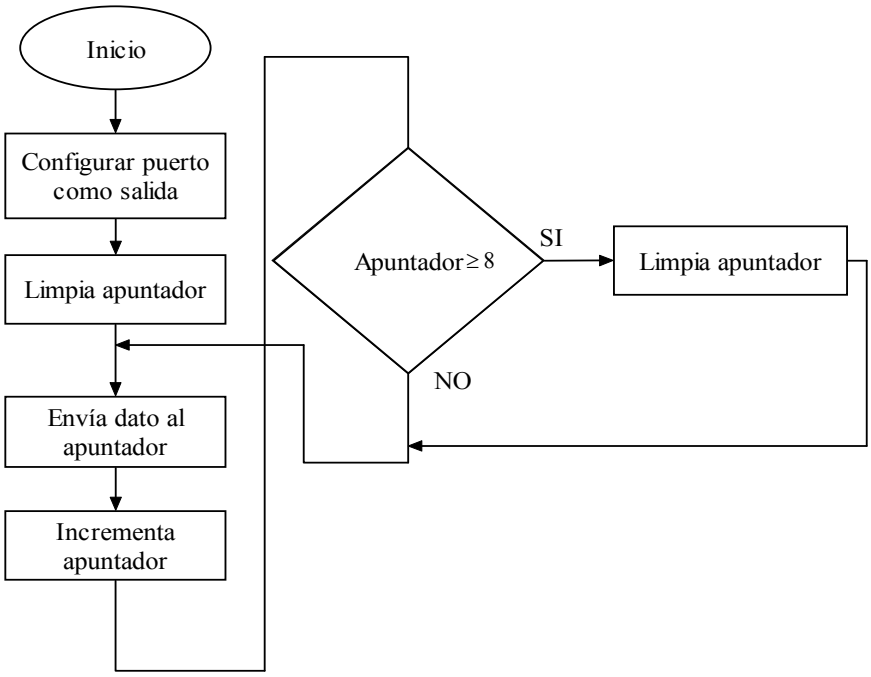


Figura 7.38

Diagrama de flujo para codigo contador 0 a 9.

Código del programa.

```
short dato [ 10 ] = {63, 6, 91, 79, 102, 109, 125, 7,127,111};
short apunta;
void main(void)
{
TRISD = 0;                                  // Configura puerto
apunta = 0;                                 // Limpia apuntador
while(1)                                    // Ciclo infinito
   {
   PORTD = dato [ apunta ];                 // Envía dato
   Delay_ms(1000);
   apunta ++;                               // Incrementa
   apuntador    if ( apunta >=  10 )        // Si apuntador = 8
   apunta = 0;                              // Limpia apuntador
   }
}
```

Figura 7.39

Código a ejecutar para contador 0 a 9

Tomando como base los programas anteriores resulta sencillo entender el código que se muestra en la figura 7.39. Dado que el arreglo contiene todos los datos que corresponden a los números 0 a 9 pero en su equivalente a código de 7 segmentos, cada vez que se apunte a uno de los datos se visualizará su correspondiente en el display del circuito montado con una duración de 1 segundo.

A continuación se debe compilar el código en mikroC con los datos pertinentes, simular y programar al microcontrolador para llevarlo al circuito de prueba en el protoboard.

FOTOGRAFIA DEL CIRCUITO.

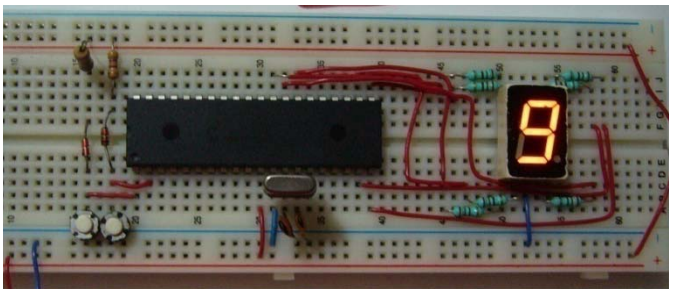


Figura 7.40

Fotografía del circuito sobre protoboard .para visualizar el conteo de 0 a 9.

CONTADOR 00 A 99

Este programa realiza un evento de conteo incremental iniciando en 00 y termina en 99. La cuenta se visualiza en dos displays de 7 segmentos de ánodo común.

El material a utilizar es el siguiente:

7 Resistencia de 470Ω.
2 Capacitores de 27 ρF
1 Diodo 1N4148
1 Microcontrolador PIC16F877A
1 Fuente de Voltaje de 5 V.
2 Resistencias de 4.7 K Ω
2 resistencias de 1 K Ω
1 resistencia de 100Ω.
1 Cristal de 4 MHz
2 Display de catodo común
1 Pulsador
1Protoboard
2 transistores BC558

DIAGRAMA ELÉCTRICO.

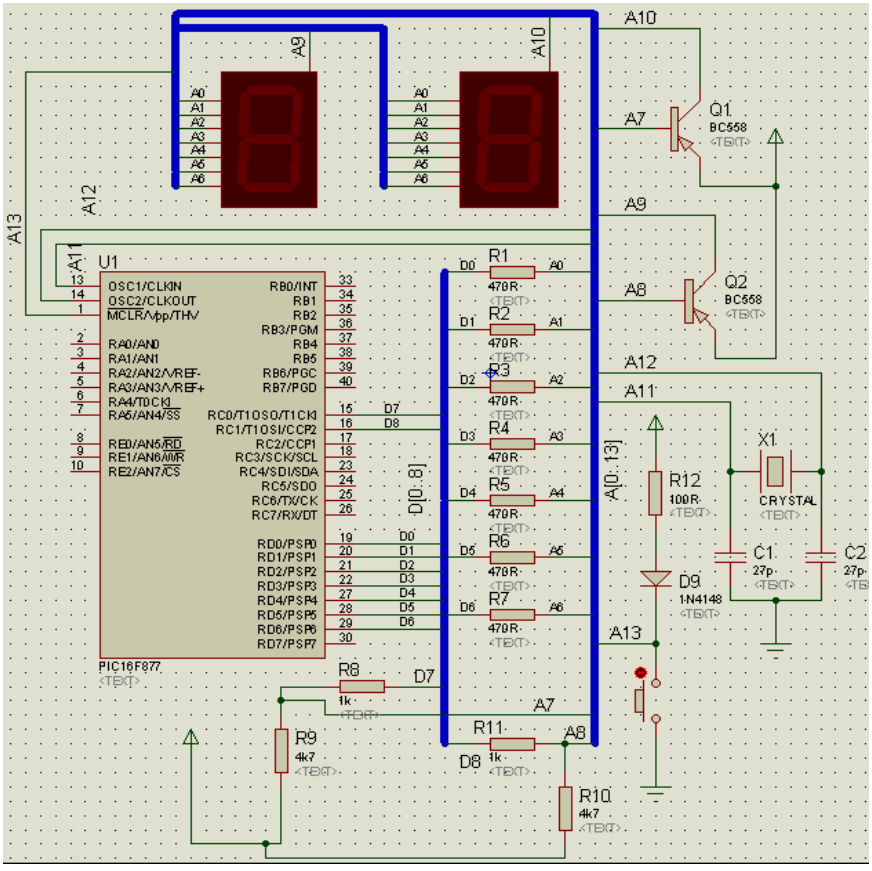


Figura 7.41

Diagrama eléctrico para simular en Proteus e implementar en protoboard .

El circuito mostrado en la figura 7.41 se observa el uso de 2 display de 7 segmentos para la visualización de el conteo, de acuerdo a la

configuración de los transistores es fácil deducir que se tratan de ánodo común. También se pueden utilizar displays de catado común realizando la configuración correcta.

ALGORITMO

1. Declarar variables
2. Configurar puertos
3. Iniciar contador.
4. Obtener decenas y unidades.
5. Visualizar decenas.
6. Visualizar unidades.
7. Si el contador es mayor a 99 reinicia el contador
8. Ir a 4.

DIAGRAMA DE FLUJO

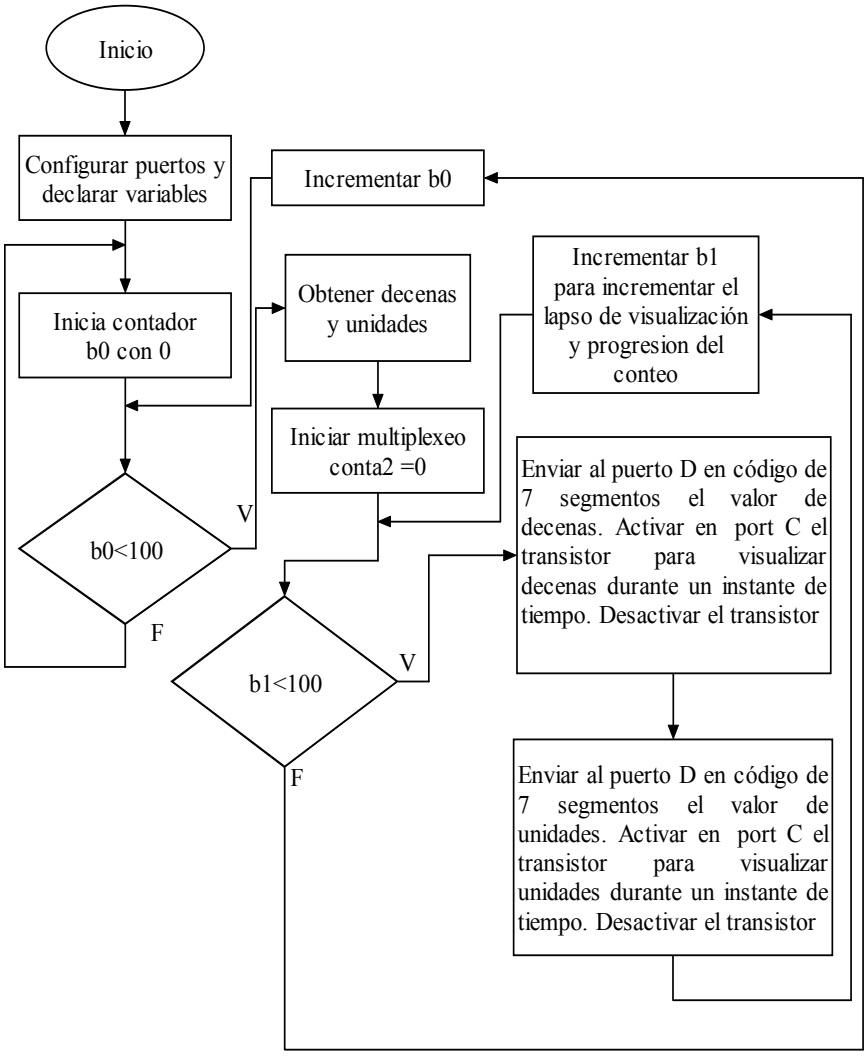


Figura 7.42

Diagrama de flujo .en su implementación se observa el uso de ciclos for para el desrrollo del programa.

CÓDIGO DE PROGRAMA

El programa se desarrolla de la siguiente manera de acuerdo al diagrama de flujo, no siendo la única forma de llevar a cabo el desarrollo del mismo

```
//declarar  función get_value.
unsigned char get_value(unsigned char  y);
unsigned char  y;
//Declaración de variables globales
unsigned char tens;              //declara decenas como unsigned char.
unsigned char units;             //declara unidades como  unsigned char
unsigned int b0;                 //declara b0 como unsigned integer
Unsigned char b1;                //declara b1 como unsigned character.
                    //comienza el programa principal
      void main(void)
{
  trisc=0;                       //portC como salida.
   portc=0;                      //se colocan las terminales del portC en bajo.
  trisd=0;                       //portD como salida.
  portd=0;                       //se colocan las terminales del portD en bajo.

  while(1)                       //se inicia un loop infinito.
  {
  for (b0=0;b0<100;b0++)         //ciclo for para generar 00-99.
  {
  decenas=(b0/10);               //se obtiene el valor de las decenas.
  unidades=b0%10;                //se obtiene el valor de las unidades.
 //Aquí se realiza el multiplexado
for (b1=0;b1<100;b1++)           //se muestran los datos cien veces.
{
portd=get_value(decenas);        //Se obtiene el código 7 segmentos para las
                                 //decenas y se envía al puerto d.
portc.F1=1;                      //Se enciende el transistor (D).
delay_ms(1);                     //pausa de un ms para ver los datos.
portc.F1=0;                      //Se apaga el transistor (D).
```

Figura 7.43

Código del programa Contador 00-99 a compilar en mikroC

Figura 7.43 (continuacion)

```
portd=get_value(unidades);      //Se obtiene el código 7 segmentos para las
                                //unidades  y se envía al puerto d.
portc.F0=1;                     //Se enciende el transistor (U).
delay_ms(1);                    //pausa de un ms para ver los datos.
portc.F0=0;                     //Se apaga el transistor (U).
}
}
}
}
//Tabla de búsqueda para código de cada digito o segmento.
 unsigned char get_value(unsigned char y)
{
 unsigned char segments[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
 return segments[y];
}
```

Una vez compilado el programa en mikroC, se realiza la simulación en Proteus, para así cerciorarse que el programa funcionará correctamente sobre el montaje de prueba como el que se muestra en la figura 7.44, previa programación del microcontrolador.

