ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

DISEÑO Y CONSTRUCCIÓN DE UN PROGRAMADOR DE MICROCONTROLADORES PIC Y ATMEL MEDIANTE EL PUERTO USB DEL PC.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

GANDHI JAVIER ACOSTA HERRERA
WILSON IVÁN JAMI GÓMEZ

DIRECTOR: MSc. PATRICIO CHICO H

Quito, agosto del 2006

DECLARACIÓN

Nosotros, Gandhi J. Acosta Herrera y Wilson I. Jami Gómez, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____ _____

Gandhi J. Acosta Herrera Wilson I. Jami Gómez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Gandhi Acosta Herrera y Wilson Jami Gómez, bajo mi supervisión.

_____

MSc. Patricio Chico H.

CONTENIDO

RESUMEN

El presente trabajo abarca todo lo relativo al proceso de programación de las familias de microcontroladores PIC (familias 16CXX, 16FXX, 16FXXA, 16F87X y 16F87XA) y ATMEL (AT89CXX y AT89SXX), hasta llegar al punto culminante de implementar un programador.

Este programador está constituido por el hardware y su respectivo software, el enlace entre ambos es mediante la interfaz USB. Además se aprovecha la energía que provee el puerto, eliminándose así la necesidad de una fuente externa.

Además para darle generalidad al proceso de programación de los dispositivos que fabrica Microchip (PIC), y en general cualquier dispositivo que pueda programarse en forma serial, también se ha implementado una herramienta dentro del software, que le permite al usuario de una forma sencilla incorporar un nuevo elemento a la lista de elementos que posee el programador, con esta idea el programar va creciendo y se va actualizando conforme a las necesidades del usuario. Esto constituye una gran ventaja frente a los tradicionales programadores de dispositivos seriales, los cuales soportan un limitado número de dispositivos.

PRESENTACIÓN

El proyecto que se presenta cubre todas las áreas concernientes al proceso de programación de un determinado grupo de elementos de las familias de microcontroladores PIC y ATMEL, y para lograr este fin se ha desarrollado un programador cuya interfaz de comunicación es a través del puerto USB.

Se ha empezado a atacar este problema desde el conocimiento de la norma USB, la cual constituye todo un protocolo de comunicaciones, y por lo tanto involucra un cierto grado de complejidad. Por lo que en el Capítulo 1 se ha visto en la necesidad de poner énfasis en la explicación de los puntos más relevantes de la norma, tal como las generalidades, terminología, elementos de la comunicación, y lo que se considera esencial para cualquier desarrollador de dispositivos USB.

Para que un microprocesador sea programado, se debe conocer la forma como se han de escribir, leer y borrar sus respectivas localidades de memoria, siguiendo el proceso que diga el fabricante, y esto es justo lo que cubre el Capítulo 2.

El Capítulo 3 abarca todo lo relativo al diseño e implementación de los circuitos electrónicos, los cuales satisfacen los niveles de voltaje que se necesiten aplicar a los pines de programación de los elementos propuestos. Toda la energía requerida en la programación se la toma del puerto USB, por lo que se implementa un circuito elevador.

Cubierta las partes concernientes a la forma de programación, y los niveles de voltaje que requiere determinado dispositivo, resta por diseñar e implementar un software que facilite esta tarea, el mismo que de una manera amigable le permita al usuario programar sus microcontroladores. Dentro del Capítulo 4 se presenta el software implementado para la programación de microcontroladores PIC y ATMEL. Además en este capítulo se desarrolla una herramienta que le permitirá

al usuario añadir nuevos microcontroladores PIC a la lista de elementos que posee el programador.

En el Capítulo 5, se comprueba el buen funcionamiento del programador, con la puesta en marcha de un conjunto de proyectos que han sido desarrollados dentro de los respectivos laboratorios de Electrónica.

Por último en el Capítulo 6, se presentan conclusiones y recomendaciones concernientes a este proyecto de titulación.

CAPÍTULO 1

DESCRIPCIÓN DEL PÓRTICO USB

En el presente capítulo se revisa la norma USB y se explica las características más relevantes para que el usuario culmine con éxito la comunicación de datos entre el PC y un artefacto USB.

Se empieza con un acercamiento genérico a la norma USB, para posteriormente dar a conocer los elementos involucrados en la comunicación, funciones, y como ellos se enlazan.

1.1 CARACTERÍSTICAS GENERALES DE LA NORMA USB

El USB constituye una manera de intercambiar datos en forma serial entre un computador y un artefacto. La norma USB es todo un protocolo de comunicaciones, a diferencia de lo que constituye por ejemplo la norma RS 232, la complejidad del USB se hace notar desde el principio, debido a que si la computadora desea intercambiar los datos propios de alguna determinada aplicación, tuvo que anteriormente conocer todas las características del artefacto conectado, dentro de la características más relevantes se pueden citar: la velocidad, la cantidad de datos transferidos, números identificadores del artefacto.

Por este motivo si se desea enviar un único byte de información, este irá acompañado de un conjunto de bits de soporte para lograr que el byte de información llegue con el mínimo de errores hacia el receptor de la comunicación, pero no hay que preocuparse de estos bits que se añaden al dato porque la norma USB ya se encarga de eso.

El cable USB está constituido de cuatro alambres, dos de los cuales corresponden a la energía (5 V, GND) y los dos restantes son los encargados de transportar los datos de una manera diferencial. (D- y D+).

Las partes esenciales dentro de cualquier comunicación la constituyen dos elementos. En el caso específico de la norma USB estos dos elementos son la computadora y un artefacto externo. Dentro de la terminología del protocolo USB a la computadora se la nombra como Host, mientras que al artefacto se lo denomina Dispositivo **[1]** por otro lado esta norma permite que se puedan conectar simultáneamente hasta 127 Dispositivos a un Host.

Tanto el Host como el Dispositivo poseen unidades inteligentes que les permiten cifrar y descifrar los datos que viajan por el par de cables de una forma automática, así el usuario únicamente se limita a recibir o enviar sus datos. El dispositivo al poseer circuitos controladores inteligentes necesita polarizarlos con una fuente de poder, razón por la cual podría tomar esta energía del par de cables de alimentación que entrega el puerto USB. En caso de que los circuitos del Dispositivo requieran energía más allá de la que pueda proveer el puerto USB (5V, 500 mA) **[1],** poseerá su propia fuente.

Esta interfaz combina muchas ventajas, como son: fácil de conectar, configuración automática, fuente de energía disponible, rapidez y flexibilidad en la transferencia de datos, bajo costo, entre otros beneficios.

La velocidad a la que viajan los datos, describe la proporción de información que viaja en los cables a una determinada unidad de tiempo.

USB soporta tres velocidades: Baja Velocidad (1.5 Megabits/seg), Mediana Velocidad (12 Megabits/seg) y Alta Velocidad (480 Megabits/seg) **[1].**

Pero la velocidad real de un dato transferido, es menor que la velocidad del bus. Esto se debe a que, el bus además de los datos, debe llevar el estado, control y señales de comprobacion de errores. Además hay que tomar en cuenta que todos los periféricos deben compartir el mismo camino de transferencia de datos, por lo tanto la velocidad de la transferencia también depende del tipo de transferencia que se use y que tan ocupado esté el bus en ese instante. Por estas razones la máxima velocidad teórica de la transferencia de un dato es aproximadamente: 800 bytes/seg a baja velocidad, 1.2 Megabytes/seg a full velocidad y 53 Megabytes/seg a alta velocidad.

Los fabricantes de microcontroladores USB ponen a disposición una amplia gama de circuitos integrados, de esta forma el usuario puede escoger el que más se ajuste a sus necesidades. Una de las características a tener en cuenta corresponde la velocidad y versión de USB a la que pertenece dicho elemento.

El cable USB se conecta al Host a través de un puerto USB tipo A, mientras que por el otro lado se conecta a un Dispositivo a través de un puerto USB tipo B.

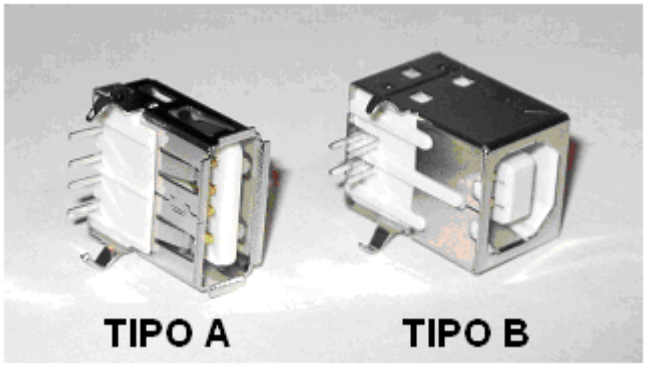


FIGURA 1-1: CONECTORES TÍPICOS USB [6].

Para que un Host pueda soportar a los 127 Dispositivos que nombra la norma USB debe ayudarse de un Dispositivo en particular denominado Hub, este Dispositivo hace las veces de un multiplexor de puertos, USB usa una topología en estrella dispuesta en filas, en donde cada Hub es el centro de una estrella que puede conectarse a varios Dispositivos o Hubs adicionales hasta completar los 127 Dispositivos según se muestra en la Figura 1-2. Cada Host posee por defecto un Hub interno denominado Hub Raíz.

La longitud máxima de cable USB, desde un periférico hasta el Host es de 5m de largo. Con Hubs intermedios, un periférico puede estar hasta 30 m del Host. **[1].**

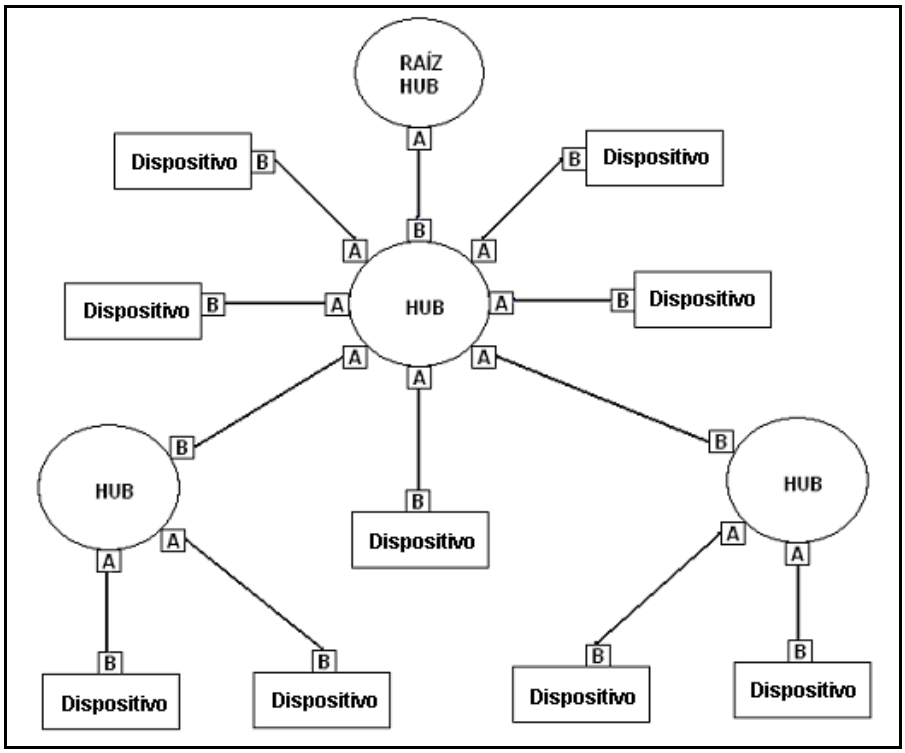


FIGURA 1-2: TOPOLOGÍA DEL BUS USB

1.2 FORMA DE FUNCIONAMIENTO DE UNA COMUNICACIÓN USB

Como anteriormente se comentó, la comunicación USB es todo un protocolo, así por ejemplo, al conectar un Dispositivo, el Host debe percatarse de su presencia, y posteriormente solicitará todo un conjunto de información al Dispositivo, con esta información el Host decide que vínculo usar para intercambiar los datos, a este vínculo se lo denominará como Driver, según la terminología utilizada por la norma USB **[1].**

El proceso de identificación que el Host realiza con un Dispositivo se denomina Enumeración y este proceso es común para cualquier tipo de Dispositivo USB.

Dentro de este proceso de Enumeración, el Host realiza una lectura de tablas que residen dentro de un Dispositivo, estas tablas poseen información acerca de las características de dicho Dispositivo, a estas tablas según la norma USB se las denomina Descriptores.

Luego de que un Dispositivo haya sido enumerado, se estará en condiciones de intercambiar cualquier tipo de información concerniente al funcionamiento del Dispositivo.

El Host es el que inicia las comunicaciones, esto es, el Dispositivo está subordinado y únicamente responderá cuando el computador lo requiera.

Cada vez que el Host solicite información, el Dispositivo le hará llegar la correspondiente respuesta. El Dispositivo logra esto mediante su controlador, el cual es el encargado de ubicar los datos sobre el cable USB, estos datos anteriormente han de ser sacados de unos buffers denominados Endpoints. El caso inverso en el cual se estén enviando datos hacia el Dispositivo, estos irán directamente a depositarse en estas mismas localidades de memoria. Entonces al Endpoint se lo denomina como un conjunto de espacios de memoria que residen dentro del Dispositivo.

Haciendo un símil con los puertos físicos que posee cualquier microcontrolador y la forma como éstos deben de ser configurados, cuando se quiera escribir información hacia el exterior o al contrario recibir información, un determinado Endpoint también será configurado como de entrada o de salida antes de usarlo. Endpoints sólo existen en los Dispositivos, no en el Host.

La Enumeración es invisible al usuario, sólo se la vería si en un primer enlace, Windows no encontrase el Driver correspondiente al Dispositivo.

Los Endpoints dentro del Dispositivo consisten de un número y sentido. El número es un valor de 0 a 15. El sentido se define según lo que se va a realizar, es decir: un Endpoint IN provee datos para enviar al Host y un Endpoint OUT almacena datos enviados desde el Host.

Durante la Enumeración la información transferida entre el Host y el Dispositivo es mediante el Endpoint cero.

Para mediana o alta velocidad además de Endpoint 0 los Dispositivos pueden soportar 30 Endpoints adicionales. En cambio para baja velocidad se soporta 2 Endpoints adicionales además del Endpoint 0.

Al llegar una transacción OUT, el Endpoint guarda los datos y el Dispositivo dispara una interrupción para procesar los datos recibidos.

Al llegar una transacción IN, se envían los datos desde el Endpoint especificado y luego se dispara una interrupción para tener listos los datos para la próxima transacción.

Todos los Dispositivos USB usan Endpoint 0 para administrar la comunicación entre el Dispositivo y el PC. Esto quiere decir que Endpoint 0 es bidireccional y en realidad son dos Endpoints, EP0 in y EP0 out.

1.3 TIPOS DE TRANSFERENCIA QUE PERMITE EL USB.

Una Transferencia es el nombre técnico que da la norma USB para indicar cualquier tipo de intercambio de datos en general **[1].**

Cada tipo de Transferencia tiene habilidades y límites, lo que hace a cada Transferencia conveniente para diferentes propósitos

Existen cuatro tipos de Transferencia USB y son: Control, Masiva, Interrupción y Asincrónica. La Tabla 1-1, resume las características y usos de los diferentes tipos de Transferencias.

Tipo de Transferencia	**Control**	**Masiva**	**Interrupción**	**Asincrónica**
Uso Típico	Identificación y Configuración	Impresión, Scanner, Drive	Mouse, Teclado	Secuencia Audio, Video
Requerido?	si	no	no	no
Permite baja velocidad?	si	no	si	no
Datos bytes/mseg por Transferencia, máximo posible por Endpoint (alta velocidad)	15 872	53 248	24 576	24 576
Datos bytes/mseg por Transferencia, máximo posible por Endpoint (mediana velocidad)	832	1216	64	1023
Datos bytes/mseg por Transferencia, máximo posible por Endpoint (baja velocidad)	24	no permite	0.8	no permite
Dirección del flujo de Datos	IN y OUT	IN u OUT	IN u OUT (USB 1.0 sólo soporta IN)	IN u OUT
Ancho de banda reservado para la Transferencia de este tipo (porcentaje)	10% baja/mediana velocidad, 20% alta velocidad	ninguno	90 % baja/mediana velocidad, 80% alta velocidad	
Corrección de errores?	si	si	si	no
Velocidad de entrega garantizada?	no	no	no	si
Latencia Garantizada (tiempo máximo entre Transferencias)?	no	no	si	si

TABLA 1-1: TIPOS DE TRANSFERENCIA USB [1]

1.3.1 TRANSFERENCIA DE CONTROL

Las Transferencias de Control le permiten al Host conocer sobre el Dispositivo a través de un conjunto de peticiones estándar que se encuentran definidas por la norma USB.

Todo Dispositivo debe soportar Transferencias de Control sobre el Endpoint 0.
Cada Transferencia de Control debe transferir la información en ambas direcciones.

Para Dispositivos de baja velocidad, el máximo tamaño de paquete de datos es 8 bytes. Para mediana velocidad, el máximo es 64 bytes. Para alta velocidad, el máximo debe ser 64 bytes **[1].**

Las Transferencias de Control no son la mejor manera de transferir datos, se trata de utilizarlas sólo para conocer acerca de las características de un Dispositivo. En caso de manejar datos, cada Transferencia de Control incluye un sobre encabezado para baja velocidad de 63 bytes, 45 bytes para mediana velocidad o 173 bytes para alta velocidad.

1.3.2 TRANSFERENCIA MASIVA

Se utiliza este tipo de Transferencia cuando el tiempo de transmisión no es crítico. En el caso de que algunos Dispositivos estén ocupando el puerto USB al mismo tiempo, provocarán que la Transferencia Masiva se demore, caso contrario son el tipo de Transferencia más rápida que hay.

Sólo Dispositivos de mediana y alta velocidad soportan este tipo de Transferencia.

Para Dispositivos de mediana velocidad, el máximo tamaño de paquete de datos en una Transferencia Masiva es de 64 bytes. Para alta velocidad, el máximo tamaño debe ser 512 bytes **[1].**

Cada Transferencia Masiva, adiciona un sobre encabezado a sus datos de 13 bytes para mediana velocidad y 55 bytes para alta velocidad.

1.3.3 TRANSFERENCIAS DE INTERRUPCIÓN

Transferencias de Interrupción son utilizadas por Dispositivos que necesitan recibir atención del Host periódicamente.

Dispositivos de baja velocidad, que únicamente usan Transferencias de Control e Interrupción, son los que con mucha certeza ocuparán las de interrupción para transferir los datos del usuario.

Las Transferencias de interrupción pueden realizarse a cualquiera de las tres velocidades.

Para Dispositivos de baja velocidad, el máximo tamaño de paquete de datos puede ser de 8 bytes, para mediana velocidad, el máximo puede ser de 64 bytes, y para alta velocidad, el máximo es de 1024 bytes **[1].**

Al enviar datos cada Transferencia de Interrupción incluye un sobre encabezado de 19 bytes para baja velocidad, 13 bytes para mediana velocidad y 55 bytes para alta velocidad.

1.3.4 TRANSFERENCIA ASINCRÓNICA

Transferencias Asincrónicas son conocidas como Transferencias en tiempo real, ya que se garantizan el tiempo de entrega. Pero este es el único tipo de Transferencia que no soporta retransmisión automática de datos recibidos con errores, deben tolerarse los errores ocasionales.

Sólo Dispositivos de mediana y alta velocidad soportan este tipo de Transferencia.

Para Dispositivos de mediana velocidad, el máximo tamaño de paquete de datos en una Transferencia Asincrónica de 1023 bytes de datos. Para alta velocidad, el máximo tamaño es de 1024 bytes **[1].**

Cada Transferencia Asincrónica incluye un sobre encabezado de 38 bytes para una Transferencia.

1.4 ESTRUCTURACIÓN DE DATOS PARA LA COMUNICACIÓN

El Host controla el tráfico de datos dividiendo los tiempos en pedazos llamados Tramas (cada trama para baja y mediana velocidad es de 1 ms) (Ver Figura 1-3) y en Microtramas (125 µs para alta velocidad), es decir que divide a una trama en ocho microtramas.

FIGURA 1-3: TRAMAS USB A BAJA Y MEDIANA VELOCIDAD [1]

Un intercambio de información se compone de una o más transacciones, que dependiendo de cómo el Host maneja cada transacción y de que tan rápido un Dispositivo contesta, las transacciones pueden ir en una o varias tramas.

Cada transacción incluye identificación, verificación de errores, estado y control de información además de los datos a ser intercambiados..

Una transacción debe completarse ininterrumpidamente, ninguna otra comunicación puede interferir en medio de una transacción.

Existen tres tipos de transacción que se explican en la Tabla 1-2.

Tipo de Transacción	Fuente de Datos	Tipos de Transferencia que usan este tipo de Transacción	Contenido
IN	Dispositivo	todas	datos o estado de información
OUT	Host	todas	datos o estado de información
SETUP	Host	control	una demanda

TABLA 1-2: CARACTERISTICAS DE CADA TIPO DE TRANSACCION USB [1]

Una Transacción SETUP es como una Transacción OUT, es decir que los datos viajan desde el Host al Dispositivo, pero esta transacción da el comienzo de una Transferencia de Control y deben siempre ser aceptadas por el Dispositivo.

Todo lo que envía un Dispositivo está en respuesta a lo que el Host ha enviado primero (Figura 1-4).

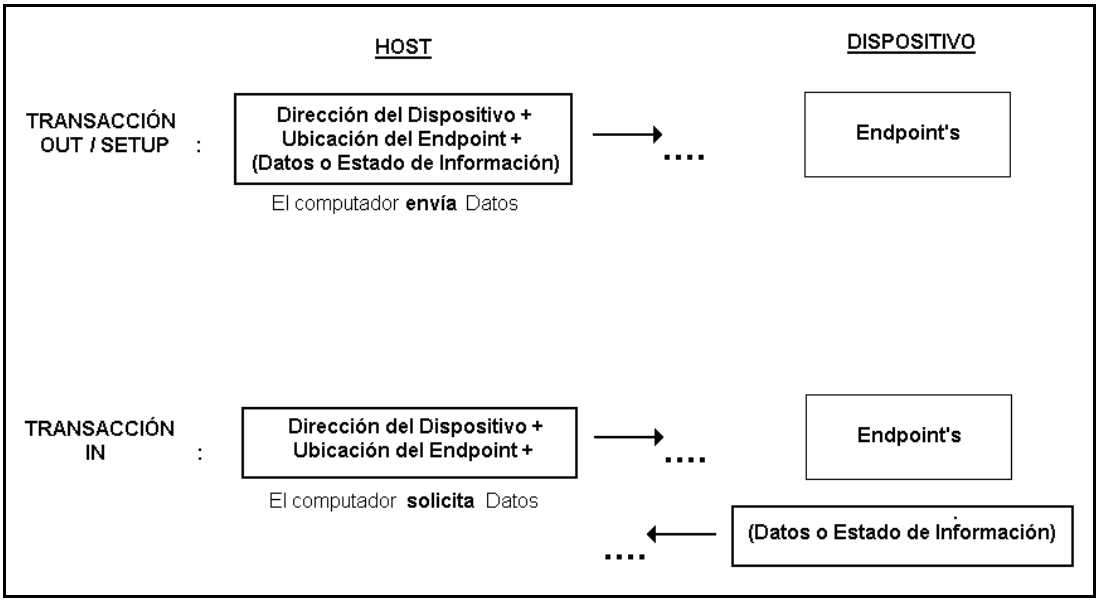


FIGURA 1-4: DIRECCIÓN DEL FLUJO DE DATOS

Cada transacción comienza cuando el Host envía un bloque con la dirección del Dispositivo y ubicación Endpoint dentro del Dispositivo.

Cada transacción se compone de dos o tres paquetes, que ocurren en secuencia: Token, Data (Opcional) y Handshake. A su vez cada paquete esta compuesto de uno o más bloques de información. Como se observa en la Figura 1-5, cada paquete comienza con un bloque identificador (PID) que contiene información identificatoria concerniente a dicho paquete.

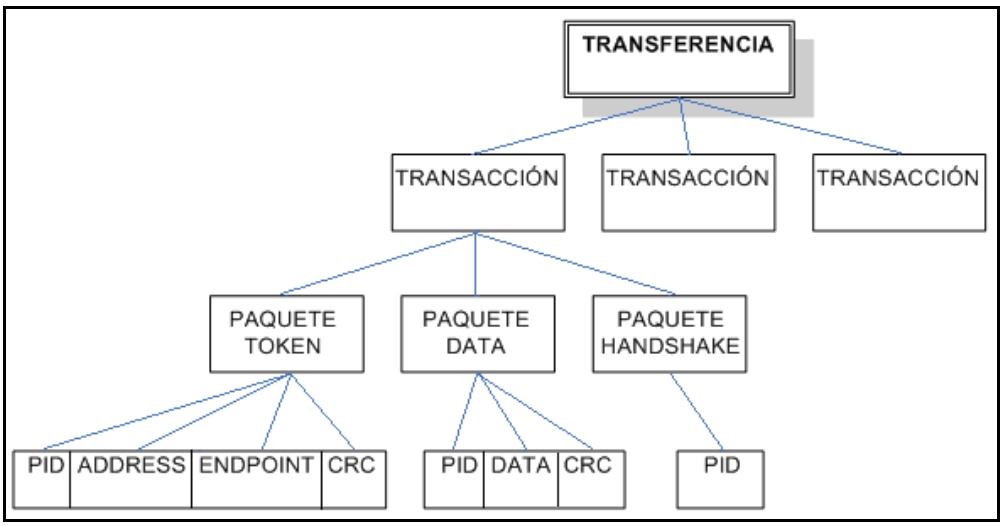


FIGURA 1-5: ORGANIGRAMA CONSTITUTIVO DE UNA TRANSFERENCIA [1]

Cada paquete contiene: un PID, información del usuario y un CRC (Chequeo de Redundancia Cíclica).

A través del Paquete Token el Host indica el inicio de una comunicación. El PID indica el tipo de transacción que se va ha realizar, tal como: Setup, IN u OUT, además aquí se puede indicar el Comienzo de Trama (SOF), que es la referencia que cronometra el tiempo de duración de cada Trama.

En el Paquete Data, en caso de existir, el Host o el Dispositivo envían datos. El PID incluye un código de seguridad llamado Data-Toggle, que es un valor que indica si un dato fue perdido o duplicado durante la transmisión.

En el Paquete Handshake, el Host o el Dispositivo envían código de estado en su PID, este código informa el éxito o fracaso de la transacción.

En toda transacción se sigue la siguiente secuencia de paquetes:

1. Cada transacción comienza con un Paquete Token, que siempre es enviado por el Host, este paquete identifica: el tipo de transacción, el Dispositivo y el Endpoint receptor y presencia o ausencia del Paquete Data

2. Dependiendo de la transacción y si el Host o el Dispositivo tienen la información lista para enviar, entonces un Paquete Data sigue a continuación del Paquete Token.

3. El receptor del Paquete Data (Host o Dispositivo) devuelve un Paquete Handshake, que indica el estado de la transacción (éxito o fracaso), la ausencia de un Paquete Handshake esperado, indica un error muy serio.

1.5 PROCESO DE ENUMERACIÓN

Este proceso comprende todo un conjunto de acciones que el Host realiza sobre un Dispositivo que se conecte por primera vez, para en lo posterior usar todos los recursos que ofrece dicho Dispositivo. Cada una de estas acciones son: la asignación de una dirección al Dispositivo, lectura de Descriptores del Dispositivo, asignación y carga de un Driver al Dispositivo. También este proceso se encarga de configurar requisitos de energía, Endpoints y otras características del Dispositivo.

El Host interroga periódicamente a su Hub raíz para averiguar si un Dispositivo se ha conectado o desconectado (Hubs externos son tratados como si fueran Dispositivos).

Al conocer que un Dispositivo se ha conectado, el Host envía demandas al Hub Raíz, estableciendo así un camino de comunicación entre el Host y el Dispositivo. Entonces el Host envía demandas en Transferencias de Control al Endpoint 0, que deben ser soportadas por todos los Dispositivos.

1.5.1 PASOS DE ENUMERACIÓN

Los pasos mostrados a continuación no tienen ninguna secuencia a seguir.

1.- El usuario conecta el Dispositivo a un puerto USB. En este momento el Hub proporciona energía al puerto y el Dispositivo es polarizado.

2.- El Hub detecta el Dispositivo mediante la supervisión continua de las líneas de datos (D+ y D-). El Hub posee una resistencia de pull down en cada línea de datos. El Dispositivo tiene una resistencia de pull up en D+ si es de mediana velocidad o en D- si el Dispositivo es de baja velocidad.

El pull up del Dispositivo, al conectarse con el Hub, lleva la línea a alto, permitiendo así detectar al Hub que un Dispositivo se ha conectado.

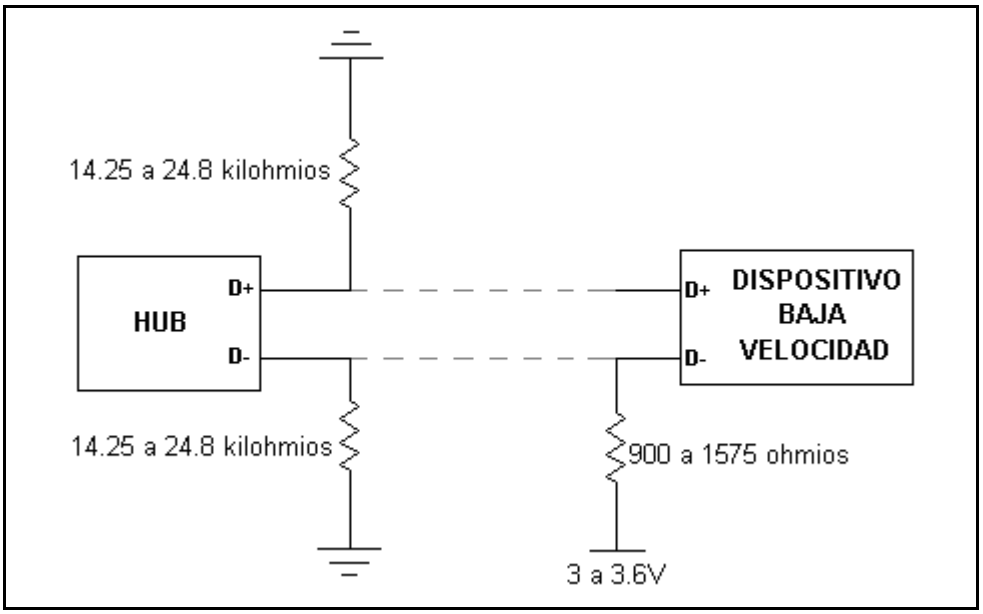


FIGURA 1-6: CONEXIÓN DE UN DISPOSITIVO DE BAJA VELOCIDAD CON UN HUB.

3.- El Host al revisar periódicamente su Hub Raíz, y mediante éste, se entera que un nuevo Dispositivo ha sido conectado o desconectado.

4.- El Hub detecta si el Dispositivo es de baja o mediana velocidad, identificando qué línea de señal de datos (D+ o D-) se puso en alto.

5.- El Hub resetea al Dispositivo. El Host ordena al Hub que resetee al Dispositivo por unos 10 ms.

6.- El Host envía nuevamente una demanda al Hub, para enterarse si el Dispositivo ya terminó su reseteo. Si ya salió del reseteo el Dispositivo toma la dirección 00H y entonces ya existe un camino de comunicación directa entre el Host y el Dispositivo. A partir de aquí el Dispositivo puede recibir Transferencia de Control al Endpoint 0, esto lo hace máximo con 100 mA del puerto.

7.- El Host envía una demanda al Dispositivo, para poder conseguir el tamaño máximo de paquete del Endpoint 0. Por precaución el Host puede resetear nuevamente al Dispositivo.

8.- El Host asigna una nueva dirección al Dispositivo, Toda comunicación desde este punto utilizará esta nueva dirección.

9.- El Host aprende sobre las habilidades del Dispositivo en la nueva dirección asignada, esto lo hace pidiendo sus Descriptotes.

10.- El Host asigna y carga un Driver al Dispositivo. Después de recoger toda la información sobre el Dispositivo, el PC trata de emparejar esta información con algún archivo **.**INF ya existente. En caso de encontrarlo, el archivo **.**INF, informa el nombre y ubicación del Driver que mejor se acople al Dispositivo. Si no lo encuentra, el software de la PC, pedirá al usuario que cargue un Driver propio del fabricante de dicho Dispositivo

1.5.2 DESCONECTANDO UN DISPOSITIVO

Al desconectar un Dispositivo, el Hub Raíz desactiva su puerto y el Host se entera de que algún evento ha ocurrido. Luego Windows quita el Dispositivo de la pantalla Manejador de Dispositivo, esta ventana está dentro del Panel de Control y deja la dirección del Dispositivo disponible para otro que quiera conectarse en lo posterior.

1.6 LOS DESCRIPTORES

Los Descriptores constituyen una de las partes más importantes de un Dispositivo USB, y es aquí donde el programador debe trabajar para darle a su Dispositivo el comportamiento deseado. Son estructuras de datos, o un conjunto de bloques de información que le permiten al Host (PC) conocer acerca de un Dispositivo

Los Descriptores residen en el Firmware del Dispositivo.
Se puede tratar a los Descriptores como un conjunto de tablas de información, que conforme el Host lo solicite, se las harán llegar durante el proceso de Enumeración. La Enumeración es automática y transparente al usuario.

Dentro de estas tablas (Descriptores) se encuentra información relativa al Dispositivo, tal como: versión del USB al que pertenece, la Clase, número y tamaño de Endpoints, cantidad de energía que necesita (máximo 500 mA), números de identificación de fabricante y producto, texto de identificación, etc.

Cabe destacar que un Descriptor en particular puede tener uno o más Descriptores subordinados.

A cada uno de los elementos de un Descriptor se los conoce como campos, estos están expresados en notación hexadecimal y, un campo puede estar formado por 1, 2 o 3 bytes.

El primero y el segundo campo corresponden al tamaño y tipo del Descriptor respectivamente. En la Tabla 1-3 se observan los números asignados a cada tipo de Descriptor **[1].**

La mayoría de nombres de los campos usan prefijos que indican el formato o contenido de sus datos dentro de ese campo, así: *b* = byte (8 bits), *w* = palabra (16 bits), *bm* = mapa de bits, *bcd* = código binario decimal, *i* = índice, *id* = identificador. Por ejemplo *bMaxPacketSize,* como tiene el prefijo *b*, entonces este Descriptor es de 1 byte y contiene el máximo tamaño de paquete de datos.

bDescriptorType	Tipo de Descriptor	Requerido?
01h	Dispositivo	Si
02h	Configuración	Si
03h	Cadena	No, Texto descriptivo Opcional
04h	Interfase	Si
05h	Endpoint	No, si el Dispositivo sólo usa Endpoint 0
06h	Calificador de Dispositivo	Si, para Dispositivos que soportan ambas velocidades mediana y alta. No requerido para otros Dispositivos
07h	Configuración de Otra Velocidad	Si, para Dispositivos que soportan ambas velocidades mediana y alta. No requerido para otros Dispositivos
09h	OTG	Sólo para Dispositivos On-The-Go
0Ah	Depurador	No.
0Bh	Asociación de Interfase	Para Dispositivos compuestos.

TABLA 1-3: DESCRIPTORES STANDARD[1]

En la Figura 1-7 se muestra el formato del Descriptor bDescriptorType:

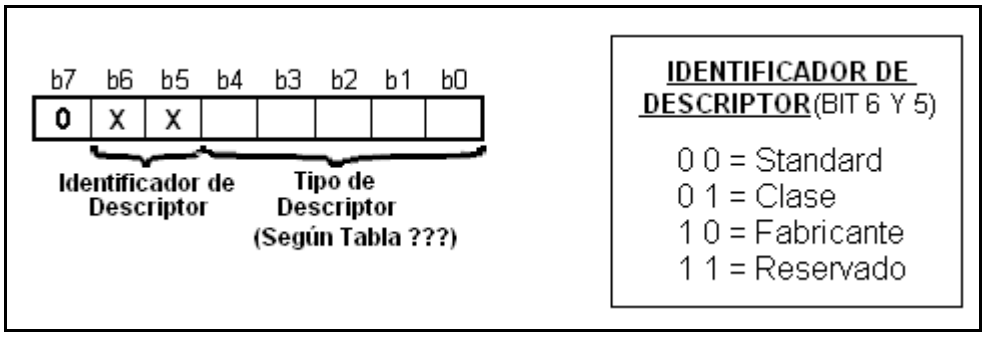


FIGURA 1-7: FORMATO DEL DESCRIPTOR bDescriptorType

El conjunto de Descriptores estándar, y la información que cada uno almacena se detalla a continuación.

1.6.1 TIPOS DE DESCRIPTORES

Todos los tipos de Descriptores que se revisan a continuación serán explicados haciendo referencia a los Descriptores pertenecientes al programador “EPNprog”.

1.6.1.1 Descriptor de Dispositivo

Este Descriptor contiene información básica acerca del Dispositivo, este es el primero de todo el conjunto de Descriptores que el Host solicita; consta de catorce campos y la información que cada uno de ellos lleva es:

Descriptor de Dispositivo del Programador “EPNprog”

```
0x12          ; bLength Longitud  de este Descriptor
0x01          ; bDescType Este es un Descriptor de Dispositivo
0x01,0x00     ; bcdUSBUSB version 1.00
0x00          ; bDeviceClasszero
0x00          ; bDeviceSubClass
0x00          ; bDeviceProtocol
0x08          ; bMaxPacketSize0 – tamaño de paquete
0xD8,0x04     ; idVendor - 0x04D8 Microchip es Vendor ID
0x00,0x00     ; idProduct
0x00,0x00     ; bcdDevice
0x02          ; iManufacturer
0x01          ; iProduct
0x00          ; iSerialNumber - 3
NUM_CONFIGURATIONS; bNumConfigurations
```

1er Campo (0x12), consta de 1 byte y da el número total de bytes que posee el Descriptor del Dispositivo.

2do Campo (0x01), consta de 1 byte y da el tipo de Descriptor y conforme a la Tabla 1-3, a éste le corresponde el número 0x01.

3er Campo (0x01, 0x00), consta de 2 bytes y nombra a la versión USB al que pertenece el Dispositivo, está en formato BCD y se estructura de la siguiente forma: el byte superior representa la parte entera, los siguientes cuatro bits son

las décimas y los últimos cuatro bits representan las centésimas, por ejemplo la versión 1,1 sería:

0x01, 0x10

4to Campo (0x00), es la clase a la que pertenece el Dispositivo, cuando las funciones del mismo son definidas en este nivel pero en el caso del “EPNprog” estas funciones no se definen aquí, por lo tanto el valor asignado es cero y se definirá la clase en el Descriptor de Interfaz.

5to y 6to Campo, son la subclase y el protocolo respectivamente, pero debido a que el cuarto campo es cero, estos también deben ser cero.

7mo Campo (0x08), da a conocer el máximo tamaño del paquete para el Endpoint 0, el Host usa esta información para las futuras comunicaciones.

8vo Campo (0xD8, 0x04), consta de 2 bytes y contiene el número de identificación que le ha asignado la corporación USB-IF a un fabricante. El número de identificación de la Microchip es 0x4D8.

9no Campo (0x00, 0x00), consta de 2 bytes y es la identificación del producto, este valor va a voluntad del fabricante.

10no Campo (0x00, 0x00), son 2 bytes en formato BCD que el vendedor asigna para ayudar al Host a encontrar un Driver.

11vo Campo (0x02), es un byte que indica la existencia de una cadena de caracteres (texto) en el Descriptor String, que nombra al fabricante, si el valor es cero significa que no existe texto.

12vo Campo (0x01), es un índice que apunta a un texto en el Descriptor String, que posee la descripción del Dispositivo, si el valor es cero no existe tal texto.

13vo Campo (0x00), es un índice que apunta a un texto del Descriptor String, y nombra al número serial del producto, es cero si no existe número serial, esta valor es útil cuando idénticos Dispositivos se conectan al puerto.

14vo Campo, número de configuraciones que el Dispositivo soporta.

