UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES

ARAGÓN

DISEÑO Y DESARROLLO DE UN PROGRAMADOR USB PARA MICROCONTROLADORES PIC

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

IGNACIO MENDOZA NUCAMENDI

DIRECTOR DE LA TESIS

M. en I. ARCELIA BERNAL DÍAZ

2009

Dedicatoria

Toda buena dádiva y todo don perfecto proviene
de lo alto y desciende del Padre de las luces,
en quien no hay cambio ni sombra de variación.
Santiago 1:17

A mis padres, quienes son un regalo bendito para mí, ya que con su ejemplo, apoyo y amor he podido dar un paso más.

Agradecimientos

Cuando Jehová hiciere volver la cautividad de Sion,
seremos como los que sueñan.
Salmo 126:1

Son muchas las personas a las que tengo que agradecer por apoyar de alguna forma a que este proyecto fuera terminado, pero de manera especial quiero expresar mis agradecimientos:

A Dios, quién me ha dado una nueva identidad, recursos, propósito y destino para despertar a un nuevo sueño.

A mis padres Ignacio Mendoza y Josefa Nucamendi, a quienes debo este logro y todo lo que soy.

A mis hermanos Esperanza, Jorge y Luis Enrique, quienes han compartido conmigo todos estos años.

A mis amigos y pastores Raúl de la O y Nayara Flores, quienes con cariño me han formado estos años, pronto los alcanzaré en su viaje por el mundo.

A mi asesora. M. en I. Arcelia Bernal Díaz, gracias por todas estas horas de trabajo y tus palabras de aliento, seguiremos hasta alcanzar los objetivos del proyecto LATI.

A mi casa de estudios la Universidad Nacional Autónoma de México, en especial a la FES Aragón quién a través de sus profesores me ha dado las bases para desarrollarme profesionalmente y continuar estudiando.

Índice

Capítulo 4. Diseño y desarrollo del programador USB de microcontroladores PIC

Índice de figuras

Capítulo 1. El mundo de los microcontroladores

Capítulo 2. Herramientas de Microchip para el desarrollo de aplicaciones con microcontroladores PIC

Capítulo 3. Herramientas de terceros para el desarrollo de aplicaciones con microcontroladores PIC

Capítulo 4. Diseño y desarrollo del programador USB de microcontroladores PIC

Capítulo 5. Resultados

Índice de tablas

Índice de cuadros de código fuente

Capítulo 2. Herramientas de Microchip para el desarrollo de aplicaciones con microcontroladores PIC

Capítulo 4. Diseño y desarrollo del programador USB de microcontroladores PIC

Introducción

Introducción

¿Qué tan difícil es pulsar una tecla? Si presionamos la tecla correcta en el panel un elevador podremos encontrarnos rápidamente en el último piso de un gran rascacielos. Una tecla de un teléfono celular, podría hacer una llamada a un familiar que vive en otra ciudad. ¿Y qué haría una combinación de teclas? Tal vez proporcionar dinero de un cajero automático, controlar un transbordador espacial, interactuar con un video juego o simplemente activar la alarma del reloj digital para no llegar tarde a nuestro primer día de trabajo. Todos estos elementos tienen en común el uso de sistemas electrónicos e informáticos, que interactúan entre sí, para satisfacer una necesidad o resolver un problema, con lo que se consigue hacer nuestra vida cotidiana más simple.

Son muchos los conocimientos que se requieren para diseñar soluciones a problemas aplicando opciones tecnológicas, por lo que se han creado diferentes carreras en universidades en el área de ingeniería, por ejemplo: mecatrónica, sistemas, computación, electrónica, control y otras más; cada una con un campo de especialización. Por el contrario, al diseñar sistemas que aplican estos conocimientos, se busca que sean lo más sencillos de utilizar para el usuario. Muestra de esto es que ya no tenemos que marcar el número del consultorio de nuestro dentista, si dicho número está registrado en el teléfono, bastará con decirle: “marca a dentista” y el teléfono marcará por nosotros.

Las computadoras como hoy las conocemos han sufrido grandes cambios a través de los últimos 60 años. Desde aquellos días en que las computadoras eran exclusivas para expertos en universidades hasta el día de hoy en que casi cualquier equipo electrónico por simple que parezca, posee una microcomputadora o microcontrolador.

En el caso de los microcontroladores de 8 bits, los más populares a nivel mundial son los microcontroladores PIC. Microchip Technology Inc. una empresa estadounidense que fabrica los microcontroladores PIC, brinda herramientas en hardware y software de bajo costo para diseñar, simular, programar y depurar una aplicación de principio a fin. De igual forma ofrece, de manera gratuita, una gran cantidad de información electrónica disponible en su sitio de la Internet (www.microchip.com) como son: hojas de datos de sus productos, documentación técnica, notas de aplicación, códigos fuente de ejemplo, *firmware* para aplicaciones, bibliotecas de *software* y un foro para desarrolladores.

De manera paralela, existen empresas, universidades y emprendedores en todo el mundo que desarrollan herramientas en *hardware* y *software* para crear aplicaciones con microcontroladores PIC®, algunos de ellas sin ánimo de lucro, como el programa IC-Prog, desarrollado por Bonny Gijzen[1] y el programador de microcontroladores PIC JDM desarrollado por Jens Dyekjær Madsen (1998)[2].

Uno de los elementos a tener en cuenta cuando se desarrolla un proyecto o se enseña a utilizar microcontroladores, es si se dispone de las herramientas en hardware y software que se emplean para desarrollar la aplicación, así como su costo y facilidad de uso. Para poder desarrollar proyectos con microcontroladores es necesario, de manera mínima, un equipo de cómputo, un *software* para compilar el código fuente y un dispositivo programador. El disponer de estas herramientas, permiten al estudiante que su aprendizaje sea significativo, ya que la teoría enseñada en clase es aplicada en el desarrollo de un proyecto.

En el caso del compilador, Microchip distribuye sin costo alguno, el compilador MPASM® y otras herramientas en un entorno de desarrollo integrado llamado MPLAB® IDE, lo que permite que estudiantes y emprendedores puedan desarrollar programas para microcontroladores PIC sin necesidad de invertir en una licencia de uso de software. Las herramientas en hardware de Microchip tienen un costo relativamente bajo, pero esto no implica que puedan ser adquiridas por estudiantes o aficionados al desarrollo de hardware. En su lugar, escuelas y facultades realizan su compra para ser prestadas a sus alumnos, bajo las políticas que cada una determina.

[1] http://www.ic-prog.com
[2] http://www.jdm.homepage.dk/newpic.htm

Gracias a que Microchip ofrece la información del protocolo usado para la programación serie en circuito o ICSP™ (*In-Circuit Serial Programming*) con que se programan sus microcontroladores PIC, existe gran cantidad de propuestas para dispositivos programadores que pueden armarse con un muy bajo presupuesto.

La forma de conectar un periférico a una PC ha sido una evolución constante. Hoy en día, el puerto de comunicaciones USB ha reemplazado casi por completo a los antiguos puertos serie y paralelo, con lo que las herramientas en hardware para programar microcontroladores PIC no podrían ser la excepción, pues ahora son diseñadas para ser usadas con este nuevo puerto de comunicaciones.

Así el objetivo de este trabajo de investigación es: diseñar y desarrollar un programador con interfaz USB para microcontroladores PIC de la marca Microchip con *software* de aplicación, de bajo costo y registrado bajo la licencia GPL GNU, para su uso en la enseñanza de microcontroladores en el desarrollo de soluciones tecnológicas. Lo que implica que cada estudiante o emprendedor en el mundo podrá beneficiarse de esta herramientas para aprender a usar microcontroladores de una manera más amigable ya que “contará con la herramienta en casa”.

Para lograr este proyecto será necesario cumplir lo siguientes objetivos específicos

- Diseñar e implementar en lenguaje C, mediante el paradigma de la programación estructurada, un *firmware* para un microcontrolador PIC18F2550, encargado tanto de controlar la operación de grabación de microcontroladores PIC a través del protocolo ICSP™ como de la comunicación a través puerto USB.
- Diseñar e implementar un circuito impreso que realice la implementación de las señales eléctricas utilizadas tanto por el protocolo ICSP™ para la programación de microcontroladores PIC como del protocolo USB para la interfaz física del sistema de comunicaciones.
- Diseñar e implementar un *software* de aplicación para la operación del programador USB de microcontroladores PIC.
- Diseñar e implementar un protocolo de comunicación entre el programador de microcontroladores USB y el *software* de aplicación.

- Adquirir una licencia GLP/GNU para registrar el programador USB de microcontroladores.

Durante el capítulo 1 se expondrán las características generales de los microcontroladores PIC. En los capítulos 2 y 3 se comentarán las herramientas necesarias para desarrollar aplicaciones con estos microcontroladores, tanto las herramientas comerciales provistas por Microchip y otros fabricantes como las de libre distribución que pueden encontrarse en Internet. El capítulo 4 presentará los conceptos y describirá las características los estándares utilizados en el diseño del programador USB de microcontroladores PIC. El capítulo 5 expondrá los resultados obtenidos, como son las características de los productos terminados en *hardware* y *software*, mientras que en el capítulo 6 serán presentadas las conclusiones finales obtenidas al implementar el sistema.

Capítulo 1

El mundo de los microcontroladores

Capítulo 1
El mundo de los microcontroladores

Un microcontrolador es un circuito integrado programable que en su interior posee los mismos elementos que una computadora digital, como se muestra en la figura 1.1, (unidad central de procesamiento, memoria y dispositivos de entrada-salida) y opcionalmente algunos dispositivos periféricos como convertidores analógicos-digitales, digitales-analógicos, circuitos generadores de modulación por ancho de pulsos, puertos de comunicación serie, comparadores analógicos, entre otros, que auxilian al microcontrolador a cumplir la tarea para la que ha sido programado.

Se dice que un microcontrolador tiene una ***arquitectura cerrada*** porque todos los elementos que necesita para funcionar están dentro del mismo chip y no es posible modificar ni las características ni el número de éstos. Es capaz de realizar cálculos con gran rapidez, obedeciendo a un conjunto de instrucciones muy específicas y elementales conocidas como ***programa***.

Los microcontroladores son de ***propósito específico***, ya que una vez programado y configurado, reside en el sistema a controlar y sólo sirve para la tarea que ha sido dispuesto, a diferencia de los microprocesadores que son de ***propósito general***. Los microcontroladores son usados en diversas áreas como comunicaciones digitales, medicina, instrumentación científica, aparatos electrodomésticos, sistemas industriales, agricultura y un gran número de aplicaciones más.

Una de las empresas líderes en el desarrollo de microcontroladores es Microchip Technology Inc.[1], quien ofrece un gran número microcontroladores de aplicación general (MCU) y controladores para el ***procesamiento digital de señales*** (DSP). Dentro de estos dispositivos destacan los microcontroladores PIC (*Peripheral Interphase Controller*) quienes han tenido gran aceptación debido a que Microchip ofrece abundante información sobre sus productos, además de herramientas en hardware y software de bajo costo para el desarrollo de aplicaciones.

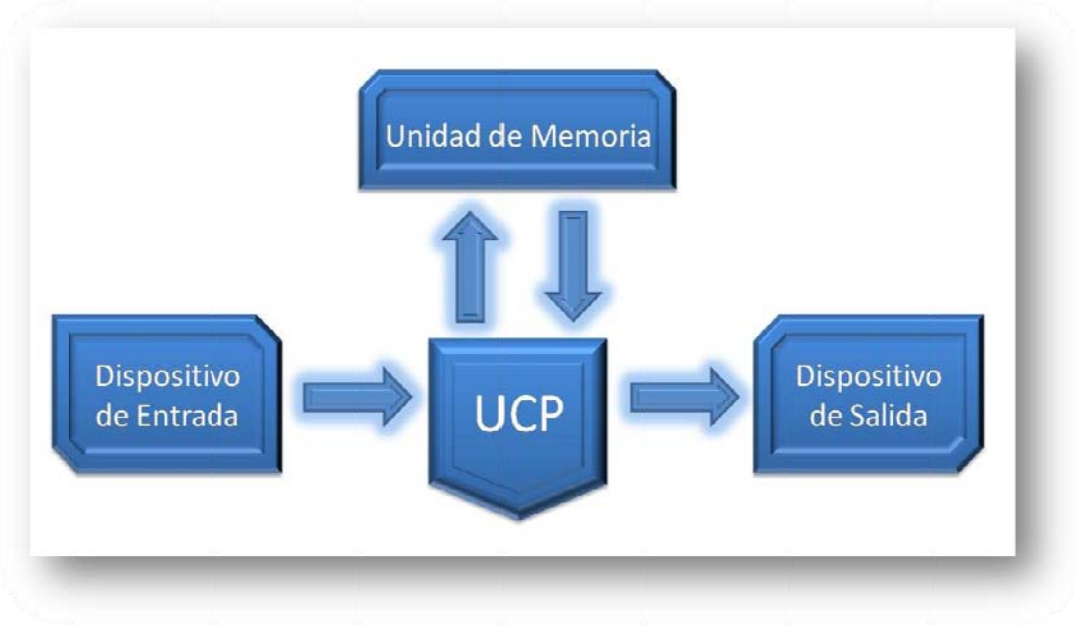


Figura 1.1. Diagrama de bloques de una computadora digital. Los microcontroladores poseen dentro de su arquitectura los elementos funcionales de una microcomputadora.

[1] La dirección del sitio en la Internet de la compañía Microchip Technology Inc. es www.microchip.com, en ella es posible encontrar información sobre todos los productos y microcontroladores que esta compañía ofrece.

1.1 Diferencia entre microprocesador y microcontrolador

Para 1970, los circuitos integrados eran sumamente populares en el desarrollo de sistemas de computación, con lo que se consiguió reducir el tamaño, la potencia de consumo y el costo de los equipos que los usaban. La tendencia continuó, se agregaban más y más arreglos de transistores formando circuitos integrados más potentes, complejos, pequeños y económicos. Durante este momento histórico, los circuitos integrados se encargaban de operar el dispositivo para el que eran diseñados, es decir, difícilmente podrían adaptarse a un dispositivo diferente o a una rutina para la que no fueron diseñados. A este tipo de elementos se les llama ***controladores embebidos*** (controladores incrustados). Esto cambió en 1970, cuando Intel® presenta el 4004, un microprocesador de 4 bits.

Hasta el momento, si se necesitaba un chip que operara los procesos de un dispositivo, éste sólo podía ser diseñado a la medida, para esa aplicación específica. Pero con la aparición del ***microprocesador***, se puede usar el mismo chip en diferentes dispositivos para controlar procesos de naturaleza diferente. Esto es gracias a que el microprocesador tiene una característica que los ***circuitos a la medida*** no, el microprocesador es programable.

Un ***microprocesador*** es un dispositivo electrónico que posee de manera mínima los siguientes elementos funcionales: Unidad de Control, Unidad Aritmético Lógica (ALU) y registros internos. Para funcionar, el microprocesador necesita de circuitos externos que le permitirán realizar el control del proceso o procesos para el que fue programado. A estos elementos se les llama ***sistema mínimo*** y se dice que el microprocesador tiene una ***arquitectura abierta*** porque dependiendo del sistema a controlar, podemos variar las características de los elementos que se conectan al microprocesador.

En este caso, un sistema mínimo para un microprocesador de aplicación general está formado por: una memoria permanente donde se almacenan las instrucciones para el control del proceso (programa); una memoria de tipo volátil[2] que es usada para almacenar información del estado del sistema o del proceso (memoria de datos); un circuito que genera un pulso continuo para sincronizar las actividades del microprocesador (reloj y oscilador principal); los circuitos integrados que permitirán que el microprocesador se conecte a otros periféricos como teclado, pantalla, impresora, alarma, sensores de temperatura, etc. (puertos de entrada y salida). Si unimos todos estos elementos a un microprocesador ya estamos hablando de una ***computadora digital***.

Con lo comentado hasta este momento podemos deducir que para controlar procesos con sistemas digitales tenemos dos opciones: recurrir al diseño a la medida de un controlador embebido o usar un microprocesador de aplicación general. Ambas técnicas de diseño poseen mayores beneficios que la otra, dependiendo del proceso que se desee controlar, el monto de la inversión, el tiempo de desarrollo y el tamaño del dispositivo donde se colocará el sistema de control[3].

Regresemos un momento. Pensemos nuevamente en los controladores embebidos. Éstos surgen como resultado de reunir gran cantidad de circuitos integrados dentro de un nuevo chip. La pregunta que surge a simple vista es ¿podría colocarse una computadora digital dentro de un chip? La respuesta es sí. Así nacen los ***microcontroladores*** que son circuitos integrados que incluyen las tres unidades funcionales de una computadora digital: la Unidad Central de Proceso (UCP), unidades de memoria y puertos de entrada-salida. Pero este microcontrolador no tiene la misma velocidad de procesamiento si la comparamos con el procesador de nuestra computadora de escritorio, ni la capacidad de manejar las grandes cantidades de memoria de datos y en algunos casos, no tiene la posibilidad de manejar más de un programa a la vez por su pequeña memoria de programa y datos.

[2] Memoria Volátil. Es una memoria que por la tecnología que usa, al suspender la alimentación eléctrica pierde la información que contiene. Generalmente es usada para almacenamiento de datos.
[3] Sistema de Control. Todos los dispositivos electrónicos involucrados en el control del proceso, como las memorias, sensores, microprocesadores, entre otros.

Una vez que el microcontrolador es programado se encargará de controlar uno o varios procesos, dependiendo del programa alojado en su memoria. Como todos los elementos que necesita para trabajar se encuentran dentro del mismo circuito integrado y estos no pueden ser modificados su número o características se dice que tiene una ***arquitectura cerrada*** y una vez programado es de ***propósito específico***. Existen algunos microcontroladores de ***arquitectura abierta***, es decir, que pueden sacar sus líneas de control o de programa a través de sus terminales para extender su capacidad de memoria de programa o datos y poder comunicarse con otros periféricos, pero esta configuración se parece más a una opción de diseño con microprocesador.

Gracias a estas características, podemos encontrar el mismo microcontrolador manteniendo estable la temperatura de una pecera o en un sistema de seguridad, claro, cada uno con su respectiva programación. El uso de microcontroladores permitiré reducir los tiempos y los costos de producción, de investigación y desarrollo del equipo en el que se colocará.

1.2 Campo de aplicación de los microcontroladores

Ahora podemos encontrar microcontroladores en casi todos los dispositivos electrodomésticos de un hogar, como el horno de microondas, lavadoras, televisores y reproductores de música; controlando sistemas de acceso en edificios inteligentes, elevadores, sistemas contra incendio, sistemas de suministro de agua, control de temperatura e iluminación; equipos electrónicos como en el teclado de una PC, controles remoto, relojes digitales, alarmas; sistemas de navegación satelital, control de frenos de un automóvil y dentro de juguetes que ahora parecen más inteligentes e interactúan con el usuario (figura 1.2).

Debido a que los microcontroladores tienen una arquitectura cerrada, los fabricantes ofrecen una gran cantidad de modelos con diferentes características, como el tamaño y tipo de memoria de programa o los periféricos con que cuenta; así puede escogerse el modelo que más se adecue a las necesidades del problema que queremos resolver usando microcontroladores.

En general, las áreas donde se están proponiendo soluciones tecnológicas usando microcontroladores son:

1. En accesorios y periféricos de computadoras.
2. Aparatos electrodomésticos.
3. Instrumentación científica y equipo médico.
4. Industria automotriz.
5. Robótica y control industrial.
6. Sistemas de navegación.
7. Sistemas de seguridad y alarma.
8. Comunicaciones digitales.
9. Investigación y docencia.
10. La agricultura y ganadería.
11. En la industria militar.

Figura 1.2. Ejemplos de aplicaciones donde se usan microcontroladores. Podemos encontrar microcontroladores en cámaras digitales, teléfonos celulares, reproductores de música, entre otros.

1.3 Principales fabricantes de microcontroladores

Desde que apareció el primer microprocesador en los años 70, son muchas las empresas que han trabajado en el desarrollo de nuevos circuitos integrados, dedicando parte de su investigación al desarrollo de nuevos microcontroladores. Se han aplicado diferentes tecnologías en el diseño de sus procesadores como: arquitectura Von Neumann[4] o Harvard[5]; procesadores tipo RISC[6], CISC[7] y SISC[8]; procesadores segmentados[9] y de tipo ortogonales[10]; y otras características que serán explicadas a lo largo de este capítulo. Entre los principales fabricantes de microcontroladores a nivel mundial se encuentran:

Freescale Semiconductor® (Antes Motorola®)[11]

Ofrece microcontroladores de arquitectura Harvard de 8, 16 y 32 bits. Sus chips son usados por marcas conocidas a nivel mundial como los teléfonos celulares Motorola, en automóviles como VMW, electrodomésticos GL, en equipo de cómputo como iPods y periféricos de Apple entre otras marcas.

NEC®[12]

Ofrece microcontroladores con arquitectura Von Neumann CISC de 8 y 16 bits y Harvard RISC de 32 bits.

ST Microelectronics®[13]

Ofrece también una gama muy amplia de microcontroladores de 8, 16 y 32 bits; cada una con diferentes capacidades de reloj; tamaño y tipo de memoria; así como diversos dispositivos periféricos. Su uso es extendido en la industria automotriz, las comunicaciones, las computadoras y periféricos, el control industrial y la seguridad.

[4] Técnica empleada en el diseño de microprocesadores donde existe un bus único para conectar la memoria de datos y la memoria de instrucciones con la UCP.
[5] Técnica empleada en el diseño de microprocesadores donde existen buses independientes para conectar la memoria de datos y la memoria de instrucciones con la UCP.
[6] RISC *Reduced Instruction Set Computer*. Computadora con Conjunto de Instrucciones Reducido.
[7] CISC *Complex Instruction Set Computer*. Computadora con Conjunto de Instrucciones Complejo.
[8] SISC *Specific Instruction Set Computer.* Computadora con Conjunto de Instrucciones Específico.
[9] Técnica empleada en el diseño de sistemas digitales para incrementar su rendimiento.
[10] Técnica empleada en el diseño de microprocesadores donde se implementa una arquitectura basada en bancos de registros.
[11] Freescale Semiconductor http://www.freescale.com
[12] NEC Semiconductor http://www.nec.com
[13] ST Microelectronics http://www.st.com

Microchip®[14]

Los microcontroladores PIC tienen una arquitectura Harvard RISC de 8 bits y ofrece una gran cantidad de opciones que divide en familias como enana, básica, media y alta, dependiendo de las características de cada microcontrolador; además de ofrecer gran cantidad de información para desarrollar aplicaciones con sus microcontroladores y de herramientas de desarrollo en hardware y software de bajo costo.

Algunas otras marcas son: Atmel[15], con sus microcontroladores AVR; Zilog®[16], que en su historia desarrolló el Z80® y controladores basados en esta UCP; Toshiba®[17], Philips®[18], Mitsubishi®[19] e Infineon, entre muchas otras (figura 1.3).

Figura 1.3. Existen muchas empresas que dedican parte de su investigación al diseño y desarrollo de microcontroladores que implementan diversas arquitecturas y tecnologías.

[14] Microchip Inc http://microchip.com
[15] Atmel Corporation http://www.atmel.com
[16] Zilog http://www.zilog.com
[17] Toshiba Semiconductor Company http://www.semicon.toshiba.co.jp/eng
[18] Philips NXP http://www.standardics.nxp.com/
[19] Mitsubishi http://www.mitsubishichips.com/Global/index.html

1.4 Recursos comunes a todos los microcontroladores

Los microcontroladores aparecen para la década de los 80s' gracias a la tendencia de ingresar cada vez más elementos en un solo circuito integrado (miniaturización) con lo que se logra colocar una microcomputadora dentro de un solo chip. Los elementos que componen a esta microcomputadora varían de un fabricante a otro, debido a que cada fabricante agrega elementos diferentes, dependiendo de las necesidades. La mayoría de los microcontroladores poseen elementos mínimos que encontraremos en cada chip como su procesador, arquitectura, memoria, oscilador o puertos de entrada y salida, que aunque son de diferentes características entre microcontroladores, los podemos encontrar en todas las marcas.

1.4.1 Unidad central de proceso (UCP)

La Unidad Central de Proceso o CPU por sus siglas en inglés es dónde se realizará el procesamiento de la información y la ejecución del programa. Una UCP está formada por diferentes elementos: la Unidad Aritmético-Lógica, registros de propósito específico y unidad de control, entre otros elementos que varían de un fabricante a otro.

1.4.2 Arquitectura básica

La ***arquitectura*** se refiere a las características de los elementos internos de una computadora y a su conexión de sus buses para que viaje la información de unos a otros. En un principio la arquitectura que se implementó fue la propuesta por el matemático John Von Neumann. Posteriormente se implementó una segunda arquitectura desarrollada en la Universidad de Harvard que es donde toma su nombre, Arquitectura Harvard.

Arquitectura Von Neumann

John Von Neumann (28 de diciembre de 1903 - 8 de febrero de 1957) propuso el modelo que se aplica a la mayoría de la computadoras digitales y en honor a él lleva su nombre. Para que una computadora pueda realizar una tarea debe seguir una serie de instrucciones a las que conocemos como programa. El programa se guarda en una memoria no volátil de sólo lectura, es decir, que no pierde la información aunque se apague la computadora. De manera general, la arquitectura Von Neumann posee sólo una memoria principal de tipo volátil donde se colocan programas y datos. Como un microcontrolador tiene sólo un programa activo, no es necesario copiarlo a la memoria principal o RAM (Memoria de acceso aleatorio) por lo que se lee directamente de una memoria de tipo ROM (Memoria de sólo lectura). La Unidad Central de Proceso (UCP) accede a las instrucciones de manera secuencial a través de un conjunto de alambres a los que conocemos como líneas de datos o ***bus de datos*** (figura 1.4).

Los resultados que se obtienen de ejecutar las instrucciones (operaciones aritméticas, lógicas, de corrimiento, entre otras) pueden almacenarse en una memoria de tipo temporal (volátil) para un uso futuro. El camino que recorren las instrucciones de la memoria de programa a la UCP y los resultados de la UCP a la memoria de datos es el mismo, es decir comparten el bus de datos. Esto puede apreciarse en la figura 1.4.

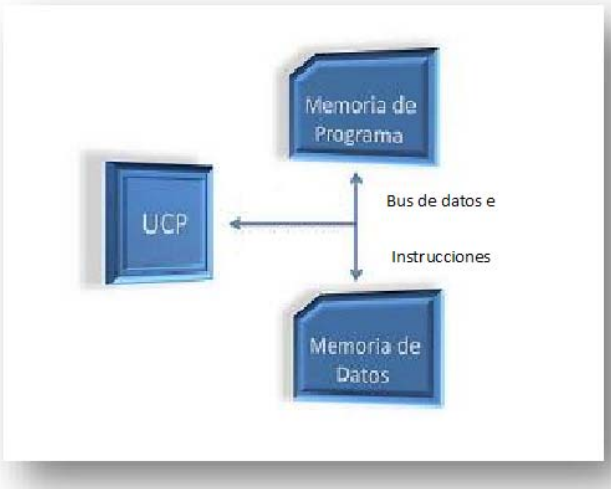


Figura 1.4. Diagrama de la arquitectura Von Neumann para un microcontrolador.

Por las mismas líneas (*bus)* estará presente una instrucción o un dato, dependiendo de la memoria se encuentre activa, es decir tienen que ser usadas de forma independiente en diferentes tiempos.

Sus principales características son:

- La longitud de la palabra para representar las instrucciones y los datos, así como la longitud del bus por el que viajan es del mismo tamaño.
- No puede accederse a una instrucción y a un dato al mismo tiempo ya que comparten la misma línea o bus. Esto trae en consecuencia que la ejecución de un programa se realice en mayor tiempo.

En el momento histórico en que fue presentado se adoptó sobre otras arquitecturas al grado de convertirse en la arquitectura tradicional[20] ya que era más fácil de construir comparada con una arquitectura Harvard por el avance tecnológico que se había alcanzado hasta ese momento.

Arquitectura Harvard

Para optimizar el trabajo de un microprocesador podemos recurrir a diversas soluciones, una de ellas es buscar que sea más rápido, logrando así que realizase mayor cantidad de operaciones por segundo. Muchas de esas operaciones son de lectura y escritura a la memoria de programa y a la memoria de datos. Así que por muy rápido que sea el microprocesador, si la memoria es lenta, la UCP tendrá que esperar a que el dato o la instrucción estén listos para usarse, lo que implica desperdiciar ciclos de reloj. Una solución es hacer memorias más rápidas pero el costo del sistema aumenta. Otra solución considera usar más de una UCP en diferentes arreglos para aumentar el rendimiento, pero esta solución sigue influyendo en el precio final del sistema por lo que el uso de alguna de estas opciones tiene que justificarse. Una tercera alternativa está en modificar como viajan las instrucciones y los datos de las memorias a la UCP. Si podemos hacer que viajen por caminos separados, podemos acceder a la memoria de instrucciones y a la memoria de datos al mismo tiempo con el diseño apropiado. Esta propuesta se conoce como ***arquitectura Harvard*** (figura 1.5).

[20] Aun se siguen construyendo computadoras con esta arquitectura para guardar la compatibilidad entre programas escritos para diferentes computadoras.

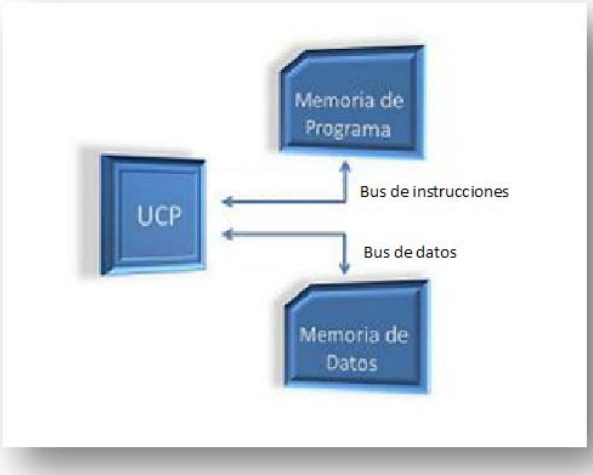


Figura 1.5. Diagrama de la Arquitectura Harvard para un microcontrolador

La ***arquitectura Harvard*** fue desarrollada en la Universidad de Harvard en 1944 por Howard Aiken. Fue implementada en la IBM *Automatic Sequence Controlled* Calculador (ASCC) llamada Mark I, una computadora electromecánica. Esta computadora era programada a través de un panel de interruptores y los datos eran introducidos en un módulo lector de tarjetas perforadas.

Tener buses independientes para la memoria de datos y la memoria de instrucciones tiene ventajas sobre la arquitectura Von Neumann:

- El acceso a cada memoria puede ser simultáneo y a través de líneas independientes por lo que se reduce el tiempo que la UCP emplea en acceder a la memoria.
- El tamaño de los buses de datos e instrucciones puede ser de diferente tamaño. Lo que implica que puede ser optimizado el conjunto de códigos de operación con que se representan las instrucciones para que ocupen una localidad de memoria y su ejecución sea más rápida.

Una versión diferente de esta arquitectura es llamada ***arquitectura Harvard modificada*** y consiste en colocar una pequeña cantidad de memoria muy rápida entre la memoria de datos y la UCP, donde se colocarán los datos que son más frecuentemente utilizados, para disminuir el tiempo que la UCP está esperando por un ciclo de lectura o escritura a la memoria. Esta pequeña memoria se llama ***CACHE*** y lo mismo se hace entre la memoria de instrucciones y la UCP.

1.4.3 Oscilador principal

Los circuitos digitales emplean una señal que permite la sincronización de los procesos dentro del sistema. Esta señal es conocida comúnmente como señal reloj. La ***señal de reloj*** es una señal uniforme como una señal de *seno* o tren de impulsos.

Dentro del microcontrolador existe un circuito llamado ***oscilador principal***, encargado de generar esta señal y distribuirla a través de los elementos del microcontrolador que la requieran. Por lo general, para generar la señal de reloj, el oscilador principal necesita de elementos externos como cristales de cuarzo o un circuito RC para filtrar la señal y fijar la velocidad de trabajo.

Un elemento a tener en cuenta cuando elegimos el tipo o la velocidad de nuestro reloj principal es que mientras más rápido sea el reloj, es decir, mayor sea su frecuencia, más rápido se realizará el procesamiento de datos y aumentará también el consumo de corriente y el calor disipado por el circuito.

1.4.4 Memoria

La memoria del microcontrolador consiste en un conjunto de circuitos integrados que sirven para poner las instrucciones o los datos con los que trabajará. Según su uso podemos dividirla en ***Memoria de Programa*** y ***Memoria de Datos***. El programa debe estar presente cada vez que se use el microcontrolador por lo que debe colocarse en una memoria que no se borre cuando pierda el suministro de energía, es decir necesita una memoria no volátil. Existen varias tecnologías que permiten crear memorias de este tipo, algunas de ellas son:

ROM con Máscara[21]

Este tipo de memoria sólo puede ser programada por el fabricante en el momento en que se elabora el chip. Se usa cuando es necesario un gran volumen de microcontroladores con el mismo programa, por ejemplo en un modelo de hornos de microondas del que se espera fabricar millones.

OTP[22]

Este tipo de memoria es programable eléctricamente sólo una vez, después no puede ser modifica y puede ser usada para volúmenes más pequeños de los que se realizarían con una ROM con máscara, por ejemplo para prototipos finales.

EPROM[23]

Este tipo de memoria puede ser programado eléctricamente en un dispositivo hardware al que se le llama programador y reprogramado una vez que es borrado. Para borrar la memoria, su encapsulado cuenta con una pequeña ventana por donde se aplicará luz ultravioleta que provocará que la memoria se borre (figura 1.6).

EEPROM[24]

Un avance significativo respecto a las memorias EPROM se encuentre en la forma en que ésta puede ser borrada. Ya no es necesario aplicarle luz ultravioleta, ahora puede ser borrada eléctricamente en el mismo programador. Este tipo de memorias es útil en sistemas donde se está en la etapa de diseño y prueba.

FLASH

Una mejora con respecto a la memoria EEPROM es la memoria FLASH, que reduce los tiempos que requiere para grabar y reduce la energía que necesita. Además de aumentar los ciclos de lectura y escritura, es decir, puede ser borrada y grabada mayor número de veces que la memoria EEPROM.

[21] ROM *Read-Only Memory*. Memoria de sólo lectura.
[22] OTP *On-Time Programmable.* Memoria de sólo una grabación.
[23] EPROM *Erase Programmable Read-Only Memory* Memoria de sólo lectura borrable.
[24] EEPROM *Electrically-Erasable Programmable Read-Only Memory*. Memoria de sólo lectura borrable eléctricamente.

Figura 1.6**.** Memoria tipo EPROM. Para borrar este tipo de memorias poseen una ventana donde al aplicar luz ultravioleta se borra su contenido.

En el caso de la memoria de datos, en comparación con la memoria de programa es mucho más pequeña. Una parte puede ser de tipo volátil (RAM) e incluir una de tipo permanente para guardar datos (EEPROM) que no se pierdan a pesar de que se suspenda la energía eléctrica al microcontrolador.

1.4.5 Puertos de entrada y salida

Todo sistema de control tiene que recibir información del medio que controla y tiene que ejecutar acciones sobre ***actuadores***[25]. Por ejemplo, un horno de microondas puede estar programado para preparar palomitas de maíz. Al seleccionar la opción correcta, el horno calculará el tiempo y la intensidad a la que trabajará. En general, para que el microcontrolador del horno pueda trabajar debe conectarse con otros dispositivos a los que llamaremos ***periféricos***. Estos periféricos se conectan físicamente a terminales dispuestas en el chip para permitir la entrada y salida de datos.

[25] Un actuador es un dispositivo que transforma una señal de entrada en una acción o movimiento, como un motor, una electroválvula o una lámpara.

1.5 Recursos especiales en los microcontroladores

Podemos usar los puertos de entrada y salida para conectar dispositivos periféricos de tipo analógico o digital que ayuden al microcontrolador a realizar su tarea, como convertidores analógico-digitales, módulos de comunicación serie, temporizadores, filtros activos, entre otros. Pero si la integración de circuitos integrados ha permitido construir una microcomputadora dentro de un chip, también permite integrar estos recursos en nuestro microcontrolador.

Temporizadores (Timers)

Los ***temporizadores*** son circuitos integrados que pueden ser usados como contadores finitos ascendentes o descendentes, de manera sincronía o asíncrona. Cuando son usados de manera síncrona es para controlar eventos en intervalos de tiempo regulares gobernados por una ***señal de reloj***[26], como es el caso de liberar la presión de una caldera cada cierto periodo. Cuando se usan de manera asíncrona no es para medir intervalos de tiempo sino sólo "contar" el número de veces que ocurre un evento como la entrada de una persona a un almacén que no ocurre a en periodos regulares y por lo que no es necesario el uso de una señal de reloj.

Puertos de entrada y salida digitales

Para la comunicación con los periféricos puede hacerse uso de terminales dispuesta en el chip para la entrada y salida de datos, éstas se agrupan en puertos. Un microcontrolador puede tener uno o más puertos para conectar los periféricos o actuadores.

Modulador de ancho de pulsos (PWM)

Un modulador de ancho de pulsos es capaz de generar una onda periódica y variar el tiempo que dura la onda positiva. Una de las aplicaciones más usadas de estos elementos es el control de velocidad de motores. Esto se hace variando el ciclo de trabajo de la señal que se está modulando. También se utiliza en comunicaciones digitales, control de fuentes conmutadas y algunas otras aplicaciones.

[26] Señal de reloj. Es una señal periódica generada por un circuito integrado. Es usada para sincronizar los procesos de un sistema digital.

Generadores de secuencias pseudoaleatorias

En algunos casos necesitaremos números aleatorios para controlar un proceso, por ejemplo, en una máquina de azar (dado electrónico) deberá generarse una secuencia diferente cada vez que se utilice, lo que hará parecer que el número fue generado al azar. Lo cierto es que las secuencias están ya preestablecidas y sólo es necesario alimentarlas con un número de inicio que dará un resultado diferente en cada ocasión simulando una secuencia aleatoria. Estas secuencias pueden ser generadas por software, pero algunos controladores incorporar en su arquitectura generadores de secuencias pseudoaleatorias.

Generador de CRC

Los Generadores de Códigos de Redundancia Cíclica generan códigos polinómicos que se usan en la verificación de la transmisión de bits. Podemos considerar una palabra de longitud *n* como los coeficientes de un polinomio de orden n-1. Por ejemplo, los datos 11110 pueden tratarse como el polinomio $x^4 + x^3 + x^2 + x^1$. A estos bits de datos se le añaden *m* bits de redundancia de forma que el polinomio resultante sea divisible por un polinomio generador G(x) de grado *m*. El receptor verificará si el polinomio recibido es divisible por G(x). Si no lo es, habrá un error en la transmisión. Este método es usado por ejemplo por el estándar de comunicaciones IEEE 802[27].

Puertos de comunicación serie

Los puertos de comunicación nacen de la necesidad de que el microcontrolador comparta información con otro controlador. Para ahorrar en la cantidad de hilos que involucra la comunicación puede realizarse de forma serial. Existen diferentes protocolos[28] bien definidos y algunos modelos de microcontroladores los implementan para facilitar la comunicación. Ejemplo de algunos protocolos de comunicación son UART[29], I^2C[30]; USB[31] y ETHERNET[32].

[27] Conjunto de estándares propuesto por la IEEE para la comunicación en redes de área local y metropolitana de computadoras.

[28] Conjunto de reglas o normas establecidas por una comisión.
[29] UART *Universal Asynchronous Receiver-Transmitter* Transmisor-receptor universal asíncrono.
[30] I^2C *Inter-Integrated Circuit*. Bus de comunicación serie.
[31] USB *Universal Serie Bus*. Bus serie universal.
[32] ETHERNET. Se refiere a las redes de área local y dispositivos bajo el estándar IEEE 802.3

Comparador analógico

Un comparador analógico permite comparar entre una señal fija y otra variable para indicar si ésta es mayor o menor con un 1 o un 0 haciendo uso de un amplificador operacional.

Perro guardián (*Watchdog*)

El perro guardián es un temporizador síncrono que tiene una función específica. Cuando este temporizador se desbordad, es decir completa un ciclo, "ladra" o genera un aviso. Este aviso puede ser usado para diferentes funciones dependiendo del microcontrolador pero por lo general se usa para reiniciar el programa porque el microcontrolador se ha quedado mucho tiempo en una operación o ciclo, tal vez por un error. Para evitar que se reinicie el programa a causa del perro guardián, el temporizador debe ser reiniciado por una instrucción en el programa cada cierto tiempo.

Estado de bajo consumo (SLEEP)

Muchos sistemas no requieren de realizar gran cantidad de cálculos por lo que la mayoría del tiempo se invierte en esperar a que ocurra un evento para procesarlo. Si consideramos que el sistema funciona con una batería, lo que más nos preocupa es que el sistema consuma la menor cantidad de batería que sea posible. El estado de bajo consumo es usado para "dormir" al microcontrolador. En este estado se apagarán todos los recursos que no son necesarios, se suspenderá la ejecución del programa pero las líneas de entrada y salida conservarán su estado.

Convertidores Analógico Digital CAD (ADC *Analogue to Digital Converter*)

Para que un microcontrolador pueda interactuar con el sistema al que opera necesitará de información. Esta información es recibida por sensores pero la forma en que entregan esa información puede no ser entendida por el microcontrolador digital, ya que esta señal puede ser de tipo analógico. Es necesario convertir esa información en un formato que el microcontrolador entienda, es decir, en una señal digital. Las señales analógicas, como las de audio, son convertidas a un formato digital para que sean procesadas y convertidas nuevamente a un formato analógico para que puedan ser reproducidas.

Convertidores digital analógico CDA (DAC *Digital to Analogue Converter*)

En el caso de los CDA se usan para manipular actuadores de tipo analógico. Previamente debe haber sido usado un convertidor analógico digital para convertir una señal a un formato digital. Esta señal es procesada por el microcontrolador y puede ser regresada de nueva cuenta en un formato analógico. Esto lo podemos ver en los controladores encargados de procesar señales de audio y video, que capturan una señal analógica, para convertirla a un formato digital, ser luego procesada, almacenada y posteriormente reproducida de manera analógica.

Etapas analógicas de amplificación

En ocasiones las señales que resultan de la conversión digital analógica son muy pequeñas para ser usadas por un actuador como una bocina. Es necesario amplificar esa señal y para ello se usan unos circuitos llamados amplificadores, que en algunos controladores vienen integrados al chip y sólo es necesario el uso de algunos dispositivos eléctricos como capacitores y resistencias para hacerlos funcionar correctamente.

1.6. Los microcontroladores PIC de Microchip

Los microcontroladores PIC tienen su origen en los años 70. Fueron diseñados por un empresa llamada GI *Microelectronics Division* (que más tarde se convertiría en *Microchip Technology Inc.*) como parte de un proyecto para manejar aplicaciones para el microprocesador CP1600. Este microprocesador de 16 bits no manejaba eficazmente las entradas y salidas, así que se diseño un circuito integrado, con una cantidad pequeña de instrucciones, que se dedicará a auxiliar al microprocesador en el control de E/S: el PIC (*Peripheral Interface Controller*).

1.6.1 Los microcontroladores PIC están agrupados en familias

Los microcontroladores PIC son controladores de 8 bits, es decir, la longitud de la palabra que es usada para representar un dato es de 8 bits. Su UCP[33] está basada en una arquitectura Harvard RISC modificada. La capacidad de la memoria de programa va desde 384 Bytes a 128 KBytes.

El formato que guarda cada conjunto de instrucciones es el mismo por lo que es relativamente fácil migrar el código fuente[34] entre microcontroladores teniendo en cuenta algunas restricciones. Una de las características más interesantes es la implementación de un procesador segmentado (*pipeline*), lo que permite la ejecución de las instrucciones en un menor tiempo.

Los encapsulados varían en diferentes diseños como DIP[35] (*Dual In line Package*), SOIC[36] (*Small Outline Integrated Circuit*) y PLCC[37] (*Plastic Leaded Chip Carrier*) formados desde 6 hasta 80 terminales. Lo que permite elegir el encapsulado más adecuado a la aplicación que estemos diseñando para construir aplicaciones más pequeñas, de ser necesario.

[33] UCP Unidad Central de Proceso (*CPU Central Process Unit*)
[34] Hace referencia al conjunto de instrucciones escritas en un lenguaje de programación como ensamblador, C o BASIC.
[35] Encapsulado de doble línea
[36] Circuitos Integrados de Líneas de Salidas Pequeñas
[37] Circuito Integrado Portador de Plástico con Plomo

Por último, cada microcontrolador posee diferentes periféricos dentro del mismo chip, entre los que se encuentran módulos de captura y comparación, circuitos generadores de modulación por ancho de pulsos, contadores-temporizadores, temporizadores de perro guardián, convertidores analógicos digitales de 10 y 12 bits, circuitos de protección contra fallo en la alimentación, circuitos para referencia de voltaje, un estado de bajo consumo y circuitos para la comunicación serie, entre otros controladores que soporten los protocolos de comunicación más comunes como RS232/RS485[38], SPI®[39], I²C®[40], CAN[41], USB[42], LIN[43], Radio RF[44], TCP/IP[45].

Una forma de Microchip para organizar sus microcontroladores PIC de 8 bits es en familias. Esta división se realiza dependiendo de las características que cada microcontrolador incorpora, como la tecnología y el tamaño de la palabra de la memoria de programa, el número de instrucciones para programarlo, el número de terminales de su encapsulado y los dispositivos periféricos que agrega a cada chip. Al pasar el tiempo Microchip ha desarrollado nuevos circuitos, mientras que otros ya han salido del mercado, por lo que hablar de una división rigurosa de que microcontrolador pertenece a que familia es un poco difícil. Al momento de escribir este trabajo Microchip habla de 4 familias, la familia 10F, 12F, 16F y 18F.

De manera general los microcontroladores de estas familias se dividen en 4 gamas conocidas como gama enana, baja, media, alta y mejorada.

[38] Protocolo para comunicación serie. En la computadora hace referencia al puerto COM.
[39] Serial Peripheral Interface. Interface Serial de Periféricos.
[40] Protocolo de comunicación serie desarrollado por Phillips.
[41] Controller Area Network.
[42] Universal Serial Bus. Bus Serie Universal.
[43] Estándar para redes en sistemas de automóviles
[44] Radio Frecuencia. Usado para comunicaciones inalámbricas.
[45] Transmisión Control Protocol/Internet Protocol. Protocolo de Control de Transmisión/Protocolo de Internet

Familia enana y baja

Conocida así porque sus encapsulados son de 6 y 8 terminales. Está formado por los microcontroladores PIC 10F y 12F. Sus principales características son:

- La memoria de programa varía de 256 a 512 Bytes para la familia 10F y de 512 Bytes, 1KB y 2KB para la familia 12F.
- El tamaño de la palabra de la memoria de instrucciones es de 12 bits.
- Su juego de instrucciones es de 33.
- La pila sólo tiene dos niveles y No admiten interrupciones.
- La frecuencia máxima de operación de la familia 10F es de 4 y 8 MHz y de 4, 8 y 20MHz para la familia 12F, dependiendo del modelo.
- La familia 10F no implementa memoria de datos EEPROM[46] y en la familia 12F sólo 4 modelos la implementan, 3 con 128 Bytes y uno con 256 Bytes.
- Los recursos que implementa son:
- Sistema *Power On Reset* (POR). Sistema que inicializa el controlador a un estado conocido al conectarse a la alimentación.
- Sistema de Protección de Código (CP). Utilizado para proteger el programa almacenado en el PIC de copias no autorizadas, provocando que no pueda ser leída la memoria de instrucciones.
- Perro Guardián (WDT). Es un temporizador[47] que si no es refrescado antes terminar su cuenta reinicia el microcontrolador, evitando que éste se quede atrapado en un proceso sin fin.
- Líneas de entrada y salida de alta corriente. Usadas para excitar ciertos periféricos como LEDs[48]. Proveen entre 20 y 25 mA.
- Modo de Reposo o Bajo Consumo (SLEEP). Es un estado en el que entra el microcontrolador al ejecutarse la instrucción SLEEP. En este estado se detiene el oscilador principal y los periféricos asociados a éste. El estado de las líneas de E/S se conserva.
- Temporizador/Contador de 8 bits (TMR0). Algunos modelos de la familia 12F cuentan con más de un temporizador de 8 bits (TMR0) y uno más de 16 bits (TMR1).

[46] Electrically-Erasable Programmable Read-Only Memory. Memoria de Sólo Lectura Programable y Borrable Eléctricamente
[47] Un temporizador es un registro que se incrementa o decrementa y funciona como un cronómetro.
[48] LED. Light Emitting Diode. Diodo Emisor de Luz. Es un indicador luminoso.

Familia media

Existen tres microcontroladores en la familia media que son utilizados para la enseñanza de la programación de microcontroladores, los modelos 16F84A, 16F628, 16F877. Sus principales características son:

- La pila tiene ocho niveles.
- Admite diferentes fuentes de interrupción.
- La longitud en bits de las instrucciones es de 14 bits.
- Su juego de instrucciones es de 35.
- La memoria de programa varía de 1 y 2 KB a 4 y 8 KB para los microcontroladores de mayor capacidad de memoria.
- Pueden alcanzar una frecuencia máxima de operación de 20MHz.
- La memoria de datos EEPROM va desde 64 Bytes a 128 y 256 Bytes.

Implementan los mismos recursos que la familia baja y enana, pero dependiendo del modelo pueden variar sus prestaciones y periféricos. Algunos de ellos son:

- Convertidor Analógico Digital (CAD). Algunos modelos implementan un convertidor analógico digital de varios canales para trabajar con señales analógicas.
- Temporizadores de 8 y 16 bits (TMRx). Los microcontroladores de la familia media pueden incluir uno o más temporizadores de 8 o 16 bits para ser usados como contadores, ser usados junto al módulo CCP (Módulo de Captura, Comparación y Modulación de Pulsos) o para generar una reloj en tiempo real.
- Puertos de Comunicación Serie. La mayoría de los modelos incorporan terminales conectadas a periféricos que permiten la comunicación serie con otros microcontroladores, capaces de soportar uno o dos de los protocolos más usados, el RS-232 y el I^2C.
- Comparadores Analógicos. Son usados para comparar valores de voltajes analógicos.
- Módulo de Captura, Comparación y Modulación de Pulsos (CCP). Estos módulos son usados para el envío y recepción de señales digitales, por ejemplo, para el control de motores de corriente directa (CD).

Familia alta

La gama alta envuelve a los miembros de la familia PIC17F. Sus principales características son:

- Su juego de instrucciones es de 58.
- La longitud de la palabra de la memoria de instrucciones es de 16 bits.
- La pila cuenta con 16 niveles.
- Poseen 4 vectores de interrupción.
- Su arquitectura puede ser cerrada o abierta[49].

Familia mejorada

Con la aparición de nuevos modelos de microcontroladores se han aumentado las prestaciones y los recursos que se incluyen en cada modelo. La gama mejorada incluye microcontroladores de la familia 18F que son los microcontroladores de 8 bits más potentes de Microchip. Sus características son:

- Cuentan con un repertorio de 77 instrucciones de 16 bits.
- Una pila de 32 niveles.
- 4 vectores de interrupción.

Además incorporan nuevos periféricos, por ejemplo: módulos de comunicación serie que soportan protocolos CAN, USB o Ethernet; controladores de motores; pantallas de cristal líquido (LCD) y varios temporizadores de 8 y 16 bits.

Su principal característica es que pueden ser configurados para trabajar con una ***arquitectura cerrada***, es decir, haciendo uso de los periféricos que se encuentran dentro del mismo chip como un microcontrolador normal; o puede ser configurarlo para trabajar con una ***arquitectura abierta,*** como si se tratara de un microprocesador sacando las líneas de datos y de control a través de sus terminales, lo que permite aumentar las prestaciones del microcontrolador como la memoria de datos o los periféricos que incorpora.

[49] Una arquitectura abierta está relacionada con el diseño con microprocesadores, lo que permite sacar los buses internos para expandir las prestaciones del microcontrolador como la memoria de datos o de programa.

1.6.2 Arquitectura general de los microcontroladores PIC

En general la ***arquitectura de una computadora*** se refiere al tipo de datos, a como viaja la información a través de sus dispositivos y las características de cada elemento que la forma. Son dos las arquitecturas que podemos encontrar en el diseño de microcontroladores y microprocesadores: la ***arquitectura Von Neumann*** y la ***arquitectura Harvard.***

Arquitectura Harvard

Los microcontroladores PIC y los nuevos modelos de microcontroladores como los ***DsPIC***[50] de Microchip incorporan la arquitectura Harvard modificada, con lo cual el tamaño de la palabra de la memoria de programa puede ser 12, 14 o 16 bits, dependiendo de la familia a la que pertenezca el microcontrolador y no de 8 como la palabra de la memoria de dato

Procesador segmentado

Una técnica empleada en el diseño de microprocesadores para agilizar el procesamiento de las instrucciones es hacer uso de la ***segmentación encausada*** (en inglés *pipeline*). Consiste en dividir la ejecución de una instrucción en un número finito etapas, cada una con la misma duración. Cada etapa es realizada por un elemento del procesador de manera independiente, pasando el resultado de cada etapa a la etapa siguiente.

Para ejecutar una instrucción en un microcontrolador PIC es necesario invertir 8 ciclos de reloj, 4 ciclos de reloj son empleados en la búsqueda de la instrucción (etapa 1) y 4 son ocupados en su ejecución (etapa 2). Al implementar en un microcontrolador PIC la ***segmentación encausada*** es posible ejecutar la instrucción actual y de manera simultánea preparar la siguiente para su ejecución, reduciendo el tiempo que toma su ejecución a sólo 4 ciclos de reloj. Estos 4 ciclos de reloj son llamados ***ciclo de instrucción*** y es el tiempo que tarda el UCP del PIC en buscar, decodificar y ejecutar una instrucción. La mayoría de las instrucciones en un microcontrolador PIC duran un ciclo de instrucción, con excepción de las instrucciones que involucran saltos en el orden del programa, las cuales duran dos ciclos de instrucción, debido a que no se

[50] DsPIC ®. Digital Signal PIC ®. Es un microcontrolador diseñado para el procesamiento digital de señales, usado en aplicaciones de audio, video y comunicaciones, entre otras.

conoce el valor de la posición de la siguiente instrucción en la memoria de hasta que no haya sido ejecutada la instrucción de salto. Como ejemplo podemos analizar la siguiente lista de instrucciones donde la instrucción 4 es una instrucción de salto.

```
Instrucción No. 1   MOVLW   0x05     ;Cargar el registro de trabajo con 0x05
Instrucción No. 2   ADDWF   0x20, 0  ;Sumar el registro de trabajo con el registro 0x20
                                     ;y guardar resultado en el registro de trabajo
Instrucción No. 3   NOP              ; Consume un ciclo de instrucción
Instrucción No. 4   GOTO    0x05     ;Saltar a la dirección 0x05 de la memoria de
                                     ;programa
Instrucción No. 5   CLRF    0x06     ;Borrar el contenido del registro 0x06 de la
                                     ;memoria de datos
```

Con ayuda de la figura 1.7podemos ver que durante los primeros 4 pulsos de reloj se genera el primer ciclo de máquina: el ciclo 0. En este ciclo no existe una instrucción lista para ejecutarse ya que la primera instrucción aun está siendo preparada, es decir se encuentra en la etapa de búsqueda. Durante el ciclo 1, la instrucción 1 está lista para ser ejecutada y de manera simultánea el microcontrolador prepara la siguiente instrucción a ejecutarse. A partir del ciclo 2 de instrucción todas las instrucciones (excepto las de salto) duran un ciclo de instrucción en su ejecución.

En el ciclo 3 se prepara la instrucción de salto y en el ciclo 4 es ejecutada. La próxima instrucción a prepararse no se conoce hasta que la instrucción de salto se ejecute por completo, por lo que en el ciclo 5 no se ejecuta ninguna operación mientras es buscada la instrucción 5, haciendo que se pierda un ciclo de instrucción y que una instrucción de salto consuma 2 ciclos de instrucción para ser ejecutada. A partir del ciclo 6 las instrucciones se ejecutan en un ciclo de instrucción a menos que nos encontremos una nueva instrucción de salto en el programa.