En este programa se observa que se utilizan ciclos *for* para la generación del conteo así como para realizar el multiplexeo y tener un control de la velocidad del conteo. También se observa que la función que genera el código de 7 segmentos contiene datos para displays de ánodo común los cuales se deben modificar para el caso de utilizar displays de cátodo común. Siendo este el caso, entonces también se debe modificar la etapa de multiplexeo.

Modificar este programa utilizando ciclos *if* y verificar el funcionamiento del mismo, así como la generación de código, es decir la cantidad de memoria utilizada.

FOTOGRAFÍA DEL MONTAJE

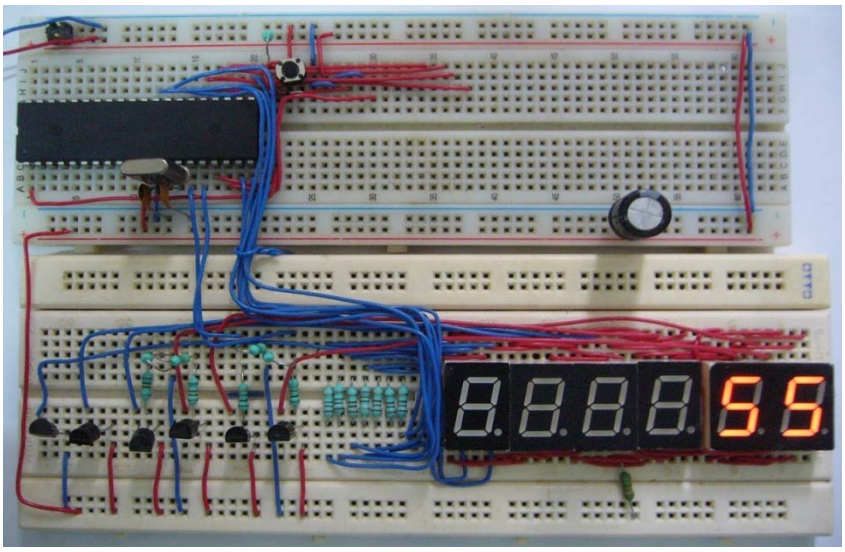


Figura 7.44

Fotografía del circuito sobre protoboard para verificar el programa cotador 00-99.

CONTADOR 000 A 999

En este programa se utiliza la misma lógica que el programa anterior, solo se agregan algunas líneas de código que se aprecian en el desarrollo del código.

EL MATERIAL A UTILIZAR ES EL SIGUIENTE:

7 Resistencia de 470Ω.
2 Capacitores de 27 ρF
1 Diodo 1N4148
1 Microcontrolador PIC16F877A
1 Fuente de Voltaje de 5 V.
3 Resistencias de 4.7 K Ω
3 resistencias de 1 K Ω
1 resistencia de 100Ω.
1 Cristal de 4 MHz
3 Displays de cátodo común
1 Pulsador
1Protoboard
3 transistores BC558

DIAGRAMA ELÉCTRICO.

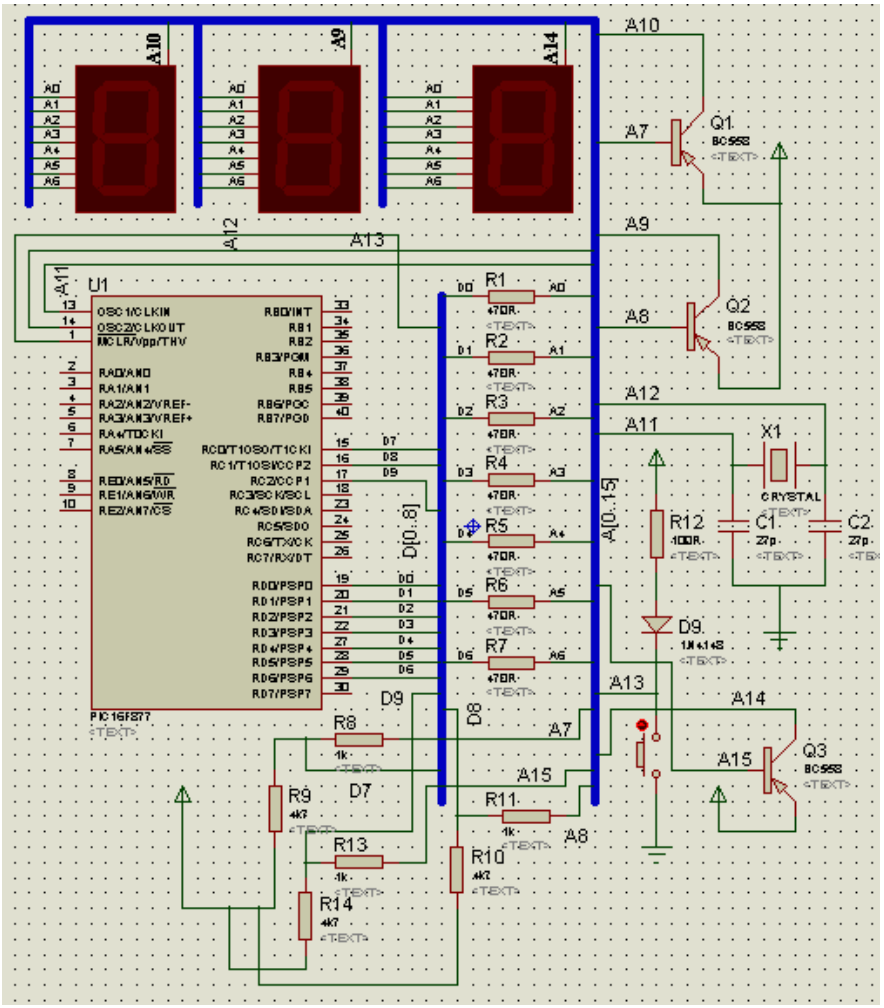


Figura 7.45

Diagrama eléctrico para simular en Proteus e implementar sobre un protoboard .

ALGORITMO

1. Declarar variables
2. Configurar puertos
3. Iniciar contador.
4. Obtener decenas y unidades.
5. Visualizar decenas.
6. Visualizar unidades.
7. Incrementar contador.
8. Ir a 4.
9. Si el contador es mayor a 999 reinicia el contador

DIAGRAMA DE FLUJO

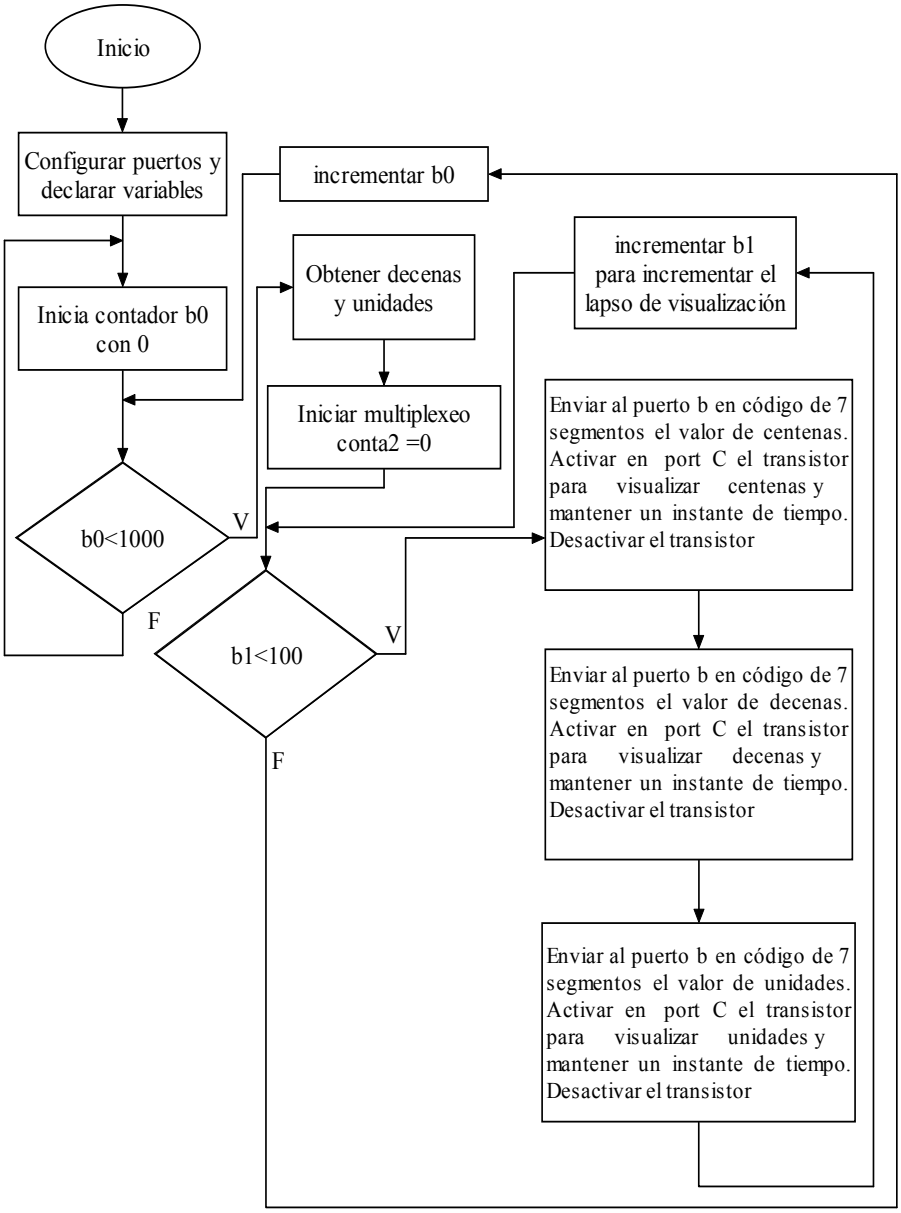


Figura 7.46

Diagrama de flujo conservando una lógica semejante al de la figura 42.

CÓDIGO DE PROGRAMA

De acuerdo al diagrama de flujo de la figura 7.46 el programa a compilar es el que se muestra en la figura 7.47.

```
// declarar la funcion get_value.
unsigned char get_value(unsigned char y);
unsigned char y;
//declaracion de variables globales
unsigned char unidad;
unsigned char decena;
unsigned char centena;
unsigned char miles;
      unsigned int t;
//se inicia el programa principal
void main(void)
{
 trisc=0;                  //se declara portc como salida
 portc=0;
 trisd=0;                  //se declara el portd como salida
 portd=0;
 while(1)
{
 unsigned int b0;          //Declarar b0 como unsigned integer variable.
 unsigned char b1;         //Declare b1como unsigned character.
for (b0=0;b0<1000;b0++)    //Ciclo for para generar 0-999.
{
 centena=b0/100;           //Se obtiene el valor de las centenas.
 decena=(b0%100)/10;       //Se obtiene el valor de las decenas.
 unidad=b0%10;             //Se obtiene el valor de las unidades.
//en este bloque se realiza el multiplexado
for (b1=0;b1<100;b1++)  //
{
portd = get_value(centena);   //se obtiene el codigo para las centenas  y se envía
                              //al puerto d
 portc.F0=0;                  //se enciende el transistor (C)
delay_ms(1);                  //durante un ms
 portc.F0=1;                  //se apaga el transistor (C)
portd= get_value(decena);     //se obtiene el codigo para las decenas y se envía
                              //al puerto d
portc.F1=0;                   //se enciende el transistor (D)
delay_ms(1);                  //durante un ms
 portc.F1=1;                  //se apaga el transistor (D)
portd= get_value(unidad);     //se obtiene el codigo para las unidades y se envía
                              //al puerto d
 portc.F2=0;                  //se enciende el transistor (U)
delay_ms(1);                  //durante un ms
portc.F2=1;                   //se apaga el transistor (U)
```

Figura 7.47

Código del programa Contador 000-999 a compilar en mikroC .