1.6.1.2 Descriptor de Configuración

En este Descriptor residen las características del Dispositivo tal como: cantidad de energía que consume, número de interfases soportadas por el Dispositivo y otras que aquí se detallan:

Descriptor de Configuración del Programador "EPNprog"

```
0x09    ; bLength Longitud del Descriptor
0x02    ; Tipo de Descriptor= CONFIGURACIÓN
0x0C    : total de bytes Descriptor
0x01    ; bNumInterfaces Número de interfases
0x01    ; bConfigValue Valor de configuración
0x04    ; iConfigString Indice de esta configuración = #01
0x80    ; bmAttributesattributes – Energía del puerto
0x4B    ; MaxPowerself-powered
```

Este Descriptor consta de ocho campos:

1er Campo (0x09), Longitud del Descriptor.

2do Campo (0x02), consta de 1 byte y da el tipo de Descriptor según la Tabla 1-3.

3er Campo, suma total de bytes de este Descriptor y de sus Descriptores subordinados.

4to Campo (0x01), número de interfases en la comunicación, el mínimo valor debe ser 1.

5to Campo (0x01), identificador de configuración y debe ser uno o superior.

6to Campo (0x04), índice que apunta al texto en el Descriptor string, describe la configuración, un valor de cero indica que no hay texto.

7mo Campo (0x80), este nombra a través del bit 7 si el Dispositivo toma la energía del bus (1), o si tiene fuente independiente (0), esto es para la versión USB 1.0, en cambio para versiones superiores el bit de energía es el sexto.

La Figura 1-8 muestra el formato de este campo:

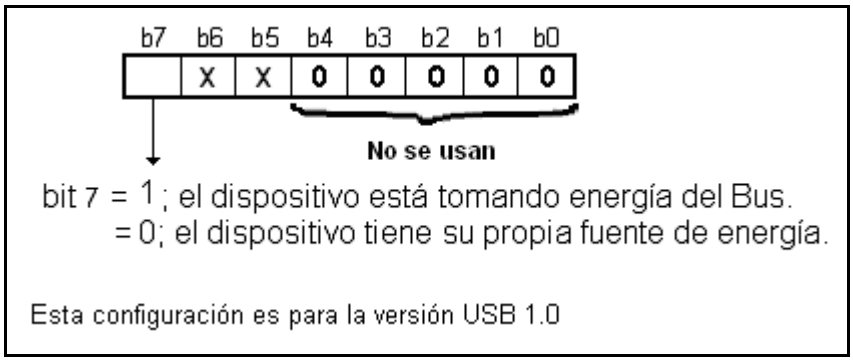


FIGURA 1-8: FORMATO DEL DESCRIPTOR bmAttributes

8vo Campo (0x4B), dice cuanta corriente (en miliamperios) el Dispositivo requiere. El valor colocado, es igual a la mitad de la corriente requerida, por ejemplo el programador EPNprog requiere 115 mA, se pone a un de valor 75 para que provea 150 mA (0x4B en hexadecimal).

La máxima cantidad de corriente que un Dispositivo puede solicitar teóricamente es 500mA, si se pone un número superior a 250 el Host se negará a configurar el Dispositivo.

1.6.1.3 Descriptor de Interfaz

Aquí se encuentra la información respecto a la clase a la que pertenece el Dispositivo (EPNprog pertenece a la clase HID), además del número de Endpoints que esta comunicación utiliza.

Descriptor de Interfaz del Programador "EPNprog"

```
0x09        ; Longitud del  Descriptor
0x04        ; Tipo de Descriptor
0x00        ; número de interfase
0x00        ; múltiples
0x02        ; número de Endpoints usados en esta interfase
0x03        ; Clase a la que pertenece el Dispositivo
0x00        ; Subclases
0x00        ; Protocolo de  interfaz
0x00        ; Cadena enlazada
```

1er Campo (0x09), Longitud del Descriptor.

2do Campo (0x04), consta de 1 byte y da el tipo de Descriptor según la Tabla 1-3.

3er Campo (0x00), identifica a la interfase, cuando el Dispositivo tiene múltiples interfases. El valor por defecto es cero.

4to Campo (0x00), usado para seleccionar una interfase, cuando el Dispositivo tiene múltiples interfases.

5to Campo (0x02), número de Endpoints soportados por la interfaz, a parte del Endpoint 0, para el EPNprog este valor es 0x02. (Endpoint 1 IN, Endpoint 1 OUT).

6to Campo (0x03), nombra a la clase a la cual pertenece el Dispositivo. Un valor igual a 0x03 corresponde a la clase HID para otras clases ver la Tabla 1-4.

7mo Campo (0x00), define una subclase dentro de la clase, si la clase es HID el valor de este campo debe ser cero **[1].**

8vo Campo (0x00), es similar al bDeviceProtocol en el Descriptor de Dispositivo, para una clase HID este valor debe ser cero **[1].**

9no Campo (0x00), texto asociado a la interfase.

Código Hexadecimal	Descripción
01	Audio
02	Interfaz de Comunicación
03	Dispositivo de Interfaz Humana (HID)
05	Física
06	Imagen
07	Impresora
08	Almacenamiento masivo
09	Hub
0A	Interfaz de Datos
0B	Tarjeta Inteligente
0D	Contenido de Seguridad
0E	Video

TABLA 1-4: DESCRIPCION DEL CAMPO bInterfaceClass[1]

1.6.1.4 Descriptor de Endpoint

Cada Endpoint especificado en el Descriptor de Interfase posee su propio Descriptor Endpoint. El Endpoint 0, no tiene un Descriptor ya que a cada Dispositivo debe soportar el Endpoint 0. En este Descriptor está la información relativa a: máximo tamaño para el Endpoint (especificación USB define los valores máximos), sentido del Endpoint (IN, OUT), tipo de Transferencia soportada por el Endpoint y la latencia máxima.

Descriptor del Endpoint1 del Programador “EPNprog”

```
0x07          ; Longitud del  Descriptor
0x05          ; Tipo de Descriptor
0x81          ; EP1, In
0x03          ; Tipo de Transferencia
0x08, 0x00    ; Tamaño máximo del Endpoint (8 bytes)
```

1er Campo (0x07), Longitud del Descriptor.

2do Campo (0x05), consta de 1 byte y da el tipo de Descriptor según la Tabla 1-4.

3er Campo (0x81), contiene el número del Endpoint y el sentido. El PIC16C745 por ser un Dispositivo de baja velocidad, puede tener máximo dos Endpoints además del Endpoint 0. Como se ve en la Figura 1-9, para que sea un Endpoint IN se debe poner el bit 7 a uno.

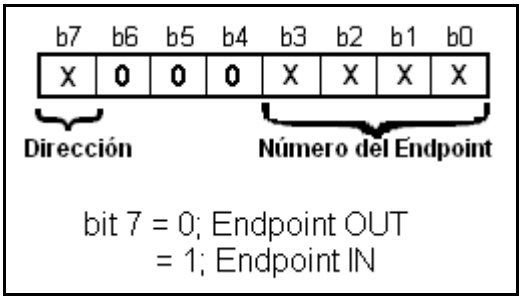


FIGURA 1-9: FORMATO DEL DESCRIPTOR bEndpointAddress

4to Campo (0x03), aquí se utilizan los dos bits menos significativos para escoger el tipo de Transferencia a utilizarse, el programador utiliza Transferencia de Interrupción.

```
00 = Control
01 = Asincrónica
10 = Masiva
11 = Interrupción
```

5to Campo (0x08, 0x00), son dos bytes que contiene el número máximo de datos que el Endpoint puede transferir en una transacción, para Dispositivos de baja velocidad el valor máximo es 8.

1.6.1.5 Descriptores de Cadena

Este Descriptor contiene texto descriptivo, son usados para proporcionar información específica del vendedor o una aplicación específica de información. Estos Descriptores pueden ser.

bString.- Este campo contiene una cadena de caracteres Unicode (Unicode usa 16 bits para representar un carácter). Al igual que los caracteres ANSI que codifican desde 00H a 7FH, Los caracteres Unicode codifican desde el 0000H al 007FH.

1.7 CONOCIENDO LA CLASE HID

Las clases agrupan a Dispositivos que comparten funciones o atributos comunes.

Entre las clases más conocidas se tienen: Audio, **HID,** Almacenamiento masivo de datos (memorias USB), Dispositivos de Comunicación (Modems y Networks), Impresoras, Video, etc.

La clase HID (Dispositivo de Interfaz Humana, por sus siglas en inglés), incluye pero no se limita a: teclados, ratones y controles de juegos.

El término “Interfaz Humana”, sugiere que estos Dispositivos interactúan con el usuario, mediante: botones e interruptores y con lo que respecta al Host (PC), éste debe actuar rápidamente a los cambios en las entradas del Dispositivo HID, para lograr que el usuario no note el retardo entre la acción y la respuesta esperada.

Pero un Dispositivo HID, no necesariamente tiene que poseer una interfaz humana, basta con que este calce dentro de los límites que dice la especificación de clase HID. Algunos de los límites se los detalla a continuación:

- Un reporte no puede ser mayor a 255 bytes.
- En un Dispositivo de baja velocidad el tamaño máximo para un paquete (un Endpoint), es de 8 bytes, para un Dispositivo mediana velocidad el tamaño máximo para un paquete (un Endpoint), es de 64 bytes y para un Dispositivo de alta velocidad el tamaño es de 1024 bytes.

- El Host (PC) obtiene los datos mediante el polling al Dispositivo HID.
- En un Dispositivo de mediana velocidad, la tasa de censado (polling) máxima es de una transacción cada 1 milisegundo, esto es:

$$(\#\ bytes / s)_{máx} = (1\ transacción / ms) * 1000$$
$$1\ transacción \rightarrow (64\ bytes)_{máx}$$
$$(\#\ bytes / s)_{máx} = 64000\ bytes / s$$

- En un Dispositivo de baja velocidad, la tasa de censado máxima es de una transacción cada 10 milisegundos esto es:

$$(\#\ bytes / s)_{máx} = (1\ transacción / 10\ ms) * 100$$
$$1\ transacción \rightarrow (8\ bytes)_{máx}$$
$$(\#\ bytes / s)_{máx} = 800\ bytes / s$$

- Un Dispositivo HID puede intercambiar cualquier tipo de datos, pero tan solo puede utilizar tanto Transferencias de Control e Interrupción.

Lo que hace llamativo el uso de la clase HID es que Windows y otros sistemas operativos poseen Drivers por defecto para la clase HID.

Cuando se está dentro de la clase HID se debe intercambiar los datos en estructuras llamadas reportes.

1.7.1 REPORTES

Es una estructura de datos, un reporte puede constar de un buffer básico de bytes o un complejo surtido de ítems.

En cuanto a los Endpoints que utiliza un Dispositivo HID, debe tener un Endpoint IN para enviar datos al Host, el uso de un Endpoint OUT es opcional, como se ve en los Descriptores del EPNprog se hace uso de ambos. Esto debido a que los Endpoints de Interrupción intercambian los datos de una forma rápida o periódica, mientras que las Transferencias de Control pueden tardarse si el puerto está muy ocupado.

El firmware debe incluir:

- Descriptor de Interfase, que especifica la clase HID (0x03)
- Descriptor HID
- Descriptor de Endpoint IN (Interrupción)
- Descriptor de Reporte

La formación presente en estos Descriptores se detalla con relación al EPNprog:

1.7.2 DESCRIPTOR HID DEL PROGRAMADOR "EPNprog"

```
0x09          ; Longitud del Descriptor
0x21          ; Tipo de Descriptor (HID)
0x01, 0x00    ; Versión de la Clase HID (1.00)
0x00          ; País de origen
0x01          ; # de Descriptores subordinados  (1)
0x22          ; Tipo del Descriptor subordinado (HID)
0x00, 0x1D    ; Longitud del Descriptor subordinado
```

1er Campo (0x09), Longitud del Descriptor.

2do Campo (0x21), byte que identifica al Descriptor como HID.

3er Campo (0x01, 0x00), son dos byte que nombran la versión a la cual pertenece la clase HID.

4to Campo (0x00), número que da al hardware la identificación específica a un país.

5to Campo(0x01), número de Descriptores subordinados a éste. Se refiere al Descriptor de Reporte que se revisa más adelante.

6to Campo (0x22), tipo de Descriptor subordinado, y por ser un Descriptor de Reporte el valor 0x22.

7mo Campo (0x00, 0x1D), tamaño del Descriptor del Reporte.

1.7.3 DESCRIPTOR DE REPORTE

Aquí se da el tamaño de los reportes que llevan los datos, además se los nombra como reportes de entrada o salida.

El Descriptor de reporte define el formato y uso de los datos que están dentro de los reportes HID.

Cada item en el Descriptor de reporte consiste de: un byte que identifica al campo y uno o más bytes de datos.

Descriptor de Reporte del Programador "EPNprog"

```
0x06h, 0x00h, 0xffh     ; USAGE_PAGE (Vendor Defined Page 1)
0x09h, 0x01h            ; USAGE (Vendor Usage 1)
0xA1h, 0x01h            ; COLLECTION (Application)
0x19h, 0x01h            ; USAGE_MINIMUM (Vendor Usage 1)
0x29h, 0x08h            ; USAGE_MAXIMUM (Vendor Usage 8)
0x15h, 0x00h            ; LOGICAL_MINIMUM (0)
0x26h, 0xffh, 0x00h     ; LOGICAL_MAXIMUM (255)
0x75h, 0x08h            ; REPORT_SIZE (8)
0x95h, 0x08h            ; REPORT_COUNT (8)
0x81h, 0x02h            ; INPUT (Data,Var,Abs)
0x19h, 0x01h            ; USAGE_MINIMUM (Vendor Usage 1)
0x29h, 0x08h            ; USAGE_MAXIMUM (Vendor Usage 8)
0x91h, 0x02h            ; OUTPUT (Data,Var,Abs)
0xC0h                   ; END_COLLECTION LOGICAL_MINIMUM
```

1er Campo, especifica la función para la que fue diseñado el Dispositivo, el valor 0x06 es tomado como un Dispositivo genérico.

2do Campo, especifica las funciones de un reporte.

3er Campo, item que encabeza a un grupo de items que juntos trabajan para ejecutar una única función.

1.7.3.1 Elementos de la Colección

Campo LOGICAL_MINIMUM y LOGICAL_MAXIMUM , especifican el rango de valores que el reporte puede contener.

Campo REPORT_COUNT, en este campo se pone la longitud máxima de un reporte que se intercambiará entre Host y Dispositivo, y el valor máximo es 255 para la clase HID.

Campo REPORT_SIZE, en este campo se encuentra el número de bits que le pertenece a cada uno de los elementos del reporte. En el programador EPNprog cada elemento del reporte es de 8 bits.

Campo INPUT (Data,Var,Abs), el valor del byte identificador de campo 0x81h nombra al reporte como de entrada.

Campo OUTPUT (Data,Var,Abs), el valor del byte identificador de campo 0x91h nombra al reporte como de salida.

CAPÍTULO 2

ESPECIFICACIONES DE PROGRAMACIÓN PARA LAS FAMILIAS DE MICROCONTROLADORES PIC (16F87X, PIC16F87XA, PIC16CXX, PIC16FXX) Y MICROCONTROLADORES ATMEL (AT89CXX Y AT89SXX)

Todos los microcontroladores, para su normal funcionamiento disponen de espacios de memoria, donde residen: código de programa, datos, palabras de configuración, información del fabricante, etc.

Todos estos espacios de memoria deben ser programados, siguiendo las especificaciones técnicas que pone a disposición el respectivo fabricante.

En este capítulo se presenta:

- Los diferentes espacios de memoria que poseen los microcontroladores PIC y ATMEL
- Descripción de los pines involucrados
- Secuencia de operaciones para culminar con éxito la programación

2.1 ESPECIFICACIONES PARA LOS MICROCONTROLADORES PIC.

2.1.1 MAPA DE MEMORIA DE LOS MICROCONTROLADORES PIC [2]

Los microcontroladores PIC con CPU de 8 bits, dependiendo de la familia a la que pertenezcan, poseerán Memoria de Datos no volátil (EEPROM) a más de la Memoria de Programa. Se debe señalar que dentro de la Memoria de Programa existe la localidad perteneciente a la Palabra de Configuración y Localidades de Identificación (ID).

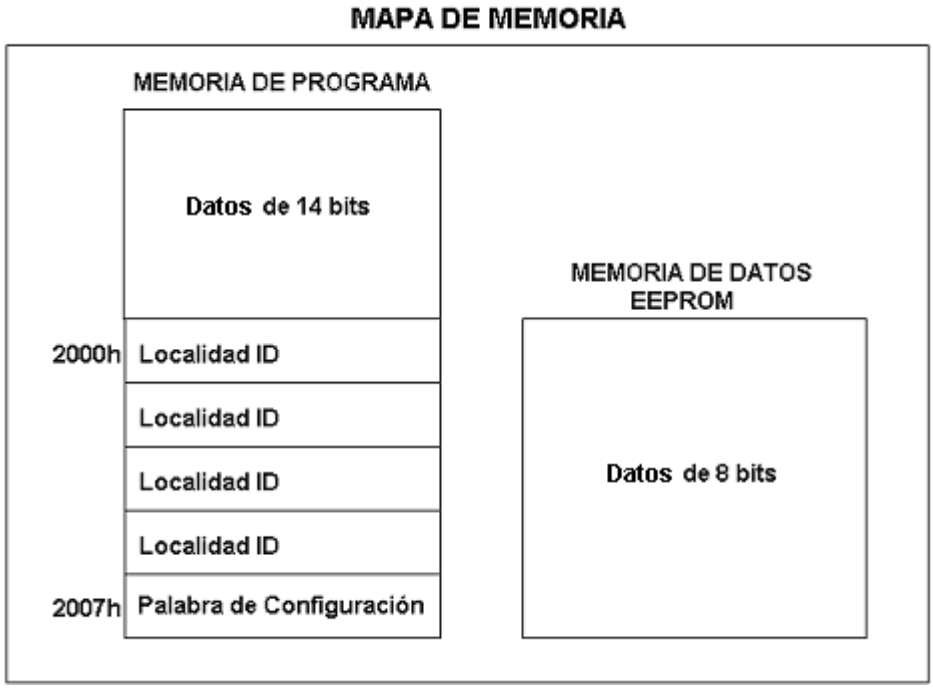


FIGURA 2-1: MAPA DE MEMORIA DE LOS MICROCONTROLADORES PIC

2.1.1.1 Memoria de Programa

La memoria de instrucciones, mejor conocida como Memoria de Programa, es el bloque donde el usuario guarda el conjunto de instrucciones que ejecutará el microcontrolador PIC en normal funcionamiento. Dependiendo de la familia a la cual pertenezca el microcontrolador PIC, la memoria va a variar tanto en tipo como en tamaño, por ejemplo, la nomenclatura de la familia 16x8x diferencia a sus modelos, por la letra intermedia (*C, F o CR*), teniendo el siguiente significado:

C : posee memoria de programa tipo EPROM
F : posee memoria de programa tipo Flash
CR: posee memoria de programa tipo ROM y se graba en fábrica.

Otra característica en cuanto tiene que ver a este espacio, es que el tamaño de cada dato de la memoria varía en función a la gama a la que pertenece el microcontrolador PIC, así se tienen las gamas: enana, baja, media y alta, con tamaños de 12, 12, 14 y 16bits, respectivamente.

2.1.1.1.1 Palabra de Configuración

Estos bits están asignados dentro de la Memoria de Programa en la localidad 2007H, que sólo se puede acceder durante la programación **[2].**
En esta localidad de memoria se pueden configurar características especiales que harán trabajar al microcontrolador de diferentes modos. Estas características son:

- Selección del Oscilador
  Tipos: RC, HS, XT, LP
- Habilitación del Reset por caída de voltaje
- Habilitación del Watch Dog Timer
- Protección del código de la Memoria de Programa o de la Memoria de Datos EEPROM
- Habilitación de la programación Serial en Bajo Voltaje

2.1.1.1.2 Localidades ID **[2].**

Cuatro localidades de Memoria de Programa (2000H a 2003H) son destinadas como Localidades ID, que el usuario puede utilizarlas para almacenar información de identificación tal como el Checksum, Número de Serie del Producto u otro tipo de identificación. Estas localidades no son accesibles durante la ejecución normal, pero se las puede leer y escribir durante la Programación/Verificación del microcontrolador. Microchip recomienda usar sólo los cuatro bits menos significativos de cada una de las cuatro Localidades ID.

2.1.1.2 Memoria de Datos

Existen microcontroladores que disponen a más de la Memoria de Programa, de una Memoria de Datos no volátil que se puede leer y escribir, ésta es de tipo EEPROM. De esta forma, un corte en la energía de alimentación no ocasionará la pérdida de la información, y podrá estar disponible al reiniciarse el programa. A la Memoria de Datos no volátil EEPROM se la puede leer y escribir dentro del proceso de programación.

El tamaño de este espacio de memoria varía con el dispositivo. (Ver Tabla 2-1)

Los programas que se ejecutan dentro del microcontrolador PIC necesitan datos que varían continuamente, y que no importa que se pierdan al momento de apagar o reinicializar el microcontrolador. Para este efecto cada dispositivo implementa una cantidad de memoria RAM estática (SRAM), que es volátil.

Dispositivo	Memoria de Programa		Memoria de Datos SRAM (Bytes)	Memoria de Datos EEPROM (Bytes)
	Bytes	# de Datos		
PIC16F84A		1024	68	64
PIC16F873/874	7.2K	4096	192	128
PIC16F876/877	14.3K	8192	368	256
PIC16F873A/874A	7.2K	4096	192	128
PIC16F876A/877A	14.3K	8192	368	256
PIC16C745/765	14.3K	8192	256	No implementada

TABLA 2-1: TAMAÑO DE CADA MEMORIA [2]

2.1.2 REQUERIMIENTOS PARA LA PROGRAMACIÓN DE LOS MICROCONTROLADORES PIC

Las familias de microcontroladores PIC usan el método de programación serial, por lo que se destinan tres pines para la programación, además de los pines de polarización.

La función de cada uno de los pines involucrados en la programación se detalla en la Tabla 2-2.

Nombre del Pin	Durante la Programación		
	Función	Tipo de Pin	Descripción del Pin
RB6	CLOCK	I	Entrada de Reloj
RB7	DATA	I/O	Entrada/Salida de Datos
MCLR	$V_{TEST\ MODE}$	P	Selector del Modo Programar
VDD	VDD	P	Alimentación de Energía
VSS	VSS	P	Referencia de la fuente

I = Entrada, O = Salida, P = Energía

TABLA 2-2: DESCRIPCIÓN DE PINES UTILIZADOS POR EL EPNprog [2]

- **Pin Vpp**, se usa para poner al PIC en estado de programación, esto se lo consigue variando el nivel de voltaje desde 0V a 13V.
- **Pin DATA**, es bidireccional y se usa para introducir los bits que conforman el dato. Por este pin también se hace la introducción de comandos.
- **Pin CLOCK**, es utilizado para ingresar los pulsos de reloj.

Existen dos métodos para la programación: en modo alto voltaje (13V±0.5V) y en modo de bajo voltaje (5 V).

En la Figura 2-2 se ha tomado como ejemplo el PIC16F877A, para visualizar la asignación de pines necesarios durante el proceso de programación.

Para otros microcontroladores PIC la posición de estos pines cambia de acuerdo a la familia a la cual pertenezca dicho microcontrolador.

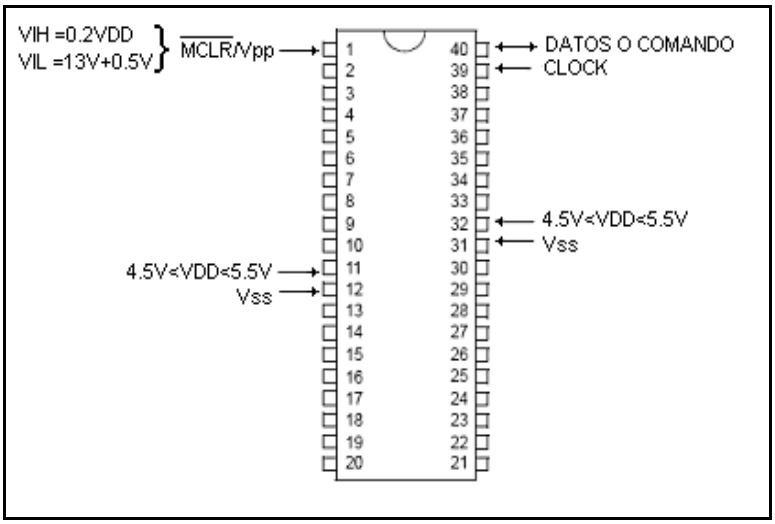


FIGURA 2-2: DISTRIBUCIÓN DE PINES PARA LA PROGRAMACIÓN DEL PIC16F877A

Para la programación de los microcontroladores PIC, en este proyecto de titulación, se trabaja en forma serial y en modo de alto voltaje, para lo que se cambia el voltaje en el pin Vpp desde el nivel VIL a VIHH, donde VIL = 0.2VDD y VIH = 13V±0.5V.

2.1.2.1 Forma de Operación de un Comando y de un Dato

Un comando o un dato está formado por bits que van sincronizados con los pulsos de reloj, por lo tanto el número de ciclos de reloj dependerá del tamaño del comando o dato. (Ver Figura 2-4).

El tamaño del comando y del dato varía en función de la familia a la cual pertenezca el microcontrolador PIC. Para los microcontroladores PIC soportados por este programador, el tamaño correspondiente a un comando es de 6 bits y el de un dato es de 14bits. Cabe destacar que cuando un dato es transmitido, está acompañado de 2 bits, uno de INICIO y otro de PARADA, por lo que se necesitarán 16 ciclos de reloj durante el proceso de transmisión. (Ver Figura 2-3).

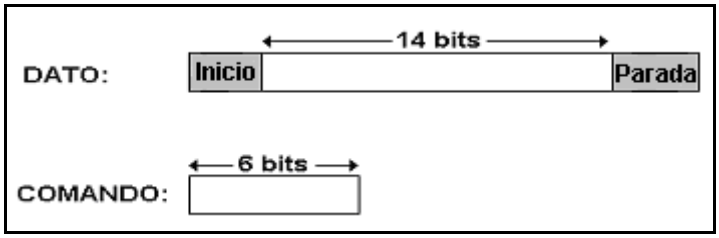


FIGURA 2-3: ESTRUCTURA DE UN DATO DE UN COMANDO

Un comando va siempre antes de un dato u otro comando. Este comando avisa al microcontrolador el objetivo del dato que prosigue a dicho comando. En la Tabla 2-3 se puede ver la lista de comandos que le corresponden a la familia PIC16F87X. El resto de tablas de comandos para los otros tipos de familias soportadas por este programador se encuentran en la sección del Anexo B.

Comando	Asignación de bits (MSB…LSB)						Dato
Load Configution (set PC=2000H)	X	X	0	0	0	0	0, datos(14),0
Load Data for Program Memory	X	X	0	0	1	0	0, datos(14),0
Read Data from Program Memory	X	X	0	1	0	0	0, datos(14),0
Increment Address	X	X	0	1	1	0	
Begin Erase/Programming Cycle	X	0	1	0	0	0	
Begin Programming Only Cycle	X	1	1	0	0	0	
Load Data for Data Memory	X	X	0	0	1	1	0, datos(14),0
Read Data from Data Memory	X	X	0	1	0	1	0, datos(14),0
Bulk Erase Program Memory	X	0	1	0	0	1	
Bulk Erase Data Memory	X	0	1	0	1	1	
Chip Erase	X	1	1	1	1	1	
End Programming	X	1	0	1	1	1	

TABLA 2-3: ASIGNACIÓN DE COMANDOS PARA PIC 16F87X [2]

Al ingresar un comando sincronizado con el reloj, cada bit del comando es almacenado en el flanco de bajada del reloj, siendo el bit menos significativo del comando el que se transmite primero.

Se requiere de un intervalo de tiempo de separación entre un comando y dato, o entre comandos consecutivos, esto es necesario para que el dispositivo que se está programando configure a su pin. Los datos también entran y salen empezando por el bit menos significativo LSB y son transmitidos en el flanco de subida, y almacenados en el flanco de bajada del reloj.

En la Figura 2-4 se aprecia el diagrama de tiempo que corresponde al Comando **LOAD DATA** (cargar dato), seguido por su respectivo dato. (Familia: PIC16F87XA).

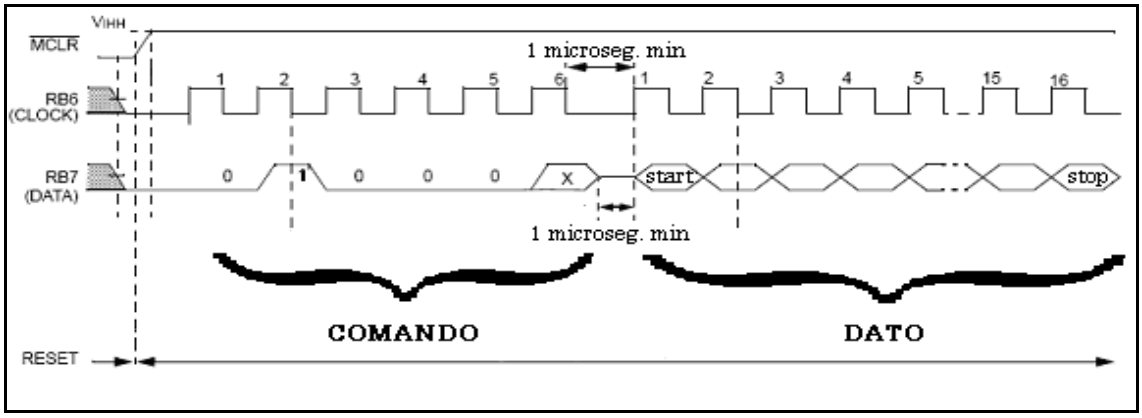


FIGURA 2-4: DIAGRAMA DE TIEMPO DE COMANDO "LOAD DATA" Y DE UN DATO [2]

2.1.2.2 Secuencia para la Programación

Para culminar exitosamente la programación se siguen los pasos que indica Microchip en sus hojas de "Especificaciones de Programación".

Debido a la similitud en la programación de los microcontroladores PIC que abarca este proyecto se ha decidido explicar tal proceso, tomando como ejemplo a la familia de microcontroladores PIC 16F87XA.

2.1.2.2.1 Secuencia de Escritura

Los espacios de memoria de datos y de programa, deben ser borrados antes de intentar programarlos.

Esta familia de microcontroladores PIC utiliza la siguiente secuencia para programar ocho datos en la Memoria de Programa **[2].**

1. Cargar un dato en la localidad de memoria direccionada usando el comando '**Load Data**'.
2. Emita un comando '**Increment Address**'.
3. Cargar un dato en la localidad de memoria direccionada usando el comando '**Load Data**'.
4. Repita el paso 2 y paso 3 seis veces.

5. Emita un comando '**Begin Erase/Programming Cycle**'.
6. Espere un tprog (aproximadamente 1[ms])
7. Emita un comando '**End Programming**'.
8. Emita un comando '**Increment Address**'.
9. Repita esta secuencia hasta completar todas las localidades de memoria.

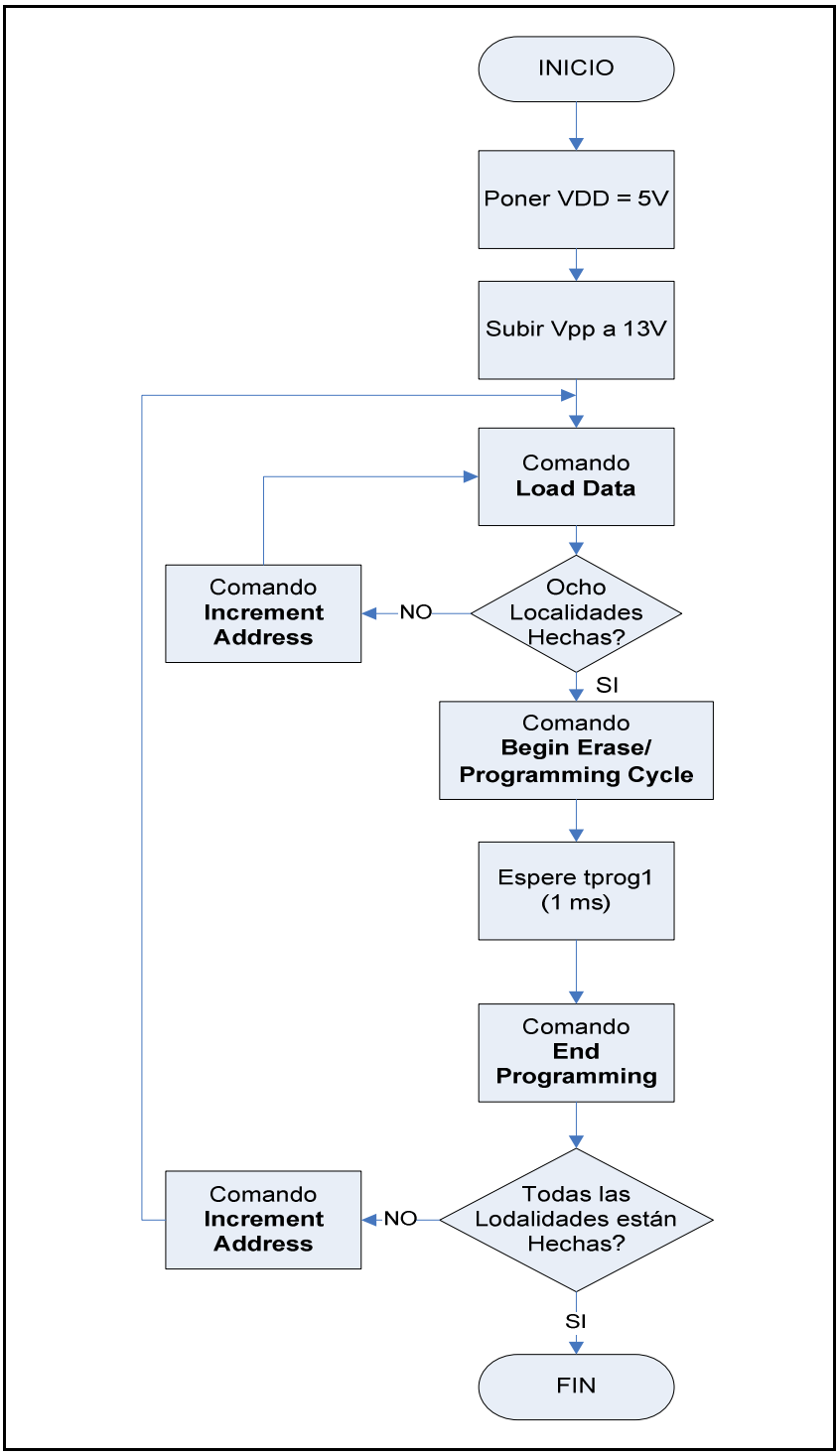


FIGURA 2-5: FLUJOGRAMA PARA PROGRAMAR LA MEMORIA DE PROGRAMA. [2]

2.1.2.2.1.1 Escritura de la Palabra de Configuración y Localidades ID.

Para el proceso de escritura y lectura de las Localidades ID, y de la Palabra de Configuración, se realiza el siguiente procedimiento:

Se comienza cargando datos en las Localidades ID:

1. Ubicarse en la dirección 2000h de la memoria de programa usando el comando '**Load ConFiguration**'.
   Luego de emitir el comando '**Load ConFiguration**', se debe cargar el dato "3F8Fh", que es un número que recomienda la Microchip para la correcta programación de las localidades que se grabarán a continuación.
2. Cargar un dato en la localidad de memoria direccionada usando el comando '**Load Data**'
3. Emita un comando '**Begin Programming Only Cycle**'.
4. Espere un tprog1 (aproximadamente 1[ms])
5. Emita un comando '**End Programming**'.
6. Emita un comando '**Increment Address**'.
7. Repetir los pasos 2 a 6, 3 veces más.

NOTA: Las palabras ID deben escribirse siguiendo el formato que indica el fabricante, esto es: "11 1111 1000 bbbb", donde 'bbbb' es la información ID del usuario

Ahora se va escribir la Palabra de Configuración:

8. Emita un comando '**Increment Address**'.
9. Repita este comando 2 veces (así se ubicará en la posición 2007H).
10. Cargar la Palabra de Configuración en la localidad de memoria direccionada usando el comando '**Load Data**'
11. Emita un comando '**Begin Erase/Programming Cycle'**.
12. Espere un tprog2 (aproximadamente 8[ms]).

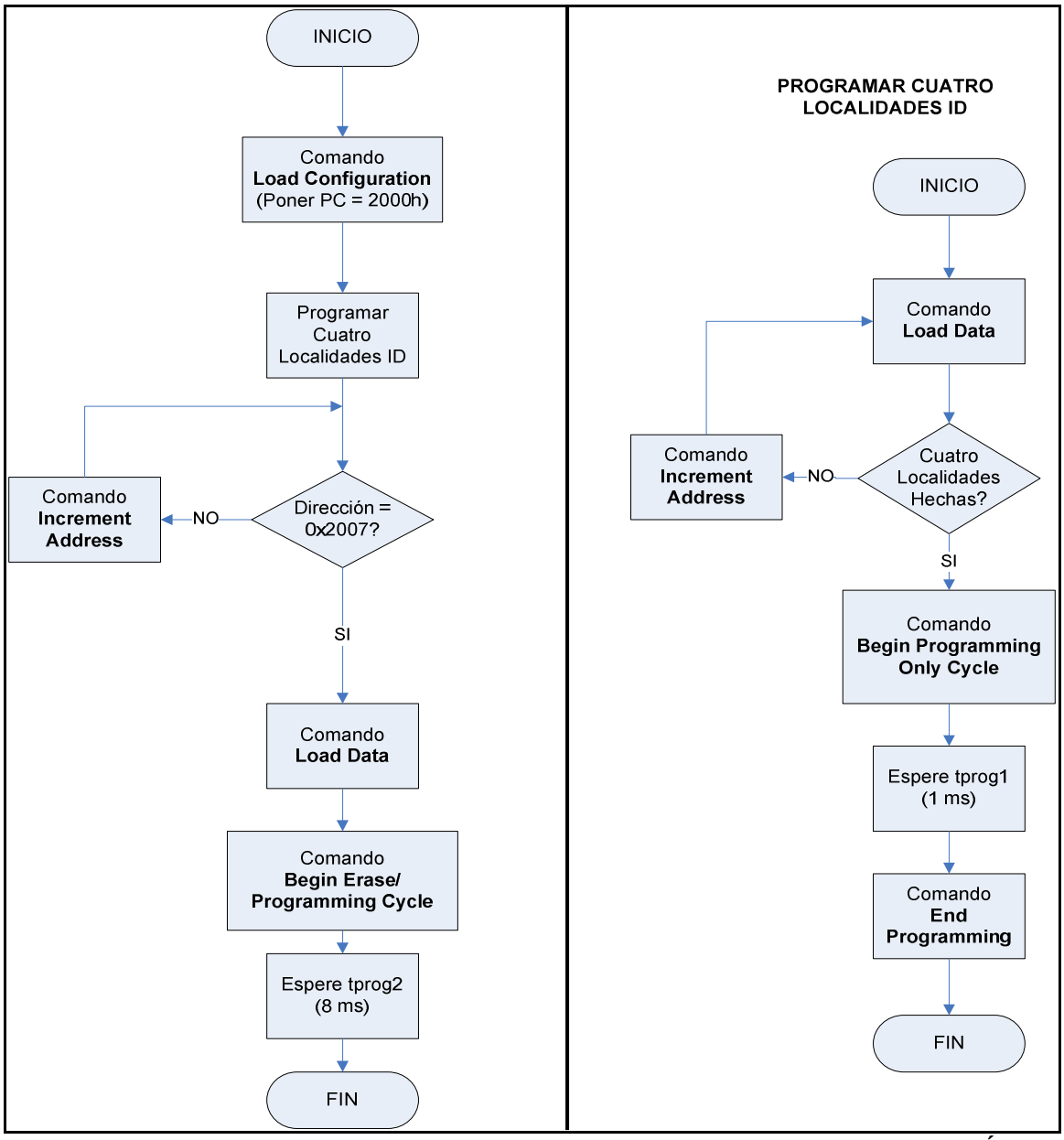


FIGURA 2-6: FLUJOGRAMA PARA PROGRAMAR LA PALABRA DE CONFIGURACIÓN Y LOCALIDADES ID [2].

2.1.2.2.2 Secuencia de Lectura

Para la lectura de la totalidad de la memoria de programa se realiza la siguiente secuencia:

1. Leer un dato de la localidad de memoria direccionada usando el comando '**Read Data**'.
2. Emita un comando '**Increment Address**'.
3. Repita esta secuencia hasta completar todas las localidades de memoria.