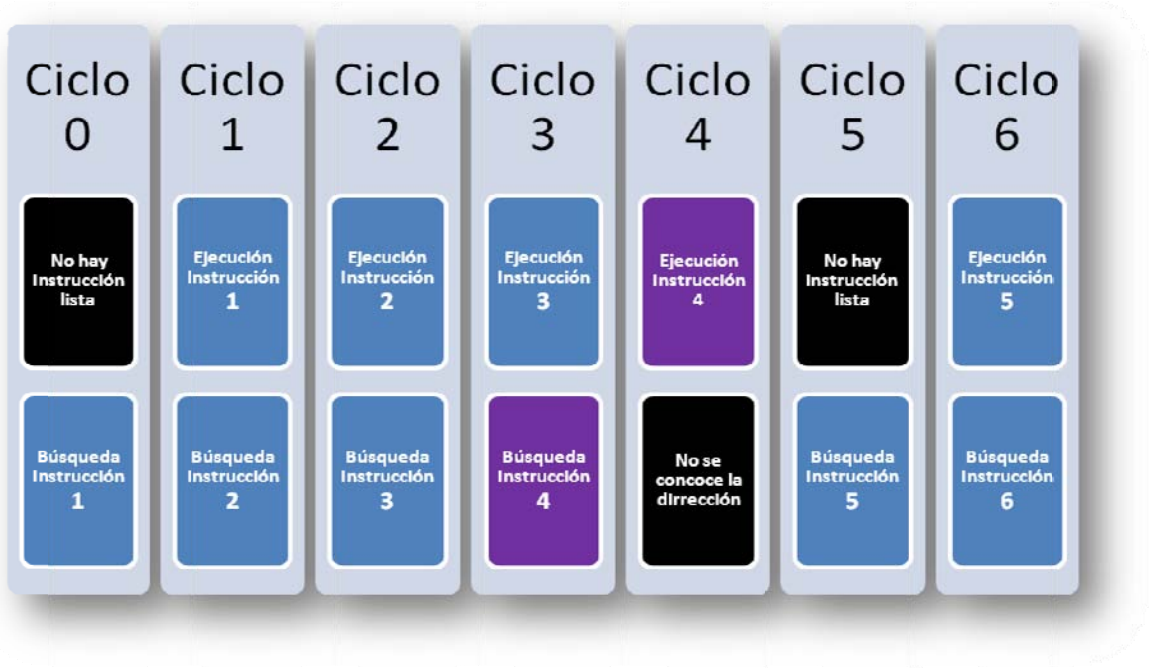


Figura 1.7. Proceso para la búsqueda y ejecución de un programa en un microcontrolador PIC.

Procesador RISC

Para los años 80 la corriente de construcción de microprocesadores estaba orienta a dotar a estos circuitos integrados con una gran cantidad de instrucciones, como consecuencia de buscar mantener la compatibilidad con antiguas versiones e incrementar su eficiencia. Algunas de estas instrucciones tardaban varios ciclos de máquina en ejecutarse y ocupaban más de una localidad de memoria por lo que se consideraba que eran complejas y poderosas.

En 1980 un grupo de investigadores de la Universidad de California Berkeley, dirigido por David A. Patterson y Carlo Séquin comenzó a diseñar chips de UCP, basados en trabajos previos, a los que dotaron de un juego de instrucciones más reducido que se ejecutaban con rapidez, con lo que podía construirse instrucciones igual de poderosas y complejas como las que tenían los demás procesadores de la época a partir de la combinación de instrucciones más sencillas.

El acrónimo ***RISC*** (*Reduced Instruction Set Computer*) que significa *computadora de juego de instrucciones reducido* fue utilizado para describir el diseño de microprocesadores que contaban con un número pequeño de instrucciones, alrededor de 50 que se ejecutan por lo regular en un ciclo de máquina y que tienen un mismo formato[51]. Un elemento del microprocesador llamado decodificador de instrucción se encarga de indicarle a la unidad aritmético-lógica del microprocesador de que instrucción se trata y de habilitar los elementos que necesita para ejecutar la instrucción. Ya que las instrucciones están diseñadas con un mismo formato, es más fácil decodificarlas y ejecutarlas.

Para guardar la compatibilidad con arquitecturas ya definidas se siguieron construyendo procesadores que fueron conocidos como de arquitectura ***CISC*** (*Complex Instruction Set Computer*) o *computadoras de juego de instrucciones complejo* que reunían un conjunto de 200 a 300 instrucciones.

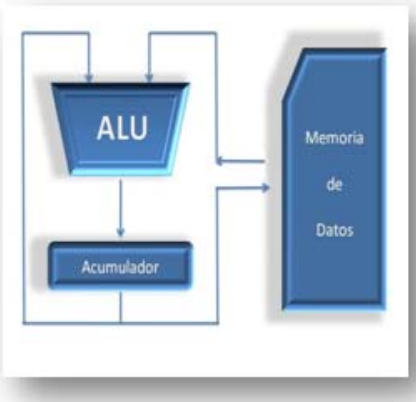
Existe otra tendencia en el desarrollo de microprocesadores, los de tecnología ***SISC*** (*Specific Instruction Set Computer*) que hacen referencia a *procesadores con un juego de instrucciones específico* como los usados para la codificación-decodificación digital y procesamiento de archivos de música, imagen y video.

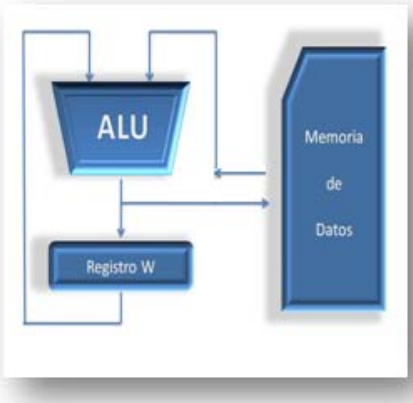
Arquitectura ortogonal

En una ***arquitectura tradicional*** o no ortogonal, los resultados de una operación que realice la Unidad Aritmética-Lógica (ALU) se guardan en un ***registro de propósito específico*** llamado ***Acumulador***. El resultado puede ser utilizado entonces como operando de una nueva instrucción o ser movido a un registro de la memoria de datos (figura 1.8).

En una ***arquitectura ortogonal*** (figura 1.9), el resultado de la ejecución de una instrucción pueden guardarse no sólo en el acumulador para ser usado como operando de una nueva instrucción, sino también puede ser movido directamente a cualquier registro de la memoria de datos sin pasar por el acumulador. En los microcontroladores PIC, el registro que funciona como *acumulador* se llama ***registro de trabajo*** o ***registro w*** (del inglés *work*).

[51] El formato de la instrucción se refiere a la cantidad de bits que son usados para representar la instrucción, y los registros involucrados, generalmente en una sola localidad de memoria.





Un microprocesador con arquitectura ortogonal tiene la ventaja de que ahorra el uso de instrucciones, ya que cualquier resultado de una instrucción puede colocarse en el registro de trabajo o en el mismo registro de donde se tomo el dato ahorrando una instrucción de movimiento en una arquitectura tradicional.

Arquitectura basada en un banco de registros

Dentro del microcontrolador se encuentran diferentes periféricos que auxilian al microcontrolador a realizar la tarea para la que ha sido programado. Estos elementos deben ser configurados y la forma de hacerlo es muy sencilla.

Existen diversos registros asociados a cada periférico, que permiten configurarlo y comprobar su estado. Estos registros se encuentran dentro de un área conocida como SRF (*Special Registers Function*) en la memoria de datos, por lo que cada periférico está implementado físicamente como registros.

1.6.3 Alimentación Eléctrica

Un microcontrolador funciona con una alimentación eléctrica de 5V como todos los circuitos integrados de la familia TTL[52]. Los microcontroladores PIC tienen dispuestos al menos dos terminales para la alimentación eléctrica, una que se conecta al voltaje positivo (5V) y otra tierra (0V). La figura 1.10 muestra como se conectan las señales de alimentación eléctrica del PIC.

De manera típica se considera que el voltaje mínimo de alimentación para que el microcontrolador funcione adecuadamente es de 4.5V y el máximo de 5.5V antes de que el microcontrolador sufra un daño irreparable. Es necesario considerar que a mayor frecuencia de trabajo el microcontrolador PIC consumirá mayor corriente, lo que implica que disipará mayor cantidad de calor. Existen modelos que soportan diferentes rangos de temperatura, humedad y frecuencia de trabajo, lo que permite elegir el microcontrolador adecuado a la aplicación que estemos diseñando.

En aplicaciones donde es necesario un consumo menor de energía eléctrica se usan los microcontroladores de bajo consumo de potencia. Estos microcontroladores se pueden alimentar desde 2.5V hasta 5.5V lo que permite que funcionen en sistemas donde el ahorro de energía y la disipación de calor son un factor importante. Estos microcontroladores funcionan bajo el mismo principio de la frecuencia de trabajo, a mayor frecuencia se consume mayor corriente, por lo que el nivel mínimo de alimentación se incrementa proporcionalmente entre los 2.5V y los 4.5V

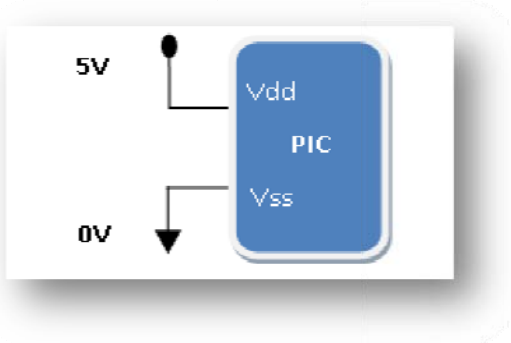


Figura 1.10. Conexión de las terminales Vdd y Vss a la alimentación eléctrica.

[52] TTL. Lógica Transistor-Transistor. La familia de circuitos integrados TTL está diseñada con arreglos de transistores.

1.6.4 Oscilador principal1

Para fijar la velocidad de trabajo del microcontrolador existen diferentes fuentes de reloj. Los elementos usados para fijar la velocidad de operación conectan a dos terminales conocidas con OCS1/CLKIN y OSC2/CLKOUT dependiendo del tipo de reloj que se utilice.

- Oscilador LP: Cristal de baja velocidad y bajo consumo de potencia.
- Oscilador XT: Cristal de cuarzo o resonador cerámico.
- Oscilador HS: Cristal de cuarzo o resonador cerámico de alta velocidad.
- Oscilador RC: Oscilador hecho con una arreglo de resistencia y capacitor.
- Oscilador IntRC: Oscilador interno hecho con una arreglo de resistencia y capacitor.
- Oscilador ER: Oscilador que utiliza una resistencia externa para hacer funcionar un arreglo
- RC Interno en el microcontrolador.

Oscilador RC

Un oscilador RC está formado por una resistencia y un capacitor. Es útil en aplicaciones donde no se requiere exactitud en los tiempos (como aplicaciones que no usan los puertos de comunicación serie) ya que su mayor inconveniente es que no se trata de un oscilador muy exacto pero si muy económico. La figura 1.11 muestra el arreglo de este oscilador y la tabla 1.1 algunos valores para el arreglo RC.

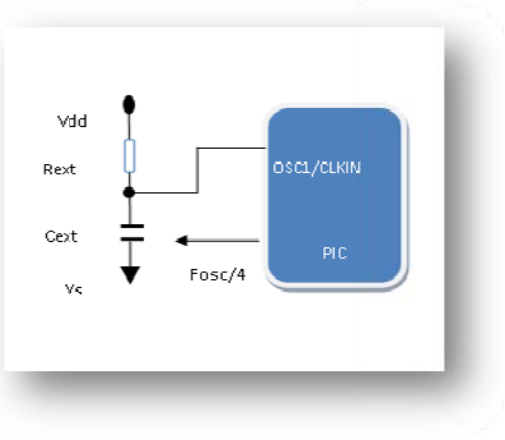


Figura 1.11. Conexión de un oscilador RC.

Tabla 1.1. Valores de los componentes para un oscilador RC.

Cext	Rext	Frecuencia
20 pF	5 KΩ	4.61 MHz
	10 KΩ	2.66 MHz
	100 KΩ	311 KHz
100 pF	5 KΩ	1.34 MHz
	10 KΩ	756 KHz
	100 KΩ	82.8 KHz
300 pF	5 KΩ	428 KHz
	10 KΩ	243 KHz
	100 KΩ	26.2 KHz

Los valores recomendados para Rext y Cext están dados por la relación: 5KΩ ≤ Rext≤ 100 KΩ y Cest >20 pF. A través de la terminal OSC2/CLKOUT se proporciona la cuarta parte de la frecuencia de oscilación total, esta señal puede ser aprovechada para ingresarse por la terminal OSC1/CLKIN de otro microcontrolador y así hacer funcionar varios microcontroladores en cascada partir de una sola señal de reloj.

Oscilador LP

Un oscilador LP es un cristal de cuarzo de baja velocidad, y por tanto, de bajo consumo de potencia. Mientras mayor sea la frecuencia del oscilador, mayor será la velocidad a la que trabaje el microcontrolador, su consumo de corriente y el calor que disipe. Un cristal LP es de bajo consumo de potencia, su frecuencia de trabajo está comprendida entre 32kHz y 200kHz. Para funcionar, deben conectarse las entradas del cristal a las terminales OSC1/CLKIN y OSC2/CLKOUT respectivamente y un capacitor cerámico entre cada terminal y la tierra del circuito. La figura 1.12 muestra el diagrama de conexión, siendo la misma para los osciladores LP, XT y HS.

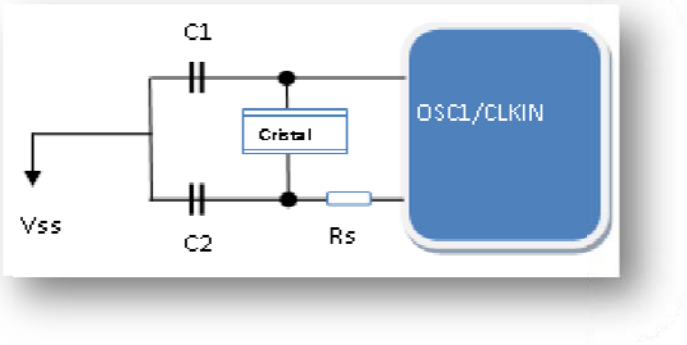


Figura 1.12. Diagrama de conexión del oscilador LP, XT y HS a las terminales del microcontrolador PIC.

Oscilador XT

Un oscilador XT puede ser un resonador cerámico o un cristal de cuarzo de mediana velocidad como el de la figura 1.13. Los cristales XT son los más usados y comprenden una frecuencia entre 100kHz y 4MhZ dependiendo del modelo de microcontrolador. Los valores típicos de los capacitores cerámicos usados para la conexión de un oscilador XT en un microcontrolador PIC16F84A están en la Tabla 1.2.

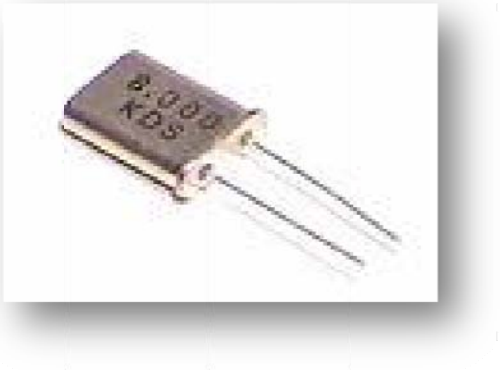


Figura 1.13. Cristal de Cuarzo de 8Mhz.

Tabla 1.2. Valor de capacitores para cristales de cuarzo para el microcontrolador PIC 16F84A.

Tipo de Oscilador	Frecuencia del Cristal	Valor del Cap. 1	Valor del Cap. 2
LP	32 KHz	68 - 100 pF	68 - 100 pF
	200 KHz	15 – 33 pF	15 – 33 pF
XT	100 KHz	100 – 150 pF	100 – 150 pF
	2 MHz	15 – 33 pF	15 – 33 pF
	4 MHz	15 – 33 pF	15 – 33 pF
HS	4 MHz	15 – 33 pF	15 – 33 pF
	20MHz	15 – 33 pF	15 – 33 pF

Oscilador HS

El oscilador de cristal o resonador de alta velocidad (HS) trabaja a una frecuencia comprendida entre 4MHz y 20MHz para el PIC16F84A que será el primer microcontrolador que analizaremos a detalle. Recordemos que para la familia de la gama alta o los PIC 18F su velocidad máxima es de 40 MHz.

En ocasiones es necesario colocar una resistencia (Rs), como lo muestra la figura 1.9, entre el oscilador y la terminal OSC2/CLKOUT dependiendo del modelo de microcontrolador, para estabilizar la frecuencia de operación.

1.6.5 Reinicio (Reset)

El sistema de Puesta a Cero o Reinicio del Sistema (RESET) se utiliza para poner el microcontrolador en un estado conocido. Saber que origina el reinicio del microcontrolador es fácil, sólo es necesario checar dos bits del registro de estado (STATUS) del microcontrolador o el registro PCON. La mayoría de los microcontroladores PIC responden a varias fuentes de reinicio:

- *Power-on* Reset (POR). Reinicio al conectar la alimentación eléctrica.
- Reinicio a través de la terminal MCLR durante la operación normal del microcontrolador.
- Reinicio a través de la terminal MCLR durante la el estado de bajo consumo (SLEEP) del microcontrolador.
- Reinicio por el desbordamiento del Perro Guardián (WTD) durante la operación normal del microcontrolador.
- *Brown-on* Reset (BOR). Reinicio por falla en la alimentación durante la operación normal del microcontrolador.
- Reinicio por error de paridad (PER).

El contenido de la mayoría de los registros del microcontrolador no son afectados al entrar en el estado de reinicio. Para conocer como es afectado cada uno de los registros de un modelo en particular es necesario revisar el capítulo dedicado al estado de reinicio del manual de operación de cada microcontrolador.

En algunas ocasiones el microcontrolador no operará de manera correcta, debido a diversas causas como errores en el diseño del programa o que el microcontrolador espere un evento que nunca ocurre. Para que el microcontrolador regrese al inicio del programa será necesario activar la señal de reinicio. Para hacerlo podemos hacer uso de varias circuitos eléctricos.

La figura 1.14a muestra cómo debe conectarse la terminal MCLR para que el microcontrolador esté en modo de operación normal y la figura 1.14b como debe conectarse esta terminal para que el microcontrolador esté en el estado de reinicio.

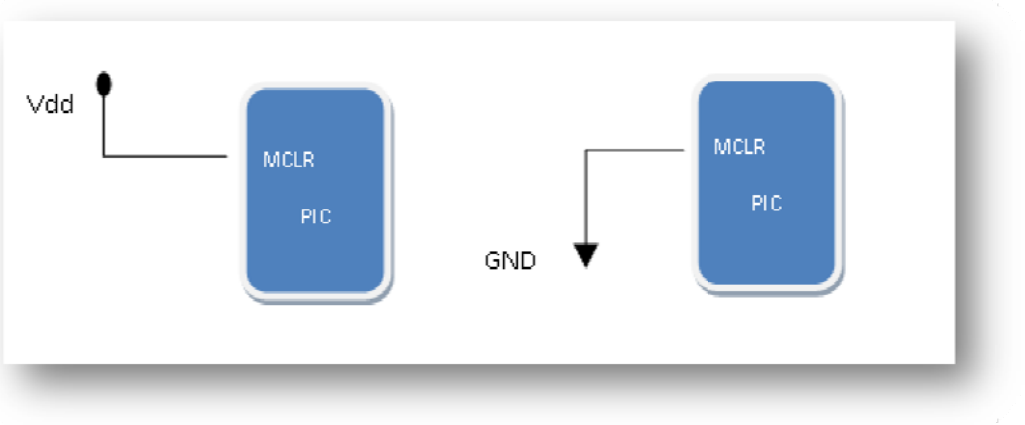


Figura 14a. Conexión de la terminal MCLR del microcontrolador en estado normal.

Figura 14b. Conexión de la terminal MCLR del microcontrolador en estado de reinicio.

Para conmutar entre los dos estados puede conectarse un circuito eléctrico como el de la figura 1.15. Si el interruptor está abierto, el microcontrolador recibirá un nivel de voltaje alto por la terminal MCLR, lo que supone que el microcontrolador está en un estado de operación normal. Cuando el interruptor es presionado, la terminal MCLR recibirá un nivel de voltaje bajo, por lo que entrará en el estado de reinicio. Mientras el microcontrolador se encuentre en estado de reinicio no trabajará. Cuando pase al estado normal el microcontrolador ejecutará el programa desde el inicio. Existen más circuitos que realizarán la misma tarea, previniendo rebotes en la señal o que retardan la generación del pulso, pero este es el más simple y económico.

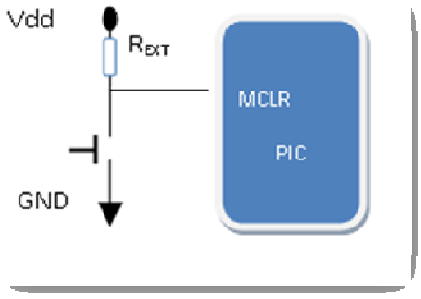


Figura 1.15. Circuito básico para conmutar entre el estado de reinicio y en estado normal.

1.6.6 Tipos de Encapsulados

Una de las etapas del diseño de soluciones tecnológicas con microcontroladores es elegir el encapsulado que usaremos en el prototipo final. Elegir el encapsulado más adecuado permite aprovechar el espacio en la placa de circuitos y una mejor distribución de los componentes.

Existen gran cantidad de encapsulados (figura 1.16) y seguramente emplearemos el encapsulado de Doble Línea (DIP) durante la etapa de desarrollo y prueba, pero en el prototipo final un encapsulado de montaje superficial o uno de PLCC (Encapsulado con terminales a los cuatro lados) pueda mejorar la distribución del espacio.

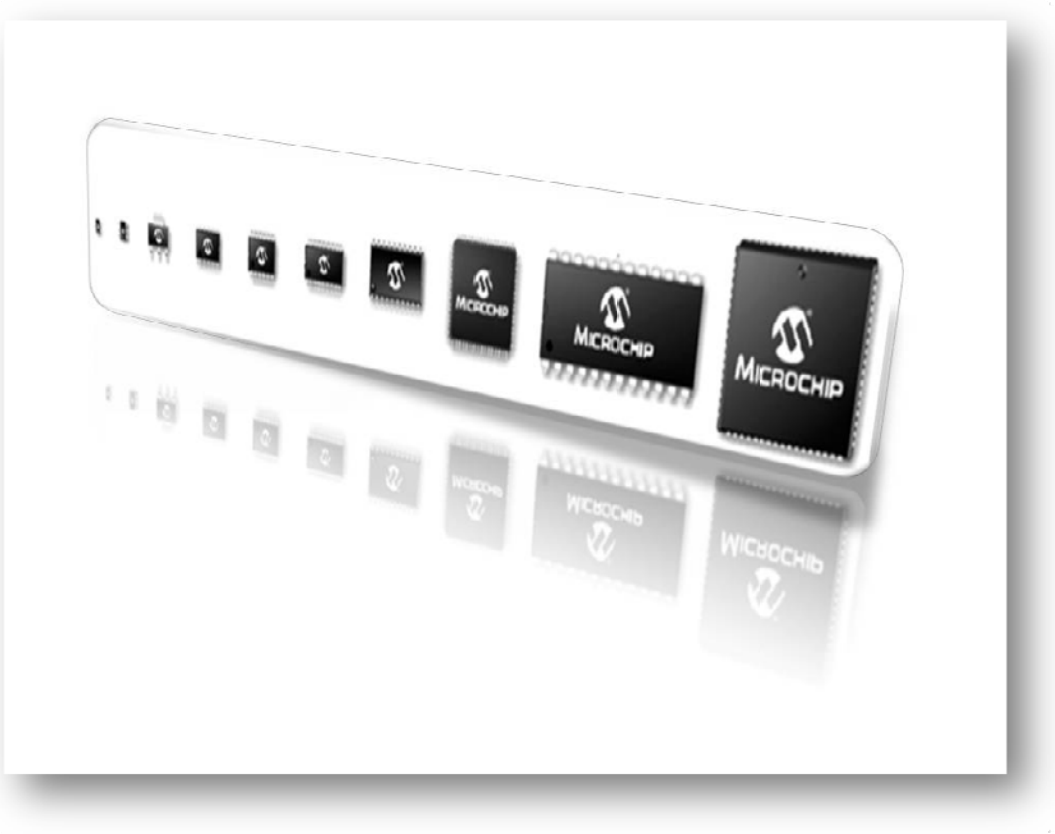


Figura 1.16. Diferentes encapsulados de Microcontroladores PIC.

DIP (*Dual Inline Package*)

Este encapsulado es quizá el más conocido (figura 1.17). Presenta dos líneas de terminales en lados opuestos (de ahí su nombre). Sus dimensiones varían dependiendo del número de terminales. Las terminales se cuentan en sentido antihorario, comenzando por la número 1 que es indicada por una marca o hendidura.

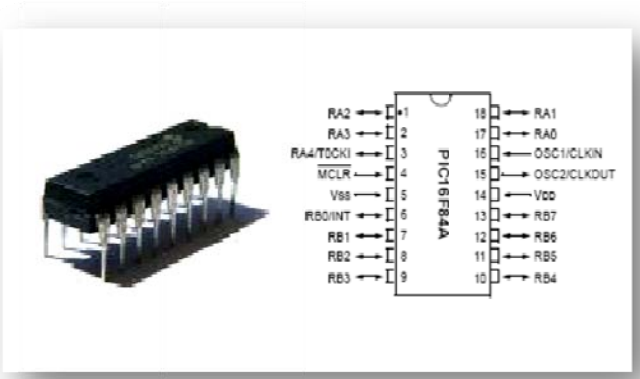


Figura 1.17.Micocontrolador PIC 16F84A en su encapsulado DIP.

SOIC (Small Out-Line Integrated Circuit)

Puede decirse que este tipo de encapsulado es el equivalente a un encapsulado DIP pero de montaje superficial, es decir, se suelda a una cara de la placa por lo que no hay necesidad de perforarla. Sus terminales simulan la forma de un ala de gaviota (figura 1.18)

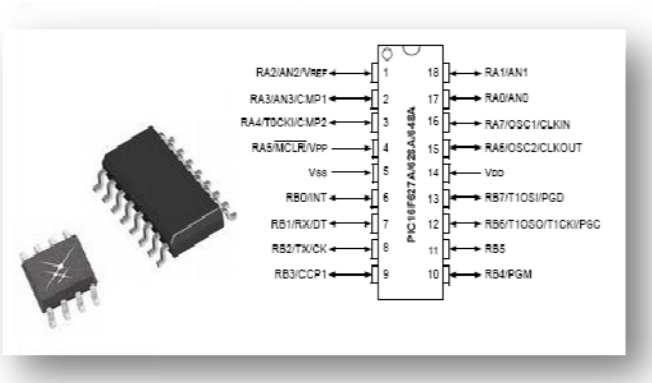


Figura 1.18.Micocontrolador PIC 16F628A en su encapsulado SOIC.

PLCC (Plastic Leaded Chip Carrier)

Este tipo de encapsulados de circuito integrado necesita de una base para ser colocado sobre la placa del circuito que gobernará (figura 1.19) por lo que no puede ser soldado directamente a la placa. Las terminales están dispuestas en los cuatro lados del encapsulado y son numeradas a partir de la terminal uno que se indica con una marca en sentido antihorario. Para retirarlo de la base es necesario auxiliarse de una pinza especial. Sus terminales tienen una forma como de *J*, lo que permite que se sujeten a la base y hagan contacto con las terminales de la misma base.

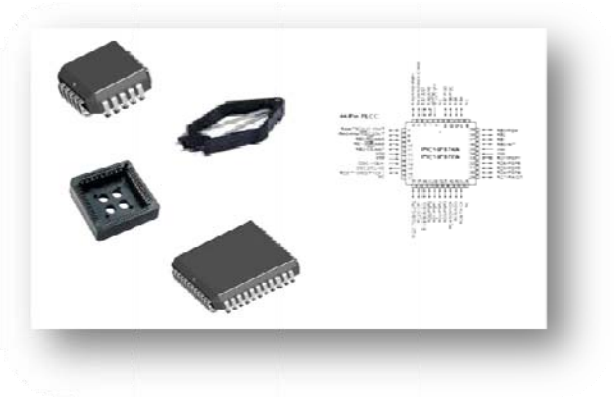


Figura 1.19. Microcontrolador PIC 16F877A en su encapsulado PLCC.

SSOP (Shrink Small Out-Line Package)

Este tipo de encapsulado es similar al SOIC (figura 1.20), con la diferencia de que el espacio entre terminales es más pequeño. Poseen dos líneas de terminales a lados opuestos del encapsulado en forma de ala de gaviota. El número de terminales varía de 6 a 28.

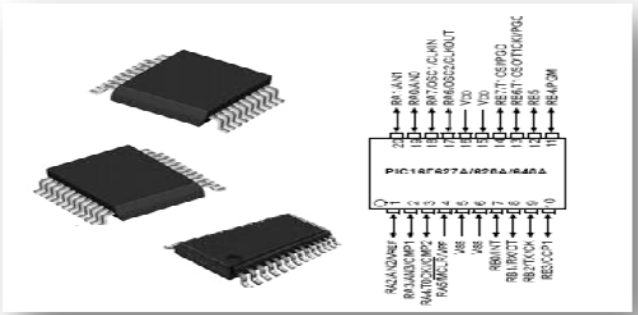


Figura 1.20. Microcontrolador PIC 16F627A en su encapsulado SSOP.

TQFP *(Thin Quad Flat-Pack Package).*

Este tipo de encapsulados es similar al PLCC ya que tiene terminales a los cuatro lados del encapsulado. Las terminales son en forma de ala de gaviota y es una variante del QFP que tiene el espacio entre terminales más pequeño (figura 1.21). Estos circuitos integrados no se sueldan a la placa, sino que se adhieren a ella con una pasta especial termo adherible.

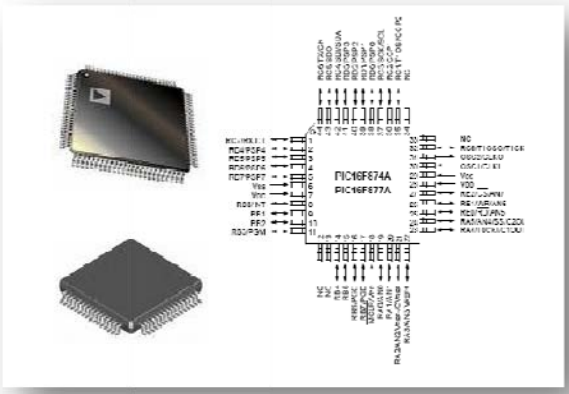


Figura 21. Microcontrolador PIC 16F877A en su encapsulado TQFP.

QFN (*Quad Flat-Pack Package No Leads*)

Un encapsulado de circuito integrado de este tipo es similar al QFP (figura 1.22), pero sus terminales no están extendidas hacia afuera, sino hacia dentro. Es unido a la placa con una pasta termo-adherible por lo que no necesita ser perforada la placa de circuito impreso.

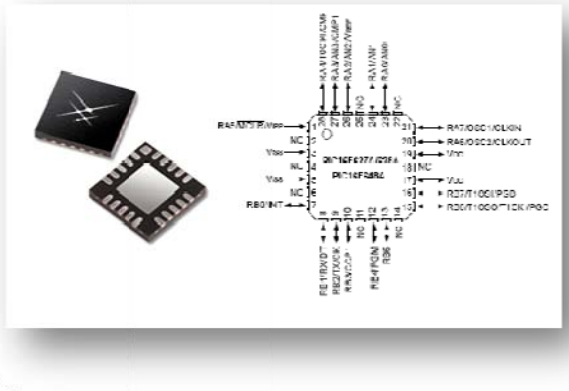


Figura 1.22. Microcontrolador PIC 16F877A en su encapsulado QFN.

1.6.7 Sistema de identificación de los microcontroladores PIC

Microchip usa un sistema para identificar cada uno de sus modelos de microcontroladores y conocer sus características. Este sistema consta de colocar una serie de números y letras en una etiqueta sobre el microcontrolador.

Una etiqueta como la de la figura 1.23 tiene el siguiente significado:

Figura 1.23. Ejemplo de una etiqueta de un microcontrolador PIC.

- Indica que el modelo de microcontrolador pertenece a la familia PIC16.
- El modelo del microcontrolador es 16F84A.
- Una letra F indica que el tipo de memoria de programa es FLASH, lo que permite que sea borrada eléctricamente. Una letra C indica que el tipo de memoria no puede ser borrada (OTP) y una R que se trata de una memoria tipo ROM.
- El número 4 indica la máxima frecuencia de operación que en este caso es de 4MHz.
- Una letra I (industrial) indica que el rango de temperatura es de -40ºC a +85ºC. una letra E (Extended) indica que se trata de un rango extendido de -40ºC a 125 ºC.

La última letra o letras indican el tipo de encapsulado que en este cado es de tipo PDIP.

- ML = QFN (Metal Lead Frame)
- PT = TQFP (Thin Quad Flat pack)
- SO = SOIC
- SP = Skinny Plastic DIP
- P = PDIP
- L = PLCC
- S = SSOP

Ejemplos de este sistema de identificación son:

1. PIC16LF876A-I/SO =Microcontrolador PIC16F84A de bajo consumo y memoria flash. Rango de temperatura industrial y encapsulado SOIC.
2. PIC16F877A-I/P = Microcontrolador PIC16F877a de consumo normal y memoria flash. Rango de temperatura industrial y encapsulado PDIP.
3. PIC16C627 –E/PT = Microcontrolador PIC 16F627 de consumo normal y memoria OTP. Rango extendido de temperatura y encapsulado QTFP.
4. PIC16LF627A - I/SO = Microcontrolador PIC16F627A de bajo consumo y memoria Flash. Rango de temperatura industrial y encapsulado SOIC.

1.7 Conclusión del capítulo

Desarrollar soluciones tecnológicas con microcontroladores supone tener conocimiento de varias disciplinas de la ciencia y la tecnología. Este primer acercamiento permitió conocer nuestra principal herramienta de trabajo, el microcontrolador PIC, sus características y su ubicación como parte de un mercado mundial.

Ahora que conocemos los pormenores de los microcontroladores, será necesario conocer las herramientas de las que puede disponerse para desarrollar soluciones tecnológicas usando microcontroladores PIC.

Capítulo 2

Herramientas de Microchip para el desarrollo de aplicaciones con microcontroladores PIC

Capítulo 2
Herramientas de Microchip para el desarrollo de aplicaciones con microcontroladores PIC

Un factor que influye en la elección del microcontrolador a usar para desarrollar un ***sistema embebido***, es si disponemos de las herramientas necesarias para implementar de principio a fin nuestro proyecto. De manera general, necesitaremos de un lenguaje de programación, compilador y programador con *software*.

Microchip es líder en la producción microcontroladores de 8 bits y proporciona sistemas completos de desarrollo que dan soporte a diseñadores y estudiantes, para el aprendizaje y prueba de los prototipos diseñados con microcontroladores PIC. Algunas de estas herramientas son de distribución libre, como el ambiente de desarrollo integrado MPLAB® IDE y algunas otras tienen un precio accesible como los programadores, tarjetas entrenadoras o depuradoras en circuito.

Además, es posible encontrar en el sitio de Internet de Microchip Inc.[1] toda la documentación técnica de sus dispositivos, artículos con ejemplos de aplicaciones desarrolladas con Microcontroladores PIC, conseguir muestras de circuitos integrados y el *software* necesario para el desarrollo de proyectos. Todo esto, junto a las características propias de estos dispositivos, hace que los microcontroladores PIC de Microchip Inc. sean una buena opción para iniciarse en la programación de microcontroladores.

[1] www.microchip.com

En este capítulo serán expuestas, de manera general, las principales características de las herramientas que Microchip ofrece para el desarrollo de aplicaciones con sus microcontroladores PIC.

2.1 Herramientas en *software* de Microchip

Las principales características de las herramientas que Microchip ofrece para el desarrollo de aplicaciones con sus microcontroladores PIC, se resumen en la tabla 2.1. Las herramientas en *software* que se usan para desarrollar aplicaciones con microcontroladores PIC básicamente son: ***compiladores*** y ***simuladores***, éstas pueden estar integradas, junto a otras herramientas, en un ***ambiente de desarrollo integrado*** o IDE (*Integrate Developed Environment)* por sus siglas en inglés.

Tabla 2.1. Principales herramientas de Microchip en *hardware* y *software* para el desarrollo de aplicaciones con microcontroladores PIC.

Herramientas de desarrollo de Microchip	
MPLAB® IDE	Entorno de Desarrollo Integrado
MPASM™ *Assembler*	Macro-ensamblador universal para microcontroladores PIC
MPLINK™ Object Linker	Enlazador
MPLIB™ Object Librarian	Biblioteca del enlazador
MPLAB C17	Compilador de C para microcontroladores MCU PIC17CXXX
MPLAB C18	Compilador de C para microcontroladores MCU PIC17CXXX
C Compilers	Compiladores de C de otros fabricantes para microcontroladores PIC
MPLAB SIM Simulator	Programa de simulación
MPLAB ICD	Depurador en circuito
ICEPIC™ Emulator	Emulador en circuito de bajo costo
MPLAB ICE 2000	Emulador en circuito
PICSTART® Plus Programmer	Juego básico de desarrollo con programador
PRO MATE® II Device Programmer	Programador modular

El lenguaje que una computadora digital emplea, está formado por cadenas de unos y ceros conocidos como bits. Cada fabricante diseña sus computadoras con un juego de instrucciones, por lo que la misma cadena de bits puede representar una instrucción totalmente distinta de una computadora a otra. Este conjunto de cadenas de bits es conocido como ***lenguaje de máquina***. Programar una computadora a este nivel es posible, pero es un trabajo complejo, por lo que es fácil cometer errores y difícil encontrarlos para darles una solución.

Existen tantos dialectos de lenguaje de máquina como arquitecturas sea posible crear. Para los años 50 se propuso una forma más humana para referirse a cada instrucción y sus argumentos donde se hace uso de expresiones simbólicas o nemónicos. El conjunto de estas expresiones (generalmente siglas en inglés) se conoce como lenguaje ensamblador y es definido por los fabricantes para cada uno de sus computadoras.

Programar en ***lenguaje ensamblador*** significa una ventaja a diferencia de hacerlo en ***lenguaje máquina***, ya que es más fácil recordar una expresión como **MOVF PORTA, W** en lugar de 00 1000 0000 0101, esta instrucción puede leerse como *copiar el contenido del registro PORTA en el registro de trabajo W*. Por la dependencia que tiene el ***lenguaje ensamblador*** con la arquitectura del microcontrolador o microprocesador se dice que es un ***lenguaje de bajo nivel***. La principal ventaja de programar en ***lenguaje ensamblador*** es la optimización de recursos (como el empleo de menor cantidad de memoria) y la implementación de algoritmos que se ejecutan más rápidamente a diferencia de generar código ejecutable a través de un compilador de lenguaje de alto nivel.

2.1.1 Lenguaje ensamblador para microcontroladores PIC

El lenguaje ensamblador MPASM es un lenguaje universal para todos los microcontroladores PIC. Esto permite la migración y reutilización de código entre dispositivos de la misma familia o de un microcontrolador con menores prestaciones a uno con mayores recursos. Un programa en ensamblador es escrito en un editor de texto sin formato (texto ASCII). Este archivo es conocido como código fuente y suele tener la extensión “.asm” y en casos especiales la extensión “.inc”.

Un archivo de cabecera (.inc) es un archivo que contiene código ensamblador válido y definiciones de registros específicos del dispositivo y sus bits, este archivo puede ser incluido en el código fuente para ser reutilizado en otros programas, de manera similar a como se hace con los archivos de cabecera "*.h*" en lenguaje C.

Estructura del lenguaje ensamblador MPASM

El lenguaje ensamblador está formado por 6 tipos de ***componentes léxicos*** (*tokens*)***:*** *etiquetas*, *mnemónicos*, *directivas*, *macros*, *operandos* y *comentarios*.

Las ***etiquetas*** son usadas para representar un valor constante o un grupo de líneas de código, por ejemplo, en el caso de las instrucciones de salto. Inician en la columna 1 y pueden ir seguidas de un signo de dos puntos (:), espacio, tabulación o un salto de línea. Deben iniciar con una letra o con un guión bajo (_) y van seguidos de uno o más caracteres alfanuméricos, el guión bajo o el signo de cierre de interrogación (?).

Las ***etiquetas*** no pueden ser palabras reservadas por el compilador, ni iniciar con un número o dos guiones bajos. Pueden contener hasta 32 caracteres, pero no se recomienda manejar etiquetas tan largas, ya que puede convertirse en un trabajo difícil para el programador debido a que las ***etiquetas*** son sensibles de ser escritas en mayúsculas o minúsculas. Así que dos ***etiquetas*** son iguales sólo si se escriben de manera idéntica, si una sola letra cambia de mayúscula a minúscula o viceversa, será tratada como una ***etiqueta*** totalmente diferente. Si los dos puntos son usados al definir la ***etiqueta***, éstos son tratados como un operador y no como parte de la etiqueta en sí, por lo que pueden ser omitidos es usos diferentes a la definición.

Los ***mnemónicos*** son palabras o siglas en idioma inglés que representan las instrucciones que serán traducidas a ***códigos de operación*** para ser ejecutadas directamente por la arquitectura del microcontrolador, por ejemplo *add* (suma), *goto* (salta) o *mov* (mueve), A diferencia de las ***etiquetas*** que son definidas por el programador, los ***mnemónicos*** están bien definidos en el lenguaje ensamblador y no son sensibles a su escritura en mayúsculas o minúsculas, por lo que al escribir MOVLW, movlw o MOVlw son interpretadas por el compilador como la misma instrucción.

Las ***directivas*** son instrucciones que le indican al compilador cómo debe generar el ***código máquina***, por lo que generalmente no se traducen a ***códigos de operación***. Son usadas para controlar las entradas y salidas del compilador, así como la localización de los datos. Al igual que los ***mnemónicos***, no son sensibles a su escritura en mayúsculas o minúsculas.

Las ***macros*** son un conjunto de instrucciones y directivas definidas por el usuario para ser evaluadas en el momento de la compilación siempre que la macro es invocada. Los ***mnemónicos***, ***directivas*** y llamadas a ***macros*** deben iniciar en la columna 2.

Los ***operandos*** dan información al compilador sobre el dato que debe ser usado para formar el ***código de operación***, como su valor o lugar donde se almacena. Si la instrucción necesita de múltiples ***operandos***, estos deben estar separados por comas. Una ***literal*** puede ser usada como un ***operando***, lo mismo que una ***etiqueta***. Existen diferentes formas para representar ***literales***, como son en formato ASCII, binario, hexadecimal, octal y decimal, según la tabla 2.2

Tabla 2.2. Representación de literales en lenguaje ensamblador MPASM. Todos los ejemplos representan el mismo valor constante en diferentes formatos.

Tipo de literal	Sintaxis	Ejemplo
Hexadecimal	0x<literal>	0x41
	H'<literal>'	H'41'
	h'<literal>'	h'41'
	<literal>H	'41'H
	<literal>h	'41'h
Decimal	D'<literal>'	D'65'
	d'<literal>'	d'65'
	. <literal>	.65
Binario	B'<literal>'	B'01000001'
	b'<literal>'	b'01000001'
Octal	O'<literal>'	O'101'
	o'<literal>'	o'101'
ASCII	'<literal>'	'A'
string	"<literal>"	"A"

Los ***comentarios*** son texto dentro del ***código fuente*** que sirve al programador para documentar su programa. A través de los ***comentarios*** es posible explicar el funcionamiento de una o varias líneas de código. El compilador MPASM interpreta como un ***comentario*** todo el texto que encuentre después de un punto y coma (;) y hasta el final de esa línea. Los ***comentarios*** no pueden agruparse en múltiples líneas como se hace en lenguaje C++ con los símbolos /* y */. Para que varias líneas sean interpretadas como comentario, será necesario escribir un punto y coma al inicio de cada línea. Si un carácter de punto y como se encuentra dentro de una cadena de caracteres (*string*), es decir, dentro de comillas dobles (“), los caracteres que se encuentren después del punto y coma no serán tomados como un ***comentario***.

Como ejemplo de la estructura de un archivo de código fuente, ya sea para un archivo de ensamblador (.asm) o de cabecera (.inc), podemos observar la tabla de código 2.1, donde su estructura sigue los siguientes criterios:

Tabla de Código 2.1 Distribución del código fuente en lenguaje ensamblador.

```
                Mnemónicos
                Directivas
Etiquetas       Macros          Operandos       Comentarios
↓               ↓               ↓               ↓

          LIST          P = 18f452
          INCLUDE       <p18f452.inc>
Dest      EQU           0x0B            ; Definir constante

          ORG           0x0000          ; Vector de reset
          GOTO          Inicio

          ORG           0x0020          ; Inicio del programa
Inicio
          MOVLW         0x0A
          MOVWF         Dest
          BCF           Dest, 3         ; Esta línea usa dos operandos
          GOTO          Inicio

          END
```

1. Cada línea del ***código fuente*** puede contener hasta cuatro campos:
    - Etiquetas
    - Mnemónicos, directivas y macros
    - Operandos
    - Comentarios
2. El orden y posición de los elementos en cada campo es importante. Las ***etiquetas*** se colocan en la columna uno; los ***mnemónicos***, ***directivas*** y ***macros*** en la columna dos; los ***operandos*** siguen a los ***mnemónicos*** y los ***comentarios*** a los ***operandos***. El ancho de cada columna no debe superar los 255 caracteres.
3. Los espacios en blanco o columnas deben separar las ***etiquetas*** de los ***mnemónicos*** y a su vez a los ***mnemónicos*** de los ***operandos***. Una lista de múltiples ***operandos*** debe estar separada por comas. Los espacios en blanco son uno o más espacios o tabulaciones. Son usados para hacer el ***código fuente*** más legible, pues separan y ordenan los campos de una línea.
4. Cada línea del ***código fuente*** puede contener hasta cuatro campos:
    - Etiquetas
    - Mnemónicos, directivas y macros
    - Operandos
    - Comentarios
5. El orden y posición de los elementos en cada campo es importante. Las ***etiquetas*** se colocan en la columna uno; los ***mnemónicos***, ***directivas*** y ***macros*** en la columna dos; los ***operandos*** siguen a los ***mnemónicos*** y los ***comentarios*** a los ***operandos***. El ancho de cada columna no debe superar los 255 caracteres.
6. Los espacios en blanco o columnas deben separar las ***etiquetas*** de los ***mnemónicos*** y a su vez a los ***mnemónicos*** de los ***operandos***. Una lista de múltiples ***operandos*** debe estar separada por comas.

Los espacios en blanco son uno o más espacios o tabulaciones. Son usados para hacer el ***código fuente*** más legible, pues separan y ordenan los campos de una línea.

2.1.2 Compiladores de lenguaje ensamblador de Microchip

El *software*[2] que se utiliza para traducir el ***código fuente*** del lenguaje ensamblador a ***código máquina*** se denomina ***compilador de ensamblador*** o simplemente ***ensamblador***[3].

Existen dos formas de generar ***código máquina*** para microcontrolares:

- Generar ***código absoluto*****,** que puede ser ejecutado directamente por un microcontrolador.
- Generar ***código relocalizable***, que puede ser enlazado con otros módulos de código ensamblado o precompilado para luego generar código de máquina absoluto.

Código de máquina absoluto

El ***código de máquina absoluto*** se genera cuando el ***ensamblador*** crea ***código máquina ejecutable*** para un microcontrolador en particular. El ***código de máquina absoluto*** es la salida estándar del compilador MPASM *Assembler* y el resultado del proceso de ensamblado se guarda en un archivo con extensión .hex (figura 2.1). Este archivo puede ser directamente descargado a la memoria de un microcontrolador a través de un dispositivo programador o depurado con un emulador en circuito o un simulador en *software*.

Código de máquina relocalizable

El compilador MPASM tiene la capacidad de generar módulos de ***código máquina relocalizable***. Estos módulos no son código que pueda ser ejecutado directamente por un microcontrolador en particular, en su lugar, puede ser unido con otros módulos a través del enlazador MPLINK de Microchip, para formar el ***código máquina ejecutable***.

[2] Debido a la ambigüedad que se presenta al traducir al español los tecnicismos *software* y *program* se utiliza el término en inglés para dar el sentido correcto al párrafo.
[3] Del termino *assembler* en inglés.

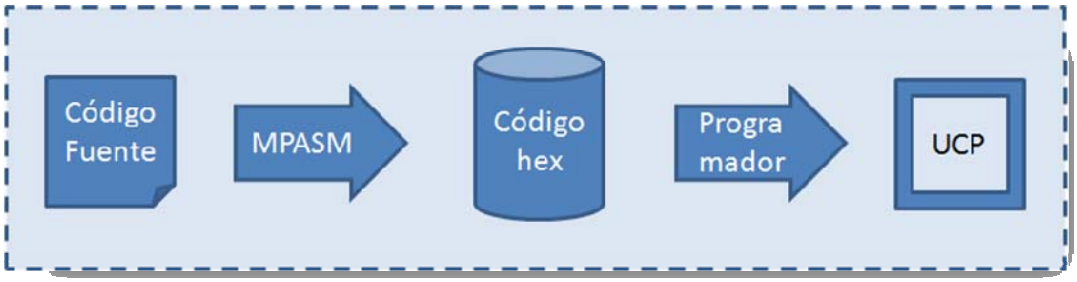


Figura 2.1 Proceso de ensamblado de un archivo de código fuente para generar código de máquina absoluto.

La figura 2.2 muestra como dos archivos de código fuente son compilados como unidades independientes de un mismo proyecto para generar archivos de ***código intermedio relocalizable*** (archivos de código objeto). Estos archivos son la entrada al enlazador MPLINK que será el encargado de generar el ***código maquina absoluto*** para un microcontrolador en particular. Este método es muy útil para crear módulos de código reutilizable precompilado.

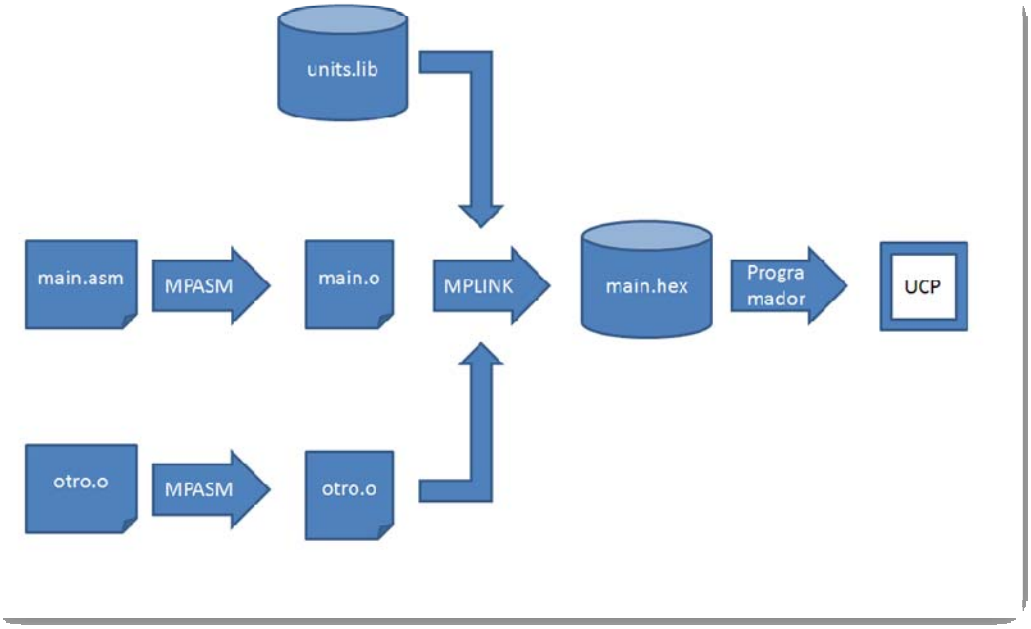


Figura 2.2 Creación de código ejecutable a través de múltiples archivos de código fuente.

Módulos relacionados pueden ser agrupados y almacenados juntos en un archivo o biblioteca usando el enlazador de bibliotecas MPLIB de Microchip. Las bibliotecas requeridas pueden ser especificadas en el tiempo de enlace y sólo las rutinas que son necesarias serán incluidas en el ***código máquina ejecutable*** final (figura 2.3).

Microchip MPASM *Toolsuite*

El compilador MPASM Toolsuite es un conjunto de programas encargados de compilar el código fuente en lenguaje ensamblador a código máquina para cada microcontroladores PIC de las familias PIC10/12/16/17/18 de 8 bits.

Entre las herramientas que forman el MPASM *Toolsuite* se encuentran MPASM *Assembler* (*mpasmwin.exe*), el compilador de lenguaje ensamblador para microcontroladores de Microchip; MPLIB *Librarian* (*mplib.exe*), el enlazador de bibliotecas; y MPLINK *Object Linker* (*mplink.exe*), el programa enlazador del código objeto (figura 2.4).

El programa MPASM tiene una interfaz gráfica en ambiente Windows, que permite configurar las opciones del compilador como son: el tipo de procesador para el que se generará ***código máquina***, el formato del archivo de salida, si se generarán los archivos intermedios, el archivo de código fuente a compilar, entre otras opciones.

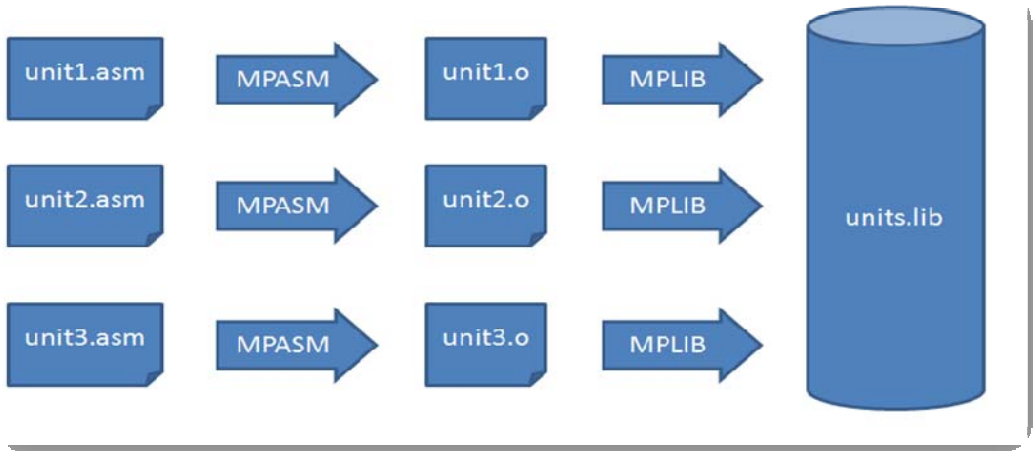


Figura 2.3 Creación de un archivo biblioteca a partir de archivos de código objeto precompilado.

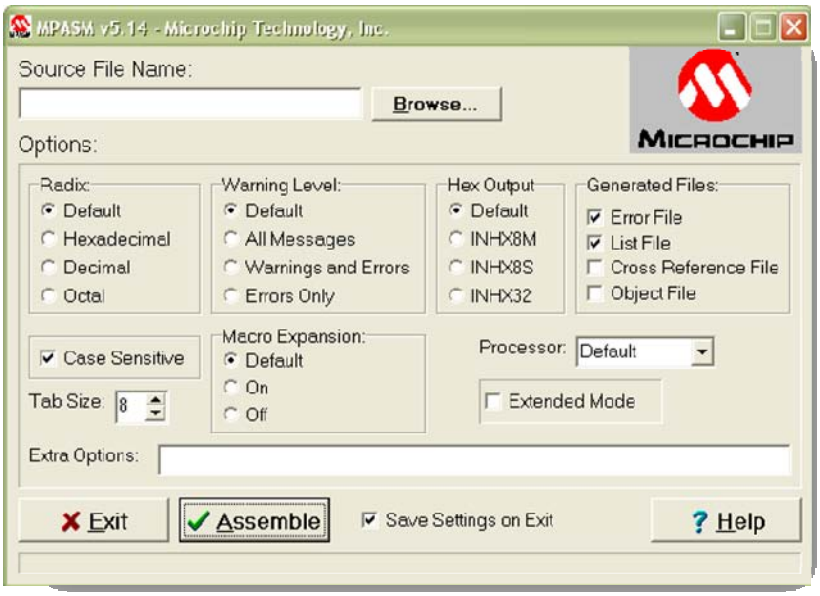


Figura 2.4 Pantalla del compilador MPASM WIN

Este programa realiza las llamadas al enlazador de bibliotecas y al enlazador de código objeto, para obtener el archivo de ***código máquina*** (archivo con extensión .hex) sin necesidad de hacer las llamadas a estos programas por separado, como se acostumbraba hace algunos años atrás en ambientes de consola MS-DOS.