Figura 7.47 (continuacion)

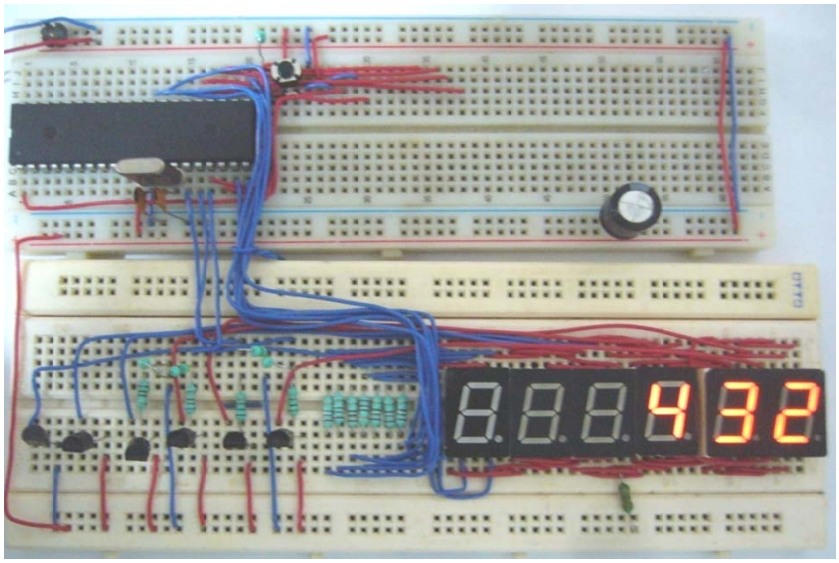
Código del programa Contador 000-999 a compilar en mikroC .

```
        }
      }
    }
}
/*en esta funcion se genera la tabla que contiene
cada digito codificado en codigo de siete segmentos */
unsigned char get_value(unsigned char y)
{
unsigned char segments[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
return segments[y];
}
```

FOTOGRAFÍA DEL MONTAJE

Figura 7.48

Fotografía del circuito sobre protoboard mostrando un evento del programa en funcionamiento "contador 000-999".



ADQUISICIÓN DE DATOS ANALÓGICOS

Este programa en forma básica ejemplifica la utilización de la librería *ADC* (ADC library) para posteriormente utilizar la misma en la medición de pequeños voltajes. En este programa se muestra la adquisición de una señal analógica (0 – 5 volts) y muestra su equivalente en código de 10 bits (1023 bcd).

El material a utilizar es el siguiente:

10 Resistencia de 330 Ω ó 470Ω.	1 resistencia de 100Ω.
2 Capacitores de 27 ρF	1 Cristal de 4 MHz
1 Diodo 1N4148	10 Leds
1 Microcontrolador PIC16F877A	1 Pulsador
1 Fuente de Voltaje de 5 V.	1Protoboard
1 potenciometro de 1 K Ω	

DIAGRAMA ELÉCTRICO.

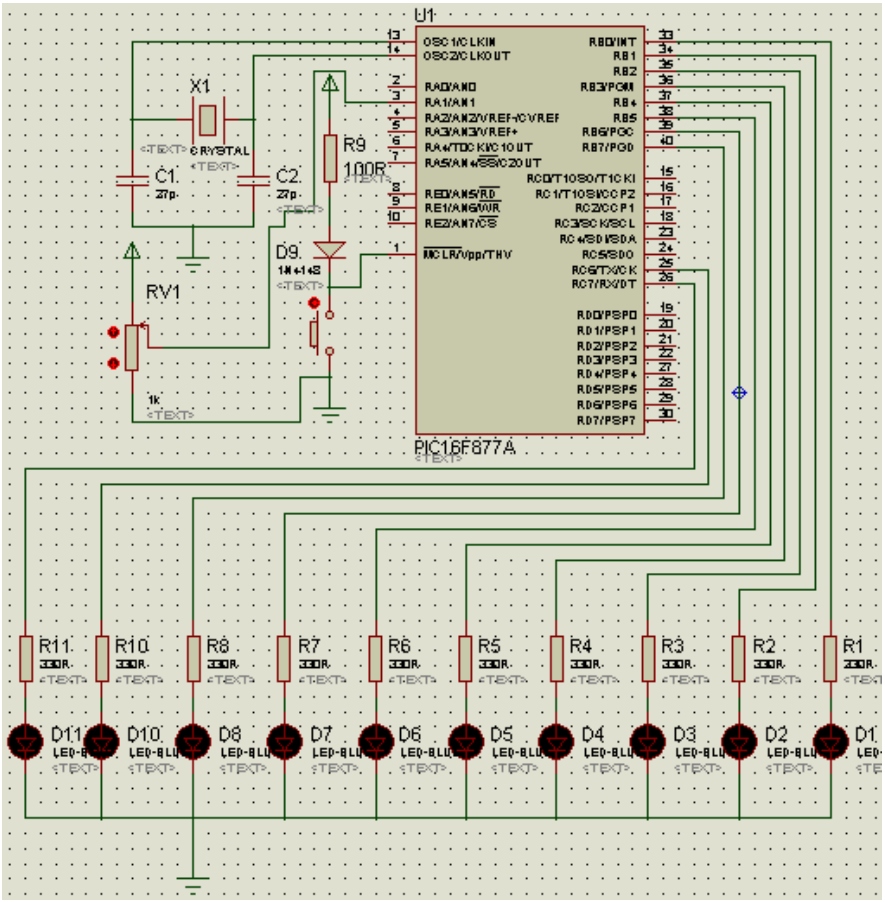


Figura 39

Diagrama eléctrico para simular en Proteus e implementar en protoboard .

ALGORITMO

1. Declarar variables
2. Configurar puertos y registros
3. Habilitar convertidor analógico a digital.
4. Obtener dato analógico.
5. Mostrar dato LSB.
6. Mostrar dato MSB.
7. Regresar a 4.

DIAGRAMA DE FLUJO

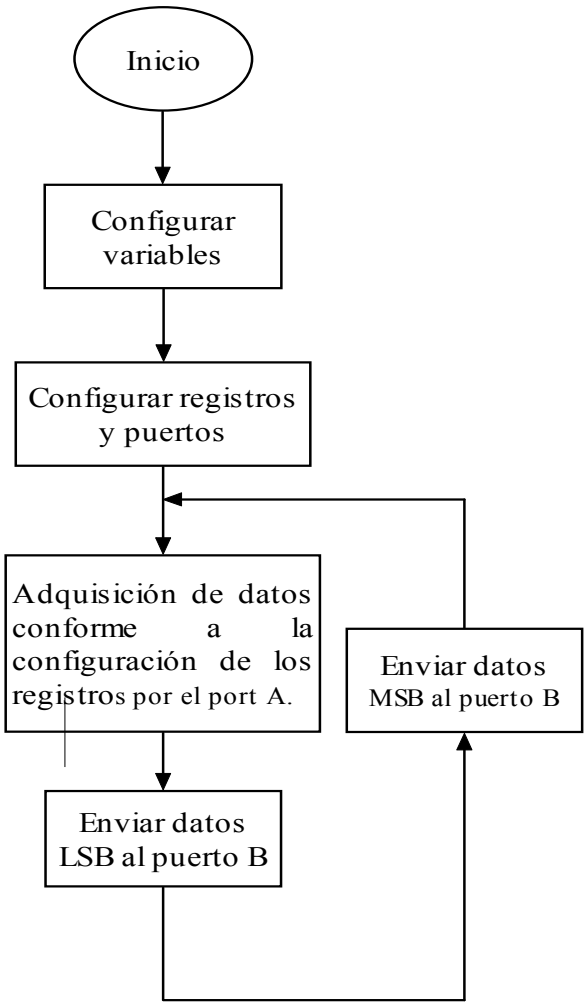


Figura 7.50

Diagrama de flujo para generar el código del programa de conversión analógico a digital.

CÓDIGO DE PROGRAMA

Otra de las ventajas que presenta el compilador mikroC es que tiene una serie de librerías implementadas que provocan que la programación se torne sencilla, tal es el caso la librería ADC (library ADC). Para hacer uso de ella solo basta con seguir el protocolo descrito en capitulo X o referirse al manual de usuario de mikroC. En este programa se hace uso de dicha librería y se observa que solo se necesitan pocas líneas de código para llevar a cabo la programación y así poder adquirir datos analógicos en el puerto A del microcontrolador. En la figura 7.51 se muestra el programa explicado con comentarios para compilar en

mikroC. En seguida en la figura 7.52 se observa el circuito de prueba armado sobre un protoboard.

```
/*ESTE PROGRAMA ADQUIERE DATOS ANALOGICOS EN EL PUERTO A Y LOS CONVIERTE A DATOS DIGITALES MOSTRANDOLOS EN LOS PUERTO B Y D RESPECTIVAMENTE*/
unsigned temp_res; //variable tipo unsigned
//comienza el programa principal
void  main()
{
ADCON1=0X80;          // se habilita el convertidor AD
 TRISA=0XFF;          //se configura el puerto A como entrada
 TRISB=0;             //se configura puerto B como salida
 TRISC=0X3F;          // se configura terminales C6 y C7 como salida
 Do
 {
 temp_res=Adc_Read(1); //se adquiere el dato analógico en la terminal A1
 PORTB=temp_res;       //se envían los bits menos significativos al puerto
                       //B
 PORTC=temp_res>>2;    //se visualizan los 2 bits más significativos en el
                        //puerto C
 } while (1) ;
}
```

Figura 7.51

Código del programa Adquisición de Datos analógicos para compilar en mikroC .

FOTOGRAFÍA DEL MONTAJE

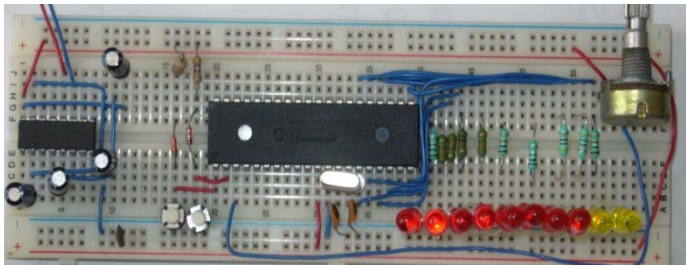


Figura 7.52

Fotografía del circuito sobre protoboard el dato ADC en formato de 10 bits.

DATOS SOBRE DISPLAY LCD

El objetivo de este proyecto es enviar un dato o mensaje a una pantalla LCD. Se observara que el código escrito en C es breve y sin complicaciones, solo se debe utilizar la librería adecuada ya sea para LCD's de 8 bit's o 4 bit's según sea el caso.

El material a utilizar es el siguiente:

1 Display LCD 2 X16.	1 resistencia de 100Ω.
2 Capacitores de 27 ρF	1 Cristal de 4 MHz
1 Diodo 1N4148	
1 Microcontrolador PIC16F877A	1 Pulsador
1 Fuente de Voltaje de 5 V.	1Protoboard

DIAGRAMA ELÉCTRICO.

El siguiente diagrama eléctrico de la figura 7.53 es el que se utiliza para su simulación y para montarse en el circuito de prueba.

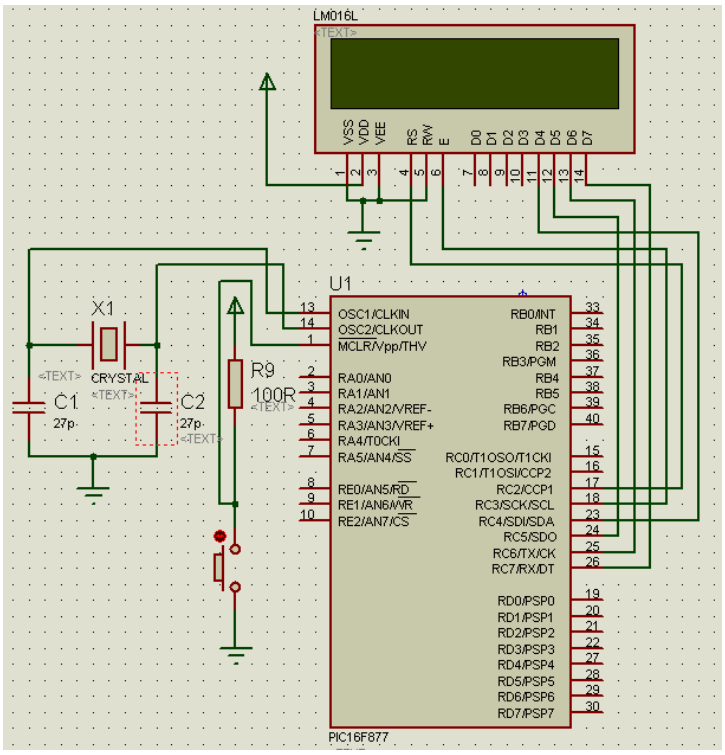


Figura 7.53

Diagrama eléctrico para simular en Proteus e implementar en protoboard .

ALGORITMO

Dada la versatilidad de mikroC, se plantea el siguiente algoritmo

1. Declarar variables.
2. Configurar puerto con comandos LCD (interface a 4 bits).
3. Enviar dato.

A continuación en la figura 7.54 se desarrolla el diagrama de flujo pertinente al algoritmo anterior.