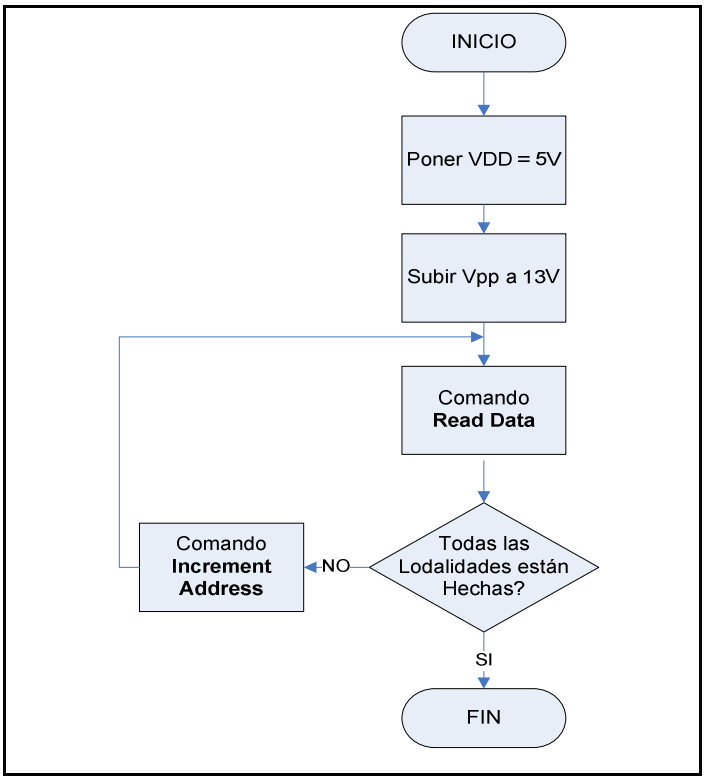


FIGURA 2-7: FLUJOGRAMA PARA LEER LA MEMORIA DE PROGRAMA [2]

2.1.2.2.3 Secuencia de Borrado

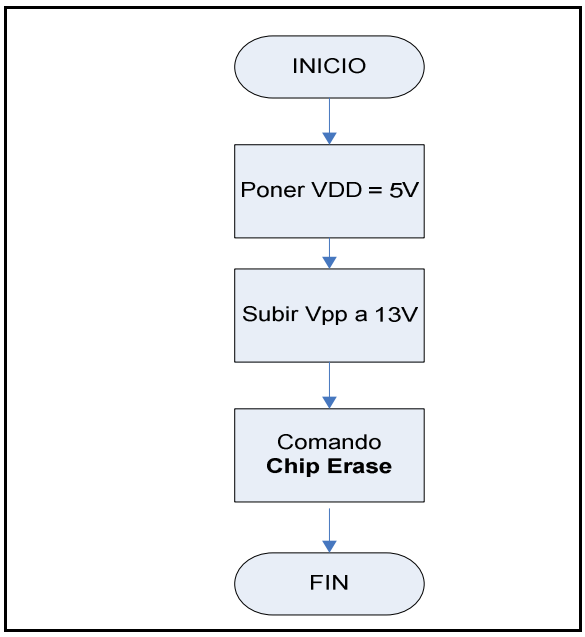


FIGURA 2-8: FLUJOGRAMA PARA BORRAR LA MEMORIA DE PROGRAMA [2]

Los flujogramas presentados sirven tanto para la memoria de programa, como para la de datos no volátil EEPROM. Teniendo en cuenta que se debe poner el comando apropiado de acuerdo al espacio de memoria que se desea programar.

2.2 ESPECIFICACIONES PARA MICROCONTROLADORES ATMEL

2.2.1 MAPA DE MEMORIA DE LOS MICROCONTROLADORES ATMEL AT89CXX Y AT89SXX

Los microcontroladores ATMEL que soporta este programador poseen internamente un espacio dedicado para memoria de programa.

2.2.1.1 Memoria de Programa

Este espacio de memoria es de tipo Flash y es aquí donde el usuario almacena el código de programa.

Debido a que este tipo de dispositivos pueden expandir su memoria de programa por la adición de una memoria externa, hasta 64 Kbytes de memoria entre la interna y externa; el programador de este proyecto de titulación programa únicamente la memoria interna que corresponda al dispositivo.

La familia de microcontroladores ATMEL AT89X51 posee 4 Kbytes de memoria de programa interna, para el resto de familias se muestra en la Tabla 2-4.

Cada dato de este bloque de memoria tiene una extensión de un byte.

2.2.1.2 Memoria de Datos

Este espacio de memoria es de tipo RAM, es decir contiene datos volátiles.

Cada microcontrolador ATMEL admite hasta 64 Kbytes de memoria externa y a su vez contiene 256 bytes de RAM interna, distribuidos de la siguiente forma: 128 bytes de propósito general, y los restantes 128 bytes de Registros de Función Especial, (esto en los AT89X51, para los otros microcontroladores ATMEL ver Tabla 2-4). Los 128 bytes de propósito general pueden ser accedidos en forma directa (MOV dato dirección) o indirecta (MOV @Ri), mientras que los SFRs son registros propios del procesador y pueden ser accedidos sólo de forma directa **[3].**

Dispositivo	Memoria Flash de Programa Interna (Bytes)	Memoria de Datos RAM Interna de Propósito General. (Bytes)
AT89C51/S51	4 K	128
AT89C52/S52	8 K	256
AT89C55/S55	20K	256

TABLA 2-4: TAMAÑO DE MEMORIA PARA VARIOS MICROCONTROLADORES ATMEL

2.2.1.3 Bits de Bloqueo de la Memoria de Programa

El chip tiene tres bits de bloqueo los cuales pueden ser manipulados y puestos a: programado (P) o no-programado (U), y que sólo pueden ser modificados en el modo programación, obteniendo así características adicionales que se especifican en la Tabla 2-5

Bits de Bloqueo de Programa				Tipo de Protección
	LB1	LB2	LB3	
1	U	U	U	Ninguna característica de bloqueo del programa
2	P	U	U	Instrucciones MOVC ejecutadas desde la memoria externa de programa son deshabilitadas desde los bytes de código extraídos desde la memoria interna y además la programación de la Flash es deshabilitada
3	P	P	U	Igual que el modo 2, además la verificación es deshabilitada
4	P	P	P	Igual que el modo 3, además la ejecución externa es deshabilitada.

TABLA 2-5: MODOS DE PROTECCIÓN DE LOS BITS DE BLOQUEO [3]

2.2.1.4 Bytes Descriptores

Los bytes descriptores son localidades especiales donde reside información acerca del dispositivo que se va a manipular, un ejemplo de estos valores que corresponden a la familia AT89C51 se muestran a continuación:

(030H) = 1EH indica fabricado por ATMEL
(031H) = 51H indica 89C51
(032H) = FFH indica programación a 12V
(032H) = 05H indican programación a 5V **[3]**

2.2.2 REQUERIMIENTOS PARA LA PROGRAMACIÓN DE LOS MICROCONTROLADORES ATMEL AT89CXX Y AT89SXX

Las familias ATMEL usan un método de programación paralela. Aunque las familias AT89SXX también dan la posibilidad de programación serial.

Para poder programar las familias ATMEL, su Memoria Flash de Programa debe estar en el estado de borrado (es decir, conteniendo = FFH). La interfaz de programación para las familias AT89CXX acepta dos modos de programación: una en alto voltaje (12 V) y la otra en bajo voltaje (VCC). Las familias AT89SXX sólo admiten una interfaz de programación en alto voltaje (12 V) **[3].**

Se utilizará el método de programación paralela y en modo alto voltaje que consiste en la elevación de voltaje en el pin EA desde VIH, a Vpp, donde VIH = 5V y Vpp = 12V.

Además de manipular los pines de EA/Vpp, Vcc (5v) y GND, se tienen otros pines que utiliza el microcontrolador ATMEL para ser programado y se muestran en la Tabla 2-6.

Nombre del Pin y del Puerto	**Durante la Programación**		
	Función	**Tipo de Pin**	**Descripción del Pin**
EA	$V_{TEST\ MODE}$	P	Selector del Modo Programar
VCC	VCC	P	Alimentación de Energía
GND	GND	P	Referencia de la fuente
Port 1 y P2.0 – P2.3	DIRECCION	I	Entrada de Dirección
Port 0	DATOS	I/O	Entrada/Salida de Datos
ALE	ALE	I	Entrada del Pulso de Programación
XTAL1		I	Entrada de oscilación
XTAL2		O	Salida de oscilación
RESET		I	Entrada de Reset

I = Entrada, O = Salida, P = Energía

TABLA 2-6: DESCRIPCIÓN DE PINES UTILIZADOS POR EL EPNprog PARA LOS MICROCONTROLADORES ATMEL [3]

Las señales de control determinan la operación que realiza el microcontrolador ATMEL.(Vease Tabla 2-7). En la Figura 2-9 se pueden observar los pines en los que se deben ubicar todas las señales de control.

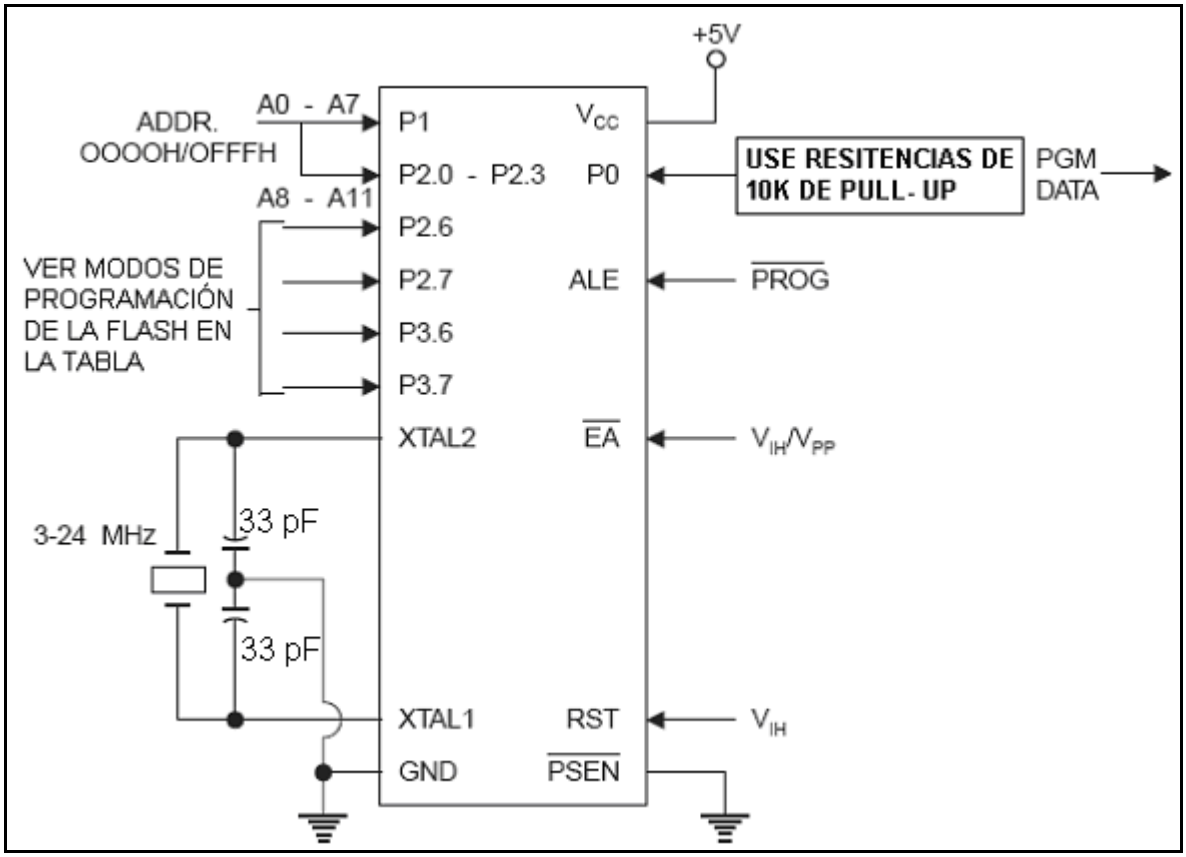


FIGURA 2-9: DISTRIBUCIÓN DE PINES PARA LA PROGRAMACIÓN DE LOS MICROCONTROLADORES ATMEL [3]

Modo		RST	PSEN	ALE/ PROG	EA/ Vpp	P2.6	P2.7	P3.3	P3.6	P3.7	P0.7-0 Datos	P2.3-0 Dirección	P1.7-0 Dirección
Código de Escritura de Datos		H	L	(2)	12V	L	H	H	H	H	D_{IN}	A11-8	A7-0
Código de Lectura de Datos		H	L	H	H	L	L	L	H	H	D_{OUT}	A11-8	A7-0
Escribir Bloqueo	Bit - 1	H	L	(2)	12V	H	H	H	H	H	X	X	X
	Bit - 2	H	L	(2)	12V	H	H	H	L	L	X	X	X
	Bit - 3	H	L	(2)	12V	H	L	H	H	L	X	X	X
Leer Bits de Bloqueo 1, 2 y 3		H	L	H	H	H	H	L	H	L	P0.2, P0.3, P0.4	X	X
Borrar Chip		H	L	(1)	12V	H	L	H	L	L	X	X	X
Leer ID del ATMEL		H	L	H	H	L	L	L	L	L	1EH	0000	00H
Leer ID del Dispositivo		H	L	H	H	L	L	L	L	L	51H	0001	00H
Leer ID del Dispositivo		H	L	H	H	L	L	L	L	L	06H	0010	00H

TABLA 2-7: SEÑALES DE CONTROL PARA LA PROGRAMACIÓN DE LA MEMORIA DE PROGRAMA [3]

Nota:

1 Para Borrar el Chip cada pulso PROG está entre 200 ns -500 ns para las familias AT89SXX y está en 10 ms aproximadamente para las familias AT89CXX

2 Cada pulso PROG está entre 200 ns -500 ns para las familias AT89SXX y está entre 1 µs - 110 µs para las familias AT89CXX.

3 X = no importa

4 En las familias ATMEL AT89CXX no se utiliza la señal de control en el pin P3.3.

2.2.2.1 Secuencia de Lectura/Escritura de los microcontroladores ATMEL

Antes de programar, leer o borrar los microcontroladores ATMEL; las líneas correspondientes a dirección, datos y señales de control deben conectarse según la Figura 2-7.

Posteriormente aplique la siguiente secuencia:

1. Active la combinación correcta de señales de control según la Tabla 2-7.
2. Espere aproximadamente 3 µs.(con cristal de 16 MHz).
3. Aumente EA/VPP a 12V (Para la lectura no se necesita de este paso).
4. Ingrese la dirección de memoria deseada en las líneas de dirección.
5. Ingrese o extraiga los bytes de datos apropiados en las líneas de datos.
6. Espere aproximadamente 3 µs.(con cristal de 16 MHz).
7. Dar un pulso ALE/PROG una vez para programar un byte en la Flash o los bits de bloqueo. (Para la lectura no se necesita de este paso)
8. Repita los pasos del 1 al 5, incrementando la dirección y cambiando los datos hasta que se alcanza el fin del archivo. (Para borrar los microcontroladores ATMEL no se necesitan implementar los pasos 3, 4 y 6) **[3].**

Para la lectura de datos se usan resistencias de pull up de 10K.

2.2.2.3 Borrando el Dispositivo

La Memoria Flash se borra eléctricamente usando la combinación apropiada de señales de control y manteniendo ALE/PROG en bajo (aproximadamente 10 ms para las familias AT89CXX y 300 ns para AT89SXX). La operación de borrado debe ejecutarse antes de que el código de memoria pueda re programarse.

Las características eléctricas y de tiempo se presentan en el Anexo C.

CAPÍTULO 3
DISEÑO DEL HARDWARE NECESARIO PARA LA PROGRAMACIÓN.

El presente capítulo tiene como propósito el diseño del hardware para la programación de los microcontroladores PIC y ATMEL, partiendo del diagrama de bloques.

Se explica el diseño de cada elemento constitutivo de una fuente conmutada elevadora de voltaje.

Se expone, también, el diseño de los circuitos de control que cumplen las siguientes funciones:

- Manejar los niveles de voltaje en el pin Vpp.
- Proteger los pines del microcontrolador USB de posibles daños por parte del dispositivo a ser programado.

3.1 DIAGRAMA DE BLOQUES DEL PROGRAMADOR

En la Figura 3-1, se presenta el diagrama de bloques del programador, en el cual se ven claramente las diferentes partes que constituyen dicho programador. Estas partes se detallan a continuación:

1.- Para la comunicación entre el computador y el programador se utilizará la interfaz USB, la cual además de proveer y receptar datos, servirá de fuente de alimentación para los diferentes elementos del programador (5 V).

2.- Se diseñará una fuente elevadora de voltaje, que elevará el voltaje nominal de 5V que provee USB a 17V.

3.- Regulador de voltaje, el cual tomará el voltaje de salida de la fuente elevadora y lo mantendrá en 12V (para Vpp).

4.- El microcontrolador USB (PIC 16C745), comanda todas las actividades de cada elemento dentro del programador:

5.- El microcontrolador USB se ayuda de circuitos de control para realizar las siguientes actividades:

- Manejo de niveles de voltaje (Vpp) en los diferentes microcontroladores.
- Protección del microcontrolador USB, de posibles daños por parte del dispositivo a ser programado.

6.- Los leds, indican de una forma visual que el programador está encendido y que el flujo de datos y reloj está ocurriendo para las diferentes actividades que realiza el programador.

7.- Los contadores, proveen las direcciones de los espacios de memoria en los cuales se van a ubicar los distintos datos en la programación de los microcontroladores ATMEL.

8.- Los zócalos sirven de soporte para conectar a los diferentes dispositivos a programar.

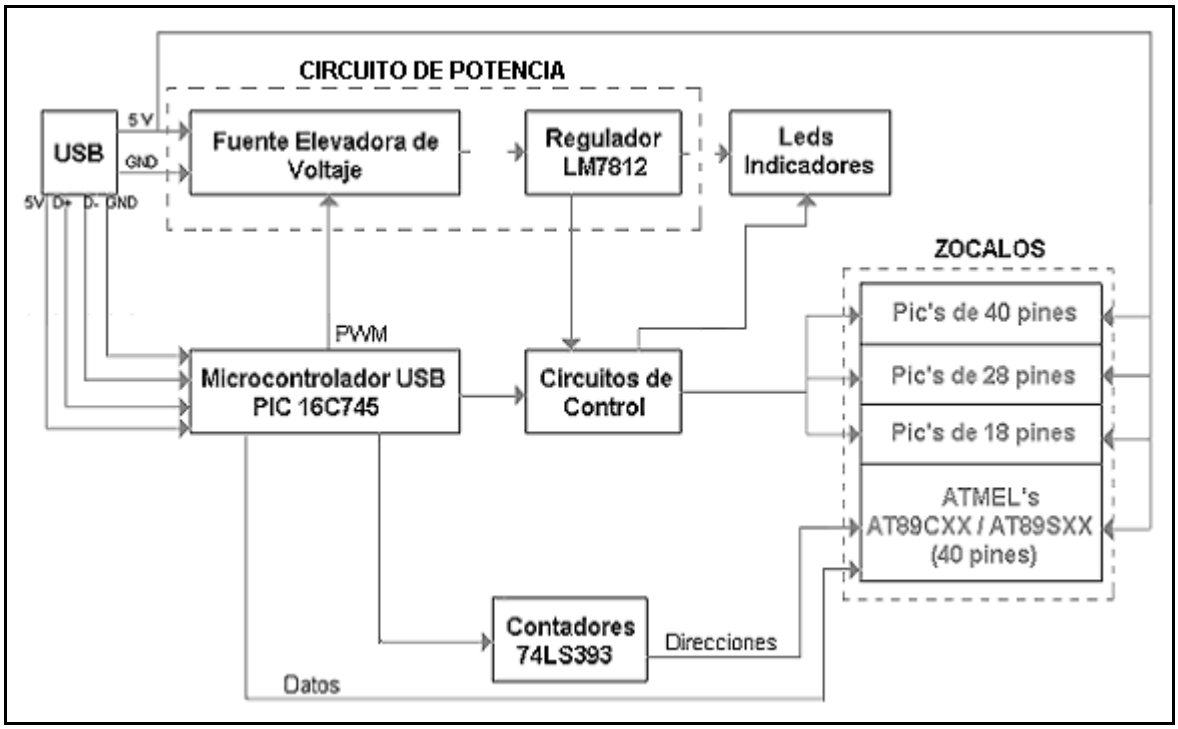


FIGURA 3-1: DIAGRAMA DE BLOQUES DEL PROGRAMADOR

3.2 DISEÑO DEL CIRCUITO DE POTENCIA.

El programador presentado en este proyecto de titulación obtiene toda su energía del puerto USB para alimentar todos sus elementos, esta decisión se debió al siguiente análisis:

- La interfaz USB posee un suministro de energía cuyo voltaje nominal es 5V y 500 mA de corriente máxima **[1]** (es decir provee una potencia máxima P =V I = 5V * 0.5A = 2.5 W).
- Diseño y prueba del circuito en el que la corriente máxima que consume es de 115 mA (Ver Tabla 3-1). Dando así una potencia de entrada de 0.575 Watts.
- El uso de una fuente externa al programador, encarecería su costo y ocuparía demasiado espacio dentro del programador, ya que éstas son voluminosas.

ELEMENTO	CANTIDAD	CONSUMO DE CORRIENTE MÁXIMA POR UNIDAD	CONSUMO TOTAL
PIC 16C745	1	15 mA	15 mA
PIC y ATMEL a Programarse	1	50 mA	50 mA
Regulador 7812	1	8 mA	8 mA
LEDs	3	4 mA	12 mA
Contador 393	2	15 mA	30 mA
TOTAL			**115 mA**

TABLA 3-1: CONSUMO DE CORRIENTE DE LOS DIFERENTES ELEMENTOS DEL PROGRAMADOR EPNprog (TOMADAS DEL ANEXO D).

3.2.1 ELEVADOR DE VOLTAJE

Ya que las diferentes familias de microcontroladores PIC y ATMEL, necesitan de un voltaje Vpp (13 V) para entrar en modo de programación en alto voltaje, se requiere del diseño de un conversor que permita elevar el voltaje nominal del USB (5V) hasta el respectivo voltaje Vpp que requiere cada dispositivo. Por este motivo se diseñó una fuente conmutada elevadora de voltaje (Conversor DC-DC Elevador).

3.2.1.1 Funcionamiento de la fuente conmutada elevadora de voltaje.

Las fuentes de conmutación son dispositivos que actualmente se utilizan en muchas aplicaciones de la industria electrónica, gracias a su capacidad de entregar niveles en su salida de voltaje bastante estables, así como una eficiencia mucho mayor que las fuentes de alimentación lineales. Este tipo de fuentes se encuentran implementadas en áreas de computación, telefonía, control, biomédica, etc.

En la Figura 3-2 se muestra el esquema de una fuente conmutada elevadora utilizando un transistor MOSFET canal N.(BS170)

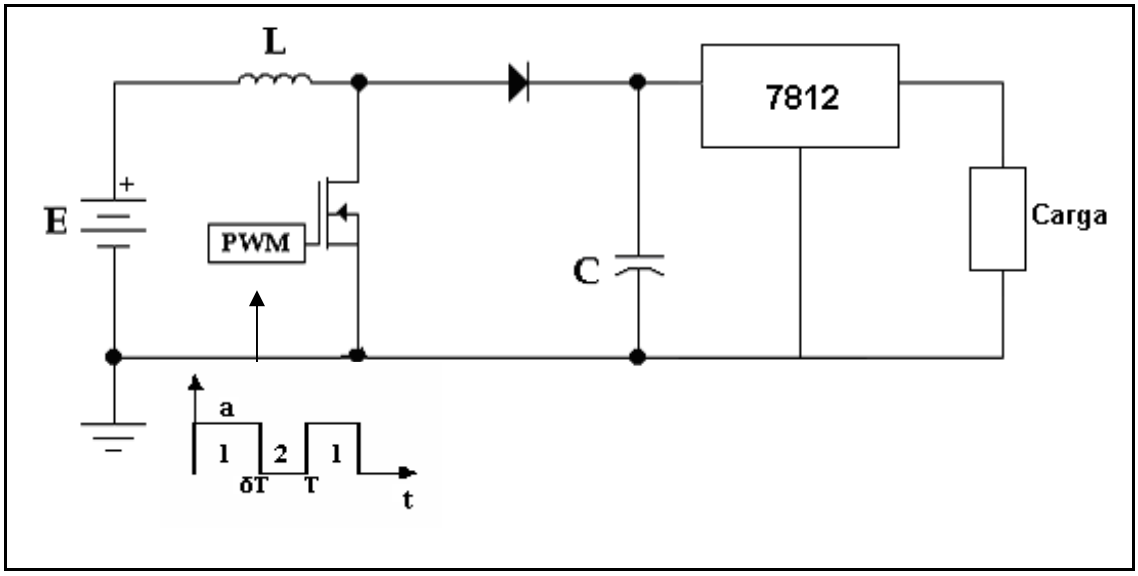


FIGURA 3-2: CONVERSOR DC – DC ELEVADOR

En la Figura 3-3 se pueden observar la señal de salida en el condensador (Vc), voltaje sobre la inductancia (VL), y las corrientes en cada elemento. El circuito básico del conversor DC-DC, convierte la señal de voltaje continuo de la fuente (E) en otra señal de voltaje continuo pulsatorio, luego se pasa por un filtro C que se encarga de dar una señal de voltaje continua proporcional a la relación de trabajo (δ). A continuación se conecta la carga, que es la que absorbe la energía proporcionada por la fuente.

Para facilitar el diseño, se coloca un regulador 7812, entre la carga y el conversor elevador, que mantendrá constante el voltaje para Vpp.

Este tipo de circuito debe necesariamente estar conectado a una carga. De no ser así, la energía que se almacena en el condensador en forma de voltaje se elevaría demasiado, dañando así al elemento.

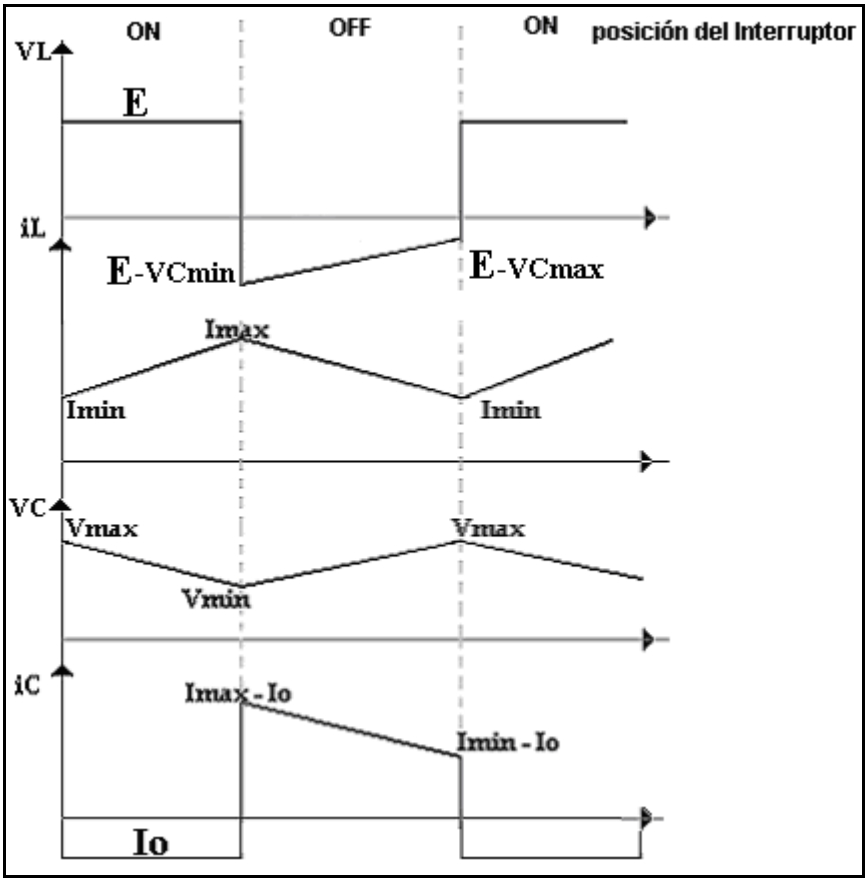


FIGURA 3-3: FORMAS DE ONDA DEL CONVERSOR DC/DC ELEVADOR

En este tipo de conversores el voltaje de salida es mayor que el voltaje de entrada, de allí que su nombre es “conversor elevador”. La operación se la puede dividir en dos partes: En la primera parte se considera el transistor en estado de saturación (cerrado), la corriente de entrada, fluye a través del inductor L y del transistor MOSFET, elevándose en magnitud; en la segunda parte se considera el transistor en estado de corte (abierto), y la corriente que almacenó la inductancia obliga a conducir al diodo, fluyendo así la corriente hacia el condensador, y también hacia la carga. La corriente en el inductor disminuye, y así hasta que se activa el siguiente ciclo.

3.2.1.2 Diseño del Conversor DC/DC Elevador

De las formas de onda de la Figura 3-3, el voltaje medio en una inductancia es cero, entonces:

$$V_{L_{DC}} = \frac{1}{T}\left[\int_0^{\delta T} E dt + \int_0^{(1-\delta)T} (E - Vc) dt\right] = 0$$

$$\frac{1}{T}\left[E(\delta T) + (E - Vc)(1-\delta)T\right] = 0$$

$$E - Vc(1-\delta) = 0 \Rightarrow Vc = \frac{E}{(1-\delta)}$$

Se puede controlar el voltaje de salida variando la relación de trabajo, de acuerdo a la siguiente relación:

$$Vo = \frac{E}{1-\delta} \qquad \textbf{(Ec. 3.1)}$$

En donde $0 \le \delta \le \delta \max$
$E \le Vo \le V \max$

Entonces:

$$\delta = 1 - \frac{E}{Vo} \qquad \textbf{(Ec. 3.2)}$$

Para que el regulador opere con normalidad, necesita un voltaje de entrada en el rango comprendido entre 14.5V y 27V **[5]**. Por lo tanto se escoge un valor de 17 V y según la ecuación 3.2 se obtiene un δ = 0.7.

Para el diseño del conversor fue necesario seguir los siguientes pasos:

1.- Seleccionar la frecuencia de operación, dentro del rango PWM que maneja el PIC16C745 (f=200 KHz). Se escoge un valor de frecuencia alto, para que el valor de inductancia sea pequeño.

2.- Calcular los componentes que requiere el conversor como son la bobina y el capacitor de salida.

Para el cálculo de las componentes del conversor se deducen las siguientes ecuaciones:

- Para encontrar la inductancia L,

Cuando el transistor está encendido:

$$E = V_L + V_{drain-source}$$

$$\Rightarrow V_L \approx E \qquad \textbf{(Ec. 3.3)}$$

y como:

$$V_L = L\left(di/dt\right) \qquad \textbf{(Ec. 3.4)}$$

reemplazando la ecuación 3.4 en 3.3

$$E = L\left(di/dt\right)$$

$$di/dt = E/L$$

como la variación de corriente con respecto al tiempo es una constante, se tiene :

$$\Delta I = \frac{E}{L}\Delta t \qquad \textbf{(Ec. 3.5)}$$

con referencia a la Figura 3-3 se tiene:

$$\Delta I = I\max - I\min$$

entonces reemplazando en la ecuación 3.5 se tiene

$$I\max - I\min = \frac{E}{L}\Delta t$$

$$L = \frac{E}{I\max - I\min}\Delta t$$

Para el intervalo $\Delta t = a = \delta T = \delta (1/f)$

$$L = \frac{E}{I\max - I\min}\left(\delta\frac{1}{f}\right) \qquad \textbf{(Ec. 3.6)}$$

Imin = 0 (por trabajar en conducción discontinua)

Imáx=Io=50mA (corriente máxima que requiere el microcontrolador al momento de programación).

Reemplazando E = 5V, f = 200 KHz y δ = 0.7, en la ecuación 3-6, se tiene:

$$L = 350\mu H$$

- Para encontrar el valor del condensador C

De acuerdo a la Figura 3-3, cuando el transistor está encendido:

$$Ic = -Io \qquad \textbf{(Ec. 3.7)}$$

y como:

$$Ic = C\frac{dVc}{dt} \qquad \textbf{(Ec. 3.8)}$$

$$dVc = \frac{Ic}{C}dt \qquad \textbf{(Ec. 3.9)}$$

como la variación de voltaje con respecto al tiempo es una constante, se tiene:

$$\Delta Vc = -\frac{Io}{C}\Delta t$$

$$\therefore C = -\frac{Io}{\Delta Vc}\Delta t \qquad \textbf{(Ec. 3.10)}$$

asumiendo que se desea un $\left|\Delta Vc\right|$ máximo de 0.2V, para un consumo de corriente en la carga de 50 mA y para un intervalo de tiempo de a = δ T = δ (1/f), entonces se obtiene la siguiente ecuación:

$$C = -\frac{Io}{\Delta Vc}\left(\frac{\delta}{f}\right) \qquad \textbf{(Ec. 3.11)}$$

reemplazando todos los valores en la ecuación 3.11

$$C = 0.87\mu F$$

entonces se aproxima a un valor de:

$$C = 1\mu F$$

3.2.1.2.1 Resultados Reales

En las siguientes figuras se muestran las señales obtenidas en el circuito real:

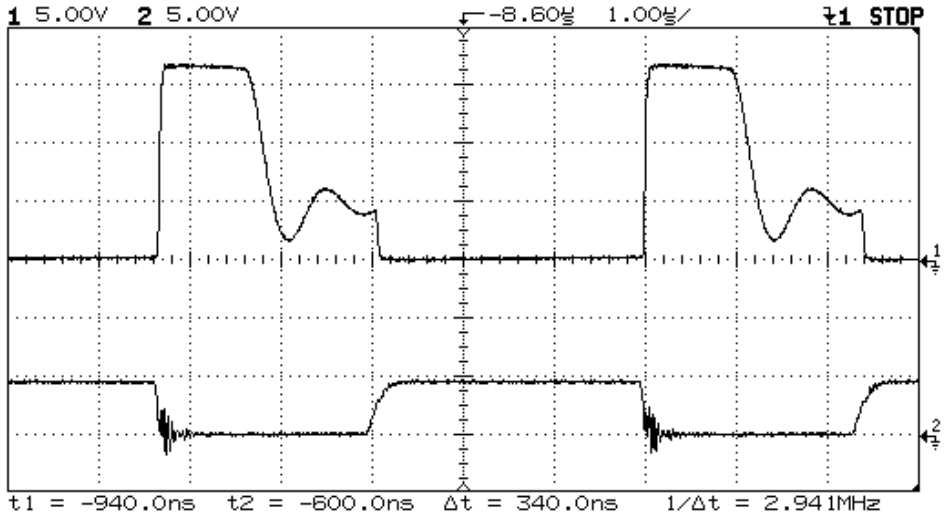


FIGURA 3-4 : VOLTAJE EN EL DRENAJE DEL MOSFET , Y SEÑAL PWM A LA ENTRADA DE LA COMPUERTA DEL MOSFET.

Se tiene un periodo de 5 µs aproximadamente y un tiempo en alto de 3.4 µs lo que da una relación de trabajo:

$$\delta = \frac{\text{tiempo en alto}}{\text{periodo}} = \frac{3.4\mu s}{5\mu s} = 0.68 \qquad \textbf{(Ec. 3.12)}$$

Se puede observar también que al activar el transistor (pulso en alto en la base), el voltaje en el drenaje del MOSFET se hace cero y cuando se desactiva el transistor (pulso en bajo), el voltaje del drenaje toma un valor de 17V aproximadamente, que es el voltaje en el capacitor del circuito. Se ve claramente que no todo el tiempo se mantienen estos 17V, ya que se está trabajando en conducción discontinua, mostrada en la siguiente Figura.

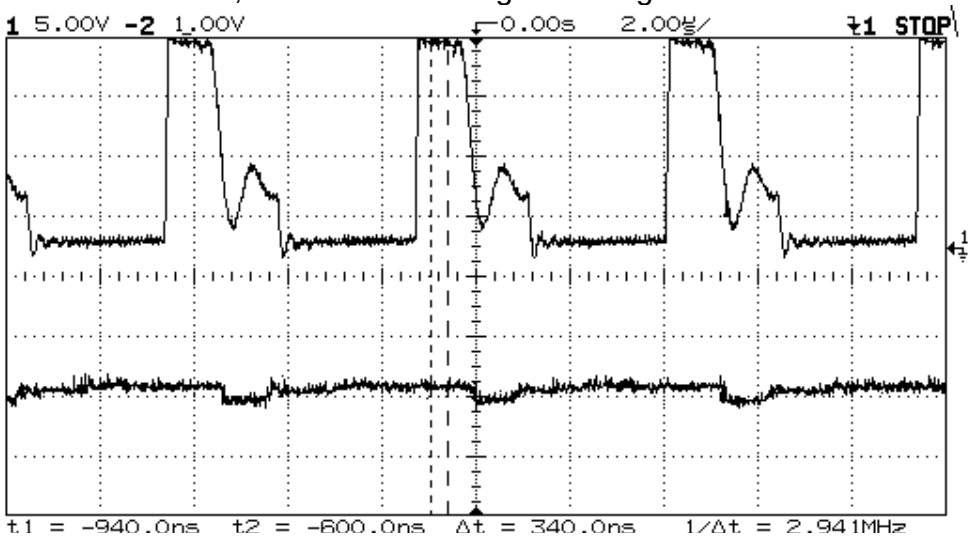


FIGURA 3-5: VOLTAJE EN EL DRENAJE DEL MOSFET (1) Y CORRIENTE QUE CIRCULA A TRAVÉS DE LA BOBINA.

La corriente en el inductor llega a un punto en que se hace cero, debido a que se está trabajando en conducción discontinua y a este tiempo el voltaje en el drenaje del MOSFET cae de los 17V a 5V, ya que se deja de conducir corriente por el diodo, entonces este se abre y en realidad se está midiendo el voltaje de la fuente (5V).

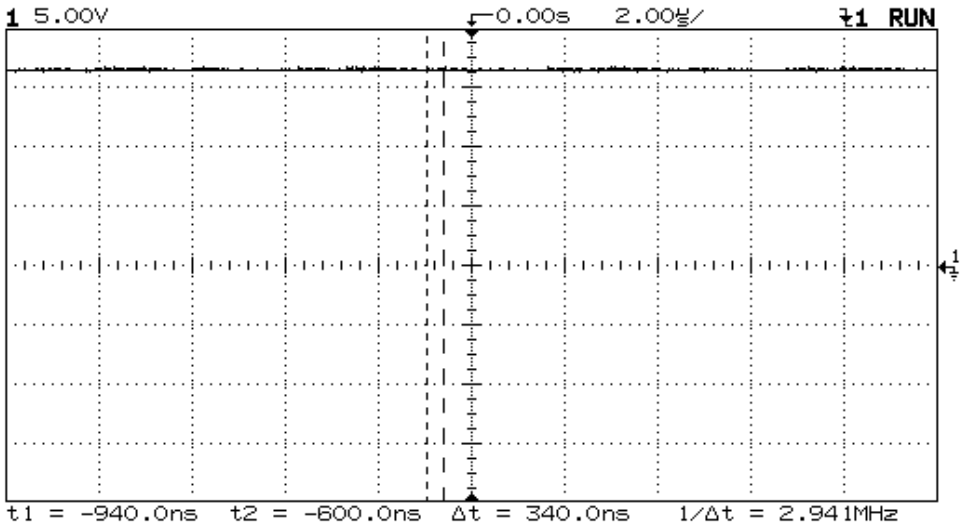


FIGURA 3-6: VOLTAJE EN EL CAPACITOR (A LA ENTRADA DEL REGULADOR 7812)

En la Figura anterior se muestra el voltaje obtenido a la salida del conversor elevador. Se observa que el voltaje se mantiene constante y que el voltaje se eleva de los 5V a 17V aproximadamente, que era lo esperado para un δ de 0.7.