Microchip MPASM30 Toolsuite

El compilador MPASM30 *Toolsuite* es un conjunto de programas encargados de compilar el código fuente en lenguaje ensamblador a ***código máquina*** para los microcontroladores PIC24 y dsPIC, con arquitectura de 16 bits. La suite está formada por el compilador MPLAB ASM30 *Assembler* y el enlazador MPLAB LINK30.

Microchip MPASM32 Toolsuite

El compilador MPASM32 *Toolsuite* es un conjunto de programas encargados de compilar el código fuente en lenguaje ensamblador a ***código máquina*** para los nuevos microcontroladores dsPIC, con arquitectura de 32 bits. La suite está formada por el compilador MPLAB ASM32 *Assembler* y el enlazador MPLAB LINK32. Es la herramienta más nueva de Microchip lanzada el año pasado.

2.1.3 Archivos generados por el MPASM *Assembler*

Después de compilar el código fuente de ensamblador por el MPASM se generan los archivos mostrados en la figura 2.5, si así está configurado el programa ensamblador.

Archivos de listado

Este archivo que posee la extensión .lst. Es un archivo de texto ASCII que puede ser revisado por cualquier editor de texto, su contenido muestra información sobre el análisis seguido del código fuente durante el proceso de ensamblado para generar el ***código máquina ejecutable***.

Entre la información que muestra podemos observar, la numeración de cada línea de código fuente, su respectivo ***código máquina*** con la dirección asignada en la memoria de programa, los errores generados en el proceso de compilación, entre otra información. Como ejemplo de un archivo de listado de ensamblado podemos ver la tabla de código 2.2.

Archivo de errores

Es un archivo con la extensión .err. Contiene los errores producidos en el proceso de compilación, mismo que fueron mostrados en el archivo de listado. La tabla de código 2.3 muestra un ejemplo de un archivo de errores después de realizar la compilación del código de la tabla 2.5.

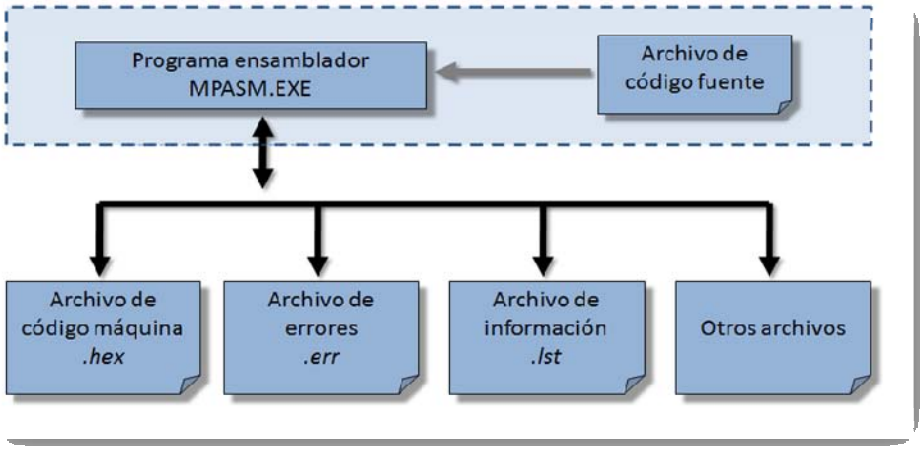


Figura 2.5 Archivos generados por el MPASM.

Tabla de Código 2.2. Archivo de listado *ejemplo.lst* generado tras la compilación del código fuente de la tabla de código 2.5.

```
MPASM  5.12                    EJEMPLO.ASM   9-20-2008  18:55:26         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; ********************* VARIABLES **********************
                     00002
  00000000           00003 W         EQU     0       ; Destino: W
  00000001           00004 F         EQU     1       ; Destino: registro
  00000005           00005 RP0       EQU     05h     ; Bit 5 registro STATUS
                     00006
                     00007 ; ************** Igualdades de la TCP ***************
                     00008 ; ********** Definición del mapa de memoria ************
                     00009
  00000006           00010 PORTB           EQU     06h     ; Puerto B
  00000086           00011 TRISB           EQU     86h     ; Registro uni-estado: PortB
  00000003           00012 STATUS          EQU     03h     ; Registro STATUS
  00000020           00013 Contar1         EQU     20h     ; Registro del primer contador
  00000021           00014 Contar2         EQU     21h     ; Registro del Segundo contador
                     00015
                     00016 ; ********** Sección Código de Reset **********
                     00017
                     00018         ORG     00h             ; Dirección del Vector de reset
0000   2???          00019         GOTO    INICIALIZA      ; Coloca el programa una
                     00020                                 ; posición después del Vector de reset
                     00021         ORG 05h                 ; Escribir el programa una posición
                     00022                                 ; después del Vector de interrupción
                     00023
                     00024 ; ********** Sección Inicialiva **********
                     00025
0005                 00026 INICIALIZA
0005   1683          00027         BSF     STATUS, RP0     ; Seleccionar el banco 1 dela memoria
Mensaje[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0006   0186          00028          CLRF TRISB             ; programar el Puerto B como salida
0007   1283          00029         BCF STATUS, RP0         ; Regresar al Banco 0 de la memoria
                     00030
0008   0186          00031         CLRF PORTB              ; Apagar todas las leds
                     00032
0009   01A0          00033         CLRF Contar1            ; Borra el registro contador .
000A   01A1          00034         CLRF Contar2            ; Borra el registro contador 2
000B   3001          00035         MOVLW 01h               ; Cargar el registro de trabajo con 01h
000C   0086          00036         MOVWF PORTB             ; Prender el primer led.
                     00037 ; ********** Sección Principal **********
                     00038
000D                 00039 PRINCIPAL
000D   0D86          00040         RLF     PORTB, F        ; Hace un corrimiento a la izquierda
000E   2???          00041         CALL    RETARDO         ; Espera 1 Seg
000F   2???          00042         GOTO    PRINCIPAL       ; Ciclo Infinito
```

Tabla de Código 2.2. Archivo de listado *ejemplo.lst* generado tras la compilación del código fuente de la tabla de código 2.5. (Continuación)

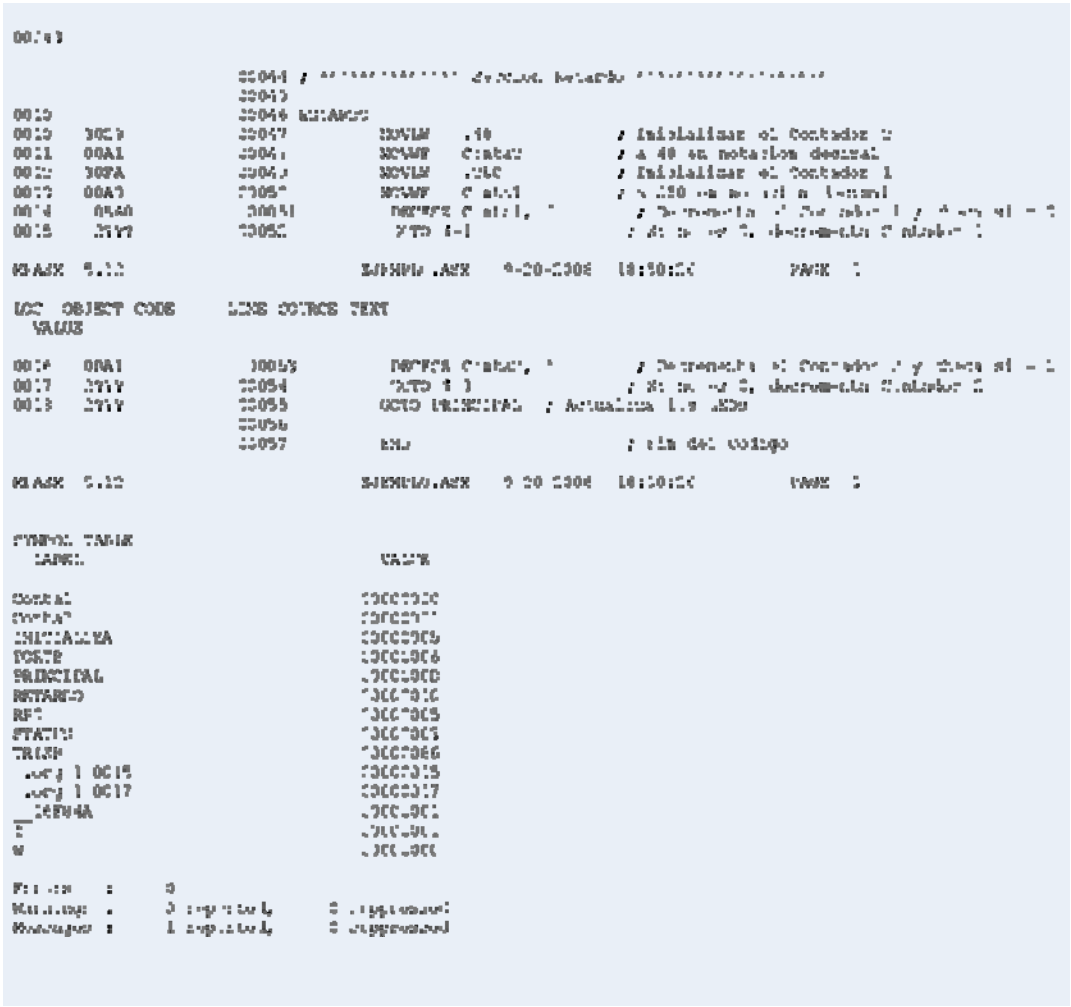
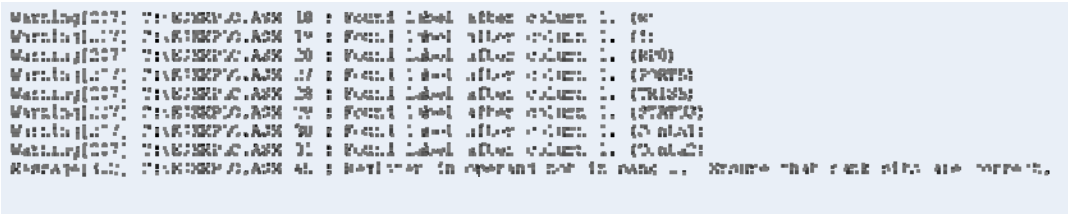


Tabla de Código 2.3. Archivo de error *ejemplo.err* generado tras la compilación del código fuente de la tabla de código 2.5



Archivo de código ejecutable

Es un archivo que dependiendo del formato en que se codifique su contenido tendrá la extensión .hex, .hlx o .hxh, por lo general, la salida estándar es con la extensión .hex (de hexadecimal) haciendo referencia al formato hexadecimal Intel de 8 bits.

Al igual que los archivos de listado y de error se trata de un archivo de texto ASCII, generado por el MPASM *Assembler* y el MPLINK *linker*. Los formatos de código ejecutable que el compilador es capaz de generar son: *Intel Hex Format* INH8M (.hex), para programadores con núcleo de 8 bits; *Intel Split Hex Format* INH8S (.hxl*,* .hxh), para programadores *odd/even*; *Intel Hex 32 Format INHX32*, para programadores de núcleo de 16 bits. La tabla de código 2.4 muestra un ejemplo del aspecto de un archivo de código en formato *Intel Hex 32 Format INHX32.*

Tabla de Código 2.4 Archivo de código ejecutable *ejemplo.hex* generado tras la compilación del código fuente de la tabla de código 2.5.

```
:020000040000FA
:020000000528D1
:06000A008316860183123B
:100010008601A001A10101308600860D10200D2867
:100020002830A100FA30A000A00B1428A10B14283E
:020030000D2899
:00000001FF
```

El formato INH8M genera un archivo hexadecimal de 8 bits con la combinación de un byte bajo y uno alto. Ya que cada dirección sólo puede contener 8 bits en este formato, todas las direcciones usan dos bytes para ser representadas. Cada línea comienza con un prefijo de 9 caracteres y termina con dos caracteres de suma de comprobación bajo el siguiente formato:

:BBAAAATTHHHH....HHHHCC

Donde:

BB: Son dos dígitos en hexadecimal que representan el número de bytes de datos por línea.

AAAA: son 4 dígitos que representan la dirección en hexadecimal donde inician los datos.

TT: Son dos dígitos que representan el tipo de línea, por lo general son 00 excepto la última que siempre será 01.

HH: Son dos dígitos que representan los 12, 14, o 16 bits de datos. Se constituyen con una combinación de byte bajo y alto, al igual que los 4 dígitos de dirección.

CC: Son dos dígitos que representan el complemento a dos de la suma de comprobación (*checksum*) de toda la línea.

2.1.4 Compiladores de lenguaje C de Microchip

Otra opción para escribir un programa para un microcontrolador es hacerlo en un ***lenguaje de alto nivel***, para luego usar un ***intérprete*** o un ***compilador*** que será el encargado de traducirlo a instrucciones en ***lenguaje máquina*** de un microcontrolador en particular. Un ***lenguaje de alto nivel*** es aquel que permite expresar un algoritmo de manera más adecuada a la capacidad cognitiva humana, en lugar de la capacidad ejecutora de los microcontroladores. Los ***compiladores*** para microcontroladores más difundidos están basados en lenguajes de programación como BASIC y C.

Hacer uso de un ***lenguaje de alto nivel*** tiene ventajas sobre el ***lenguaje ensamblador***. Una de ellas es la ***portabilidad de código***, es decir, podemos usar el mismo código fuente para generar ***código máquina*** para diferentes arquitecturas sólo con utilizar el ***compilador*** adecuado para tal arquitectura, lo que permite que el ***código fuente*** sea independiente de la arquitectura. Además, los programas son más fáciles de entender y depurar.

Un punto en contra del uso de ***compiladores*** de ***lenguajes de alto nivel*** está relacionado con los algoritmos que utilizan para generar el ***código máquina***, es decir, pueden generar código eficaz pero que utiliza más recursos de la arquitectura que si se hubiera diseñado el mismo programa directamente en ***lenguaje ensamblador***.

Generar código eficiente es una ardua tarea para los diseñadores de ***compiladores*** y es un tema muy importante cuando se trata de microcontroladores, ya que estos tienen recursos muy limitados comparados, por ejemplo, con el microprocesador de una computadora personal.

Al usar lenguaje ensamblador se trabaja directamente con la arquitectura del microcontrolador, a diferencia de cuando hacemos uso de un lenguaje de alto nivel. Ejemplo de esto es, en un lenguaje de alto nivel el compilador es el encargado de reservar la memoria de datos que se utilizará, lo que libera al usuario de tener que recordar que registro está en uso y cual está libre.

Otra ventaja es que en un lenguaje de alto nivel podemos hacer uso de los tipos y estructuras de datos de una manera más sencilla, pues el compilador es el encargado de generar los algoritmos necesarios para su manejo, sin importar si sus variables tienen más de un byte (8 bits) de longitud o la manera como debe ser interpretado el contenido de una palabra (tipo de dato), es decir si se trata de un número entero o de coma flotante. También el compilador es el encargado de administrar el uso de la memoria de datos.

Tabla de Código 2.5. Ejemplo de un archivo de código fuente en ensamblador.

```
; *************************************************
; *  UNAM FES Aragón                              *
; *  IGNACIO MENDOZA NUCAMENDI                    *
; *  Secuencia de LED'S                           *
; *  Revisión 1.0                                 *
; *  Programa para PIC16F877                      *
; *  Tipo de Reloj XT                             *
; *  Frecuencia de Reloj 8 MHz                    *
; *                                               *
; *************************************************
; *  Este programa prende una secuencia de leds,  *
; *  uno a la vez                                 *
; *************************************************
; ****************** IGUALDADES *******************

      w     EQU   0     ; Destino w
      f     EQU   1     ; Destino registro
      RP0   EQU   05h   ; Bit 5 registro STATUS

; ************** Igualdades de la UCP ***************
; ************** del mapa de memoria ****************

      PORTB       EQU   06h   ; Puerto B
      TRISB       EQU   86h   ; Registro tri-estado PortB
      STATUS      EQU   03h   ; Registro STATUS
      Conta1            EQU   20h   ; Registro del primer contador
      Conta2            EQU   21h   ; Registro del Segundo contador

; *********** Sección Código de Reset ***************

      ORG   00h         ; Dirección del Vector de reset
      GOTO  INICIALIZA  ; Comienza el programa una
                        ; posición después del vector de reset
      ORG 05h           ; Escribir el programa una posición
                        ; después del Vector de interrupción

; ************* Sección Inicializa ******************
INICIALIZA
      BSF   STATUS, RP0 ; Seleccionar el banco 1 de la memoria
       CLRF TRISB       ; programar el Puerto B como salida
      BCF STATUS, RP0   ; Regresar al Banco 0 de la memoria

      CLRF PORTB        ; Apagar todos los leds

      CLRF Conta1       ; Borra el registro contador 1
      CLRF Conta2       ; Borra el registro contador 2
      MOVLW 01h         ; Cargar el registro de trabajo con 01h
      MOVWF PORTB       ; Prender el primer led.

; *************** Sección Principal ***************
PRINCIPAL
      RLF   PORTB, f          ; Hace un corrimiento a la izquierda
      CALL  RETARDO           ; Espera 1 seg
      GOTO  PRINCIPAL         ; Ciclo infinito
```

Tabla de Código 2.5. Ejemplo de un archivo de código fuente en ensamblador. (Continuación).

```
; ************** Sección Retardo ******************

RETARDO
      MOVLW .40         ; Inicializar el Contador 2
      MOVWF Conta2            ; a 40
      MOVLW .250        ; Inicializar el Contador 1
      MOVWF Conta1            ; a 250
      DECFSZ Conta1, f  ; Decrementa el Contador 1 y checa si = 0
       GOTO $-1         ; Si no es 0, decrementa Contador 1
      DECFSZ Conta2, f  ; Decrementa el Contador 2 y checa si = 0
       GOTO $-3         ; Si no es 0, decrementa Contador 2
      GOTO PRINCIPAL    ; Actualiza los LEDs

      END               ; Fin del Código
```

Como ejemplo de lo comentado, podemos analizar dos códigos fuente que realizan la misma tarea. Uno en lenguaje ensamblador para el compilador MPASM (tabla de código 2.5) y otro en lenguaje C para el Compilador MPLAB C18 (tabla de código 2.6), ambos compiladores de Microchip Technology Inc. La primera diferencia que podemos notar entre cada uno de estos códigos fuente es el número de líneas utilizadas para implementar cada algoritmo. En lenguaje C, el programa es más fácil de leer e interpretar que en lenguaje ensamblador, pero no hay que perder de vista que una vez compilado, el ***código máquina*** resultante puede ser más extenso que el generado por el ***ensamblador***.

Compiladores de lenguaje C

Los microcontroladores PIC de 8 bits no son el único producto de Microchip. De igual forma fabrica microcontroladores de propósito general (MCU) y para el procesamiento digital de señales (DSP) de 16 y 32 bits. Programar estas nuevas arquitecturas en lenguaje ensamblador supone un reto mayor que hacerlo en un lenguaje de alto nivel.

El lenguaje C es el lenguaje de alto nivel elegido por Microchip para desarrollar aplicaciones de una manera más amigable, en un menor tiempo y de manera eficiente. Microchip ofrece diferentes compiladores para generar ***código máquina*** para programas escritos en este lenguaje. Se conservan las reglas de sintaxis, las palabras reservadas, el manejo de estructuras de datos, entre otras características propias del lenguaje, pero se enriquece dotándolo de bibliotecas de funciones dedicadas al manejo de los recursos de la arquitectura de los microcontroladores PIC y dsPIC.

Tabla de Código 2.6. Ejemplo de un programa en un lenguaje de alto nivel.

```
// ************************************************
// *  UNAM FES Aragón                             *
// *  IGNACIO MENDOZA NUCAMENDI                   *
// *  Secuencia de LED'S                          *
// *  Revisión 1.0                                *
// *  Programa para PIC16F877A                     *
// *  Tipo de Reloj XT                            *
// *  Frecuencia de Reloj 8 MHz                   *
// *                                              *
// ************************************************
// *  Este programa prende una secuencia de leds,  *
// *  uno a la vez                                *
// ************************************************


// ****************** IGUALDADES ********************

   void main(void)                    // Función principal
   {
         int8 iPuerto = 1;          // Declaración de la variable
         while(true)                 // Ciclo infinito
         {
               iPuerto<<iPuerto,1;   // Corrimiento y asignación
               out(PORTB, iPuerto) ; // Actualizar puerto B
               Delay(1000);          // Retardo de 1 seg
         }
   }
```

Para cada una de las familias de microcontroladores de 8, 16 y 32 bits, fue creado un compilador que optimiza el código máquina que genera, para aprovechar al máximo los recursos de cada dispositivo. En el caso de los microcontroladores de 8 bits, fue desarrollado el compilador MPLAB C18 *Toolsuite*, capaz de generar código para los microcontroladores de la familia PIC18 o de la gama mejorada; para la familia de controladores de 16 bits PIC24 y dsPIC30 el compilador MPLAB C30 *Toolsuite*; y para la nueva familia de microcontroladores de 32 bits el compilador MPLAB C32 *Toolsuite*. Es posible obtener una versión de evaluación desde el sitio de Internet de Microchip.

Los microcontroladores de 8 bits de las gamas media y baja no son soportados por el compilador C18, en su lugar, es posible usar un compilador de otro fabricante de *software* como HI-TECH, IARm Byte Craft. B. Knudsen, CCS, microEngineering Labs, Segger, entre otras, que desarrollan compiladores en lenguajes C o BASIC para estas gamas de microcontroladores.

2.1.5 Simuladores

Los ***simuladores*** son un *software* que muestra en pantalla el estado de los recursos de un microcontrolador cuando se ejecutan las instrucciones del programa, por lo que durante la etapa de diseño no es necesario colocar al microcontrolador en el sistema de manera física para verificar como funciona una vez programado.

Un ***simulador*** mostrará paso a paso los cambios que se originen en los recursos del microcontrolador. Existen ***simuladores*** gráficos que muestran la interacción del microcontrolador con dispositivos periféricos e instrumentos, como diodos LED, motores u osciloscopios. Microchip ofrece el simulador MPLAB-SIM. Para hacer uso de este simulador es necesario utilizar el programa MPLAB IDE.

2.1.6 Ambiente de desarrollo integrado MPLAB IDE

MPLAB IDE (*Integrated Development Environment*) es un Ambiente de Desarrollo Integrado (figura 2.6), es decir, un conjunto de herramientas para el desarrollo de aplicaciones embebidas empleando microcontroladores PIC y microcontroladores dsPIC que Microchip distribuye de forma gratuita desde su sitio de Internet diseñada para ejecutarse en un ambiente Windows. Desde el mes de septiembre de 2007 está disponible la versión 8.0.

Sirve como interfaz de usuario a cada una de las herramientas de Microchip que se utilizan para el desarrollo y depuración de una solución; para administrar los archivos de proyecto, código fuente, código objeto; acceder fácilmente al compilador, simulador y otras muchas herramientas como las de grabación; también permite integrar algunas herramientas fabricadas por terceros como compiladores de lenguaje C.

El que esta herramienta sea distribuida por Microchip de forma gratuita es una de las razones que ha ayudado a que los microcontroladores PIC se hayan vuelto tan populares en el aprendizaje de microcontroladores de propósito general de 8 bits.

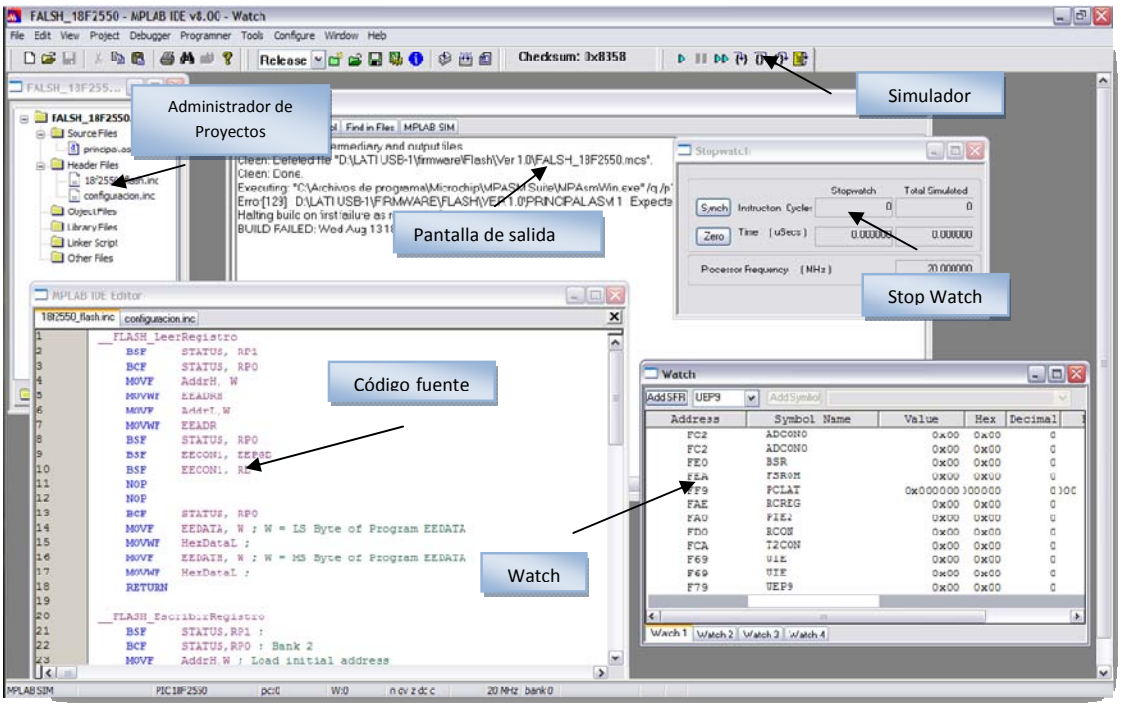


Figura 2.6. Pantalla del MPLAB IDE durante un proceso de simulación.

El que está herramienta sea distribuida por Microchip de forma gratuita es una de las razones que ha ayudado a que los microcontroladores PIC se hayan vuelto tan populares en el aprendizaje de microcontroladores de propósito general de 8 bits.

2.2 Herramientas en *hardware* de Microchip

Las herramientas en *hardware* que Microchip ofrece podemos dividirlas en cuatro grupos: ***programadores***, ***tarjetas entrenadoras,*** **emuladores** y ***depuradores en circuito***.

2.2.1 Programadores

Para descargar las instrucciones a la memoria del programa del microcontrolador es necesario un dispositivo llamado ***grabador*** o ***programador***. El dispositivo programador posee un zócalo para colocar el microcontrolador (figura 2.7). Se conecta a una computadora personal, por lo regular a través del puerto serie, paralelo o USB, donde se ejecuta el programa que se encargará del proceso de grabación de la memoria.

En el caso de microcontroladores con memoria ROM, el fabricante se encarga del proceso de grabación en el momento de fabricación del chip y no puede ser modificado posteriormente. Si la memoria es EPROM puede usarse un programador para grabar la memoria del dispositivo, pero será necesario un borrador de luz ultravioleta para borrarla. Los dispositivos con memoria EEPROM y FLASH pueden ser grabados y borrados en múltiples ocasiones en un programador.

a) **b)** **c)**

Figura 2.7 Diferentes modelos de programadores para productos de Microchip.
a)PICSTART plus b)PRO MATE II *Programmer* c) MPLAB PM 3 *Programmer*.

Los programadores de microcontroladores PIC soportan un protocolo diseñado por Microchip llamado protocolo ICSP® o de ***Programación Serie en Circuito***[4]. Este protocolo permite que los microcontroladores PIC sean programados directamente en el sistema embebido, es decir, dentro de la aplicación sin necesidad de tener que desmontar el circuito integrado para colocarlo en un zócalo del programador. En su lugar, el programador dispone de terminales donde se conecta un cable para tomar las señales que son usadas para la programación del microcontrolador. Para poder emplear esta técnica de programación, el sistema embebido debe disponer de unos pocos componentes como diodos, resistencias y un conector para el cable del programador. Un ejemplo que provee Microchip de cómo implementar la ***programación serie en circuito*** puede verse en la figura 2.8.

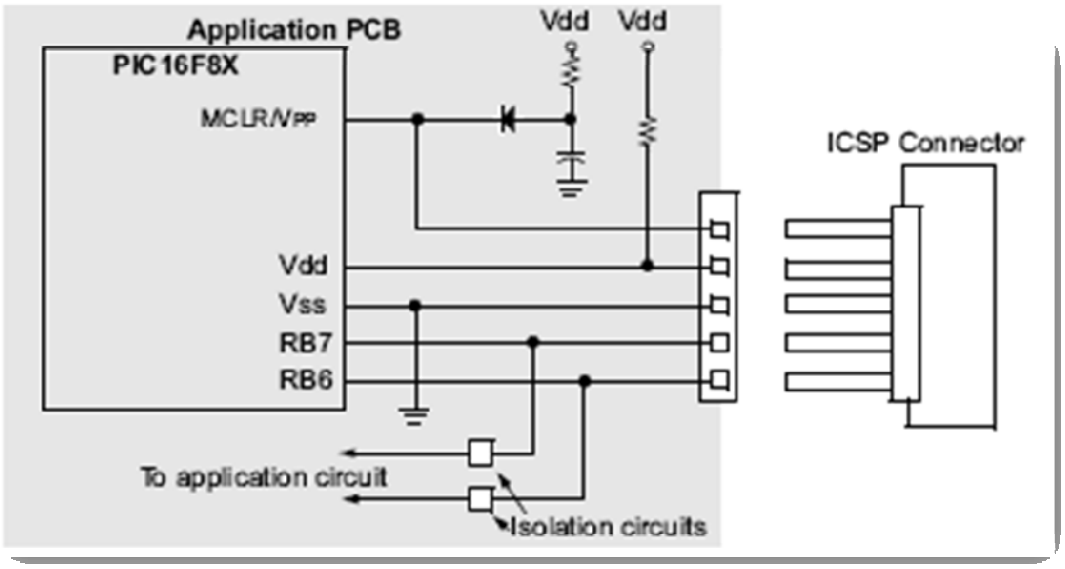


Figura 2.8. Circuito usado para la programación serie en circuito de un microcontrolador de la familia 16F8X.

[4] *In Circuit Serial Programming*

PIC START plus

PICSTART plus (figura 2.9) es un programador económico, diseñado por Microchip para el desarrollo de aplicaciones con microcontroladores PIC. Se conecta vía puerto serie RS-232 o USB de la PC y es operado con el programa MPLAB IDE que viene incluido de forma gratuita con el programador junto a todos los componentes necesarios como: cable serie, eliminador y guía de usuario en formato electrónico.

El programador soporta la mayoría de los microcontroladores de encapsulado DIP disponibles de Microchip y para programar un encapsulado diferente, es posible hacerlo en circuito, obteniendo los voltajes necesarios de un conector que se encuentra en el programador o a tevés de un adaptador. En la figura 2.9 puede observarse los componentes que integran al programador PIC START plus.

Figura 2.9. Programador PIC STRART plus de Microchip.

Otros programadores de Microchip son el MPLAB® PM 3 Programmer y ProMate II. Tiene la característica de que puede ser usado de forma independiente o junto a una computadora que ejecute MPLAB IDE, a diferencia del programador PIC START plus. Esta característica de trabajar sin necesidad de una PC, permite que el programador pueda ser llevado a la línea de producción y programar un gran número de microcontroladores de memoria FLASH con el mismo *firmware* a través de la *programación en sistema* que proporciona el protocolo ICSP®. El *firmware* del programador puede ser actualizado con cada nueva versión del MPLAB IDE.

2.2.2 Tarjetas entrenadoras

Las ***tarjetas entrenadoras*** son tarjetas de circuitos impresos que poseen una base para circuito integrado, donde se coloca el PIC, y diferentes componentes eléctricos, electrónicos y mecánicos conectados a las terminales del zócalo. Su objetivo es ahorrar tiempo en el ensamblado de circuitos típicos durante la etapa de prueba de un programa, como los reguladores de voltaje de 5V, arreglos de LEDs[5] o teclados, pues estos circuitos ya están implementados dentro de la misma tarjeta.

Por lo general, una ***tarjeta entrenadora*** está diseñada para un modelo de microcontrolador en particular, pero dependiendo del número de componentes que posea la tarjeta entrenadora y los recursos con que cuenta el microcontrolador puede usarse una tarjeta para una familia de microcontroladores (figura 2.10).

Las terminales de los microcontroladores PIC suelen estar multiplexadas a diferentes recursos o periféricos, por tanto éstas pueden ser configuradas como puertos de entrada y salida digitales, o ser entradas al comparador analógico, puertos de comunicación serie o estar asociadas a los módulos CCP para trabajar en cualquiera de sus configuraciones. Esta característica presenta un problema al diseñar una tarjeta entrenadora, ya que requerirá de un diseño más elaborado para proveer de los componentes necesarios para cada una de las configuraciones de los puertos, lo que se ve reflejado en el precio de la tarjeta.

[5] LED Light Emitter Diode - *Diodo emisor de luz.*

Figura 2.10. Tarjeta entrenadora PIC18F4xk20 Starter Kit.

Una opción para solucionar este problema es diseñar tarjetas entrenadoras que sean modulares, es decir que en la tarjeta principal posean los componentes básicos como: zócalo para el microcontrolador, circuito de alimentación, circuito de reset, oscilador, conectores para otros módulos y opcionalmente diodos LED e interruptores o botones; para que en tarjetas secundarias se armen otros circuitos como teclados matriciales, sistemas para el control de motores, circuitos de comunicación serie o USB y puertos analógicos.

Las tarjetas entrenadoras también son usadas en el proceso de enseñanza del uso de microcontroladores, brindando la misma virtud que ofrece cuando son usadas en la etapa de prueba al diseñar una aplicación, es decir liberan al diseñador de tener que armar un primer prototipo para poder probar su programa físicamente.

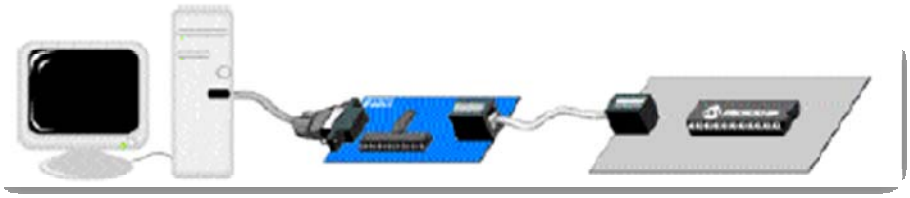
2.2.3 Depuradores y emuladores en circuito

Los emuladores en circuito (*In-circuit Emulator*) y depuradores en circuito (*In-circuit Debuger*) son herramientas de desarrollo muy similares. En ambos casos se trata de tarjetas que se conectan a un equipo de cómputo con una interfaz en *software* encargada de la comunicación entre la PC la tarjeta. A la vez, el ICD o ICE se conecta mediante un conector al sistema embebido donde reside el microcontrolador PIC.

El *software* en la PC iniciará al dispositivo del sistema embebido en un modo especial de operación (modo de depuración) que permitirá ejecutar el código fuente de manera similar a como se hace en un simulador. La principal diferencia con un simulador en *software* es que al emplear un ICD no sólo es posible conocer el estado interno del microcontrolador, sino que puede apreciarse de manera física y en tiempo real las condiciones y funcionalidad de un dispositivo determinado. La figura 2.11 muestra el diagrama para conectar un ICD o ICE.

Los depuradores en circuito permiten ejecutar, detener y reanudar un programa sin la necesidad de grabar la memoria de programa del dispositivo embebido. Para establecer la comunicación, entre el ICD y el microcontrolador embebido se hace uso de las mismas líneas que para la ***programación serie en circuito***. Los emuladores en circuito permiten imitar el comportamiento de diferentes dispositivos que pertenecen a una misma familia de microcontroladores.

Microchip ofrece al menos 4 herramientas de este tipo que permiten la eliminación de fallos (depuración) de la mayoría de sus microcontroladores PIC y dsPIC, estos son MPLAB® REAL ICE, MPLAB ICD 2 y MPLAB PICkit 2.



PC ICD/ICE Sistema Embebido

Figura 2.11. Diagrama de conexión de un ICE y ICD.

MPLAB® REAL ICE® *In-Circuit Emulator*

El MPLAB® REAL ICE® ICE es un emulador de alta velocidad para dispositivos con memoria FLASH. Su control se realiza mediante una interfaz en MPLAB IDE y su conexión al PC a través del puerto USB 2.0. Puede ser empleado junto con otros equipos como el MPLAB ICD 2 a través de un conector RJ11. El *firmware* del dispositivo puede ser actualizado a través del puerto USB del PC desde la interfaz MPLAB IDE para dar soporte a nuevos dispositivos. La figura 2.12 muestra el MPLAB® REAL ICE®.

Figura 2.12. Sistema MPLAB® Real ICE®. Un emulador en circuito de Microchip.

MPLAB® ICE 2000 In-Circuit Emulator

El MPLAB® ICD 2000 es un sistema emulador con una velocidad de 25MHz (*full speed*), un voltaje de operación bajo, 32K por 128 bit trace y 65535 puntos de alto (*breakpoints*). Es posible cambiar el procesador para configurarlo y emulador un procesador diferente. Es posible capturar en tiempo real las condiciones del estado del procesador y sus periféricos. Este dispositivo posee su propia interfaz y se conecta mediante cable paralelo al PC.

Figura. 2.13 Sistema MPLAB® ICE 2000 ICE.

MPLAB® ICD 2® In-Circuit Debugger

Es un dispositivo que permite la depuración de programas en tiempo real y programar algunos microcontroladores PIC y dsPIC. El programa es descargado, ejecutado entiempo real y examinado con las funciones de depuración del MPLAB IDE. Puede ejecutarse el programa en tiempo real o realizarse una ejecución paso a paso (*single-stepped*). Durante este proceso se ocupan algunos registros de propósito específico del procesador del microcontrolador, un nivel de la pila y las mismas dos líneas usadas para la transmisión de datos y reloj por el protocolo ICSP™. Además, se puede examinar el estado de variables, leer y escribir en la memoria FLASH y colocarse puntos de alto (*breakpoints*) en la ejecución del programa que detengan su ejecución durante la depuración, como ocurre en un simulador por *software*. La figura 2.14 muestra el MPLAB® ICD 2® *In-Circuit Debugger*.

Figura 2.14. Sistema MPLAB® ICD 2.

PICkit® 2 Programmer/Debugger

El PICkit® 2 *Programmer/Debbuger* posee su propia interfaz, a diferencia del resto de los dispositivos depuradores se conectan al PC a través del puerto UBS y soporta la mayoría de microcontroladores PIC y dsPIC de Microchip para su programación y, en el caso de los dispositivos que implementan ICD, también soporta a la mayoría. La figura 2.15 muestra es sistema de desarrollo PICkit® 2 *Programmer/Debbuger* junto a un sistema embebido.

Figura 2.15. Sistema PicKit® 2 ICD *Programmer*.

2.3 Otras herramientas de Microchip para el desarrollo de aplicaciones

2.3.1 Application Maestro *Software*

El programa Microchip *Application Maestro™ Software* es una herramienta en *software* independiente, es decir no pertenece al entorno de desarrollo MPLA® IDE, que incorpora módulos de *firmware* prescritos para la mayoría de dispositivos PIC. Su utilidad radica en no tener que reescribir segmentos de código que son de uso común, como la configuración de una UART, manejo de interrupciones o temporizadores. Basta con seleccionar los módulos a agregar y configurar algunos parámetros dentro de la interfaz para que el programa entregue una serie de archivos con el código fuente necesario que podrá ser agregado a nuestra aplicación, ya sea en lenguaje ensamblador o en lenguaje C, con lo que se reduce el costo y el tener que reescribir una sección de código de uso frecuente. En la figura 2.16 puede apreciarse la pantalla principal de la aplicación.

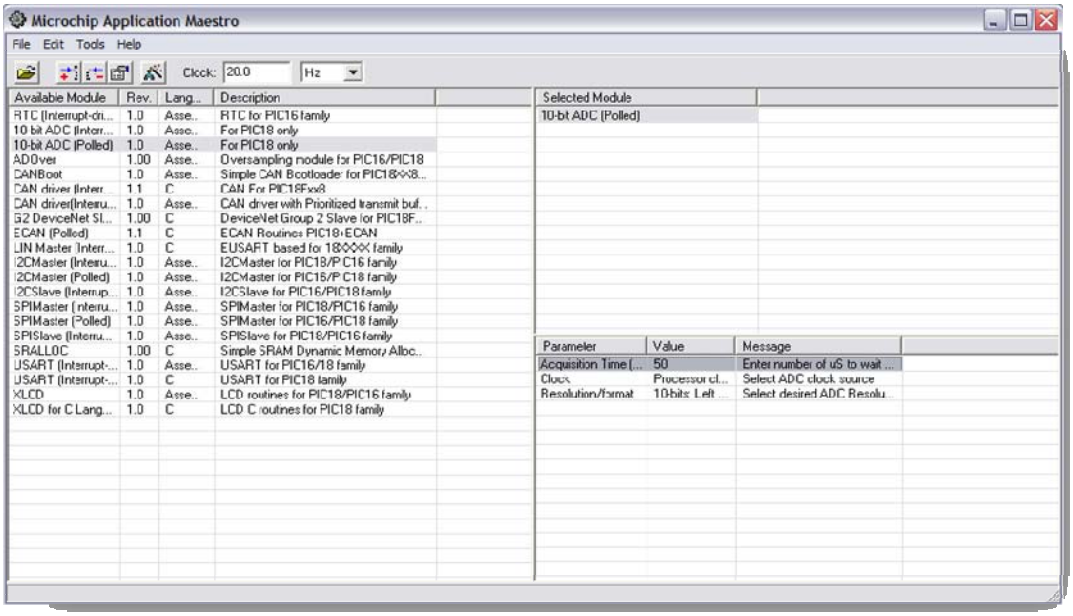


Figura 2.16 Pantalla principal del sistema *Application Maestro*.

2.3.2 Code Module Library

El programa *Code Module Library* es una aplicación gratuita provista por Microchip, de la cual se puede obtener una copia desde el sitio de Internet del fabricante. Es usado para administrar segmentos de código reutilizable, los cuales guarda dentro de carpetas bajo un sistema jerárquico para tener un acceso fácil y rápido.

Al igual que *Application Maestro* es una herramienta fácil de usar, independiente y trabaja junto a cualquier editor de textos. Al contrario de la aplicación maestro, no genera archivos que puedan ser adjuntados a nuestro proyecto, en su lugar, el segmento de código es copiado en el portapapeles de Windows y pegado en el proyecto. La figura 2.17 muestra la pantalla principal de la aplicación.

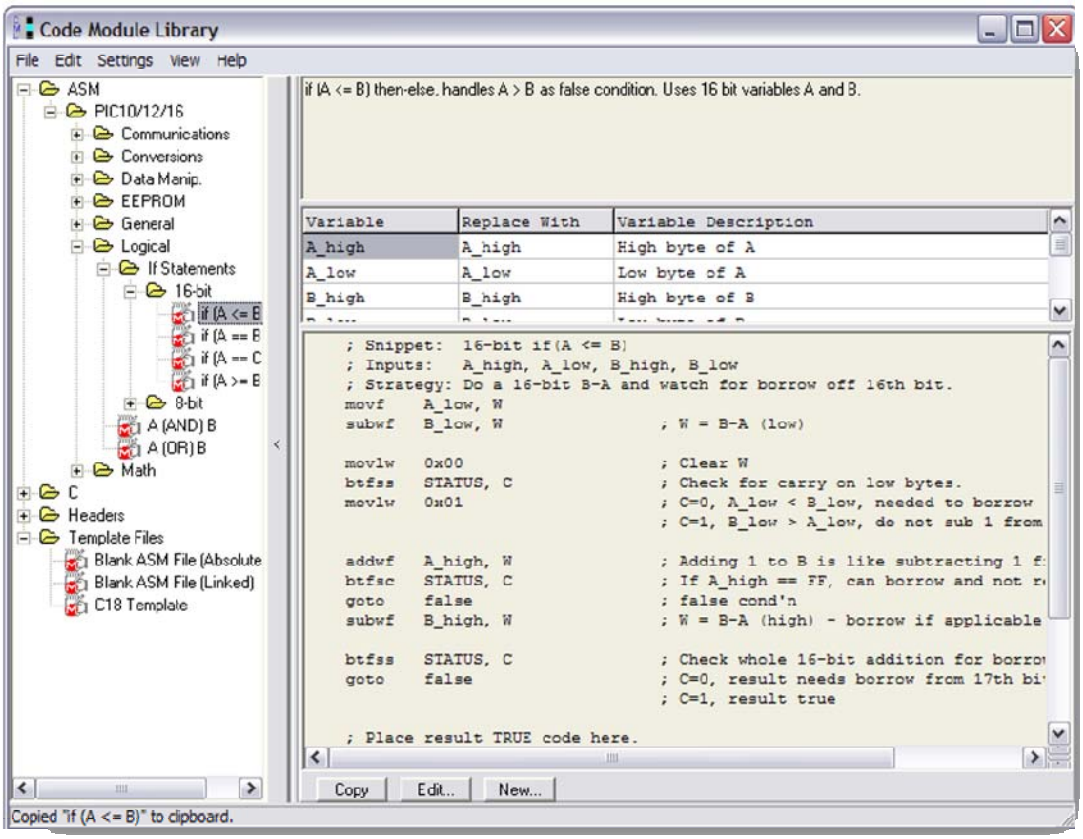


Figura 2.17. Pantalla principal de la aplicación *Code Module Library*.

La aplicación posee segmentos de código ya prediseñado, pero es posible agregar código propio y administrarlo desde la interfaz. Posee una herramienta para buscar y reemplazar, por ejemplo, el nombre de las variables o registros usados en esa sección de código.

2.4 Conclusiones del capítulo

Disponer de las herramientas para diseñar y desarrollar aplicaciones con microcontroladores PIC nos permitirá construir prototipos más eficientes. Dependiendo de la naturaleza del problema al que se quiere dar solución con un sistema embebido basado en un microcontrolador PIC, otro tipo de herramientas y equipo puede ser necesarias, como fuentes de poder de laboratorio, osciloscopios, generadores de funciones, entre otros, por lo que al dedicarse profesionalmente al desarrollo de aplicaciones, la adquisición de estas herramientas y equipo es más vista como una inversión que como un gasto. Por lo que considero que es rentable realizar una inversión por herramientas que tendrán garantía y soporte por parte del fabricante.

Ahora bien, cuando el desarrollo de aplicaciones con microcontroladores PIC no se realiza de forma profesional, es decir, como un pasatiempo o cuando se aprende en la universidad a desarrollar sistemas embebidos como parte de un curso curricular, es indispensable tener a la mano, por lo menos, un compilador y un programador para que el aprendizaje sea más significativo.

Capítulo 3

Herramientas de terceros para el desarrollo de aplicaciones con microcontroladores PIC

Capítulo 3
Herramientas de terceros para el desarrollo de aplicaciones con microcontroladores PIC

Desde 1992, Microchip ofrece abundante información técnica sobre el protocolo ICSP, un protocolo de comunicación serie síncrona usado para programar los microcontroladores PIC, así como ejemplos que explican cómo implementarlo para aprovechar al máximo las características de la programación en sistema (ISP). Gracias a esto, numerosas personas y empresas han diseñado herramientas en hardware y software para el desarrollo de aplicaciones con microcontroladores PIC, con o sin ánimo de lucro.

Una de estas organizaciones es GNUPIC, que basada en las ideas de Richard Stallmant que proponen que el software debe ser distribuido libremente, promueve el desarrollo de herramientas en hardware y software para construir aplicaciones con microcontroladores PIC que puedan ser distribuidas libremente. La dirección en Internet de esta organización donde personas de todo el mundo publican los avances obtenidos en sus proyectos es http://www.gnupic.org/.

En el presente capítulo serán comentadas las características principales de algunas herramientas en hardware y software para el desarrollo de aplicaciones con microcontroladores PIC, sin ahondar en detalles de su operación, con el fin de poder hacer un comparativo de prestaciones-beneficios cuando son usadas en la enseñanza del diseño y desarrollo de soluciones tecnológicas con microcontroladores PIC.

3.1 Herramientas en hardware

Diseñar herramientas en hardware para dar soporte al protocolo ICSP es relativamente sencillo. Sólo es necesario generar las señales adecuadas de acuerdo al protocolo ICSP y aplicarlas a las terminales del PIC especialmente diseñadas para tal fin. Las señales que intervienen en el proceso de programación de un microcontrolador PIC son: voltaje de programación (V_{PP}), voltaje de alimentación (V_{CC}), referencia a masa (V_{SS}), datos (PGD) y reloj (PGC).

3.1.1 Programadores (Programmers)

Debido a que el protocolo ICSP está diseñado para ser aprovechado en la programación en sistema (ISP), es decir, cuando el microcontrolador ya está montado en la tarjeta del sistema a controlar, no siempre es necesario el uso de zócalos o bases para circuito integrado dentro del mismo programador. En su lugar, las señales se toman del programador a través de un cable que se conecta a la tarjeta de la aplicación. Lo que permite que el programador sea muy pequeño y pueda programar un microcontrolador sin importar su tipo de encapsulado.

Cuando aprendemos a usar microcontroladores, uno de los encapsulados más fáciles de usar es el PDIP[1], ya que puede ser montano sin problemas en una tableta de prototipos o *protoboard* junto a los demás elementos de nuestro prototipo. Los programadores suelen incluir una base universal de inserción de cero esfuerzo (ZIF) donde podemos colocar el microcontrolador, como el de la figura 3.1c. Para otros tipos de encapsulados es posible usar adaptadores, como el de la figura 3.1d, o que el programador posea más de una base de diferentes tamaños, como los programadores de la figura 3.1a y b.

El programador se conecta a una PC a través de uno de sus puertos, por lo regular se hace uso del puerto serie o paralelo, pero de manera más reciente existen programadores con interfaz USB. Algunos de ellos necesitan de una fuente de alimentación externa y otros toman la corriente necesaria del mismo puerto de la computadora.

[1] PDIP Dual In-Line Package. Encapsulado de doble línea.

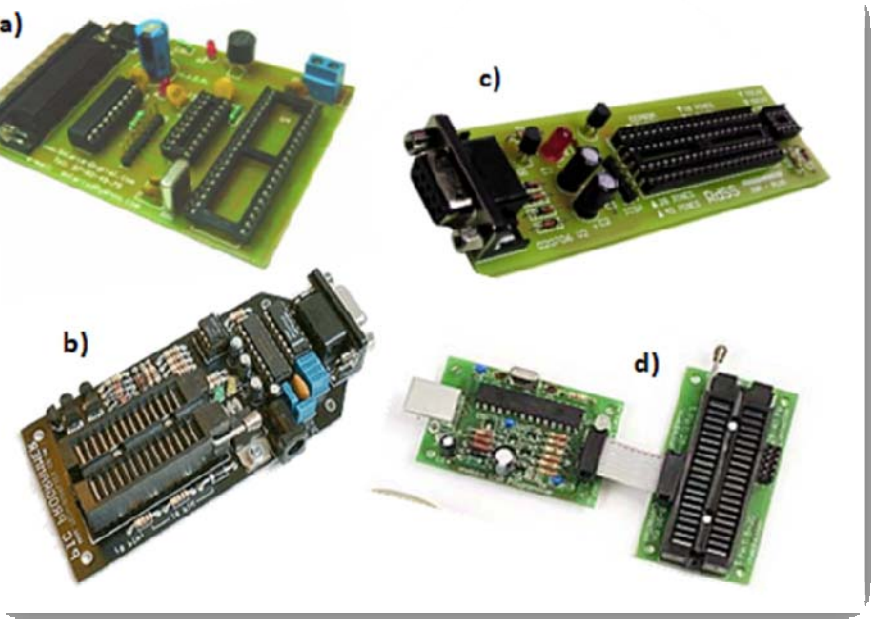


Figura 3.1. a) Programador por puerto paralelo. b) Programador por puerto serie y zócalo ZIF. C) Programador por puerto serie y múltiples zócalos. d) Programador USB con base de expansión ZIF.

De manera general existen dos técnicas para el desarrollo de un programador: el control del protocolo ICSP se implementa en el firmware de un microcontrolador o por software en un PC, ambos necesitan de una interface en hardware, para generar los voltajes requeridos para la programación.

Programador operado por software

En este caso, el protocolo ICSP es implementado en software en un lenguaje de programación, por ejemplo C, y se auxilia de algunos dispositivos eléctricos y electrónicos, como resistencia, capacitores y transistores, para la correcta obtención de la tensión de las señales. Este programa se encargará de la activación de las señales de programación y alimentación, generación de las señales de reloj y datos, así como del control del puerto que se emplee para conectar la interfaz en hardware (figura 3.2).

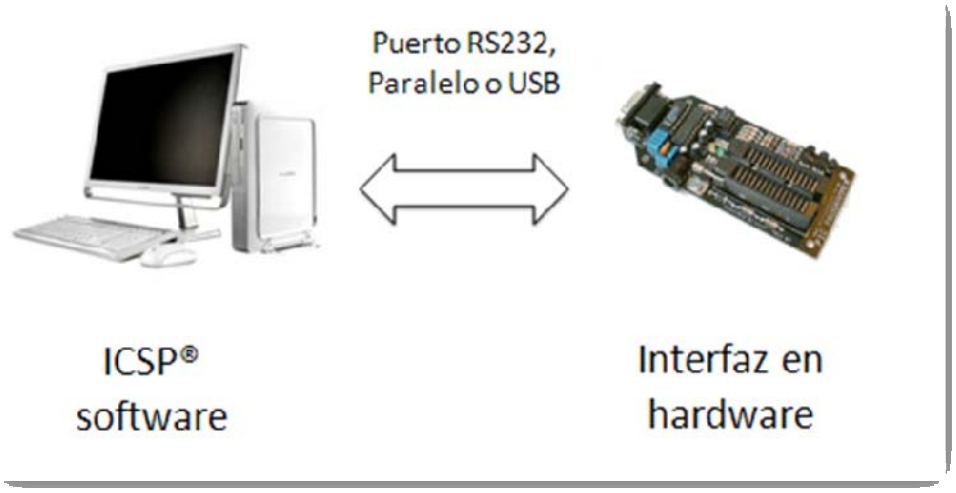


Figura 3.2. Diagrama de un programador que implementa el ICSP por *software.*

Programador operado por microcontrolador

Una alternativa un poco más compleja de construir que la anterior, pero más eficiente, es programar a un microcontrolador para que implemente el protocolo ICSP junto con el hardware necesario para suministrar los voltajes correctos. Este programador se auxilia de una interfaz en software, que permite controlar al programador, pero que no implementa en sí misma el protocolo ICSP, por el contrario, sólo se encarga de la comunicación entra la PC y el microcontrolador del programador, es decir, el firmware del microcontrolador del programador es el responsable de implementar el protocolo ICSP (figura 3.3).

Bajo estos dos principios existen diferentes opciones de programadores que podemos encontrar en la Internet. En esta sección serán comentaras tres de ellas por ser las más usadas entre estudiantes y aficionados a la electrónica, su bajo costo para ser construidas y porque su diseño es de libre distribución.

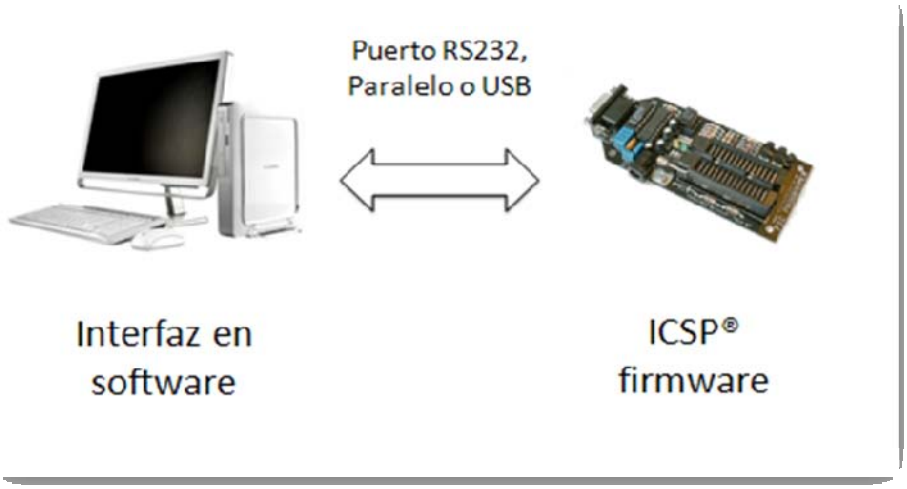


Figura 3.3. Diagrama de un programador que implementa el ICSP por *firmware*.