DIAGRAMA DE FLUJO

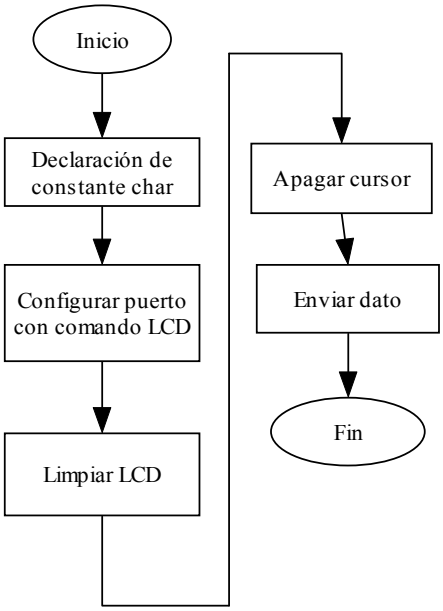


Figura 7.54

Diagrama de flujo para implementar el programa datos sobre display LCD

CÓDIGO DE PROGRAMA

El diagrama de flujo da la pauta y ratifica que el código a generarse es sencillo y solo se deben seguir las instrucciones de la librería LCD con interface de 4 bit tal como se muestra en la figura 7.55. El circuito de prueba se muestra en la figura 7.56.

```
/*ESTE PROGRAMA MUESTRA COMO ENVIAR DATOS A UN
DISPLAY LCD UTILIZANDO EL PUERTO C*/
char *text = "Ing. Electronica";        // se declara una constante de tipo char
void  main()
{
LCD_Init(&PORTC);                       // inicializa LCD conectado a PORTC
LCD_Cmd(LCD_CLEAR);                     // limpiar display
LCD_Cmd(LCD_CURSOR_OFF);                // apaga el cursor
LCD_Out(1,1, text);                     // imprime lo q contiene texto en,
                                        //1er renglón, 1er columna
 Delay_ms(1000);
     }
```

Figura 7.55

Código del programa Datos sobre display LCD para compilar en mikroC .

FOTOGRAFÍA DEL MONTAJE

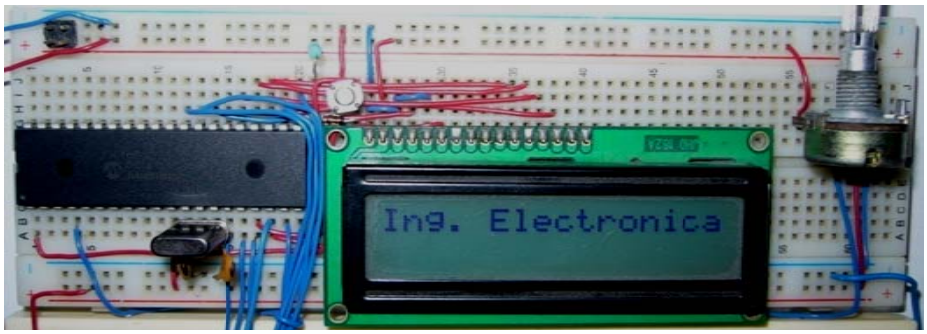


Figura 7.56

Fotografía del circuito sobre protoboard mostrando texto en un display LCD.

DATOS ADC EN DISPLAY DE 7 SEGMENTOS

En base a los programas realizados anteriormente se propone realizar un programa que adquiera datos analógicos (0-5 Volts) por alguno de los canales analógicos y los muestre en un arreglo de 4 displays de 7 segmentos de ánodo común como el que se observa en el diagrama de la figura 7.57.

El material a utilizar es el siguiente:

7 resistencias de 470 Ω	1 Resistencia de 100Ω.
2 Capacitores de 27 ρF	1 Cristal de 4 MHz
1 Diodo 1N4148	4 Displays de 7 segmentos ánodo común
1 Microcontrolador PIC16F877A	1 Pulsador
1 Fuente de Voltaje de 5 V.	1Protoboard
1 potenciómetro de 1 K Ω	4 resistencias de 1KΩ
4 transistores BC558	4|resistencias de 4.7K Ω

DIAGRAMA ELÉCTRICO.

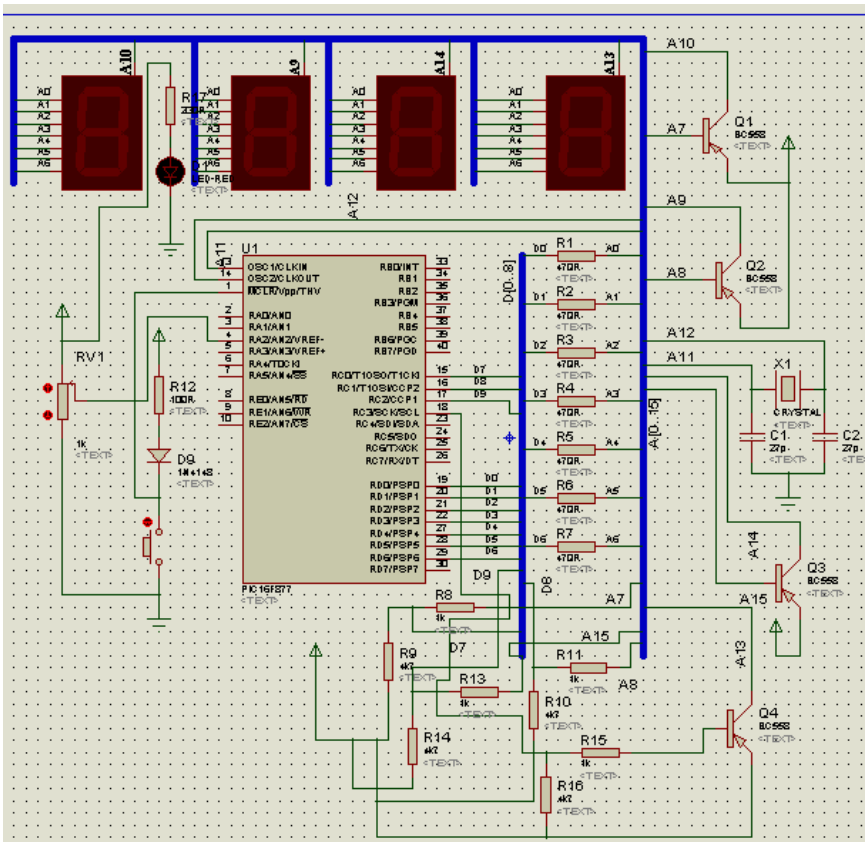


Figura 7.57

Diagrama eléctrico para simular en Proteus e implementar en protoboard.

ALGORITMO

1. Declarar variables
2. Configurar puertos y registros
3. Habilitar convertidor analógico a digital.
4. Obtener dato analógico.
5. Convertir a formato decimal
6. Adquirir parte entera, decimas, centésimas y milésimas
7. Convertirlas a código de 7 segmentos..
8. Enviar al display.
9. Regresar a 4

DIAGRAMA DE FLUJO

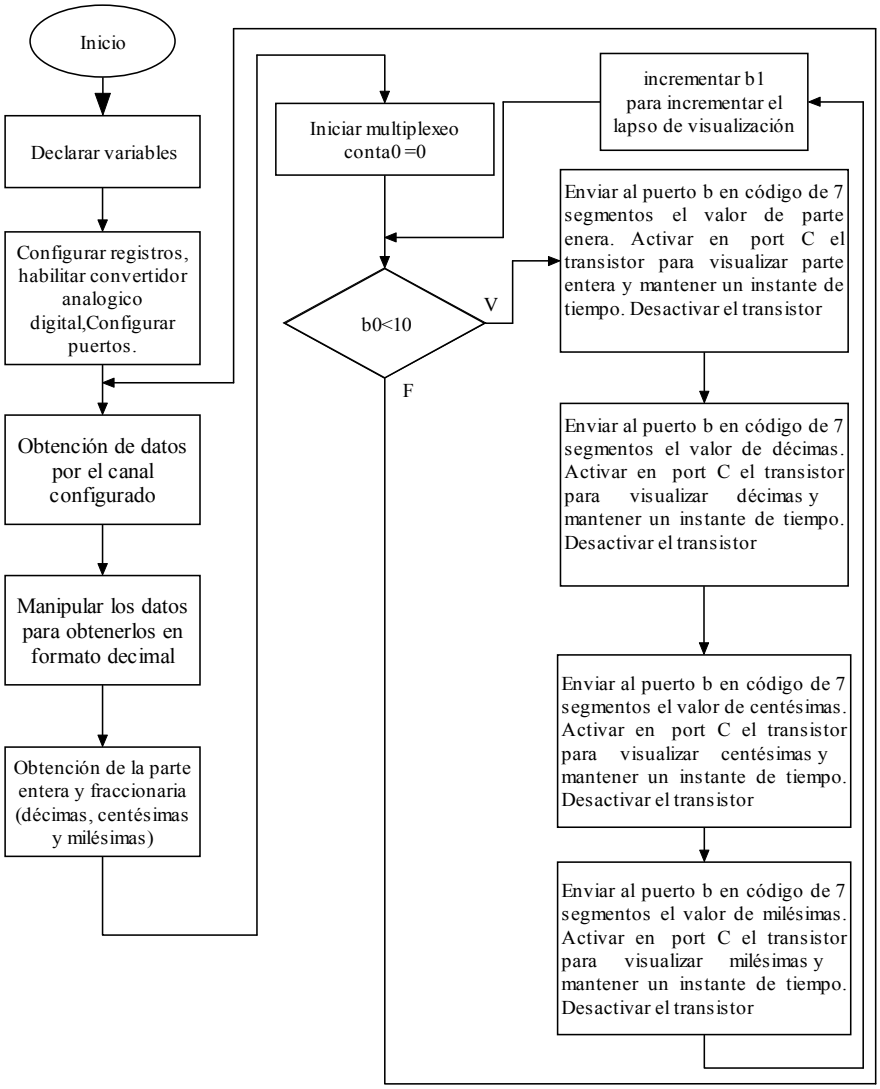


Figura 7.58

Diagrama de flujo. En su implementación se observa que evoca el uso de ciclos for para llevar a cabo el programa.

CÓDIGO DE PROGRAMA

Siguiendo la lógica del diagrama de flujo de la figura 7.58, se propone el código de la figura 7.59 para compilar en mikroC.

```
//declare function get_value.
unsigned char get_value(unsigned char y);
unsigned char y;
//declaracion de variables globales
unsigned char  mil;
unsigned char  cent;
unsigned char  dec;
unsigned char  volt;
long tlong;
unsigned int t;

//comienza el programa principal
void main(void)

{
//configuracion de registros y puertos
intcon=0;                   //se deshabilitan todas las interrupciones
//option_reg = 0x80;        //desactivar resistencias pull-up si se usa portB
adcon1=0x82;                //configura VDD como Vref y canales analógicos
trisa=0xff;                 //designa el puerto A como  entrada
trisc=0;                    //designa el puerto C como salida
portc=0;                    //coloca  el puerto C en 0
trisd=0;                    //configura el puerto D como salida
portd=0;                    //coloca el puerto D en 0

 //se inicia un ciclo infinito
while(1)
{
 unsigned char b1;

 t=adc_read(2);             // obtiene el dato analógico del canal 2

 tlong = t*5000;            //multiplicación de tipos int en vez de long
 /* en el siguiente bloque se hace uso de la instrucción asm para utilizar código ensamblador y así llenar manualmente los bits más significativos de la variable tlong  */

asm{
   movf stack_2,w
   movwf _tlong+2
   movf stack_3,w
   movwf _tlong+3
   }
 t=tlong>>10;               //se recorren 10 bits a la derecha

 volt=t/1000;               //obtención de cada volt
 dec=(t/100)%10;            // obtención de decimas de volt
 cent=t%10;                 //obtención de centésimas de volt
                            //obtención de milésimas de volt
mil=((t%1000)%100)%10;
      for                   //ciclo para mostrar datos en los display
(b1=0;b1<10;b1++)
        {
         portd              = //se obtiene el equivalente volt en código 7segmentos
```

Figura 7.59

Código propuesto para adquirir datos analógicos y mostrarlos en display de 7 segmentos.