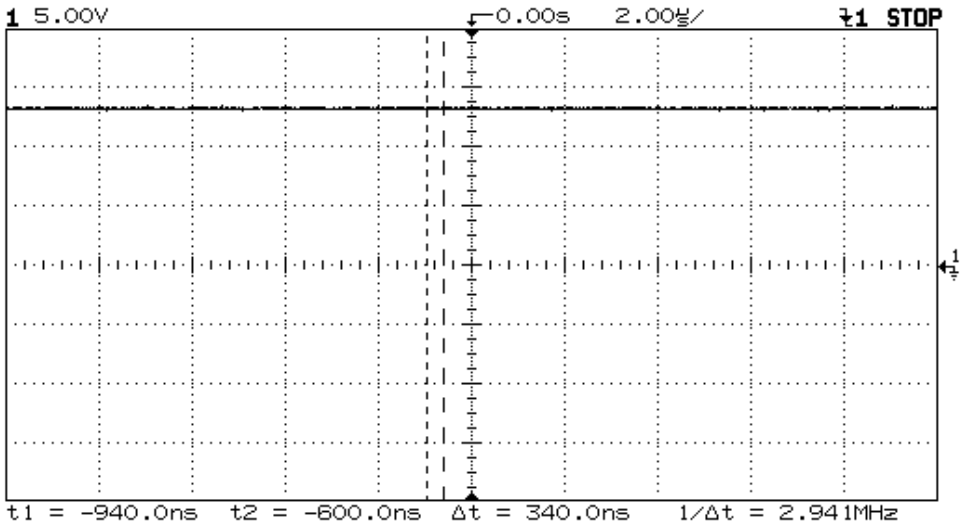


FIGURA 3-7: VOLTAJE A LA SALIDA DEL REGULADOR 7812.

Finalmente se tiene el voltaje a la salida del regulador que se mantiene constante en 12 V.

3.3 DISEÑO DE LOS CIRCUITOS DE CONTROL

3.3.1 PARA MICROCONTROLADORES PIC

Debido a que el programador soporta diversos microcontroladores, donde los pines involucrados en la programación varían físicamente en la ubicación, es necesario encaminar los niveles lógicos de voltaje a los pines correspondientes.

3.3.1.1 Pines para programación de los microcontroladores PIC.

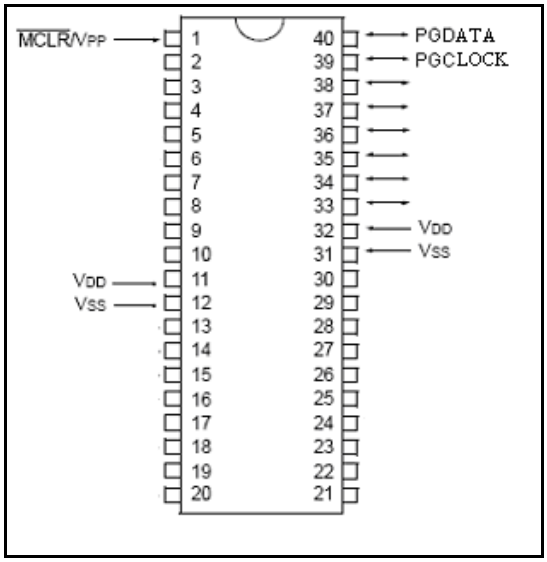


FIGURA 3-8: DIAGRAMA DE PINES DEL: PIC16F877/874/871/877A/874A PIC16C65/67/74/77

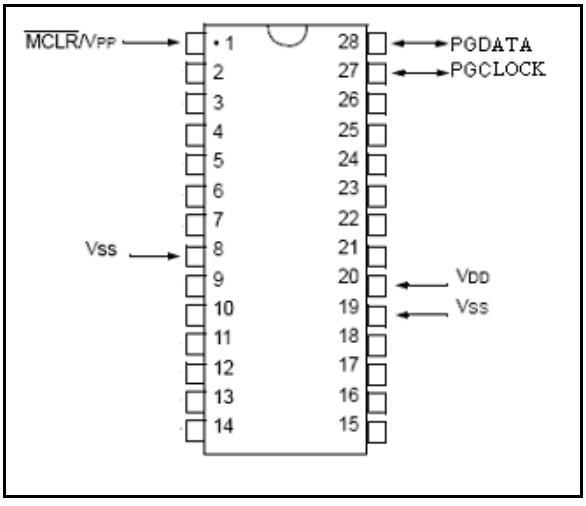


FIGURA 3-9: DIAGRAMA DE PINES DEL: PIC16F876/873/872/870/876A/873A PIC16C62/63/66/72/73/76

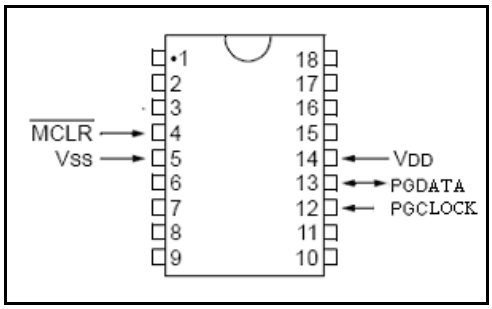


FIGURA 3-10: DIAGRAMA DE PINES DE: PIC16F83/84/84A

Sabiendo que cada pin del PIC 16C745 drena una corriente de hasta 20 mA **[4]**, se lo utiliza directamente sobre la base del transistor, sin la necesidad de amplificadores de corriente de por medio. En la Tabla 3-2 se muestra la asignación de pines del PIC 16C745 con los pines del dispositivo a ser programado.

PIC 16C745	PINES DEL PIC
RC0	Vpp
RC1	CLOCK
RA3	LEER DATO
RA5	ESCRIBIR DATO
RA4	VDD

TABLA 3-2: ASIGNACIÓN DE PINES DEL PIC 16C745 PARA PROGRAMAR PICs

3.3.1.2 Circuito para Vpp

El circuito que maneje este pin necesita cambiar el nivel 0V a 13V, por lo que el diseño del circuito se detalla a continuación.

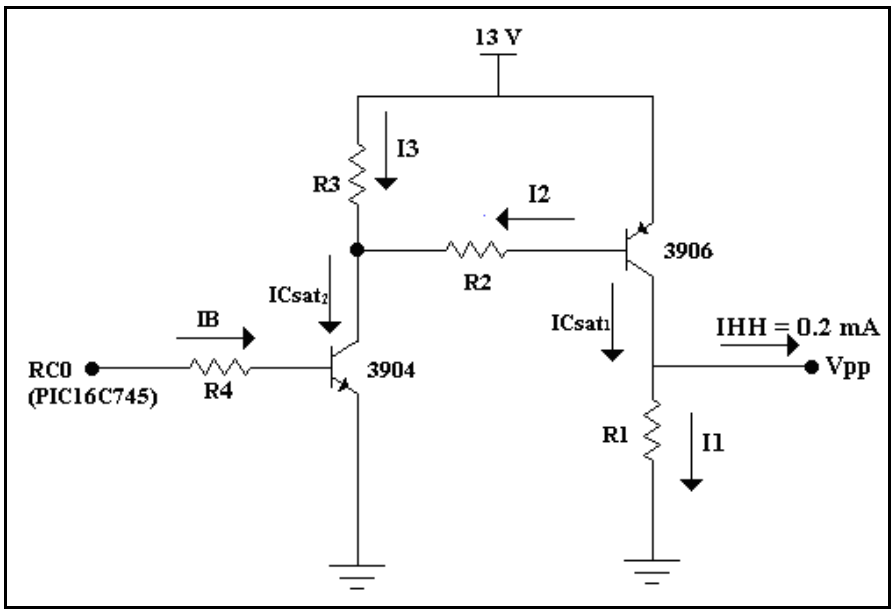


FIGURA 3-11: CIRCUITO PARA Vpp

Para asegurar un cero lógico a la entrada del pin Vpp y no desperdiciar energía de la fuente USB, se a resuelto asumir una R1 = 100KΩ.

I_{HH} = consumo de corriente máximo en la programación =200 µA = 0.2 mA (Hoja de Especificaciones para la Programación **[2]**).

Observando la Figura 3-11, se tiene:

$$I1 = \frac{13V - V_{CEsat(3906)}}{R1} = \frac{13V - 0.3V}{100K\Omega}$$

$$I1 = 0.127mA$$

De la Figura 3-11, se encuentra:

$$ICsat1_{(3906)} = I1 + I_{HH}$$

$$ICsat1_{(3906)} = 0.327mA$$

Para un β = 50 se tiene:

$$I_{B(3906)} = I2 = \frac{ICsat1_{(3906)}}{\beta} = 6.54\ \mu A$$

De la Figura 3-11, cuando el transistor 3906 está saturado (encendido), será porque está también saturado el 3904, por lo que se tiene para el cálculo de la R2 la siguiente expresión:

$$R2 = \frac{13V - V_{EB(3906)} - V_{CEsat(3904)}}{I2} = \frac{13V - 0.95V - 0.3V}{6.54 * 10^{-6} A}$$

$$R2 = 1.78M\Omega \approx 1.5M\Omega$$

Con el objetivo de ahorrar la mayor cantidad energía se asume R3= 68KΩ.

Entonces se tiene:

$$I3 = \frac{13V - V_{CEsat(3904)}}{R3} = \frac{13V - 0.3V}{68K\Omega}$$

$$I3 = 0.186mA$$

De la Figura 3-11

$$ICsat2_{(3904)} = I3 + I2$$

$$ICsat2_{(3904)} = 0.193mA$$

Para un β = 50 se tiene:

$$I_{B(3904)} = \frac{ICsat2_{(3904)}}{\beta} = 3.87\ \mu A$$

En la Figura 3-11 se puede observar que:

$$R4 = \frac{5V - V_{BEsat(3904)}}{I5} = \frac{5V - 0.95V}{3.87 * 10^{-6} A}$$

$$R4 = 1.046M\Omega \approx 1M\Omega$$

3.3.1.3 Circuito para el pin DATA, CLOCK y VDD

Para ver el proceso de programación de un dispositivo PIC, se a dispuesto de leds con el siguiente emparejamiento.

LED	FUNCIÓN	PIN PIC 16C745
ROJO	DATOS	RA5
VERDE	CLOCK	RC1
AMARILLO	VDD	RA4

TABLA 3-3: COLOR DEL LED CON SU RESPECTIVA FUNCIÓN

También, con el propósito de proteger los pines del PIC 16C745 que trabajan con las señales de DATOS, CLOCK y VDD, de eventuales daños en el dispositivo a programar, se ha diseñado el circuito de control mostrado en la Figura 3-12.

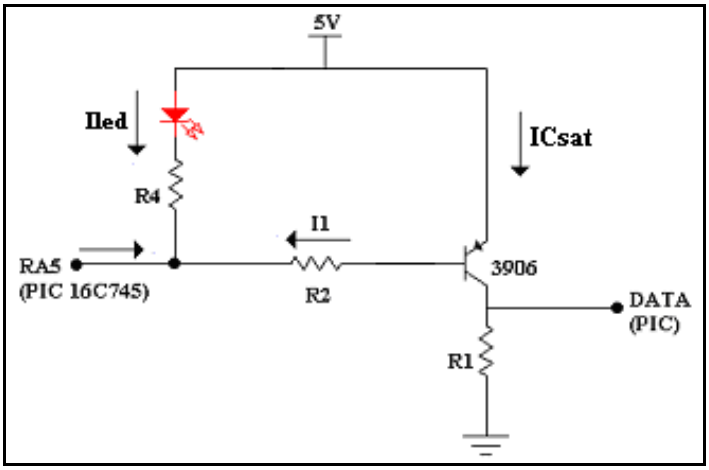


FIGURA 3-12: CIRCUITO DE CONTROL DE LEDS Y DATOS

En las hojas Técnicas de Especificaciones para la Programación no se indica la corriente que un pin DATA o CLOCK consume durante la programación y para no desperdiciar energía de la fuente USB, se asume una R1 = 1 KΩ.

En la Figura 3-12 se puede observar que:

$$ICsat = \frac{5V - V_{CEsat}}{R1} = \frac{5V - 0.3}{1K\Omega}$$

$$ICsat = 4.7mA$$

Para un β = 50 se tiene:

$$I_B = I1 = \frac{ICsat}{\beta} = 94\ \mu A$$

De la Figura 3-12, se tiene:

$$R2 = \frac{5V - V_{BEsat}}{I1} = \frac{5V - 0.95V}{94 * 10^{-6}}$$

$$R2 = 43.8K\Omega \approx 47K\Omega$$

Para un consumo en un led de 4 mA se tiene:

$$R4 = \frac{5V - 1.2V}{Iled}$$

$$R4 = 950\Omega \approx 1K\Omega$$

3.3.2 PARA MICROCONTROLADORES ATMEL

Ya que se ha diseñado los circuitos de control para los pines de programación del microcontrolador PIC, se los aprovecha para el manejo de los pines de programación de los microcontroladores ATMEL (además de otros circuitos para los pines de programación restantes del microcontrolador ATMEL).

3.3.2.1 Pines para programación de los microcontroladores ATMEL.

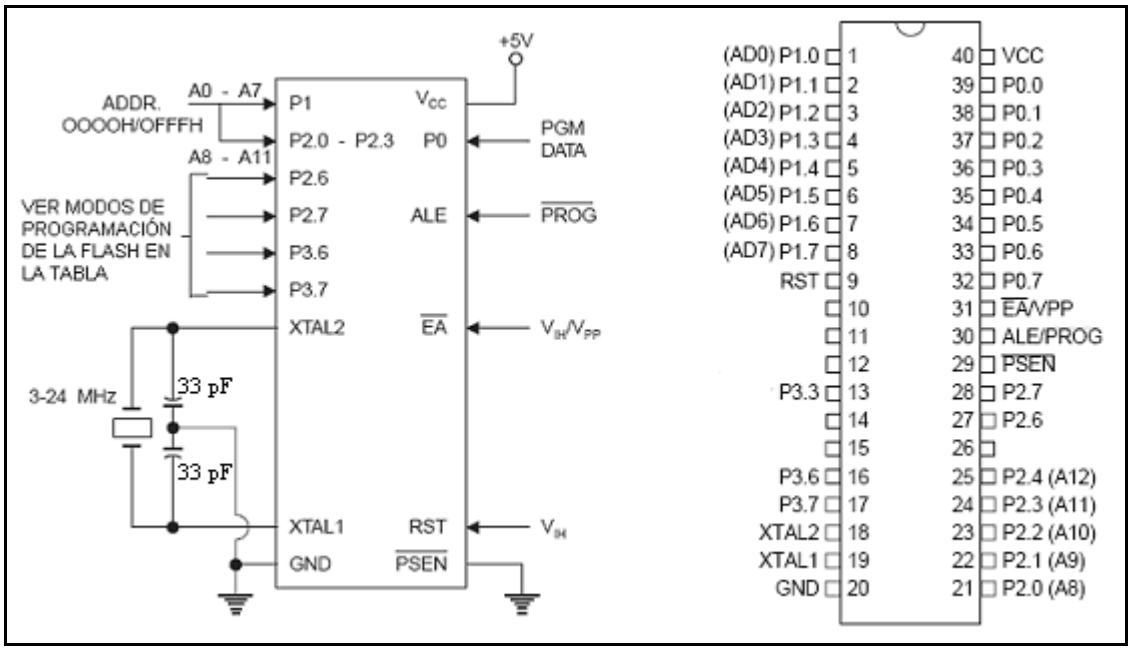


FIGURA 3-13: DIAGRAMA DE PINES DE LOS ATMEL AT89X51/52/55 [3]

Debido a que este dispositivo se programa en forma paralela, se ha utilizado todo el puerto B del PIC 16C745 para la entrada y salida de datos cuando se lea y escriba el microcontrolador ATMEL respectivamente.

Para generar el direccionamiento de las localidades de memoria del microcontrolador ATMEL a programarse se utilizarán dos circuitos integrados 393, donde cada uno posee dos contadores binarios de 4 bits. Para poder direccionar hasta 20 Kbytes de memoria (ATMEL AT89X55), se pone en cascada a los cuatro contadores binarios de 4 bits como se muestra en la Figura 3-14.

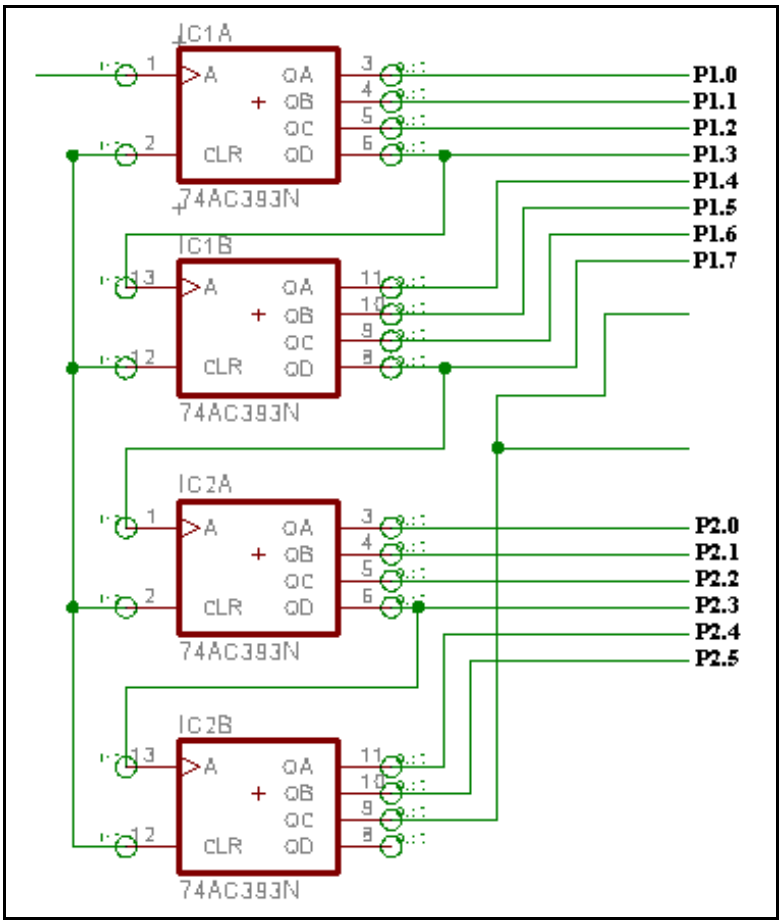


FIGURA 3-14: CONTADORES 74LS393 EN CASCADA PARA GENERAR LAS DIRECCIONES DEL ATMEL

Todos los pines involucrados en la programación de un ATMEL y su correspondencia con los pines del PIC 16C745 se detallan en la Tabla 3-4.

PINES DEL PIC 16C745	PINES DEL ATMEL (MODO PROGRAMACIÓN)
PORT B	PORT 0 (DATA)
RA0	P3.7
RA1	P3.6
RA2	P3.3
RA3	P2.7
RA4	P2.6
RA5	ALE/PROG
RC0	EA/Vpp
RC1	CLOCK(393)
Contador 393 (1)	P1.0 - P1-7
Contador 393 (2)	P2.0 - P2.3

TABLA 3-4: ASIGNACIÓN DE PINES DEL PIC 16C745 PARA PROGRAMAR ATMELs

Tanto en la Tabla 3-2 como en la Tabla 3-4, se observa que se utilizan los pines RA3 y RA5 del PIC 16C745, en diferentes funciones para microcontroladores PIC y ATMEL.

En el caso de los microcontroladores PIC, RA3 recibe datos y RA5 provee datos, lo que hace notar que estos dos pines estarían unidos en el pin DATA del microcontrolador PIC a programarse, pero como sólo pueden viajar datos en un solo sentido al programar o leer un microcontrolador PIC, entonces no trae complicaciones.

En el caso de los microcontroladores ATMEL, RA3 provee una señal de control al pin P2.7 y RA5 también provee una señal de control al pin ALE/PROG; al estar unidos estos pines, en algún momento se produciría un cortocircuito; por lo que se colocó una resistencia de 10 KΩ entre estos dos pines.

Las siguientes figuras muestran los circuitos implementados:

CAPÍTULO 4

DESARROLLO DEL SOFTWARE Y FIRMWARE DEL PROGRAMADOR “EPNprog”

Los microcontroladores y microprocesadores son dispositivos electrónicos programables y ejecutan un conjunto de instrucciones que tienen almacenadas en localidades de memoria denominada “Memoria de Programa”.

Las funciones del “EPNprog” son las de leer, borrar, programar la Memoria de Programa.

Además de la Memoria de Programa, algunos dispositivos tienen Memoria de Datos no volátil, que también puede ser programada con el EPNprog.

Para cumplir con la programación de los microcontroladores PIC y ATMEL, se sigue el procedimiento descrito en las Hojas de Programación que proporciona el fabricante.

Microchip, que es el fabricante de los microcontroladores PIC usa el método de programación serial, mientras que los dispositivos ATMEL que soporta el programador son programados en forma paralela.

Microchip agrupa a sus dispositivos en familias.

Para la programación de microcontroladores PIC, el fabricante requiere que se sigan los pasos que se encuentran en las Hojas de Programación, estos pasos están representados gráficamente como flujogramas. Las Hojas de Programación varían de familia a familia.

Los flujogramas le dicen al usuario:

- Cuando emitir un determinado comando* o dato
- Cuando ocurre un retardo de tiempo
- Cuando realizar una bifurcación
- Cuando Finalizar

*En el Capítulo 2 se explicó lo que era un Comando, y cuales eran sus funciones.

La bifurcación dentro de un flujograma quiere decir que dependiendo del resultado de la evaluación de una expresión lógica se sigue cierto camino. Esto se ve mejor representado en la siguiente Figura.

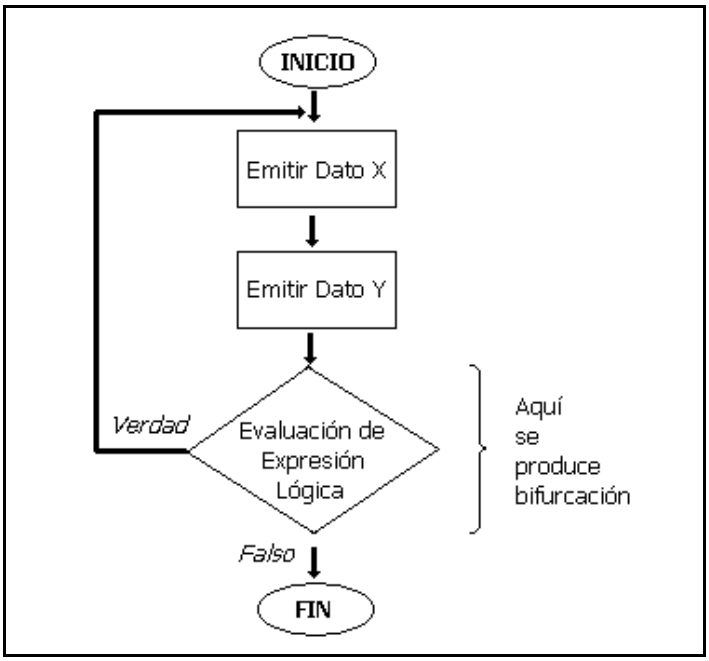


FIGURA 4-1: FLUJOGRAMA GENÉRICO DE PROGRAMACIÓN

4.1 DESARROLLO DE LA HERRAMIENTA PARA AGREGAR NUEVOS DISPOSITIVOS PIC A LA BIBLIOTECA DEL PROGRAMADOR EPNprog.

El programador EPNprog posee una herramienta que permite construir flujogramas para programar microcontroladores PIC. Así el usuario puede armar el flujograma que le permita programar un determinado dispositivo que no se encuentre en la lista de elementos soportados por el Programador EPNprog. Dentro de esta flexibilidad, el usuario puede además editar algún dispositivo existente para modificarlo o también puede eliminarlo de la lista.

Entre los atributos que posee la herramienta se encuentran:

- Emitir bit por bit un DATO**, estos bits deben de estar sincronizados con una señal de reloj.
- Permitir que la extensión de cada DATO** sea variable.
- Ordenar los DATOS** según la secuencia que mande el Flujograma.
- Poder intercalar retardos de tiempo y bifurcaciones según el Flujograma.

** Un comando es tratado como un dato, y en este caso se lo nombra como DATO.

Todo lo que anteriormente se explicó, constituyen los pasos para la programación de microcontroladores PIC y a partir de estos delineamientos se detalla toda la filosofía aplicada para cumplir el objetivo.

Para que la herramienta cumpla sus objetivos se han creado unas estructuras que tienden a representar a cada uno de los bloques que forman un flujograma. A estas estructuras se las denominará como **UNIDAD** de flujograma.

Cada UNIDAD está formada por un conjunto de bytes y cada una debe ser configurada de acuerdo a las necesidades del usuario.

Se han creado cuatro tipos de UNIDADES, con las cuales puede crear cualquier flujograma para programar microcontroladores PIC. Estas UNIDADES son:

- UNIDAD “COMANDO/DATO”.
- UNIDAD “SALTO”
- UNIDAD “TIEMPO”
- UNIDAD “FIN”

Dentro del primer byte de cada UNIDAD se encuentra el identificador del tipo de UNIDAD. (Ver Tabla 4-1).

Tipo de Unidad	Número Identificador
COMANDO/DATO	0X01
SALTO	0X02
TIEMPO	0X03
FIN	0X07

TABLA 4-1: TIPOS DE UNIDADES

A continuación se detallan las funciones y además se ve la constitución de cada UNIDAD.

4.1.1 UNIDAD “COMANDO/DATO”:

Las funciones de esta UNIDAD son las de emitir un comando o un dato sincronizado con una señal de reloj. Esta UNIDAD se compone de 5 bytes, donde en cada byte se debe almacenar información referente a:

- El tamaño del comando o dato.
- Si la UNIDAD involucra un proceso de lectura o escritura de datos adicionales.
- Tamaño de datos adicionales en caso de que existan.
- Constante de posicionamiento a la próxima UNIDAD.

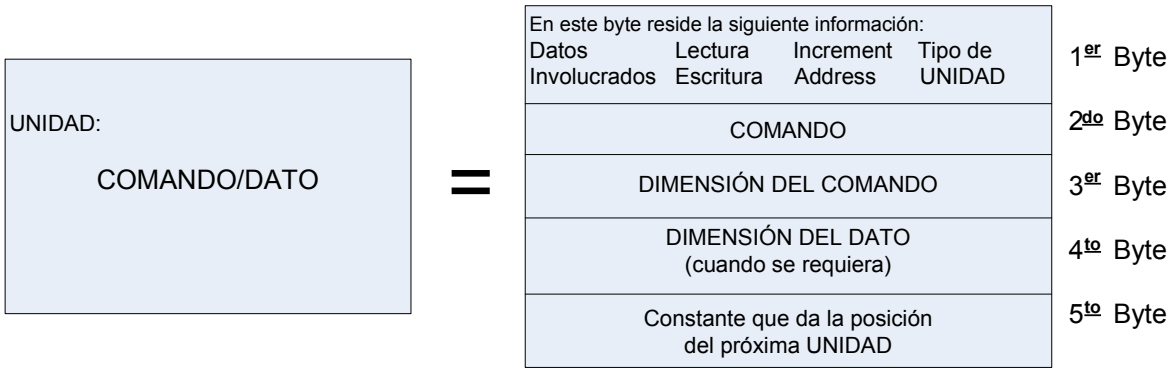


FIGURA 4-2: UNIDAD “COMANDO/DATO”

4.1.2 UNIDAD “SALTO”:

Esta UNIDAD se compone de 5 bytes y es usada para tomar la decisión de llevar el flujograma por un camino o por otro, después de haber evaluado alguna condición que el usuario desee. Los 5 bytes y la información que cada uno almacena se muestran en la Figura 4-3.

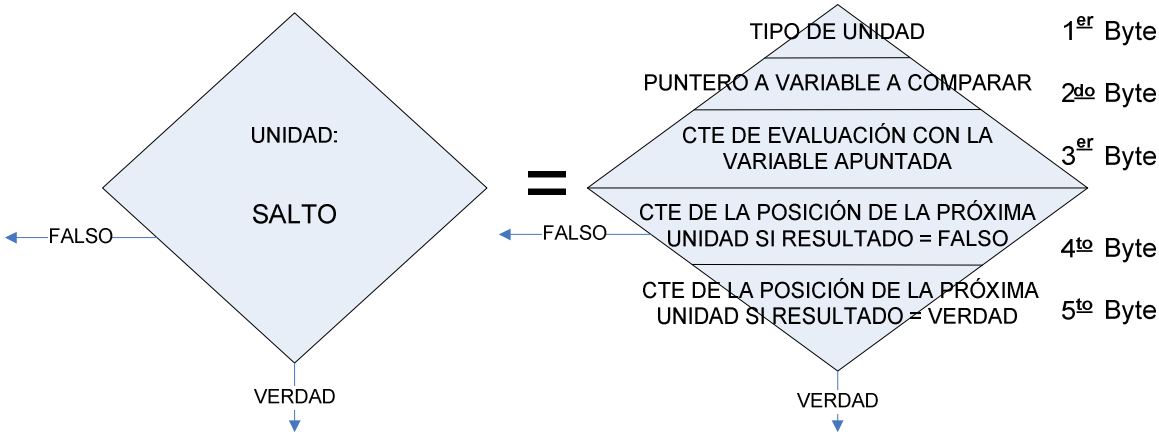


FIGURA 4-3: UNIDAD “SALTO”

4.1.3 UNIDAD “Tiempo”

Esta UNIDAD se compone de 3 bytes, y es usado para producir un retardo de tiempo donde el flujograma lo exija.

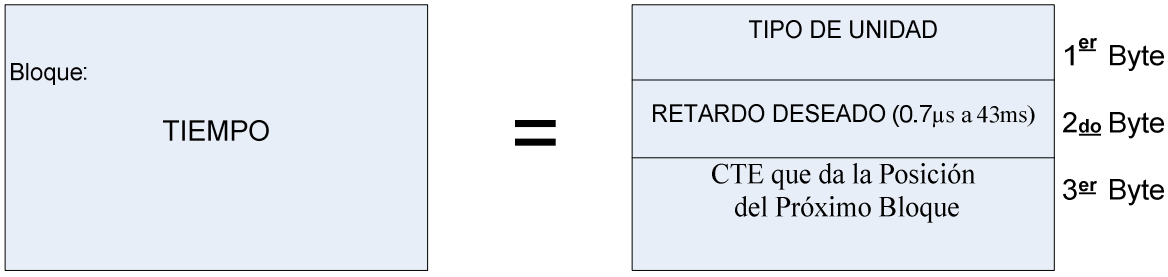


FIGURA 4-4: UNIDAD “TIEMPO”

4.1.4 UNIDAD “Fin”

Esta UNIDAD es de 1 byte y tiene el número 7 para indicar que un flujograma llegó a su final.

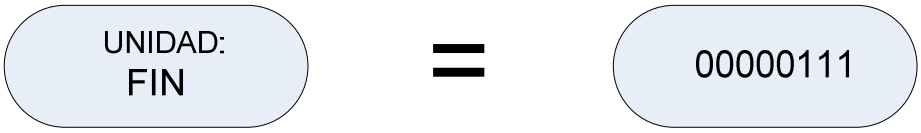
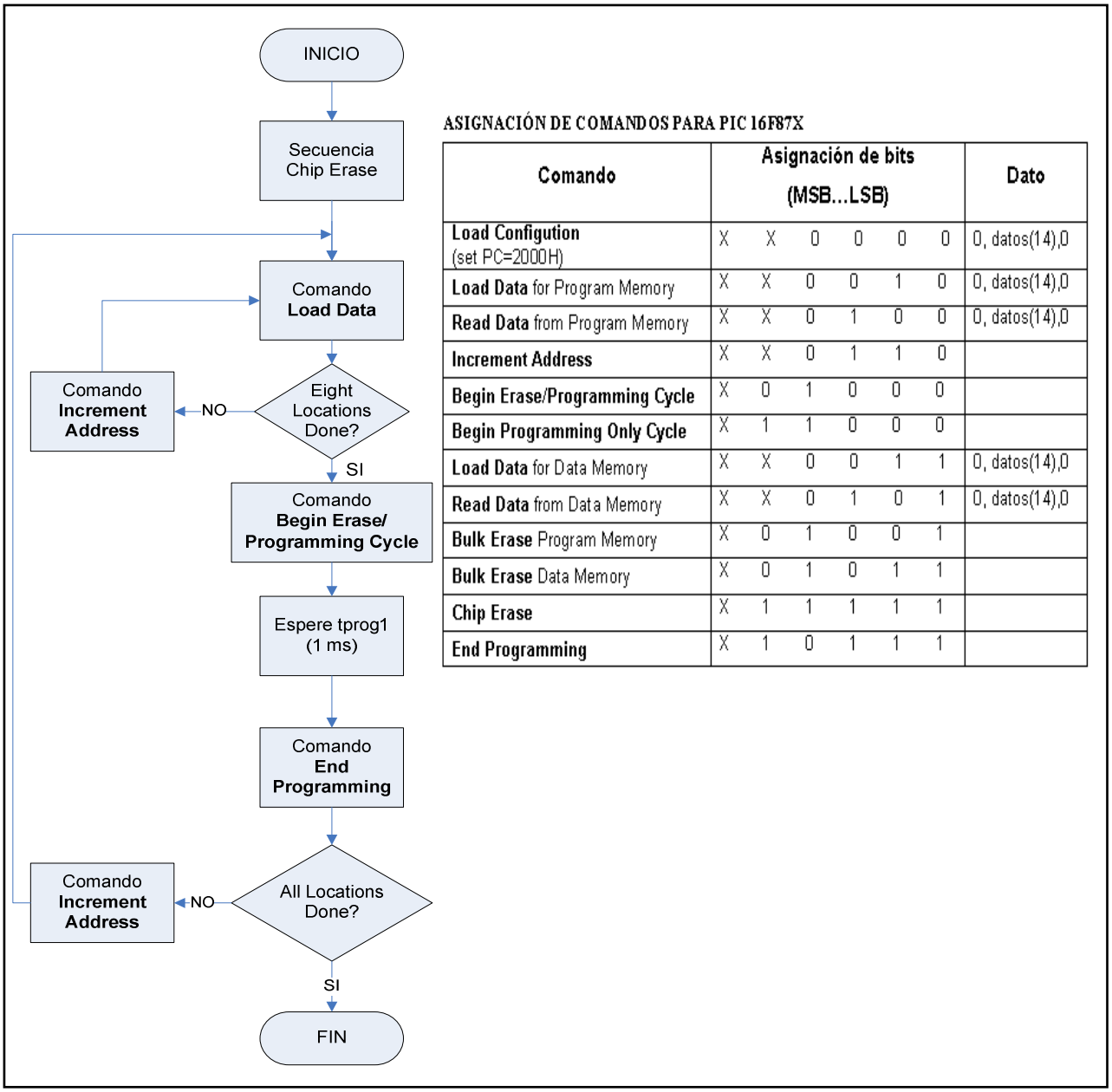


FIGURA 4-5: UNIDAD “FIN”

4.1.5 EJEMPLO DEL USO DE LAS UNIDADES

Para familiarizar al usuario con el uso de la herramienta, se ha optado por implementar un ejemplo de prueba.

En las siguientes líneas se arma el flujograma con las respectivas Unidades, para la programación de la “Memoria de Programa” de los microcontroladores PIC de la familia 16F87XA. Este flujograma se encuentra en la Figura 4-6.

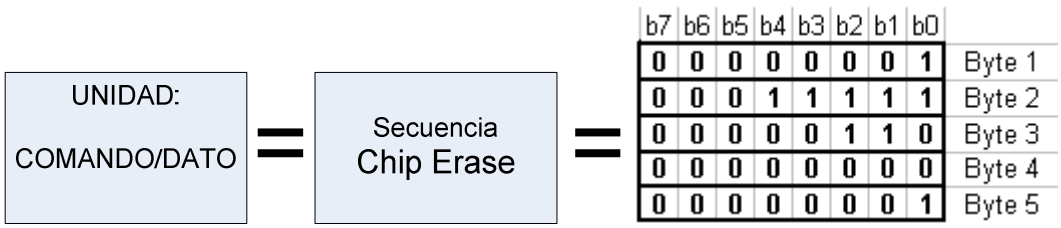


ASIGNACIÓN DE COMANDOS PARA PIC 16F87X

Comando	Asignación de bits (MSB...LSB)						Dato
Load Configution (set PC=2000H)	X	X	0	0	0	0	0, datos(14),0
Load Data for Program Memory	X	X	0	0	1	0	0, datos(14),0
Read Data from Program Memory	X	X	0	1	0	0	0, datos(14),0
Increment Address	X	X	0	1	1	0	
Begin Erase/Programming Cycle	X	0	1	0	0	0	
Begin Programming Only Cycle	X	1	1	0	0	0	
Load Data for Data Memory	X	X	0	0	1	1	0, datos(14),0
Read Data from Data Memory	X	X	0	1	0	1	0, datos(14),0
Bulk Erase Program Memory	X	0	1	0	0	1	
Bulk Erase Data Memory	X	0	1	0	1	1	
Chip Erase	X	1	1	1	1	1	
End Programming	X	1	0	1	1	1	

FIGURA 4-6: FLUJOGRAMA PARA PROGRAMAR LA MEMORIA DE PROGRAMA DEL PIC 16F87XA. [2]

1.- Como primera acción a realizarse se ve la “Secuencia Chip Erase”, la cual calza dentro de una UNIDAD “COMANDO/DATO”. A continuación se muestra lo que se debe llenar en la UNIDAD:



b7	b6	b5	b4	b3	b2	b1	b0	
0	0	0	0	0	0	0	1	Byte 1
0	0	0	1	1	1	1	1	Byte 2
0	0	0	0	0	1	1	0	Byte 3
0	0	0	0	0	0	0	0	Byte 4
0	0	0	0	0	0	0	1	Byte 5

FIGURA 4-7: BYTES DE LA UNIDAD CHIP ERASE”

Explicación de cada byte de esta UNIDAD.

Byte 1: Aquí se encuentra información referente a:

b4	b3	b2	b1	b0

Tipo de UNIDAD; y como se trata de una UNIDAD "COMANDO/DATO" el valor será 0 0 0 0 1.

b5 Bit que indica si es comando que incrementa la Address. En este caso por tratarse de un Chip Erase el valor es 0L (no implica Address).

b6 Bit que indica que este comando hace Escritura (0L) o Lectura (1L) de datos, en este caso por tratarse de un Chip Erase el valor por defecto es 0L.

b7 Bit que indica si la UNIDAD involucra el procesamiento de datos, ya sea en escritura o lectura; y como un Chip Erase para esta familia no necesita ningún dato adicional, el valor es igual a 0L.

Byte 2: Dentro de este byte se encuentra el comando que se va a emitir, y como se aprecia en la Tabla que está junto al flujograma de la Figura 4-6, el comando Chip Erase es igual 1 1 1 1 1 con lo que este byte tiene la siguiente estructura:

0	0	0	1	1	1	1	1
b7	b6	b5	b4	b3	b2	b1	b0

Byte 3: Aquí está la dimensión del comando. Esta información es necesaria porque un comando que se emite debe estar sincronizado con la señal de reloj. Como se aprecia en la Figura 4-8 (tomado de las Hojas de Especificaciones), un comando necesita 6 pulsos de reloj, por lo que el valor que hay que llenar aquí es seis.

0	0	0	0	0	1	1	0

.

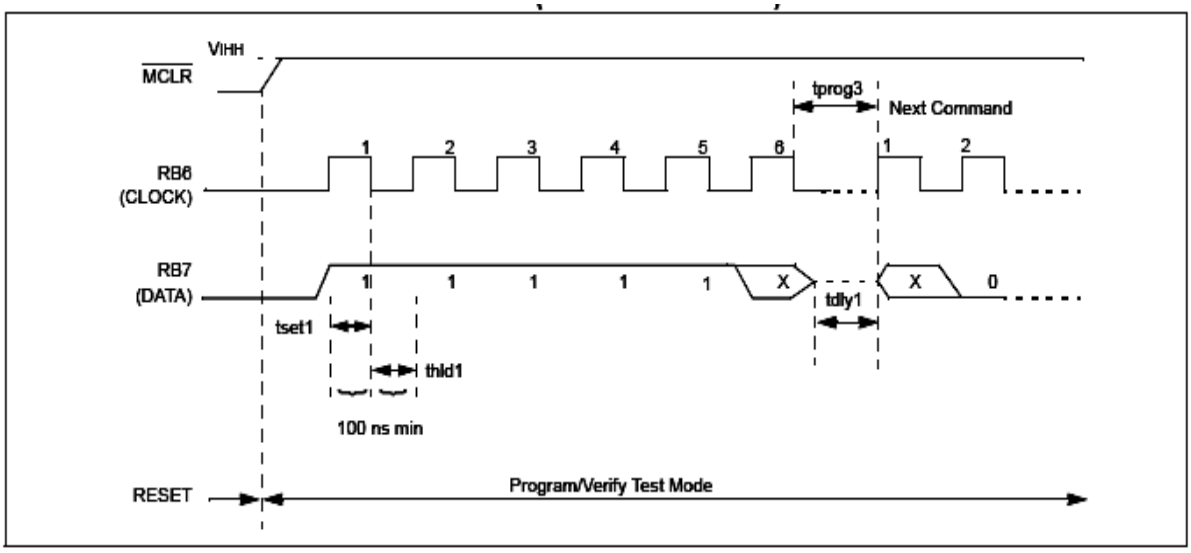


FIGURA 4-8: COMANDO CHIP ERASE (PROGRAM/VERIFY) [2]

Byte 4: En este byte lo que se pone, es la dimensión de los datos que involucra el comando, y como el comando Chip Erase no involucra datos, este byte será llenado con ceros.