Programador JDM (Jens Dyekjær Madsen)

Fue desarrollado por Jens Dyekjær Madsen (JDM) entre 1996 y 1998. Han pasado ya más de 10 años desde la publicación y liberación de la primera versión de este programador a través de Internet a la comunidad de desarrolladores, sin embargo, hoy por hoy es una de las opciones más difundidas para programar microcontroladores PIC y memorias EEPROM serie. La página oficial del programador es http://www.jdm.homepage.dk/newpics.htm. Algunos otros modelos de programadores están basados en el diseño del programador JDM, como los TE-20 y TE-20SE que se recomiendan en el libro de PALACIOS **[1]** para iniciarse en la programación con microcontroladores PIC. Incluso el programador LATI PIC S1 (Figura 3.4) está basado en el programador JDM.

Figura 3.4. Programador LATI PIC S-1 basado en el programador JDM.

El programador JDM usa dispositivos eléctricos y electrónicos como transistores, diodos y resistencias, lo que hace posible construir un programador JDM con pocos recursos y un costo reducido para la mayoría de los estudiantes y aficionados a la electrónica. Además, no necesita de una fuente de alimentación externa ya que toma la corriente necesaria para la alimentación de los circuitos eléctricos del mismo puerto serie RS-232 al que se conecta. La figura 3.5 muestra el diagrama original del programador JDM publicado en su página del autor.

La implementación del control del protocolo ICSP se realiza mediante un software que se ejecuta en un equipo de computo como un PC o una laptop, existen numerosas interfaces en software con las que el programador JDM puede interactuar, pero destacan dos de ella que serán tratadas en una sección siguiente de este mismo capítulo, los programas IC-PROG y WINPIC 800.

El programador JDM hace uso del puerto serie de una PC, pero no implementa el protocolo RS-232 de este puerto, en su lugar ocupa las terminales del puerto serie para obtener los niveles respectivos de tensión para la alimentación y programación como se muestra en la Tabla 3.1.

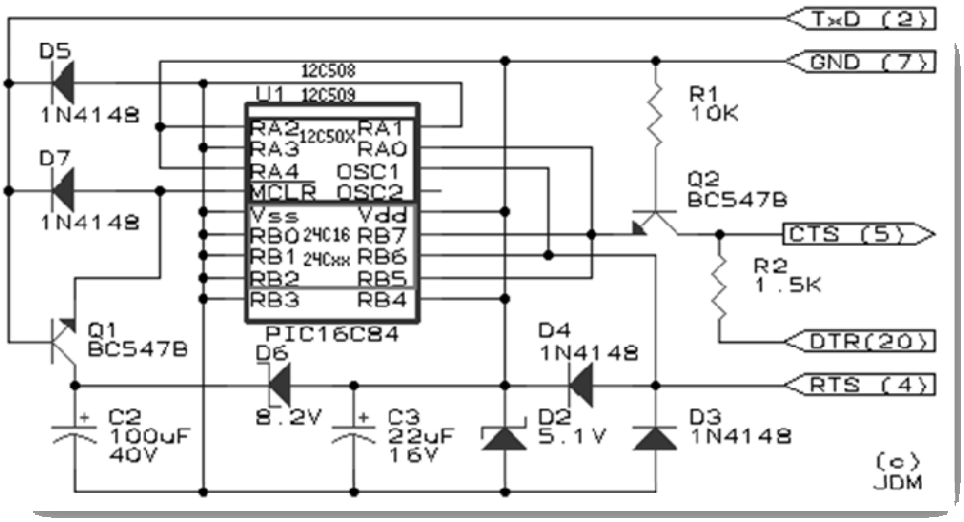


Figura 3.5. Diagrama de la versión 1.0 del programador JDM de microcontroladores PIC publicada en 1998 en la página del autor.

Analizando el diagrama eléctrico del programador JDM de la figura 3.5, podemos observar que los voltajes de programación V_{PP} (13V) y de alimentación V_{DD} (5V) se obtienen de la terminal TXD. La terminal del puerto serie TXD alimenta al capacitor C_2 para que alcance los 13V necesarios a través de la unión base-colector del transistor Q_1. El voltaje en C_2 está limitado por el diodo Zener de 8.2V (D6) y es aproximadamente el resultado de la suma del voltaje del C_2 y el C_1 limitados por los diodos Zener, es decir, 5.1V + 8.2V = 13.3V, por lo que, cuando TXD está a nivel alto el voltaje en MCLR no excede esta tensión lográndose el voltaje requerido para iniciar el estado de programación del microcontrolador. El transistor Q_1 activa y desactiva el voltaje de programación V_{PP} para la terminal MCLR.

El diodo D_7 es utilizado para poner la terminal MCLR a un nivel bajo cuando la terminal TXD este en nivel bajo, con un valor próximo a los 0.2V, con esto se garantiza que el microcontrolador estará en estado de RESET y no sea posible iniciar la ejecución del programa en el microcontrolador PIC aun cuando se encuentre polarizado por las señales V_{DD} y V_{SS}. El diodo D_5 es utilizado para limitar el voltaje en DTX y para alimentar la memoria EEPROM serie que se desea grabar cuando DTR y RTS están a nivel alto. En el caso de los microcontroladores PIC se usa para garantizar que MCLR sea casi 0V cuando TXD este a nivel bajo junto al diodo D_7.

Tabla 3.1. Terminales del puerto serie usadas en la programación de microcontroladores PIC por el programador JDM.

Terminal	Señal	Sentido PC →JDM	Uso en RS-232	Uso en JDM
3	TXD	→	Transmit Data Transmisión de datos	V_{DD}, V_{PP}
4	DTR	→	Data Terminal Ready Terminal de datos preparado	Salida serie
7	RTS	→	Request to Send Petición de envío	Reloj y Vss
8	CTS	←	Clear to Send Preparado para transmitir	Entrada serie

El voltaje de entrada para la señal de reloj en RB6 está limitada por los diodos 3 y 4 (D_3 y D_4 respectivamente). El diodo D4 sirve de igual manera para limitar la corriente que entra al microcontrolador por la terminal V_{DD}.

La resistencia R_2 es una resistencia *pull-up* que sirve para proteger al puerto. El transistor Q_2 también limita el voltaje de la entrada al PIC cuando DTR está en un nivel alto. Entonces funciona como un seguidor de emisor y se reduce el voltaje de la entrada de V_{DD} a aproximadamente 0.7V. El transistor Q_2 aumenta el voltaje de salida a niveles RS232 (próximos a 15V) en una configuración de base común. Cuando DTR se pone a nivel bajo, el transistor Q_2 trabaja invertido y siendo la ganancia aproximadamente 5. La resistencia equivalente en este arreglo del circuito es de 10K/5=2K. Este arreglo reduce la corriente de entrada de datos al PIC junto con la resistencia R_2.

Es importante tener en cuenta que al programar en sistema (ISP[2]) un microcontrolador, no debe usarse la alimentación eléctrica del circuito, ya que las señales de alimentación serán proporcionadas por el programador. En el capítulo 4, se comentará con más detalle los circuitos y arreglos que Microchip Inc. propone para implementar el protocolo ICSP en nuestras aplicaciones.

[2] ISP In-System Programming. Técnica donde de la programación del microcontrolador se realiza cuando este ya ha sido montado en la placa del sistema que controlará.

En cuanto a los inconvenientes que presenta este programador se encuentra que el puerto serie tiende a desaparecer en la mayoría de los equipos de cómputo, como en el caso de los equipos portátiles como laptops que ya no cuentan con puerto serie al ser reemplazado por el puerto USB. Otra característica es que no suele funcionar correctamente en algunos equipos, principalmente laptops ya que los puertos serie que utilizan son de bajo consumo de potencia, por lo que la corriente que proporcionan no siempre es la suficiente para poder alcanzar los voltajes de programación.

Programador TAIT

Otro programador que aparece en escena para el año de 1996 es el programador TAIT, diseñado por David Tait. Es posible encontrar el software original diseñado para la operación de este programador, así como el diagrama básico en la página de Internet http://people.man.ac.uk/~mbhstdj/piclinks.html. La figura 3.6 muestra el diagrama eléctrico del programador TAIT clásico diseñado en un principio para programar microcontroladores PIC16F84A.

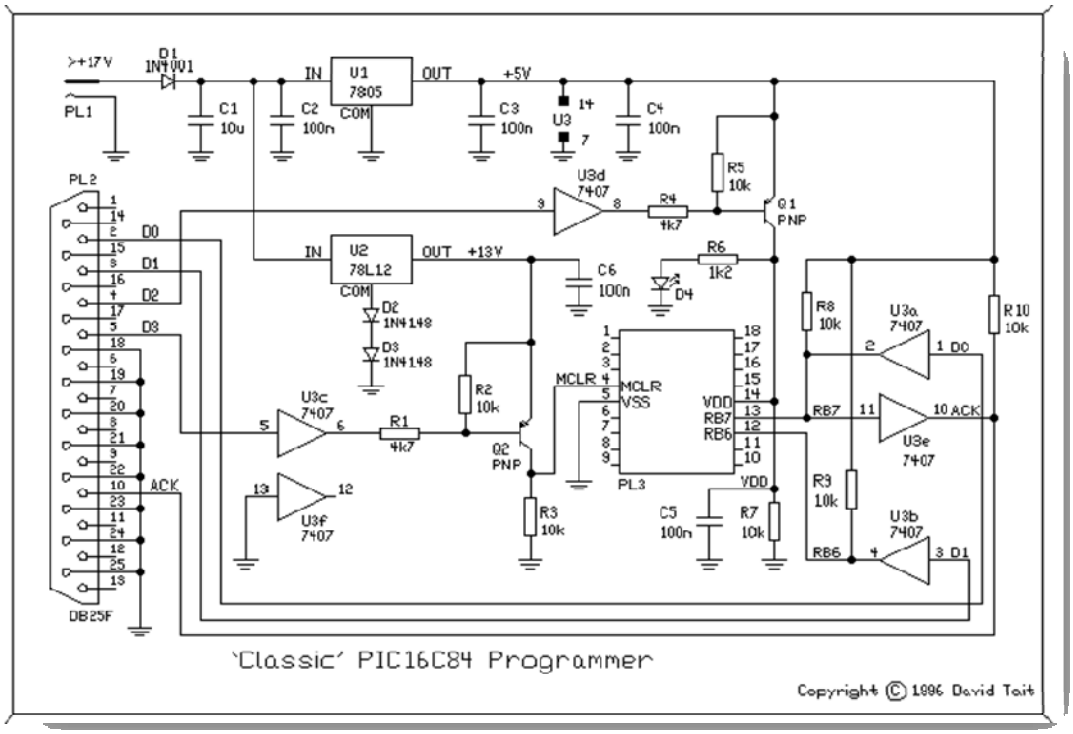


Figura 3.6. Diagrama de la versión 1.0 del programador Tait
de microcontroladores PIC publicada en 1996 en la página oficial del autor.

Como puede apreciarse en la figura 3.6, se trata de un programador que hace uso del puerto paralelo de la PC. A diferencia del programador JDM, el programador TAIT necesita de una fuente de alimentación externa, lo que puede aumentar el costo de los componentes para fabricar este programador comparado con el JDM.

La implementación del protocolo ICSP se realiza mediante un software que es ejecutado en un equipo de cómputo como una PC o una laptop. Para proteger el puerto paralelo de corrientes inversas o de consumos de corriente superiores a los 22mA, que son capaces de soportar como máximo algunas de sus terminales, se hace uso de un buffer con colector abierto o circuito integrado 74LS07.

Para habilitar los voltajes de alimentación V_{DD} (5V) y programación V_{PP} (13V) se utilizan los transistores Q_1 y Q_2 respectivamente, estos cambian de una región de corte a saturación dependiendo del valor lógico en sus bases. Estos valores pasan a través de un buffer colector abierto que proviene desde las terminales D_2 y D_3 del bus de datos del puerto paralelo para manipular el cambio entre las regiones de operación de los transistores y por tanto la activación de las tensiones de alimentación y programación.

Debido a que se trata de buffer con colector abierto, se utiliza un divisor de voltaje para polarizar la base de cada uno de los transistores. R_1 y R_2 son usadas para el transistor Q_1; R_4 y R_5 son usadas para el transistor Q_2.

Para generar la señal de sincronización o reloj, se hace uso de la terminal D_1 del bus de datos del puerto paralelo, de igual forma que con las demás terminales de este puerto, se coloca a su salida un buffer con colector abierto que limita la corriente que proporcionan o que son capaces de consumir. A la salida de este buffer se coloca una resistencia pull-up (R_8) de 10KΩ que es la encargada de proporcionar la corriente necesaria para generar una señal de 5V desde la fuente de alimentación y no directamente del puerto paralelo.

Un arreglo similar al circuito de la señal de reloj es usado para la señal de datos. En este caso, la señal de datos debe es bidireccional, es decir, en cierto momento de la terminal RB7 del microcontrolador PIC saldrán o entrarán de manera serie los bits de los comandos y datos utilizados para programar la memoria del microcontrolador.

Esto supone diseñar un arreglo más complejo que el usado en para la señal de reloj. Aunque en los modelos más recientes, el bus de datos del puerto paralelo es bidireccional, no puede ser usado bajo esta configuración, ya que no es posible configurar de manera independiente las terminales que forman en bus de datos pues todas las terminales del bus de datos cambian a un estado de entrada o salida. En su lugar se usan dos terminales, una del bus de datos configurada como salida (D_0) y una de bus de estado usada como entrada (ASK).

Entre las desventajas que podemos encontrar en este modelo de programador se puede mencionar la necesidad de una fuente de alimentación externa, ya que será necesario de disponer de un contacto a la corriente eléctrica para conectar dicha fuente de alimentación. Otro inconveniente es que el puerto paralelo tiende a desaparecer de equipos portátiles como laptops al ser reemplazado por puertos USB, aunque a diferencia del programador JDM, es posible hacer uso de un adaptador de puerto paralelo a USB para conectarlo a una laptop, solución que no es posible con el programador JDM al usar un convertidor de puerto serie a USB.

Las ventajas que se pueden enumerar de este grabador son: puede ser armado con un bajo presupuesto, su diseño es muy sencillo, por lo que puede ser armado directamente en una tableta de prototipos o *protoboard*, aunque es recomendable armar el circuito impreso o PCB (*Print Circuit Board*).

Programador GTP-USB

Todo PIC es un foro en la Internet que promueve entre sus participantes el uso y desarrollo con microcontroladores PIC y otros microcontroladores. En julio de 2003 uno de los moderadores de este foro, bajo el seudónimo de *marmatar*, publicó un tema con el título de *Programador GTP USB-232* el diagrama eléctrico de un programador por puerto USB que en honor al foro donde fue publicado lleva el nombre de programador GPT.

Este programador fue el primero de varios modelos que dieron pie al programador GTP-USB que actualmente se distribuye de manera comercial. Este foro se encuentra en la dirección de la Internet http://www.todopic.com.ar.

Este programador hace uso de un circuito transceptor de USB a RS-232 de la compañía FTDI para lograr la conversión de protocolos. Por lo que puede considerarse a este programador como un programador de puerto serie con un convertidor a puerto USB integrado en el circuito impreso. El control de protocolo ICSP se implementa en un microcontrolador PIC 18F252. Del programador GTP USB-232 sólo quedan los comentarios realizados en el foro ya que todos los diagramas, archivos necesarios para su construcción, así como el archivo HEX del firmware han sido retirados.

De manera paralela otro colaborador del foro, bajo el pseudónimo de *sispic* desarrolla un proyecto que titula *programador GTP USB Summer 2005*. En este caso, el control de protocolo ICSP se realiza con un microcontrolador PIC 18F2550 que incorpora un puerto USB, con lo que se ya no es necesario el uso de otro circuito integrado para realizar la conversión de puerto serie RS-232 a puerto USB. Al igual que el Programador GTP USB-232, la información y diagramas necesarios para su construcción ha sido retirada.

El último modelo de programadores para microcontroladores PIC por puerto USB diseñado por la comunidad de este foro en septiembre del 2003 es el programador GTP USB Lite, diseñado por un colaborador bajo el pseudónimo de J1M. Este programador está basado en el programador GTP USB Summer 2005 y hace uso del mismo firmware. De este programador se conservan en el foro los comentarios realizados sobre su funcionamiento y algunos diagramas del prototipo original, aunque el archivo hexadecimal del firmware para programar el microcontrolador PIC ha sido retirado pues este programador es hoy un producto comercial.

El proyecto puede dividirse en 3 partes: el firmware que implementa el protocolo ICSP, la comunicación USB y el control del hardware para la obtención de los niveles de programación y alimentación; la etapa de potencia para la obtención de 13V para el voltaje de programación a partir de los 5V que proporciona el puerto USB; y la interfaz para el control y operación del programador.

Como puede apreciarse en el diagrama de la figura 3.7 el programador hace uso de un microcontrolador PIC18F2550. El arreglo de los capacitores y diodos de la parte superior de la figura forman un multiplicador de voltaje, que es el encargado de generar los 13V para el voltaje de programación. El transistor Q_2 es usado como un interruptor para conmutar entre los 13V o 0V para forzar al microcontrolador PIC, que será programado, a entrar en los modos de reinicio (RESET) y programación. La comunicación con la computadora se realiza a través del puerto USB. Este modelo de microcontrolador PIC dispone de un periférico que implementa el protocolo USB 2.0 siendo necesario sólo un capacitor de 47µF entre las terminales de referencia tierra V_{SS} y V_{USB} para el correcto funcionamiento del puerto USB.

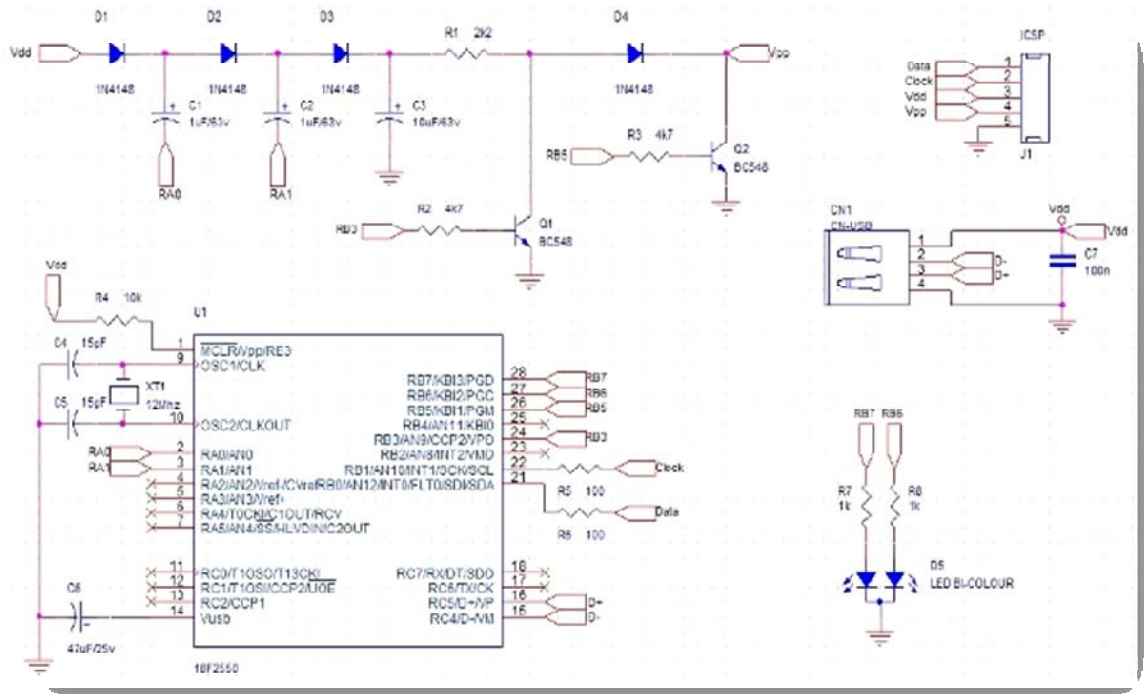


Figura 3.7. Diagrama eléctrico del programador *GTP USB lite*.

.

El software usado para la operación del programador es el WinPic 800, una interfaz gráfica para el sistema operativo Windows® que comentaremos con mayor detalle en una sección posterior de este mismo capítulo. En sus inicios, este programador era distribuido con toda la información necesaria para su construcción. Ahora, el firmware ha pasado a ser privado, es decir, es un producto comercial que no se distribuye libremente. El mantenimiento que se da tanto al firmware como al software es regular, pues cada año se agregan más y más modelos de microcontroladores a la lista de circuitos que puede soportar.

3.1.2 Tarjetas entrenadoras (Prototyping Boards)

Como ya fue mencionado en el capítulo 2, las tarjetas entrenadoras son circuitos impresos que poseen una base para circuito integrado donde se colocará un microcontrolador PIC y varios dispositivos periféricos. La figura 3.8 muestra diferentes modelos de tarjetas entrenadoras de diversos fabricantes, algunos de los diagramas pueden ser conseguidos en Internet pues son de libre distribución y otros pueden ser adquiridos a bajo precio.

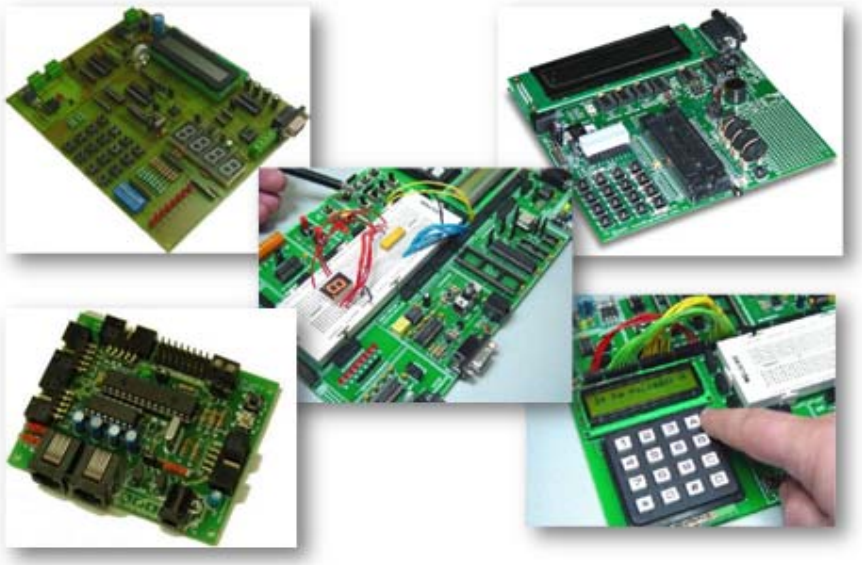


Figura 3.8. Diferentes modelos de tarjetas entrenadoras.

3.2 Herramientas en software

Entre los programas desarrollados por otras empresas o personas ajenas a Microchip que se utilizan en el desarrollo de aplicaciones para Microcontroladores PIC podemos encontrar compiladores, simuladores, entornos completos de desarrollo, ensambladores y programas para el control de programadores, emuladores y depuradores en circuito, no sólo para un ambiente Windows® sino para otras plataformas como Linux y Apple.

3.2.1 Compiladores

Los ***compiladores*** usados para el desarrollo de aplicaciones con microcontroladores PIC pueden dividirse en dos grupos: ensambladores y compiladores. Los ensambladores son programas encargados de traducir el código ensamblador en ***código máquina absoluto*** para un microcontrolador en particular. El programa encargado de traducir el código fuente de un lenguaje de alto nivel, como C o BASIC, en código de máquina absoluto es el ***compilador***.

Existen diferentes empresas diferentes a Microchip Inc. que proporcionan software para la edición y compilación de código fuente, ya sea en ensamblador o un lenguaje de alto nivel.

CCS C *Compiler*

La empresa CCS, Inc. (*Custom Computer Services*) es una compañía Norteamericana especializada en el diseño y desarrollo de software y hardware para dispositivos embebidos. Desarrolla herramientas para microcontroladores de Microchip de propósito general (MCU) y procesadores digitales de señales (DSP), como tarjetas entrenadoras, programadores y en el caso que nos ocupa, un compilador en lenguaje C del que se puede obtener una versión de evaluación para estudiantes en su sitio de Internet[3].

[3] http://www.ccsinfo.com

La versión del compilador de evaluación que puede obtenerse al momento de escribir este trabajo es la 4.073. Este compilador puede ser usado desde una línea de comandos o a través de una interfaz de usuario o IDE que hace más fácil la escritura y depuración de código fuente. La utilización de un IDE permite el uso de asistentes que generan código de manera automática una vez que han sido configurados algunos parámetros como el tipo de microcontrolador, periféricos a usar y tipo de oscilador, entre otras características.

CCS Inc. es una empresa reconocida por Microchip Inc. como un fabricante confiable de herramientas para sus microcontroladores, lo que inspira confianza al momento de elegir su compilador como una herramienta para el desarrollo de una aplicación, pues soporta la mayoría de los microcontroladores PIC de 8 bits y los más nuevos de 16 bits. La figura 3.9 muestra la pantalla principal de la aplicación.

Figura 3.9. Pantalla del compilador CCS C Compiler para microcontroladores PIC.

El compilador puede ser agregado al programa MPLAB IDE de Microchip a través de un *plug-in*, con lo que servirá como interfaz tanto para el compilador CCS C como para el administrador de proyectos y archivos. Otros *plug-in* pueden ser usados para dar soporte a otras herramientas de Microchip Inc. como los depuradores en circuito MPLAB ICD2 and MPLAB REAL ICE.

Esta herramienta no es distribuida libremente, pues se trata de un producto comercial. La versión de evaluación para estudiantes es una versión limitada en el número de modelos que puede compilar y el tiempo que se puede usar es de sólo 30 días. Una ventaja al hacer uso de este compilador es que se dispone de numerosa información, no sólo en la página oficial, sino en toda la Internet, además de cuantiosos ejemplos de código fuente y actualizaciones regulares del software y firmware para los demás productos.

PICBasic PRO *Compiler*

La empresa Engineering Labs desarrolló el compilador PICBasicPro que permite crear programas para cualquier microcontrolador PIC basado en lenguaje BASIC. Existe una versión de evaluación limitada a 30 días y documentación que puede ser encontrada en Internet para aprender a programar en este lenguaje.

HI-TECH C Compiler

La empresa HI-TECH ofrece sistemas de desarrollo para sistemas embebidos de diferentes fabricantes como Cypress, Silicon Labs y por supuesto, Microchip Inc. En el caso de los microcontroladores PIC proporciona diferentes versiones de su compilador dependiendo de la familia que se desee programar. Estas versiones van desde los microcontroladores PIC de gama baja y media, hasta los nuevos modelos de 14 y 32 bits. Existe una versión gratuita que se integra al MPLAB IDE que permite programar microcontroladores de las familias PIC10, PIC12 y PIC 16.

3.2.2 Simuladores

Existen diferentes programas de Diseño Asistido por Computadora (CAD) que permiten simular el comportamiento de diagramas eléctricos y electrónicos. Uno muy usado es la suite de Proteus (figura 3.10) de la empresa Labcenter Electronics que incluye diferentes programas para el diseño de diagramas eléctricos, simulación y diseño de PCB. Otros sistemas similares para la simulación de circuitos electrónicos con soporte para microcontroladores PIC son Multisim de la empresa National Instruments y Protel 99 SE.

Proteus

La principal ventaja de emplear programas como ISIS de la suite de Proteus es la posibilidad de usar una interfaz que permite no sólo conocer el estado de los registros del microcontrolador en un ordenamiento por tablas como ocurre con el MPLAB SIM, sino también ver de forma gráfica el comportamiento de diversos periféricos, como diodos LED, *display* de 7 segmentos, osciloscopios, pantallas de cristal líquido, teclados matriciales y otros dispositivos.

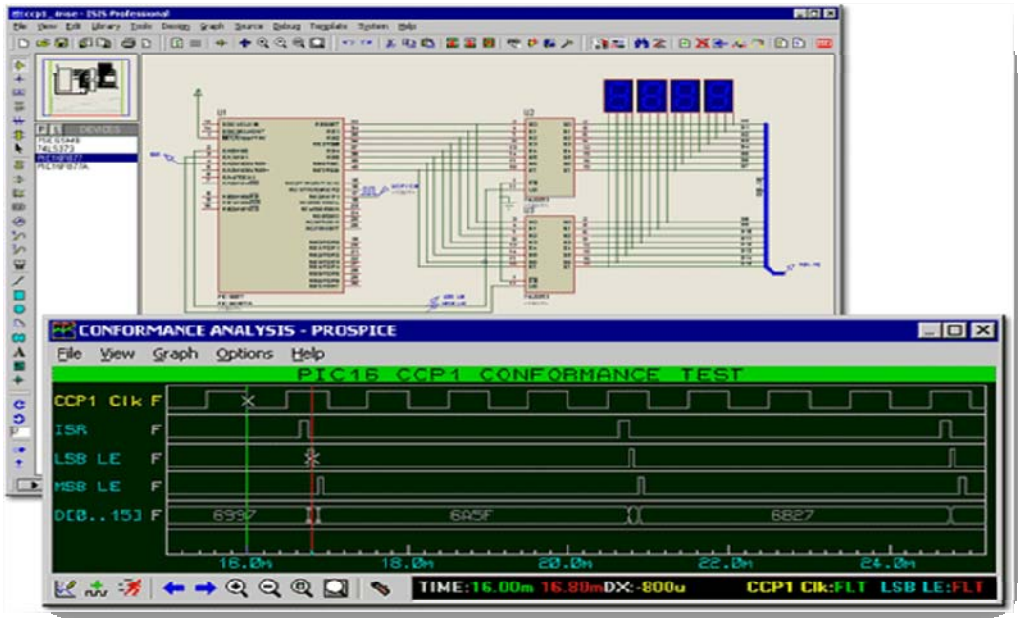


Figura 3.10. Pantallas del programa de diseño ISIS y del Analizador de señales de la suite Proteus de la empresa Labcenter Electronics.

3.3 Software de control para programadores de terceros

Debido a la popularidad de los programadores fabricados por terceros, algunos diseñadores de software ofrecen interfaces para diferentes plataformas o diversos sistemas operativos como Windows® o Linux que dan soporte a varios modelos de programadores comerciales y no comerciales. Dos de estas interfaces son IC-Prog y WinPIC 800.

IC-Prog

Hablar del programa IC-Prog (figura 3.11) es remontarse a los primeros años de la década de los 90s. Esta interfaz fue desarrollada por Ahmed Lazreg para dar soporte a los primeros programadores como el JDM y TAIT. Con el paso de los años, más modelos de microcontroladores PIC, microcontroladores AVR[4] y memorias EEPROM serie han sido agregados a la lista de componentes que es capaz de soportar, así como la cantidad de programadores.

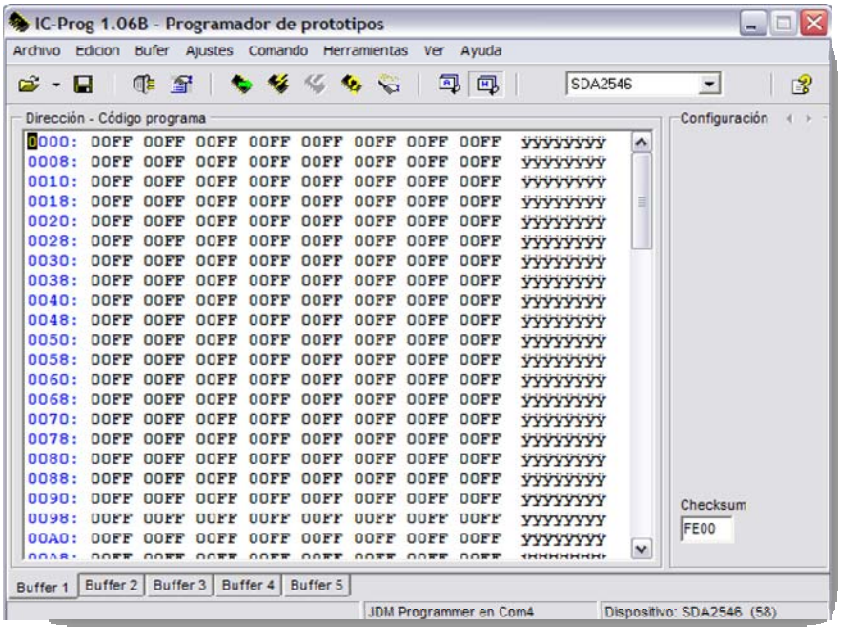


Figura 3.11. Pantalla principal del programa IC-PROG.

[4] Los microcontroladores AVR son microcontroladores Harvard RISC de 8 bits de la empresa ATMEL.

Entre las características más importantes que le permitieron durante todos estos años ser una opción muy confiable para el uso de los programadores JDM y TAIT se encuentran: el código fuente puede descargarse desde la página del autor, posee una interfaz gráfica que permite su configuración y control, además de que el idioma de la interfaz puede ser cambiado.

WinPIC800

El programa Winpic800 es de más reciente aparición en comparación con el programa IC-Prog pero dispone de las mismas capacidades. Entra las principales mejoras se encuentran la capacidad de soportar otros programadores como el GTP-USB. La figura 4.21 muestra la pantalla principal del programa WinPIC800.

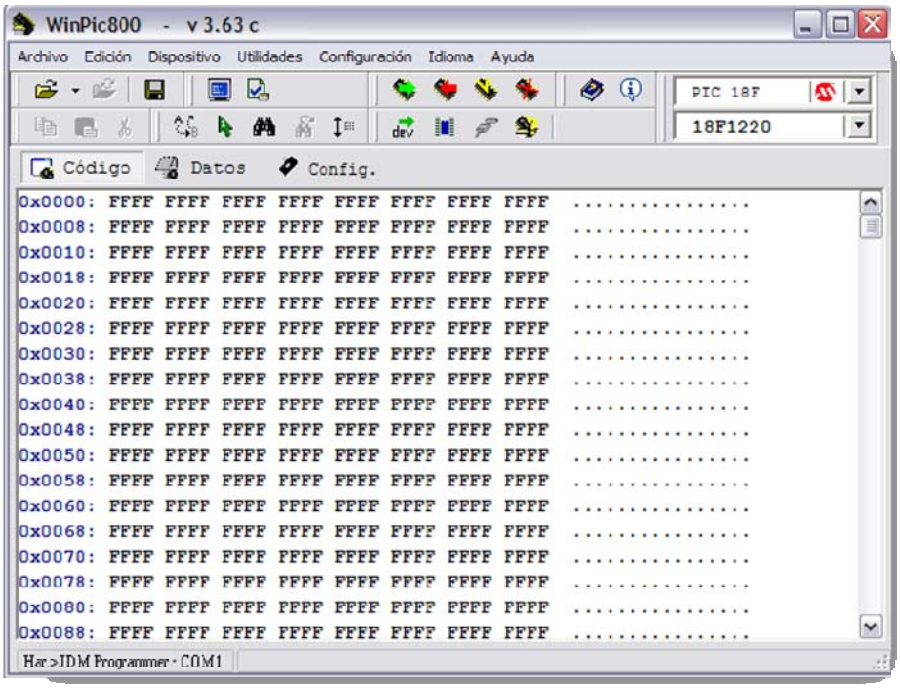


Figura 3.12. Pantalla principal de programa WinPIC800.

3.4 Conclusión del capítulo

Disponer de las herramientas para diseñar y desarrollar aplicaciones con microcontroladores nos permitirá construir prototipos que den una idea más clara de lo que sucede al probar nuestro programa. Dependiendo de la naturaleza de la aplicación, otro tipo de herramientas o equipo puede ser necesario, como fuentes de poder dobles, osciloscopios, generadores de funciones, y ciertamente el no tener acceso puede dificultar el desarrollo de una solución, pero es necesario no olvidar que el estudio e ingenio serán nuestras herramientas más importantes al aprender a desarrollar soluciones tecnológicas con microcontroladores PIC.

Capítulo 4

Diseño y desarrollo del programador USB de microcontroladores PIC

Capítulo 4
Diseño y desarrollo del programador USB de microcontroladores PIC

El diseño y desarrollo del proyecto *programador USB de microcontroladores PIC* será tratado desde tres áreas: circuito impreso (hardware), programa para el microcontrolador (firmware) y programa de aplicación (software).

Como se comentó en el capítulo 3, existen dos formas de implementar el protocolo ICSP usado por los microcontroladores PIC para su programación. La primera opción es implementar el protocolo a través de software, construyendo un programa encargado de controlar un puerto de comunicaciones de la PC al cual se conecta el hardware necesario para operar las señales de programación.

La segunda opción, es implementar el protocolo en un *sistema embebido*, el cual se comunica con una interfaz de usuario en una PC a través de un puerto de comunicaciones y es también el encargado de controlar el hardware para la operación de las señales de programación.

Para diseñar un *sistema embebido* puede recurrirse al diseño de sistemas a la medida o *Full Costume*, donde el circuito integrado encargado de la operación del sistema es diseñado completamente. En nuestro caso, la fabricación de un *controlador* mediante la técnica *Full Costume* para el programador USB de microcontroladores PIC no se justifica ya que se trata de un prototipo no comercial, además, el diseño y desarrollo de de un controlador de esta naturaleza sale fuera de los objetivos de este proyecto.

Una segunda opción es hacer uso de la ***lógica programable***, donde se emplean dispositivos lógicos programables como CPDLs o FPGAs, que son programados a través de lenguajes de descripción de hardware como VHDL o Verilog HDL. Uno de los factores en contra del uso de estos circuitos es su alto costo, comparado con otras tecnologías que nos permitirían tener un resultado similar.

La tercera opción disponible es hacer uso de un microcontrolador, que según se explicará en la sección 4.5 es la empleada para desarrollar el controlador del programador.

En el caso de la interfaz de usuario, serán expuestos los criterios utilizados para su desarrollo, así como el por qué del lenguaje elegido, en este caso, Visual C++.

El diseño del hardware en su etapa de potencia responde a los estándares fijados en los protocolos ICSP y USB. Además se proponen diferentes diseños de circuito impreso.

En la primera sección, se comentarán los requerimientos mínimos que debe cumplir el programador USB de microcontroladores PIC para su correcto funcionamiento, las especificaciones del diseño, así como los estándares utilizados para desarrollar el prototipo final. La segunda sección, abordará la implementación de los conceptos expuestos.

4.1 Características del hardware

El diseño de un sistema electrónico lleva consigo el sujetarse a estándares, normas y protocolos bien definidos que garantizarán el correcto funcionamiento de nuestro sistema, así como su interacción con otros dispositivos electrónicos y con el usuario. En este caso, el diseño del hardware está ligado a dos protocolos. El primero es el protocolo de comunicaciones USB, usado para comunicar el microcontrolador programado para controlar las operaciones del *dispositivo programador* con un equipo de cómputo como una PC o Laptop; el segundo es el protocolo ICSP que incluye las especificaciones usadas para programar los microcontroladores PIC. En esta sección se describirán las propiedades de estos dos protocolos usadas en diseño del *programador USB de microcontroladores PIC*.

4.1.1 El Puerto de comunicaciones USB

El Bus Serie Universal o USB (*Universal Serial Bus*), por sus siglas en inglés, es un estándar de comunicaciones, diseñado para conectar dispositivos periféricos a un equipo de cómputo, como lo son: teclados, ratones, cámaras web, teléfonos, entre otros. En esta sección se abordarán las características generales del estándar USB en su versión 2.0, lo que permitirá familiarizar al lector con los conceptos usados en el diseño y desarrollo del programador USB de microcontroladores PIC.

La primera versión del estándar se remonta al año de 1996 cuando las empresas IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipement Corporation y NEC, liberaron la versión 1.0, que buscaba reemplazar a los puertos de la PC usados para conectar dispositivos periféricos y tarjetas internas, y de manera paralela, eliminar los problemas que estas arquitecturas imponían al desarrollo de nuevo hardware bajo la implementación PC2000 propuesta por las compañías Intel y Microsoft.

Algunos ejemplos de periféricos que implementan el estándar USB son ratones, teclados, escáneres cámaras digitales, reproductores multimedia, memorias de almacenamiento secundario, tarjetas de sonido, tarjetas de red, bocinas, teléfonos celulares, reproductores de mp3, escáneres e impresoras, por citar algunos ejemplos (figura 4.1).

Figura 4.1. Logotipo del estándar USB. Ejemplos de dispositivos que implementan el estándar USB son periféricos de computadora, reproductores de mp3, teléfonos celulares, cámaras de video y fotográficas, equipos de sonido, lámparas, impresoras, entre otros.

Entre las características más relevantes de este bus se pueden enumerar las siguientes:

- Elimina la necesidad de usar tarjetas que se conecten al bus ISA o PCI, promoviendo el desarrollo de hardware que se conecte al puerto USB sin la necesidad de abrir el gabinete del CPU, configurar interruptores (DIP *switch*) o preocuparse por los vectores de interrupción disponibles (IRQ).
- Propone que los dispositivos que se conecten a un sistema anfitrión, sean instalados y configurados por el Sistemas Operativo, bajo la tecnología *plug-and-play*, lo que elimina la necesidad de apagar o reiniciar la PC al conectar o desconectar un dispositivo al bus.

- Permite la construcción de redes de periféricos, lo que favorece la conexión de varios dispositivos a través de concentradores a un solo sistema raíz USB. La extensión máxima de la red puede ser de 127 elementos incluidos la raíz y los concentradores.
- La alimentación eléctrica usada por los dispositivos debe ser suministrada por el mismo bus cuando los rangos necesarios no excedan los 5V ni los 500mA, por lo que para dispositivos de bajo consumo de potencia, no es necesaria una fuente de alimentación externa, ya que es posible que tomen la alimentación eléctrica de las mismas terminales del puerto.

Arquitectura del USB

Los componentes físicos del USB consisten en circuitos, conectores y cables entre un anfitrión (*host*) y uno a más dispositivos (*funtions*). Un *host* es una computadora que contiene un controlador anfitrión (*host controller*) y un concentrador inicial o raíz (*root hub*). Estos componentes trabajan juntos para permitir al sistema operativo comunicarse con los dispositivos conectados al bus. El controlador de *host* permite funciones relacionadas con la administración de la comunicación, mientras el *concentrador root* tiene uno o más puertos que permiten la conexión de dispositivos al bus. El concentrador *root* y el controlador de *host* detectan la conexión y desconexión de dispositivos, peticiones de uso del bus por parte de los dispositivos y la transmisión de datos entre dispositivos y el *host*, entre otras tareas. Los dispositivos son los periféricos y concentradores adicionales que se conectan al bus. Un concentrador tiene uno o más puertos para conectar dispositivos. Las características de los cables y los conectores que se usan para conectar dispositivos y concentradores serán comentados más ampliamente en una sección siguiente.

La arquitectura del USB está centrada en el *host*. El esquema que utiliza la arquitectura para el acceso al medio es similar a la red de área local *Token-Bus*, con la diferencia que el *host* es siempre el maestro en la comunicación y quien toma la iniciativa, preguntando a cada dispositivo si tiene información pendiente de transmitir. Sólo un elemento de la red puede comunicarse con el *host* a un tiempo, por lo que se trata de una comunicación *Half-Duplex*. La transmisión multipunto (*broadcasting*) no está implementada, ni es posible que dos funciones se comuniquen entre sí.

Un nuevo proyecto llamado USB *On-The-Go* contempla dotar de la habilidad de comunicar diferentes ***funciones*** sin necesidad de un *host* como ha sido descrito, por ejemplo una impresora con una cámara digital. El estudio de esta versión, al igual que del USB RF para aplicaciones de comunicación por radiofrecuencia, sale del propósito de este documento.

La arquitectura USB está diseñada teniendo presente los modelos de capas o niveles, como lo muestra la figura 4.2. Cada nivel está asociado con un nivel funcional dentro del dispositivo. La capa más alta es la configuración. Un dispositivo puede tener múltiples configuraciones. Por ejemplo, un dispositivo en particular puede tener requerimientos de potencia basados en el modo *Self-Power*[1] o *Bus-Power*[2]. Para cada configuración, puede haber múltiples interfaces. Cada interfaz podría soportar un modo en particular de configuración. Bajo la interfaz están los puntos finales o *endpoints*.

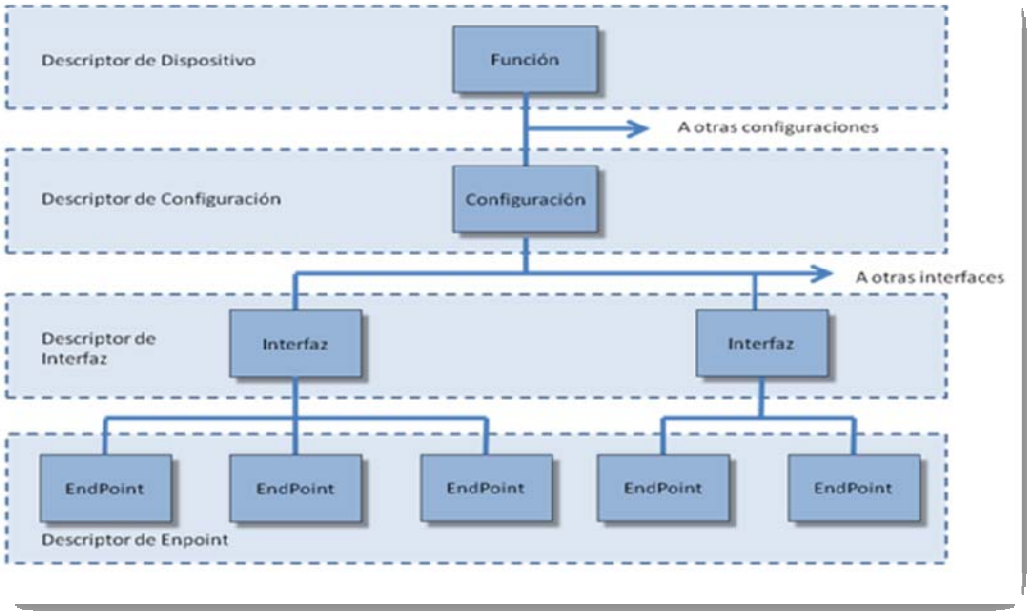


Figura 4.2. Modelo lógico de capaz del dispositivo USB.

[1] Es decir, cuenta con una fuente propia de alimentación.
[2] Es decir, cuenta toma la alimentación eléctrica de las terminales del USB.

Topología

USB es implementado bajo una topología física de estrella (*Tiered Star Topology*) y lógica de bus. Al representar está topología, es más fácil pensar en ella como una estructura de árbol, donde la raíz se encuentra en la cima, de ella se desprenden ramas que llegan a nodos, de los cuales se desprenden nuevas ramas que llegan a nodos terminales. El nodo principal es ocupado por el ***controlador del puerto*** o *host* y el ***concentrador raíz*** o *root* (desde ahora simplemente *host*), los nodos que se encuentran al centro de cada rama son ocupados por los ***concentradores*** *o hubs* y al final de cada rama encontraremos dispositivos individuales, también llamados ***funciones*** o *funtions*. La arquitectura del árbol está limitada a 6 niveles como máximo, pudiéndose conectar hasta 127 dispositivos sobre el bus. La figura 4.3 muestra un diagrama de la topología del bus USB.

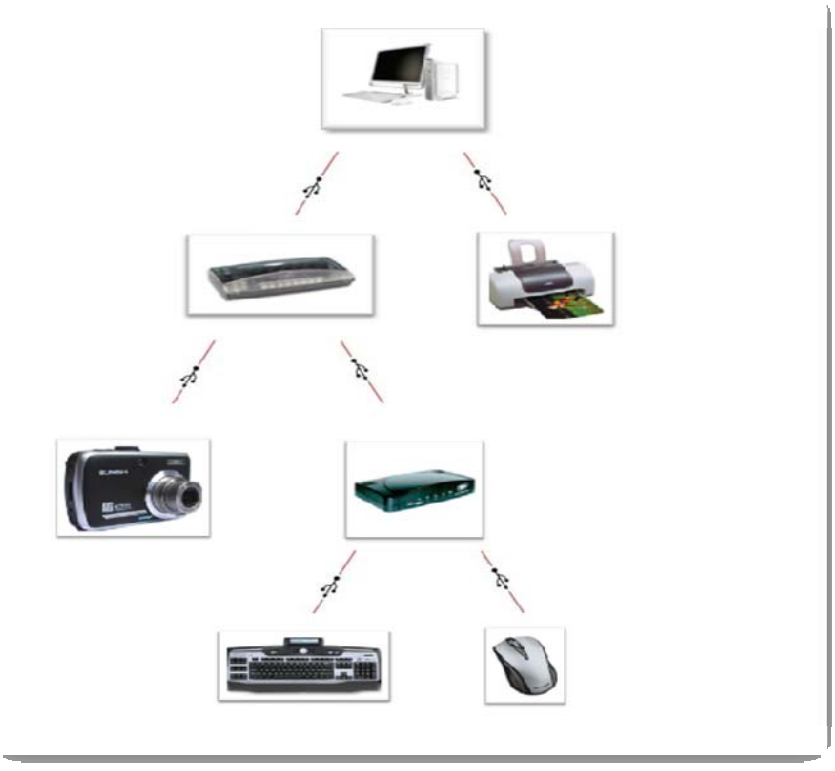


Figura 4.3. Topología del bus USB. En la cima de la arquitectura se encuentra el *host*, al final las funciones o dispositivos. Entre un *host* y una función siempre existe un concentrador o *hub*.

Existen diferentes versiones del estándar desde la publicación de la primera versión en 1996, actualmente se encuentra en desarrollo la versión 3.0. Las versiones 1.0 y 1.1 serán llamadas 1.1 de aquí en adelante, debido a que comparten la mayoría de las características y a que los nuevos periféricos que son construidos están basados en el estándar 1.1 y 2.0. Cuando sea necesario se hará referencia en particular a la versión 1.0.

Tramas

La información transmitida por el bus es agrupada en un formato llamado ***tramas*** (*frames*). Cada trama tiene una duración de 1mS de duración y está compuesta por múltiples ***transferencias***. Cada tipo de transferencia puede formar parte de una trama sólo una vez.

Protocolos de transferencia

Dependiendo de las necesidades de la comunicación, como cantidad de información a transmitir, latencia y naturaleza de la información, entre otras, los dispositivos USB usan diferentes protocolos para comunicarse con el *host*. Dependiendo de la versión del estándar USB bajo la cual se diseñó cada dispositivo, implementará uno o más de lo siguientes protocolos:

Transferencia de control (*Control Transfer*). Es usada para propósitos de configuración de dispositivos, tanto en el momento de su instalación, como en los instantes que requiera los parámetros de control del *host*. Se envían tramas de 8, 16, 32 o 64 bytes, dependiendo de la velocidad del dispositivo que se pretende usar. Todos los dispositivos implementan este protocolo desde la versión 1.0.

Transferencia por Interrupción (*Interrupt Transfer)*. Está diseñada para comunicar dispositivos que tienen un requerimiento moderado de flujo de datos con el *host*, de forma poco frecuente. Dispositivos conocidos como HID (*Human Interface Device*) o Dispositivos de Interfaz Humana como lo son teclados, ratones y controles de juego, son un ejemplo de los dispositivos que hacen uso de este protocolo. La trama de datos tiene las mismas dimensiones que en la transferencia de control.

Transferencia Isócrona (*Isochronous Transfer*). Provee un método de transferencia para grandes paquetes de datos (superiores a 1023 bytes). Este protocolo trata de garantizar una transmisión continua de datos en intervalos de tiempo regulares. La integridad de los datos no es comprobada porque no hay tiempo de reenviar paquetes erróneos, considerando la pérdida de datos como no significativa, por lo que puede decirse que se trata de una transferencia en tiempo real. El dispositivo negocia con el *host* una porción del ancho de banda del bus, para poder realizar el envío de tramas de forma periódica y con una latencia determinada. Un ejemplo de transferencia isócrona es la transmisión de audio.

Transferencia masiva (*Bulk Transfer*). Es lo opuesto a la transferencia isócrona. La integridad de los datos es comprobada, es decir, se valida la correcta transmisión de la información, pero no se garantiza que su llegada sea a intervalos regulares de tiempo y es usada para transferencia de datos que requieren usar todo el ancho de banda disponible en un determinado momento. Si no es posible realizar la transmisión bajo estas características, la transmisión puede demorarse hasta que el ancho de banda esté totalmente disponible. La transmisión de datos a dispositivos como unidades de almacenamiento secundario es un ejemplo del uso de este tipo de transferencia.

Full Speed y *High Speed* soportan los cuatro tipos de transferencia, es decir, soportan la transferencia Isócrona, Masiva, Interrupción y Control, mientras que *Low Speed* sólo soporta dos tipos de transferencia, Interrupción y Control.

Velocidades de Transferencia

La versión 1.0 posee un ancho de banda de 1.5Mbps, conocido como ***baja velocidad*** o *Low Speed*. Es utilizada para comunicar dispositivos con el *host*, que no requieren enviar grandes volúmenes de información y lo hacen de manera asíncrona, como es el caso de teclados, ratones, tabletas digitalizadoras y controles de juegos.

La versión 1.1 posee un ancho de banda de 12Mbps, conocida como ***velocidad completa*** o *Full Speed*. Una de las aportaciones a esta versión es la capacidad de dividir el ancho de banda de la conexión USB entre los dispositivos conectados al *host*, basada en un algoritmo de *buffers* FIFO (*First Input, First Output*). En esta versión sólo es posible el uso de los protocolos de Transferencia de Control (*Control Transfers*) e Interrupción (*Interrupt Transfers*).

La versión 2.0 utiliza un ancho de banda de 480Mbps, denominada ***High Speed***. En esta versión son agregados dos nuevos protocolos de transferencia, el protocolo de transferencia masiva (***Bulk Transfer)*** e isócrona (***Isochronous Transfer***).

La versión 3.0 actualmente se encuentra en fase de prueba con planes a ser liberada a finales de este año. El ancho de banda de esta versión se considera en 4.8 Gbps, gracias a que los cables de cobre serán sustituidos por fibra óptica que junto a transceptores permitirán el uso de los mismos conectores de cobre, para hacerlo compatible con las versiones anteriores.

Tuberías (*pipe*s) y Puntos finales (Endpoint)

El flujo de datos del bus USB puede entenderse como una serie de puntos finales (*endpoints*), que se agrupan en conjuntos, dando lugar a interfaces, las cuales permiten controlar la ***función*** del dispositivo. La figura 4.4 muestra un diagrama sobre el flujo de la comunicación USB entre un *host* y un dispositivo USB. Cada dispositivo USB está compuesto por *endpoints* independientes y una dirección única asignada de manera dinámica por el sistema en el tiempo de enumeración.

En el nivel más bajo de la capa de enlace de datos, cada dispositivo controla uno a más *endpoints*. Un *endpoint* puede ser descrito como un puerto virtual y son usados para comunicarse con una ***función***. Cada *endpoint* puede ser fuente o destino de un dato, por lo que están agrupados en pares, uno de entrada (IN) y otro de salida (OUT). Como máximo, pueden existir 15 *endpoints* para ser utilizados por los dispositivos *Full Speed* y 6 *endpoints* para cada dispositivo *Low Speed*. Si recordamos, la arquitectura del bus USB es centralizada en el *host*, por lo que la entrada y salida es con respecto al *host* y no al dispositivo.

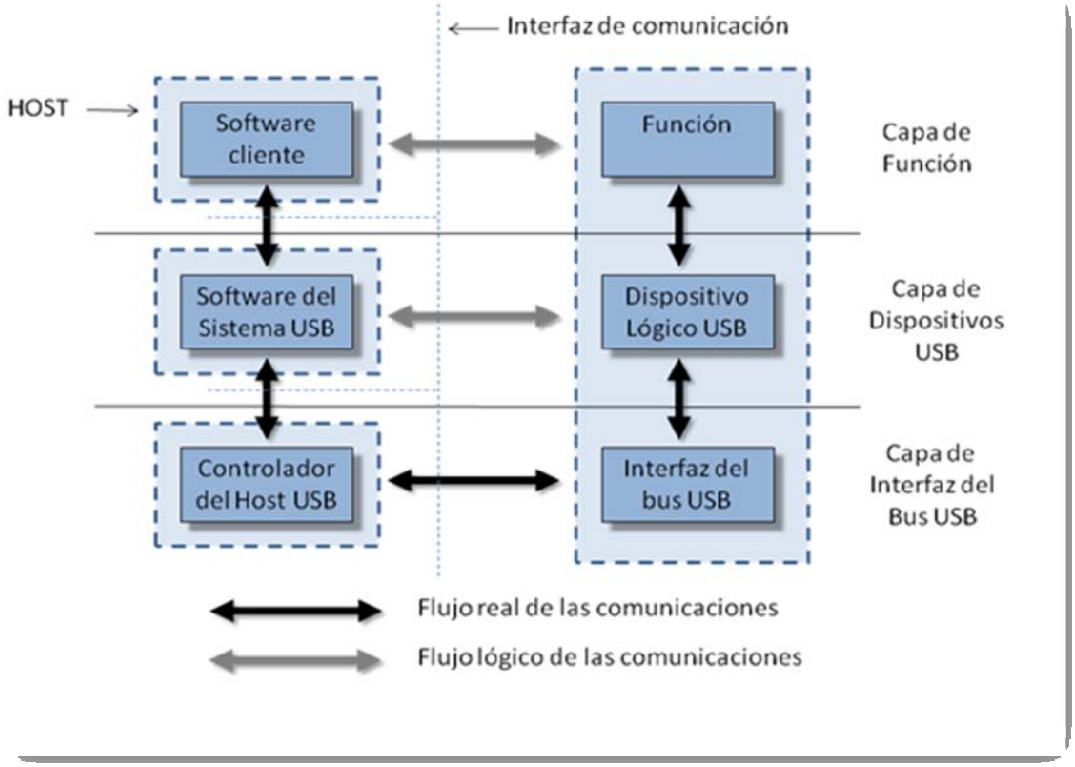


Figura 4.4 Flujo de de datos entre un *host* y un dispositivo USB.