Continuación Figura 7.59

```
get_value(volt);
        portc.F0=1;             // se enciende el transistor (V)
        delay_ms(1);                    // se mantiene por 1ms
        portc.F0=0;                     // se apaga el transistor (V)
        portd=get_value(dec);
        portc.F1=1;
        delay_ms(1);
        portc.F1=0;
        portd=get_value(cent);
        portc.F2=1;
        delay_ms(1);
        portc.F2=0;
        portd=get_value(mil);
        portc.F3=1;
        delay_ms(1);
        portc.F3=0;
      }
      }
      }

      //tabla de equivalencias en código de 7 segmentos
      unsigned char get_value(unsigned char y)
      {
      unsigned                                                    char
segments[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
      return segments[y];
      }
```

FOTOGRAFÍA DEL MONTAJE

La fotografía 7.60 muestra el circuito montado en un protoboard, además se observa que el microcontrolador esta mostrando el valor del voltaje adquirido por el canal analógico A2.

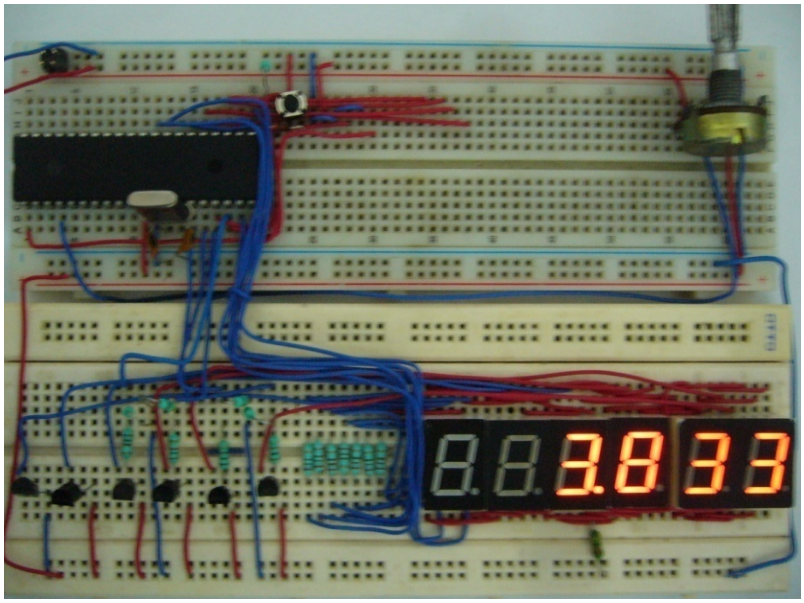


Figura 7.60

Fotografía del circuito en un protoboard desplegando el valor de un voltaje en un rango de 0.000 a 5.994 Volts .

DATOS ADC EN DISPLAY LCD

Modificando el programa de la figura 7.55 y tomando como base el programa de la figura 7.59 se implementa el programa Datos ADC sobre display LCD, ahora los datos analógicos se mostraran en un display LCD de 2 X 16. Para la conversión a caracteres se hace uso del código ASCII.

El material a utilizar es el siguiente:

1 Display LCD 2 X16.
2 Capacitores de 27 ρF
1 Diodo 1N4148
1 Microcontrolador PIC16F877A
1 Fuente de Voltaje de 5 V.
1 resistencia de 100Ω.
1 Cristal de 4 MHz
1 Potenciometro de 1 K Ω
1 Pulsador
1Protoboard

DIAGRAMA ELÉCTRICO.

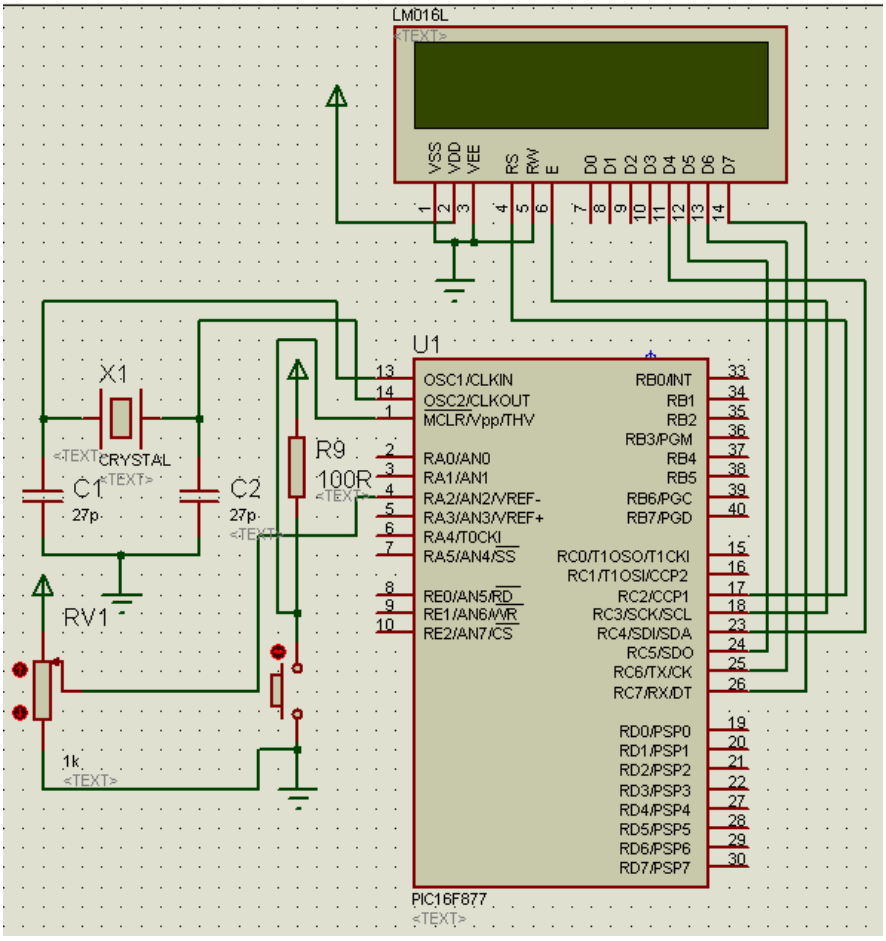


Figura 7.61

Diagrama eléctrico para simular en Proteus e implementar en protoboard .

ALGORITMO

1. Declarar variables
2. Configurar interface de 4 bits
3. Configurar puertos y registros
4. Habilitar convertidor analógico a digital.
5. Obtener dato analógico.
6. Convertir a formato decimal
7. Adquirir dato a dato(parte entera, decimas, centésimas y milesimas)
8. Convertir cada dato a su equivalente ASCII.
9. Enviar a datos a display LCD.
10. Regresar a 4

DIAGRAMA DE FLUJO

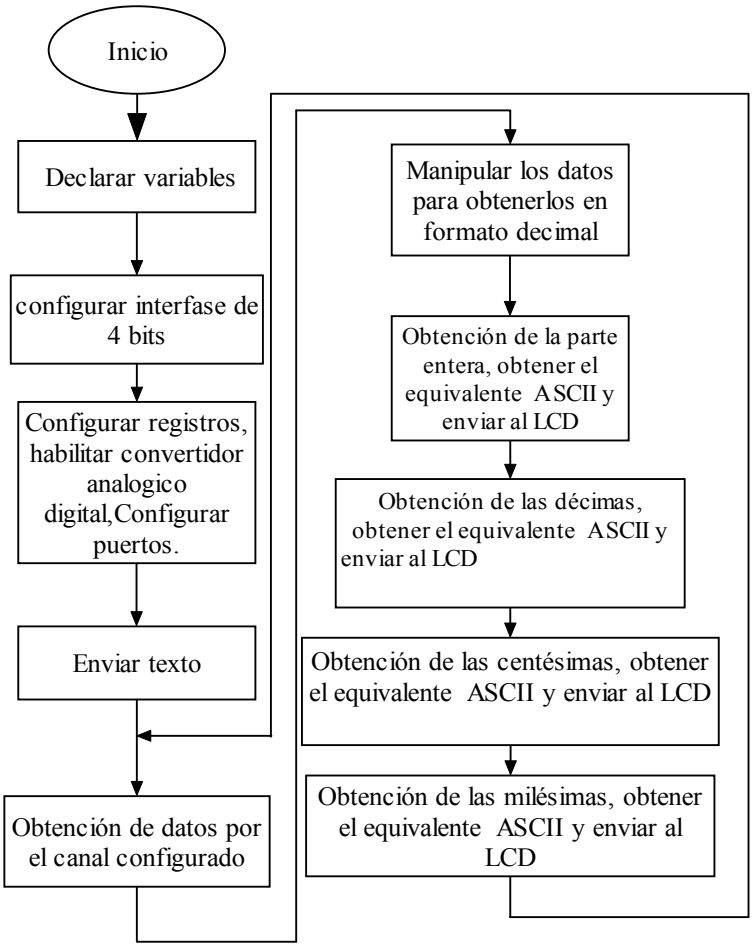


Figura 7.62

Diagrama de flujo.

CÓDIGO DE PROGRAMA

A partir del diagrama a bloques de la figura 7.62 se desarrolla el código de la figura 7.63 para visualizar datos ADC en el display LCD.

```
//Declaracion de variables globales
unsigned char ch;
unsigned int t;
char a[17], *tc;
long tlong;
//inicia programa principal
void main() {
  INTCON = 0;                    // deshabilitar todas las interrupciones
  LCD_Init(&PORTc);              // iniciar  (interface  de conexión a 4 bits)
LCD_Cmd(LCD_CURSOR_OFF);         // enviar comando a LCD (cursor off)
 LCD_Cmd(LCD_CLEAR);             // enviar comando a LCD (clear LCD)
  tc = "Ing Electronica";        // asignar texto a la cadena tc
  LCD_Out(1,1,tc);               // imprimir la cadena en LCD, 1er renglón,
                                 //primera columna
  tc = "LCD ejemplo";            // asignar texto a la cadena tc
  LCD_Out(2,1,tc);               // imprimir la cadena en LCD, 2o renglón,
                                 //primera columna
  OPTION_REG  = 0x80;
  ADCON1     = 0x82;             //configura VDD como Vref, y canales //analógicos
  TRISA      = 0xFF;             // designar el porta como entrada
  TRISC  = 0;
  TRISb  = 0;
  Delay_ms(2000);
  tc  = "voltage:";              // asignar texto a la cadena tc
  while (1) {
    t  = ADC_read(2);            // obtener el voltaje  del canal analógico A2
    LCD_Out(2,1,tc);             // imprimir la cadena en LCD, 2o renglón, primera
                                 //columna
    tlong  = t * 5000;           // usar multiplicación (int)  en vez de (long)
    asm {                        //y finalmente llenar los bytes superiores
                                 //manualmente
          MOVF STACK_2,W
          MOVWF _tlong+2
          MOVF STACK_3,W
          MOVWF _tlong+3
    }
    t = tlong >> 10;
    ch     = t / 1000;           // preparar el valor para el display
LCD_Chr(2,9,48+ch);              // escribir el equivalente ASCII en 2o renglón , 9ª
                                 //columna
    LCD_Chr_CP('.');
          ch    = (t / 100) % 10;
          LCD_Chr_CP(48+ch);
          ch    = (t / 10) % 10;
          LCD_Chr_CP(48+ch);

          ch    = t % 10;
          LCD_Chr_CP(48+ch);
          LCD_Chr_CP('V');

          Delay_ms(1);
        }
      }
```

Figura 7.63

Código para compilar en mikroC. Se observa el uso de comandos de la librería LCD library 4Bits

FOTOGRAFIA DEL MONTAJE

La figura 7.64 muestra el circuito armado sobre un protoboard en el cual se puede observar un mensaje en el display y a continuación la lectura de un voltaje aplicado al canal RA2

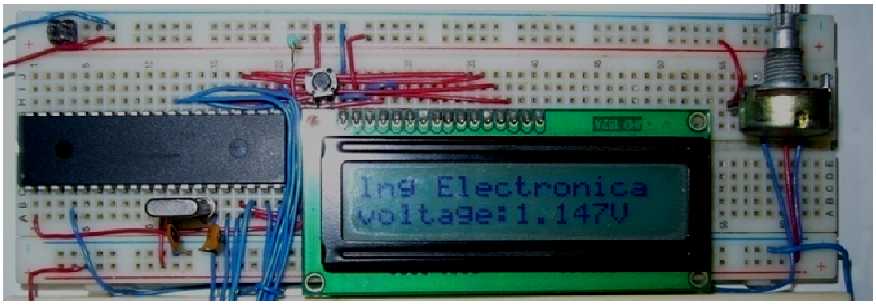


Figura 7.64

Fotografía del montaje del circuito sobre una placa protoboard.

A manera de conclusión se hace hincapié que los proyectos aquí expuestos son solo algunos de los muchos que se pueden realizar utilizando el compilador mikroC. Hasta este punto se puede apreciar la versatilidad con la que se puede trabajar con este compilador, pues cuenta con muchas librerías que se pueden utilizar con el microcontrolador PIC16F877A o con algunos otros según sea el caso de la aplicación.

Se deja al lector la propuesta abierta a la continuación de la investigación o la puesta en práctica de las aplicaciones que se pueden desarrollar con este compilador.