0	0	0	0	0	0	0	0

Byte 5: Aquí se pone el número de pasos que se necesita para alcanzar al próxima UNIDAD del flujograma, este número de pasos puede ser hacia adelante o hacia atrás, esta característica se la coloca en el bit 7 de este byte.

0	0	0	0	0	0	1
b6	b5	b4	b3	b2	b1	b0

Como se observa en la Figura 4-6, se necesita un paso para alcanzar la UNIDAD “Comando Load Data”.

0
b7

Bit que indica que el paso será hacia delante { **0L** hacia delante / **1L** hacia atrás

2.- La siguiente UNIDAD es un “Comando Load Data”, el cual es una UNIDAD tipo “COMANDO/DATO” y su forma de llenado es la siguiente:

UNIDAD: COMANDO/DATO	=	Comando Load Data	=

b7	b6	b5	b4	b3	b2	b1	b0	
1	0	0	0	0	0	0	1	Byte 1
0	0	0	0	0	0	1	0	Byte 2
0	0	0	0	0	1	1	0	Byte 3
0	0	0	1	0	0	0	0	Byte 4
0	0	0	0	0	0	0	1	Byte 5

FIGURA 4-9: BYTES DE LA UNIDAD “LOAD DATA”

Explicación de cada byte:

Byte 1:

b4	b3	b2	b1	b0

Tipo de UNIDAD; y como se trata de una UNIDAD "COMANDO/DATO" el valor será 0 0 0 0 1.

b5

Aquí el valor a llenarse es 0L porque es una UNIDAD que no incrementa la Address.

b6

Aquí el valor a llenarse es 0L porque es un comando que no involucra Escritura.

b7

Se llena con 1L porque inmediatamente después de que el comando sea emitido, se necesitará cargar un dato en la posición de Memoria de Programa, por lo tanto este comando (UNIDAD) sí involucra datos.

Byte 2: Aquí se llena con el comando que se va a emitir y como se aprecia en la Tabla de Asignación de Comandos para el PIC 16F87XA, que se encuentra junto al flujograma de la Figura 4-6 es 0 0 0 1 0.

0	0	0	0	0	0	1	0

Byte 3: Aquí está la dimensión del comando anterior, y como se explicó en la UNIDAD Chip Erase el valor a llenarse es seis.

0	0	0	0	0	1	1	0

Byte 4: Como se está dentro de una UNIDAD donde se va a escribir un dato de 14 bits de longitud en la Memoria de Programa, aquí se llena el número de ciclos de reloj asociados con el dato.

Supuestamente se colocaría un valor igual a 14, pero en realidad es 16, porque además de los 14 bits se necesitan 2 bits adicionales, que son el de INICIO y el de PARADA, que en las Hojas de Especificaciones los nombra, pero de este par de bits no hay que preocuparse porque el Firmware ya los ubica automáticamente en la posición adecuada.

Para entender mejor vease la Figura 4-10.

0	0	0	1	0	0	0	0

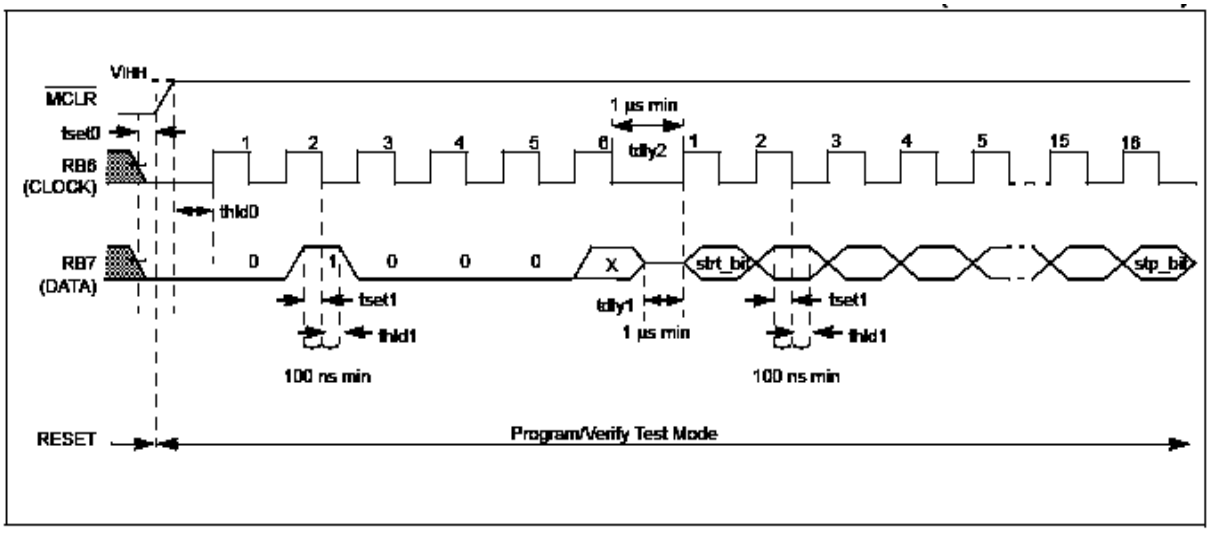


FIGURA 4-10: COMANDO LOAD DATA PARA LA MEMORIA DE PROGRAMA DEL PIC 16F87XA [2]

Byte 5: En esta UNIDAD se coloca el número 1, porque se avanza un sólo paso hasta la siguiente UNIDAD (y es hacia adelante).

0	0	0	0	0	0	0	1

3.- La siguiente UNIDAD corresponde a una UNIDAD "SALTO" y su forma de llenado es la siguiente:

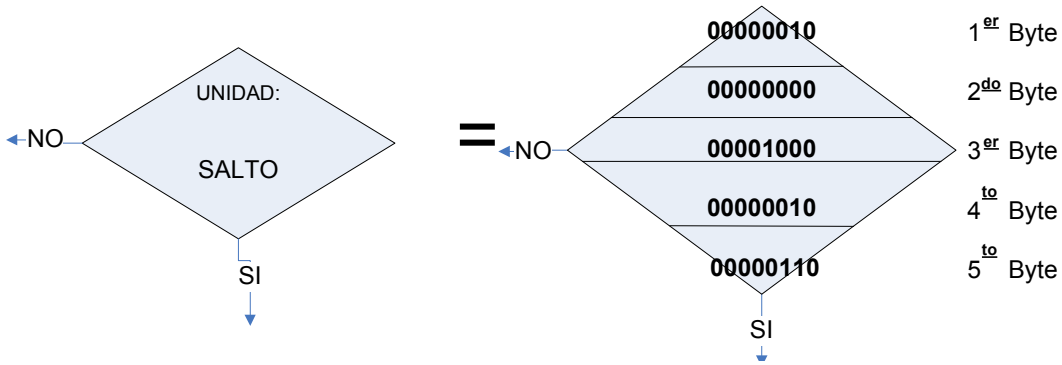


FIGURA 4-11: BYTES DE LA UNIDAD "EIGHT LOCATIONS DONE?"

Explicación de cada byte de la UNIDAD "SALTO"

Byte 1: Byte identificador del tipo de UNIDAD, en este caso es igual a 2 decimal.

0	0	0	0	0	0	1	0

Byte 2: En esta UNIDAD el segundo byte, lo que en realidad representa es un puntero a una posición de memoria RAM del dispositivo USB, las tres posibles localidades de memoria RAM que se pueden apuntar, llevan en cuenta los siguientes datos:

LOCALIDAD	FUNCIÓN
0X00	Cada que se emita un comando que involucre datos, se incrementa esta localidad en uno.
0X01	Cuando se completa la lectura o escritura de todo el mapa de memoria se pone este byte en 010000.
0X02	Cada que se emita un comando que incrementa la Address esta localidad se incrementa en uno.

TABLA 4-2: FUNCIÓN DEL SEGUNDO BYTE DENTRO DE LA UNIDAD "SALTO"

Por lo tanto el segundo byte de esta UNIDAD tiene que ser igual a 0X00 ,

0	0	0	0	0	0	0	0

porque se va a evaluar el número de datos que se han cargado.

Byte 3: En este byte se almacena la constante que se involucra con el contenido de la localidad apuntada por el byte anterior, y dependiendo del resultado se pasará al cuarto byte o quinto byte de la misma UNIDAD.

Para este ejemplo, y como lo muestra el flujograma de la Figura 4-6, el valor será igual a ocho (Eight Loads Done?).

Byte 4: Byte que da el número de pasos necesarios, para alcanzar la UNIDAD correspondiente, en caso de que la condición evaluada de un resultado FALSO.

Byte 5: Byte que da el número de pasos necesarios, para alcanzar la UNIDAD correspondiente, en caso de que la condición evaluada de un resultado VERDAD.

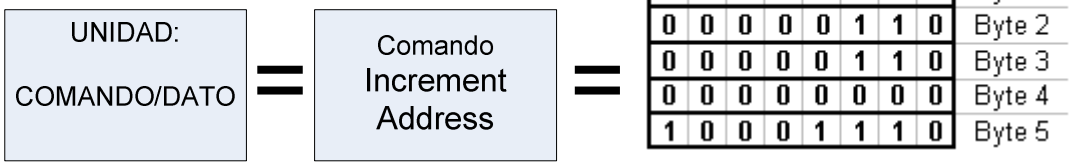
Los números con los cuales el usuario llene estos dos bytes dependerá de la UNIDAD siguiente que se coloque (en el ejemplo hay dos UNIDADES posibles: a) UNIDAD Increment Address Comand ó b) Begin Erase/Program Comand).

Se optará por ubicar como UNIDAD siguiente al "Comando Increment Address". Por consiguiente el cuarto byte de la UNIDAD "SALTO" se llenará con el número 2, que es el número de pasos necesarios para alcanzar a la UNIDAD "Comando

Increment Address" que es lo que se hace cuando la condición evaluada es FALSO.

El quinto byte por consiguiente será llenado con un número 6, que corresponde al número de pasos que hay que dar para alcanzar a la UNIDAD "Comando Begin Erase/Program" saltándose así todo la UNIDAD "Comando Increment Address".

4.- La UNIDAD "Comando Increment Address" corresponde a una UNIDAD "COMANDO/DATO" y su forma de llenado se muestra a continuación:



b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	0	0	1	Byte 1
0	0	0	0	0	1	1	0	Byte 2
0	0	0	0	0	1	1	0	Byte 3
0	0	0	0	0	0	0	0	Byte 4
1	0	0	0	1	1	1	0	Byte 5

FIGURA 4-12: BYTES DE LA UNIDAD "INCREMENT ADDRESS"

Explicación de cada byte de esta UNIDAD:

Byte 1: La explicación es semejante a los anteriores UNIDADES "COMANDO/DATO" implementados, con la acotación de que aquí se pone 1L en el b5, para llevar en cuenta, cuantas veces se ha emitido este comando (Address), en caso de que esta información se necesite en otro punto del flujograma.

Byte 2: Se ve que de la Tabla que está junto al flujograma de la Figura 4-6, éste es igual a: Increment Address = 0 0 1 1 0.

Byte 3: Número de ciclos de reloj sincronizados con el comando Increment Address, y al igual que los anteriores comandos el valor es 6 decimal = 0 0 1 1 0.

Byte 4: Esta UNIDAD no involucra datos, por lo tanto el valor a llenarse es cero.

Byte 5: Número de pasos para alcanzar la siguiente UNIDAD, y como se aprecia en la Figura 4-6, la UNIDAD que sigue, es de nuevo la UNIDAD "Comando Load Dato", se debe decrementar 14 posiciones.

0	0	0	1	1	1	0
b6	b5	b4	b3	b2	b1	b0

= 14 posiciones.

= Indicativo de que el número de posiciones son de decremento.

5.- La siguiente UNIDAD es “Comando Begin Programming Only” y su forma de llenado corresponde a:

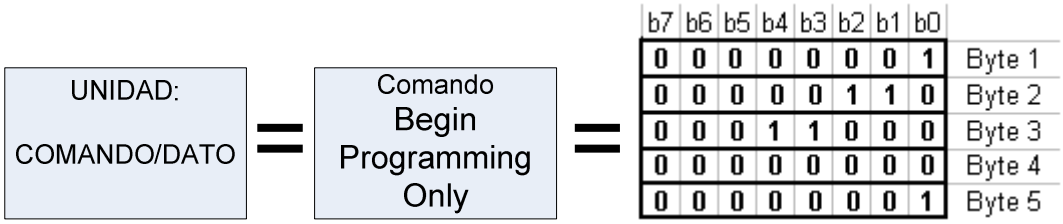


FIGURA 4-13: BYTES DE LA UNIDAD “BEGIN PROGRAMMING ONLY”

Todo lo anterior explicado para una UNIDAD “COMANDO/DATO” se aplica a esta UNIDAD, por lo tanto no se vuelve a repetir.

6.- UNIDAD “Espere tprog1 (1 ms)” corresponde a la UNIDAD “TIEMPO” y se lo llena para alcanzar el retardo de 1 ms que se ve en la Figura 4-6, de la siguiente manera:

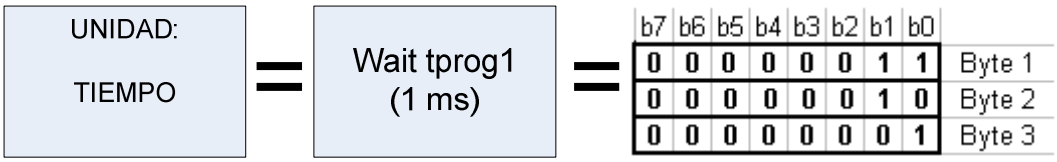


FIGURA 4-14: BYTES DE LA UNIDAD “WAIT tprog 1”

Una UNIDAD “TIEMPO” consta de 3 bytes, como ya se explicó, y la función de cada uno se detalla a continuación:

Byte 1: Número identificador de UNIDAD y es igual $3_D = 1\ 1_B$

Byte 2: División de frecuencia para el timer 0 del PIC USB 16C745 (para generar retardo variable).

Byte 3: Número de pasos para alcanzar la siguiente UNIDAD = $0\ 1_B$

Entonces para este caso se requiere un retardo de 1 ms, esto se logra asignando el byte = $0\ 1\ 0_B$ para un retardo de 1.36 ms como se ve en la Tabla 4-3.

Byte 2	Preescaler	Retardo (ms)
000	8	0.34
001	16	0.68
010	32	1.36
011	64	2.73
100	128	5.46
101	256	10.92
110	512	21.84
111	1024	43.69

TABLA 4-3: RETARDO QUE CUMPLE CADA BIT DENTRO DEL BYTE 2.

Los valores de la Tabla 4-3 son tomados según la ecuación:

$$\text{Retardo} = \frac{4}{fosc} * 256 * \text{Pr}\,eescaler \qquad \textbf{(Ec. 4-1)}$$

7.- A continuación se tiene la UNIDAD “Comando End Programming” que corresponde a una UNIDAD “COMANDO/DATO” y la forma de llenarlo se muestra a continuación:

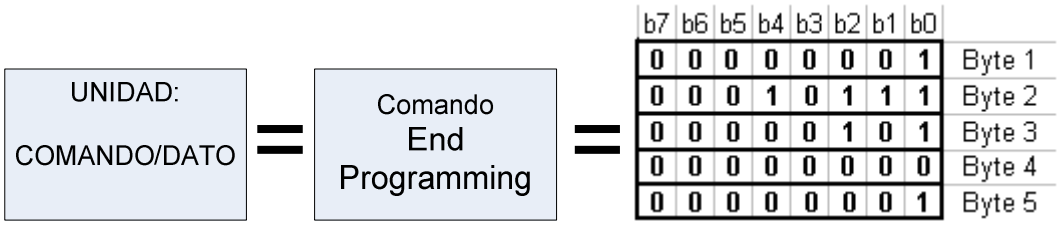


FIGURA 4-15: BYTES DE LA UNIDAD “END PROGRAMMING”

8.- Por último se tiene otra UNIDAD “SALTO” (All Locations Done?) en el que se evalúa, para ver si todos los datos que se quiere programar, ya se han programado.

Esta evaluación se la hace mediante el segundo byte, que apunta a la segunda localidad de Memoria RAM del microcontrolador PIC 16C745 donde se encuentra esta información.

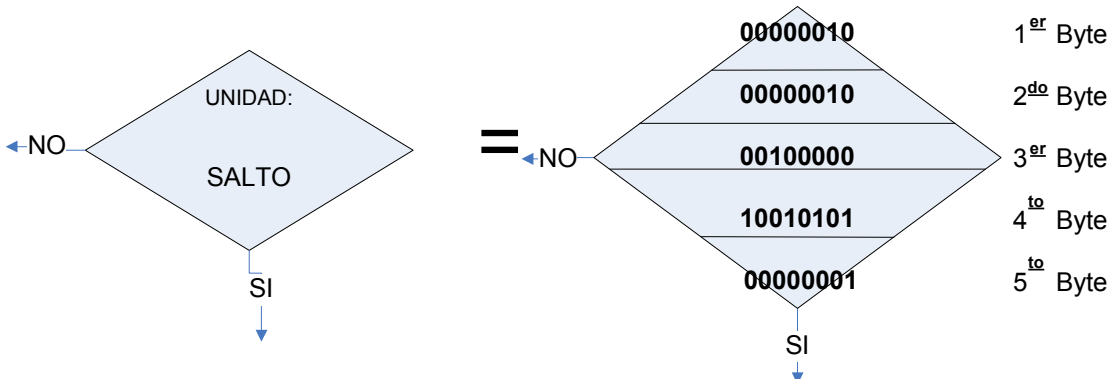


FIGURA 4-16: BYTES DE LA UNIDAD "ALL LOCATIONS DONE?"

Mientras no se complete de programar todos los datos, la segundad localidad de Memoria RAM vale cero, y si se completa, pasa a valer 100000_B, por lo que la constante de evaluación que va en el byte 3 de la UNIDAD SALTO es 100000_B.

El byte 4 dará el número de pasos que se a de seguir para alcanzar la UNIDAD "Comando Increment Address" cuando no se termine de programar todas los datos.

Se puede ubicar una nueva UNIDAD "Comando Increment Address", pero debido a que el flujograma posibilita a que se use la UNIDAD "Comando Increment Address" anterior, se lo ha usado, Entonces los 7 bits menos significativos del byte 4, se llenan con el número 0010101_B que en decimal es el número 21 y es el número de pasos necesarios para alcanzar la UNIDAD "Comando Increment Address" anterior, no olvidarse que el número de pasos es hacia atrás, por lo que el bit 7 del byte 4 debe ser igual a 1L.

Si se ha completado de programar todo, el byte 5 de la UNIDAD "SALTO" indica el número de posiciones para alcanzar la próxima UNIDAD, en este caso es igual a 1.

9.- El flujograma llega a su fin por una UNIDAD "FIN" que tiene como clave el número 7.

Después de haber armado algún flujograma, cada una de las UNIDADES usadas aportará con bytes para crear el código cifrado.

En la siguiente Figura se visualiza al flujograma implementado junto con el que provee Microchip en las Hojas de Programación (PIC 16F87XA).

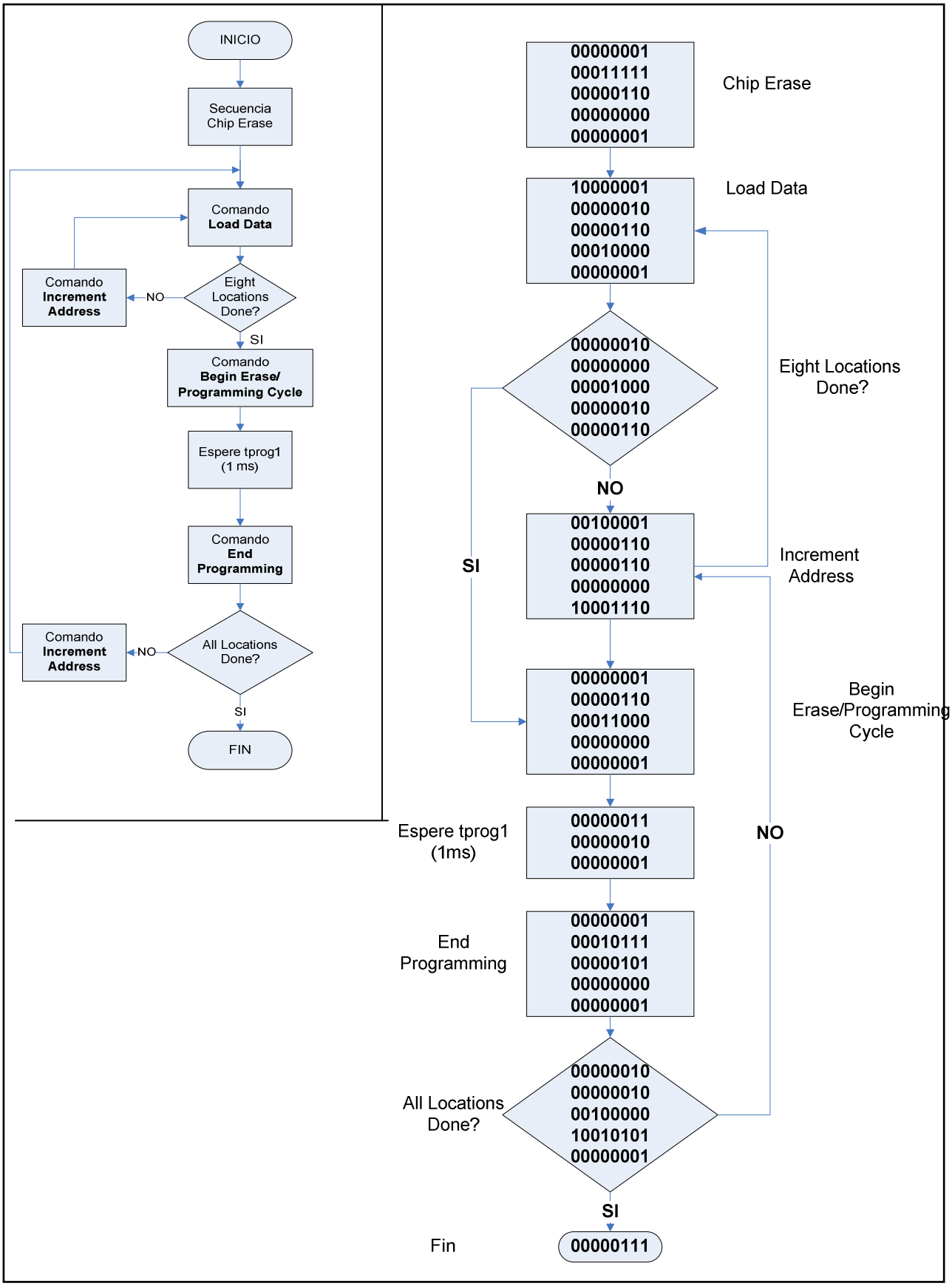


FIGURA 4-17: FLUJOGRAMA IMPLEMENTADO CON RELACIÓN AL FLUJOGRAMA DE MICROCHIP

4.2 DESARROLLO DEL SOFTWARE Y FIRMWARE

En el presente proyecto de titulación se programan los microcontroladores ATMEL (51/52/55) y PIC mediante la ejecución del software del PC, juntamente con el Firmware del dispositivo USB PIC 16C745.
Para realizar la programación, el proceso seguido por el Programador EPNprog es el siguiente:

- El software es una interfaz con el usuario, donde se puede escoger el dispositivo deseado, para realizar lectura, escritura, borrado.
- Luego de que el usuario ha decidido hacer alguna de las tres acciones, se transmite la acción cifrada en código, al Firmware (16C745) mediante la interfaz USB.
- El firmware, luego de recibir el código cifrado, lo traduce e interpreta para así manejar los diferentes pines de programación.
- Por último, en caso de que la acción requerida por el usuario implique el tráfico de datos (leídos o escritos) en cualquiera de las dos direcciones, se procede a realizarlos en paquetes discretos, conforme avanza la operación.

4.2.1 DESARROLLO DEL FIRMWARE PARA EL PROGRAMADOR EPNprog.

Luego de haber presentado la forma de construir flujogramas, que sirven para la programación de dispositivos seriales, se puede empezar el desarrollo del firmware que se ejecuta en el dispositivo PIC 16C745; este dispositivo soporta como interfaz de comunicación la norma USB, además de la RS 232.

El firmware está desarrollado para el PIC 16C745, a continuación se detallan las características relevantes de este dispositivo, desde la perspectiva de esta aplicación:

- Posee 22 pines de I/O **[4]**
- 256 bytes de Memoria RAM **[4]**

- 3 Timers/Counters **[4]**
- 2 módulos de Captura/Comparación/**PWM [4]**
- Bus de Comunicación “Universal Serial Bus” (USB 1.1), dispositivo de **Baja Velocidad [4]**
- Posee 4 opciones de Oscilador, y con la que se trabaja es:
  H4 PLL; que es para cristales/resonadores de alta velocidad con habilitación del PLL **[4]**
- Modo de Borrado del dispositivo, mediante Luz Ultravioleta **[4]**, que se constituye en una desventaja

En cuanto tiene que ver al Oscilador, se debe decir que se utilizó un cristal de 6 MHz y como PLL **[4]** está habilitado, se tiene una frecuencia de trabajo de 24 MHz, esta información fué tomada de las Hojas de Datos para el PIC 16C745/765 (Anexo A, Sección 13.2 Oscilator Configurations). Todos los cálculos posteriores, que involucren a la frecuencia, serán con un valor de 24 MHz.

Siguiendo con el desarrollo del Firmware, se implementa el código necesario para llevar adelante la programación de microcontroladores PIC o ATMEL.
El código conserva una estructura modular, para facilitar el uso de partes de código que son llamadas en forma reiterada por el programa principal.

4.2.1.1 Programa Principal del FIRMWARE

Corresponde a la parte fundamental del Firmware donde el programador “EPNprog” se encontrará en modo de espera, hasta que el usuario mediante PC y comunicación USB, le solicite llevar adelante una acción de programación; pudiendo ser ésta para microcontroladores PIC o ATMEL;. En la Figura 4-18 se muestra el flujograma del programa principal.

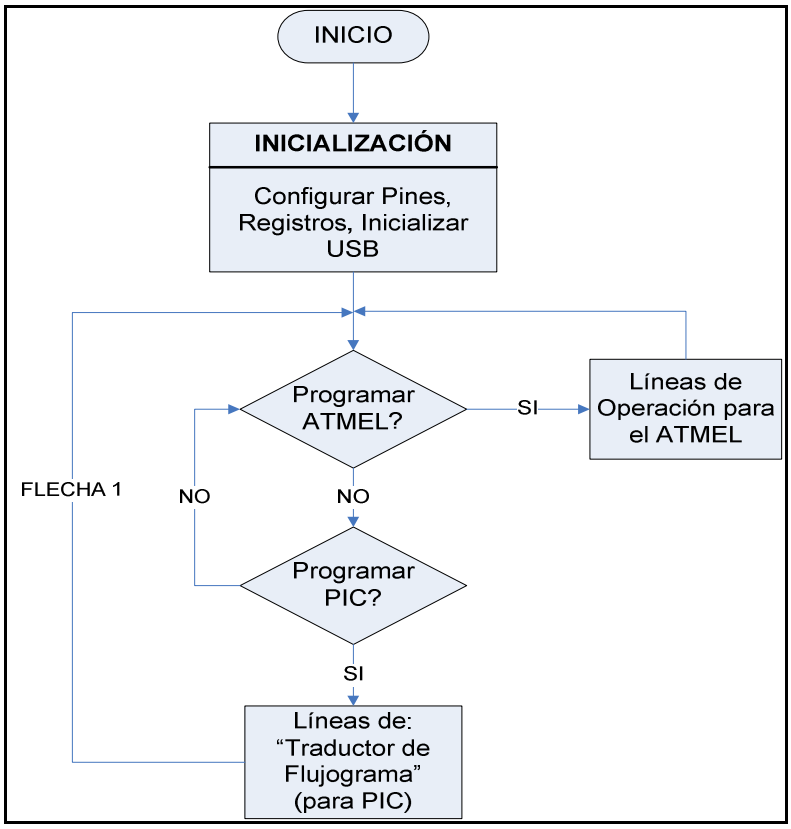


FIGURA 4-18: FLUJOGRAMA DEL PROGRAMA PRINCIPAL DEL FIRMWARE

- **Bloque "Traductor de Flujograma" (Para microcontroladores PIC)**

Aquí las funciones a realizar son las siguientes:

- Poner el pin VDD a 5 V.
- Poner en funcionamiento el Conversor DC/DC Elevador.
- Poner el pin Vpp (13 V) correspondiente al microcontrolador PIC.
- Descifrar y ejecutar el código flujo descargado del PC.
- Llamar a la subrutina Finalizar, que es la encargada de inicializar nuevamente todas las variables implicadas.
- El flujograma para este bloque se muestra a continuación:

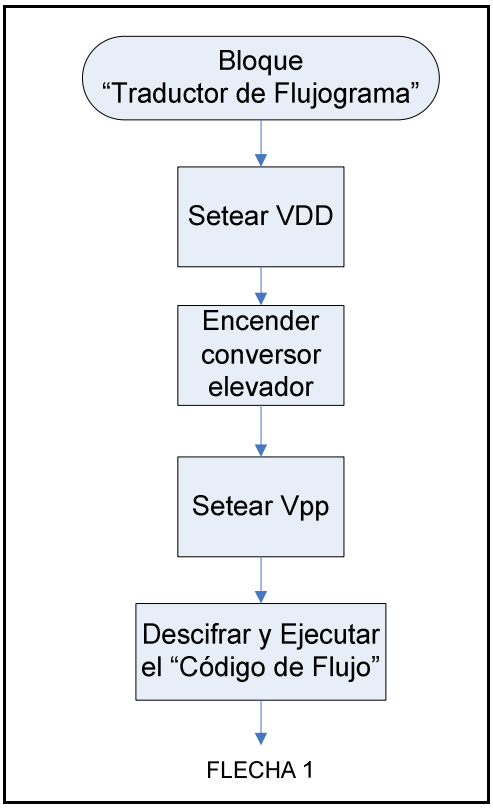


FIGURA 4-19: FLUJOGRAMA DEL BLOQUE “TRADUCTOR DE FLUJOGRAMA”

- **Bloque Descifrar y Ejecutar el” Código de Flujo”**

El bloque Descifrar y Ejecutar el “Código de Flujo” es la parte operativa encargada de programar al dispositivo PIC seleccionado.

El modo de funcionamiento de este bloque es:
Una vez que se está en este punto, es porque ya antes ha llegado y se ha almacenado en la memoria RAM el código cifrado de algún flujograma.

Se va reconociendo y ejecutando cada una de las UNIDADES que pertenecen al flujograma, mediante punteros (direccionamiento indirecto).

Las partes constitutivas de este bloque son:

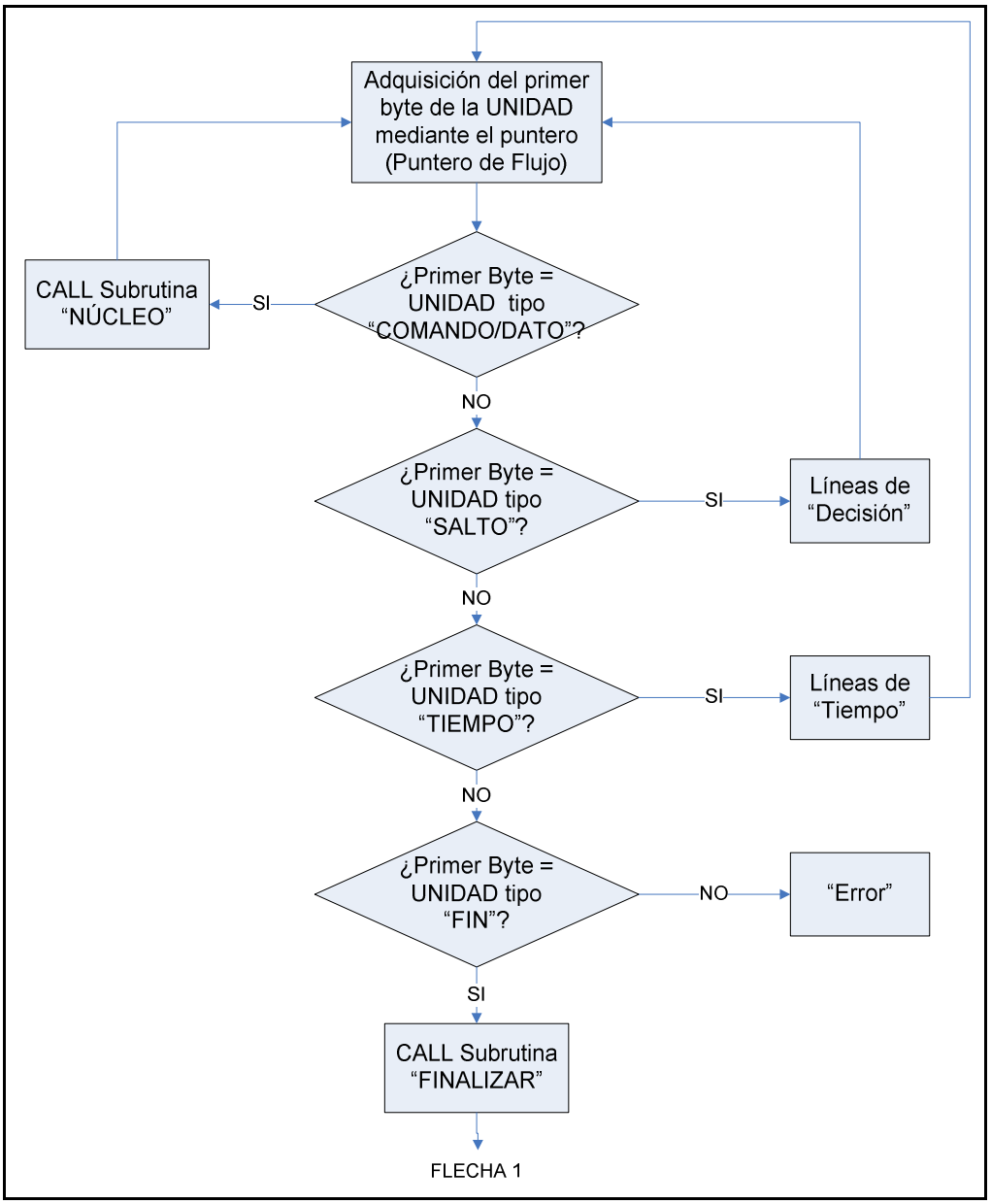


FIGURA 4-20: FLUJOGRAMA DEL BLOQUE "DESCIFRAR Y EJECUTAR CÓDIGO DE FLUJO"

- **Subrutina "NÚCLEO"**

Esta es la subrutina en donde se emite el comando o datos sincronizados con la señal de reloj. Otras funciones de esta subrutina son:

- Decidir si la UNIDAD tiene datos asociados con el comando (b7 del Byte 1).
- Decidir si la UNIDAD va a ser una operación de Escritura (0L) o lectura (1L) sobre el dispositivo a programar (b6 del Byte 1).
- Decidir si la UNIDAD es de Address (b5 del Byte 1).

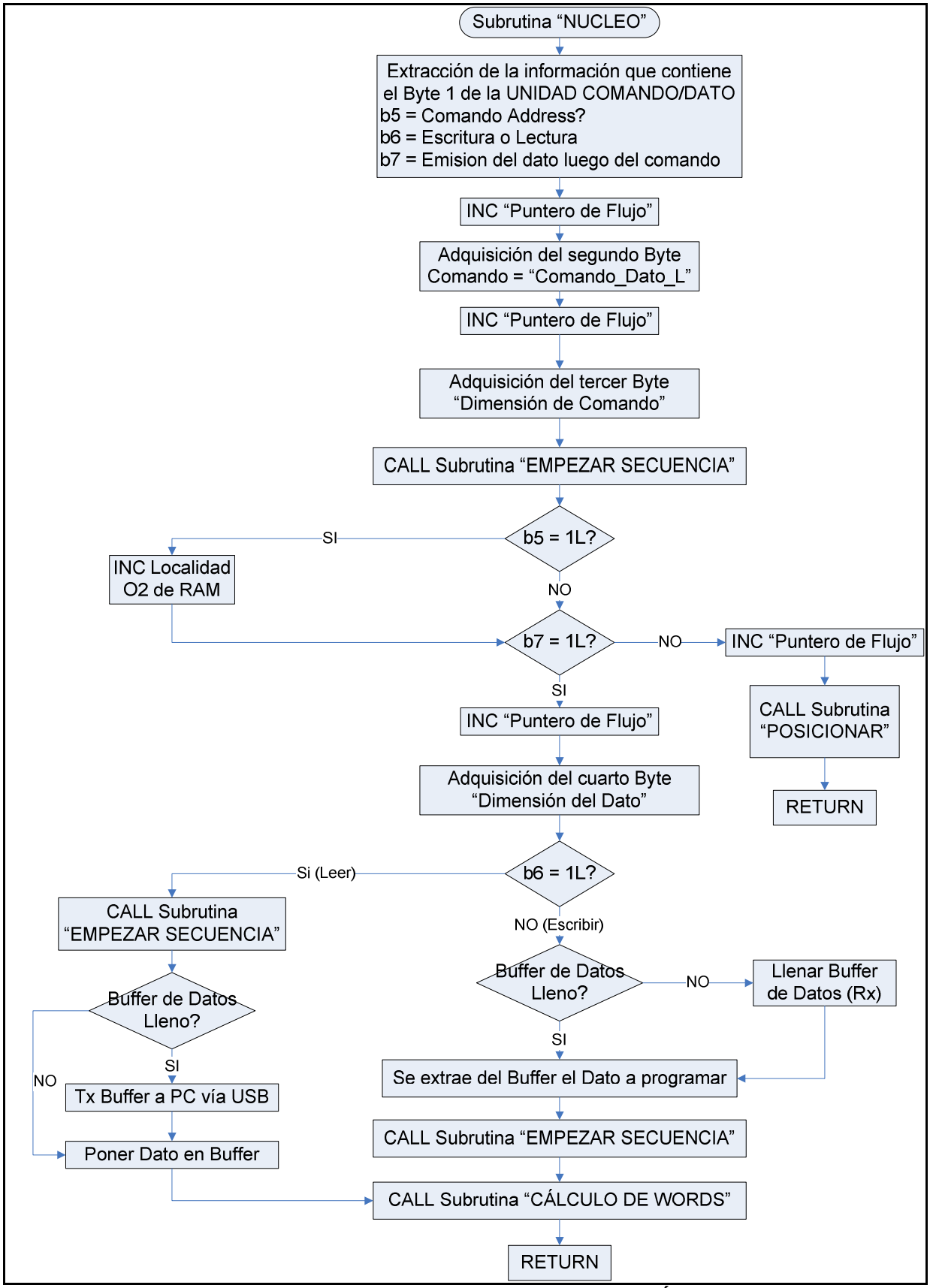


FIGURA 4-21: FLUJOGRAMA DE LA SUBRUTINA “NÚCLEO”

- **Subrutina “EMPEZAR SECUENCIA”**

En este punto, con la información de un Comando o Dato, que está almacenado en los registros COMANDO_DATO_L y COMANDO_DATO_H y su respectiva dimensión (registro “DIMENSIÓN”), se lo emite en forma serial por el pin RA5 del dispositivo USB PIC 16C745 cuando se trate de Escritura. En el caso de tratarse de Lectura, se hace la adquisición de cada uno de los bits del dato a través del pin RA3, y el resultado se lo va almacenando en los registros COMANDO_DATO_L y COMANDO_DATO_H.