El controlador se comunica con la interfaz del bus utilizando el cable USB, mientras que en un nivel superior el software USB del sistema se comunica con el dispositivo lógico utilizando la tubería de control (*pipe*) por defecto, en lo que de función se refiere, el software cliente establece la comunicación con las interfaces de la función a través de tuberías asociadas a puntos finales.

Un *pipe* o tubería es una conexión lógica entre un *endpoint* y el software del *host*, que permite el intercambio de datos entre éstos. El *pipe* formado por el *endpoint* 0 se denomina *pipe* de control principal. Este *pipe* está disponible desde el momento en que se conecta el dispositivo y se reinicia el bus. Una vez configurado el dispositivo, el resto de los *pipe*s son creados. El *pipe* de control principal es usado por el software del USB para obtener la identificación del sistema y para configurar al dispositivo.

Enumeración (*enumeration*)

Para que un dispositivo pueda comunicarse a través de bus, el *host* debe detectar que el dispositivo fue conectado y entonces iniciar un proceso de ***enumeración*** (*enumeration*). Este proceso permite al *host* preguntar al dispositivo sus características y negociar los parámetros de rendimiento, como lo son los requerimientos de potencia, protocolos de transferencia a usar y el tiempo en que le preguntará si existe información para transmitir (*polling rate*).

Cuando un dispositivo es inicialmente conectado al bus, el *host* entra en un proceso de ***enumeración*** en un intento de identificar el dispositivo. Esencialmente, el *host* interroga al dispositivo, reuniendo información como puede ser el consumo de potencia, la cuota de datos y tamaño, protocolos u otra información descriptiva, a través de la solicitud de los ***descriptores*** que son quienes contienen esta información. Un proceso típico de ***enumeración***, aunque podría cambiar de un *host* a otro, es el siguiente:

1. Reinicio del USB. Reinicia el dispositivo, así el dispositivo no está configurado y no tiene una dirección (dirección 0).
2. Obtener el descriptor de dispositivo. El *host* solicita parte del descriptor de dispositivo.
3. Reinicio del USB. El dispositivo es nuevamente reiniciado.
4. Asignar dirección. El *host* asigna una dirección al dispositivo.
5. Obtener el descriptor de dispositivo. El *host* recupera el descriptor de dispositivo, reuniendo información como el identificador del fabricante, identificador de dispositivo y tamaño máximo de paquetes de control.
6. Obtener el descriptor de configuración.
7. Obtener cualquier otro descriptor.
8. Asignar una configuración.

Para realizar la detección de la conexión de un dispositivo al bus USB por parte del *host*, se revisa la diferencia de potencial entre las líneas D+ y D- como se describe a continuación. El *hub* tiene una resistencia *pull-down* de 15 KΩ en cada una de las líneas de datos del puerto USB. El dispositivo tiene una resistencia *pull-up* de 1.5 KΩ en la línea D+ cuando se trata de *Full Speed* o *High Speed*, y en la línea D- cuando se trata de *Low Speed*, conectada a un voltaje de 3.3V.

Cuando el dispositivo se conecta al puerto, la línea que tiene la resistencia *pull-up* es más positiva que el valor lógico alto del *hub*. El *hub* detecta el cambio en el voltaje y asume que un dispositivo está conectado, además de determinar la velocidad según la línea que está conectada a nivel alto.

El ***descriptor de dispositivo*** (*Device Descriptor*) provee información general sobre el dispositivo, como puede ser, nombre del fabricante, número o identificador de producto, número de serie, clase del dispositivo USB (USB device class) y número de diferentes configuraciones soportadas. Sólo puede haber un descriptor de dispositivo por cada dispositivo.

El ***descriptor de configuración*** (*Configuration Descriptor*) da información sobre los requerimientos de potencia del dispositivo y cuántas interfaces diferentes son soportadas para la configuración. Puede haber más de un descriptor de configuración para un dispositivo, por ejemplo, un dispositivo de alto consumo de potencia puede soportar una configuración de bajo consumo.

El ***descriptor de interface*** (*Interface Descriptor*) determina el número de *endpoints* usados en esta interfaz, así como el controlador de clase (class driver) a usar para soportar las funciones del dispositivo por el sistema operativo. Sólo puede haber un descriptor de interfaz por cada configuración.

El ***descriptor de endpoint*** (*Endpoint Descriptor*) detalla los registros actuales para una función dada, la información sobre los tipos de transferencia soportada, dirección (I/O), ancho de banda (*bandwidth*) requerido e intervalo en que se estará preguntado al dispositivo si tiene información para transmitir (*polling rate*). Puede haber más de un descriptor de *endpoint* en un dispositivo y los *endpoints* pueden ser compartidos entre diferentes interfaces.

Estos descriptores incluirán uno o más ***descriptores string***. Los descriptores *string* son usados para proporcionar información en forma de cadena de caracteres, como el nombre del fabricante o información específica de la aplicación. Pueden ser opcionales y son codificados en formato “Unicode”.

Clases de dispositivos (Device Class) y controladores de clase (Class Driver)

Para que la comunicación entre una aplicación o programa y un dispositivo se logre, se hace uso de un controlador (*driver*) que le proporciona al sistema operativo un grupo de funciones para el acceso, control, administración y operación del dispositivo. Bajo el criterio de unificar controladores para dispositivos con funciones comunes, se creó el concepto de ***clase de dispositivo***. Ejemplos de clases de dispositivo son comunicaciones, almacenamiento, y HID. La tabla 4.1 muestra la lista completa de clases de dispositivo.

La forma de asociar una aplicación con un controlador y a su vez con un dispositivo es a través de los descriptores de dispositivo e interfaz. Los sistemas operativos más nuevos tienen ya incluidos los controladores para clases de dispositivos definidos en el estándar de la versión 2.0. Si el dispositivo necesita de funciones que no son soportadas por alguno de los controladores de clase, el fabricante puede proveer su propio controlador y un archivo ".inf" que contenga la información de la instalación, como es el caso de Microchip, que provee un controlador para sus microcontrolador PIC cuando estos no son configurados para ser instalados como una clase conocida por el sistema operativo, por ejemplo, como un dispositivo HID.

Interfaz físico-eléctrica

La capa física incluye los tipos y características de cables y conectores usados, así como de los niveles eléctricos y la codificación para la representación de valores lógicos. Para realizar una transmisión, la arquitectura del USB dispone de 4 líneas, organizadas en pares. Un par (líneas D+ y D-) es usado para el envío y recepción de datos y el par restante (líneas V_{USB} y GND) es usado para suministrar la alimentación eléctrica a los dispositivos de bajo consumo que no dispongan una fuente de alimentación propia, como lo son el teclado, ratón o unidades de almacenamiento secundario de memorias flash (*pen drive*). La tabla 4.2 describe el uso y distribución de las líneas del cable USB.

Clase	Descripción	Descriptor	Ejemplo
00h	Unspecified	Dispositivo	No está especificada. El descriptor de interfaz es usado para determinar los requerimientos del controlador.
01h	Audio	Interfaz	Bocinas, micrófonos y tarjetas de sonido.
02h	Communications and CDC control	Dispositivo e Interfaz	Adaptadores Ethernet, modem y adaptadores de puerto serie.
03h	Human Interface Device (HID)	Interfaz	Teclados, ratones y controles de juego.
05h	Physical Interface Device (PID)	Interfaz	Fuerza la retroalimentación a controles a de juego.
06h	Image	Interfaz	Cámaras digitales (la mayoría de las cámaras funcionan bajo la clase Mass Store para permitir acceder directamente al medio de almacenamiento).
07h	Printer	Interfaz	Impresoras laser y de inyección de tinta.
08h	Mass Storage	Interfaz	Reproductores de música, discos duros externos, unidades de memoria Flash y lectores de tarjetas de memoria.
09h	USB hub	Dispositivo	Concentradores Full Speed, Concentradores High Speed.
0Ah	CDC-Data	Interfaz	Esta clase es usada junto con la clase 02h.
0Bh	Smart Card	Interfaz	Lectores USB de tarjetas smart.
0Dh	Content Security	Interfaz	
0Eh	Video	Interfaz	Cámaras Web
0Fh	Personal Healthcare	Interfaz	
DCh	Diagnostic Device	Dispositivo e Interfaz	
E0h	Wireless Controller	Interfaz	Adaptadores Wi-Fi y adaptadores Bluetooth.
EFh	Miscellaneous	Dispositivo e Interfaz	Dispositivos ActiveSync
FEh	Application Specific	Interfaz	Puentes IrDA
FFh	Vendor Specific	Dispositivo e Interfaz	Indica que el dispositivo necesita de un controlar específico provisto por el fabricante.

Terminal	Normal	Descripción	Color
1	V_{CC}	+ 5V	Rojo
2	D -	Data -	Blanco
3	D+	Data +	Verde
4	GND	Señal de tierra	Negro
-	ID	Permite la distinción entre Micro-A y Micro-B Tipo A conectada a tierra Tipo B no conectada	No definido

Los pulsos de reloj usados para la sincronización de la comunicación son transmitidos en los mismos bits de datos mediante la codificación NRZI (non Return to Zero Invert), es decir, las tramas son auto sincronizadas. En esta codificación cada estado lógico se representa por un nivel de tensión distinto de cero. A través de las líneas D+ y D- se transmiten los datos de manera simétrica, de forma similar a como se realiza en equipos de audio de alta fidelidad, con la finalidad de reducir al mínimo el ruido generado por campos electromagnéticos durante la transmisión.

Las líneas V_{BUS} y GND son usadas para llevar la alimentación eléctrica a los dispositivos, con una tensión de +5V sobre la línea V_{BUS} con referencia a GND. Los requerimientos de los cables USB para *High Speed* y *Full Speed* difieren de los L*ow Speed*. La tabla 4.3 resume las principales características de los cables usados para cada velocidad.

Como máximo, la longitud permitida es de 3 metros para *Low Speed* y de hasta 5 metros para *High Speed* y *Full Speed*, entre un dispositivo USB y el siguiente. Como ya se mencionó, el máximo número de niveles que puede tener la estructura de árbol de la topología es de 6 niveles, iniciando por el ***concentrador raíz*** (*root*), por lo que es posible tener una máximo de 25 metros entre el concentrador raíz y el dispositivo al final de la red.

Tabla 4.3. Principales características de los cables usados en la arquitectura USB.

Especificación	Low Speed	Full/High Speed
Longitud máxima (metros)	3	5
Cable blindado y referencia a tierra física requeridos	Si (Definido para 2.0)	Si
Malla de blindaje externa requerida	No, pero recomendada	Si
Par trenzado	No, pero recomendado	Si
Impedancia de modo común (ohm)	No especificado	30 ± 30%
Impedancia diferencial (ohm)	No especificado	90
Diámetro del cable (AWG #)	20-28	
Retardo del cable	18 nS (un sentido)	5.2 nS
Localización en el dispositivo de la resistencia pull up	D-	D+

Existen dos tipos básicos de concentradores, estos pueden alimentarse del mismo bus y trabajar bajo el modo *Bus-Power*, o tener una fuente de alimentación independiente y trabajar bajo el modo *Self-Power*. Un concentrador *Self-Power* permite mantener los niveles de corriente y voltaje estables para los dispositivos conectados a él, sin la necesidad de tomar la corriente del ***concentrador raíz***. Esto es muy importante al considerar que los dispositivos son diseñados para consumir ***unidades de carga*** (*units loads*) definidas en 100mA y que cada puerto como máximo está en posibilidades de suministrar 5 unidades de carga o 500 mA.

Por esto, el uso de concentradores *Bus-Power* no es recomendado cuando la red de dispositivos tiende a crecer. Una solución a este problema es la adición de varias concentradores raíz en un equipo de cómputo, lo que reduce la necesidad de usar concentradores externos, ya sean activos o pasivos. Esta corriente es capaz de alimentar cualquier aparato en el que la potencia requerida no sea muy elevada (como webcams, lápices ópticos, ratones, entre otros), de esta manera se reduce el cableado. La figura 4.5 muestra el circuito de conexión para alimentar un dispositivo USB a través de los modos *Self-Power* y *Bus-Power.*

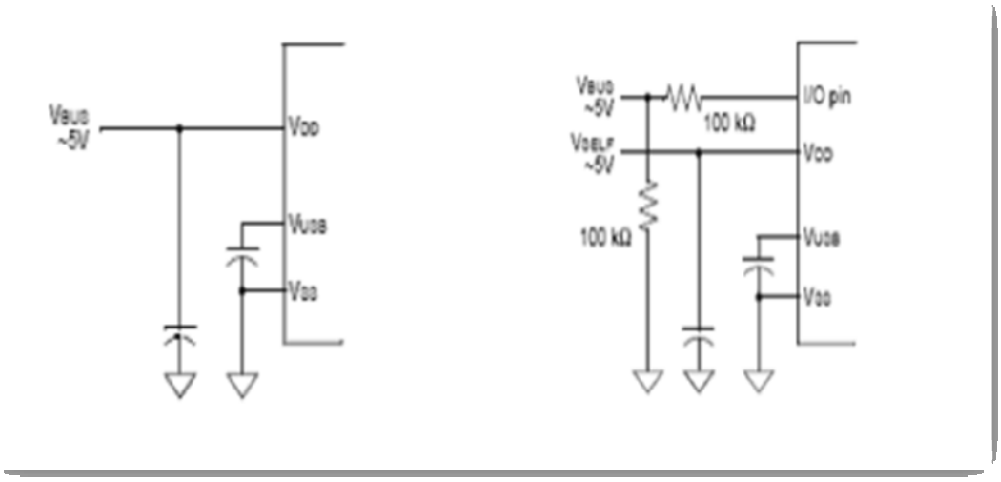


Figura 4.5 A la izquierda una configuración *Bus-Power*. A la derecha una configuración *Self-Power.*

Algunas aplicaciones pueden requería una opción de poder doble. Esto permite a las aplicaciones usar una fuente de poder primaria interna, pero cambiar a la alimentación del USB cuando la alimentación interna no está disponible. La figura 4.6 muestra un ejemplo simple de conexión de un arreglo *Dual-Power* con *Self-Power* predominante, con cambio automático entre los modos *Self-Power* y *Bus Power.*

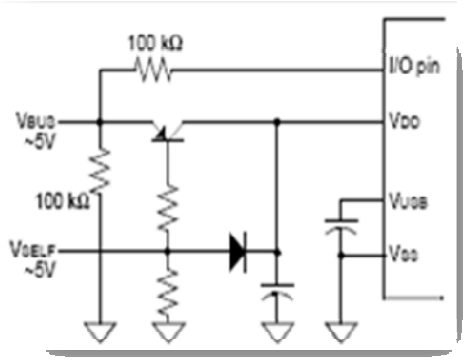


Figura 4.6. Arreglo *Dual Power*.

Conectores (Connectors)

La especificación USB 2.0 define cuatro tipos de conectores básicos. Los conectores de la serie A se utilizan para identificar el extremo donde se encuentra el *host* o el *hub*, mientras que los conectores de la serie B se utilizan para identificar el extremo donde se encuentra el dispositivo. Tener dos series de conectores permite prevenir errores de conexión, ya que una serie es totalmente incompatible con la otra, es decir, no es posible conectar en un puerto de la serie A un conector de la Serie B.

Con el desarrollo de nuevos dispositivos que usaban el estándar USB, se creó una nueva necesidad, al usar conectores de la seria B en dispositivos de dimensiones muy reducidas, como teléfonos celulares o reproductores de mp3, sus fabricantes debieron dotarlos de nuevas versiones más reducidas del conector. Con el objetivo de unificar criterios tanto en el diseño como en el desarrollo, al igual de mantener niveles de calidad y compatibilidad, se propuso un nuevo conector denominado mini USB. La figura 4.7 muestra los conectores definidos por el estándar USB Versión 2.0. Con la aparición de la versión 3.0 a finales de este año, se definirán nuevos conectores que permitan mantener la compatibilidad con la versión 1.1 y la versión 2.0, mediante el uso de un transceptor de fibra óptica.

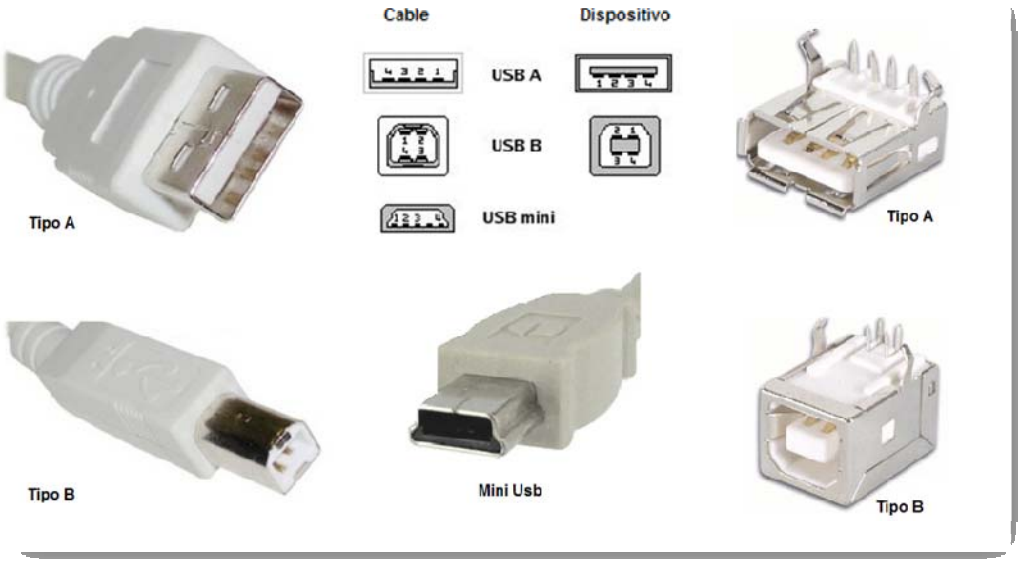


Figura 4.7. Conectores para cables y dispositivos USB.

4.1.2 El protocolo ICSP y la programación de microcontroladores PIC

El proceso de grabar la memoria de programa de un microcontrolador una vez que ha sido montado en el sistema a controlar se conoce con el nombre de ***Programación en Sistema*** o ISP *(In-System Programming*) y es utilizado para que el sistema a controlar pueda ser actualizado con la última versión de firmware, después de haber sido ensamblado y antes de salir a su venta. Una práctica diferente a la técnica ISP, que se justifica en grandes volúmenes de producción del mismo sistema, es programar los microcontroladores en el momento de su fabricación a través de una memoria ROM con máscara.

Microchip ofrece incorporado en la arquitectura de sus microcontroladores PIC el protocolo ICSP, una variante de la técnica ISP. Hacer uso de la técnica ISP y del protocolo ICSP supone los siguientes beneficios:

- El dispositivo a controlar puede ser fabricado sin necesidad de programar el microcontrolador, lo que supone que cuando sea necesario programarlo, podrá tener la última versión del firmware.
- El microcontrolador puede ser programado sin necesidad de modificar la línea de producción. En el caso de los microcontroladores con memoria ROM con máscara, si es necesario modificar su firmware, habrá que modificar el diseño del chip desde su fabricación, cosa que no ocurre si el microcontrolador es programado posteriormente a su fabricación.
- El firmware del microcontrolador puede ser actualizado por una nueva versión sin necesidad de remover el circuito integrado del sistema, es decir, el microcontrolador puede ser reprogramado en campo.
- Pueden agregarse cambios rápidamente en la línea de producción, con lo que se reduce el tiempo de desarrollo y ciclos de fabricación, al igual que se disminuye el tiempo de salida al mercado del producto final.

Para programar los microcontroladores PIC, dependiendo del modelo, podemos hacer uso de la programación en alto voltaje, en bajo voltaje o durante la operación del microcontrolador.

La programación durante la operación normal del microcontrolador se realiza, en dispositivos que soportan esta característica, a través de registros de control y estado destinados para tal propósito como si la memoria de instrucciones se tratará de un periférico más, por lo que no se hace uso del protocolo ICSP. Los dispositivos programadores son diseñados para implementar el protocolo ICSP que será descrito en la sección siguiente.

Capa física del protocolo ICSP

El protocolo ICSP está dividido en dos capas, la capa física y lógica (figura 4.8). En la capa lógica se describen los comandos definidos para la programación de los microcontroladores PIC, mientras que en la capa física se definen las características eléctricas, distribución de terminales usadas por el protocolo, así como la técnica de sincronización y temporización a nivel de bits.

Son 5 las señales que se usan para manejar el protocolo ICSP: V_{PP}, V_{DD}, V_{SS}, PGC y PGD. Las señales V_{DD} y V_{SS} están destinadas para proveer la alimentación eléctrica del circuito con valores de 4.5V a 5.5V (típicamente 5V) y referencia a tierra respectivamente. La señal V_{PP} es el voltaje de programación con un valor por lo general de 13.5V, aunque esté puede variar en un rango de los 12V a 14.V, lo cual depende del modelo de microcontrolador. La señal V_{PP} se aplica en la terminal MCLR/V_{PP}. Su función es hacer entrar al microcontrolador en **modo de programación** o "*Program/Verify mode*".

Figura 4.8. Modelos de dos capas del protocolo ICSP.

En modo de programación, las terminales asociadas a PGC y PGD, por ejemplo RB6 y RB7 en un PIC16F877A (figura 4.9), dejan de ser puertos digitales y pasan a ser entradas *Trigger Schmitt*, por donde serán enviadas las señales de reloj y de datos. La terminal PGD es bidireccional, por lo que puede ser configurada como entrada o salida en tiempos diferentes. A través de ella se envían comandos o datos al dispositivo a programar y se reciben los datos leídos de la memoria del microcontrolador. La señal de reloj PGC es generada por el dispositivo programador, quien siempre tiene el papel de maestro en la comunicación. De esta descripción de las terminales y su funcionamiento puede concluirse que el protocolo ICSP hace uso de una comunicación serie síncrona, maestro-esclavo, en modo *Half-duplex*. La tabla 4.4 resume las principales características de las terminales usadas por el protocolo ICSP.

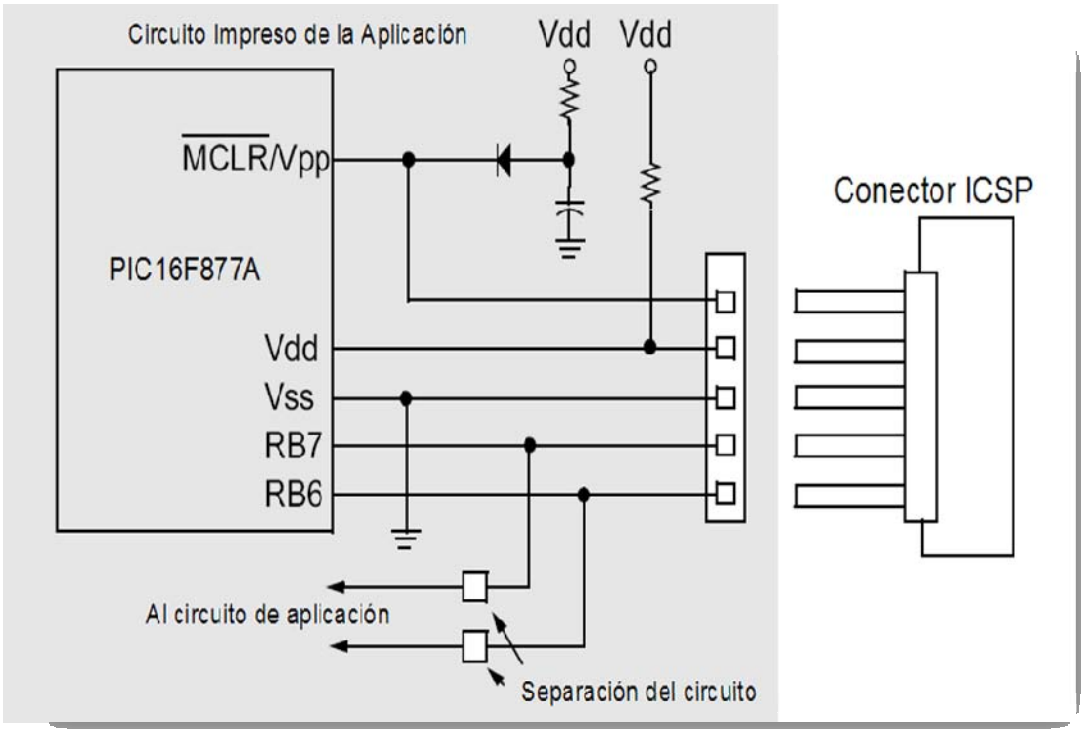


Figura 4.9 Conexión de terminales en un microcontrolador PIC16F877A para implementar el protocolo ICSP.

Tabla 4.4. Terminales usadas por el protocolo ICSP.

Terminal	Función	Tipo de Terminal	Descripción	Valores de Tensión
RB6	CLOCK	Entrada	Entrada de reloj	0V-5V
RB7	DATA	Entrada/Salida	Entrada y salida de Datos	0V-5V
RB3	PGM	Entrada	Voltaje de programación	5V
MCLR	V_{PP}	Programación	Activación de programación a bajo nivel	13V
Vdd	V_{DD}	Alimentación	Voltaje positivo	5V
Vss	V_{SS}	Alimentación	Voltaje de referencia a tierra	0V

Para entrar en modo de programación a bajo nivel se hace uso de una terminal extra, la terminal PGM. No todos los modelos de microcontroladores PIC soportan este modo de programación, por lo que será necesario revisar la hoja de datos de cada modelo para corroborar si implementa este recurso.

La programación en bajo nivel permite al microcontrolador entrar en el modo de programación sin la necesidad de suministrar los 13.5V en la terminal MCRL/$_{VPP}$. En su lugar, la terminal PCM es puesta en una configuración especial que permite que cuando el microcontrolador inicie la operación en modo normal y el valor lógico de esta terminal sea un nivel alto, el microcontrolador entre en modo de programación, todo con una fuente de alimentación de 5V.

Para poder configurar la terminal PMG y activar en modo de programación a bajo voltaje es necesario configurar el bit LVP de la palabra de configuración, es decir, será necesario programar el microcontrolador una primera con un programador que haga uso del modo de programación en alto voltaje.

La forma más común de hacer entrar a un microcontrolador en modo de programación es a través de la siguiente secuencia[3]:

[3] Los valores de voltaje de cada señal son mostrados en la tabla 4.4 de características eléctricas.

- Hacer entrar al microcontrolador en modo de reinicio (RESET), es decir, proporcionar alimentación eléctrica al dispositivo mientras el estado de la terminal MCRL se mantiene a un nivel próximo a V_{IL} (0V). Esto permite que todas las terminales se encuentren en un estado de alta impedancia.
- Mantener las terminales RB6 y RB7 a nivel bajo, mientras un flanco de subida es aplicado en la terminal MCLR desde un voltaje V_{IL} (0V) hasta un voltaje V_{IHH} (13.5V). El tiempo para pasar del nivel bajo al alto debe ser inferior a 72 ciclos de reloj para asegurar que el dispositivo ha entrado en modo de programación.

Una vez en este modo puede ser leído o escrito, de forma serie través de las terminales PGC y PGD, el contenido de la memoria de programa, de la EEPROM de datos o de localidades especiales de configuración e identificadores (ID). En microcontroladores con memoria ROM, sólo la memoria de datos EEPROM y el bit de código de protección (CDP) de la palabra de configuración pueden ser modificados.

En modo de programación, el espacio direccionable de la memoria de programa se extiende por arriba de la última dirección implementada hasta una región especial, que sólo puede ser accedida en modo de programación. Para los microcontroladores de la familia PIC12, está región varia de un dispositivo a otro. Por ejemplo, en un modelo 12F508 la primera dirección de memoria de configuración se encuentra en la dirección 0x1FF mientras que para el modelo 12F509 está localizada en la dirección 0x3FF.

Para la familia PIC16 la memoria de programa direccionable puede extenderse hasta los 8K Words, pero es en la dirección 0x2000 donde comienza el primer registro de la memoria de configuración según puede apreciarse en la figura 4.10. En modelos de microcontroladores como el 16F84A, la región que inicia en la dirección 00h hasta 3FFh (1KW) puede ser accedida durante la operación en modo normal del dispositivo, pero una vez en modo de programación el rango de memoria direccionable por el PC se extiende hasta la dirección 3FFFh. Las regiones en color oscuro muestran regiones de memoria no implementadas físicamente o reservadas, según el modelo de microcontrolador.

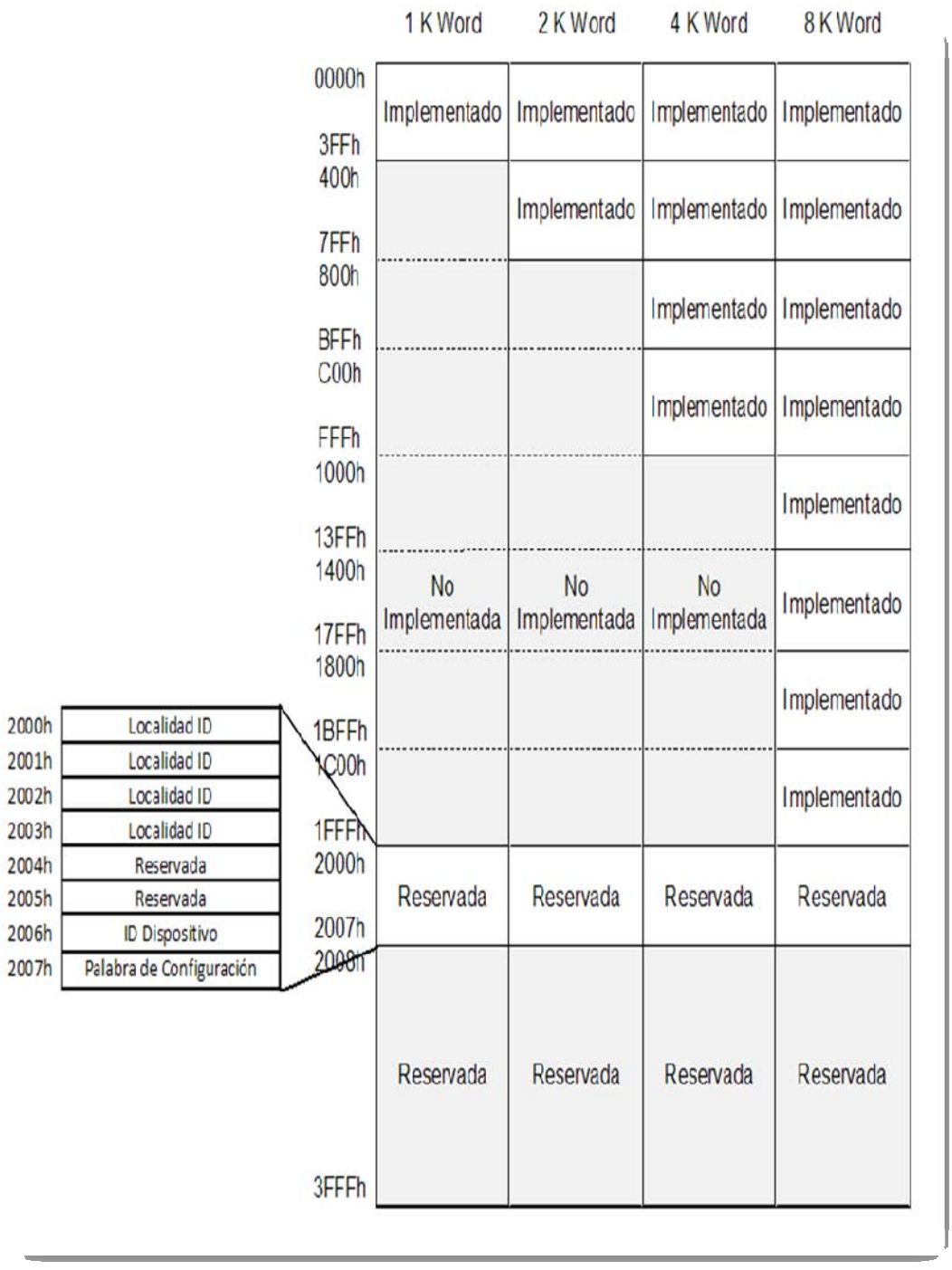


Figura 4.10. Distribución de la memoria de usuario y configuración para diferentes modelos de microcontroladores.

En el caso de un modelo 16F877A, la región de 0000h hasta 1FFFh puede ser accedida durante la operación normal del microcontrolador, es decir, un total de 8KW. Esta región será llamada de aquí en adelante memoria de usuario. La región que se encuentra posterior a la dirección 2000h marca el inicio de la memoria reservada para propósitos de configuración para todos los dispositivos de la gama media y será referida como memoria de configuración.

La memoria de configuración se compone de 8 registros. Los primeros 4 son llamados localidades ID, su uso no está definido, por lo que es usuario quien da interpretación a la información que almacena en estos registros, por ejemplo, pueden ser usados para colocar un número de serie del producto o bien indicar la versión del *firmware*. Sólo los 4 bits menos significativos de cada registro son usados para guardar el valor deseado, por lo que cada registro representa un valor en hexadecimal del 0h al Fh.

Los siguientes tres registros son marcados como reservados, aunque en algunos modelos como el 16F84A, 16F128A y 16F18F77A son usados por Microchip para colocar un valor especial que hace referencia al modelo del dispositivo. La última localidad de memoria es usada para la palabra de configuración, la implementación y uso de cada bit de este registro depende del modelo de microcontrolador, aunque por lo general es aquí donde se configuran los recursos especiales como el tipo de oscilador, habilitación del Perro Guardián, la protección de código, entre otros recursos.

Capa de comandos

Durante la operación del microcontrolador en ***modo de programación*** la terminal PGC se usa para la entrada de la señal de reloj, mientras que la terminal PGD es usada para la entrada de comandos, así como para la entrada y salida de datos de manera serie. Todos los comandos tienen una longitud de 6 bits mientras que los datos tienen una longitud de 14 para los dispositivos de familia media. La tabla 4.5 muestra los comandos del protocolo ICSP implementados en un microcontrolador PIC16F84A

Tabla 4.5. Comandos del protocolo ICSP para un microcontrolador PIC16F84A.

Comando	Descripción	HEX	MSB…LSB	Dato
Load configuration	Cargar configuración	00H	XX 0000	0, Dato, 0
Load data for program memory	Cargar dato en memoria de programa	02h	XX 0010	0, Dato, 0
Load data for data memory	Cargar dato en memoria de datos	03h	XX 0011	0, Dato, 0
Read data from program memory	Leer dato de memoria de programa	04h	XX 0100	0, Dato, 0
Read data from data memory	Leer dato de memoria de datos	05h	XX 0101	0, Dato, 0
Increment address	Incrementar dirección	06h	XX 0110	
Begin erase programming cycle	Inicio de programación con borrado	08h	00 1000	
Bulk erase program memory	Borrar memoria de programa	09h	XX 1001	
Bulk erase data memory	Borrar memoria de datos	0Bh	XX 1011	
Begin programming only cycle	Inicio de la programación	18h	01 1000	
Leyendas de la tabla: **X - No importa** **Dato – Se representan con 14 bits para la gama media**				

Algunos de los comandos tienen datos asociados y la longitud de cada dato dependerá de la gama a que el microcontrolador pertenezca. En el caso de los microcontroladores de gama media, ya que cada código de operación se representa con 14 bits, los datos de un comando del protocolo ICSP son de 14 bits. En total es necesario hacer uso de 22 bits por cada comando a transmitir, ya que a los bits de datos es necesario agregar un ***bit de inicio*** (*start bit*) y un ***bit de*** alto (*stop bit*) por cada trama. La figura 4.11 muestra la trama que se genera al enviar el comando *Load Data Program Memory.*

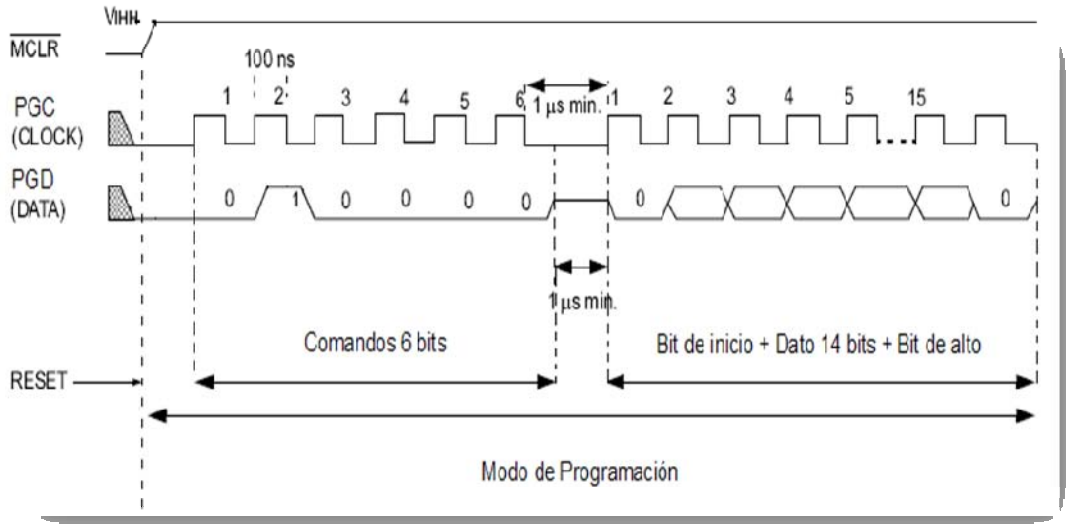


Figura 4.11. Trama que genera el comando *Load Data Program Memory*.

Una trama está formada por los 6 bits que forman el comando y por los bits de datos, donde el bit menos significativo de cada sección se transmite primero. La línea PGD es actualizada durante el flanco de subida de la señal de reloj, mientras que en el flanco de baja, el valor de la señal PGD es capturado por el microcontrolador a programar. Para enviar un comando son necesarios 6 pulsos de reloj y para cada dato son necesarios un total de 16 (14 para cada bit que forman en código de operación y dos más para el bit de inicio y para el bit de alto). Entre cada comando y su respectivo dato es necesario hacer una pausa de más de 1µS. A continuación se describen los comandos usados para programar un microcontrolador PIC18F84A.

Load Configuración (00h). Después de recibir este comando, el contador de programa (PC) apuntará a la dirección 2000h, correspondientes a la zona de la memoria de configuración. Entonces, 16 bits serán leídos a través de la terminal PGD, como se describió anteriormente, durante 16 ciclos de la señal de reloj. Para salir de la zona de la memoria de configuración es necesario aplicar un reinicio al dispositivo, es decir, será necesario poner la terminal MCLR a nivel bajo.

Una vez en la zona de la memoria de configuración será posible escribir los registros de identificación (ID), la palabra de configuración o leer el identificador de dispositivo, si el modelo de microcontrolador lo incorpora en su arquitectura. El dato enviado junto a este comando es ignorado por el microcontrolador, por lo que para escribir o leer un registro de la memoria de configuración se usan los comandos *Read Data Program Memory y Load Data Program Memory.*

Load Data for Program Memory (02h). Este comando es usado para enviar al microcontrolador el código de operación que será programado en la dirección apuntada por el contador de programa. El dato sólo es almacenado temporalmente, es decir no es cargado directamente al registro, por lo que la memoria de programa no es grabada durante la ejecución de este comando. Para iniciar el ciclo de programación será necesario enviar el comando *Begin Programming Cycle*.

Load Data for Data Memory (03h). Este comando es usado para enviar al microcontrolador el dato que será programado en un registro de la memoria EEPROM. Al igual que pasa con el comando *Load Data for Program Memory*, durante la ejecución de este comando, el dato recibido sólo es almacenado en un registro interno del microcontrolador y es hasta la llegada del comando *Begin Programming Cycle* que el registro apuntado por el PC es modificado con el nuevo valor.

Dependiendo de la capacidad de la memoria de datos EEPROM, sólo los 6, 7 u 8 bits menos significativos del PC son utilizados para poder direccionar un total de 64, 128 o 256 registros respectivamente. De igual forma, aunque el dato requerido por el comando necesita de 16 bits, sólo los primeros 8 bits que siguen al bit de inicio son tomados en cuenta para ser programados en la memoria EEPROM de datos, pero son requeridos por la arquitectura del dispositivo para sincronizar la comunicación.

Read Data from Program Memory (04h). Después de recibir este comando, el microcontrolador enviará una palabra de 16 bit, por lo que la terminal PGD será configurada como salida. Esto toma lugar en el segundo flanco de subida de la señal de reloj y al terminar de transmitirse el bit de alto esta terminal regresará a un estado de alta impedancia. El dato leído puede corresponder a la memoria de programa o a la memoria de configuración, dependiendo de si un comando *Load Configuration* fue enviado con anterioridad o no.

Read Data from Data Memory (05h). Este comando funciona de manera similar al comando *Read Data from Program Memory*. Una vez recibido, la terminal PGD es configurada como salida durante el segundo ciclo de reloj y regresa a un estado de alta impedancia después de que el bit de alto ha sido transmitido. Son transmitidos 16 bits en total, pero sólo los 8 bits más significativos (después del bit de inicio) poseen la palabra leída desde la memoria de datos. Dependiendo de la extensión de la memoria de datos EEPROM serán usados los 6, 7 u 8 bits menos significativos del contador de programa para poder direccionar un total de 64, 128 o 256 registros.

Increment Address (06h). Los registros a ser escritos o leídos, tanto de la memoria de programa, configuración o EEPROM de datos, son direccionados a través del contador de programa. Un reinicio en el dispositivo provocará que el contador de programa sea inicializado con la dirección 00h al entrar en modo de programación, por lo que para poder direccionar un registro en particular será necesario enviar tantos comandos *Increment Address* como sean necesarios. Este comando no necesita el envío de un dato.

Begin Erase Programming Cycle (08h). Este comando es usado para iniciar un ciclo de borrado de la memoria de instrucciones o de datos EEPROM, por lo que antes de enviar este comando debe haber sido enviado un comando *Bulk Erase Program Memory* o *Bulk Erase Program Memory*. Las memorias de programa y datos EEPROM deberán ser borradas antes de un ciclo de programación para garantizar su correcta escritura.

Bulk Erase Program Memory (09h). Después de ejecutar este comando es necesario enviar un comando *Begin Programming Only Cycle*, con lo que la memoria de programa del dispositivo será completamente borrada. El proceso de borrado debe seguir la siguiente secuencia:

1. Ejecutar el comando *Load Data for Program Memory* con todos los bits de datos a '1' (3FFFh).
2. Ejecutar el comando *Bulk Erase Program Memory.*
3. Ejecutar el comando *Begin Programming*.
4. Esperar por 10mS a que se complete el ciclo de borrado.

Bulk Erase Data Memory (0Bh). Este comando es similar al comando *Bulk Erase Program Memory* Después de su ejecución es necesario enviar un comando *Begin Programming Only Cycle*, con lo que la memoria de datos del dispositivo será completamente borrada. El proceso de borrado debe seguir la siguiente secuencia:

1. Ejecutar el comando *Load Data for Program Memory* con todos los bits de datos a '1' (3FFFh).
2. Ejecutar el comando *Bulk Erase Program Memory.*
3. Ejecutar el comando *Begin Programming*.
4. Esperar por 10mS a que se complete el ciclo de borrado.

Begin Programming Only Cycle (18h). Este comando es usado para escribir el dato que ha sido enviado al dispositivo a través de los comandos *Load Data for Data Memory* y *Load Data for Data Memory.* Es necesario esperar 10mS a que la operación de escritura finalice antes de enviar un nuevo comando al dispositivo. La memoria debe ser borrada antes de poder programar una palabra.

Dependiendo del modelo de microcontrolador, pueden estar presentes otros comandos, pero de manera general todos los dispositivos de la familia PIC16 implementan en su arquitectura el soporte para estos comandos.

4.1.3 Fuentes de alimentación

Todo sistema electrónico requerirá de una fuente de alimentación que provea la energía eléctrica necesaria para el funcionamiento del sistema. Los sistemas digitales usan fuentes de corriente directa que toman de pilas, baterías, celdas solares o adaptadores de corriente (eliminadores). En esta sección se describirán los conceptos que permitieron decidir el diseño de la fuente de alimentación y programación para el programador USB de microcontroladores PIC, tanto para un dispositivo *Bus-Power* como para un dispositivo *Self-Power*. De manera general, las fuentes de alimentación pueden agruparse en dos conjuntos, fuentes de alimentación lineal y fuentes de alimentación conmutadas.

Fuentes de alimentación lineal

Cuando un dispositivo toma energía proveniente de la red eléctrica, lo hace de una señal senoidal de 127V, la cual no es compatible con circuitos digitales, por lo que es necesario la transformación de los parámetros de esta señal a través de un proceso de reducción, rectificación, filtrado y regulación (figura 4.12).

Reducción. Esta etapa inicia con un trasformador reductor, encargado de reducir la amplitud de la señal senoidal proveniente de la red eléctrica. También es posible encontrar un elemento protección con fusibles en esta etapa.

Rectificación. Esta etapa es la encargada de recibir la tensión alterna proveniente de la etapa de reducción y convertirla en continua pulsatoria.

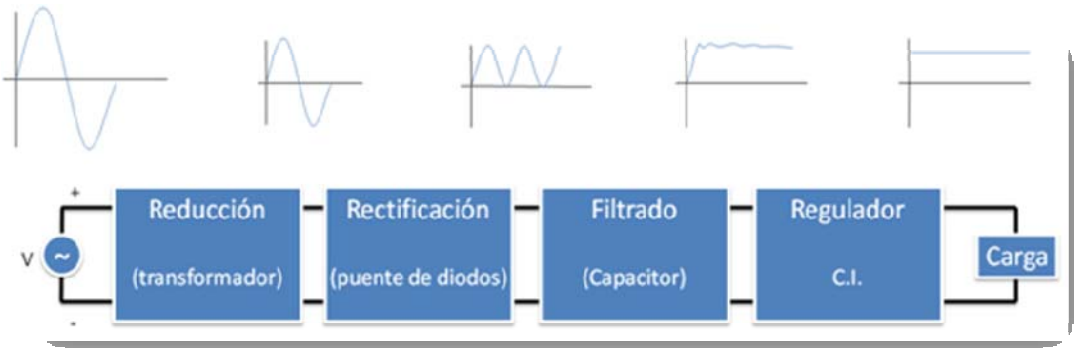


Figura 4.12. Diagrama a bloques de una fuente de alimentación lineal.

Filtrado. Esta etapa es usada para eliminar en cierto grado la componente alterna de la señal que es entregada por la etapa de rectificación. Los filtros se utilizan en rectificadores de baja potencia, dado que los elementos que conforman el filtro (bobinas y capacitores) son caros y voluminosos para potencias elevadas.

Regulador y estabilizador. Su objetivo es obtener a la salida de la fuente de alimentación los valores deseados con una estabilidad garantizada, es decir, los circuitos reguladores y estabilizadores impiden fluctuaciones del valor de la salida mayores a los del diseño de la fuente.

Un elemento que incluye las tres primeras etapas son los dispositivos llamados eliminadores de corriente. Estos dispositivos suelen usar un transformador reductor para adaptar el nivel de tensión de la red eléctrica al nivel usado por la fuente de alimentación. Además incluyen un puente de diodos, de media onda o de onda completa, usado para la etapa de rectificación de la señal y por último suelen usar un capacitor electrolítico para la etapa de filtrado.

El uso de adaptadores de corriente como parte de la fuente de alimentación del programador de microcontroladores PIC en una configuración *Self-Power* es una buena opción debido a que el circuito impreso se reduciría significativamente, comparado si todos estos elementos son puestos directamente en la tarjeta del sistema, es decir, sólo será necesario colocar sobre el circuito impreso los componentes necesarios para la etapa del regulador y estabilizador, que en este caso deberán proveer tensiones con valores de 5V y 13.5V para la alimentación del dispositivo y voltaje de programación respectivamente.

El principal inconveniente del uso de eliminadores de corriente, en cualquier sistema, es que a la salida de la etapa de filtrado la tensión puede sufrir variaciones provenientes de la red eléctrica. De igual modo, los parámetros del transformador y de los componentes utilizados en el rectificador y filtro pueden sufrir variaciones debido con las condiciones ambientales como la temperatura y humedad, lo que afecta la tensión de salida. Esto lleva a concluir que es necesario el uso de un elemento que mantenga estable el valor de la tensión de salida cuando se hace uso de un eliminador de corriente.

Existen diferentes circuitos, principalmente armados con redes de transistores, que permiten construir un regulador o estabilizador de tensión. Puede definirse al regulador como el dispositivo que se intercala entre la fuente primaria y la carga, además de que acoplándose a las características eléctricas de la primera, la adecua a la segunda. En esta sección serán tratados dos ejemplos de circuitos estabilizadores, el uso de diodos zener y de reguladores de tensión en circuitos integrados.

En ambos casos, los circuitos recortan la tensión de entrada hasta un nivel deseado con base en una tensión de referencia de alta estabilidad, por lo que la tensión de entrada debe ser en todo momento de magnitud superior al deseado en la salida. La potencia que se genera por la diferencia de tensión entre la entrada y la salida se disipa por efecto Joule, es decir en forma de calor, con lo que el circuito regulador debe disponer de un radiador o disipador de aluminio cuando la potencia es elevada.

En la figura 4.13 se observa un circuito típico de estabilización de tensión mediante el uso de un diodo zener. El diodo zener polarizado en inversa mantiene constante la tensión en sus extremos cuando su punto de trabajo está situado en la región de ruptura. En esta región, un diodo zener puede ser encontrada la corriente I_{ZT} (*test*) para la cual está definida la tensión del zener V_Z, I_{ZK} (*knee*) como la corriente en la que comienza a doblarse la curva característica y por último I_{ZM} (*max*) como la corriente máxima que puede circular por él (figura 4.14)

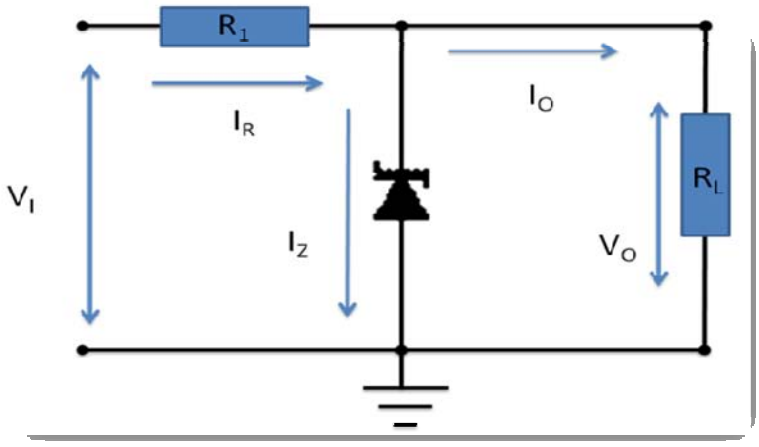


Figura 4.13. Circuito de estabilización con zener y resistencia.

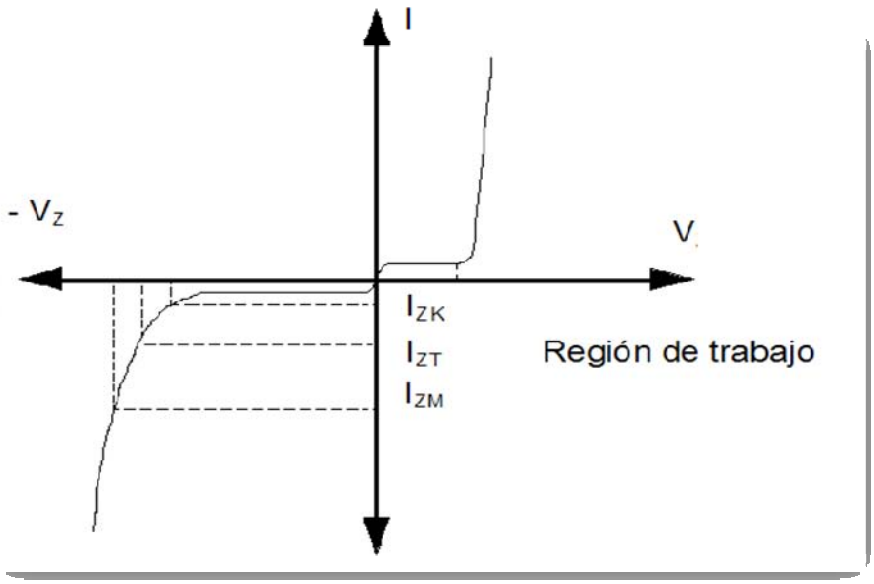


Figura 4.14. Curva característica I-V del diodo zener.

Al usar un diodo zener como estabilizador de tensión, se parte de la idea que la tensión de ruptura del zener (V_Z) es igual a la tensión con la que será alimentada la carga (Vo). Para que el punto de trabajo del diodo zener sea el adecuado, la corriente I_{Zmin} debe estar comprendida entre I_{ZK} e I_{ZM}. La figura 4.13 muestra el circuito de un estabilizador de tensión con un diodo zener y una resistencia (R_1) para una resistencia de carga (R_L).

Este circuito funciona como un divisor de voltaje, por lo que a los extremos de la resistencia R_1 se tendrá la diferencia de tensión entre la entrada y la salida (V_I-V_O). Para calcular el valor de la resistencia R_1 debe hacerse un balance de las corrientes en el circuito, de donde se obtiene que:

$$I_R = I_O + I_Z \qquad \textit{Ecuación 4.1}$$

I_R: Corriente de la resistencia R_1.
I_O: Corriente de la resistencia de carga R_L.
I_Z: Corriente del diodo I_Z.

$$I_R = (V_I - V_O)/R_1 \qquad \textit{Ecuación 4.2}$$

I_R: Corriente de la resistencia R_1.
I_O: Corriente de la resistencia de carga R_L.
V_I: Tensión de entrada.
V_O: Tensión de salida.

Las ecuaciones 4.1 y 4.2 muestran que la corriente I_R depende de la resistencia R_1 y de la diferencia de tensión entre la entrada y la salida. La corriente I_R debe ser suficiente para garantizar la corriente máxima para la carga y al menos la corriente mínima para el diodo zener, así se obtiene que:

$$I_{Rmin} = I_{Omax} + I_{Zmin} \qquad \textit{Ecuación 4.3}$$

I_{Rmin}: Corriente máxima de la resistencia
I_{Omax}: Corriente máxima
I_{Zmin}: Corriente mínima

La ecuación 4.4 determina el valor máximo que puede tomar la resistencia R_1.

$$R_{1max} = \frac{(V_I - V_O)_{min}}{I_{Omax} + I_{Zmin}} \qquad \textit{Ecuación 4.4}$$

R_{1max}: Valor de la resistencia R_1.
I_{Omax}: Corriente máxima de salida.
I_{Zmin}: Corriente mínima del diodo.
V_I: Tensión de entrada.
V_O: Tensión de salida.

Si la resistencia R_1 tiene un valor menor al calculado, dará por consecuencia que la corriente que pasa por ella aumente y por tanto la potencia que ésta disipa. Además, como la corriente I_R no depende de la carga, el diodo zener debe absorber la corriente que la carga no ocupa. Lo que lleva a que es necesario elegir un dispositivo que sea capaz de soportar este exceso de corriente. Las aplicaciones donde se hace uso de este circuito deben requerir niveles bajos de corriente, ya que será el diodo zener quien consuma la corriente cuando la resistencia de la carga aumenta o sea retirada.

La ecuación 4.5 define la corriente que paso por el diodo zener cuando el circuito no tiene carga.

$$I_{Zmax} = \frac{(V_I - V_O)_{max}}{R1}$$ *Ecuación 4.5*

R_1: Valor de la resistencia R_1.
I_{Omax}: Corriente máxima de salida.
V_i: Tensión de entrada.
V_O: Tensión de salida.