Los proyectos que se pueden realizar son extensos, pues se pueden realizar con este compilador programas para realizar la comunicación RS232 entre dispositivos, comunicaciones utilizando el protocolo USB, tratamiento digital de señales, adquisición de datos y manipulaciones matemáticas de las mismas, etc.,

Las demás librerías se encuentran en el manual de usuario del compilador mikroC que se puede descargar de la página del fabricante misma que se encuentra al final de este libro.

EJERCICIOS

1. Encender y apagar en forma alternada dos LED's ubicados en el bit 2 y 3 del puerto B respectivamente. Los retardos serán de 500 milisegundos para ambos. Usando asignación directa a bits.

2. Encender y apagar un LED ubicado en el bit 5 del puerto C. Los retardos serán de 100 milisegundos y 2 segundos, respectivamente. Usando asignación de byte.

3. Enviar una secuencia de datos distinta por el puerto B, utilizando retardos por software de distintas duraciones, con incrementos de 100 milisegundos entre sí.

4. Enviar la secuencia de datos por el puerto A, utilizando retardos por software con duración de 800 milisegundos.

   100001
   010010
   001100
   010010
   100001

5. Enviar una secuencia por el puerto B usando los valores almacenado en un arreglo.

   00000011
   00000110
   00001100
   00011000
   00110000
   01100000
   11000000

6. Realizar un programa que envíe al puerto C los siguientes valores utilizando para generarlas, las instrucciones de desplazamiento y/o aritméticas.

   1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45

7. Lo mismo que el ejercicio anterior con la siguiente secuencia:

   3, 6, 12, 24, 48, 92, 172, 1, 3, 6, 9, 12, 15, 18, 21, 24

8. Desarrollar un programa que envíe una secuencia de números BCD a un display de 7 segmentos ubicados en el puerto D. Si la terminal RA1 ubicada en el puerto A, es igual a 0, la numeración debe ser incremental; en caso contrario debe decrementarse.
9. Realizar un programa que muestre los números pares en forma incremental en un intervalo de 00 a 99, desplegar en un par de display's de 7 segmentos. El programa debe realizar la visualización utilizando la técnica de multiplexado de datos, utilizando el puerto B como bus de datos y las terminales RC0 y RC1 como terminales de habilitación de display.
10. Realizar el mismo ejercicio anterior con la modificación de mostrar los números primos en forma consecutiva del intervalo 000- 999.
11. Modificar los dos ejercicios anteriores habilitando un interruptor, de forma que al existir un 0 incrementar o al existir un 1 decrementar
12. Desarrollar un programa que realice el conteo del 0 al 9 y que se visualice en un display LCD
    Tomar como base el siguiente algoritmo:
    1. Configura el LCD
    2. Inicializa el contador
    3. Convierte a ASCII el valor del contador
    4. Envía valor en ASCII al LCD
    5. Incrementa el contador
    6. Regresa a 3
13. Diseñar un programa en donde se muestre en un LCD un conteo que inicie en 0 y termine en 99. Con un intervalo de tiempo entre cambio igual a 500 milisegundos
14. Realizar el ejercicio 11 pero con la condición que se realice la cuenta en forma iterativa.

APÉNDICE A: INSTALACIÓN DE COMPILADOR MIKROC

Como primer paso se debe descargar la versión libre del compilador mikroC de la página www.**mikroe**.com, cabe mencionar que la versión libre está limitada a 2k palabras de programa con extensión *hex* (para más información referirse a la página web aquí mencionada). La versión puede ser más actual a la utilizada en este libro, el formato de descarga puede estar comprimido, por lo que se tiene que descomprimir a una de las carpetas del sistema y entonces se mostrara un icono como el que se muestra en la figura A1.

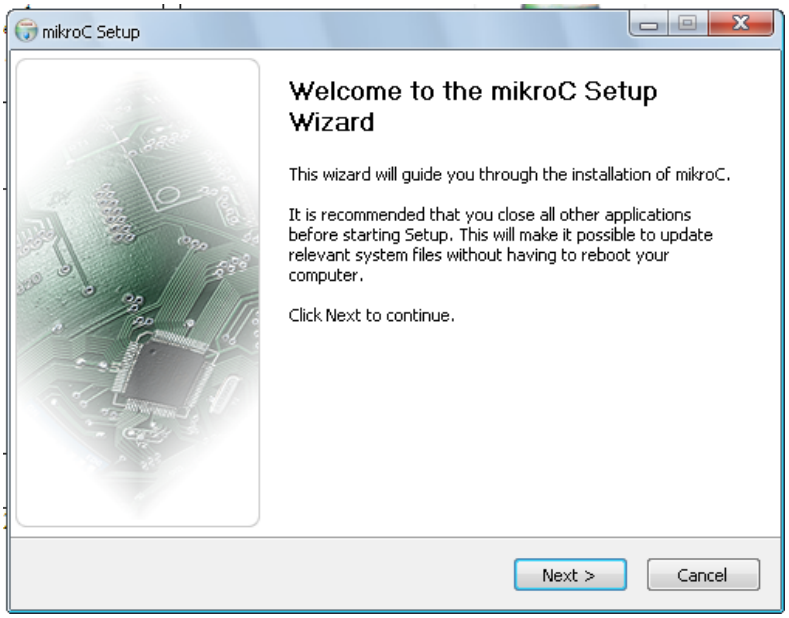
Figura A1

Icono del compilador mikroC previamente descomprimido antes de su instalación

Entonces se procede a dar doble "clic" en el icono y de esta forma se desplegara una ventana de bienvenida al programa de instalación como la que se muestra en la figura A2. En dicha ventana dar "clic" en la casilla con la leyenda next.

Figura A2

Ventana de bienvenida para la instalación del compilador mikroC



Enseguida emerge otra venta como la que se muestra en la figura A3 que muestra los acuerdos de la licencia del compilador. Si no se aceptan los términos de licencia el programa no se instalara y se saldrá del procedimiento de instalación, de lo contrario dar “clic” en el icono con la leyenda next.

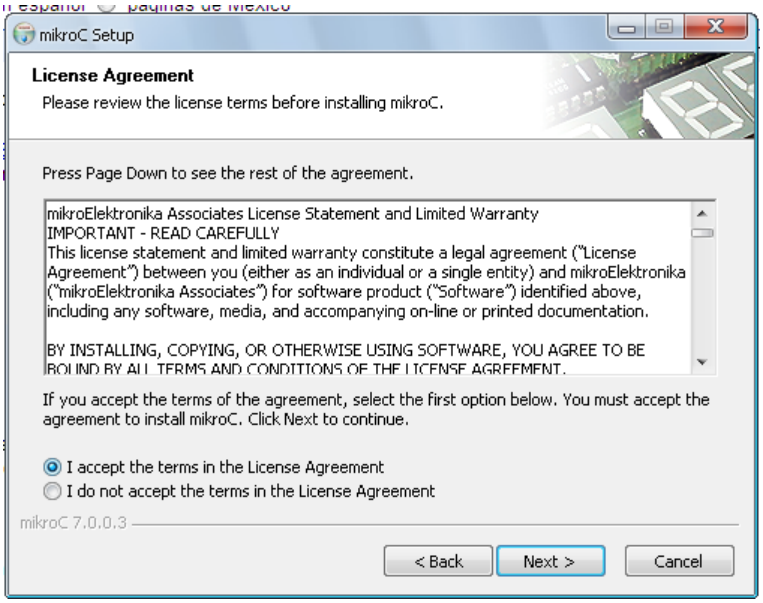


Figura A3

Ventana de términos de Licencia del compilador mikroC

Después de lo anterior aparece otra ventana (ver figura A4) con funciones y/o características adicionales al programa, dar “clic” de igual forma en next.

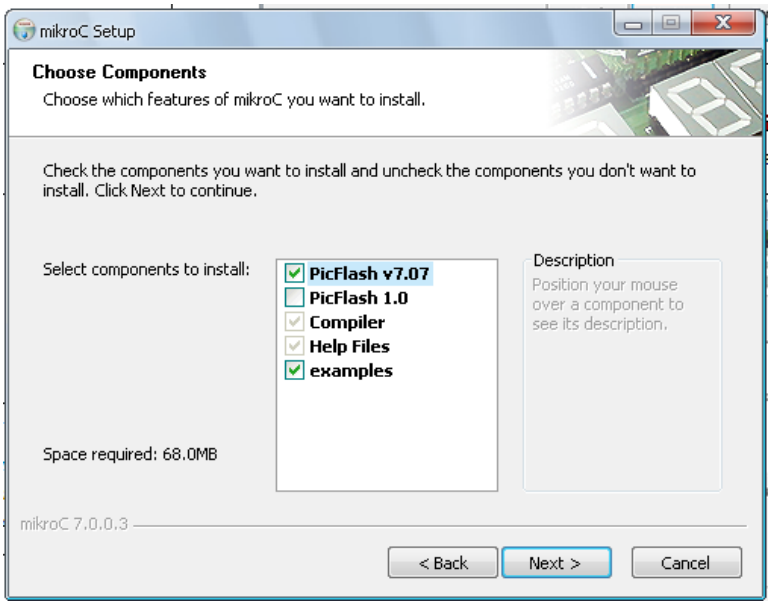


Figura A4

Ventana para instalar funciones adicionales al programa.

La ventana que se muestra en la figura A5 aparece después del paso anterior en ella se escoge la dirección donde se desea instalar el programa. Una vez establecida la ruta se procede a dar "clic" en install y así comenzara la instalación del compilador mikroC.

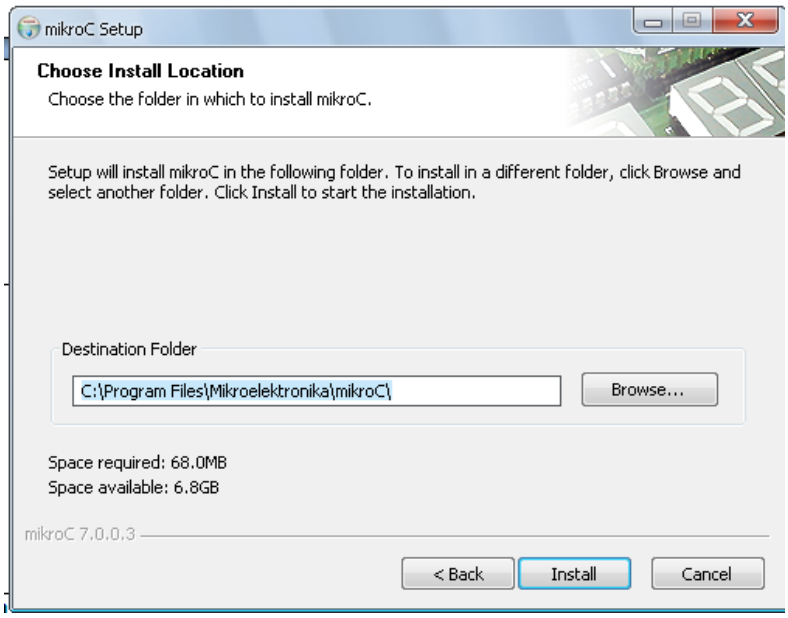


Figura A5

Ventana que permite escoger la ruta de instalación del programa.

La ventana mostrada en la figura A6 emerge después del paso anterior e indica o muestra el proceso de instalación

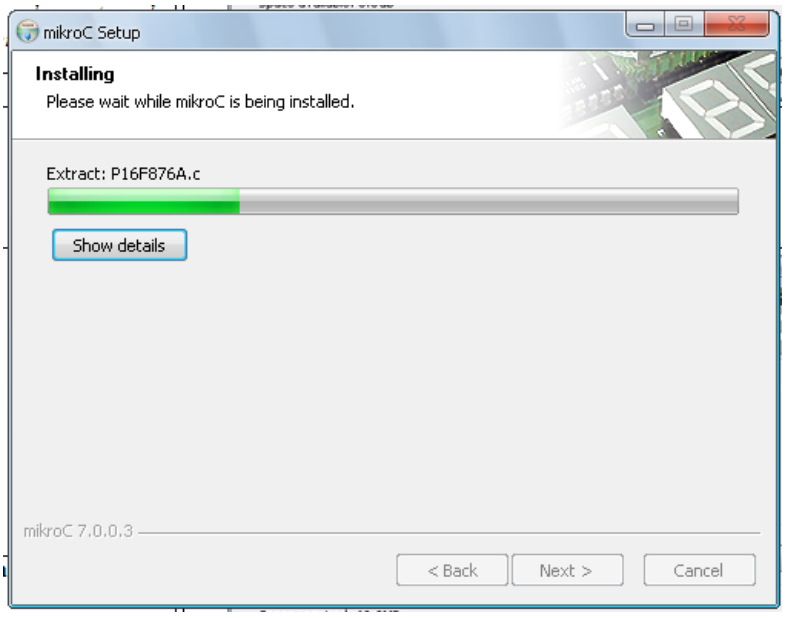


Figura A6

Ventana mostrando el porcentaje de instalación del compilador mikroC.