Para generar la señal de Reloj, se utiliza el Timer 1 y se la emite a través del pin RC1 del PIC 16C745; además se habilita la interrupción del TMR1.

Cuando sea el caso de Escritura, se emite un bit cuando el reloj pasa de 0L a 1L y cuando sea Lectura, se adquiere el bit cuando el reloj pasa de 0L a 1L.

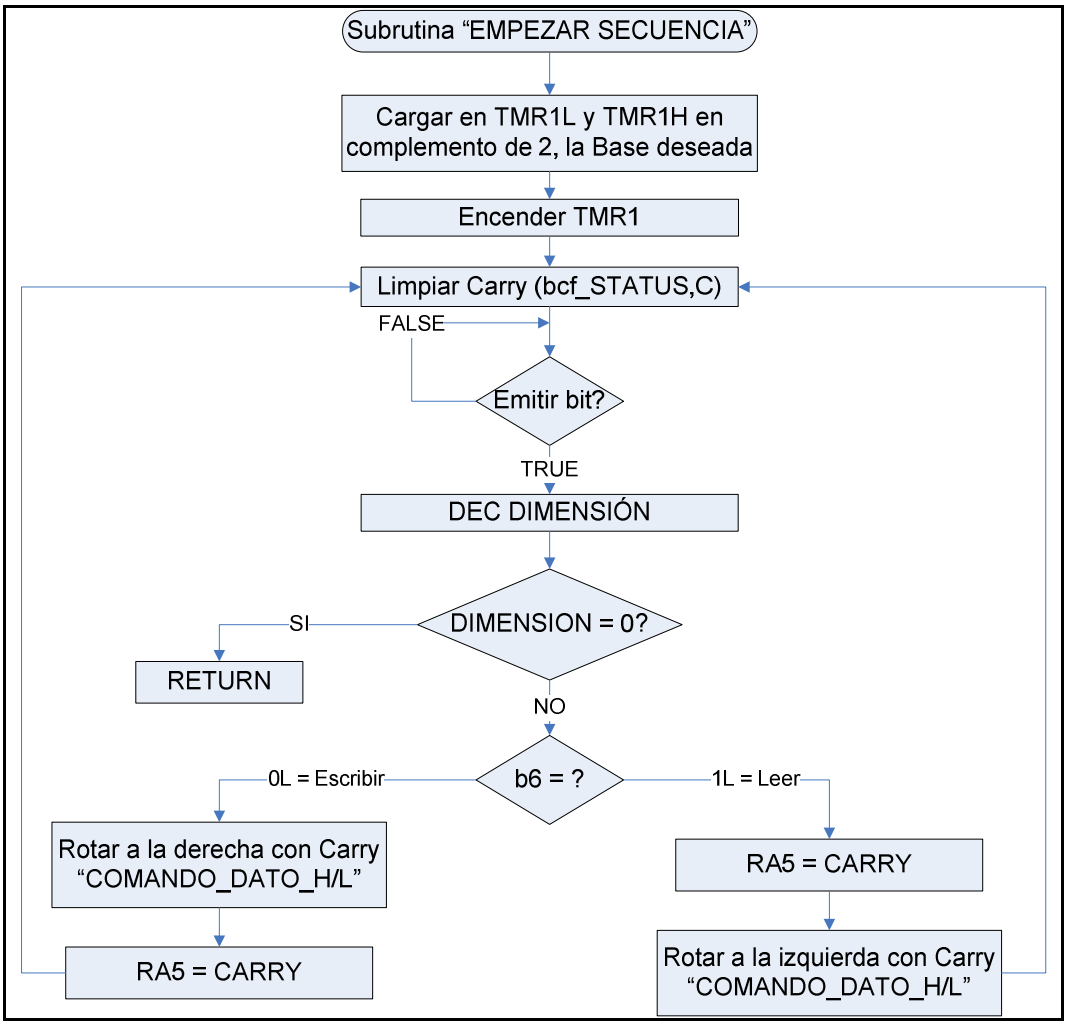


FIGURA 4-22: FLUJOGRAMA DE LA SUBRUTINA “EMPEZAR SECUENCIA”

- **Interrupción Timer 1”**

Las líneas que conforman a la Interrupción por el Timer 1, se encargan de emitir la señal de reloj, y además dan la señal para que la subrutina “EMPEZAR SECUENCIA” escriba o lea un bit de dato.

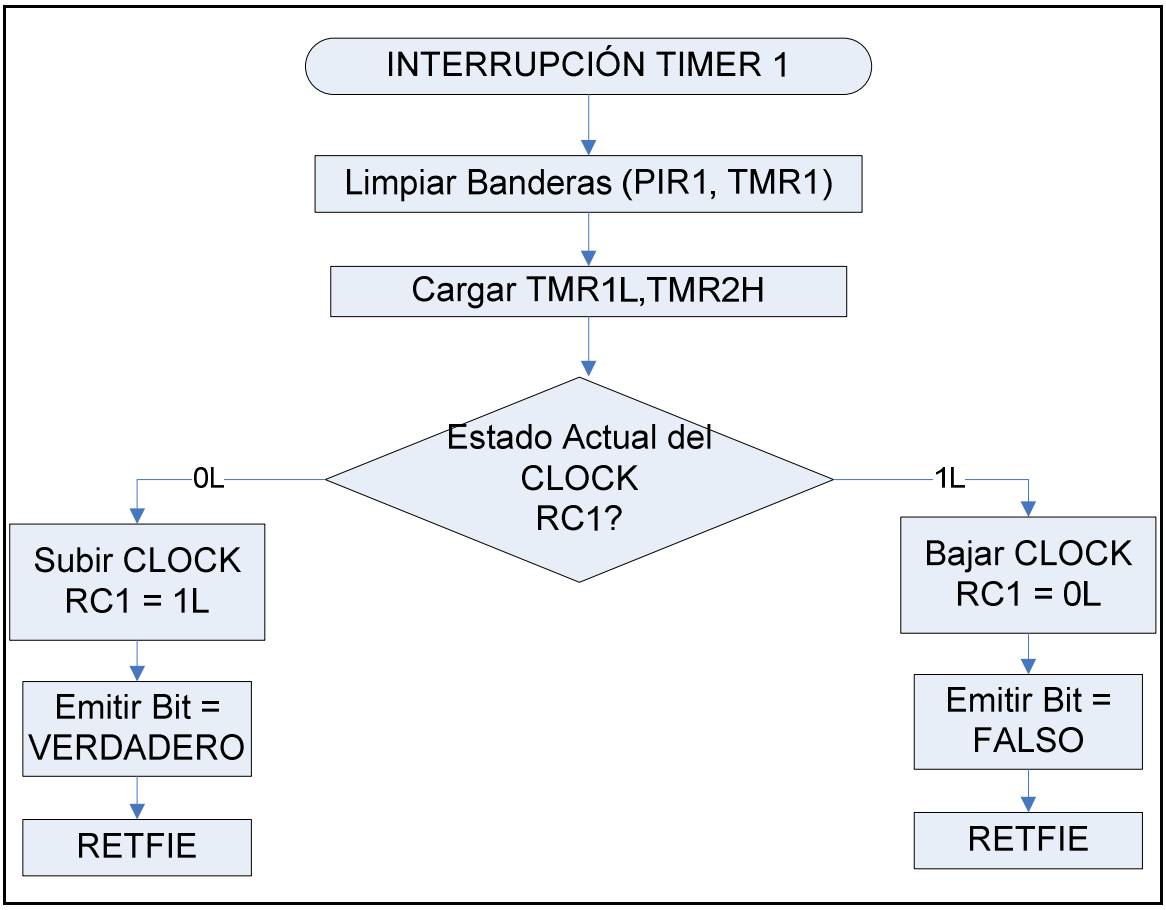


FIGURA 4-23: FLUJOGRAMA DE LA INTERRUPCIÓN TIMER 1

- **Bloque ”Llenar Buffer de Datos Rx”**

Las líneas referentes a “Llenar Buffer de Datos Rx”, hacen la operación de adquirir datos desde el PC. Esto se hace debido a que el dispositivo USB cuando está dentro de un proceso de escritura necesita los datos que van a ser programados, estos datos deben ser transferidos desde el PC en una forma discreta (de 16 bytes en 16 bytes) y almacenados en memoria RAM del PIC USB, por lo tanto cuando se necesite estos datos, se revisará las respectivas localidades de memoria, y en caso de estar vacías se solicitará un nuevo paquete de datos. A este espacio de memoria se lo denomina PILA DE PALABRAS.

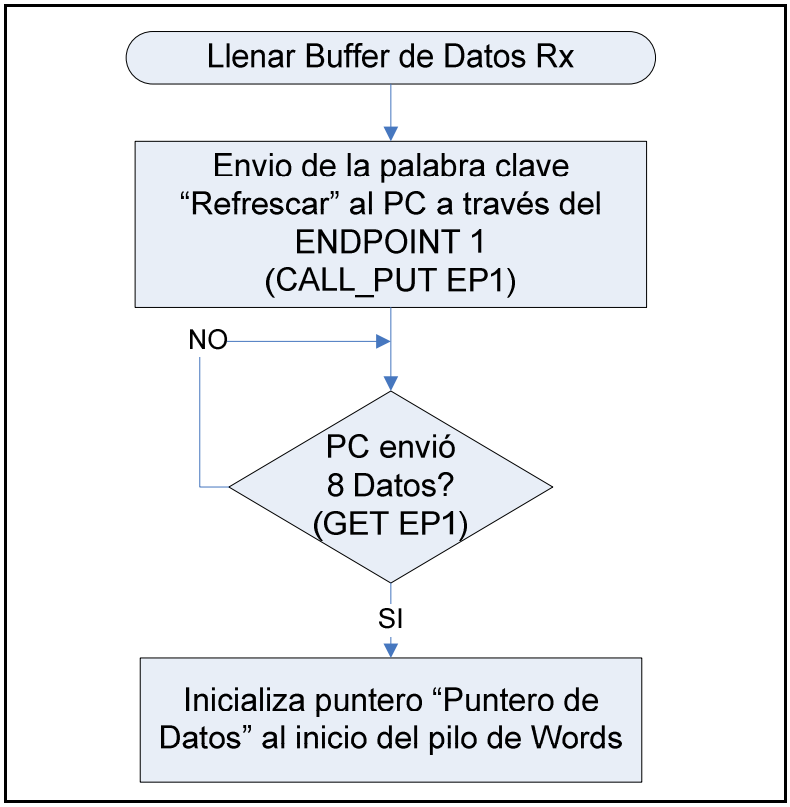


FIGURA 4-24: FLUJOGRAMA DEL BLOQUE "LLENAR BUFFER DE DATOS Rx"

La PILA DE PALABRAS corresponde a un espacio de Memoria RAM del PIC 16C745, específicamente éste se inicia en la posición 0X40 y termina en la 0X4F, abarcando así los 16 bytes, y cada dato está contenido en 2 bytes;

Vía USB se provee los datos utilizando la Transferencia por Interrupción, por tratarse, junto con la Transferencia de Control, en la única forma de transmitir Datos, para dispositivos de Baja Velocidad **[1].**

La subrutina GET EP1, es provista por el Fabricante Microchip.

- **Bloque "Tx Buffer PC vía USB"**

Las líneas "Tx Buffer PC vía USB" son las encargadas de enviar una PILA DE PALABRAS de 16 bytes leídos, a continuación se encuentra la secuencia seguida:

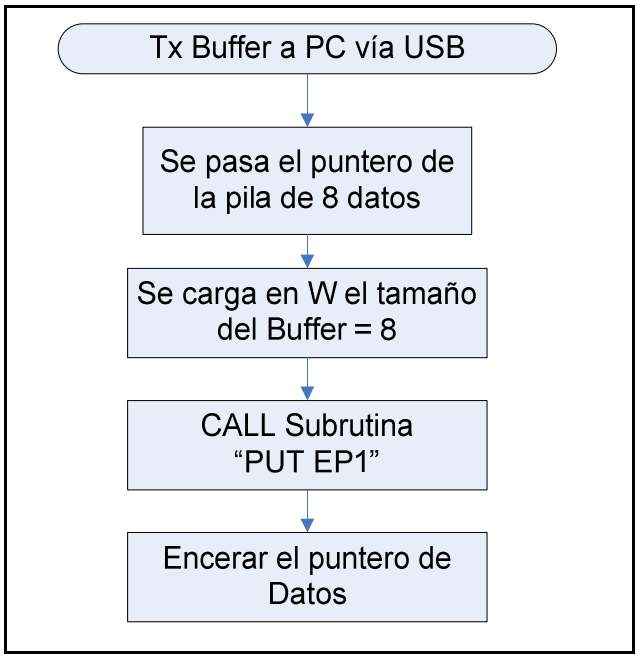


FIGURA 4-25: FLUJOGRAMA DEL BLOQUE "Tx BUFFER PC VÍA USB"

La subrutina PUT EP1, usa el Endpoint 1 IN para enviar los 8 datos leídos. Esta subrutina provee el Fabricante Microchip.

- **Subrutina "POSICIONAR"**

Esta subrutina es la encargada de ubicar el puntero del flujo en la posición que indica los últimos bytes de cada UNIDAD; esta subrutina viene a ser las flechas que se avistan en el flujograma de la Figura 4-6, y que unen una UNIDAD con otra.

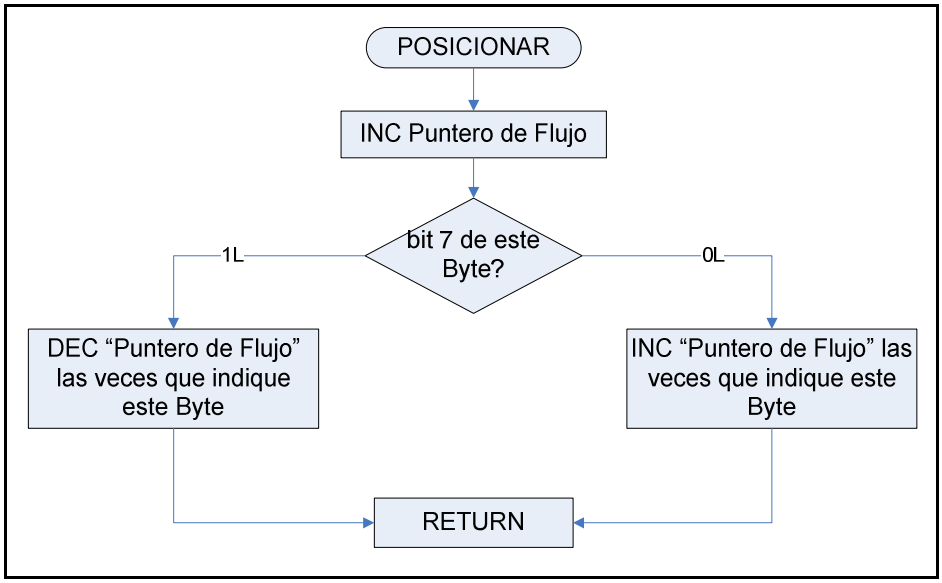


FIGURA 4-26: FLUJOGRAMA DE LA SUBRUTINA "POSICIONAR"

- **Subrutina “CALCULO WORDS”**

Es la encargada de incrementar en uno la posición de RAM que almacenan el número de datos procesados, pudiendo ser éstas tanto las de Escritura como las de Lectura.

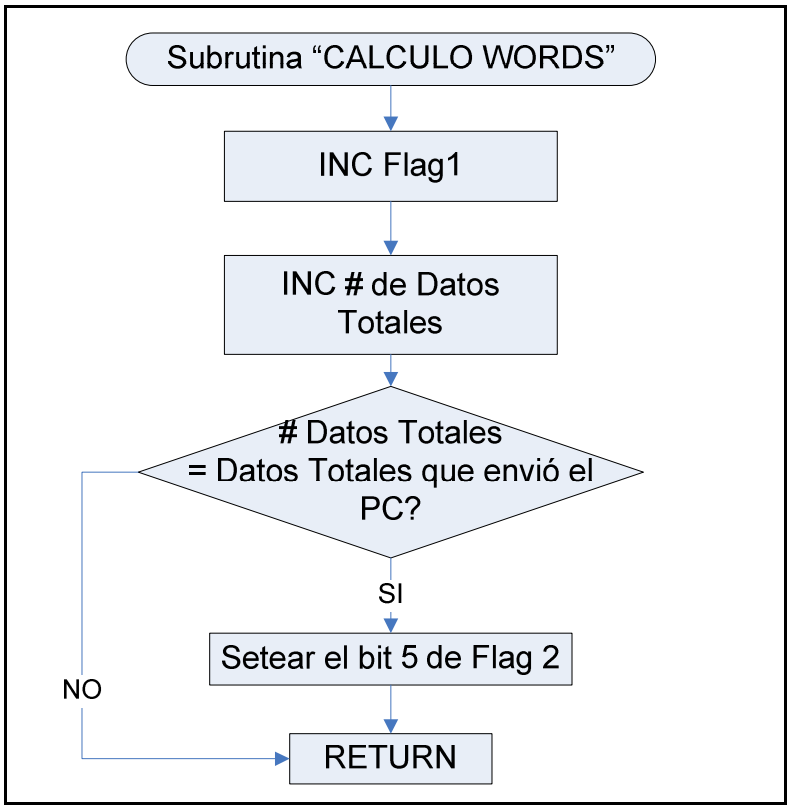


FIGURA 4-27: FLUJOGRAMA DE LA SUBRUTINA “CÁLCULO WORDS”

En Flag 1 está el registro de datos parciales procesadas.

Flag 2 es una bandera que se pone en alto cuando el número total de datos procesados coincide con el número total de datos que envió el PC a procesar al PIC 16C745.

- **Subrutina “FINALIZAR”**

Esta subrutina es la encargada de inicializar todas las variables, además de apagar los pines VDD, Vpp y el módulo generador del PWM (conversor elevador), apaga también a los leds visualizadores.

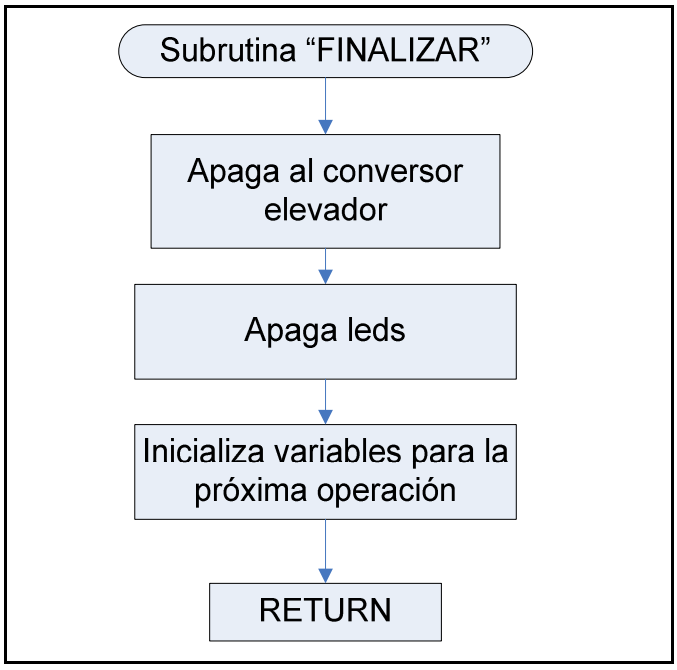


FIGURA 4-28: FLUJOGRAMA DE LA SUBRUTINA “FINALIZAR”

Se entabla la comunicación entre el PC y el microcontrolador PIC, únicamente cuando el PC lo solicita; en todo momento está sincronizada la trasferencia de datos, en ningún momento el microcontrolador PIC puede empezar una comunicación.

4.2.1.2 Líneas de Operación para ATMEL

El modo de programar un dispositivo ATMEL (familias 51, 52, 55) difiere con respecto a un microcontrolador PIC. Estos microcontroladores ATMEL se programan en forma paralela, y es justamente por la limitación de pines que tiene el PIC 16C745, que se ha implementado el direccionamiento de la Memoria del dispositivo a programarse, mediante un par de CI contadores 393, que se conectan en cascada, para obtener así los pines necesarios para programar 2, 8 y 20 K de memoria.

Los pines de control de estos CI corresponde a:

CLR Encera al dispositivo (pone todas sus salidas en nivel bajo 0L)

CLOCK Es la señal de alimentación del reloj. Con cada flanco positivo los contadores se incrementan en uno.

Otra diferencia es respecto al tamaño de cada uno de los datos de la Memoria de Programa. En el caso de ATMEL éste es de 8 bits (1 byte), Para microcontroladores PIC esta longitud es variable y depende de la gama a la cual pertenezca el dispositivo.

El procedimiento seguido para la programación de un microcontrolador ATMEL se detalla a continuación:

En primer lugar, se averigua que operación se desea realizar sobre el microcontrolador ATMEL: leer, borrar, escribir.

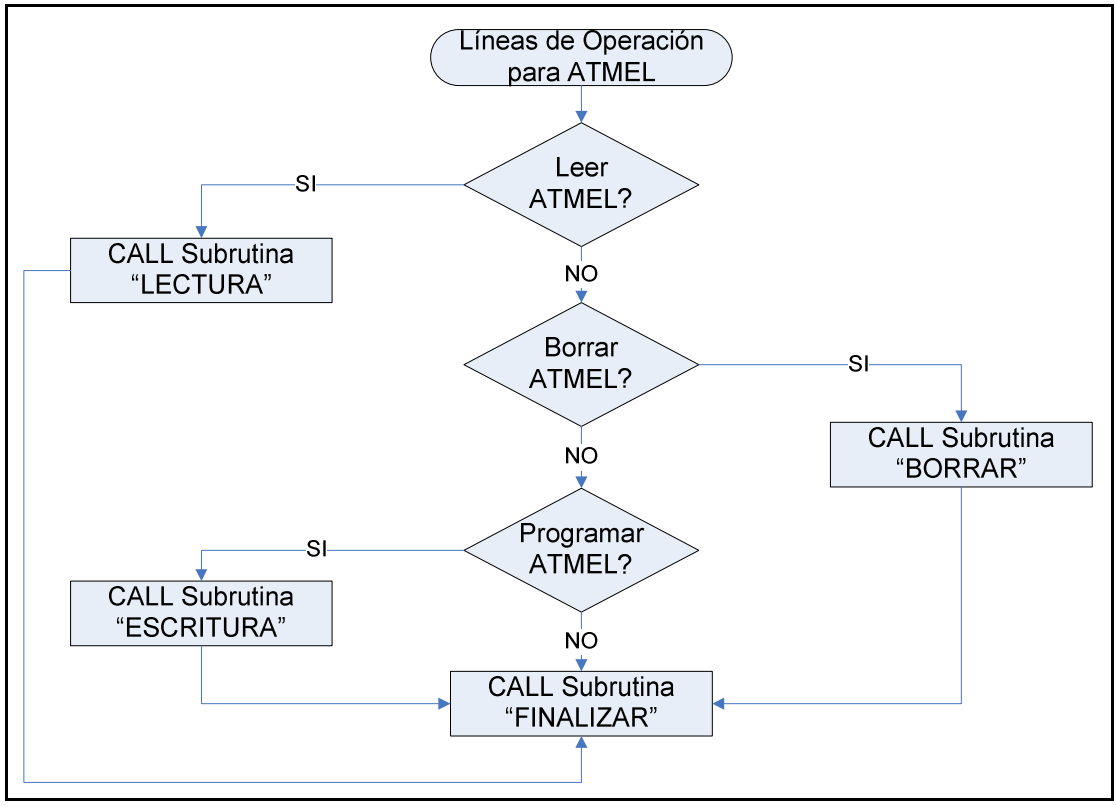


FIGURA 4-29: FLUJOGRAMA PARA AVERIGUAR OPERACIÓN EN MICROCONTROLADOR ATMEL

En la Tabla 3-4 se muestra la correspondencia entre los pines del PIC USB y los pines necesarios para programar el microcontrolador ATMEL, por lo que se dispone el Puerto A como salida sea cual sea la operación a realizar.

Para el Puerto B del PIC USB se lo debe disponer como entrada para lectura y como salida para escritura.

- **Operación "LECTURA"**

Se sigue la secuencia, que indica el Fabricante ATMEL, para sus dispositivos en sus Hojas de Especificaciones.

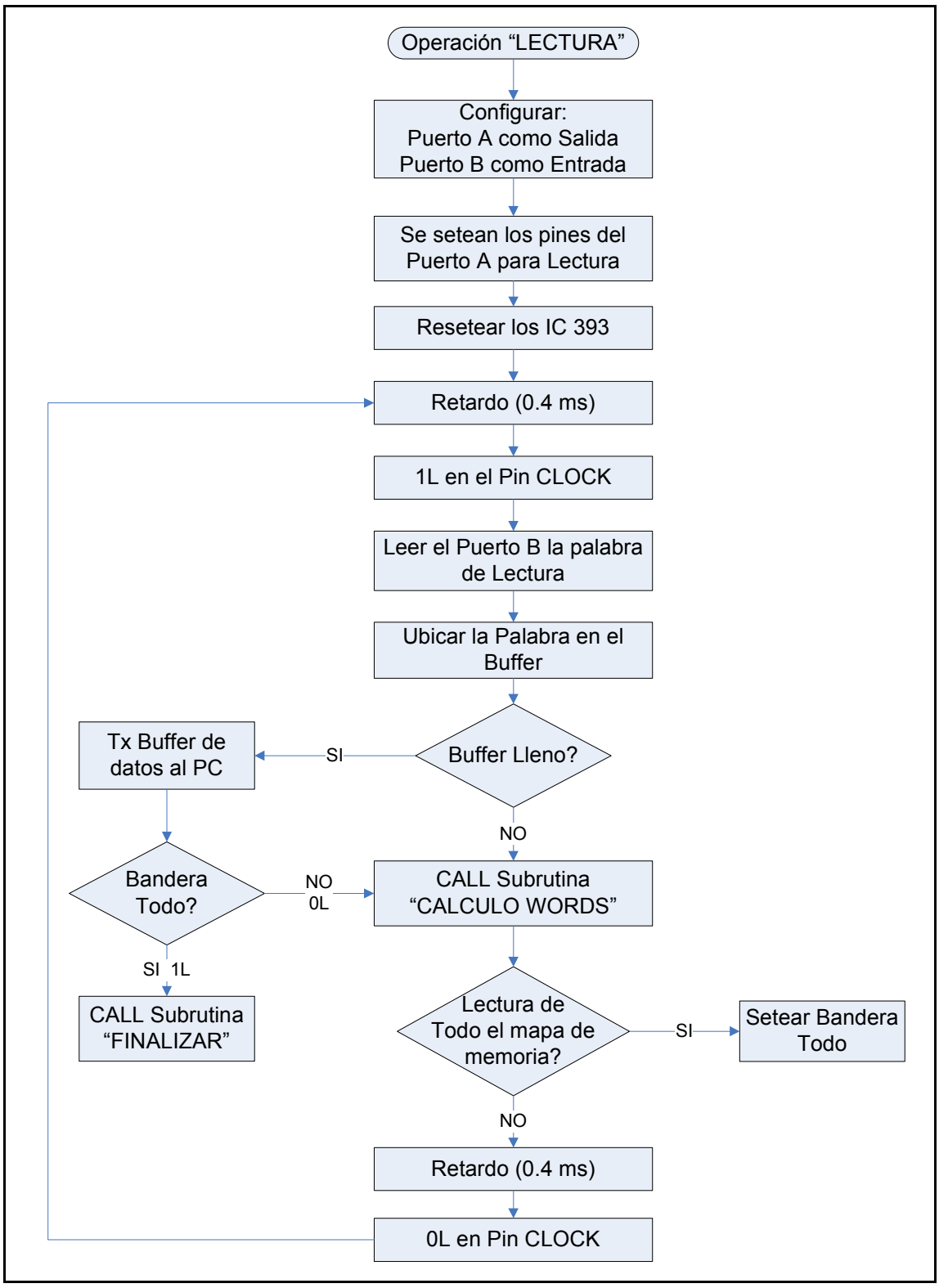


FIGURA 4-30: FLUJOGRAMA DE LA OPERACIÓN "LECTURA"

En estos flujogramas se llama a las subrutinas “Tx Buffer de datos al PC”y “Finalizar” que son las mismas utilizadas para los dispositivos PIC.

- **Operación “BORRAR”**

Aquí no se usa el Puerto B, por otro lado se necesita generar los 12 V para el pin Vpp, que sube de 5 V a 12 V **[2]**. La Figura que corresponde al Borrado es la siguiente:

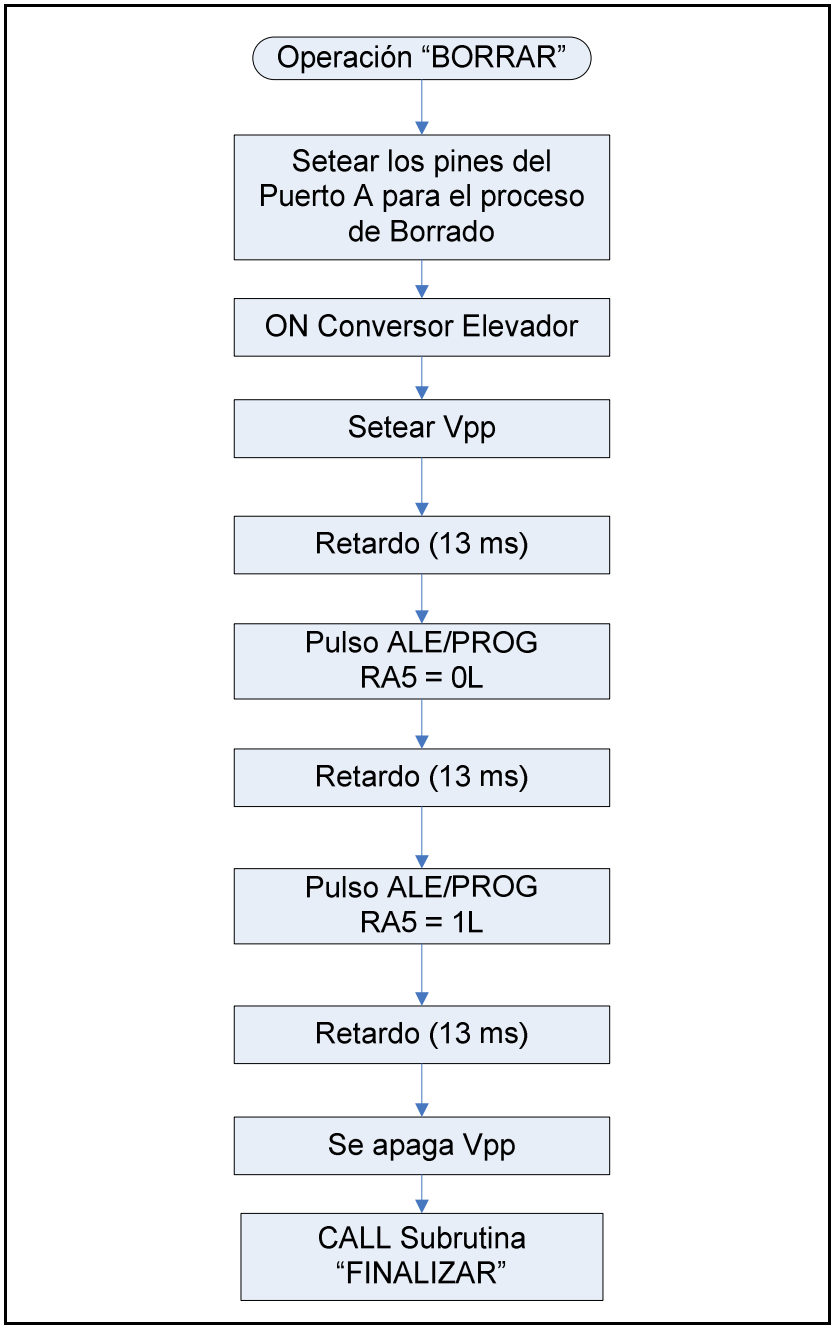


FIGURA 4-31: FLUJOGRAMA DE LA OPERACIÓN “BORRAR”

- **Operación “ESCRIBIR”**

El Puerto B se dispone como salida. Se genera 12 V para el pin Vpp, que sube de 5 V a 12 V **[2].**

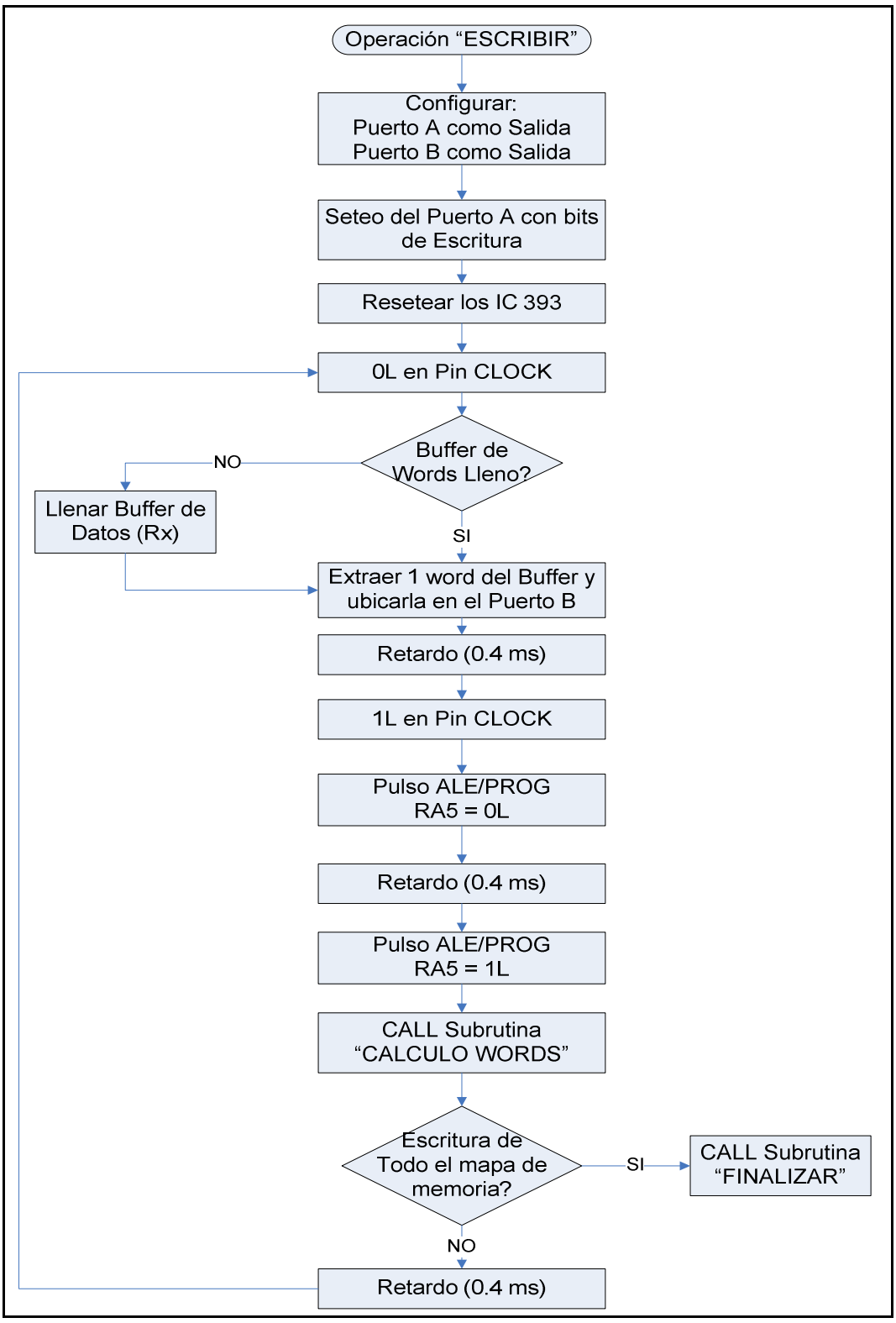


FIGURA 4-32: FLUJOGRAMA DE LA OPERACIÓN “ESCRIBIR”

4.2.2 DESARROLLO DEL SOFTWARE PARA EL PROGRAMADOR “EPNprog”

El software que corre sobre el PC, se ha desarrollado usando la herramienta Visual Basic. La razón de escogerla, se debe a su facilidad en el modo de programación y por disponer del objeto HID, que corresponde a un Active X dedicado para la comunicación USB, este Active X fué desarrollado por el fabricante Microchip.

Las principales partes del presente software consisten en:

- Adquisición y visualización de los archivos hexadecimales, que son compilados por alguna otra aplicación.
- Elección de la familia de dispositivos: Elección del dispositivo.
- Elección de la operación a ejecutar sobre el dispositivo elegido.

4.2.2.1 Adquisición y Visualización de un Archivo HEX

Cuando se trata de la Escritura de un dispositivo, es necesario abrir un archivo Hex. Este archivo se lo abre en forma binaria, con el fin de leer cada uno de los bytes que lo conforman. Después de escoger cada uno de los bytes válidos, se los almacena en una matriz tipo byte, el tamaño de esta matriz se la dimensiona conforme se ejecuta el programa, porque es aquí donde se puede conocer el número de bytes que conforman al archivo Hex.

Luego de tener cargada la matriz de datos, se la visualiza en una grilla MSHFlexGrid, de Visual Basic.

Por lo tanto estas líneas de programa, proporcionan:

- El número de bytes que conforman el archivo Hex
- Los datos a programar (matriz de datos)

El flujograma relativo a este proceso es:

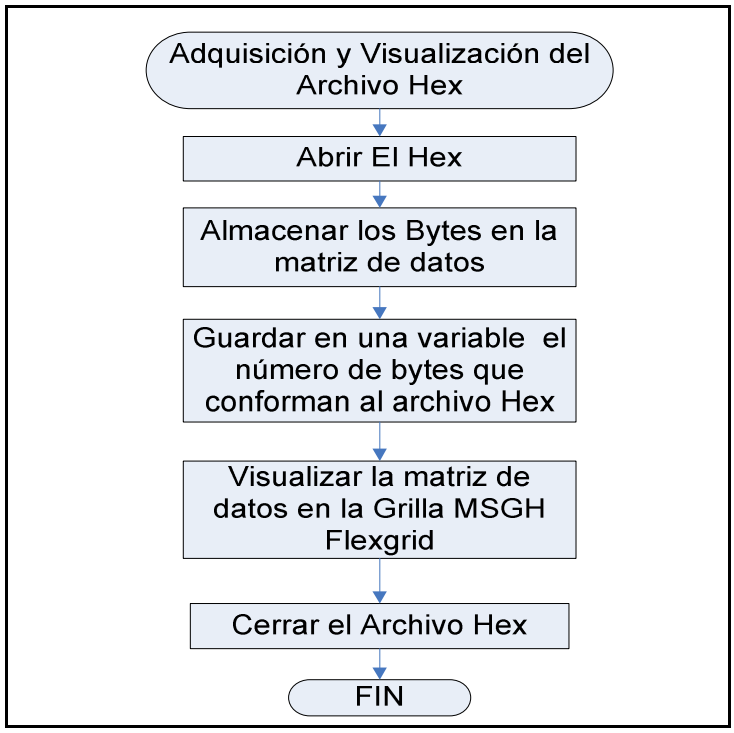


FIGURA 4-33: FLUJOGRAMA DE ADQUISICIÓN Y VISUALIZACIÓN DEL ARCHIVO HEX.

La adquisición de un archivo Hex, es independiente del dispositivo o familia que se encuentre actualmente seleccionada, debido a que el formato es INTEL, lo que si se debe tener presente es que: para un dispositivo PIC, un dato corresponde a la unión de 2 bytes, mientras que para un dispositivo ATMEL, un dato corresponde a 1 byte.

En el caso de tratarse de un proceso de Lectura, se ubicarán los bytes leídos en la misma matriz de datos y posteriormente se los visualiza en la grilla.

La lectura corresponde a la totalidad del mapa de memoria, en este proceso, la variable que guarda el tamaño hexadecimal, contendrá al número que corresponde a la totalidad de la memoria.

4.2.2.2 Elección de la Familia y Dispositivo

Al cargarse la aplicación en el PC, se deben inicializar por defecto parámetros relativos a un dispositivo, como son:

- Tamaño de Memoria
- Palabra de Configuración
- Espacio de Memoria EEPROM (en caso de disponer)

Todos estos parámetros son necesarios para que el programador esté dispuesto para visualizar el mapa de memoria, y tener disponibles los flujogramas de Lectura, Escritura, Borrado y proceso de Verificación.