El uso de transistores en el diseño de fuentes de alimentación reguladas es un estudio extenso. La evolución de los circuitos integrados ha permitido la integración de redes de transistores encargadas de la etapa de regulación en circuitos integrados a los que sólo se conectan algunos elementos como resistencias y capacitores para fijar la tensión de salida, filtrar la señal de entrada y salida del ruido de rizo o diodos para la protección frente a corrientes inversas. Estos reguladores pueden dividirse en dos grupos, reguladores de tensión fija y de tensión variable.

Los reguladores de tensión fija, como su nombre lo indica, mantienen a su salida un nivel fijo de tensión positivo o negativo. Cada fabricante agrega diferentes prestaciones a sus dispositivos, pero de manera general, sus principales características se enumeran como:

1. No requieren de ningún ajuste. Están diseñados para proporcionar una tensión constante, definida en un rango normalizado como 2, 5, 6, 8, 10, 12, 15, 18 ó 24V.
2. Los valores de tensión de salida son positivos.
3. Los niveles de corriente nominales de salida están normalizadas (100mA, 500mA, 1A, 1.5A)
4. Su encapsulado es de plástico en forma de transistor.

La figura 4.15 muestra el diagrama de una fuente diseñada con un regulador fijo y una imagen del modelo 7805 en un encapsulado de plástico de tres terminales.

Los reguladores lineales operan con corriente continua a la entrada, que es de un nivel siempre superior a la de la salida deseada y equivalente a una resistencia cuyo valor se ajusta automáticamente. De igual forma que el circuito estabilizador con diodo zener, disipan la potencia que la carga no consume en forma de calor, por lo que su rendimiento energético es siempre inferior a la unidad. Un regulador electrónico es un dispositivo activo, que se ajustará de forma pertinente para que la tensión de salida permanezca estable. Esto lo consigue al ajustar la salida con base en la comparación de la tensión de salida con una referencia fija de buena estabilidad.

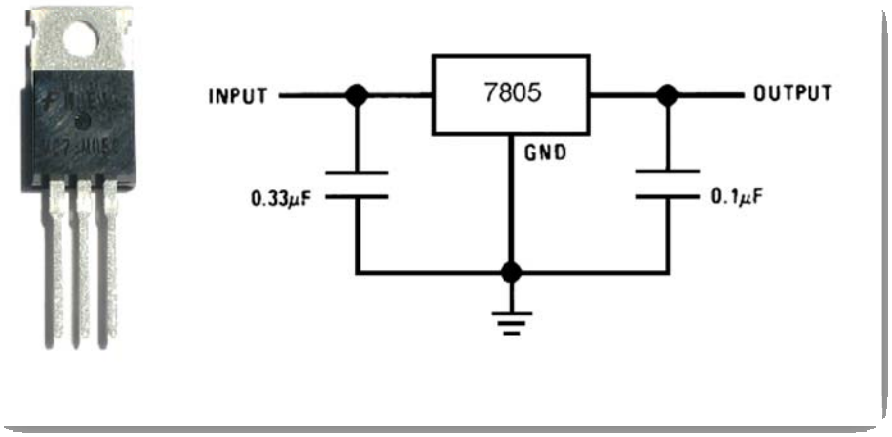


Figura 4.15. Regulador 7805 y diagrama de eléctrico de un ejemplo de conexión.

Fuentes de alimentación conmutadas

Los reguladores conmutados funcionan de una forma diferente a los reguladores lineales. En las fuentes conmutadas, con el objetivo de no desperdiciar energía en forma de calor, se hace uso de un parámetro no eléctrico, es decir, el tiempo. Como su nombre indica, su principio de funcionamiento está basado en un interruptor que realiza cambios muy rápidos entre sus estados, para permitir e impedir el suministro de corriente proveniente de la fuente primaria. Al variar la frecuencia de conmutación se permite que el valor medio de la energía a la salida del regulador coincida con las necesidades de la carga, lo que reduce la pérdida de energía en forma de calor al grado de que el rendimiento teórico se aproxima a la unidad.

Entre las desventajas de los reguladores conmutados se pueden enumerar:

1. La respuesta frente a cambios en la entrada es más lenta que en los reguladores lineales.
2. Tanto en la fuente primaria como en la carga existen señales de ruido a la frecuencia de conmutación, las cuales son difíciles de eliminar.

Existen diferentes arreglos para poder construir fuentes conmutadas, que permitan elevar la tensión de una señal de corriente directa (DC) a una magnitud mayor, para ser usadas en la implementación de la fuente de poder del programador USB de microcontroladores PIC para una configuración *Bus- Power,* donde a partir de los 5V a 100 mA que proporciona el USB deberá generarse un voltaje de programación de 13.5V a 20mA. En esta sección sólo será tratada la teoría de dos de estos circuitos por ser las más fáciles de implementar en este sistema, las fuentes *Boost* y los multiplicadores de voltaje.

En la figura 4.16 puede verse el circuito correspondiente a una fuente conmutada de ***convertidor directo***, cuyo funcionamiento es el siguiente:

1. Cuando el interruptor está cerrado, el diodo es polarizado inversamente por lo que la fuente primaria aplica tensión a la carga R_1. Mientras tanto, en la bobina se almacena energía debido a la corriente que circula por ella.

2. Cuando el interruptor está abierto, la corriente que fluye por la bobina se detiene pero al mismo tiempo aparece en la bobina un f.e.m inducida con una polaridad positiva que impide su descarga inmediata, pasando esta energía a través de la resistencia de carga y del diodo.

La tensión a la salida de la fuente equivale a la medida de la tensión obtenida por la aplicación de la señal pulsante. La ecuación 4.6 muestra la relación entre la el voltaje de entrada u el voltaje de salida:

$$t_c \cdot V_i = V_O.T \qquad \textit{Ecuación 4.6}$$

De esta ecuación puede comentarse que $V_O = \delta V_i$, donde δ es la relación entre el tiempo de conducción del conmutador y el periodo. El valor de la salida V_O es controlado a través de variar el valor de δ desde 0 hasta 1, es decir, el tiempo en que el interruptor conduce respecto al periodo, la tensión de salida varía entre 0 y la tensión de entrada V_i, por lo que la tensión máxima de salida que se puede obtener es igual a la tensión de entrada proveniente de la fuente primaria.

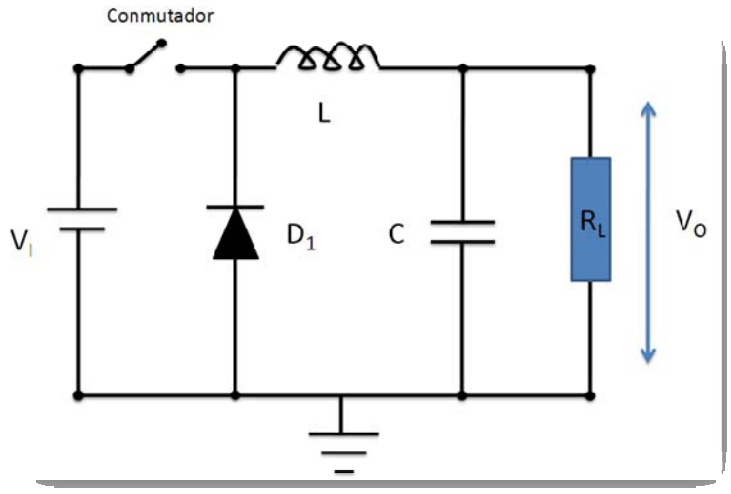


Figura 4.16. Fuente conmutada de convertidor directo.

Otro modelo de regulador conmutado es el ***circuito indirecto o inverso*** cuyo funcionamiento es totalmente diferente al explicado para los reguladores conmutados directos (figura 4.17).

En este modelo ocurrirá lo siguiente:

1. Cuando el interruptor esté cerrado, el diodo se polarizara inversamente, lo que desconectará la carga de la fuente primaria V_i.la bobina absorbe entonces energía de la fuente primaria.
2. Al abrir el interruptor, la bobina produce una f.e.m. inducida, que se opone a que la intensidad de corriente que fluye por ella se anule, lo que polariza a la bobina con su polo positivo en la parte inferior.

Para un modelo indirecto, el valor de la bobina debe ser mayor al usado en un modelo directo, ya que debe almacenar la mayor cantidad de energía y no sólo una parte de la que le es suministrada desde la fuente primaria. Se puede demostrar que la relación de la tensión de la fuente primaria (V_i) y de la de salida del convertidor es:

$$\frac{V_o}{V_i} = \frac{\delta}{1-\delta}$$ *Ecuación 4.7*

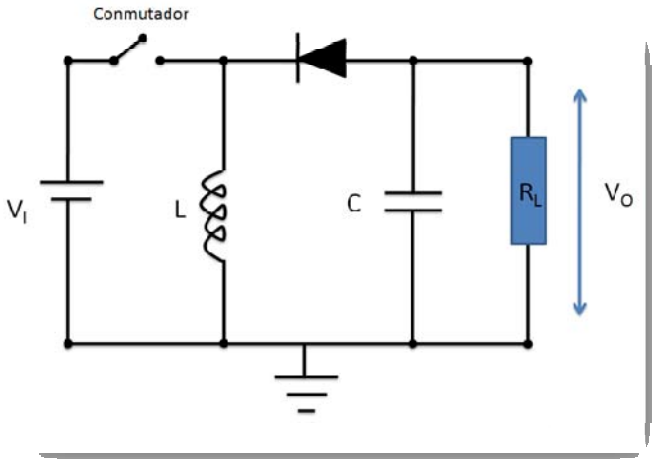


Figura 4.17. Fuente conmutada de circuito inverso.

Representando gráficamente esta igualdad se obtiene la gráfica de la figura 4.18 en la que se aprecia que la tensión de salida no solamente iguala a la de entrada cuando δ es igual a 0.5, sino que si δ aumenta, teóricamente la tensión de salida aumentará hasta un valor infinito cuando δ es igual a la unidad, es evidente que podrán obtenerse tensiones muy superiores a la de entrada variando solamente el valor de δ y regularlas adecuadamente mediante su control.

Tanto en las fuentes de convertidor directo e inverso, durante su funcionamiento normal no debe interrumpirse la corriente que circula por la bobina, ya que de lo contrario la tensión de salida se haría cero y el funcionamiento sería incorrecto. Algo similar ocurre si la carga no existe, la energía almacenada en la bobina hace que la tensión de salida aumente con lo que podría deteriorarse el circuito.

Los convertidores estudiados anteriormente presentas los siguientes inconvenientes:

1. El convertidor directo no puede entregar a la carga una tensión mayor que la de la fuente primaria.
2. El convertidor inverso o indirecto invierte la polaridad de la tensión de salida respecto de la de entrada y además la máxima tensión de salida está limitada por las características del conmutador.

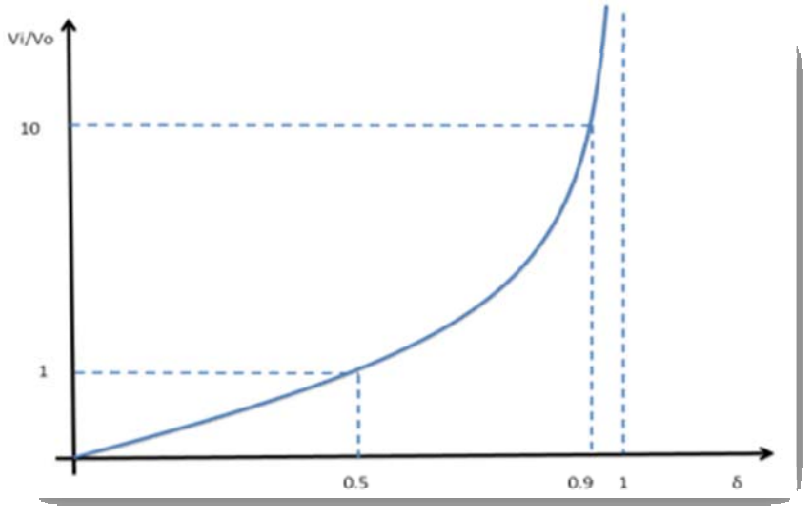


Figura 4.18. Relación de la tensión de entrada respecto a la tensión de salida.

4.2 Características del firmware

El sistema de control para el programador USB de microcontroladores PIC se implementará con un microcontrolador de 8 bits. Sus atribuciones como controlador serán las de manejar tanto el protocolo de comunicaciones USB como el protocolo ICSP proporcionado por Microchip para programar sus familias de microcontroladores; así como la lógica de negociación con la interfaz de usuario.

Como se mencionó en el capítulo 1, podría afirmarse que cada uno de los fabricantes de circuitos integrados tiene una línea de microcontroladores, por lo que existen diferentes opciones aún dentro de un mismo fabricante para implementar el sistema de control del programador USB de microcontroladores PIC.

Las características y periféricos con que debe contar un dispositivo para poder ser usado en este proyecto son:

- Módulo Convertidor Analógico a Digital de 8 a 10 bits de resolución.
- Módulo Generador de Pulsos de Ancho Modulado (PWM)
- Puerto de comunicaciones USB interno, con soporte para *Low Speed*, Regulador de Voltaje interno y configuración *Self-Power*.
- Memoria de programa tipo FLASH y con soporte para actualizar su firmware a través del programa.
- UART con soporte para RS232C a 9600 baudios.
- 8 Terminales digitales de entrada y salida.

Como puede apreciarse, los recursos necesarios pueden encontrarse en muchos modelos de microcontroladores, pero si se toman en cuenta los comentado realizados en los capítulos 2 y 3 respecto a los factores que influyen en la selección y elección del fabricante y modelo de dispositivo a usar en una aplicación, podrá reducirse la lista significativamente. Los criterios usados para la selección del fabricante y del dispositivo son:

- El fabricante provee documentación técnica sobre el dispositivo, así como notas de aplicación, código fuente o firmwares para explicar el uso de los dispositivos y sus periféricos.
- Se tiene acceso a herramientas en hardware y software que permitan el diseño, desarrollo e implementación del prototipo.
- Se cuenta con experiencia en el desarrollo de proyectos con la familia de microcontroladores seleccionada.

Al tomar en cuenta estos criterios, la lista de fabricantes se reduce a dos opciones, hacer uso de microcontroladores AVR de ATMEL o de microcontroladores PIC de Microchip. Ambos fabricantes comparten características en su arquitectura, como un procesador Harvard-RISC de 8 bits, y herramientas en software que son distribuidas en de manera gratuita en el sitio de internet del fabricante para la captura, compilación, depuración y simulación de proyectos. De igual forma, es posible encontrar herramientas diseñadas por terceros que son fáciles de construir con un presupuesto reducido, como programadores y tarjeta entrenadoras.

Por lo tanto, con base en los criterios y requerimientos descritos, se ha optado por elegir los microcontroladores PIC de Microchip para implementar el sistema de control del programador USB de microcontroladores PIC, ya que se cuenta con un programador para dispositivos de este fabricante y experiencia previa en el uso de esta familia de dispositivos, además de que es posible obtener muestras gratuitas, lo que reducirá el costo del desarrollo del prototipo.

Para elegir el modelo de microcontrolador a usar será necesario tener terminado el firmware, ya que el tamaño de la memoria de programa y datos no se será conocido hasta que una versión estable y funcional del firmware esté terminada. Los microcontroladores PIC de la familia PIC18F2X50 cumplen con las características requeridas por la aplicación, y será a partir del modelo PIC18F2550 que se desarrollará el primer prototipo. El capítulo 5, donde se presentarán los resultados obtenidos, se comentarán los modelos que mejor cumplen con la aplicación.

4.2.1 El microcontrolador PIC18F2550

El microcontrolador PIC18F2550 pertenece a la Gama Mejorada cuyas características fueron descritas en el capítulo 1. Este microcontrolador pertenece a una subfamilia formada por 4 dispositivos, los microcontroladores PIC18F2450, PIC18F2550, PIC18F4450 y PIC18F4550.

Esta subfamilia de dispositivos, en especial el microcontrolador PIC18F2550, fueron dotados de una serie de características diseñadas para disminuir el consumo de potencia durante su operación (*Power-Managed Modes*), como son:

- **Modos alternativos de operación**. La fuente de reloj usada para alimentar el oscilador principal puede ser tomada de un oscilador conectado al TIMER1 o de un oscilador interno para modos de baja velocidad. Además de poder dividir o multiplicar la frecuencia la frecuencia de la señal de reloj para asignar al CPU y al módulo USB velocidades de reloj diferentes.
- **Múltiples modos de inactividad**. Debido al diseño del oscilador principal, el núcleo o CPU del dispositivo puede estar detenido mientras los periféricos están activos, o viceversa; y entrar en un modo de bajo consumo (SLEEP mode). Para tal fin, el microcontrolador posee un sistema de cambio al vuelo (On-the-fly) que permite el uso del administrador de energía a través del programa, lo que permite el diseño de rutinas de ahorro de energía.
- **Bajo consumo en módulos clave**. Los requerimientos de energía de los módulos TIMER1 y Perro Guardián ha sido minimizados.

Entre los periféricos que incorpora se pueden enumerar puertos de comunicaciones digitales de alta corriente (25 mA), distribuidos en 5 puertos; un módulo de comunicación USB bajo la versión 2.0; cuatro temporizadores de 8 o 16 bits, enumerados como TIMER0 a TIMER3; dos módulos CCP (módulos de captura, comparación y PWM), que trabajan junto a los temporizadores para poder realizar su función; puerto maestro serie de comunicaciones (MSSP), 13 canales de entrada a un convertidor analógico digital (ADC) de 10 bits, con tiempo de adquisición programable y dos comparadores analógicos dobles con entrada multiplexada.

En núcleo de la UCP se alimenta a partir de un regulador interno de 3.3V, lo que permite un ahorro de energía, separando los niveles de tensión del núcleo de los periféricos, como los puertos digitales, que necesitan un nivel de 0V a 5V para ser compatibles, por ejemplo, con sistemas TTL.

Po último, posee multiplicador hardware de 8x8 bits. Soporta el protocolo ICSP para su programación e ICD para la depuración del programa. Su arquitectura está optimizada para el compilador C18 de lenguaje C e incorpora una memoria de datos EEPROM que no forma parte del rango de direcciones de la RAM, en su lugar para su acceso y control se hace uso de registros SFR en la memoria de RAM, como si se tratara de cualquier otro periférico.

4.2.2 Organización de la memoria de programa (Program Memory)

Los microcontroladores PIC18 implementan un contador de programa (PC) de 21 bits con lo que es posible direccionar hasta 2048 registros o palabras (*words*) de la memoria de programa. Con dispositivos que no implementan una cantidad menor de memoria de la que es posible direccionar, la lectura de un registro que se encuentre en el espacio entre la última dirección implementada y la posición más alta del PC dará como resultado una instrucción NOP, es decir, se leerá un código de operación 0000h. El microcontrolador PIC18F2550 tiene 32KB de memoria FLASH y puede almacenar 16384 registros o instrucciones simples, ya que cada código de operación ocupa 16 bits para ser representada.

Las instrucciones son guardadas en dos o cuatro bytes en la memoria de programa, los bytes menos significativos son guardados en las direcciones par, es decir las direcciones que tienen definido a ‘0’ su bit menos significado. Por lo que para ejecutar una instrucción, el contador de programa debe incrementarse en pasos de 2 en 2, siendo siempre su bit menos significativo 0.

Los dispositivos PIC18 tienen dos vectores de interrupción (Interruption Vector), lo que permite un diseño de rutinas para servicio de interrupción con prioridad. El vector de inicio (Reset Vector) tienen la dirección 0000h y los vectores de interrupción 0008h y 0018 respectivamente. La figura 4.19 muestra el mapa de la memoria de programa para el microcontrolador PIC18F2550.

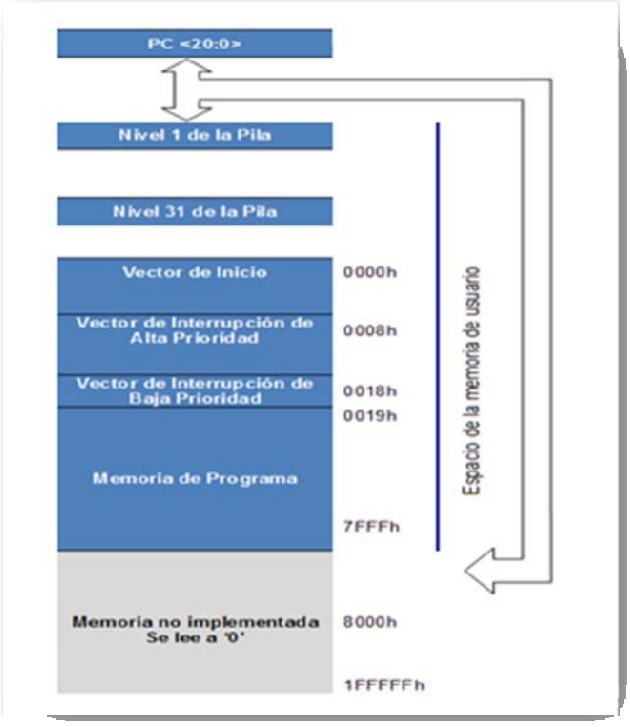


Figura 4.19. Mapa de memoria de programa del microcontrolador PIC18F2550.

4.2.3 Organización de la memoria de datos (Data RAM)

La memoria de datos en los dispositivos PIC18 está implementada como RAM estática. Cada registro de datos tiene 8 bits de longitud y una dirección de 12 bits, permitiendo direccionar hasta 4096 bytes. El total de la memoria de datos para el modelos PIC18F2550 está dividido en 16 bancos que contienen 256 bytes cada uno, de los cuales, sólo están implementados los primeros 8, para un total de 2048 bytes (figura 4.20).

La memoria de datos contiene registros con función específica (SFR o *Special Function Registers*) y registros de propósito general (GPR o *General Purpose Register*), los SFR son usados para el control y estado del núcleo del microcontrolador y de las funciones de los recursos periféricos., mientras que los GPR son usados para almacenar de manera temporal resultados derivados de la ejecución del programa. Cualquier lectura realizada sobre una localidad no implementada dará por resultado una lectura de todos como ‘0’.

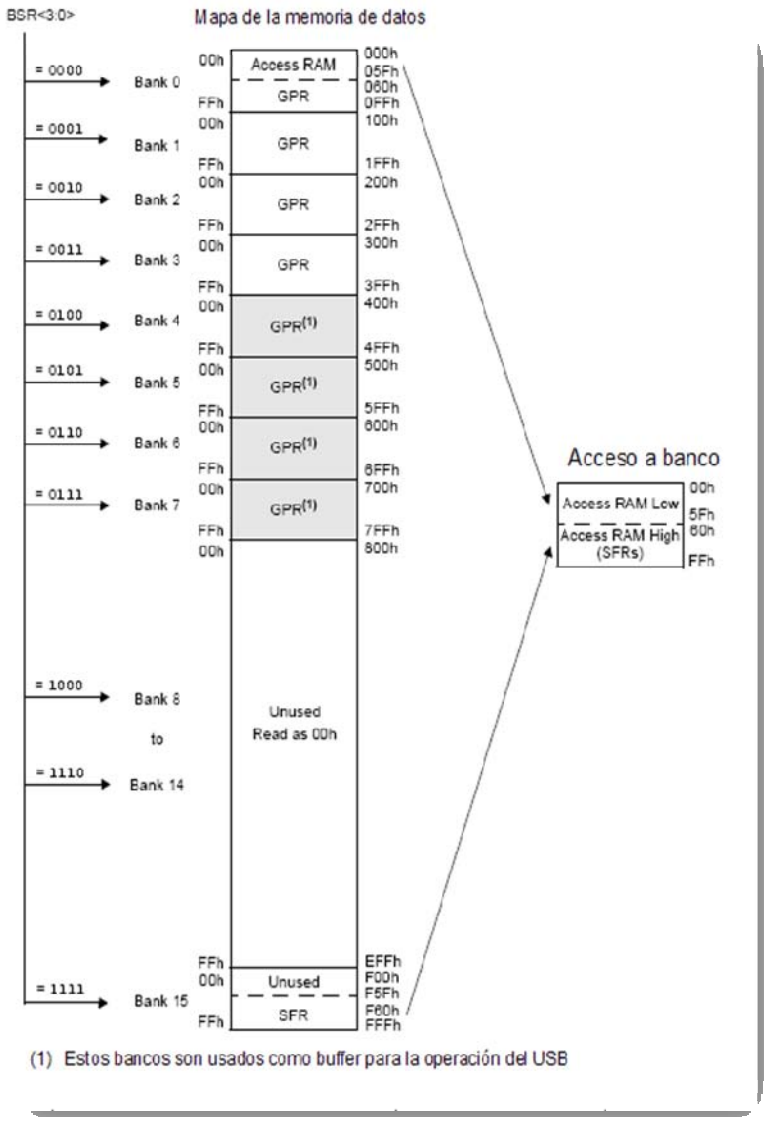


Figura 4.20. Mapa de la memoria de datos del microcontrolador PIC18F2550.

El juego de instrucciones y la arquitectura permiten operaciones entre los diferentes bancos, a través del uso de direccionamiento directo, indirecto o indexado. Además, los dispositivos de la familia PIC18 incluyen un nuevo modo de direccionamiento llamado direccionamiento por acceso a banco (BSR o Bank Select Register), con lo que el posible acceder de manera rápida una sección de 256 bytes, formada por los registro SFR y la porción menos significativa de los GPR del banco 0.

Los SFR se encuentran en las direcciones más altas de la memoria de datos implementada, en decir, en lo alto del banco 15. Los SRF están clasificados en dos secciones, los asociados con las funciones del núcleo de la UCP (ALU, reinicio e interrupciones) y los relacionados con la función de los periféricos.

4.2.4 Oscilador Principal

Es oscilador principal es un recurso de la arquitectura encargada de distribuir la señal de reloj al núcleo de la UCP y dispositivos periféricos que la requieran, como son puertos de comunicaciones serie, convertidor analógico digital, temporizadores, entre otros.

El oscilador forma parte del módulo USB en el dispositivo PIC18F2550, a diferencia de otros modelos que no incorporan este recurso, lo que permite que el núcleo de la UCP pueda trabajar con un oscilador secundario, al igual que los recursos periféricos, mientras el oscilador principal general frecuencias de 6MHz o 48MHz para los modos *Low Speed* y *High Speed* respectivamente.

- Todos los dispositivos de la familia PIC18F2550 ofrecen 12 tipos diferentes de oscilador, descritos en la tabla 4.6, pero que pueden ser resumidos de la siguiente manera:
- Cuatro tipos de reloj externo. Esta configuración ofrece la opción de usar dos terminales del microcontrolador para conectar el oscilador, una para entrada y otra para salida con la cuarta parte de la frecuencia de reloj; o una sola terminal, dejando libre la segunda para ser usada como puerto de entrada y salida digital.
- Cuatro tipos de cristal. Esta configuración permite conectar diferentes tipos de osciladores cerámicos y cristales de cuarzo.
- Dos osciladores internos. Posee dos osciladores interno RC, donde el primero provee una de reloj de 8 MHz, mientras el segundo puede ser configurado para producir una frecuencia en el rango de 125kHz a 4MHz, en 6 escalas.
- Un módulo de multiplicador de frecuencia (*PLL Phase Lock Loop*) disponible para los modos *High Speed Crystal* y Osciladores externos, con lo que es posible obtener a partir de una frecuencia menor un rango de 4MHz a 48MHz.

Tabla 4.6. Tipos de oscilador soportados por el microcontrolador PIC18F2550.

Tipo de oscilador	Descripción	Frecuencia típica de operación
XT	Cristal o resonador cerámico estándar.	4MHz
XTPLL	Cristal o resonador estándar con Módulo PLL habilitado.	4MHz
HS	Cristal o resonador cerámico de alta velocidad	4MHz- 8MHz-20MHz
HSPLL	Cristal o resonador cerámico de alta velocidad con Módulo PLL habilitado.	4MHz- 8MHz-20MHz
EC	Reloj externo con salida de Fosc/4 en la terminal CLKO.	4MHz- 8MHz-20MHz
ECIO	Reloj externo con salida de Fosc/4 con configuración de RA6 como puerto digital.	4MHz- 8MHz-20MHz
ECPLL	Reloj externo con salida de Fosc/4 con Módulo PLL habilitado y con salida de Fosc/4 en la terminal CLKO.	4MHz- 8MHz-20MHz
ECPIO	Reloj externo con salida de Fosc/4 con Módulo PLL habilitado y con configuración de RA6 como puerto digital.	4MHz- 8MHz-20MHz
INTHS	Oscilador HS interno usado también por el módulo USB.	8MHz
INTXT	Oscilador XT interno usado también por el módulo USB.	31KHz a 8Mhz
INTIO	Oscilador EC interno usado también por el módulo USB y que permite la configuración de RA6 como puerto digital.	31KHz a 8Mhz
INTCKO	Oscilador EC interno que usado también por el módulo USB y con salida de Fosc/4 en la terminal CLKO.	31KHz a 8Mhz

La operación del oscilador es controlada a través de dos registros de configuración y dos de control. Los registros de configuración CONFIGL1 y CONFIG1H, son usados para configurar el modo del oscilador y las opciones del valor de la preescala del USB. Estos registros sólo pueden ser accedidos mediante el protocolo ICSP durante la reprogramación del dispositivo y nunca desde el programa de usuario, pues se encuentran en una zona especial de la memoria de programa inaccesible desde la operación normal del microcontrolador.

El registro OSCON selecciona el modo de reloj activo y es usado para controlad el cambio en el control de reloj en los modos de administración de energía. El registro OSCTU_NE es usado para configurar la frecuencia del oscilador interno INTRC, como puede ser, seleccionar la fuente de reloj de baja frecuencia u otro oscilador secundario.

Un recurso asociado al oscilador principal es el *Fail-safe Clock Monitor* o Monitor del Reloj a Prueba de Fallos. Esta opción consiste en monitorear la fuente de reloj principal, si un erro ocurre en esta fuente de reloj, la señal que alimenta al núcleo de la UCP es cambiada a un oscilador interno, permitiendo que se continúe trabajando a baja velocidad o entra en una rutina de apagado del sistema.

Otro recurso usado por el oscilador es el *Two-Speed Start Up*, o doble velocidad de Arranque. Esta opción permite que el oscilador sirva como fuente de reloj para el *Power-on Reset*[4], hasta que la fuente de reloj principal esté completamente disponible.

Uno de los recursos más interesantes del microcontrolador usado por el oscilador principal es el circuito *Phase Locked Loop* (PLL). Este recurso se emplea principalmente en aplicaciones que requieren el uso del módulo USB, pero que hacen uso de un cristal con una frecuencia entre 4MHz a 20Mhz, insuficiente para poder operar el módulo USB en modo *High Speed*, pero que puede ser usada como una fuente de reloj para el núcleo de la UCP y de los periféricos. La figura 4.21 muestra la es diagrama a bloques de la arquitectura del oscilador del microcontrolador PIC18F2550.

[4] Este recurso periférico fue comentado más ampliamente en el capítulo 1. Su función principal es iniciar un temporizador que no permitirá el inicio de la ejecución del programa hasta que éste se desborde, lo que permite que la señal de reloj sea lo más estable posible.

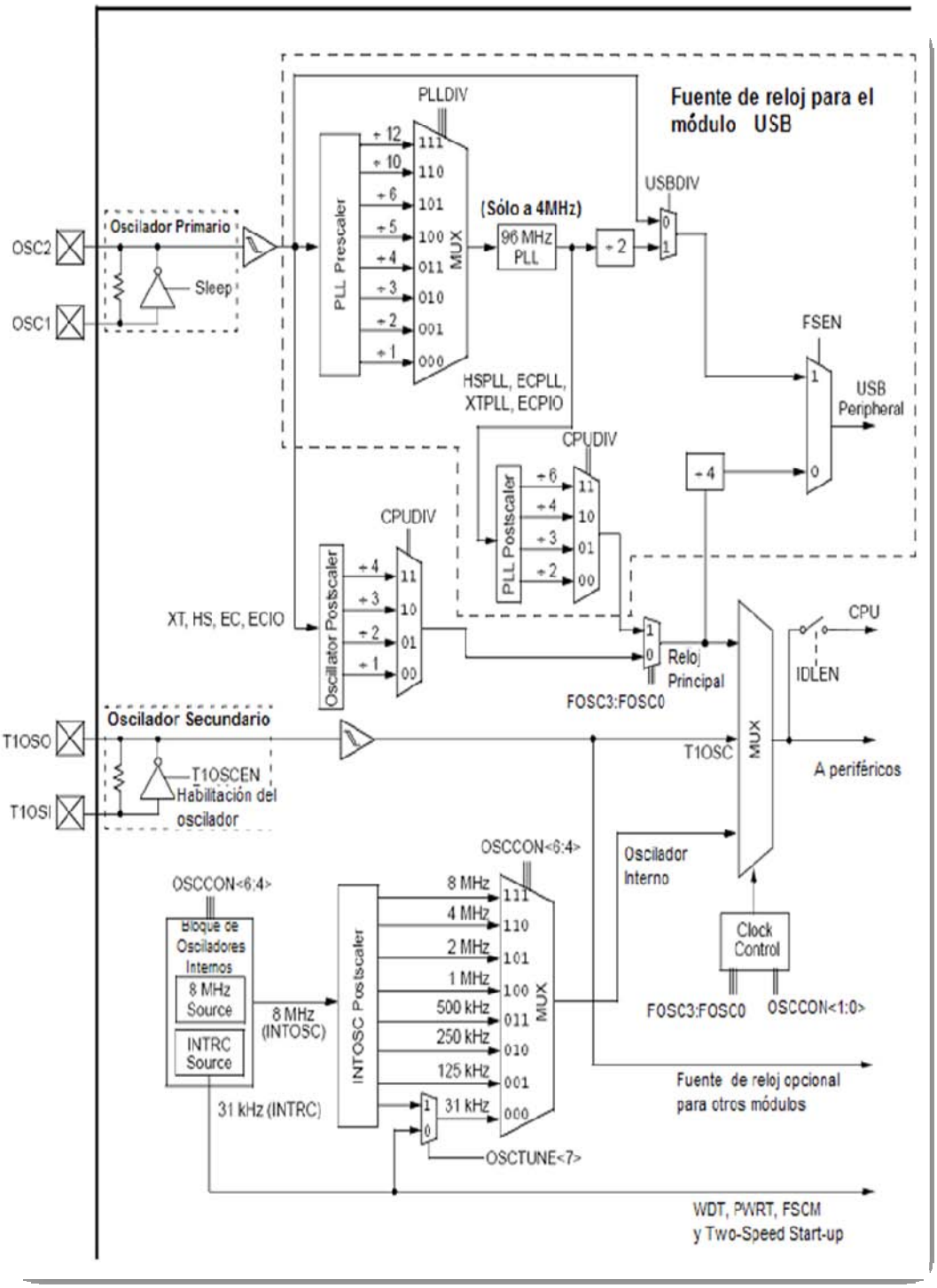


Figura 4.21. Diagrama a bloques del circuito Oscilador del microcontrolador PIC18F2550.

El circuito PLL está diseñado para producir una frecuencia fija de 96MHz a partir de una señal de reloj de 4MHz. La salida puede ser dividida para ser sincronizar el módulo USB y el núcleo del microcontrolador. El módulo PLL tiene 8 opciones de preescala para dividir la entrada de la frecuencia de oscilación. También incorpora una posescala con opciones para derivar el reloj del microcontrolador al PLL, esto permite al periférico USB y microcontrolador usar la misma entra del oscilador y aun operar a diferentes velocidades de reloj. El PLL es habilitado cuando se selecciona uno de los siguientes tipos de reloj: HSPLL, XTPLL, ECPLL o ECPIO.

4.2.5 Módulo Convertidor Analógico Digital

El microcontrolador PIC18F2550 incorpora un módulo convertidor analógico digital de 10 bits de resolución que trabaja mediante la técnica de aproximaciones sucesivas. Las entradas al CAD están multiplexadas y suman un total de 10. Para la operación y control del módulo CAD se hace uso de 5 registros de la tabla 4.7.

Los niveles para la referencia de voltaje positivo y negativo son seleccionables a través del programa, para tomarse de una referencia interna o externe a través de las terminales RA3/AN3/VREF+ y RA2/AN2/VREF-/CVREF.

El convertidor A/D está diseñado para estar disponible mientras el dispositivo se encuentra en estado de bajo consumo (SLEEP *mode*). Al operar en modo de bajo consumo, la fuente de reloj del convertidor A/D debe provenir del oscilador RC interno del mismo módulo.

Cuando ocurre un reinicio en el dispositivo, el contenido de los registros del módulo A/D son llevados a un estado conocido, que provoca el apagado del módulo A/D. Cada terminal asociada con el convertidor analógico digital puede ser configurada de manera individual con entrada analógica o como un puerto digital. El contenido de los registros ADRESH y ADRESL, concatenan el resultado de un proceso de conversión. La figura 4.22 muestra un diagrama del módulo A/D.

Tabla 4.7. Registro de configuración y estado del módulo Convertidor Analógico Digital del microcontrolador PIC18F2550.

A/D Control Register 0	Controla la operación del módulo.
A/D Control Register 1	Configura las funciones de las terminales.
A/D Control Register 2	Configura la fuente de reloj, el tiempo de adquisición y su resolución a 8 o 10 bits.
A/D Result High Register	Sección alta de la conversión
A/D Result Low Register	Sección baja de la conversión

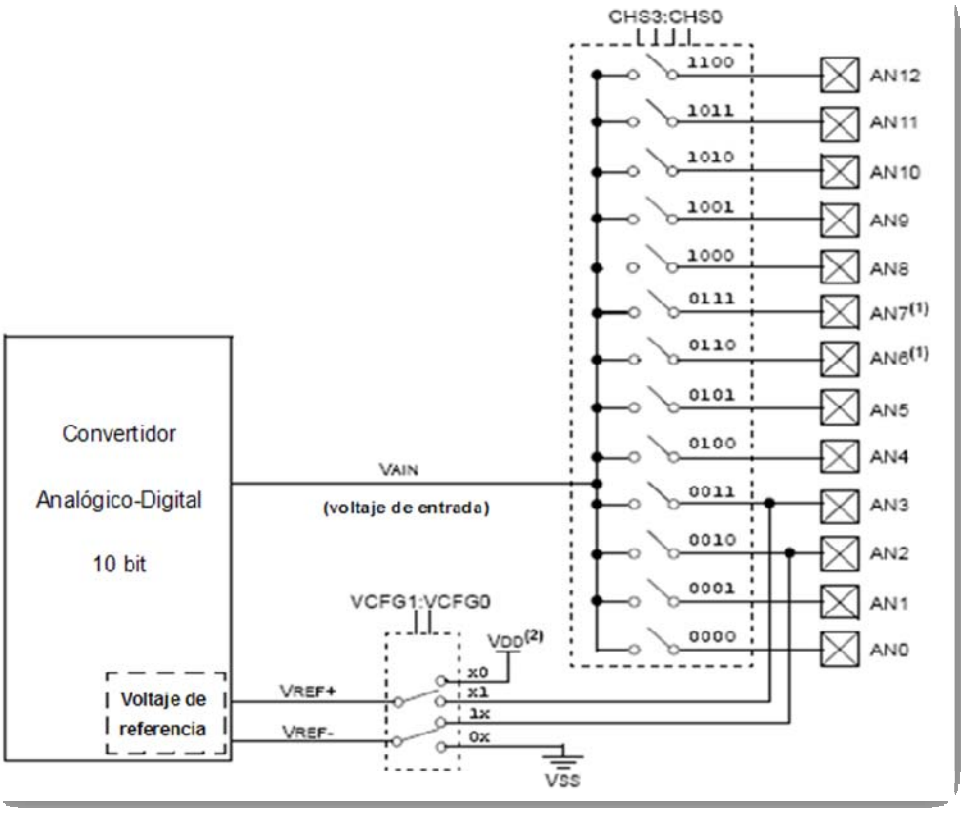


Figura 4.22. Diagrama a bloques del convertidor analógico digital.

Un proceso de configuración y operación del módulo A/D podría ser el siguiente:

1. Configurar el módulo A/D.
    a. Configurar las terminales analógicas, los voltajes de referencia y las terminales digitales en el registro ADCON1.
    b. Seleccionar el canal analógico a leer en el registro ADCON0.
    a. Seleccionar el tiempo de adquisición y la fuente de reloj en el registro ADCON2.
    b. Encender el módulo A/D en el registro ADCON0.
2. Configurar interrupciones si son requeridas.
    a. Borrar la bandera de interrupción del módulo A/D, es decir, poner a '0' el bit ADIF del registro PIR1.
    b. Habilitar las interrupciones del módulo A/D, es decir, poner a '1' el bit ADIE del registro PIE1
    c. Habilitar las interrupciones globales, es decir, poner a '1' el bit GIE del registro OPTION.
3. Esperar el tiempo requerido por una adquisición.
    a. Iniciar una conversión, es decir, poner a '1' el bit GO/DONE
    b. Esperar a que la conversión se realice y el bit GO/DONE regrese a '0' o se active una interrupción.
4. Leer los registros de resultado.
    a. Leer el contenido de los registros ADRESH y ADRESL.
    b. Limpiar la bandera ADIF si es necesario.
5. Iniciar una nueva conversión.
    a. Ir al paso 2.

4.2.6 Módulo TIMER 2

Un temporizador es un recurso periférico que incorpora un registro que tiene la capacidad de autoincrementarse o decrementarse con base en un flanco de bajada o de subida. Si este flanco es proporcionado por una señal de reloj el temporizador se incremente de manera periódica, a intervalos de tiempo bien conocidos por lo que es posible que sea usado para medir el tiempo transcurrido desde un estado a otro. Cuando la señal ocurre a intervalos de tiempo no regulares, puede ser usado para contar eventos.

En el caso del microcontrolador PIC18F2550, se dispone de 4 temporizadores, de los cuales, se describirá el TIMER 2 ya que será usado junto al módulo CCP para generar una señal de PWM.

El TIMER 2 es un registro de 8 bits que se autoincrementa hasta coincidir con el contenido de un registro de periodo PR2. Cuando el contenido de ambos registros coinciden, el registros TRM2 se reinicia 00h y empieza una nueva cuenta. El módulo TIMER 2 incorpora las siguientes características:

- Registro TMR2 y PR2 de 8 bits cada uno. Ambos de lectura y escritura.
- Preescala configurable por programa (1:1, 1:4, 1:16).
- Posescala configurable por programa, con rango desde 1:1 hasta 1:16
- Generación de interrupción cuando el contenido de los registros TMR2 y PR2 coinciden.

La configuración del módulo se realiza a través del registro T2CON. Este registro permite la habilitación y deshabilitación del módulo, así como la configuración de la preescala y posescala. La figura 4.23 muestra un diagrama a bloques simplificado que ayudará a explicar el funcionamiento del módulo.

El TIMER 2 está diseñado para incrementarse desde 00h cada ciclo de máquina (Fosc/4) cuando la preescala está configurada con el rango 1:1. Si la preescala está configurada en un rango diferente, como puede ser 4 o 16, la señal de reloj será dividida en esta escala, es decir, la preescala es un contador que generará un pulso de reloj a la entrada de registro TMR2 cada 4 o 16 pulsos de reloj. Para configurar la preescala se hace uso de los bits T2CKPS1:T2CKPS0 (T2CON<1:0>) del registro T2CON.

El valor del TIMER 2 es comparado con el contenido del registro de periodo PR2, en cada ciclo de máquina. Cuando los valore coinciden, el comparador genera una señal indicando que el evento ha tenido lugar, provocando el reinicio del registro TMR2 y mandando esta señal a la posescala. Los registros TMR3 y PR2 pueden ser leídos o escritos, a diferencia de otros modelos anteriores, en los que para poder modificar el periodo, era necesario detener el módulo TIMER 2 para poder modificar el contenido del registro PR2.

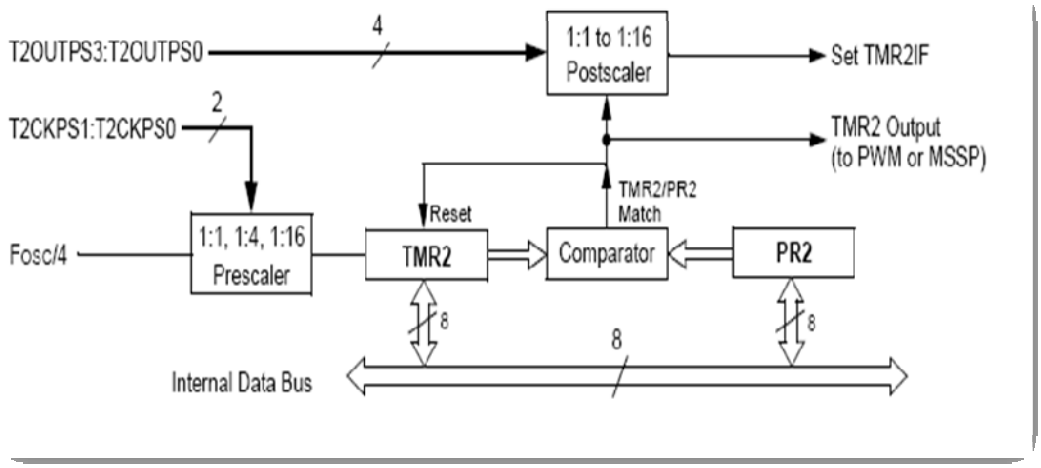


Figura 4.23. Diagrama a bloques del módulo TIMER2.

El contenido del registro TMR2 es puesto a cero cuando ocurre un reinicio del dispositivo, mientras el registro PR2 es inicializado a FFh. Tanto la preescala como la posescala son limpiadas cuando ocurren los siguientes eventos:

- Al escribir en el registro TMR2
- Al escribe en el registro T2CON
- Cuando ocurre un reinicio del dispositivo.

4.2.7 Módulo CCP (Capture, Cooperation, PWM)

Los módulos CCP son recursos periféricos que permiten trabajar en una de tres configuraciones posibles conocidas como captura comparación y generación de pulsos con ancho modulado. En el caso del microcontrolador PIC18F2550, se cuenta con dos módulos. Cada módulo CCP está asociado con un registro de control llamado CCPxCON, y un registro de datos CRPP, compuesto por dos registros de 8 bits, los registros CCPRxH y CCPRxL. Todos estos registros están implementados para lectura y escritura. Para trabajar en alguno de los modos mencionados, el módulo CCP trabaja junto a un temporizador, según se muestra en la tabla 4.8.

Tabla 4.8. Modos de operación del módulo CCP.

Modo	Temporizador usado
Captura	Timer 1 o Timer 3
Comparación	Timer 1 o Timer 3
PWM	Timer 2

En los modos de Captura y Comparación puede elegirse entre el Timer1 y el Timer3 para funcionar. La selección de alguno de estos dos temporizadores se realiza en los bits de habilitación de los registros TCON3. Ambos módulos pueden ser activados para trabajar con el mismo temporizador si son configurados para trabajar en el mismo modo.

En el modo PWM (*Pulse-Width Modulation*), la termina CCPx produce una salida de PWM de hasta 10 bits de resolución. Cada terminal está multiplexada con el registro TRISC apropiado, por lo que será necesario configurar la terminal como salida.

La señal de PWM tiene un tiempo base conocido como periodo y un tiempo llamado ciclo de trabajo (*duty cycle*) en el que la señal permanece en alto una porción del periodo. Para calcular la frecuencia de la señal puede calcularse el inverso del valor del periodo (1/Periodo).

Para definir el valor del periodo del PWM debe escribirse en el registro PR2 el valor obtenido de la fórmula:

PWM periodo = [(PR2)+1]+4Tosc*(TMR2 valor preescala)

Los siguientes tres eventos ocurren en el siguiente incremento del ciclo cuando el contenido del registro TIMER2 es igual al contenido del registro PR2:

- La terminal CCPx es puesta en alto si el ciclo de trabajo es diferente de cero.
- El registro CCPRxH es actualizado con el valor del registro CCPRxL, con lo que se actualiza en valor del ciclo de trabajo.

- Se espera a que el contenido del registro TIMER2 sea igual al contenido del registro CCPRxH para poner la terminal CCPx a nivel bajo.
- Se espera una nueva concordancia entre los registros TIMER2 y PR2.

El ciclo de trabajo es definido por el contenido del registro CCPRxL y los bits CCPxCON<5:4>. La siguiente ecuación puede ser usada para determinar el ciclo de trabajo de la señal de PWM.

PWM Duty cycle = (CCPRxL:CCPxCON<5:4>*Tosc+(TMR2 preescala valor)

El valor del registro CCPRxL puede ser escrito en cualquier momento, pero el ciclo de trabajo no será actualizado sino hasta después que concuerden los contenidos de los registros TMR2 y PR2. El registro CCPxH es usado como un doble buffer del ciclo de trabajo de PWM, este doble buffer es esencial para garantizar la operación del PWM y durante este modo de operación este registro es configurado como de sólo lectura.

4.2.8 Módulo USB

La familia de dispositivos PIC18Fx455/x550 contiene un Serial Interface Engine (SIE) compatible con USB *Low Speed* y *High Speed*. Que permite la comunicación entre cualquier *host* USB y microcontrolador PIC. El SIE puede ser conectado directamente al USB, utilizando un transceptor interno o puede ser conectado a través de un transceptor externo. Un regulador interno de 3.3V permite alimentar eléctricamente al transceptor interno en aplicaciones de 5V.

Algunas características especiales de hardware han sido incluidas para mejorar el rendimiento (funcionamiento). Un puerto doble en el espacio de la memoria de datos del dispositivo (USB RAM) ha sido suministrado para compartir el acceso directo a memoria entre el núcleo del procesador y el SIE. Los descriptores de Buffer son también provistos, permitiendo al usuario programar libremente los *endpoints* y la memoria usada con el espacio USB RAM. Un *Streaming Parallel Port* ha sido provisto para soportar la transferencia ininterrumpida de grandes volúmenes de datos, como son datos isócronos, a buffers de memoria externa. La figura 4.24 muestra la arquitectura del módulo USB.

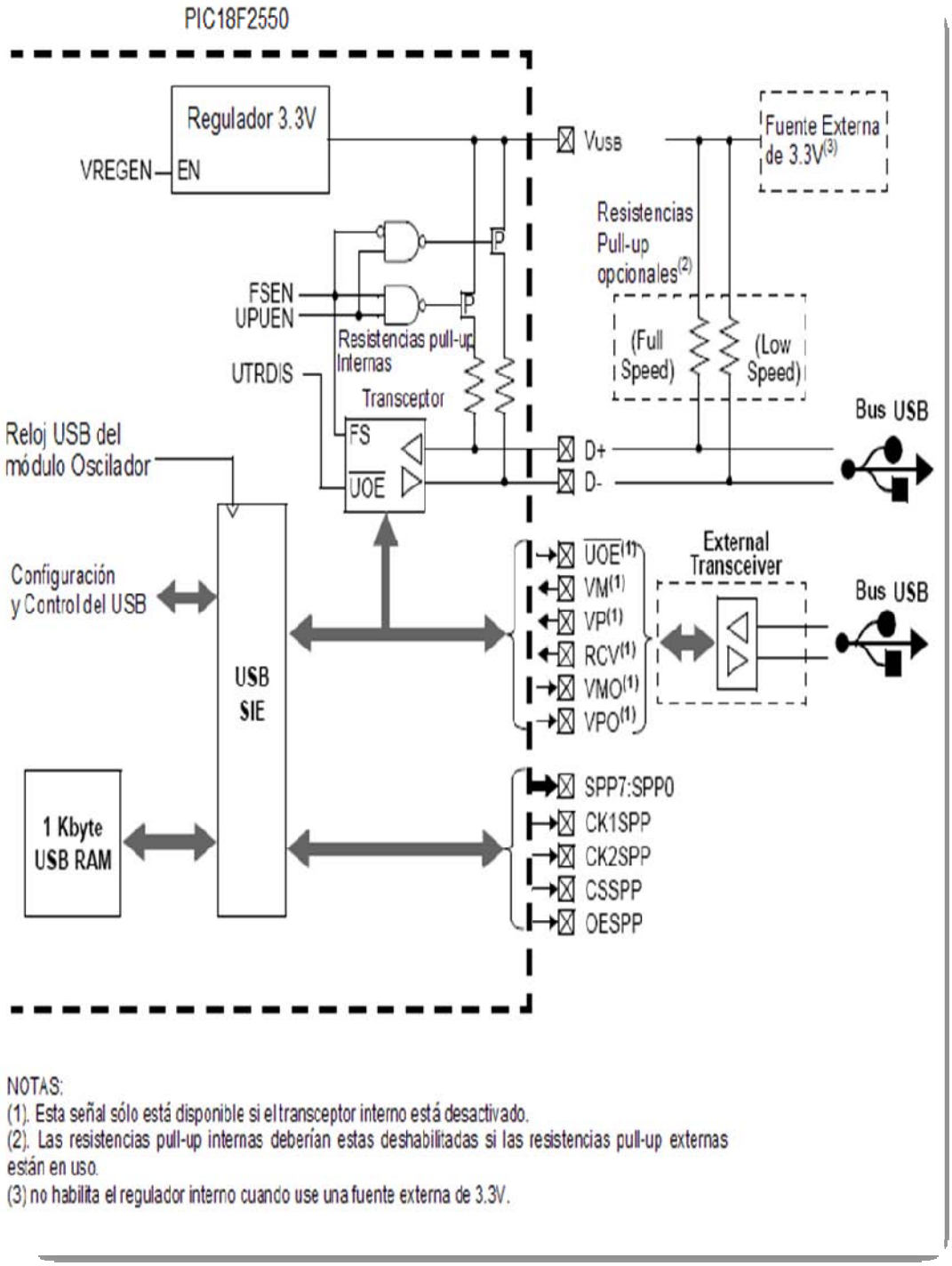


Figura 4.24. Diagrama a bloques del modulo USB del microcontrolador PIC18F2550.

El microcontrolador PIC18F2550 posee un módulo *Serial Interface Engine* (SIE) compatible con USB bajo la versión 2.0 del estándar. Esto permite la comunicación entre cualquier *host* USB y el microcontrolador PIC a velocidades de 1.5Mbps (*Low Speed*) y de 12Mbps (*Full Speed*). Además de soportar transferencias de control, interrupción, masiva e isócronas y ser capaz de manejar 16 *endpoints* bidireccionales, es decir un total de 32 *endpoints*.

Incluye también un transceptor USB en circuito y las resistencias *pull up* que pueden ser activadas por programa, lo que reduce la complejidad del impreso y la necesidad de circuitos externos, sin embargo si se requiriera de una configuración o recursos diferentes, existe la posibilidad de conectar mediante terminales dedicadas, un transceptor externo. Las resistencias pull up internas sólo pueden usarse si el transceptor interno está activo. También dispone de un regulador interno de 3.3V para suministrar los niveles de tensión usados por el transceptor interno durante la comunicación y únicamente requiere de un capacitor de 47uF para estabilizar el voltaje del regulador (V_{USB}). Este voltaje puede ser igualmente suministrado por una fuente externa.

La figura 4.25 muestra la conexión usada para un microcontrolador PIC18F2550 al puerto USB haciendo uso del transceptor y regulador internos, con las resistencias *pull up* habilitadas. Cuando es necesario hacer uso de un flujo continuo de datos y la memoria reservada es insuficiente, es posible conectar una memoria externa al módulo del puerto paralelo SPP en los encapsulados de 40 y 44 terminales.

El módulo USB soporta los dos modos de alimentación descritas en la sección 4.1.2 de este mismo capítulo acerca del puerto USB, los modos *Bus Power* y *Self-Power, además* un modo mixto llamado *Dual Power,* que consiste en aprovechar ambos modos. Cuando el sistema puede opera en modo *Self-Power* y ocurre en error en la fuente de alimentación, éste puede cambiar a un estado de bajo consumo y hacer uso del modo *Bus Power,* con lo que podría mantenerse una rutina que permita el manejo del error sin perder toda la funcionalidad del sistema. El módulo USB es configurado y controlado a través de tres registros, junto a otros 32 que se encargan de manejar las transacciones de la comunicación a través del USB. Estos registros se describen en la tabla 4.9.