Una ves terminada la instalacion emerge otra ventana como la que se muestra en la figura A7 que informa que el programa a sido instalado correctamente, entonces solo resta dar “clic” en el icono con la leyenda finish .

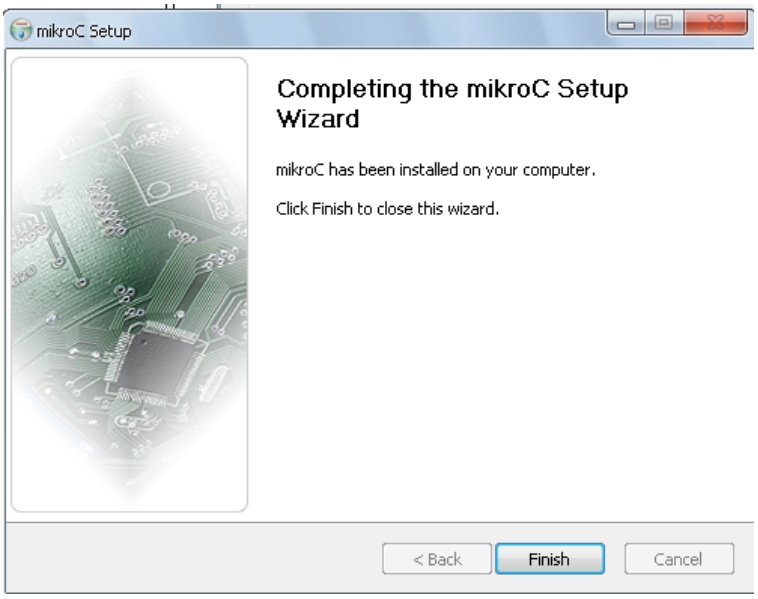


Figura A7

Ventana indicando que el programa ha sido instalado satisfactoriamente.

Para finalizar emergen dos ventanas, una después de la otra las cuales requieren que el usuario instale software adicional, estos softwar’s estas opciones solo se escogerán en el caso de que se cuente con el hardware necesario para su uso, el cual se puede adquirir en la misma página donde se descargo el compilador mikroC.

APÉNDICE B: SIMULACIÓN CON PROTEUS ISIS.

Como primer paso para iniciar el programa, este se debe ejecutar desde el menú de programas de Windows donde se tiene instalado. Hecho lo anterior se desplegara una ventana principal como la que se muestra en la figura B1. El sistema PROTEUS consta de dos módulos o programas diferenciados: ISIS que es el modulo que permite hacer la captura esquemática y las simulaciones -y que es objeto de este apendice- y ARES que es el modulo dedicado al diseño de placas de circuito impreso (pcbs).

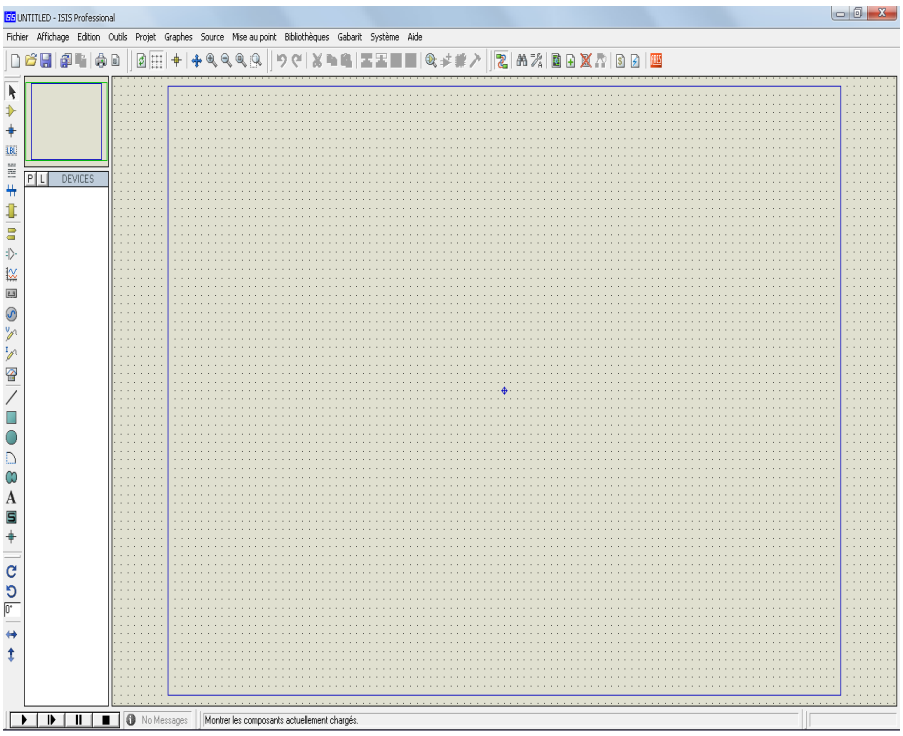


Figura B1

Ventana principal de modulo PROTEUS ISIS

Inicialmente se debe crear un proyecto vacío (File>>New) o abrir uno ya existente (File>>Open).

Si se parte de un nuevo proyecto se distinguen varias partes en la ventana de la aplicación que son: la barra de herramientas en la parte superior; debajo de los menús, una barra de estado que permitirá modificar geométricamente los componentes, y que además permitirá mediante cuatro sencillos botones arrancar la simulación pararla, ejecutar un paso, etc. Además existe otra barra de herramientas en formato vertical que va acompañada de una lista de dispositivos. Por último la hoja donde se deben colocar los distintos componentes.

En un primer diseño se situara el montaje básico de un microcontrolador: el PIC16F877A. Para ello es necesario seleccionarlo. Tecleando la letra P se logra situar cualquier componente como se ve en la figura B2.

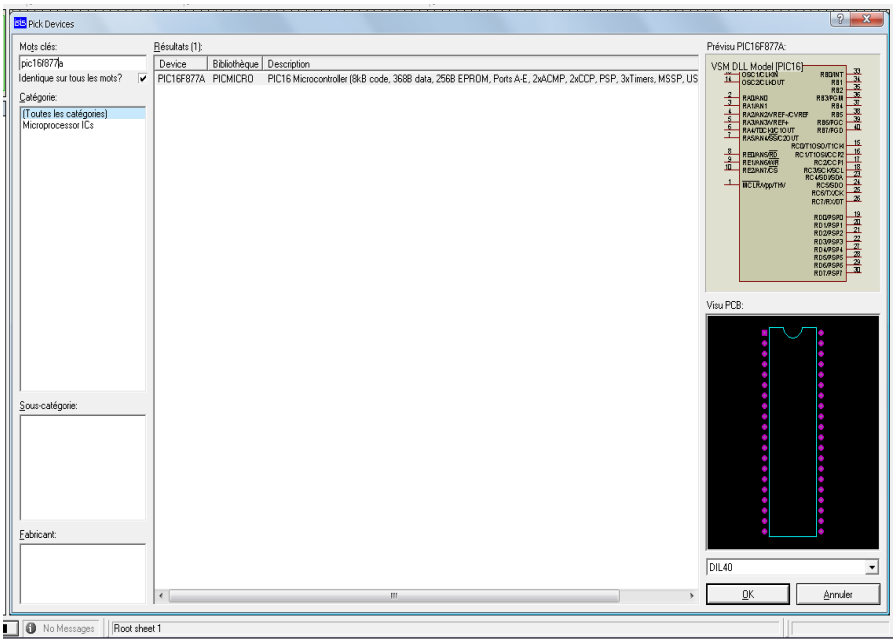


Figura B2

Ventana Pick Device. Selección de dispositivos

Simplemente se debe teclear el nombre o parte del nombre del dispositivo buscado y aparecerá una lista de posibles candidatos, y con la selección de uno de ellos el esquema grafico que lo define.

Una vez aceptado el componente solo solo se tiene que posicionar un lugar con el ratón y pulsar el botón izquierdo para situarlo en la hoja de diseño (figuara B3).

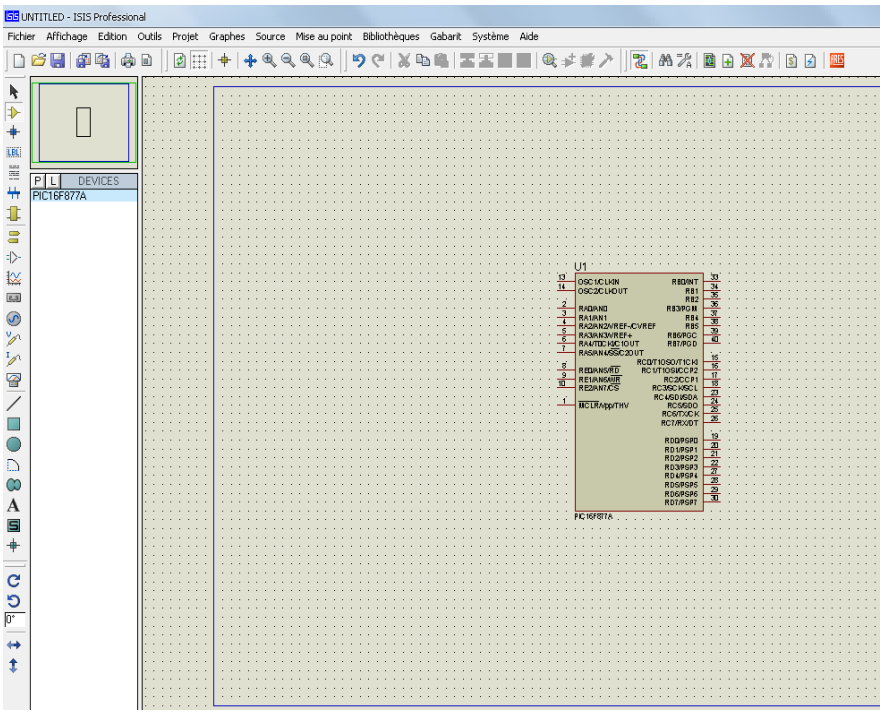


Figura B3

Dispositivo seleccionado y posicionado en el área de trabajo

Se hace lo mismo con otros componentes necesarios como el cristal de cuarzo (CRYSTAL) (ver figura B4).

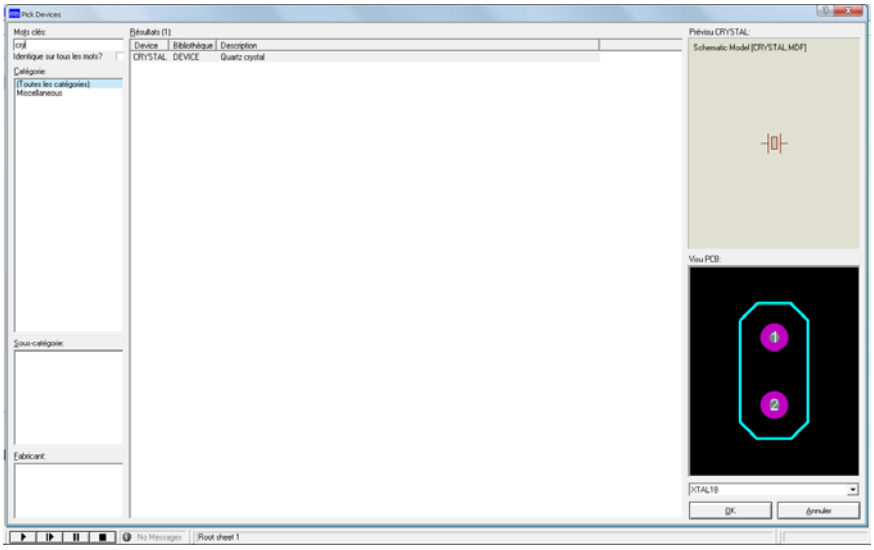


Figura B4

Crystal de cuarzo a utilizar en la simulacion

Para trazar un cable entre dos elementos simplemente se aproxima el cursor hasta la terminal correspondiente y se pulsa el botón izquierdo del ratón para trazar automáticamente el cable (Ver figura B5).

Si se desea que el cable se trace en una figura determinada, simplemente se pulsa a lo largo del camino a recorrer y termin en el segundo elemento.