El dispositivo cargado por defecto, corresponde al último dispositivo que se utilizó en la anterior llamada a la aplicación.

4.2.2.2.1 Para microcontroladores PIC

En cuanto tiene que ver a los flujogramas de Lectura, Escritura y Borrado, para microcontroladores PIC, éstos residen de un archivo de Access (BASE.mdb), y la organización por familia se representa en la Figura 4-34.

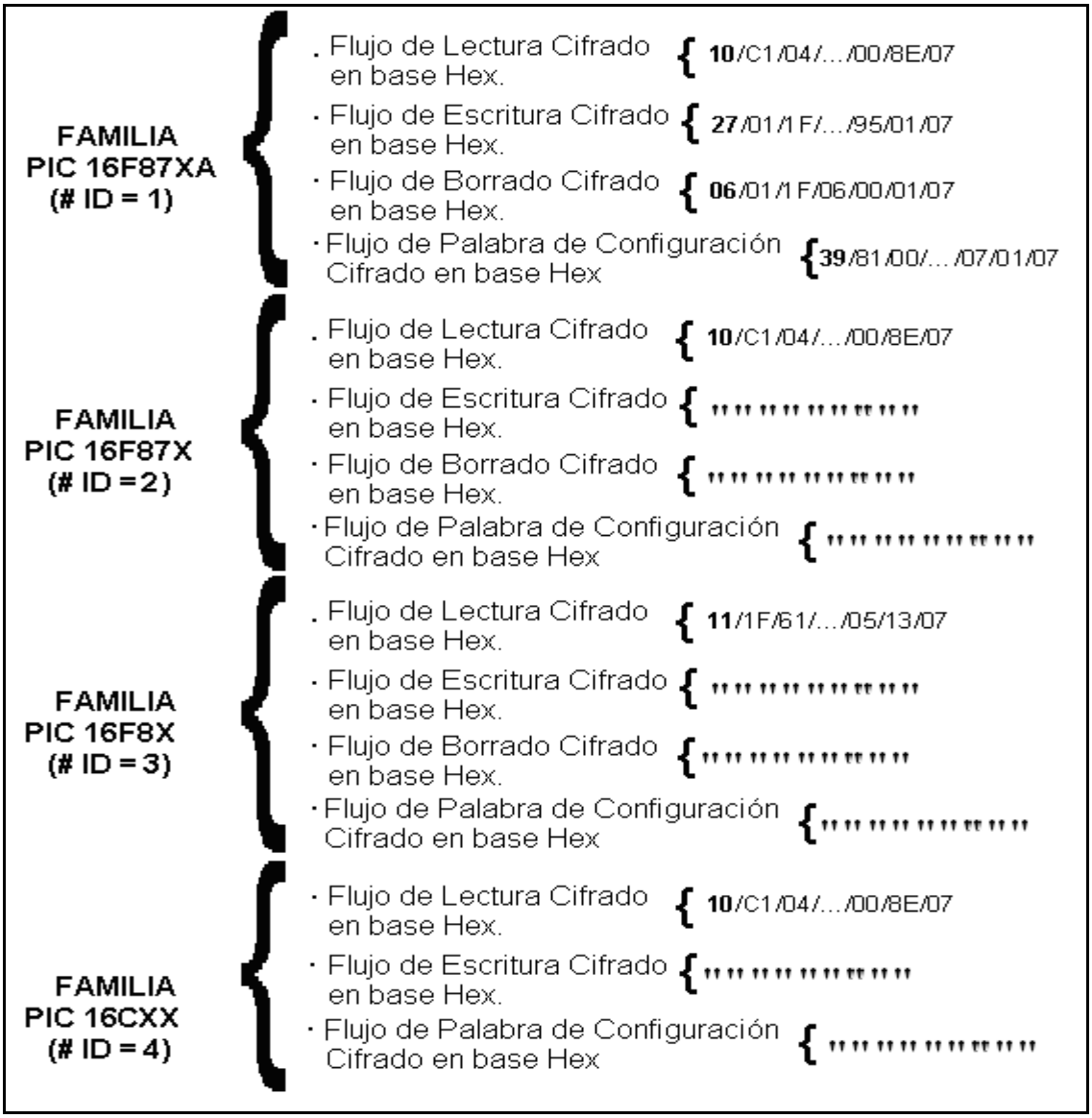


FIGURA 4-34: FLUJO CIFRADO EN BASE HEX DE LAS DIFERENTES FAMILIAS PIC

El primer número hexadecimal del flujo cifrado de la Figura 4-34, representa el número total de cantidades hexadecimales que representan al flujo cifrado.

En otra tabla que pertenece a la misma Base, se encuentran cada uno de los dispositivos PIC que soporta en la actualidad el programador “EPNprog” y la estructura se muestra en la Tabla 4-4.

DISPOSITIVO	TAMAÑO Mem. Prog.	TAMAÑO Mem. EEPROM	# I D
PIC 16F870A/71A/72A	2048	64	1
PIC 16F873A/74A	4096	128	
PIC 16F876A/77A	8192	256	
PIC 16F870/71/72	2048	64	2
PIC 16F873/74	4096	128	
PIC 16F876/77	8192	256	
PIC 16F83	512	64	3
PIC 16F84	1024	64	
PIC 16F84A	1024	64	
PIC16C61/71	1024	-	4
PIC16C72	2048	-	
PIC16C63	4096	-	
PIC16C66/67/76/77	8192	-	

TABLA 4-4: ESTRUCTURA DE LOS DISPOSITIVOS PIC DENTRO DEL PROGRAMADOR

Cuando se escoge un dispositivo PIC de la lista desplegable, da como resultado los siguientes eventos:

- Nombre del dispositivo
- Tamaño de La Memoria de Programa
- Tamaño de La Memoria EEPROM (cuando se disponga)
- Número ID.

A estos datos se les da el siguiente uso:
- Nombre del Dispositivo, para ubicarlo en las ventanas de información tal como la Figura 4-35.

FIGURA 4-35: VENTANA DE OPERACIÓN DE BORRADO DEL PROGRAMADOR.

- Tanto el Tamaño de la Memoria de Programa, como la Memoria de Datos EEPROM, se usan para dimensionar: la matriz de datos, la grilla MSHFlexGrid, para pasarlos de argumento en la comunicación cuando se esté en proceso de Lectura, para dimensionar la barra de progreso en la operación de Lectura.

- El Número ID, es el número que identifica a un microcontrolador PIC como elemento de alguna familia. Este número es necesario para encontrar los flujos (Escribir, Borrar, Leer) que pertenecen a esta familia.

Todos los dispositivos PIC que pertenecen a una determinada familia, van ha compartir el mismo procedimiento de Lectura, Escritura y Borrado.

4.2.2.2.2 Para microcontroladores ATMEL

En el caso de dispositivos ATMEL, no se transfiere ningún flujograma. Los flujogramas ya residen dentro del Firmware, lo que se pasa es una clave que obliga al Firmware a realizar Lectura, Escritura y Borrado.

El software contiene la información relativa al microcontrolador ATMEL como constantes, con lo que el usuario no puede modificar los parámetros para un microcontrolador ATMEL.

El flujograma de Elección de dispositivo está representado en la Figura 4-36.

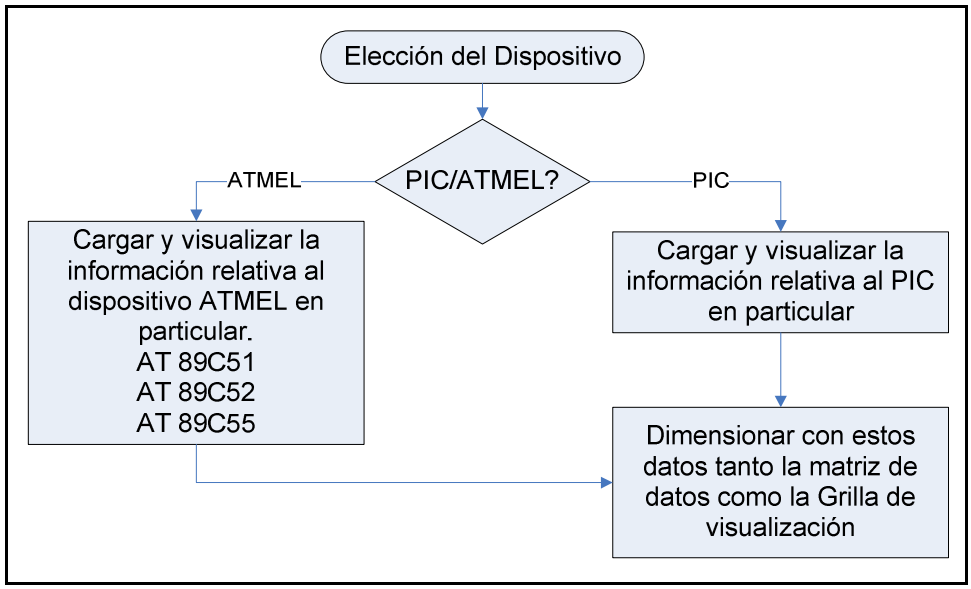


FIGURA 4-36: FLUJOGRAMA DE ELECCIÓN DEL DISPOSITIVO

4.2.2.3 Elección de la Operación a Ejecutar sobre el Dispositivo

Luego de tener identificado al dispositivo seleccionado, lo que resta a esperar es que el usuario decida realizar, alguna de las cuatro operaciones.

Para cualquier operación se abre la comunicación USB a través del Endpoint 1 OUT, que es de tamaño de 8 bytes, y es a baja velocidad, con una taza de transferencia de 6400 bits/s (Transferencia por Interrupción; USB Firmware User's Guide de Microchip, pág. 3).

El procedimiento de intercambio de datos entre PC y PIC USB es el siguiente:

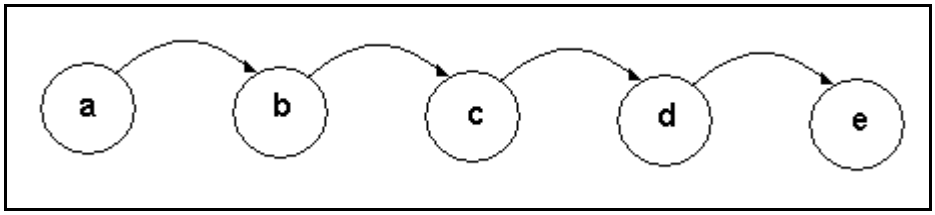


FIGURA 4-37: PROCEDIMIENTO DE INTERCAMBIO DE DATOS ENTRE PC Y PIC

a) Tamaño del Hex.
b) Información de Elección (PIC o ATMEL).
c) Tamaño del Flujo.
d) Bytes del Flujo en sí.
e) Byte Clave que ordena al PIC 16C745 empezar a operar.

La transferencia en paquetes de 8 bytes que transmite el PC lleva la siguiente información:

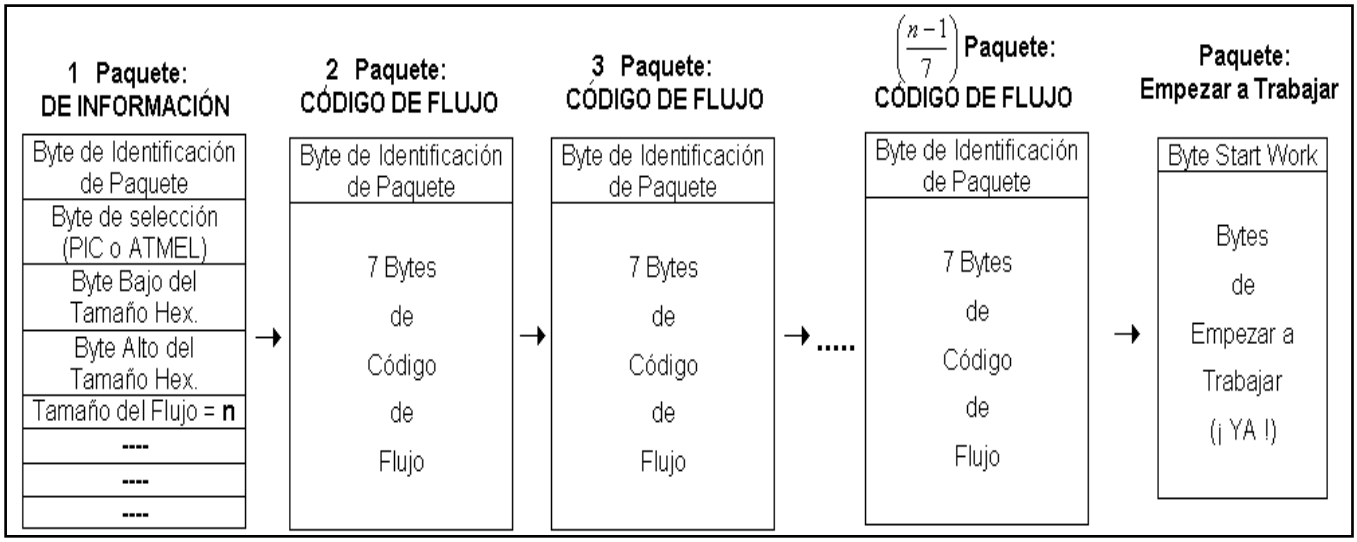


FIGURA 4-38: PAQUETES QUE TRANSMITE EL PC AL PIC 16C745.

Como se aprecia, cada uno de estos paquetes tiene su primer byte que lo identifica. A continuación se presentan las claves para la identificación de un paquete (Primer Byte):

- Si Primer Byte = 26 entonces, es un Paquete de Información.
- Si Primer Byte = 29 entonces, es un Paquete que contiene 7 bytes de Código de Flujo.
- Si Primer Byte = 20 entonces, es un Paquete Inicio de Operación.

Al recibir el PIC 16C745 este conjunto de paquetes, los procesa según el flujograma de la Figura 4-39.

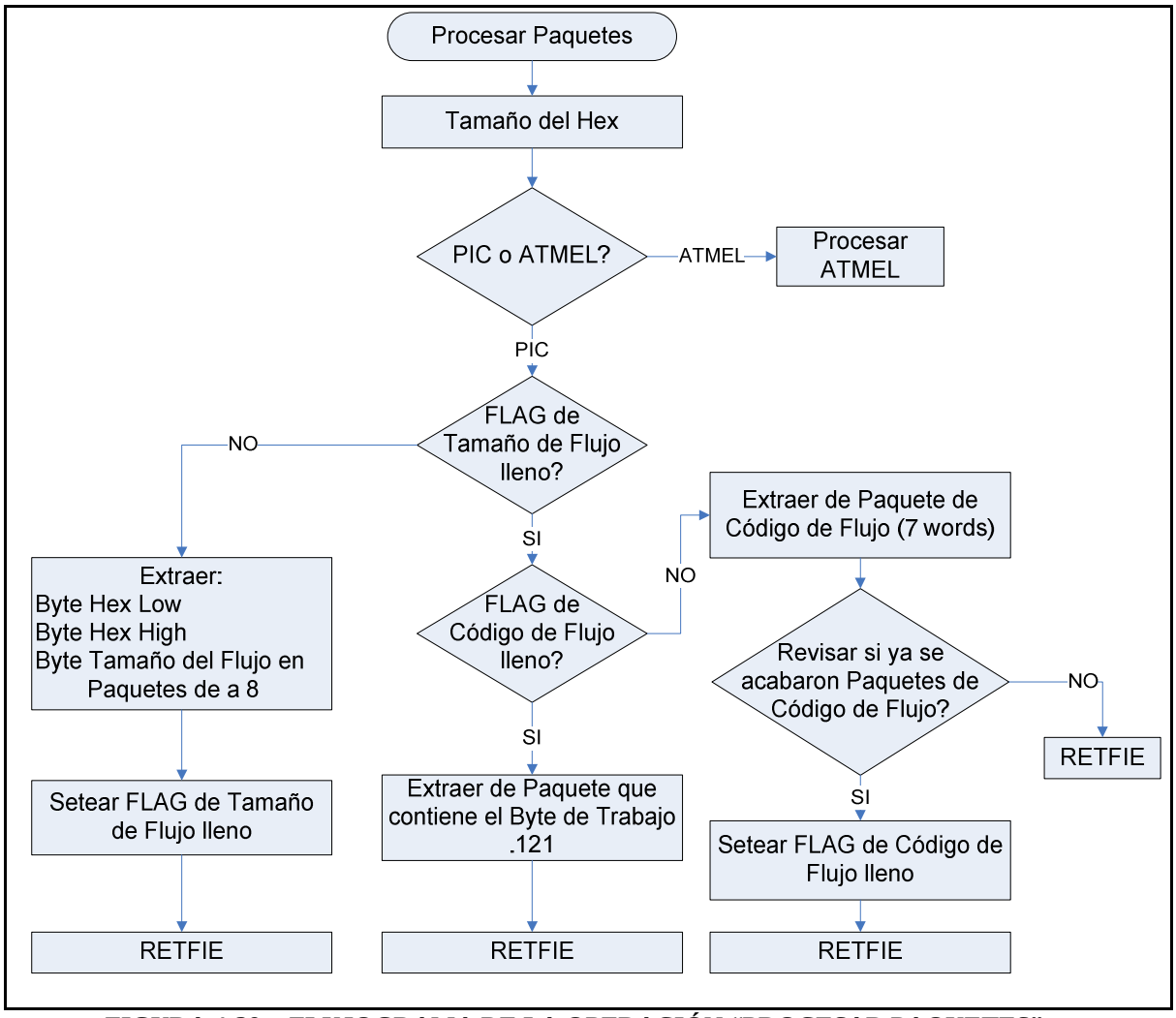


FIGURA 4-39: FLUJOGRAMA DE LA OPERACIÓN "PROCESAR PAQUETES"

Debido a que la comunicación USB es de tipo Poolling (quiere decir que el PC es el único encargado de comenzar la comunicación), el proceso de intercambio de Datos está sincronizado. El PC enviará datos y esperará dentro de un lazo, los datos que transmita el microcontrolador PIC, cuando corresponda.

4.2.2.3.1 Operación de Escritura

Se llenan los campos del primer paquete con el tamaño del Hex, que se adquiere cuando se realizó la "Adquisición y Visualización del Archivo Hex".

En el caso del microcontrolador ATMEL se llena el campo "Tamaño del Flujo"con una constante igual a 3, esto quiere decir que en el siguiente paquete se envía la información relativa a:

- Segundo Byte = Este es la Clave de la operación del ATMEL.

CLAVE	OPERACIÓN
191	Leer ATMEL
234	Borrar ATMEL
197	Escribir ATMEL

TABLA 4-5: CLAVE QUE DA LA OPERACIÓN DEL ATMEL

- Tercer Byte = Líneas de control, que se asignará al Puerto A.
- Cuarto Byte = Es la combinación de Loock Bits (En la Escritura ATMEL).

Una vez que se ha enviado toda esta información, el Software se queda esperando el número clave (Refrescar = 85H), para empezar a descargar la matriz de datos, en paquetes de 8 bytes.

4.2.2.3.2 Operación de Lectura

El tamaño del Hex en este caso se llena con el número total del mapa de memoria.

En forma inversa de lo que se realizó en la Escritura, aquí se espera cada uno de los paquetes de 8 bytes que envía el PIC 16C745, y los va ubicando en la matriz de datos, para luego ser visualizados.

4.2.2.3.3 Operación de Borrado

Aquí no existen datos involucrados, además de los de inicialización, por lo que no existe una espera por parte del PC para el envío de datos.

CAPÍTULO 5

PRUEBAS Y RESULTADOS

En el presente capítulo se desarrolló todo lo referente al buen funcionamiento del programador tal como el proceso de instalación, uso del software y resultados obtenidos.

Se avaliza el buen funcionamiento del programador con la puesta en marcha de tres proyectos pertenecientes a diferentes laboratorios.

5.1 PRUEBAS RELATIVAS AL PROGRAMADOR

Dentro del conjunto de pruebas y resultados se muestra un ejemplo de las formas de onda capturadas dentro del proceso de programación de un dispositivo PIC 16F83, en la cual se aprecia la señal de reloj sincronizada con la señal de datos. (Figura 5-1).

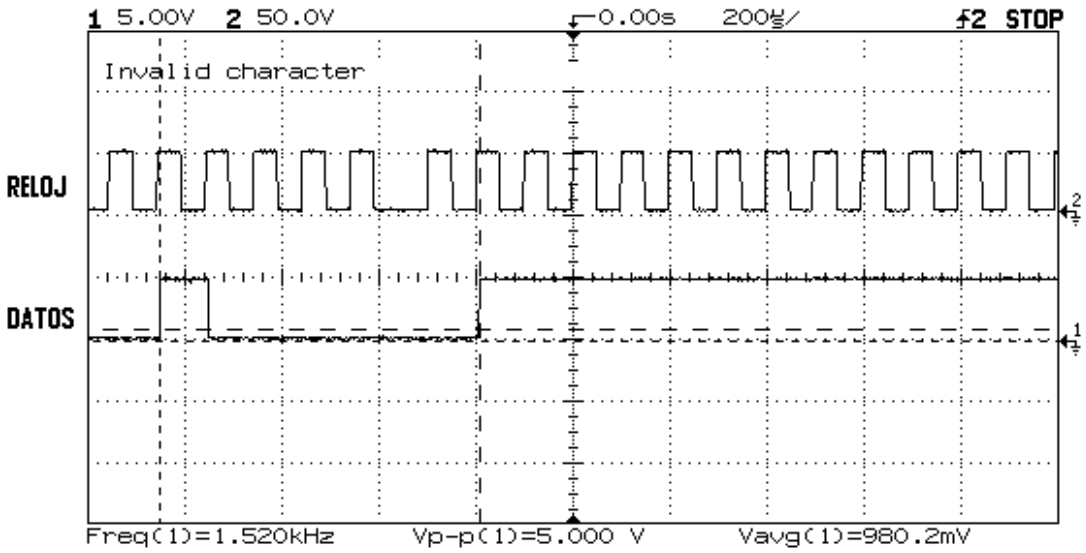


FIGURA 5-1: SEÑALES DE RELOJ Y DATOS DURANTE UNA PROGRAMACIÓN DE UN PIC

Como se aprecia, en la señal de datos ocurre el comando LOAD COMMAND DATA (00010) seguido por su respectiva palabra dato, según fue explicado en el Capítulo 2 del presente proyecto.

Para corroborar el buen funcionamiento del programador, éste ha sido usado en proyectos desarrollados como tarea de los estudiantes en sus respectivas materias, como son Instrumentación Industrial (Carrera de Ingeniería Electrónica y Control) y Sistemas Análogo Digitales (Carrera de Ingeniería Electrónica y Telecomunicaciones). Obteniendo los siguientes resultados:

5.1.1 PROYECTO DE "CONTROL DE POSICIÓN DE MOTORES A PASOS CON REALIMENTACIÓN USANDO SENSORES DE ULTRASONIDO, UTILIZANDO EL PIC 16F877"

En este proyecto se busca el posicionamiento de un ascensor a una determinada altura mediante el control de un par de motores a pasos ubicados en lo alto del ascensor, en cambio para el sensado de la altura instantánea se usan sensores ultrasónicos, el valor actual de la altura se visualiza en un display según la Figura 5-2. En este proyecto se utilizó el microcontrolador PIC 16F877 y cuyo hexadecimal fué grabado con el programador EPNprog, y el resultado fue satisfactorio.

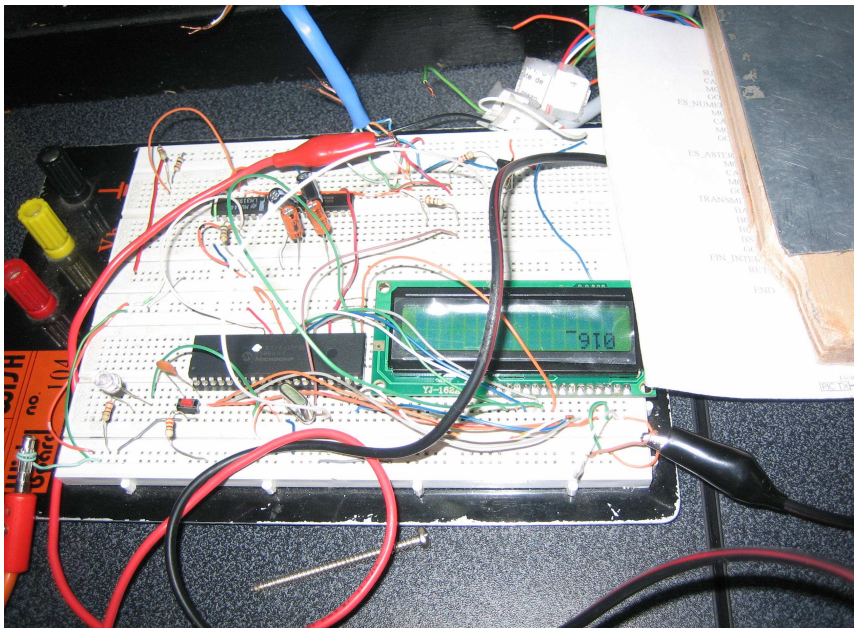


FIGURA 5-2: HARDWARE PERTENECIENTE AL PROYECTO DE CONTROL DE POSICIONAMIENTO.

5.1.2 PROYECTO DE “ANIMACIÓN DE UN ALLOSAURIO MECÁNICO A ESCALA NATURAL, CONTROLADO POR EL PIC 16F877A”

Esta animación comprende el movimiento de la cabeza, cuello, tórax y mandíbulas de un Allosaurio mecánico mediante el control de motores de corriente continua controlados por las señales entregadas por el microcontrolador PIC.

A continuación se muestra la estructura completa del Allosaurio.

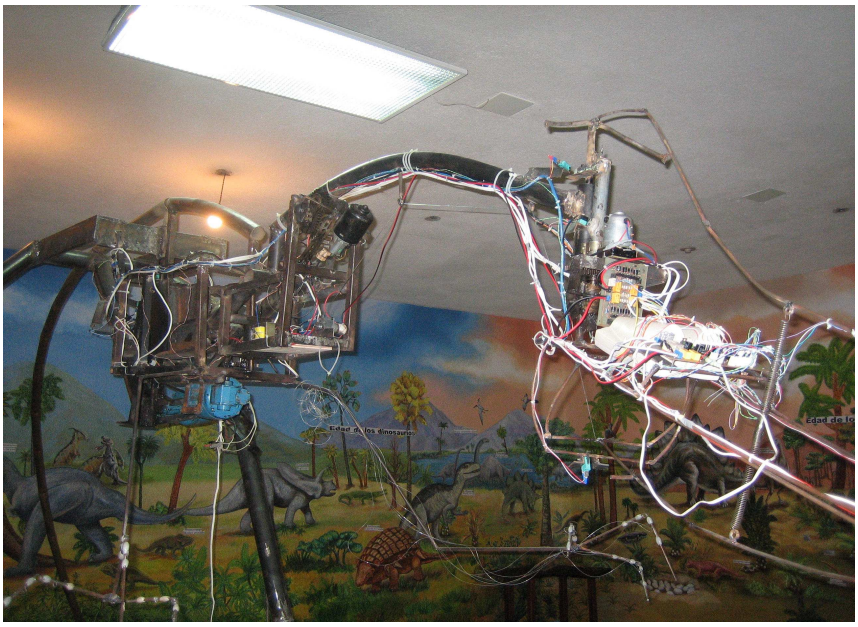


FIGURA 5-3: ESQUELETO DEL ALLOSAURIO

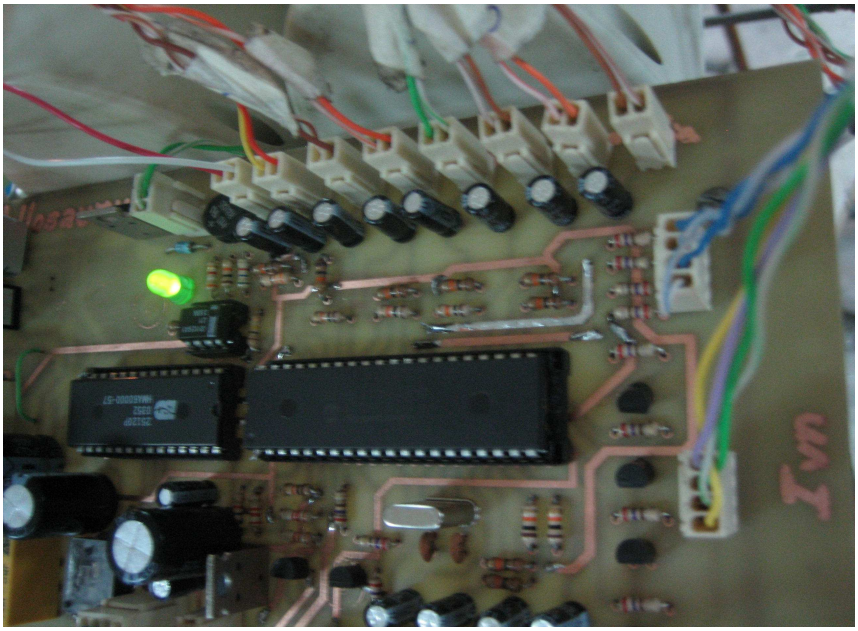


FIGURA 5-4: TARJETA DE CONTROL LOCALIZADA DENTRO DE LA CABEZA DEL ALLOSAURIO

5.1.3 PROYECTO DE "CONTROL DE ENCENDIDO Y APAGADO DE LED'S DE UN ÁRBOL DE NAVIDAD CONTROLADO POR UN ATMEL AT89C51".

Este proyecto consiste de un conjunto de led's distribuidos de tal forma que grafican un árbol de navidad sobre una placa electrónica. El usuario mediante una interfaz que corre sobre el PC, escoge una de cuatro posibles opciones de encendido de led's, y esta opción es llevada a través de comunicación serial al microprocesador AT89C51 que es el encargado de ejecutar la secuencia deseada.

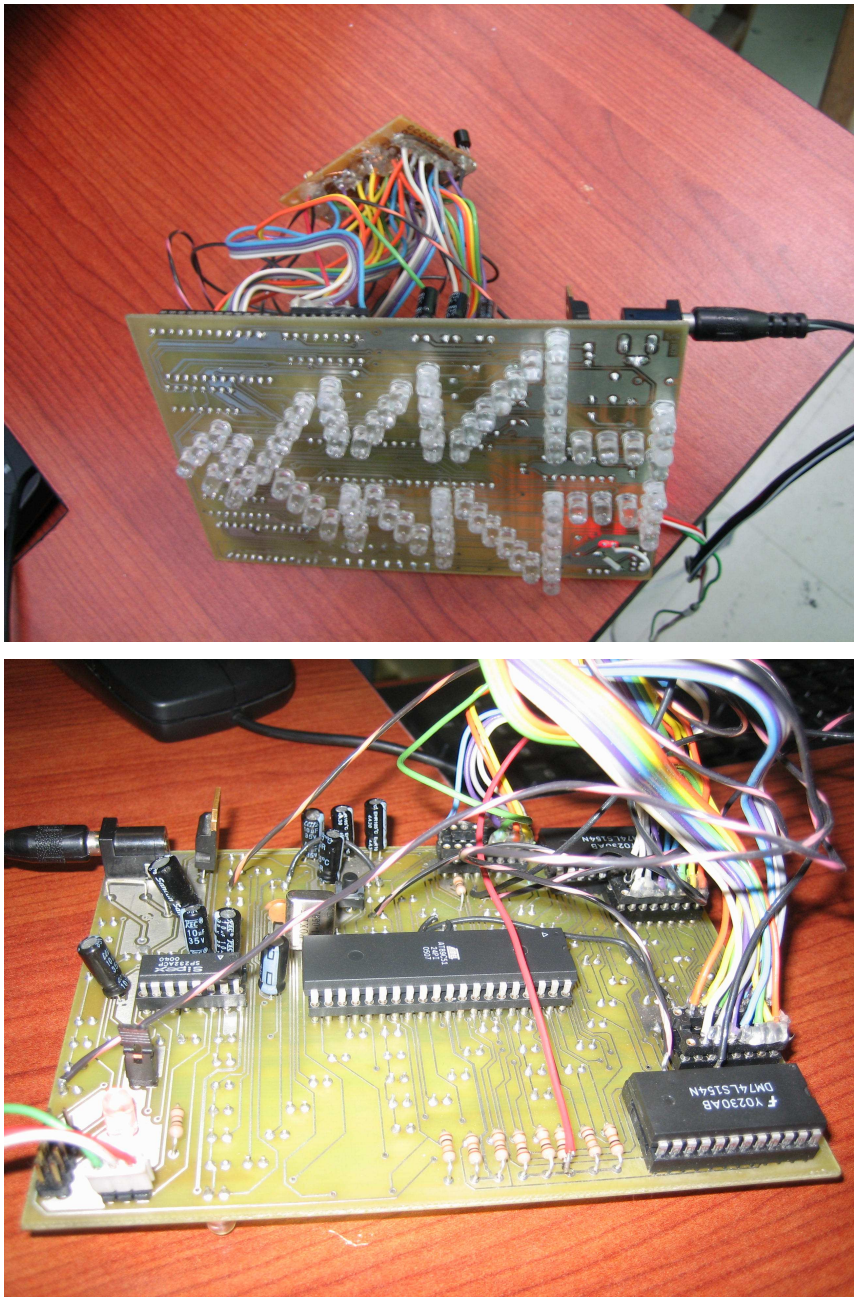


FIGURA 5-5: HARDWARE PERTENECIENTE AL ENCENDIDO Y APAGADO DE LEDs DE UN ÁRBOL DE NAVIDAD CONTROLADO POR EL ATMEL AT89C51

A continuación se muestra la interfaz del PC.

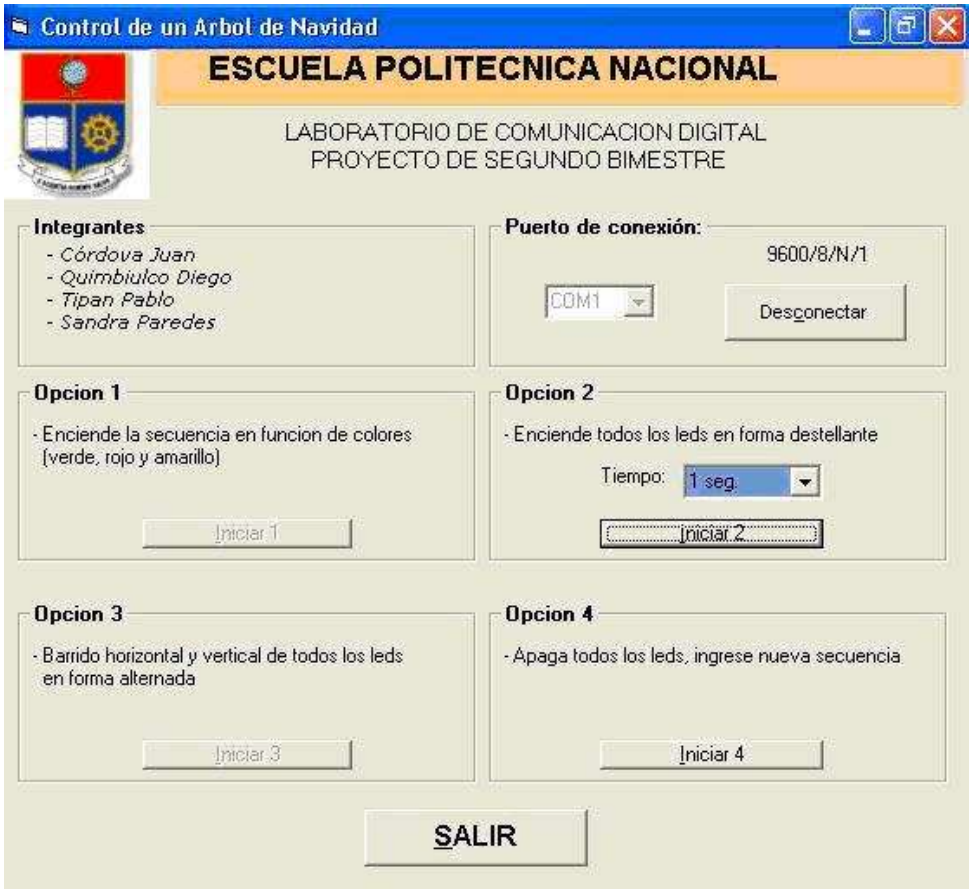


FIGURA 5-6: INTERFAZ DEL PROYECTO

Y los resultados de cada una de las opciones posibles, se grafican a continuación:

- Opción 1:

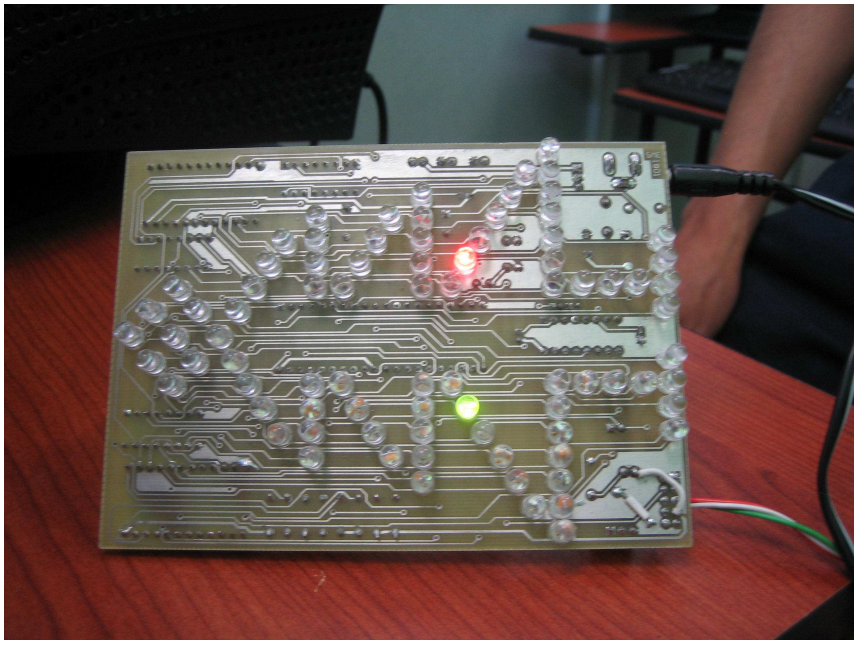


FIGURA 5-7

- Opción 2.