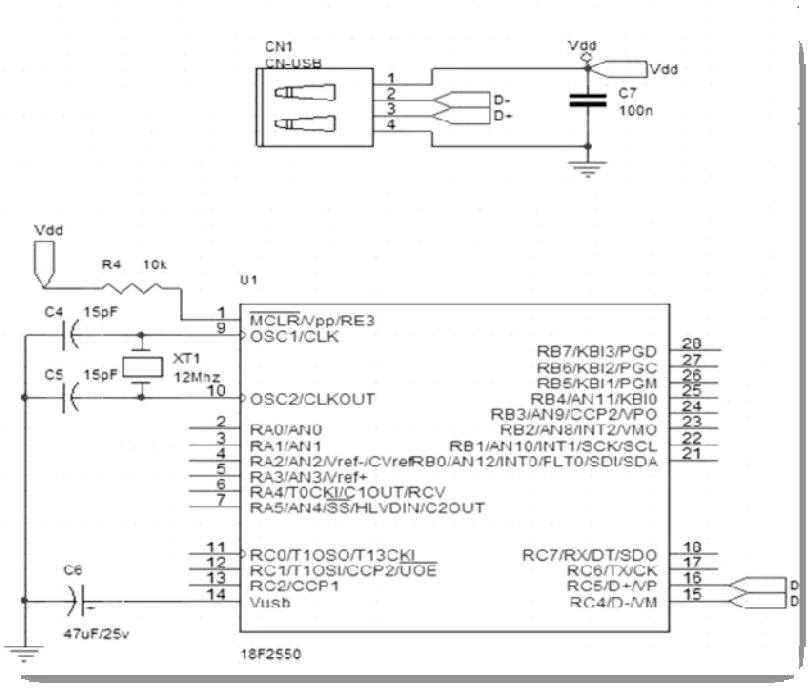


Figura 4.25. Diagrama de conexión de un microcontrolador PIC18F2550 al USB en una configuración *Bus Power* y con el transceptor y regulador internos activos.

Tabla 4.9. Registro usados en la configuración del módulo USB.

Registro	Nombre	Dirección	Descripción
UCON	Control Register		Registro de control
UCFG	Configuration Register		Registro de configuración
USTAT	Transfer Status Register		Registro de estado de la transferencia
UADDR	Device Address Register		Registro de dirección del dispositivo
UFRMH:UFRML	Frame Number Register		Registro de número de trama
UEPn	Endpoint Enable Register		Registro de habilitación de Endpoint 0 al 15

4.3 Características del software de la aplicación e interfaz de usuario

Para poder acceder a los servicios que el firmware ofrece y así operar el programador USB de microcontroladores PIC, deberá diseñarse un software que permita la comunicación entre la interfaz de usuario y el controlador del programador. El puerto de comunicaciones seleccionado para implementar el programador ha sido el USB 2.0, por lo que el software de aplicación también deberá encargarse del flujo de datos de E/S desde la aplicación hacia el puerto y viceversa.

El diseño del software de aplicación se considerará primeramente para un sistema operativo Microsoft Windows XP, ya que es el más utilizado hasta el momento en computadoras personales y equipos portátiles, pero sin dejar de tener en cuenta que una nueva versión de este sistema operativo llamada Windows Vista ha salido al mercado en 2007 y que desplazará gradualmente la versión anterior.

El código fuente del software de la aplicación será diseñado bajo el paradigma orientado a objetos. Las clases generadas se codificarán en lenguaje C++, lo que permitirá la portabilidad de código a otros sistemas operativos como Linux y Mac OS. Por lo tanto, el diseño del software de aplicación debe contemplar un esquema modular que separe el acceso al puerto USB de las funciones del programador y a su vez de la interfaz de usuario.

En la siguiente sección se abordará de manera muy general las principales características que tiene una aplicación Windows, así como la metodología usada para el desarrollo de aplicaciones en este sistema operativo. De igual forma se comentará sobre el lenguaje de programación *Microsoft Visual C++*, que será usado tanto en el desarrollo del software de aplicación como en la interfaz de usuario.

4.3.1 Programación en Microsoft Windows

Windows es un sistema operativo desarrollado por la compañía Microsoft. Este sistema operativo provee al usuario una interfaz gráfica que permite la interacción y control, de forma intuitiva, de los recursos del sistema así como de las aplicaciones que se ejecutan en él.

Una aplicación diseñada para un ambiente Windows® presenta todas las opciones posibles a través de la interfaz gráfica por medio de ventanas, controles y menús, esto da lugar a un diseño de software diferente al que se lleva bajo el paradigma de la programación estructurada, pues la programación es conducida por eventos. Un ***evento*** es una acción reconocida por una ventana, control o menú, de manera general llamados objetos. Un evento puede ser generado por el usuario, sistema operativo o indirectamente por la aplicación. Cuando se desarrolla una aplicación bajo este tipo de programación, la secuencia en que se ejecutan las sentencias no puede ser prevista por el programador. Es por esto que el programador debe implementar código para cada evento posible.

Para poder desarrollar aplicaciones para un ambiente Windows®, Microsoft provee una serie de funciones que pueden ser llamadas desde el lenguaje de programación C, estas funciones son llamadas API de Windows (*Application Programming Interface*) o API WIN32. Las funciones de la API de Windows están almacenadas en archivos de bibliotecas, principalmente dinámicas o con extensión "*.dll*" como son los archivos *kernell32.dll*, *user32.dll* y *gdi32.dll*. Las funciones de la API están encargadas de tareas como la depuración y manejo de errores, entrada y salida de datos de dispositivos, procesos e hilos, manejo de memoria, monitoreo de desempeño, manejo de energía, almacenamiento, interfaces para dispositivos (GDI) e interfaz de usuario.

La comunicación entre aplicaciones y el sistema operativo se realiza a través de mensajes. Un ***mensaje*** es una notificación que Windows envía a una aplicación en ejecución para indicar que un evento ha sucedido. La aplicación entonces responderá al mensaje, y en consecuencia al evento, con una acción específica. El sistema de mensajes de Windows hace posible que distintas tareas compartan el procesador, es decir, permite que Windows sea un sistema multitarea, pues los mensajes son formados en una cola y distribuidos a la aplicación correspondiente, con lo que se permite que una aplicación alterne el uso del procesador a intervalos de tiempo muy cortos (tiempo compartido).

Para que Windows pueda mantener el control de la multitarea, debe estar entre la aplicación y el hardware, por lo que todo contacto que la aplicación tenga con el hardware deberá ser administrado por el sistema operativo. Esto implica que al diseñar un sistema en hardware que se comunique con una aplicación Windows deberá hacerse uso de las funciones de la API para poder tener acceso al hardware del sistema y así garantizar tanto el correcto funcionamiento como la compatibilidad con otras plataformas. Para vincular las funciones de la API incluidas en bibliotecas dinámicas (dll) con el compilador de lenguaje C se hace uso de un archivo de cabecera llamado "*windows.h*".

Un último concepto que es importante conocer al desarrollar aplicaciones Windows es el concepto de ***handle***. Un *handle* es un valor representado por un tipo de dato entero largo sin signo utilizado por Windows para identificar un objeto. Como ya se mencionó, las aplicaciones no acceden directamente al hardware del sistema o a los dispositivos periféricos, en su lugar hacen uso de una estructura de datos asociada al dispositivo que es referenciada por un *handle* en particular.

4.3.2 Microsoft Visual C++

Microsoft Visual C++ es un entorno integrado de desarrollo (IDE) para aplicaciones Windows, gráficas y de consola, bajo el paradigma de programación orientado a objetos. Aunque una aplicación gráfica suele ser fácil de utilizar para el usuario, su desarrollo a través de la API resulta ser una tarea compleja. Por esto, Microsoft integró junto a sus sistemas de desarrollo como Visual Basic y Visual C++ un conjunto de clases llamado *Microsoft Fundation Class* o MFC que permite crear y manejar aplicaciones para Windows, así como sus componentes mediante el paradigma orientado a objetos, con lo que se aprovechan conceptos como herencia, polimorfismo y modularidad que permiten la reutilización de código, entre otras ventajas. Las clases que conforman las MFC se han diseñado a partir de las funciones que forman la API de Windows, por lo que al usar Visual C++ es posible acceder directamente a las funciones de la API o desde los objetos de la MFC.

Entre las herramientas que el IDE proporciona se encuentra un diseñador de formularios que permitirá construir interfaces o ventanas con controles de una manera rápida y sin escribir una sola línea de código. Cada control o ventana deberá estar asociado con un objeto de la MFC, para poder dotarlo de la funcionalidad necesaria para responder a un evento, lo cual hará a través de un método de la instancia correspondiente. Una aplicación gráfica que maneja documentos está compuesta por cuatro clases base, las clases aplicación (*CApp*), marco (*CFrame*), vista (CView) y documento (*CDocument*). Las aplicaciones Windows que incluyen estas cuatros clases base se diseñan bajo la ***arquitectura documento-vista***.

La vista es una ventana que actúa como interfaz entre el usuario y los datos almacenados en un archivo o en una base de datos, por cada vista deberá instanciarse un objeto derivado de una clase *CView*. Una vista es una ventana hija de la ventana marco, es decir de una instancia de la clase *CFrame*. Por su parte, el manejo de los datos es delegado a una instancia de un objeto derivado de la clase *CDocument*. Existen aplicaciones donde es posible que una ventana marco pueda tener múltiples vistas y éstas a su vez, estar asociadas cada una a un documento. De igual forma, un mismo documento puede estar asociado a diferentes vistas, con lo que la presentación de los datos puede realizarse de diferentes formas frente al usuario, por ejemplo, en forma de gráfico o tabla. Cuando una aplicación inicia se crea una instancia derivada de la clase *CWinApp*, esta instancia es la encargada de negociar con el sistema operativo los recursos necesarios para iniciar y terminar de forma ordenada la aplicación.

4.4 Diseño del hardware

Un proyecto como el programador USB de microcontroladores PIC es más fácil de implementar si éste es divido en pequeños módulos que funcionan bien y de manera autónoma, donde la salida de uno de ellos es la entrada del siguiente. Este nivel de ***abstracción***, permite enunciar y modelar el problema con base en el concepto de ***caja negra***.

Durante las secciones previas fueron expuestas las principales características de los protocolos y estándares que serán usados en el diseño y desarrollo del Programador USB de Microcontroladores PIC, con lo que al implementar el proyecto se ha optado por un diseño modular de tres capaz, donde la capa más baja es la interfaz físico eléctrica, denominado hardware, encargada de generar los voltajes de alimentación y programación; la siguiente capa es llamada firmware y será desarrollada en lenguaje C para un microcontrolador PIC18F2550 encargado del control de la máquina de estados que operará tanto la capa de hardware como los protocolos ICSP y USB; por último en el nivel más alto, se encuentra la interfaz de usuario, llamada software de la aplicación.

El ***modelo de caja negra*** permite a las capas que componen el sistema interactúen entre sí de manera independiente, pues cada capa tienen responsabilidades bien definidas que cumplen con alguno de los estándares comentados en las secciones anteriores, por lo que no es necesario conocer como hacen su trabajo, sólo es necesario saber que a partir de una entrada bien definida se obtendrá una salida.

4.4.1 Fuente de poder para un sistema *Self-Power*

Un sistema USB basado en un modo *Self-Power* toma la alimentación eléctrica de una fuente de voltaje diferente a la terminal V_{USB} del puerto USB, que proporciona una tensión de 5V. El programador USB de microcontroladores PIC, necesita dos tensiones con valores de 5V para la alimentación del controlador y 13.5V para el voltaje de programación, por lo que para lograr estos voltajes se hará uso de dos reguladores comerciales, con lo que se diseñará dos fuentes lineales que den por resultado los niveles de tensión deseados. La figura 4.26 muestra el diagrama de las fuentes implementadas con un circuito 7805 y un circuito 7812, ambos reguladores lineales de voltaje positivo fijo a 5V y 12V respectivamente.

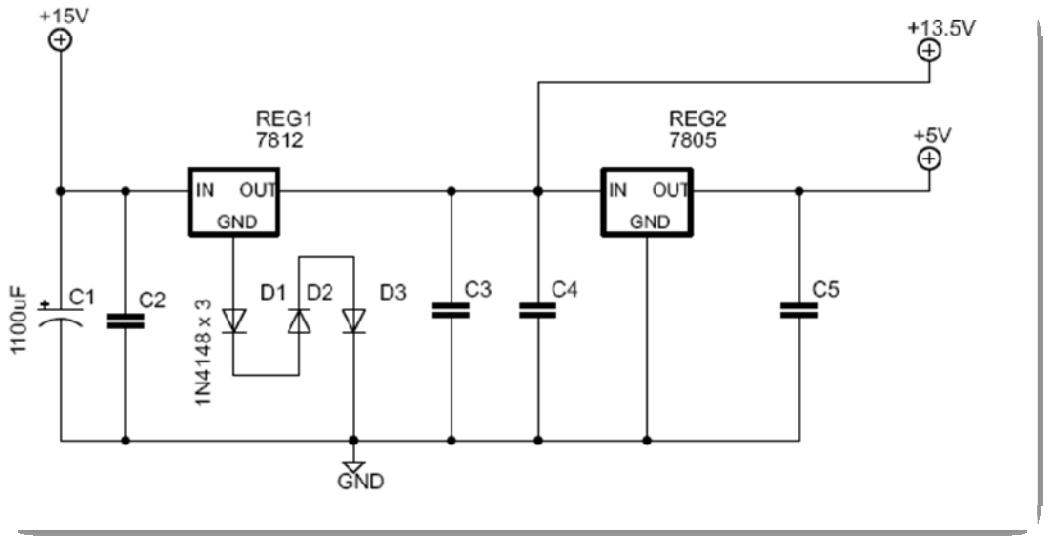


Figura 4.26. Fuente de alimentación para el programador USB de microcontroladores PIC en una configuración *Self-Power.*

El diagrama de la figura 4.26 sólo contempla la etapa de regulación, la cual permite obtener los niveles de tensión deseados a un valor estable, por lo que a la entrada de la fuente es necesaria una fuente primaria de por lo menos 15V de corriente directa. Esta tensión de entrada puede obtenerse desde la red eléctrica a través de un eliminador de voltaje, que será el encargado de las etapas de reducción, rectificación y filtrado.

El dispositivo REG2 (7805) recibe los 13.5V provenientes de la salida del REG1 y los acondiciona a un nivel estable de 5V que serán usados para la alimentación del controlador del programador y del microcontrolador a programar. Los 13.5V necesarios para el voltaje de programación se obtienen a través de un arreglo con un regulador 7812 y tres diodos de bajo consumo 1N4148 conectados en serie a la salida de la terminal dos del regulador. El capacitor C1 de 1100 µF es usado para estabilizar el voltaje de entrada de la fuente mientras que los capacitores C2 a C4 son capacitores cerámicos de 100nF usados como acopladores de voltaje y para filtrar el rizado de altas frecuencias. Tanto el arreglo para el REG1 como para el REG2 fueron tomados de la hoja de datos del dispositivo de la sección de aplicaciones frecuentes.

4.4.2 Fuente de poder para un sistema *Bus-Power*

Hacer trabajar al programador USB de microcontroladores PIC a partir de la tensión y corriente proporcionadas por el puerto USB supone una ventaja sobre hacerlo trabajar en una configuración *Self-Power*, ya que se reducirá su costo y tamaño ya que no será necesario conectarlo a la red eléctrica o a una batería externa, pero de igual modo supone un nuevo reto pues será a partir de una fuente de 5V que debe obtenerse el voltaje de programación con una tensión en el rango de 12V a 14V, típicamente 13.5V.

Para poder elevar la tensión a los niveles requeridos se estudió el uso de las fuentes conmutadas, en particular los ***multiplicadores de voltaje***, pero el uso de estos circuitos supone obtener voltajes altos a razón de una reducción importante de corriente. La segunda opción que fue comentada en la sección 4.1.3 referente a las fuentes de alimentación conmutadas, es hacer uso de un multiplicador DC-DC a partir de un arreglo BOST. La figura 4.28 muestra el diagrama eléctrico del circuito.

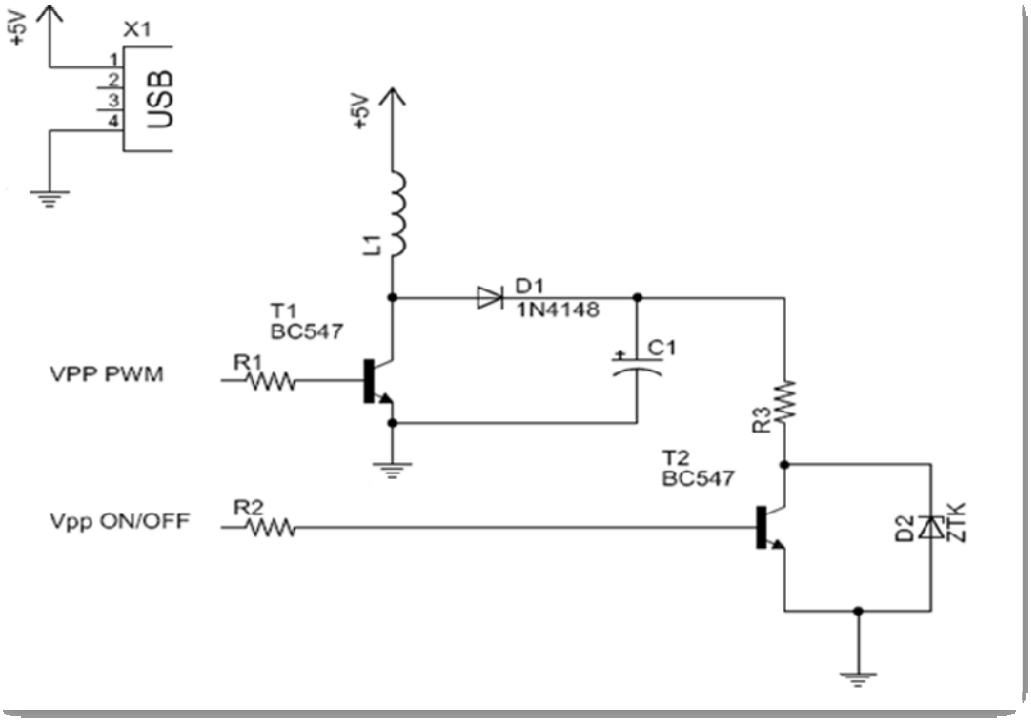


Figura 4.27. Circuito para una fuente conmutada para la configuración *Bus-Power*

El funcionamiento teórico de la fuente ya ha sido explicado en la sección 4.1.3. Este arreglo permite elevar la tensión en el capacitor C1 con respecto al valor que se presente en el ciclo de trabajo de la señal VPP PWM dada en el transistor T1 que sirve como interruptor electrónico. La corriente que será suministrada a la carga está limitada por la resistencia R3, a su vez, el voltaje de la carga será regulado no solo por el ciclo de trabajo de la señal VPP PWM, sino que también el diodo zener D2 estabilizará la tensión a un nivel de 13V, suficientes para el voltaje de programación.

El transistor T2 funcionará como interruptor para la señal VPP, al ponerlo en la región de corte, la tensión entre las terminales del diodo zener será de 13V, mientras que si es puesto en su región de saturación el voltaje en sus extremos será casi 0V.

Puede observase que la resistencia R3 y el diodo D2 se encuentran en serie, por lo que el voltaje suministrado por la fuente y a la salida del capacitor C1 será divido entre ambos, es decir, la corriente suministrada estará en función del valor total de la tensión suministrada por la fuente conmutada, ya que el diodo zener estabilizará a 13V la tensión en sus extremos por lo que si la tensión en la fuente aumenta también lo hará en la resistencia R3 según la relación que marca la ley de Ohm.

$$V = RI \qquad \textit{Ecuación 4.6}$$

V: Voltaje
R: resistencia
I: Corriente

Las señales VPP PWM y VPP ON/OFF utilizadas tanto para proporcionar el ciclo de trabajo del conmutador de la fuente como habilitar y deshabilitar el voltaje de VPP respectivamente son generadas por el controlador del programador USB de microcontroladores PIC. Estas señales son controladas a través del firmware del controlador.

Además del voltaje de programación, el controlador deberá habilitar las señales de Vdd para el microcontrolador a programar, esto lo hará a través de un transistor PNP que funcionará como interruptor electrónico. En el apéndice A puede encontrase el diagrama eléctrico completo para la configuración *Self-Power* y *Bus-Power* con los valores y números de parte de los componentes para el diseño final.

4.4.3 Hardware para la implementación de los protocolos ICSP y USB

Independiente de si se usa una fuente de poder *Self-Power o Bus-Power,* el hardware del sistema debe dar soporte para habilitar y deshabilitar las señales usadas por el protocolo ICSP y USB. Para realizar esta tarea será necesario incluir interruptores electrónicos que permitan la activación y desactivación de estas señales, además de un sistema de realimentación que permita corroborar que los niveles de tensión suministrados al dispositivo a programar se encuentran en los límites adecuados.

La figura 4.28 muestra el diagrama simplificado para la etapa de activación de las señales usadas por el protocolo ICSP para la programación de un dispositivo, además de la conexión del microcontrolador PIC12F2550 al conector USB.

A través del conector JP1 saldrán las 5 señales usadas por el protocolo ICSP para programar un microcontrolador PIC. La primera señal es ICSP-VDD, esta señal es usada para alimentar el microcontrolador que será programado, con una tensión de 5V. Esta señal es habilitada a través del transistor T4, un transistor PNP.

La segunda y tercera señal provienen directamente de las terminales del controlador, son usadas para la señal de reloj ICSP-CLK y datos ICSP-DATA respectivamente. Una resistencia de 100Ω es usada en cada línea para limitar la corriente que sale de las terminales del controlador.

Por último las señales ICSP-VPP1 y ICSP-VPP2 proveen ambas 13.5V de manera independiente. Estas señales nunca son usadas de manera simultánea. Cuando se uso de la técnica ISP[5], es decir, cuando las señales del protocolo ICSP son tomadas de un conector del programador a través de un cable para que un microcontrolador sea programado en el sistema en el que reside, la señal ICSP-VPP2 puede ser omitida, ya que su uso está reservado a un impreso en el que se haga uso de una base para circuito integrado de cero fuerza (ZIF), como la que usan los programadores universales, y sea más sencilla la distribución de las pistas en el impreso para poder programar diferentes modelos de microcontroladores PIC. La última terminal del conecto JP1 es la referencia a tierra del circuito, es decir GND.

[5] In System Programming

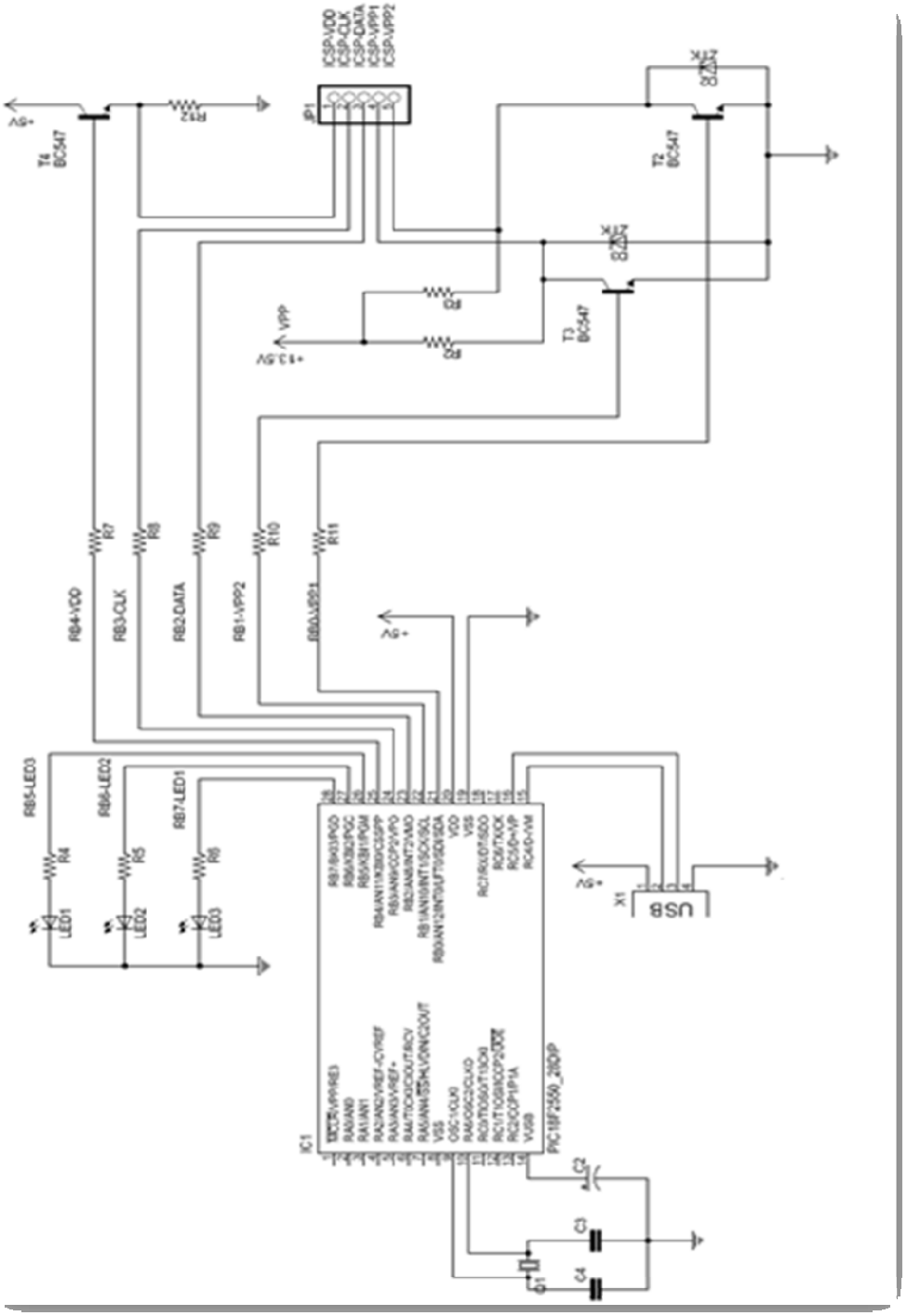


Figura 4.28. Diagrama reducido del circuito del programador USB de microcontroladores PIC para las señales de control para el protocolo ICSP y USB

Los diodos zener D1 y D2 sólo son ocupados cuando se trata de una fuente conmutada, para estabilizar el voltaje de programación a 13V, cuando se hace uso de la fuente lineal implementada con el regulador 7812 estos diodos no serán necesarios.

La velocidad de reloj que se ha escogido es de 12MHz, provista por un cristal de cuarzo al que sólo es necesario conectar dos capacitores para estabilizar la señal de oscilación de un valor de 22pF.

En el caso del hardware necesario para implementar el protocolo USB, sólo es preciso el uso de un capacitor de 47µF a 16V pues todos los recursos necesarios para hacer uso del protocolo, como el transceptor, el regulador de voltaje de 3.3v y las resistencias *pull-down*, se encuentran dentro del microcontrolador 18F2550, por lo que sólo hará falta configurarlos adecuadamente a través del firmware.

4.4.4 Realimentación del sistema

Para poder garantizar que los niveles de tensión emitidos por la fuente conmutada se encuentran en el rango de los 12V a 14V y de igual modo controlar el ciclo de trabajo de la señal PWM CONTROL se hará uso del módulo convertidor analógico a digital integrado en el microcontrolador PIC18F2550. El voltaje de referencia positivo estará determinado por la alimentación de 5V proveniente de la terminal V_{USB} del puerto USB si se trata de una fuente conmutada o del regulador 7805 si se hace uso de una fuente lineal, mientras que el voltaje de referencia negativo estará determinado por la referencia a tierra.

Como puede apreciarse, el rango propuesto de 0V a 5V no logra cubrir en rango necesario de aproximadamente 0V a 14V del voltaje de programación. Este rango ha sido elegido para tener un voltaje estable de una magnitud conocida sin necesidad de usar una tercera fuente de alimentación de la cual tomar los valores de referencia. Entonces, el cálculo de la magnitud del voltaje de programación deberá realizarse de manera indirecta. Para este propósito se ha dispuesto un arreglo en serie de resistencias que serán conectadas en paralelo con el voltaje de programación, es decir, en paralelo con el diodo zener de 13V para el caso de una fuente conmutada.

Si se realiza un arreglo de con dos resistencias del mismo valor la caída de tensión en cada una de ellas será una proporción de 50 por ciento. Por lo que al tomar una lectura en el nodo que se forma entre ambas resistencias, como se aprecia en la figura 4.28, se leerá un valor próximo a la mitad de la tensión total entre los extremos del arreglo. Una proporción de esta magnitud posee el inconveniente que el rango a medir aún se encuentra fuera de los limites de los voltajes de referencia. Una solución a este problema puede ser agregar una tercera resistencia, por lo que la caída de tensión en cada elemento del arreglo tendrá una proporción de 33 por ciento. Si el voltaje suministrado en el arreglo no es superior a 15V, la lectura que se realice entre los nodos estará entre el rango marcado por los voltajes de referencia.

Otra opción que ahorra el uso de la tercera resistencia es variar la proporción que existe entre las dos resistencias R6 y R7, es decir, hacer uso de resistencias de diferentes valores que den una caída de tensión proporcional aproximada a 0.333 para la resistencia R7 con respecto a tierra.

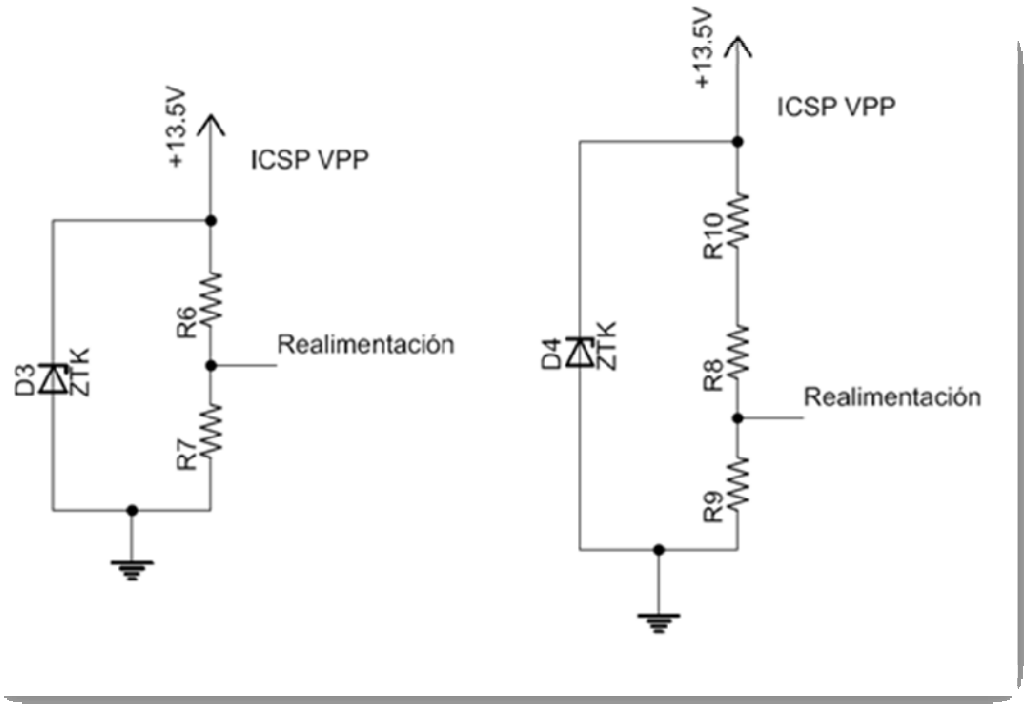


Figura 4.29. Divisores de voltaje para el cálculo indirecto de la magnitud de la tensión en ICSP-VPP.

Un divisor de voltaje armado con una resistencia de 4.7KΩ y una de 2.7KΩ dan por resultado una relación .635 y .365. Esto puede comprobarse con base en el siguiente análisis:

Con base en el diagrama de la figura 4.30, el arreglo que permitirá calcular la tensión en el diodo D2 está dado por las resistencias R4 y R5. En condiciones normales el nivel de tensión en este arreglo será el mismo que en el diodo D2, que por tratase de un diodo zener en polarización inversa mantendrá el voltaje estable a 13V. Este voltaje será llamado $V_{DIVISOR}$. La corriente que circuidla por el arreglo puede calcularse mediante la ley de Ohm, donde el voltaje del arreglo es $V_{DIVISOR}$, mientras que la resistencia total llamada $R_{DIVISOR}$ es la suma de los valores de las resistencias R4 y R5. Al despejar de la fórmula y sustituyendo los argumentos tenemos que el valor de la corriente es de 1.7567mA.

Ley de ohm:

$$V_{DIVISOR} = R_{DIVISOR}\, I_{DIVISOR}$$ *Ecuación 4.8*

Despejando $I_{DIVISOR}$:

$$I_{DIVISOR} = V_{DIVISOR}/R_{DIVISOR}$$ *Ecuación 4.9*

Sustituyendo los argumentos conocidos:

$I_{DIVISOR} = V_{DIVISOR} / (R4+R5)$

$I_{DIVISOR} = 13V / (4.7Ω+2.7Ω)$

$I_{DIVISOR} = 13V / 7.4Ω$

$I_{DIVISOR} = 1.7567mA$

Con base nuevamente en la ley de Ohm es posible ahora calcular la caída de tensión e cada resistencia para luego determinar la relación que existe entre cada una de ellas. Así se obtiene, que la tensión en la resistencia R4 es de 8.2607V mientras que para la resistencia R5 es de 4.7430V

Sustituyendo en la fórmula de la Ley de Ohm (Ecuación 4.7) para obtener V_{R4}:

$V_{R4} = R4\ I_{DIVISOR}$

$V_{R4} = (4.7K\Omega)\ (1.7567mA)$

$V_{R4} = 8.2607V$

Sustituyendo en la fórmula de la Ley de Ohm para obtener V_{R5}:

$V_{R5} = R5\ I_{DIVISOR}$

$V_{R5} = (2.7K\Omega)\ (1.7567mA)$

$V_{R5} = 4.7430V$

Con base en los valores obtenidos puede calcularse la relación entre la caída de tensión en cada resistencia. A través de una regla de tres puede determinarse que si $V_{DIVISOR}$ es 1, para V_{R4} se tiene que la relación es de 0.635 mientras que para V_{R5} es de 0.365. Entonces, la lectura realizada por el convertidor analógico digital revela casi una tercera parte del valor real.

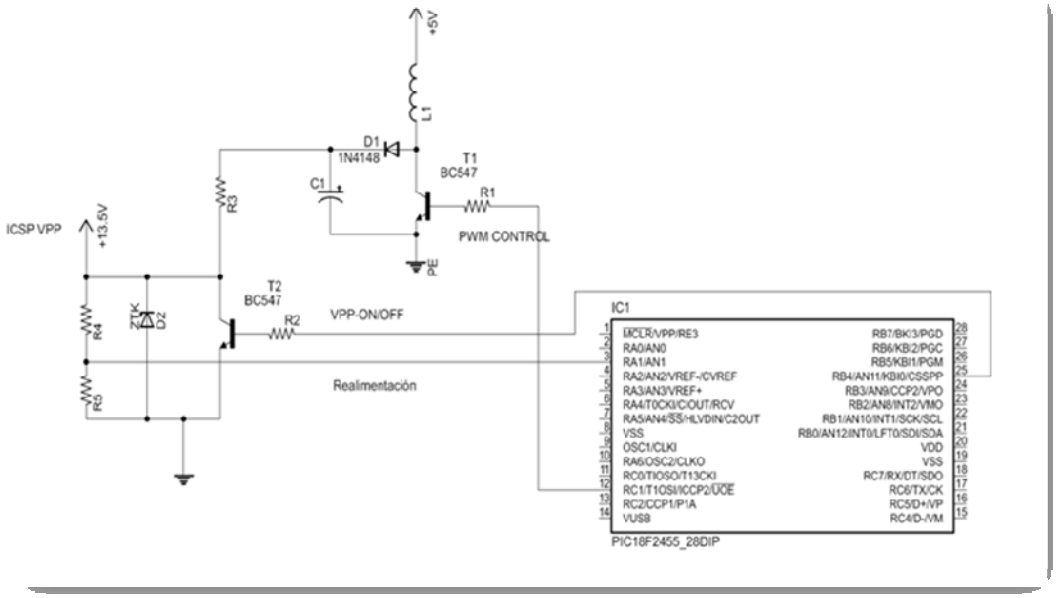


Figura 4.30. Diagrama de conexión del divisor de voltaje para realimentación en una configuración de fuente conmutada.

4.5 Diseño del firmware

El firmware usado por el controlador del programador USB de microcontroladores PIC está diseñado en lenguaje C para el compilador PCH de la compañía CCS. Fueron usados conceptos del paradigma de ***programación estructurada*** como ***modularidad*** y el ***modelo de caja negra*** para hacer más sencillo el diseño y desarrollo del firmware.

El firmware posee diferentes archivos de código fuente, donde los principales archivos que se pueden comentar se encuentran *firmware.c* y *firmware.h*, que son los archivos donde se encuentra la función main, las principales macros y opciones de configuración del microcontroladores PIC18F2550; *descriptors.h*, que contiene la configuración del módulo USB así como los descriptores necesarios y funciones para su control; *inhx32.h*¸que contiene las funciones usadas para codificar y decodificar archivos de código hexadecimal en formato Intel-Hex32; *icsp14.h*, que contiene las funciones necesarias para implementar el protocolo ICSP y por último los archivos con el prefijo *state_X.h*, que de manera respectiva, contienen las funciones que responden a los comandos que puede ejecutar el dispositivo programador.

4.5.1 Máquina de estados

El diseño del El firmware está basado en una máquina de estados que abstrae los comandos a los que el dispositivo programador puede responder con base en una entrada o mensaje desde la interfaz de usuario. La tabla 4.10 muestra los estados de la máquina principal.

Tabla 4.10. Valores posibles de la máquina de estados para el controlador.

Estado	Código	Descripción
STATE_MACHINE_RESERVED	0x00	Valor reservado (RESET)
STATE_MACHINE_INITIATING	0x01	Estado de INICIALIZACION
STATE_MACHINE_WAITING	0x02	Estado de ESPERA
STATE_MACHINE_CONFIGURING	0x03	Estado de CONFIGURACION
STATE_MACHINE_TESTING	0x04	Estado de PRUEBA
STATE_MACHINE_BOOTLOADING	0x05	Estado de CARGA DE FIRMWARE
STATE_MACHINE_PROGRAMMING_12	0x06	Estado de PROGRAMACION
STATE_MACHINE_PROGRAMMING_14	0x07	Estado de PROGRAMACION
STATE_MACHINE_PROGRAMMING_16	0x08	Estado de PROGRAMACION

Para realizar el cambio entre cada uno de los estados es necesario modificar el valor del puntero de la máquina de estados, es decir, una variable de tipo entero llamada ***StateMachine***. Si el puntero de la máquina de estados toma un valor diferente a los mostrados en la tabla, este será redireccionado al estado STATE_MACHINE_WAITING. El diagrama de flujo de la figura 4.31 muestra las transiciones posibles entre los diferentes estados.

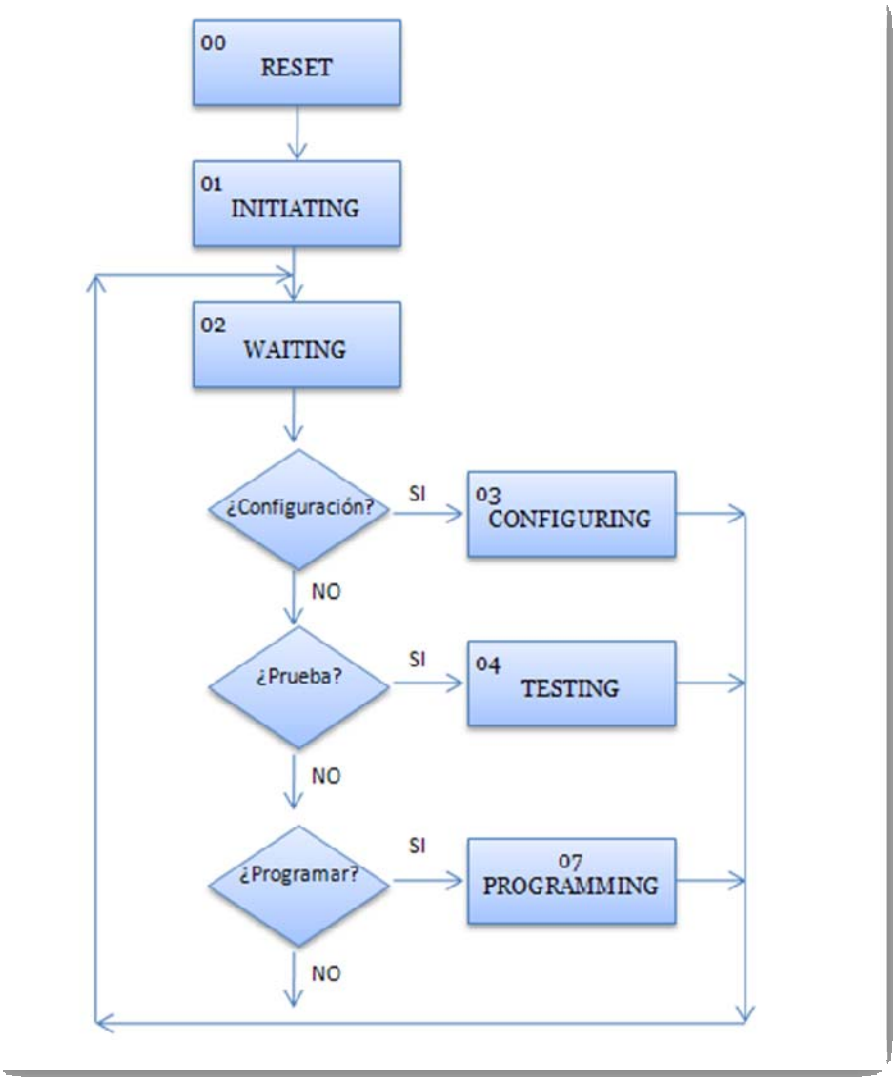


Figura 4.31. Diagrama de flujo de la máquina de estados implementada para el programador USB de microcontroladores PIC.

Cada uno de los estados responde a uno de los grupos de comandos que puede ejecutar el programador USB de microcontroladores PIC, por ejemplo, para poder borrar la memoria de programa de un dispositivo será necesario entrar en el esta de STATE_MACHINE_PROGRAMMING_14 y una vez en este estado, deberá recibirse el comando adecuado para que una vez ejecutado se regrese al estado STATE_MACHINE_WAITING.

Los estados STATE_MACHINE_PROGRAMMING_12, STATE_MACHINE_ PROGRAMMING _16 y STATE_MACHINE_BOOTLOADING están reservados para versiones posteriores del firmware.

El estado STATE_MACHINE_CONFIGURING es el encargado de inicializar los periféricos necesarios para ser usados por el controlador, como lo son el puerto de comunicaciones USB, donde será necesario esperar a que el dispositivo sea enumerado y plenamente instalado por el *HOST*; los módulos CCP1 y ADC, para el sistema de realimentación y control de la fuente conmutada; el puerto serie, cuando se trate de un firmware en modo de depuración de errores; los puertos digitales, para el control de los interruptores electrónicos de las señales de programación ICSP_VPP, alimentación ICSP_VDD, reloj ICSP_CLK y datos ICSP_DATA. Una vez configurados estos periféricos el puntero de la máquina de estados será actualizado con el valor correspondiente para pasar al estado STATE_MACHINE_WAITING.

El estado STATE_MACHINE_WAITNG espera por un comando proveniente desde la interfaz de usuario. A la llegada de un comando, el puntero de la máquina de estados es actualizado con el valor correspondiente del estado donde debe ejecutarse el comando, si es un mensaje que es decodificado como erróneo no se saldrá del esta de espera.

El estado STATE_MACHINE_CONFIGURING es usado para configurar parámetros del programador como lo son si se usará la terminal ICSP_VPP1 o ICSP_VPP2, el retardo, si se verificará el contenido de cada registro justo después de programarlo o no y la familia del microcontrolador a programar.

El estado STATE_MACHINE_TESTING es usado para una configuración prueba manual de las terminales usadas por el protocolo ICSP. A través de la interfaz de usuario podrán activarse una a una las terminales VPP, VDD, DATA y CLK, para realizar una verificación manual del funcionamiento del dispositivo programador.

El estado STATE_MACHINE_PROGRAMMING14 engloba los comandos usados para programador, borrar, y verificar la memoria de datos y programa de un microcontroladores PIC de la familia 16F; es decir, con una palabra de memoria de instrucciones de 14 bits.

Cada uno de estos estados responde a funciones que tienen el mismo nombre que el estado. Estas funciones se almacenan en los archivos con el prefijo state_X.h.

4.5.2 Descriptores USB

Existen diferentes tipos de descriptores para el puerto USB, como los son, configuración, función, interfaz, *endpoint* y string. Estos descriptores fueron definidos como cadenas de caracteres en el archivo *descriptors.h.*

En el descriptor de dispositivo se definen algunos parámetros como el PID y VID, clase de dispositivo, número de interfaces, entro otros parámetros descritos en los comentarios del cuadro de código 4.1.

```
char const USB_DEVICE_DESC[] ={
   USB_DESC_DEVICE_LEN,         //longitud del reporte
   0x01,             //constante del DEVICE (0x01)
   0x10, 0x01,       //versión del USB en BCD
   0x00,             //código de la clase (0 si está definido en la interfaz
                     //FF si es definida por el fabricante)
   0x00,             //código de subclase
   0x00,             //código de protocolo
   USB_MAX_EP0_PACKET_LENGTH,   //tamaño máximo del paquete por endpoint 0
   0xD8, 0x04,       //identificador del fabricante (0x04D8 es Microchip)
   0x11, 0x00,       //identificador del producto
   0x01, 0x00,       //numero de liberación de dispositivo
   0x01,             //índice de la cadena con la descripción del fabricante.
   0x02,             //índice de la cadena con la descripción del producto
   0x00,             //índice de la cadena con el número de serie
   USB_NUM_CONFIGURATIONS       //número de posibles configuraciones
};
```

Cuadro de código 4.1. Descriptor de dispositivo.

El descriptor de configuración define parámetros como la entrada a los descriptores de interfaz y *endpoint* utilizados por la aplicación. El cuadro de código 4.2 describe el contenido del descriptor.

Cuadro de código 4.2. Descriptor de configuración.

```
//descriptor de configuración
char const USB_CONFIG_DESC[] = {
//descriptor de configuración
   USB_DESC_CONFIG_LEN,    //longitud del descriptor
   USB_DESC_CONFIG_TYPE,   //constante CONFIGURATION (0x02)
   USB_TOTAL_CONFIG_LEN,0, //tamaño de todo el dato regresado por esta
                           //configuración
   1,                      //número de interfaces que este dispositivo
                           //soporta
   0x01,                   //identificador para esta configuración
   0x00,                   //índice de la cadena de descripción para esta
                           //configuración
   0xC0,                   //bit 6=1 si es self powered, bit 5=1 si soporta
                           //remote wakeup, bits 0-4 reservado y bit7=1
   0x32,                   //corriente máxima requerida para Bus-Power
                           //(máximo miliampers/2)  (0x32 = 100mA)

//descriptor de interfaz
   USB_DESC_INTERFACE_LEN, //longitud del descriptor
   USB_DESC_INTERFACE_TYPE,//constante INTERFACE (0x04)
   0x00,                   //número que define esta interfaz
                           //(si se tiene más de una interfaz)
   0x00,                   //configuraciones alternativas
   2,                      //número de endpoints, sin contar endpoint 0
   0xFF,                   //código de la clase, FF = definida
                           //por el fabricante
   0xFF,                   //código de la subclase, FF = definida
                           //por el fabricante
   0xFF,                   //código del protocolo, FF = definida
                           //por el fabricante
   0x00,                   //índice de la cadena de descripción para
                           //la interfaz

//descriptor de endpoint
   USB_DESC_ENDPOINT_LEN,  //longitud del descriptor
   USB_DESC_ENDPOINT_TYPE, //constante ENDPOINT (0x05)
   0x81,                   //número de endpoint y dirección (0x81 = EP1 IN)
   0x02,                   //tipo de transferencia soportada //(0 = control,
                           //1 = isócrona, 2 = bulk, 3 = interrupción)
   USB_EP1_TX_SIZE, 0x00,  //tamaño máximo del paquete soportado
   0x01,                   //intervalo de comprobación
                           //(en ms sólo para transferencia de interrupción)

//descriptor de endpoint
   USB_DESC_ENDPOINT_LEN,  //longitud del descriptor
   USB_DESC_ENDPOINT_TYPE, //constante ENDPOINT (0x05)
   0x01,                   //número de endpoint y dirección (0x01 = EP1 OUT)
   0x02,                   //tipo de transferencia soportada //(0 = control,
                           //1 = isócrona, 2 = bulk, 3 = interrupción)
   USB_EP1_RX_SIZE, 0x00,  //tamaño máximo del paquete soportado
   0x01,                   //intervalo de comprobación
                           //(en ms sólo para transferencia de interrupción)
};
```

Por último, el cuadro de código 4.3 define los descriptores string con la información sobre el fabricante y descripción del dispositivo, en este caso, UNAM FES Aragón y Programador LATI USB.

Cuadro de código 4.3. Descriptor de string.

```
char const USB_STRING_DESC[]={
//string 0
         4,                    //longitud de la cadena
         USB_DESC_STRING_TYPE, //tipo del descriptor 0x03 (STRING)
         0x09, 0x04,           //Definido para US-English

//string 1. Información del fabricante
         42,                   //longitud de la cadena
         USB_DESC_STRING_TYPE, //tipo del descriptor 0x03 (STRING)
         'L',0,
         'A',0,
         'T',0,
         'I',0,
         ' ',0,
         'U',0,
         'N',0,
         'A',0,
         'M',0,
         ' ',0,
         'F',0,
         'E',0,
         'S',0,
         ' ',0,
         'A',0,
         'r',0,
         'a',0,
         'g',0,
         'ó',0,
         'n',0,

//string 2. Nombre del dispositivo
         42,                   //longitud de la cadena
         USB_DESC_STRING_TYPE, //tipo del descriptor 0x03 (STRING)
         'P',0,
         'r',0,
         'o',0,
         'g',0,
         'r',0,
         'a',0,
         'm',0,
         'a',0,
         'd',0,
         'o',0,
         'r',0,
         ' ',0,
         'L',0,
         'a',0,
         't',0,
         'i',0,
         '-',0,
         'U',0,
         'S',0,
         'B',0
};
```

4.5.3 Implementación del protocolo ICSP

La implementación del protocolo ICSP se realiza mediante diferentes funciones encargadas de activar y desactivar las señales de VPP, VDD, DATA y CLK. Estas funciones se encuentran en el archivo *icsp12.h*, *icsp14.h* y *icsp16.h* para la programación de dispositivos de las gamas baja, media y mejorada de 12, 14 y 16 bits de la palabra de instrucción respectivamente.

Todos los comandos posibles de usar para cada modelo de microcontrolador fueron embebidos en las funciones de la tabla 4.11. Las macros que definen a los comandos están declaradas en el estado de programación, en el archivo *state_programming.h*. En este mismo archivo se construyen nuevas funciones para controlar los comandos más complejos del protocolo, como son borrar la memoria de datos, borrar la memoria de instrucciones, incrementar el puntero, ir a la dirección de la memoria de configuración, entro otros que ya fueron descritos en la sección 4.1.2.

Tabla 4.11. Comandos diseñados para implementar el protocolo ICSP en el firmware.

Función	Descripción
void Command14(int8);	Esta función envía el comando pasado como argumento. Sólo los 6 bits menos significativos del argumento son enviados a través de la línea ICSP_DATA.
void OnProgramVerifyTestMode(void);	Esta función hace entrar al microcontrolador a programar en el modo *VerifyTestMode* habilitando la señal ICSP_VPP correspondiente.
void OffProgramVerifyTestMode(void);	Esta función hace salir al microcontrolador programado del modo *VerifyTestMode* deshabilitando la seña ICSP_VPP.
void LoadData14(int8, int8);	Esta función envía un dato de 14 bits al microcontrolador a programar. El primer argumento posee los 6 bits más significativos y el segundo argumento posee los 8 bits menos significativos del dato.
void ReadData14(int8, int8);	Esta función recibe un dato de 14 bits leídos del microcontrolador. El primer argumento posee los 6 bits más significativos y el segundo argumento posee los 8 bits menos significativos del dato

4.6 Diseño del software de aplicación e interfaz de usuario

El diseño del software de la aplicación tiene un modelo modular que permite la independencia entre cada segmento, para así poder extender las prestaciones de cada uno en versiones posteriores sin la necesidad de tener que reescribir todo el programa nuevamente. La figura 4.32 muestra la arquitectura por capaz para la interfaz de usuario y del software de aplicación junto a su interacción con el sistema operativo.

La interfaz de usuario se comunica con una biblioteca dinámica o archivo "*.dll"* para acceder a las funciones del programador. A su vez, se comunica con el sistema operativo quien administra los recursos del sistema y en particular el accedo al puerto, en este caso, USB. En la capa más baja se encuentra el firmware del controlador del programador.

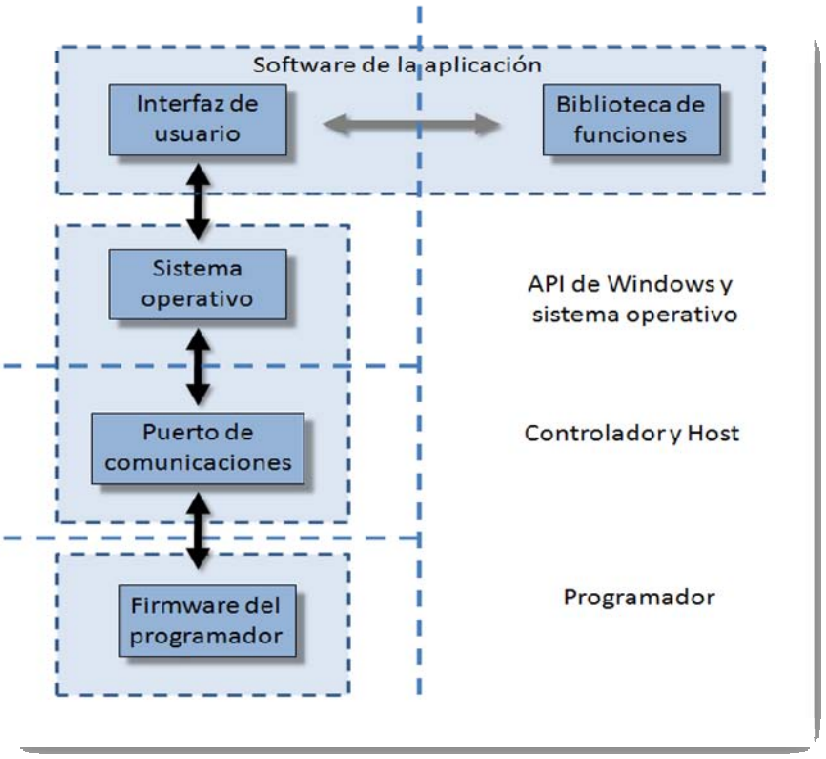


Figura 4.32. Diseño modular del software del proyecto.

La comunicación entre la interfaz de usuario y el firmware del programador se realiza mediante mensajes que fluyen en la secuencia que marca la figura 4.33. Para cargar un archivo de código hexadecimal se hace uso de la interfaz de usuario, ésta se comunica con el software de la aplicación, el cual se encuentra embebido en una serie de funciones en una biblioteca dinámica llamada *latiusb.dll*.

El software de la aplicación se comunica a su vez con el sistema operativo a través de las funciones embebidas en la biblioteca dinámica llamada *mpusbapi.dll* para el control del puerto USB. El sistema operativo regresa un *hanle* válido si el programador está correctamente instalado al USB, para posteriormente establecer un *pipe.*

Es a través del *pipe* que se mantiene la comunicación entre las funciones del software de la aplicación y el firmware de programador. A continuación se describirán con mayor detalle la interfaz de usuario, el software de aplicación y las funciones de la biblioteca mpusbapi.dll.