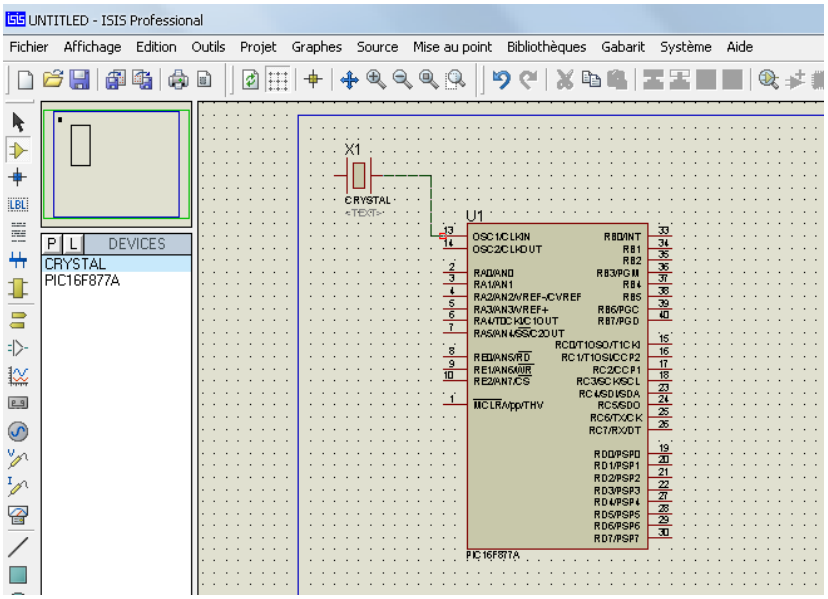


Figura B5

Conectando trazando cbles entre 2 dispositivos

Después de haber colocado y conectado otros elementos como los capacitores (CAP), se necesita colocar algunas tierras (GROUND) y alimentaciones (POWER)(ver figura B6). Estas se encuentran pulsando en la barra de herramientas vertical el icono relacionado con las terminales 

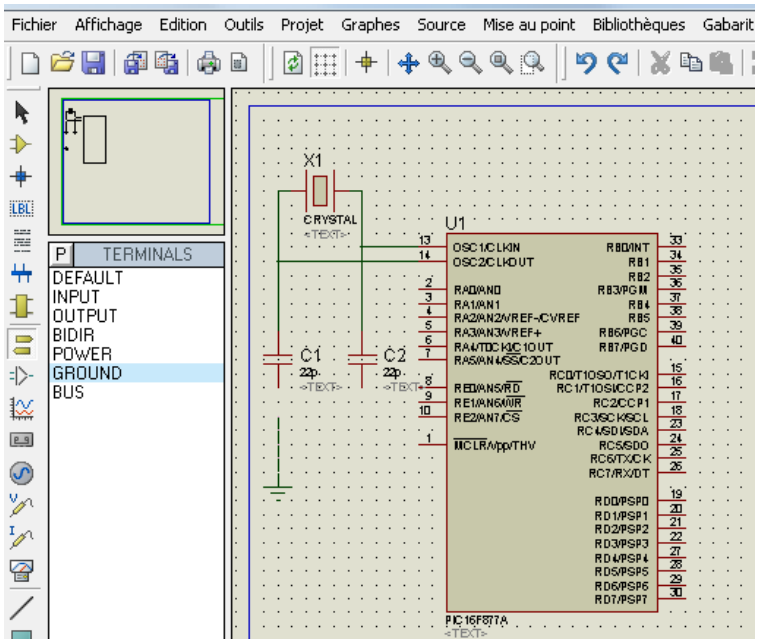


Figura B6

Adicionando dispositivos, tierra y alimentación.

La conexión de estos elementos se hace de la misma manera. Si se necesitan ver ampliada la hoja de diseño se puede hacer a través del menú, de la barra de herramientas horizontal o utilizando los atajos de teclado (teclas F5,F6,F7,F8).

Una vez colocados los componentes se podrán modificar sus valores. Para ello se debe seleccionar el componente concreto pulsando sobre él con el botón derecho del ratón, y a continuación con el botón izquierdo con dicha acción emerge una ventana com la que se muestra en la figura B7. Cuidando de no pulsar dos veces con el botón derecho del mouse para no eliminar al componente. El atajo de teclado U ayuda a recuperar (deshacer la última operación) el componente borrado por error.

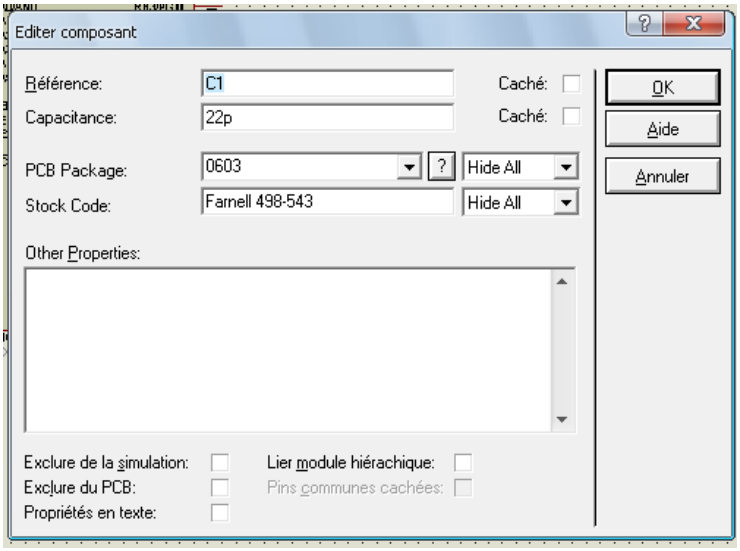


Figura B7

Ventana para editar los valores de dispositivos utilizados en una simulación.

Después de haber modificado los valores de los dispositivos a utilizar de acuerdo a las necesidades de la simulación, se procede a indicarle al microcontrolador con qué frecuencia va a ser simulado.

Seleccionando al microcontrolador se procede a editar sus propiedades en una ventana similar a la figura B8, en ella se editan los campos según las necesidades de la simulación.

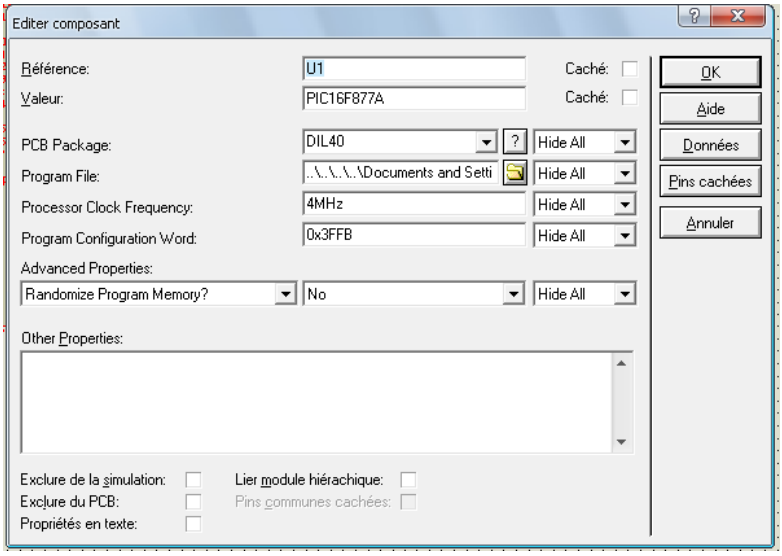


Figura B8

Ventana para modificar editar parámetros del microcontrolador, y para cargar el código con extensión .hex.

La frecuencia de reloj debe coincidir con la del cristal de cuarzo del esquema. Ahora ha llegado el momento de añadir código al microcontrolador.

En el caso particular del compilador mickoC, este genera un archivo de salida con extensión "*.hex*" que es el que basta para realizar la simulación. En la ventana de la figura B8 se observa un campo llamado "*program file*", en el cual se debe especificar la ruta donde se encuentra alojado el archivo con extensión "*.hex*" el cual debe estar alojado en la carpeta donde se realizan los proyectos del compilador en uso.

Realizado lo anterior se procede a la simulación al dar play con el icono de la parte inferior izquierda del simulador.

Cabe mencionar que PROTEUS en el modulo ISIS trae consigo una serie de instrumentos virtuales que brindan una excelente ayuda durante el proceso de desarrollo de proyectos electrónicos estos instrumentos y otras herramientas se localizan en la barra vertical izquierda del ambiente de desarrollo de PROTEUS.

GLOSARIO DE TÉRMINOS

C

CAN

Controller Area Network. Protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH, basado en una topología bus para la transmisión de mensajes en ambientes distribuidos.

CISC

Complex Instruction Set Computer. Conjunto de Instrucciones Complejas para Computadora. Los microprocesadores CISC tienen un conjunto de instrucciones que se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos situados en la memoria o en los registros internos, en contraposición a la arquitectura RISC.

CPU

Central Processing Unit. Unidad central de procesamiento. Es el procesador que contiene los circuitos lógicos que realizan las instrucciones del computador. Es un Chip que contiene millones de transistores encargados de realizar las operaciones que encomendamos al computador. No obstante, por sí solo no sirve para nada, porque debe estar conectada a la placa madre. La placa madre provee de corriente eléctrica a la CPU y le permite comunicarse con el resto de dispositivos.

D

DSP

Digital Signal Processor. Procesador Digital de Señal. Sistema basado en un procesador o microprocesador que posee un juego de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad.

E

EEPROM

Electrically Erasable Programmable Read-Only Memory. Memoria de sólo lectura, programable y borrable eléctricamente.

EPROM

Erasable Programmable Read-Only Memory. ROM borrable programmable. Una vez programada, una EPROM se puede borrar solamente mediante exposición a una fuente luz ultravioleta.

I

I^2C

Inter-Integrated Circuit. Circuito integrado de interconexión. Bus de comunicaciones serie. Se usan dos hilos para transmitir la información, por uno van los datos y por otro la señal de reloj que sirve para sincronizarlos.

N

NMOS

(**Negative-channel Metal-Oxide Semiconductor**). Transistor MOSFET de tipo incremental de canal n.

O

OTP

One-Time Programable. Memoria programable una sola vez. La única diferencia con la EPROM es la ausencia de la ventana de cuarzo, por lo que no puede ser borrada.

P

PIC

Familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650.

PMOS

Positive-channel Metal-Oxide Semiconductor. Transistor MOSFET de tipo decremental de canal p.

R

RAM

Random Acces Memory: Memoria de acceso aleatorio. Memoria donde la computadora almacena datos que le permiten al procesador acceder rápidamente al sistema operativo, las aplicaciones y los datos en uso. Tiene estrecha relación con la velocidad del computador. Se mide en megabytes.

RISC

Reduced Instruction Set Computer. Conjunto de Instrucciones Reducido para Computadora.

ROM

Read Only Memory: Memoria de sólo lectura. Memoria incorporada que contiene datos que no pueden ser modificados. Permite a la computadora arrancar. A diferencia de la RAM, los datos de la memoria ROM no se pierden al apagar el equipo.

S

SISC

Specific Instruction Set Computer. Conjunto de Instrucciones Especificas para Computadora.

U

UART.

(**Universal Asynchronous Receiver-Transmitter**). "Transmisor-Receptor Asíncrono Universal". Chip de ciertos sistemas digitales cuyo principal objetivo es convertir los datos recibidos en forma paralela, a forma serial, con el fin de comunicarse con otro sistema externo. También realiza el proceso inverso.

BIBLIOGRAFÍA

Microcontroladores PIC. Diseño práctico de aplicaciones
Angulo Usategui, José María y Angulo Martínez, Ignacio
McGraw-Hill.

Programación en C
Luis Joyanes
McGraw Hill

MikroC User's Manual
Mikroelektronika

Data Sheet Microcontroller PIC16f877a
Microchip Technology Inc.

Microcontroller Based Applied to Digital Control
Dogan Ibrahim
John Wiley and Sons, Ltd

Título: Microcontrolador PIC16f84. Desarrollo de proyectos
Autor: Enrique Palacios, Fernando Remiro, Lucas J, López
Editorial: Ra-Ma

DIRECCIONES DE INTERNET

http://picmania.garcia-cuervo.com

http://www.mikroe.com/

http://www.best-microcontroller-projects.com

http://www.alos.5u.com/

http://picprojects.blogspot.com/

http://forum.sourceboost.com/lofiversion/index.php/

http://www.pages.drexel.edu/~cy56/PIC.htm

http://www.ing.uc.edu.ve/aulavirtual/course/view.php?id=16

http://www.jvmbots.com/viewtopic.php?p=126&highlight=

http://academic1.bellevue.edu/robots/pic1.html

http://personales.ya.com/cepalacios/Proteus.htm

http://www.hobbypic.com/index.php?option=com_frontpage&Itemid=1

http://www.unicrom.com/tutoriales.asp

http://www.todopic.com.ar/

http://www.micro-examples.com/

INDICE

G

I

K

L

M

O

P

R

S

T

U

W