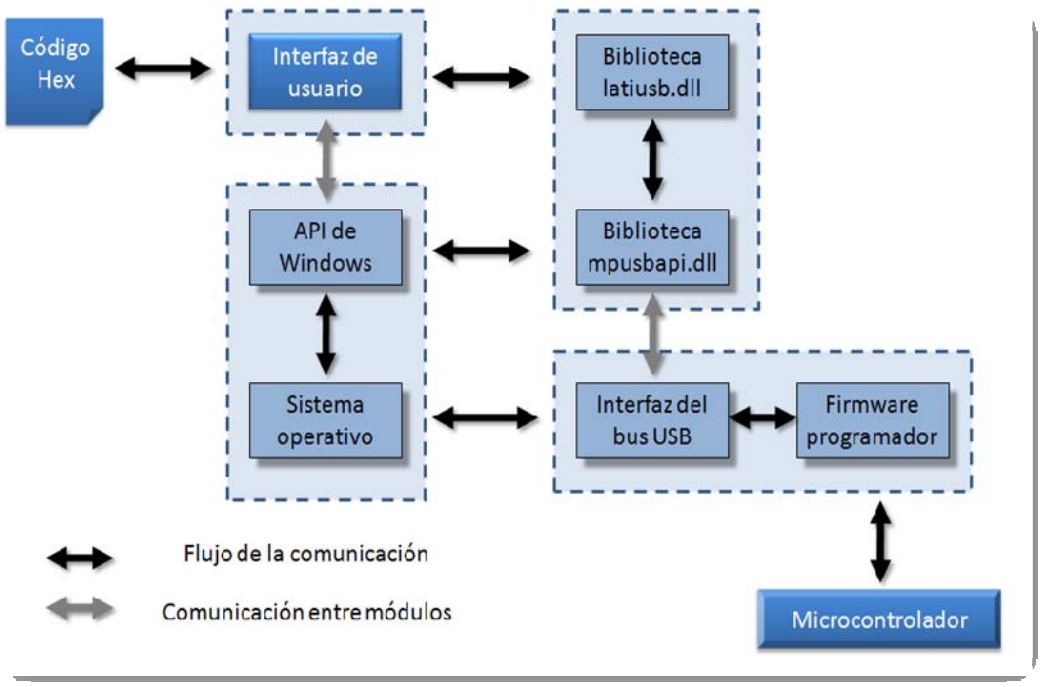


Figura 4.33. Flujo de la comunicación entre los diferentes módulos que conforman el proyecto.

4.6.1 Interfaz de usuario

La interfaz de usuario fue desarrollada mediante el paradigma orientado a objetos en un ambiente gráfico dirigido por eventos, es decir, para un sistema operativo Windows. El lenguaje elegido para implementar la interfaz fue Visual C++ por las características comentadas en la sección 4.3, entre ellas se encuentran la posibilidad de diseñar interfaces Windows, hacer uso de las MFC para dar soporte a la arquitectura documento-vista, permitir construir bibliotecas dinámicas y manejar aplicaciones multitarea a través de múltiples hilos (*threads).*

La figura 4.34 muestra la pantalla principal de la interfaz de usuario. En esta figura, puede apreciarse que se encuentra disponibles los 5 comandos más usados para programador un microcontrolador, estos son, *Programa Dispositivo, Leer Dispositivo, Borrar Dispositivo, Verificar y Seleccionar Dispositivo.* Estos mismos comandos pueden ser accedidos desde el menú *Programador.*

A diferencia de interfaces como IC-Prog y WinPIC800, los controles que muestran el contenido de los buffers para la memoria de datos EEPROM, memoria de usuario y de configuración han sido omitidos para simplificar el aspecto de la interfaz, pero pueden ser accedidos y manejados desde el menú *Buffer*.

Figura 4.34. Pantalla principal de la interfaz de usuario.

Un último comando que puede ser útil es *Probar Hardware,* del menú *Programador.* Este comando presenta una interfaz que permite activar una a una las señales del programador usadas por el protocolo ICSP, con lo que es posible hacer una prueba del nivel de tensión presenta en cada una de las terminales del conector para la programación del microcontrolador a través de un multímetro.

Como ha sido comentado, la interfaz de usuario es el intermediario entre el usuario y el dispositivo programador, pero las tareas que realiza la interfaz son totalmente independientes del modelo de grabador o puerto al que esté conectado, es decir, sólo se encarga de leer y escribir el código hexadecimal tanto en el disco duro como en los buffers en memoria, y presentar los comandos posibles a través de una interfaz gráfica. Las tareas de control del puerto y dispositivo programador se encuentran delegadas a las funciones de las bibliotecas latiusb.*dll* y mpusbapi.*dll*.

4.6.2 Biblioteca de funciones *latiusb.dll*

El software de la aplicación está implementado como un conjunto de funciones en una biblioteca dinámica que es cargada en el tiempo de ejecución. Al principio del diseño del software, estas funciones serían compiladas como código relocalizable para una aplicación de consola y su uso sería el siguiente. Una vez que la interfaz de usuario tuviera todos los parámetros necesarios para realizar la ejecución de un comando sobre el dispositivo programador, ésta llamaría a un programa con extensión ".exe" pasándole como argumentos los parámetros recabados por la interfaz, como pueden ser modelo del dispositivo a programar, ruta del archivo ".hex" y comando a ejecutar.

Al embeber estas funciones que conforman el software de la aplicación en una biblioteca dinámica, estas funciones pueden ser accedidas desde cualquier aplicación que soporte el desarrollo de aplicaciones con vínculos a bibliotecas dinámicas como es el caso de Visual Basic, Visual C++, Borlan C++, Visual C# y Delphi, por citar algunos. Por lo que en lugar de usar dos aplicaciones por separado, una como software para el control de las funciones del programador y otra para la interfaz de usuario, puede construirse una aplicación que sirva como interfaz para las tareas ya descritas y además haga uso de las funciones necesarias para la operación del programador.

Este diseño que hace uso de bibliotecas dinámicas supone la ventaja frente a uno que usa diferentes aplicaciones que puede usarse la misma interfaz para poder manipular diferentes dispositivos o versiones de firmware, desde una sola aplicación. Además la interfaz puede ser escrita en un lenguaje diferente a Visual C++, sin perder ninguna de las características que la biblioteca de funciones proporciona.

Dentro del archivo latiusb.dll se encuentra embebida una clase llamada *CLatiUSB1, la* cual encapsula los atributos y métodos necesarios para la operación del programador USB de microcontroladores PIC. Cuando se desarrolla una nueva aplicación, está deberá instanciar un nuevo objeto de esta clase, para permitir la operación del dispositivo, además de un objeto de la clase *mpusbapi* embebida en la biblioteca dinámica del mismo nombre para el control del puerto USB.

4.6.3 Biblioteca de funciones *mpusbapi.dll*

Esta biblioteca encapsula las funciones necesarias para el control del puerto USB, como escribir, leer, abrir y cerrar el puerto, esta biblioteca es provista por Microchip desde su sitio de Internet. La tabla 4.x muestra un resumen de los métodos embebidos en esta biblioteca. Los detalles de estas funciones se encuentran en el archivo *mpusbapi.cpp*.

Tabla 4.12. Resumen de funciones embebidas en la biblioteca *mpusbapi.dll*.

Función	Descripción
MPUSBGetDLLVersion	Obtiene la versión del archivo MPUSBAPI.DLL
MPUSBGetDeviceCount	Retorna el número de dispositivos que coinciden con el VID y PID
MPUSBOpen	Retorna un *handle* al *pipe* que coincide con el VID y PID.
MPUSBRead	Lee la cantidad de bytes solicitados del puerto USB.
MPUSBWrite	Escribe la cantidad de bytes solicitados en el puerto USB.
MPUSBClose	Cierra un *pipe* con base en un *handle*.

4.7 Conclusión del Capítulo

El presente capítulo es el más extenso del documento, pues en la primera parte son presentas, de manera simplificada, las normas y estándares que debieron ser tomados en cuenta en el diseño del *Programador USB de Microcontroladores PIC*, mientras que en la segunda parte se describe como estos estándares fueron implementados en el desarrollo del sistema.

Además, se describen los criterios usados para implementar las características de cada estándar para cumplir con las premisas de diseñar un sistema de amigable para el usuario, de fácil construcción y bajo costo.

Capítulo 5

Resultados

Capítulo 5
Resultados

Las premisas para el diseño y desarrollo del *programador USB de microcontroladores PIC* fueron construir:

- Una interfaz amigable para el usuario.
- Un sistema fácil de construir.
- Que el sistema sea de bajo costo.

Bajo estos principios, serán expuestos los resultados obtenidos.

5.1 Construcción del sistema

En esta sección se expondrán las características de los productos terminados, tanto en hardware como en software, como resultado de este trabajo.

5.1.1 Hardware del programador

Construir el programador USB de microcontroladores PIC a partir de los diagramas que es posible encontrar en el apéndice A de este documento es una tarea muy sencilla, pues se ha considerado el uso de elementos que sean fáciles de adquirir en cualquier tienda de dispositivos electrónicos. Además el circuito puede construirse en una tableta universal de prototipos (*portoboard*) o en circuito impreso (PCB). Del circuito impreso se presentan dos opciones posibles basadas en el mismo diagrama eléctrico.

El primer diagrama (diagrama A) está diseñado para construir un programador para estudiantes de electrónica que inician el aprendizaje en el diseño y desarrollo de soluciones tecnológicas con microcontroladores PIC, por lo que el circuito impreso posee una base para circuito integrado de cero fuerza (zócalo ZIF) donde se colocará el microcontrolador a programar. Además, las pistas del circuito impreso ocupan una sola cara de la tableta y son de un grosor de 3mm, lo que facilita su construcción comparado con uno de dos caras o con pistas más delgadas.

El segundo diagrama (diagrama B) está diseñado para aprovechar las características de la programación en sistema, por lo que no posee un zócalo para un tipo de encapsulado en particular. En su lugar, dispone de una tira de terminales donde es posible conectar un cable que transportará las señales usadas por el protocolo ICSP hasta la tarjeta donde reside el microcontrolador a programar. Este modelo del programador está dirigido a desarrolladores que desean aprovechar las ventajas de desarrollar sistemas embebidos aplicando la programación en sistema (ISP), ya sea en el prototipo final o el la tarjeta de desarrollos, ya que el programador puede ser colocado sin ningún problema en la tableta de prototipos (*protoboard*).

En el caso del diagrama A, la configuración de la fuente de alimentación se resuelve mediante una fuente conmutada, por lo que no es necesario ningún elemento externo para hacer funcionar al programador. Para el diagrama B, es necesario usar un eliminador de corriente de 15V para alimentar el programador, sin embargo el programador puede funcionar también como fuente de alimentación para proveer un voltaje de 5V a 500mA al sistema donde redice el microcontrolador a programar.

Este diseño supone una ventaja al armar y probar prototipos ya que el voltaje necesario para el funcionamiento del sistema es también proporcionado por el programador. Una desventaja de usar la programación en sistema supone que algunos componentes como diodos y resistencias serán necesarios, según se explica en el documento *In-Circuit Serial Programming*™(ICSP™) *Guide de Microchip*.

5.1.2 Interfaz de usuario

La interfaz de usuario es fácil de usar y presenta en la barra de herramientas los principales comandos a los que el programador USB de microcontroladores PIC responde, como son escribir, leer, borrar, verificar o seleccionar un dispositivo.

Los controles donde se muestra el contenido de los buffers de la memoria de usuario, EEPROM de datos y configuración han sido retirados de la pantalla principal, para ser colocados en cuadros de dialogo a los que se puede acceder desde el menú buffer, con lo que se simplifica la apariencia de la interfaz.

La interfaz está implementada en el lenguaje Visual C++, mediante una arquitectura documento-vista, haciendo uso del paradigma orientado a objetos a través de las *Microsoft Fundation Class* o MFC por sus siglas, pero el desarrollo de una nueva interfaz en cualquier otro lenguaje para aplicaciones *Windows* es posible sin tener que modificar sección de código referente al control del programador, ya que estás funciones se encuentran embebidas en una biblioteca dinámica y son cargadas en tiempo de ejecución cuando son requeridas.

De manera similar ocurre para las funciones concernientes al control de la comunicación a través de puerto USB. Las funciones necesarias para el control del puerto se encuentran embebidas en una biblioteca dinámica, que permite el enlace con un controlador proporcionado por Microchip para el control de los productos diseñados bajo sus microcontroladores PIC que implementan USB.

La primera vez que el programador de microcontroladores PIC sea conectado a un host, el sistema operativo realizará la búsqueda e instalación del controlador. Para que el sistema operativo pueda realizar esta tarea, se necesita de un archivo de instalación "*.inf*" con la información que relacione la aplicación con el controlador adecuado. Este archivo es proporcionado junto al controlador en el disco compacto que acompaña a este documento.

5.1.3 Firmware del programador

El controlador que operará las funciones tanto del control del protocolo ICSP como de la comunicación USB debe ser programador con contenido del archivo *firmware.hex* contenido en el disco compacto que acompaña a este documento. Una vez programado puede ser montado en el programador y es posible su uso.

El firmware del controlador implementa una máquina de estados que responden a mensajes provenientes de la interfaz de usuario a través del puerto USB. El código fuente del firmware también se incluye en disco compacto anexo, por lo que es posible dotar al firmware de nuevas prestaciones o características para otras familias de dispositivos, tomando en cuenta que cada año Microchip incorpora nuevos modelos de microcontroladores PIC.

5.2 Prueba del sistema

El sistema fue diseñado y probado para programar los microcontroladores PIC16F84, PIC16F628A y PIC16F877A. La metodología usada para realizar las pruebas fue:

- Generar un buffer con un valor constante para todos sus registros, en este caso se eligió el valor 0x1234 y programar el dispositivo.
- Generar un buffer con valor consecutivos, iniciando con 0x0000 hasta 0x00FF y programar el dispositivo.
- Generar un buffer con valores aleatorios y programar el dispositivo.
- Entre cada programación se verificó el contenido de la memoria de usuario, memoria de datos EEPROM y de configuración.
- Una vez realizada cada prueba, la memoria del microcontrolador era borrada y verificada que estuviera en blanco.

La figura 5.1 muestra el diagrama de flujo de los pasos a seguir para probar el firmware del programador.

Las pruebas finales fueron realizadas también con microcontroladores de las mismas subfamilias de los dispositivos ya mencionados es decir, los modelos PIC16F876A y 16F627A.

Las funciones encargadas de implementar el protocolo ICSP responden a los mensajes usados por cada uno de los modelos de microcontroladores PIC de la gama media, por lo que, bajo el estudio en particular de cada modelo, es posible extender el número de dispositivos que el programador USB de microcontroladores PIC es capaz de programar sin que estas funciones tengan cambios significativos.

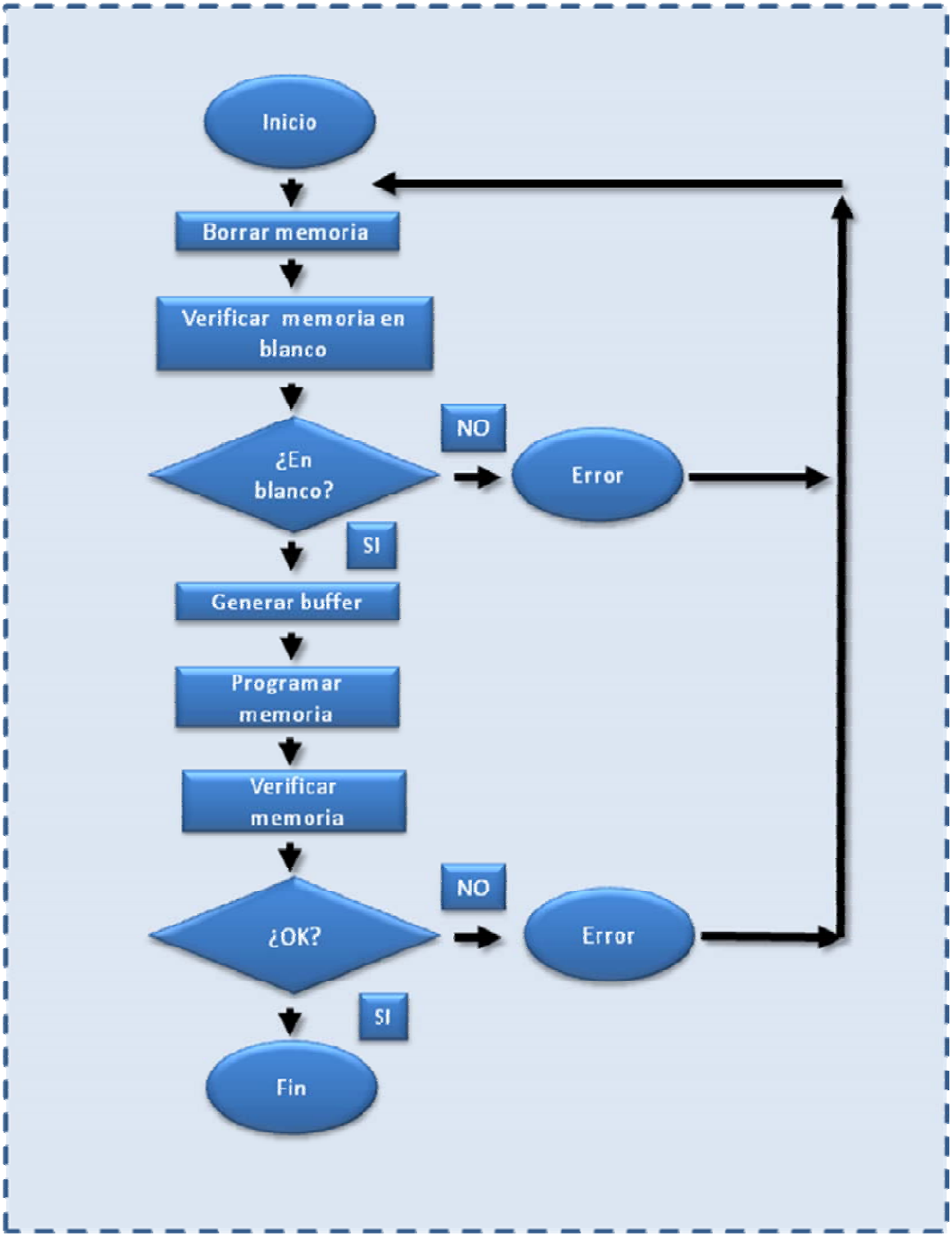


Figura 5.1. Metodología usada para la prueba del firmware.

Capítulo 6

Conclusiones

Capítulo 6
Conclusiones

Llegar a escribir las últimas páginas de este trabajo es una satisfacción muy grande, en especial el poder concretizar las aportaciones que se realizaron con la investigación, diseño y desarrollo del programador USB de microcontroladores PIC.

El presente capítulo, como su título indica, presenta las conclusiones y comentarios finales.

6.1 Conclusiones finales

Se diseñó y desarrolló un prototipo funcional de programador de microcontroladores de 8 bits de la marca Microchip, también conocidos como microcontroladores PIC, para los dispositivos que conforman la gama media con interfaz de comunicaciones USB.

Para la implementación del hardware del dispositivo programador son propuestos dos diagramas eléctricos. El primer diagrama describe una configuración *Power-Bus*, donde es posible que el dispositivo programador tome la energía necesaria a partir de la terminal V_{USB} del puerto USB para generar, a través de una fuente conmutada DC-DC en un arreglo *Boost*, los niveles de tensión necesarios de alimentación ICSP-VDD y de programación ICSP-VPP con niveles de 5V y 13.5V respectivamente.

El dispositivo programador dispone de una base para circuito integrado de cero fuerza de inserción (ZIF) para programador microcontroladores PIC con encapsulados *Dual In Line Package* (PDIP) de 8, 18, 28 y 40 terminales. Para poder programar otros dispositivos de las mismas familias pero con diferente encapsulado o para hacer uso de la programación en sistema (ISP) se emplea un cable con el cual se toman las señales usadas por el protocolo ICSP desde un conector dispuesto para tal propósito.

En el segundo diagrama se presenta una configuración *Self-Power* donde se hace uso de un eliminador de corriente que toma la energía eléctrica de la red de alimentación para así reducir, rectificar y filtrar la tensión alterna de entrada de 127V a un nivel próximo de 15V de corriente directa. Esta tensión de 15V de corriente directa sirve como entrada a una fuente lineal formada por dos reguladores embebidos en los circuitos integrados 7805 y 7812 para regular los niveles de tensión necesarios para la señal ICSP-VDD e ICSP-VPP, sin tomar corriente de la terminal V_{USB} del puerto USB.

La configuración *Self-Power* permite montar el dispositivo programador en la tarjeta universal de prototipos o *protoboard* y tomar una señal de alimentación compatible con los niveles TTL de 5V con una corriente de 500mA como máximo. Debido a este diseño, el dispositivo programador no posee una base para circuito integrado, en su lugar dispone de terminales de las cuales se toman las señales usadas por el protocolo ICSP para ser llevadas a través de un cable al sistema embebido que será programado.

Los componentes usados en la construcción de cualquiera de las dos configuraciones descritas, incluyendo los materiales para la elaboración del circuito impreso, pueden ser adquiridos en cualquier tienda de componentes electrónicos, lo que permite reducir los costos del dispositivo final. Además, el microcontrolador PIC18F2550 en su encapsulado PDIP puede ser pedido de manera gratuita en el sitio de Internet de Microchip después de realizar el registro en su sistema de muestras (*samples*).

El firmware del controlador del dispositivo programador está implementado en un microcontrolador PIC18F2550 de 18 terminales, bajo el paradigma de la programación estructurada haciendo uso del lenguaje C. El compilador usado fue PCH incorporado en el ambiente de desarrollo integrado CCS C de la empresa desarrolladora de software y soluciones para sistemas embebidos CCS, quien ofrece una versión de demostración para estudiantes que da soporte a través de un conjunto de archivos de cabecera a las funciones necesarias para la operación y configuración de los dispositivos del microcontrolador, incluyendo el puerto USB embebido en este dispositivo.

El firmware está diseñado bajo un modelo de máquina de estados el cual permite la modularidad entre cada una de las funciones que el programador es capaz de realizar. Con este modelo de máquina de estados se permite agregar nuevas funciones del dispositivo sin necesidad de variar su diseño o modificar la forma en que se realizan las tareas ya definidas, esto gracias a la implementación del concepto de ***caja negra***.

El software de aplicación que permite acceder a las funciones del programador fue embebido en una biblioteca dinámica en forma de clase. Esto permite construir una interfaz de usuario tanto para una aplicación *Windows* como para una aplicación de consola en cualquier lenguaje de programación que soporte inclusión de bibliotecas dinámicas como Visual C++, Visual C#, Visual Basic, Borlan C++ o Borlan Delphi, entre otros.

La interfaz de usuario es la encargada de presentar a través de una forma amigable al usuario los parámetros de configuración y las acciones que el dispositivo programador puede realizar mediante el uso de cuadros de dialogo, controles y menús. Está interfaz fue diseñada en Visual C++ bajo el paradigma de programación orientado a objetos y dirigida por eventos con el uso de las *Microsoft Foundation Class*.

La comunicación y control del puerto USB se logró con el uso de un controlador (*driver)* proporcionado por Microchip llamado *mchpusb.sys*. Para tener acceso a las funciones de este controlador se hace uso de una biblioteca dinámica llamada *mpusbapi.dll*.

El hacer uso de bibliotecas dinámicas para acceder a las funciones del dispositivo programador y del puerto USB permite mantener independencia entre la interfaz y estas tareas, por lo que se pueden agregarse a la interfaz de usuario nuevas bibliotecas que contengan las funciones necesarias para comunicarse con otros modelos de programadores, que hagan uso de otros puertos como lo son los programadores JDM o TAIT descritos en el capítulo 3, todo esto sin alterar la funcionalidad de la interfaz.

El software de aplicación fue diseñado bajo el paradigma orientado a objetos e implementado en lenguaje C++, con lo que se permite el poder llevar las funciones que controlan el dispositivo programador a otras arquitecturas o sistemas operativos como Linux y Mac OS construyendo en ella sus respectivas interfaces de usuario y programando lo concerniente al control y acceso al puerto USB.

Todo el código fuente desarrollado para la aplicación, como lo es el código fuente del firmware, interfaz de usuario y biblioteca dinámica para el control del programador, es incluido en el disco compacto anexo a este trabajo, con lo que se permite sea tomado como base para futuros desarrollos y actualizaciones por parte de terceros con o sin fines de lucro, siempre y cuando se cite la fuente.

Quedan concluidos los objetivos inicialmente planteados de diseñar y desarrollar un programador con interfaz USB para microcontroladores PIC de la marca Microchip con software de aplicación e interfaz de usuario, de bajo costo para el uso en la enseñanza de microcontroladores en el desarrollo de soluciones tecnológicas. Lo que implica que cada estudiante o emprendedor en el mundo podrá beneficiarse de esta herramienta para aprender a usar microcontroladores de una manera amigable con la facilidad de que contará en casa una herramienta indispensable para cumplir tal objetivo.

El registro del proyecto bajo la licencia GPL GNU o una licencia similar que permitiría su libre distribución y modificación con o sin fines de lucro siempre y cuando se cite la fuente queda en proceso de trámite.

6.2 Trabajo futuro

El proyecto presentado en el presente trabajo es base para el desarrollo de una aplicación más completa que con el tiempo puede ser desarrollada. Es por eso que se proporciona la documentación y código fuente tanto del firmware como del software desarrollado, así como los diagramas eléctricos.

Entre las mejoras del programador USB de microcontroladores PIC que se realizarán a corto plazo se encuentra en la capa del firmware:

- Incluir nuevo modelos de la gama media soportados por el firmware con ayuda de la biblioteca de funciones desarrollada.
- Diseñar nuevas funciones para el firmware para dar soporte a la gama baja de 12 bits y mejorada de 16 bits.

Para la capa de software:

- Incluir herramientas que permitan tareas como: desensamblado de código hexadecimal, conversión de entre bases de numeración y autodirección del dispositivo.

En las proyecciones a mediano y largo plazo quedan pendientes:

- Dotar al programador de la capacidad de programar de manera simultánea múltiples microcontroladores.
- Permitir que el programador pueda trabajar de manera autónoma, sin necesidad de una PC o laptop, dotando al sistema de un panel de control que permita ejecutar los mismos comandos básicos que la interfaz de usuario con el fin de realizar una actualización del firmware en campo. Para tal propósito, el archivo o archivos de código hexadecimal serán descargados a una memoria de datos EEPROM para luego poder ser descargado el programa a un microcontrolador mediante la programación en sistema.
- Construir una interfaz de usuario para sistemas operativos diferentes a Windows XP y Windows Vista, como GNU/Linux y Mac OS.
- Dotar al firmware de un sistema de auto actualización de firmware a través del puerto USB, lo que liberaría al desarrollador de tener que usar un segundo programador para actualizar el firmware de su programador USB de microcontroladores PIC.
- Dotar al programador de la capacidad de poder programar nuevos modelos de microcontroladores de Microchip de 16 y 32 bits llamados DSPIC.

Apéndice A

A1. Lista de componentes

Diagrama A.

El diagrama A del programador USB de microcontroladores PIC está diseñado para ser usado con una fuente de alimentación diferente al puerto USB. A continuación se enumeran los componentes que permiten armar el circuito impreso del apéndice A2.

Fuente de alimentación

- Regulador lineal de 5V 7805
- Regulador lineal de 12V 7812
- 3 diodos 1N4148
- 1 eliminador de 5V a 500mA

Señales del ICSP

- 1 transistor BC547
- 1 transistor BC557
- 1 resistencia 1K
- 1 resistencia 10K
- 1 resistencia 300
- 1 resistencia 220
- 2 resistencias de100

USB y placa

- 1 placa fenólica de una cara de 10x10 cm
- 1 conector USB tipo B hembra.
- 1 conector para eliminador
- 1 tira de pines dobles con ángulo en 90º
- 1 tira de pines
- 1 microcontrolador PIC18F2550
- 1 LED bicolor
- 1 LED Rojo
- 1 cristal de cuarzo de 12 MHz
- 3 resistencias 1KΩ
- 1 base para circuito integrado de 40 terminales de cero fuerza (ZIF)
- 1 dipswitch de 4 terminales

Diagrama B.

El diagrama B del programador USB de microcontroladores PIC está diseñado para ser usado con una fuente de alimentación conmutada que eleva en voltaje tomado de la terminal V_{USB} del puerto USB. A continuación se enumeran los componentes que permiten armar el circuito impreso del apéndice A3.

Fuente de alimentación

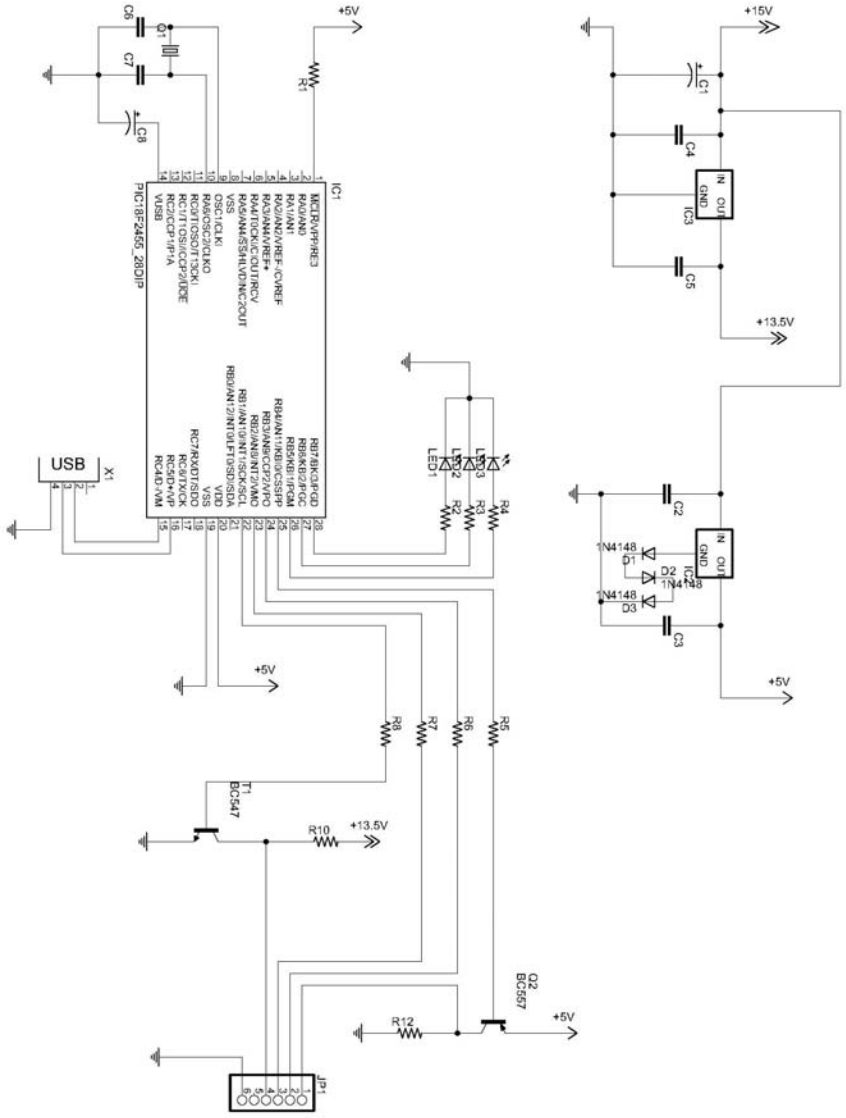
- 1 Bobina de 680 µH
- 2 transistor BC547
- 2 transistor BC557
- 1 diodos 1N4148
- 1 capacitor 47 µF a 25V
- 1 capacitor 10 µF a 16V
- 1 capacitor 0.1 µF
- 5 resistencias 10KΩ
- 1 resistencia 100KΩ
- 1 resistencia 4.7KΩ

- 1 resistencia 2.7KΩ

USB y placa

- 1 placa fenólica de una cara de 10x10 cm
- 1 conector USB tipo B hembra.
- 1 tira de pines
- 1 microcontrolador PIC18F2550
- 1 LED bicolor
- 1 LED Rojo
- 1 cristal de cuarzo de 12 MHz
- 3 resistencias 1KΩ
- 1 base para circuito integrado de 40 terminales de cero fuerza (ZIF)

A.2 Diagrama eléctrico



Apéndice B

Código fuente del firmware

```
/*****************************************
firmware.cpp
Archivo principal de la aplicacion
UANM FES Aragon 2008
*****************************************/

#include "firmware.h"

void main(void) {
   StateMachine = STATE_MACHINE_INITIATING;

   while (TRUE)
   {
      switch(StateMachine)
      {
         case  STATE_MACHINE_INITIATING:
                  Initiating();
                  break;

         case  STATE_MACHINE_WAITING:
                  Waiting();
                  break;

         case  STATE_MACHINE_CONFIGURING:
                  Configuring();
                  break;

         case  STATE_MACHINE_TESTING:
                  Testing();
                  break;

         case  STATE_MACHINE_BOOTLOADING:
                  Bootloading();
                  break;

         case  STATE_MACHINE_PROGRAMMING_12:
                  Programming14();
                  break;

         case  STATE_MACHINE_PROGRAMMING_14:
                  //Programming14();
                  break;

         case  STATE_MACHINE_PROGRAMMING_16:
                  //Programming14();
                  break;

         default: //Manejador de errores
                  StateMachine = STATE_MACHINE_WAITING;
      }
   }
}
```

```
/*******************************************
firmware.h
Archivo de configuracion de la aplicacion
UANM FES Aragon 2008
*******************************************/


#include "18F2550.h"
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL3,CPUDIV1,VREGEN
#use delay(clock=48000000)


/*******************************************
* Configurar parámetro del Puerto USB
*******************************************/
#define USB_HID_DEVICE     FALSE             // Deshalitar configuracion HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   // Habilitar EP1 como entrada en
modo bulk/interrupt transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   // Habilitar EP1 como salida en
modo bulk/interrupt transfers
#define USB_EP1_TX_SIZE    3                 // Tamaño del buffer de salida
del EP1
#define USB_EP1_RX_SIZE    3                 // Tamaño del buffer de entrada
del EP1

#include "pic18_usb.h"     // bibliotea para el uso del driver USB de
Microchip PIC18Fxx5x
#include "descriptors.h"   // Configuración del peurto USB y los descriptores
#include "usb.c"           // biblioteca de handles usb setup tokens and get
descriptor reports

#define GetUsbPacket()     usb_get_packet(1, RxBuffer, USB_EP1_RX_SIZE)
#define SetUsbPacket()     usb_put_packet(1, TxBuffer, USB_EP1_TX_SIZE,
USB_DTS_TOGGLE)
#define WaitPacket()       while(usb_kbhit(1) == false)
#define IsUSBEmpty()      usb_kbhit(1) == false


/*************************************************
* Valores para el puntero de la Maquina de Estados
*************************************************/
#define  STATE_MACHINE_INITIATING      0x00    // Estado de INICIALIZACION
#define  STATE_MACHINE_WAITING         0x01    // Estado de ESPERA
#define  STATE_MACHINE_CONFIGURING     0x02    // Estado de
CONFIGURACION_PROGRAMADOR
#define  STATE_MACHINE_TESTING         0x03    // Estado de DEPURACION
#define  STATE_MACHINE_BOOTLOADING     0x04    // Estado de CARGA_FIRMWARE
#define  STATE_MACHINE_PROGRAMMING_12  0x05    // Estado de PROGRAMACION PIC
12 bits
#define  STATE_MACHINE_PROGRAMMING_14  0x06    // Estado de PROGRAMACION PIC
14 bits
#define  STATE_MACHINE_PROGRAMMING_16  0x07    // Estado de PROGRAMACION PIC
16 bits


/*************************************************
* Valores para Variables globales
*************************************************/
#define LED_RED      PIN_B7
#define LED_GREEN    PIN_B6
#define LED_BUSY     PIN_B5
#define LedOn        output_high
#define LedOff       output_low
```

```
#define TxCommand    TxBuffer[0]
#define TxData_H     TxBuffer[1]
#define TxData_L     TxBuffer[2]

#define RxCommand    RxBuffer[0]
#define RxData_H     RxBuffer[1]
#define RxData_L     RxBuffer[2]

#define TypeMessage  TxBuffer[0]
#define Message      TxBuffer[1]

// bVerifyProgram
#define  ON_VERIFY_PROGRAM       0x00
#define  OFF_VERIFY_PROGRAM      0x01

// CommandSet
#define  SET_ICSP_12_BIT         0x02
#define  SET_ICSP_14_BIT         0x03
#define  SET_ICSP_16_BIT         0x04

// VppPin
#define  SET_VPP1    0x05
#define  SET_VPP2    0x06


/*****************************************
* Variables GLOBALES
*****************************************/

   int8 StateMachine;         // Puntero de la maquina de estados

   int8 RxBuffer[3];          // Buffer de entrada
   int8 TxBuffer[3];          // Buffer de salida

   // Varibales Globales de Configuracion del Programador
   int8  bVerifyProgram;      // Verificar despues de programar
   int16 RangeFlash;          // Valores validos para la memoria FLASH
   int16 RangeEeprom;         // Valores validos para la memoria EEPROM
   int8  VppPin;              // Usar Vpp1 o Vpp2
   int8  CommandSet;          // Seleccionar juego de instrucciones


/*****************************************
* Bibliotecas de funciones
*****************************************/
#include "handles.h"
#include "icsp14.c"
#include "state_configuring.c"
#include "state_initiating.c"
#include "state_waiting.c"
#include "state_testing.c"
#include "state_bootloading.c"
#include "state_programming14.c"
```

```
/*****************************************
descriptors.h
descriptors de la comunicacion USB
UANM FES Aragon 2008
*****************************************/
#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>


   #DEFINE USB_TOTAL_CONFIG_LEN      32 //config+interface+class+endpoint

   //configuration descriptor
   char const USB_CONFIG_DESC[] = {
   //config_descriptor for config index 1
         USB_DESC_CONFIG_LEN,     //length of descriptor size
         USB_DESC_CONFIG_TYPE,    //constant CONFIGURATION(0x02)
         USB_TOTAL_CONFIG_LEN,0,  //size of all data returned
                           //for this config
         1,      //number of interfaces this device supports
         0x01,   //identifier for this configuration.
         0x00,   //index of string descriptor for this onfiguration
         0xC0,   //bit 6=1 if self powered, bit 5=1 if supports remote wakeup
                 //(we don't), bits 0-4 reserved and bit7=1
         0x32,   //maximum bus power required (maximum milliamperes/2)
                 //(0x32 = 100mA)

   //interface descriptor 0 alt 0
         USB_DESC_INTERFACE_LEN,  //length of descriptor
         USB_DESC_INTERFACE_TYPE,      //constant INTERFACE (0x04)
         0x00,                //number defining this interface (IF we had more
                              //than one interface)
         0x00,                //alternate setting
         2,       //number of endpoints, not counting endpoint 0.
         0xFF,                //class code, FF = vendor defined
         0xFF,                //subclass code, FF = vendor
         0xFF,                //protocol code, FF = vendor
         0x00,                //index of string descriptor for interface

   //endpoint descriptor
         USB_DESC_ENDPOINT_LEN,  //length of descriptor
         USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (0x05)
         0x81,              //endpoint number and direction (0x81 = EP1 IN)
         0x02,              //transfer type supported (0 is control, 1 is iso,
2 is bulk, 3 is interrupt)
         USB_EP1_TX_SIZE,0x00,         //maximum packet size supported
         0x01,    //polling interval in ms. (for interrupt transfers ONLY)

   //endpoint descriptor
         USB_DESC_ENDPOINT_LEN, //length of descriptor
         USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
         0x01,              //endpoint number and direction (0x01 = EP1 OUT)
         0x02,              //transfer type supported (0 is control, 1 is iso,
2 is bulk, 3 is interrupt)
         USB_EP1_RX_SIZE,0x00,         //maximum packet size supported
         0x01,              //polling interval in ms. (for interrupt transfers
ONLY)

  };

#define USB_NUM_HID_INTERFACES   0
```

```
#define USB_MAX_NUM_INTERFACES   1

   const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

   #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
      #error USB_TOTAL_CONFIG_LEN not defined correctly
   #endif

   //device descriptor
   char const USB_DEVICE_DESC[] ={
         USB_DESC_DEVICE_LEN, //the length of this report
         0x01,                //constant DEVICE (0x01)
         0x10,0x01,           //usb version in bcd
         0x00,                //class code (if 0, interface defines class.
         0x00,                //subclass code
         0x00,                //protocol code
         USB_MAX_EP0_PACKET_LENGTH,   //max packet size for endpoint 0.
         0xD8,0x04,           //vendor id 0x04D8 is Microchip
         0x11,0x00,           //product id
         0x01,0x00,           //device release number
         0x01,                //index of string description of manufacturer.
therefore we point to string_1 array (see below)
         0x02,                //index of string descriptor of the product
         0x00,                //index of string descriptor of serial number
         USB_NUM_CONFIGURATIONS   //number of possible configurations
};


const char USB_STRING_DESC_OFFSET[]={0,4,46};

#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
   //string 0
         4, //length of string index
         USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
         0x09,0x04,   //Microsoft Defined for US-English
   //string 1 --> la compañia del producto ???
         42, //length of string index
         USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
         'L',0,
         'A',0,
         'T',0,
         'I',0,
         ' ',0,
         'U',0,
         'N',0,
         'A',0,
         'M',0,
         ' ',0,
         'F',0,
         'E',0,
         'S',0,
         ' ',0,
         'A',0,
         'r',0,
         'a',0,
         'g',0,
         'ó',0,
         'n',0,
   //string 2 --> nombre del dispositivo
         42, //length of string index
```

```
        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
        'P',0,
        'r',0,
        'o',0,
        'g',0,
        'r',0,
        'a',0,
        'm',0,
        'a',0,
        'd',0,
        'o',0,
        'r',0,
        ' ',0,
        'L',0,
        'a',0,
        't',0,
        'i',0,
        '-',0,
        'U',0,
        'S',0,
        'B',0
};

#ENDIF
```

```
/*****************************************
* Declaracion de funciones
* del estado Init
*****************************************/
void ConnectToHost(void);
void Initiating();

/*****************************************
* Funcion: Init
* Descripcion: Se encarga de la inicialización
*              del puerto USB, variables
*              globales, conexcion con la PC,
*              estado de las termiales de
*              programacion
* Entradas: Ninguna
* Salidas: Ninguna
* Responde al mensaje INIT_STATE_MACHINE
*****************************************/
void Initiating()
{
   // Inicializar las lineas de programación
   OffVdd();
   OffVpp1();
   OffVpp2();
   OnData();
   OnClk();

   // Inicializar las variables globales de configuracion del programador
   bVerifyProgram = ON_VERIFY_PROGRAM;
   RangeFlash = 0;
   RangeEeprom = 0;
   VppPin = SET_VPP1;
   CommandSet = SET_ICSP_14_BIT;

   // Actualizar indicadores
   LedOff(LED_GREEN);
   LedOn(LED_RED);
   LedOn(LED_BUSY);

   // Inicializacion y configuración del puerto USB
   usb_init();                      // Inicializar USB
   usb_task();                      // Habiliar USB
   usb_wait_for_enumeration();      // Esperar la enumeración desde el HOST

   if(usb_enumerated())
   {
      // Actualizar indicadores *Led Rojo encendido
      LedOff(LED_GREEN);
      LedOn(LED_RED);

      ConnectToHost();              //Establecer conexcion con la PC

      // Actualizar Indicadores *Led Verde encendido
      LedOff(LED_RED);
      LedOn(LED_GREEN);
      LedOff(LED_BUSY);

      // Salir al siguiente estado
      StateMachine = STATE_MACHINE_WAITING;
   }
   else
   {
      // ERROR: No se realizo la Enumeracion desde el HOST
```

```
      // Salir al siguiente estado
      StateMachine = STATE_MACHINE_WAITING;
   }
}

/*****************************************
* Funcion: ConnectToHost
* Descripcion: Espera por un mensaje de
*              __MSG__ARE_YOU_HERE de tipo ASK
*              desde la PC y regresa a la PC
*              el mensaje __MSG__IM_HERE de
*              tipo ASK
* Entradas: Ninguna
* Salidas: Ninguna
*****************************************/
void ConnectToHost(void)
{
    do{
        WaitPacket();
        GetUsbPacket();
      }while( !((RxCommand == __MESSAGE__ASK) && (RxData_H ==
__MSG__ARE_YOU_HERE)) );

      SendMessage(__MESSAGE__ASK, __MSG__IM_HERE);
}




/*****************************************
* Comandos para el estado WAITING
* TxCommand = __COMMAND__WAITING
* Valores validos para TxData_H:
*****************************************/
#define  __CMD__WAIT__INITING_         STATE_MACHINE_INITING      //RESERVADO
No se usa en esta version de firmware
#define  __CMD__WAIT__WAITING_         STATE_MACHINE_WAITING      //RESERVADO
No se usa en esta version de firmware
#define  __CMD__WAIT__TESTING_         STATE_MACHINE_TESTING
#define  __CMD__WAIT__BOOTLOADING_     STATE_MACHINE_BOOTLOADING
#define  __CMD__WAIT__CONFIGURING_     STATE_MACHINE_CONFIGURING
#define  __CMD__WAIT__PROGRAMMING_12_  STATE_MACHINE_PROGRAMMING_12
//RESERVADO No se usa en esta version de firmware
#define  __CMD__WAIT__PROGRAMMING_14_  STATE_MACHINE_PROGRAMMING_14
#define  __CMD__WAIT__PROGRAMMING_16_  STATE_MACHINE_PROGRAMMING_16
//RESERVADO No se usa en esta version de firmware


/*****************************************
* Funcion: Waiting
* Descripcion: Se encarga de direccionar los
*              cambios en la maquina de estados
* Entradas: Ninguna
* Salidas:  Ninguna
* Responde al mensaje STATE_MACHINE_WAITING
*
* RxCommand: __COMMAND__WAITING
* RxData_H:  Nuevo estado
* RxData_L:  No usado
*****************************************/

void waiting(void)
```

```
{
   WaitPacket();     // Esperar un paquete del HOST
   GetUsbPacket();   // Obtener el paquete

   if(RxCommand == __COMMAND__WAITING)
   {
      switch(RxData_H)
      {
         // Entrar en modo de PRUEBA
         case __CMD__WAIT__TESTING_:
                  StateMachine = STATE_MACHINE_TESTING;
                  break;

         // Entrar en modo de PROGRAMACION_PIC
         case __CMD__WAIT__PROGRAMMING_14_:
                  StateMachine = STATE_MACHINE_PROGRAMMING_14;
                  break;

         // Entrar en modo de CONFIGURACION
         case __CMD__WAIT__CONFIGURING_:
                  StateMachine = STATE_MACHINE_CONFIGURING;
                  break;

         // Entrar en modo de ACTUALIZACION
         case __CMD__WAIT__BOOTLOADING_:
                  StateMachine = STATE_MACHINE_BOOTLOADING;
                  break;

         default: //ERROR: Entrada no válida para el estado WAIT
                  StateMachine = STATE_MACHINE_WAITING;
                  break;
      }
   }
   else  // ERROR: No se trata de un comando para el estado WAIT
   {
      StateMachine = STATE_MACHINE_WAITING;
   }
}
```

```
/*****************************************
State_testing.h
Codigo para el estado de prueba
UANM FES Aragon 2008
*****************************************/

//Comandos para el estado de depuracion
//TxCommand = __COMMAN__PROGRAM
//Valores para: TxData_H
#define  __CMD__CONFIG__RESERVED__           0x00
#define  __CMD__CONFIG__SET_VPP1__           0x01
#define  __CMD__CONFIG__SET_VPP2__           0x02
#define  __CMD__CONFIG__SET_ICSP_12_BIT__    0x03
#define  __CMD__CONFIG__SET_ICSP_14_BIT__    0x04
#define  __CMD__CONFIG__SET_ICSP_16_BIT__    0x05
#define  __CMD__CONFIG__ON_VERIFY_PROGRAM__  0x06
#define  __CMD__CONFIG__OFF_VERIFY_PROGRAM__ 0x07


void Configuring(void);
void ConfigSetVpp1(void);
```

```
void ConfigSetVpp2(void);
void ConfigSetICSP12Bit(void);
void ConfigSetICSP14Bit(void);
void ConfigSetICSP16Bit(void);
void ConfigOnVerifyProgram(void);
void ConfigOffVerifyProgram(void);
void ConfigSetRangeFlash(void);
void ConfigSetRangeEEPROM(void);


void Configuring(void)
{
   switch(RxData_H)
   {
      case  __CMD__CONFIG__SET_VPP1__:
               ConfigSetVpp1();
               break;

      case  __CMD__CONFIG__SET_VPP2__:
               ConfigSetVpp2();
               break;

      case  __CMD__CONFIG__SET_ICSP_12_BIT__:
               ConfigSetICSP12Bit();
               break;

      case  __CMD__CONFIG__SET_ICSP_14_BIT__:
               ConfigSetICSP14Bit();
               break;

      case  __CMD__CONFIG__SET_ICSP_16_BIT__:
               ConfigSetICSP16Bit();
               break;

      case  __CMD__CONFIG__ON_VERIFY_PROGRAM__:
               ConfigOnVerifyProgram();
               break;

      case  __CMD__CONFIG__OFF_VERIFY_PROGRAM__:
               ConfigOffVerifyProgram();
               break;
   }
   StateMachine = STATE_MACHINE_WAITING;
}

void ConfigSetVpp1(void)
{
   VppPin = VPP1;
}

void ConfigSetVpp2(void)
{
   VppPin = VPP2;
}

void ConfigSetICSP12Bit(void)
{
   CommandSet = SET_ICSP_12_BIT;
}

void ConfigSetICSP14Bit(void)
{
   CommandSet = SET_ICSP_14_BIT;
```

```
}

void ConfigSetICSP16Bit(void)
{
   CommandSet = SET_ICSP_16_BIT;
}

void ConfigOnVerifyProgram(void)
{
   bVerifyProgram = ON_VERIFY_PROGRAM;
}

void ConfigOffVerifyProgram(void)
{
   bVerifyProgram = OFF_VERIFY_PROGRAM;
}
```

```
/*****************************************
* Comandos para el estado PROGRAMMING14
* TxCommand = __COMMAN__PROGRAM_14
* Valores para: TxData_H
*****************************************/
#define  __CMD__PROGRAM__EXIT_              0x00
#define  __CMD__PROGRAM__LOAD_DATA_FLASH_   0x01
#define  __CMD__PROGRAM__READ_DATA_FLASH_   0x02
#define  __CMD__PROGRAM__LOAD_DATA_EEPROM_  0x03
#define  __CMD__PROGRAM__READ_DATA_EEPROM_  0x04
#define  __CMD__PROGRAM__LOAD_CONF_WORD_    0x05
#define  __CMD__PROGRAM__READ_CONF_WORD_    0x06
#define  __CMD__PROGRAM__ERASE_DATA_FLASH_  0x07
#define  __CMD__PROGRAM__ERASE_DATA_EEPROM_ 0x08

/*****************************************
* Declaracion de funciones
* del estado PROGRAMMING14
*****************************************/
void ProgramLoadDataFlash(void);
void ProgramReadDataFlash(void);
void ProgramLoadDataEeprom(void);
void ProgramReadDataEeprom(void);
void ProgramLoadConfWord(void);
void ProgramReadConfWord(void);
void ProgramEraseFlash(void);
void ProgramEraseFeprom(void);
void ProgramExit(void);
void ProgramConfigCmd(void);

/*****************************************
* Funcion: Programming14
* Descripcion: Se encarga de ejecutar el comando
*              de programacion proveniente de la PC
* Entradas: Ninguna
* Salidas:  Ninguna
* Responde al mensaje PROGRAM_14_STATE_MACHINE
* RxCommand: __COMMAND__PROGRAM
* RxData_H:  Comando del estado PROGRAM
* RxData_L:  No usado
*****************************************/
```

```
void Programming14(void)
{
   switch(RxData_H)
   {
      case __CMD__PROGRAM__LOAD_DATA_FLASH_:
            ProgramLoadDataFlash();
            break;

      case __CMD__PROGRAM__READ_DATA_FLASH_:
            ProgramReadDataFlash();
            break;

      case __CMD__PROGRAM__LOAD_DATA_EEPROM_:
            ProgramLoadDataEeprom();
            break;

      case __CMD__PROGRAM__READ_DATA_EEPROM_:
            ProgramReadDataEeprom();
            break;

      case __CMD__PROGRAM__LOAD_CONF_WORD_:
            ProgramLoadConfWord();
            break;

      case __CMD__PROGRAM__READ_CONF_WORD_:
            ProgramReadConfWord();
            break;

      case __CMD__PROGRAM__ERASE_DATA_FLASH_:
            ProgramEraseFlash();
            break;

      case __CMD__PROGRAM__ERASE_DATA_EEPROM_:
            ProgramEraseFeprom();
            break;

      default:
            // ERROR: No se trata de un argumento para el estado PROGRAM
            ProgramExit();
            StateMachine = STATE_MACHINE_WAITING;
            break;
   }
   ProgramExit();
}

void ProgramExit()
{
   OffProgramVerifyTestMode();
   StateMachine = STATE_MACHINE_WAITING;
}

void ProgramLoadDataFlash(void)
{
     int8 conta = 0;
   int8 bContinue = TRUE;

   OnProgramVerifyTestMode();

   do{
      WaitPacket();
      GetUsbPacket();

      Command14(LOAD_DATA_FLASH);
```

```
      LoadData14(RxData_H, RxData_L);
      Command14(START_PROGRAM);

      if(bVerifyProgram == ON_VERIFY_PROGRAM)
      {
         //Verificar el programa
         Command14(READ_DATA_FLASH);
         ReadData14(&TxData_H, &TxData_L);
         //TxMesage = SendUSB
         Message = __MESSAGE__ASK;
         SetUsbPacket();
      }
      Command14(INC_COUNTER);
      delay_ms(8);
   }while(bcontinue && conta < RangeFlash );

   ProgramExit();

}

void ProgramLoadDataEeprom(void)
{
   int8 conta = 0;
   int8 bContinue = TRUE;

   OnProgramVerifyTestMode();

   do{
      WaitPacket();
      GetUsbPacket();

      Command14(LOAD_DATA_EEPROM);
      LoadData14(RxData_H, RxData_L);
      Command14(START_PROGRAM);

      if(bVerifyProgram == ON_VERIFY_PROGRAM)
      {
         //Verificar el programa
         Command14(READ_DATA_EEPROM);
         ReadData14(&TxData_H, &TxData_L);
         //TxMesage = SendUSB
         Message = __MESSAGE__ASK;
         SetUsbPacket();
      }
      Command14(INC_COUNTER);
      delay_ms(8);
   }while(bcontinue && conta < RangeEeprom );

   ProgramExit();
}
```

Bibliografía

Bibliografía

Libros

- ANGULO, José Ma. *Microcontroladores PIC. La Clave del Diseño.* Thomson. España. 2003.
- ANGULO, José Ma. *Microcontroladores PIC. Diseño Práctico de aplicaciones.* Mc Graw-Hill. España. Tercera edición. 2006.
- AXELSON, Jan. *USB Complete*. Lakeview Research. Madison. Tercera Edición, 2005.
- CEBALLOS, Francisco J. *Visual C++ Aplicaciones para Win32*. Alfaomega. España. 2000. Segunda Edición
- CEBALLOS, Francisco J. *Visual C++ Programación Avanzada en Win32*. Alfa Omega. España. 1999.
- CEBALLOS, Francisco J. *Curso de C/C++*. Alfa Omega. España. 2002
- MALVINO, Albert. *Principios de Electrónica*. Mc Graw Hill. 7ma Edición. España. 2007.
- MONTILLA, Fulgencio. *Fuentes de alimentación*. Universidad Politécnica de Valencia. 1997.
- PALACIOS, Enrique. *Microcontrolador PIC16F84 Desarrollo de Proyectos*. Alfaomega Ra-Ma. Primera Edición, México. 2004.
- PEÑALOZA, Ernesto, *Fundamentos de Programación C/C++*, Alfaomega, Tercera Edición, México, 2004.
- TANENBAUM, Andrew S. *Organización de Computadoras. Un Enfoque Estructurado.* Pearson Educación. Cuarta Edición. 2000.

Manuales

- *An Introduction to USB Descriptors with a Game Port to USB Game Pad Translator Example.* Reston Condit. Microchip Technology Inc. 2004.
- *Calculating Program Memory Checksums Using a PIC16F87X.* Rodger Richey. Microchip Technology Inc. 1998.
- *Downloading HEX Files to PIC16F87X PICmicro® Microcontrollers.* Rodger Richey. Microchip Technology Inc. 1998.
- *Low Cost USB Microcontroller Programmer. The building of the PICkit™ 1 FLASH Starter Kit.* Reston Condit. Dan Butler. Microchip Technology Inc. 2003.
- *Generating High Voltage Using the PIC16C781/782.* Ross Fosler. Microchip Technology Inc. 2005.
- *Implementing a Bootloader for the PIC16F87X.* Mike Garbutt. Microchip Technology Inc. 2000.
- *In-Circuit Serial Programming™(ICSP™) Guide.* Microchip Technology Inc. 2003.
- *Midrange and Enhanced Reference Manuals (Chapter 2. Oscillator).* Microchip Technology Inc.
- *Midrange and Enhanced Reference Manuals (Chapter 3. Reset).* Microchip Technology Inc.
- *Oscillator, Overview, Design Tips and Troubleshooting of the PICmicro®, Microcontroller Oscillator.* Microchip Technology Inc.