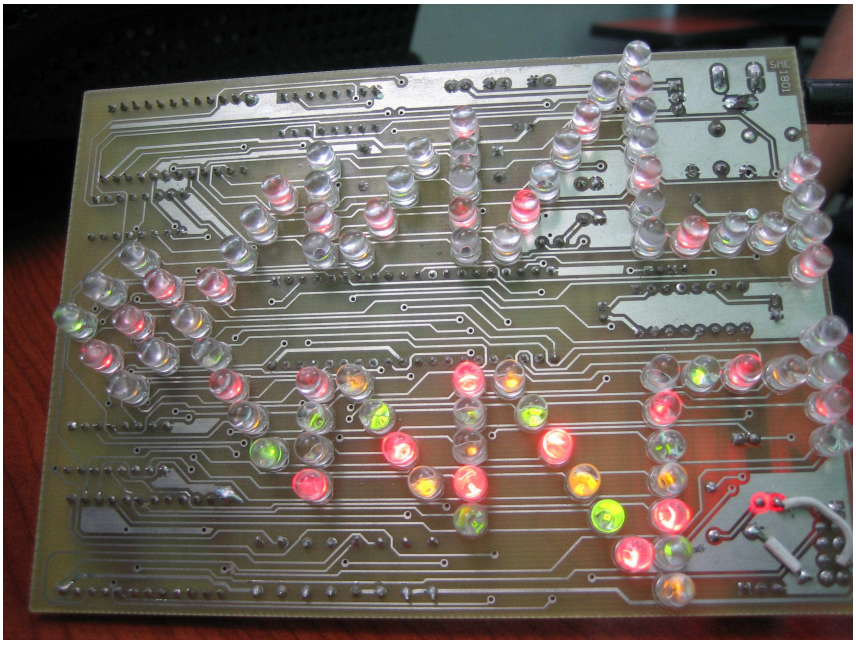


FIGURA 5-8

- Opción 3:

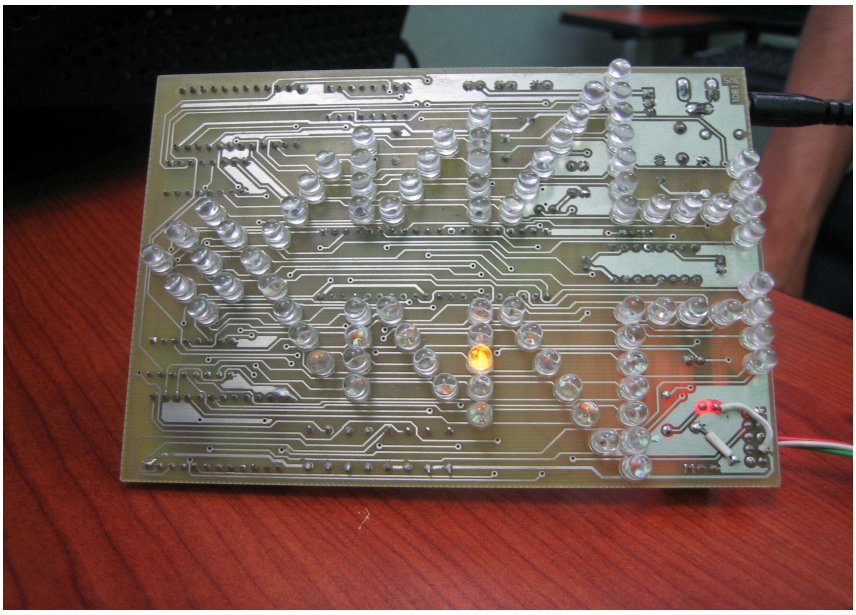


FIGURA 5-9

- Opción 4:

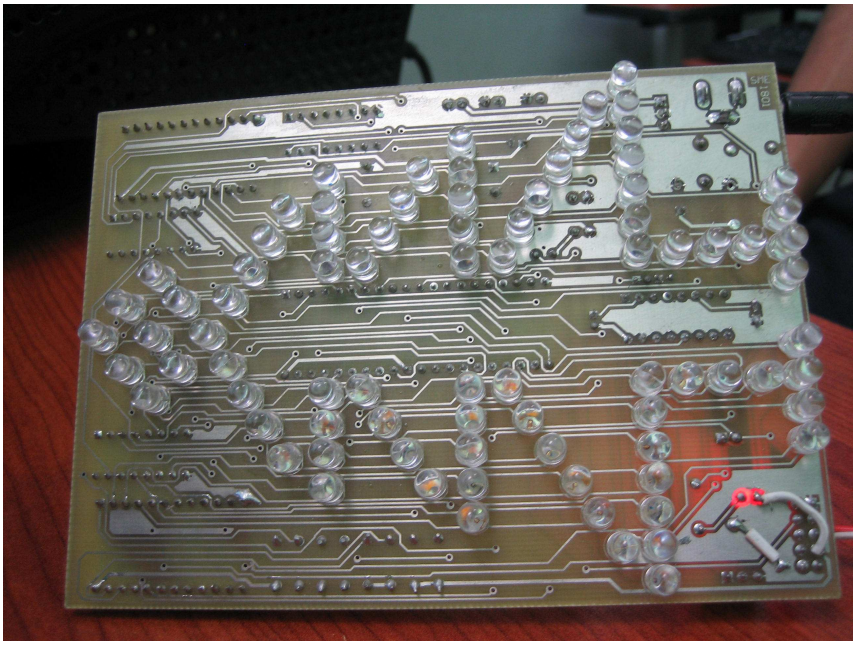


FIGURA 5-10

A continuación se muestran el resto de microcontroladores con los cuales se hicieron pruebas en diferentes áreas, obteniendo resultados satisfactorios en su programación y ejecución:

Microcontrolador	# de pines
PIC 16F83/84A	18
PIC 16F873/76A	28
PIC 16F877/77A	40
PIC 16C77/765	40
AT 89S51/52	40
AT 89C52	40

TABLA 5-1: MICROCONTROLADORES UTILIZADOS EN DIFERENTES PRUEBAS

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

De acuerdo a los resultados obtenidos en la fase de pruebas y resultados, ha sido posible confirmar el buen funcionamiento del programador, para ambos tipos de familias (PIC y ATMEL), por lo que se recomienda su uso en el desarrollo de cualquier proyecto.

Con el objetivo de que el dispositivo que maneja al puerto USB (PIC 16C745), no sufra daños, cuando se disponga programar algún elemento defectuoso, el dispositivo no maneja directamente a los pines de programación, si no que lo hace mediante un arreglo de transistores, según lo explicado en el Capítulo 3. Por lo general cuando un dispositivo está defectuoso, uno o algunos de sus pines se encuentran enclavados a Vcc., lo que afectaría al transistor encargado de proteger al pin del PIC 16C745 de este posible daño, este elemento puede ser fácilmente cambiado. Por lo que se garantiza confiabilidad en el funcionamiento del programador con dispositivos en buen estado.

Para la programación de los microprocesadores ATMEL, se necesita el direccionamiento de los espacios de memoria, que son accedidos de una forma paralela, ante esta circunstancia se ha implementado el direccionamiento mediante los circuitos integrados contadores 393, ahorrándose así el número de pines del PIC USB. Otra opción sería cambiar el PIC 16C745 (28 pines) al PIC16C765 (40 pines), encareciendo el costo del programador.

Fácil de conectar y desconectar, relativamente pequeño, sin fuente externa de alimentación, posibilidad de ampliar la biblioteca para nuevos dispositivos cuya programación sea serial. Todas estas ventajas justifican el tiempo invertido para el desarrollo de este proyecto.

Se puede ampliar la programación a otro tipo de fabricante de dispositivos que tengan como modo de programación serial, y además que su voltaje de programación sea de 13V.

Para un posterior trabajo se puede incluir la forma de programación a Bajo Voltaje (5V), bastaría con poner un switch lógico que mutiplexe entre 5V o 13V.

Todo el proceso seguido en el desarrollo del software y firmware ha sido siempre apegado al modo de funcionamiento del USB que dice: “El host (PC) es el único encargado de iniciar una comunicación” **[1].**

Si se desea aumentar la velocidad de programación que actualmente existe, se podría implementar con dispositivos de la familia 18FXXXX que soportan la interfaz USB y además que son de Alta velocidad, para este caso no se necesitaría el driver HIDComm, ya que puede trabajar directamente con las funciones API de Windows, para así asignarle un manejador al dispositivo.

En caso de no llenar el espacio correspondiente a determinado flujograma, cuando se encuentre dentro del proceso “Agregar Dispositivo”, se lo debe llenar con los números 01 y 07, para que en lo posterior no ocurra ningún problema.

Los dispositivos USB deben poseer una periodo de oscilación confiable, y en particular el PIC 16C745, que es un dispositivo de baja velocidad, trabaja con una frecuencia de oscilación de 24MHZ, esto lo logra utilizando un oscilador de 6MHZ, y su multiplicador de frecuencia (4KPLL) **[4].**

Se puede integrar dentro de este proyecto a la interfaz RS232, esto debido a que los microcontroladores PIC que manejan el USB también soportan la interfaz RS232, dándole así mayor versatilidad.

Por su bajo costo y garantía de funcionamiento, se recomienda la construcción de este programador para el uso de los estudiantes en sus respectivos laboratorios.

Para nuevos dispositivos de la familias PIC 16XXXX que se deseen agregar a la biblioteca del programador, se debe tener en cuenta que la palabra de configuración se la debe llenar en forma individual, bit a bit.

REFERENCIAS BIBLIOGRÁFICAS

[1] JANN AXELSON, “USB Complete”, Third Edition, Everything You Need to Develop Custom USB Peripherals, 2005

[2] MICROCHIP, “Hojas de Especificaciones para Programación de los microcontroladores PIC 16F87XA, 16F87X, 16F8X Y 16CXX”

[3] ATMEL, “Hojas de Especificaciones para Programación de los microcontroladores AT89C51/52/55 y AT89S51/52/55”.

[4] MICROCHIP, “Hojas de Datos del microcontrolador PIC 16C745”

[5] BAY LINEAR, “Hojas de Datos del regulador de voltaje LM78XX”

[6] http://en.wikipedia.org/wiki/USB

[7] KEC, “Hojas de Datos de los transistores 2N3904 y 2N3906”

[8] CEBALLOS FRANCISCO JAVIER, “Curso de Programación Visual Basic 6.0. Alfaomega, 2000.

[9] http://pdf1.alldatasheet.com/datasheetpdf/view/51077/FAIRCHILD/DM74LS393M.html

[10] http://www.alecmcnamara.freeserve.co.uk/pccalc/

[11] http://es.geocities.com/jnz_9zjn/

MANUAL DE USUARIO DEL PROGRAMADOR “EPNprog”

PROCESO DE INSTALACIÓN DEL “EPNprog”

El programador EPNprog debe ser instalado en la computadora a través del programa instalador, y durante este proceso se siguen los siguientes pasos

Para el normal funcionamiento del programa EPNprog, se necesita primero la instalación del driver HIDCOMM_b. Este driver lo provee el fabricante Microchip, y se encuentra incluido dentro de la carpeta de instalación “InstEPNprog”.

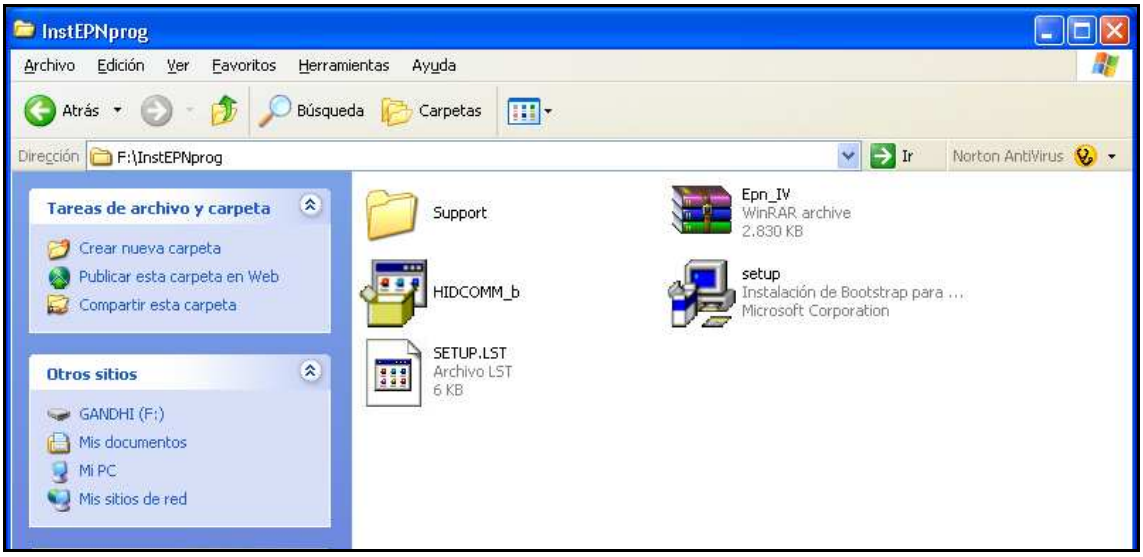


FIGURA A-1: CONTENIDO DE LA CARPETA InstEPNprog

Luego de haber instalado el driver HIDCOMM_b se procede a ejecutar el instalador “setup” del programador que se encuentra en la misma carpeta. Siguiendo los pasos solicitados en este proceso.

Es importante resaltar que automáticamente se instala el ejecutable del programador junto con el archivo BASE.mdb, este requisito es necesario para el normal funcionamiento del EPNprog. Si el usuario desea cambiar la ubicación del ejecutable, lo hará también con la BASE. mdb (Figura A-2), lo importante es mantener a la base junto al ejecutable cualquiera sea su ubicación.

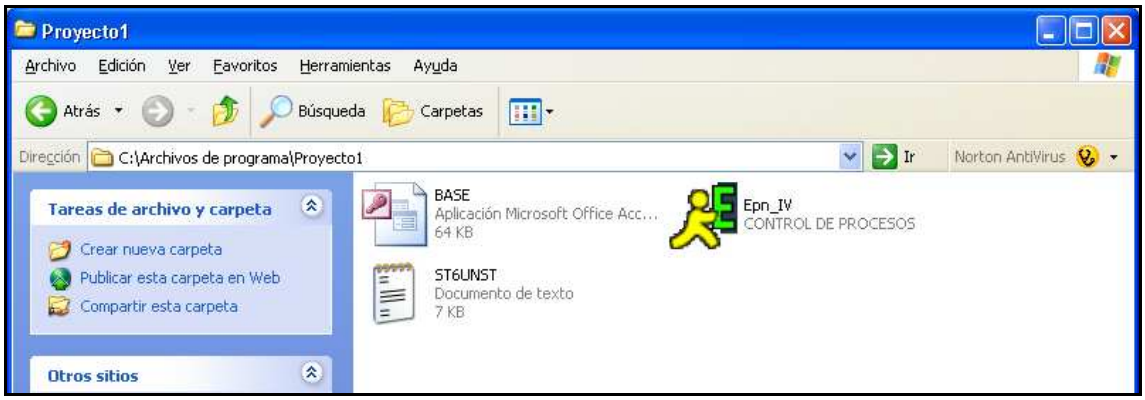


FIGURA A-2: EJECUTABLE EPNprog JUNTO AL ARCHIVO BASE

Además cuando el software se encuentra correctamente instalado y el hardware conectado (Figura A-3) se mostrará en el titulo de la ventana principal la frase “EPNprog con conexión USB” (Figura A-11), caso contrario únicamente aparecerá la frase EPNprog.

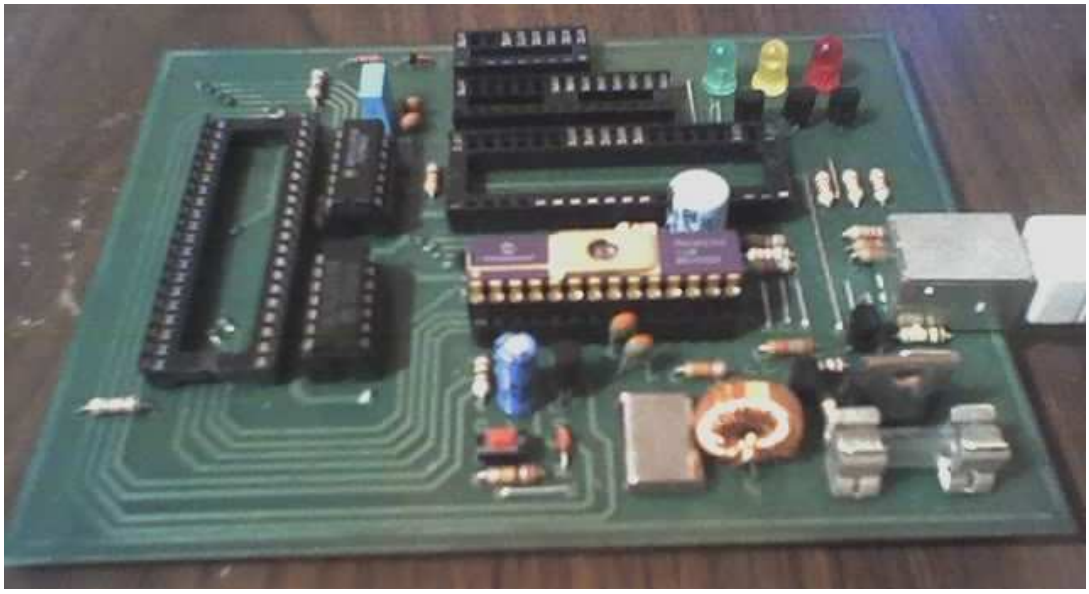


FIGURA A-3: HADWARE “EPNprog”

Una razón para que no se visualice la frase “EPNprog con conexión USB”, es que los drivers no se encuentran asignados todavía al dispositivo USB, los pasos a seguirse en este caso son:

- Abrir el Panel de control desde el menú Inicio:

FIGURA A-4

- Luego se escoge el icono Sistema, de la ventana siguiente:

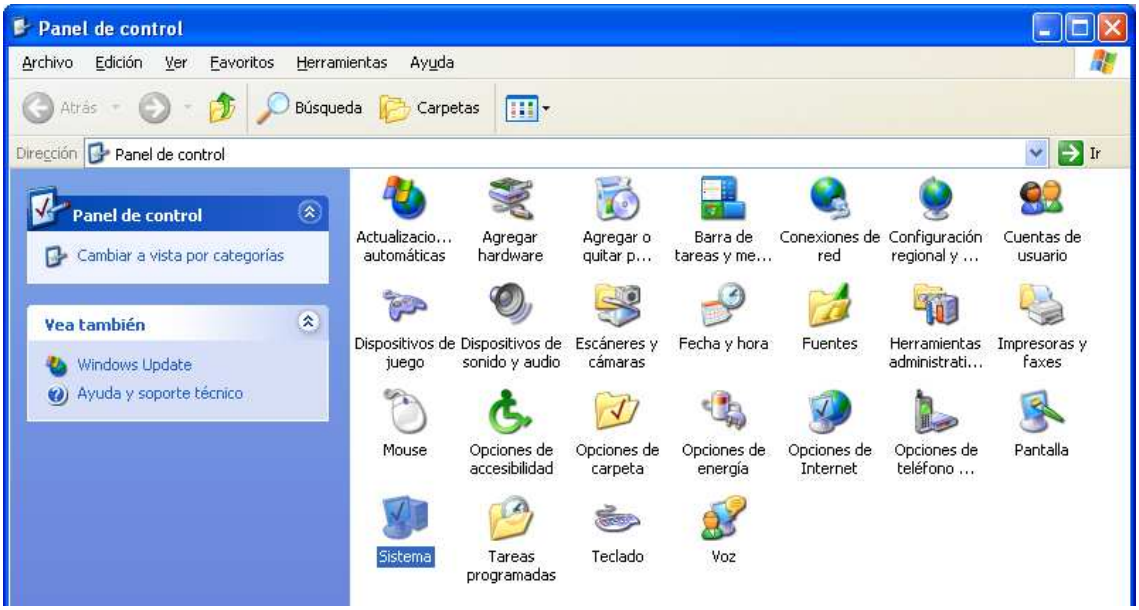


FIGURA A-5

- Posteriormente se despliega la ventana que posee algunas fichas, y dentro de la ficha hardware se encuentra la opción “Administrador de dispositivos” (Figura A-6), al desplegarse la ventana Administrador de dispositivos se ve en forma de árbol todos los recursos que posee el computador, y dentro de los USB está un dispositivo desconocido (Figura A-7), el cual corresponde al programador EPNprog, que aun no tiene asignado sus drivers.

FIGURA A-6

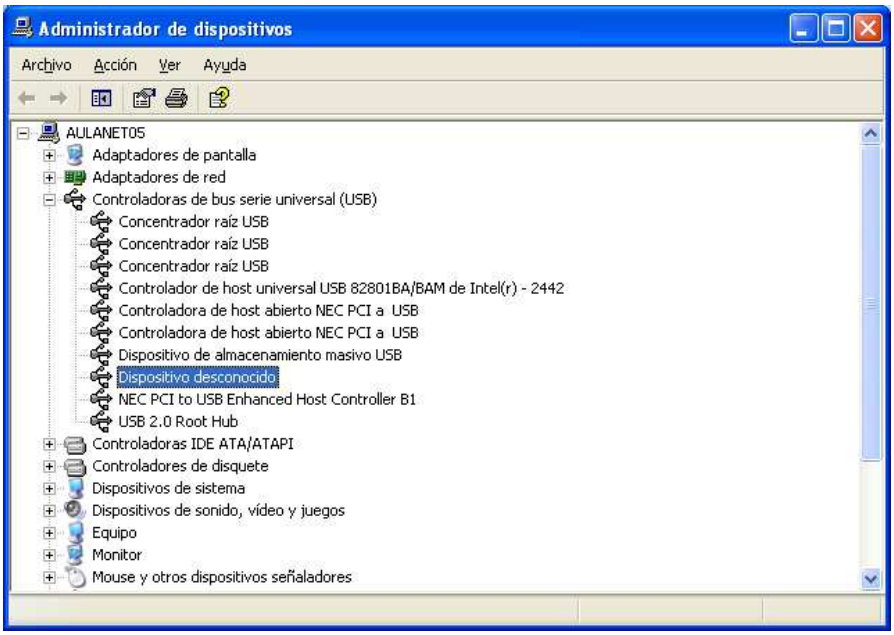


FIGURA A-7

- Dando doble click sobre el dispositivo desconocido, muestra la siguientes ventanas de actualización de drivers:

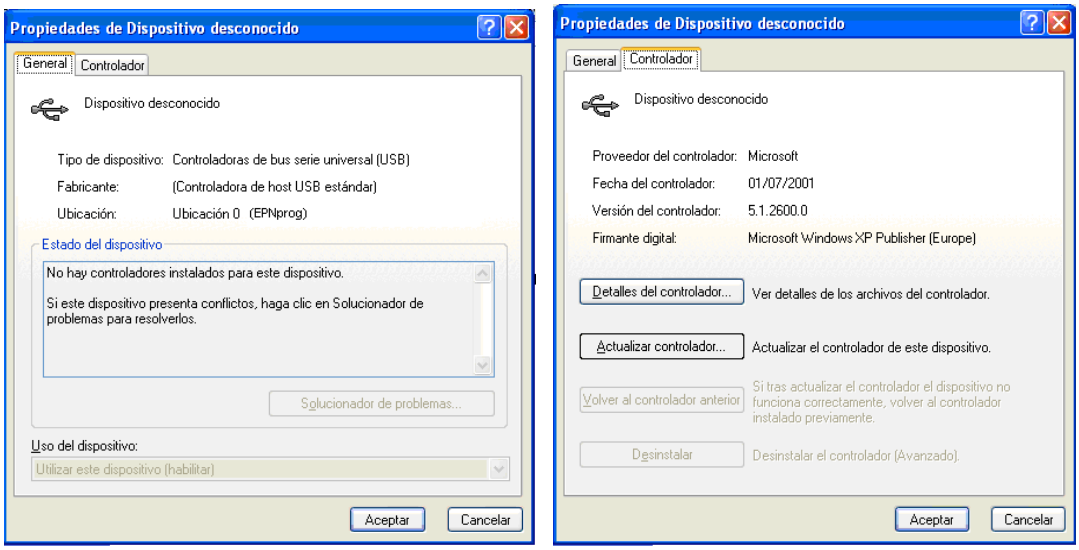


FIGURA A-8

- Y luego de escoger la opción “Actualizar controladores”, se sigue el proceso dirigido por el Asistente de Actualización de Hardware que asignará los drivers como lo muestran las siguientes figuras:

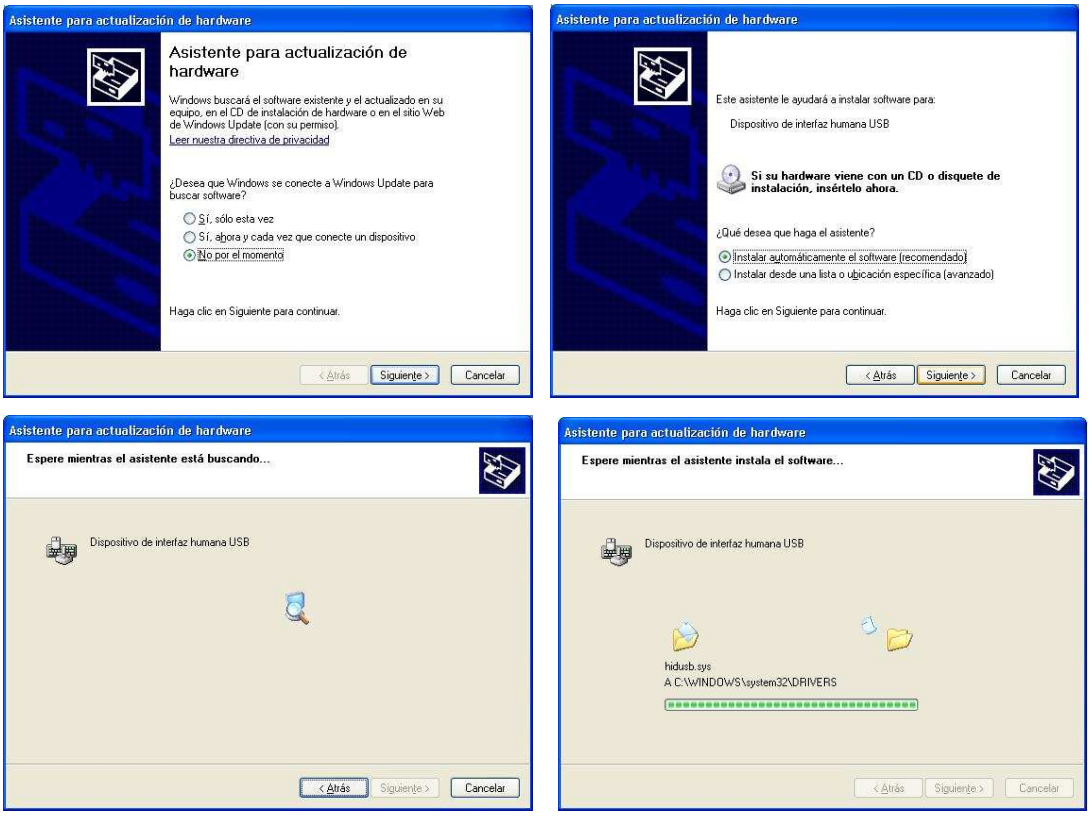


FIGURA A-9

FIGURA A-10

Una vez instalado el programador, esta listo para aprovechar todos los recursos que ofrece, en la ventana principal (Figura A-11) se ven los componentes tales como: Barra de herramientas, barra de menú, pestañas que contienen los diversos espacios de memoria según corresponda al dispositivo.

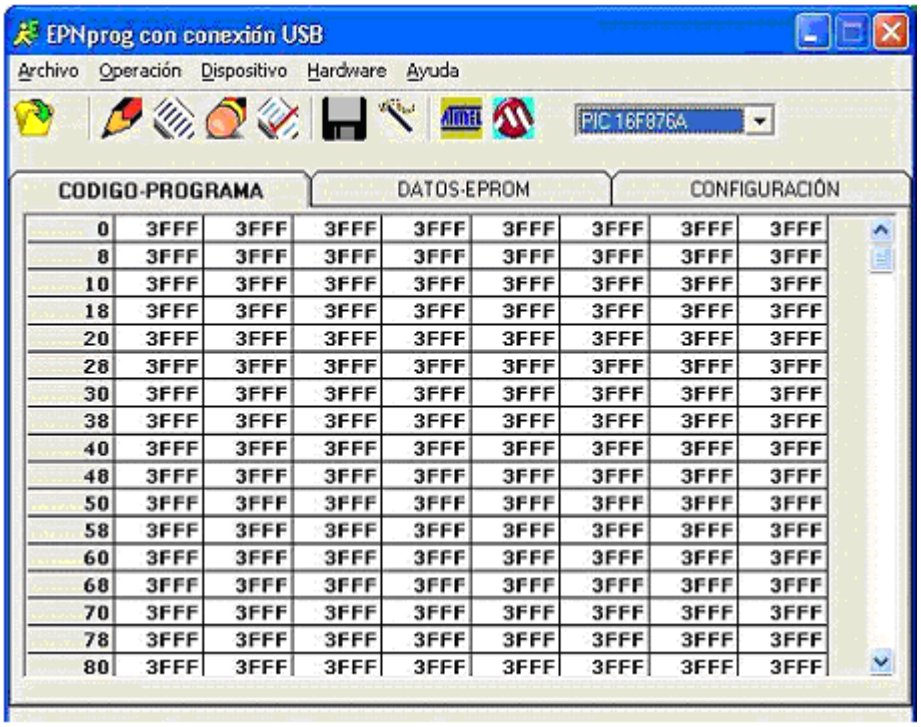


FIGURA A-11: VENTANA PRINCIPAL DEL “EPNprog”

En la barra de herramientas se muestran todas las posibles opciones que posee el programador. (Ver Figura A-12)

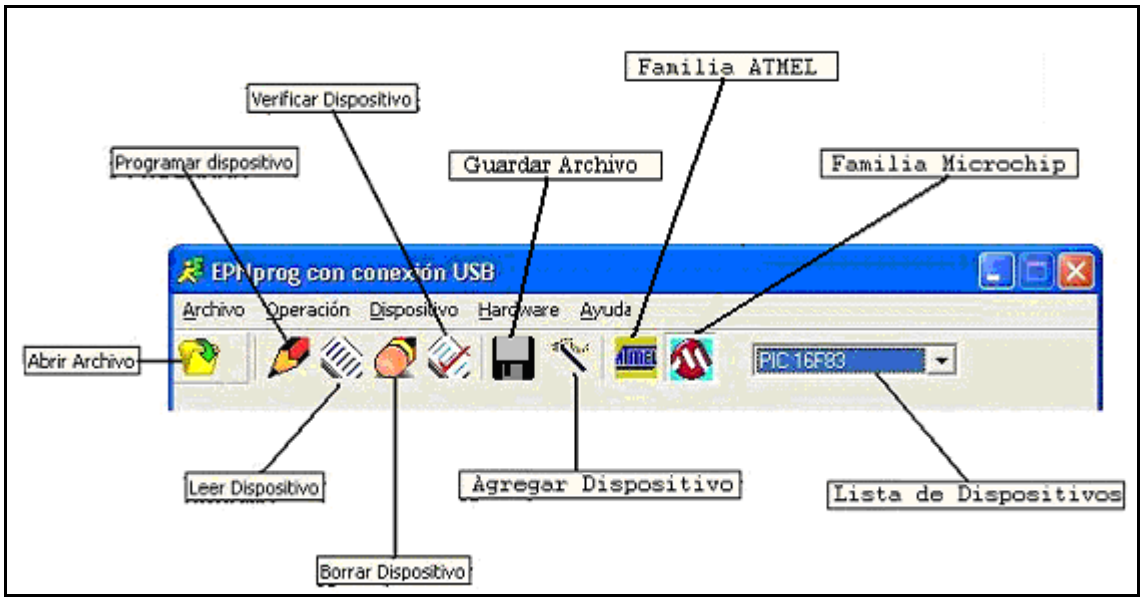


FIGURA A-12: DESCRIPCIÓN DE LOS ICONOS DE LA BARRA DE HERRAMIENTAS

En la barra de menú se encuentran también las posibles opciones. Como se muestra en la figura siguiente:

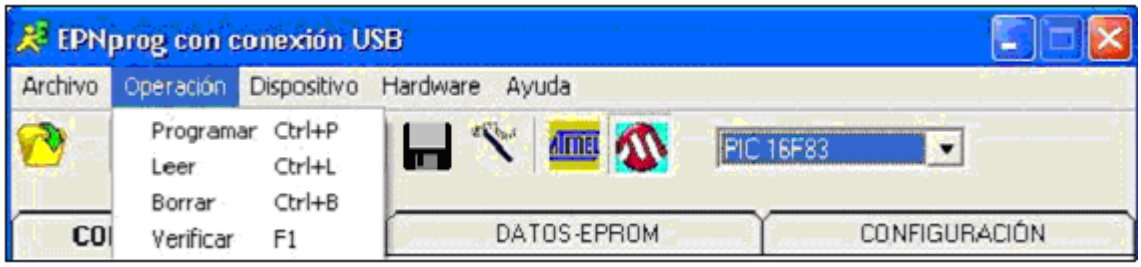


FIGURA A-13: BARRA DE MENU

En la barra de Menú al escoger la opción “Dispositivo” se despliega el respectivo submenú, que entre otras cosas tiene la opción “Agregar Dispositivo”, esta elección despliega la ventana de la Figura A-14, que tiene un texto que orienta al usuario en el uso de esta herramienta.

FIGURA A-14: VENTANA DEL ASISTENTE PARA AGREGAR DISPOSITIVO

A continuación se presenta la siguiente ventana, en la cual el usuario puede agregar, borrar y modificar dispositivos de la biblioteca

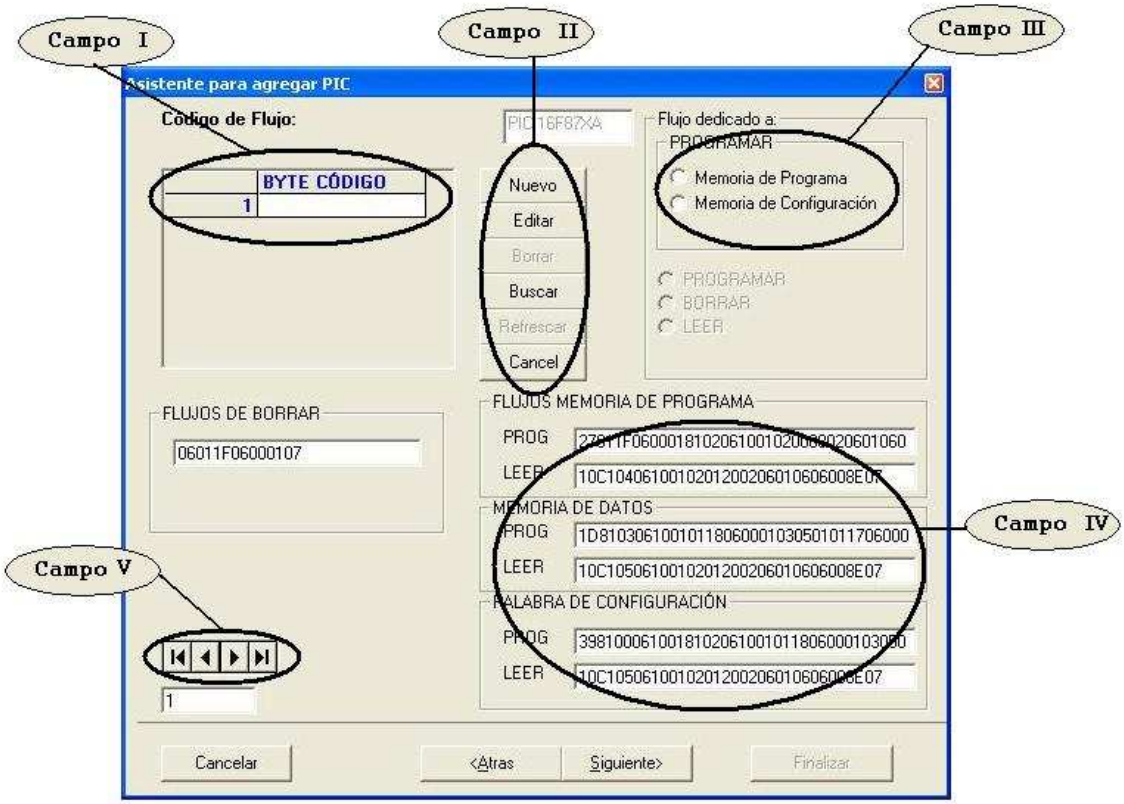


FIGURA A-15: VENTANA DE FLUJOGRAMAS

Esta ventana posee los siguientes campos:

- **Campo I**.- Región usada para introducir el flujograma correspondiente al dispositivo byte a byte.
- **Campo II**.- Opciones a ejecutarse sobre el dispositivo PIC.
- **Campo III**.- Aquí se escoge para que esta dedicado el flujograma.
- **Campo IV**.- Visualización de los distintos flujogramas dedicados al dispositivo. En caso de no disponer el flujograma correspondiente al espacio de memoria se debe llenar con la clave 0107.
- **Campo V**.- Esta barra sirve para desplazarse sobre las diferentes familias de microcontroladores PIC que están almacenadas en la biblioteca

Por ejemplo si el usuario desea ingresar los flujogramas relativos a la programación del PIC 16F83 que actualmente no está soportada, se procederá de la siguiente manera:

En la Figura A-15 se pulsa el botón “Nuevo” y automáticamente el programa solicita el ingreso de la familia a la cual pertenece el microcontrolador PIC, en este caso se llena con 16F8X, porque los flujogramas que escribirán posteriormente permiten programar además del PIC 16F83, a los dispositivos 16F84/84A.

Posteriormente y a través del *campo II* se selecciona el espacio de memoria al cual va dedicado el flujograma, así como su función (programar, leer, o borrar).

Luego de haber cumplido con los anteriores pasos, se procede a llenar byte a byte el *campo I* con el respectivo flujograma conforme lo explicado en el capítulo IV.

Una vez terminado de escribir el respectivo flujograma se pulsa el botón refrescar para que se almacene en la base de datos. Si se desea ingresar un nuevo flujograma perteneciente a la misma familia se pulsa el botón “Editar” y se procede de igual manera que los pasos anteriores

Al pulsar el botón Siguiente, se visualiza la ventana donde se hará el llenado de cada uno de los microcontroladores PIC que pertenecen a la familia, como se muestra en la Figura A-16.

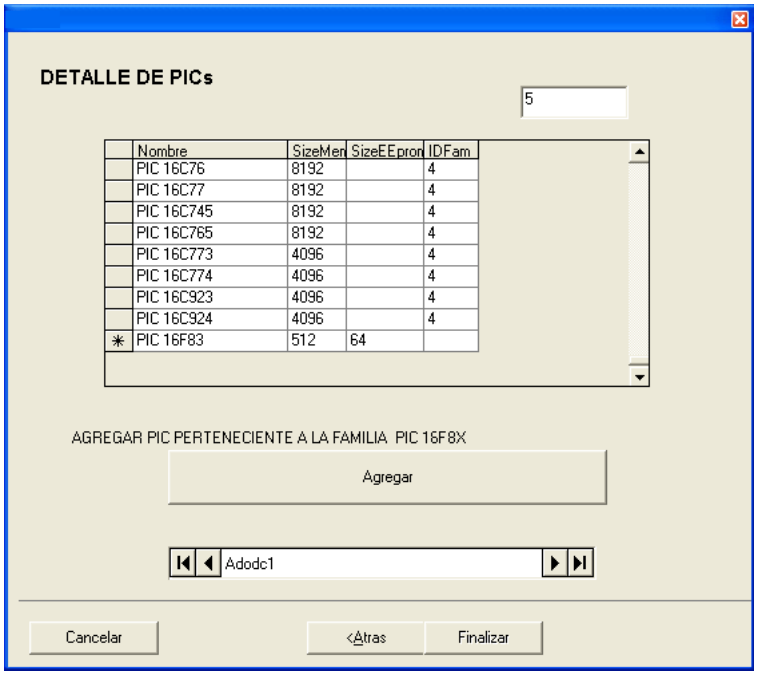


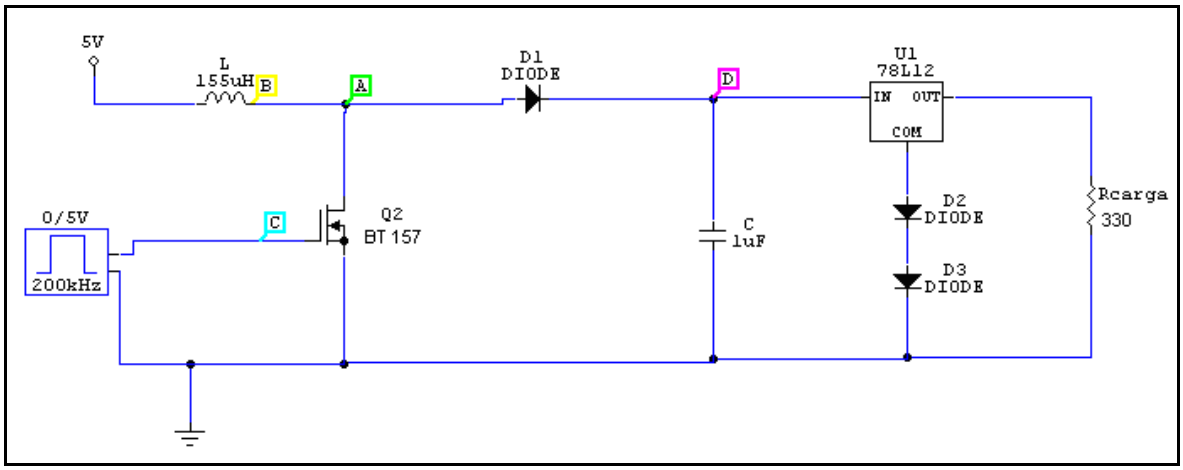
FIGURA A-16: DETALLE DE PIC's

En esta ventana además del nombre del microcontrolador PIC se requiere del tamaño de cada uno de los espacios de memoria, en caso de no poseer espacios de memoria EEPROM se deja en blanco, el campo IDFam de esta misma ventana no necesita ser llenado, este se llenará automáticamente después de pulsar la tecla siguiente del comando Adocdc1.

Una vez cumplido con los pasos anteriores y mediante la tecla Finalizar se habrá agregado los nuevos dispositivos PIC a la biblioteca del programador.

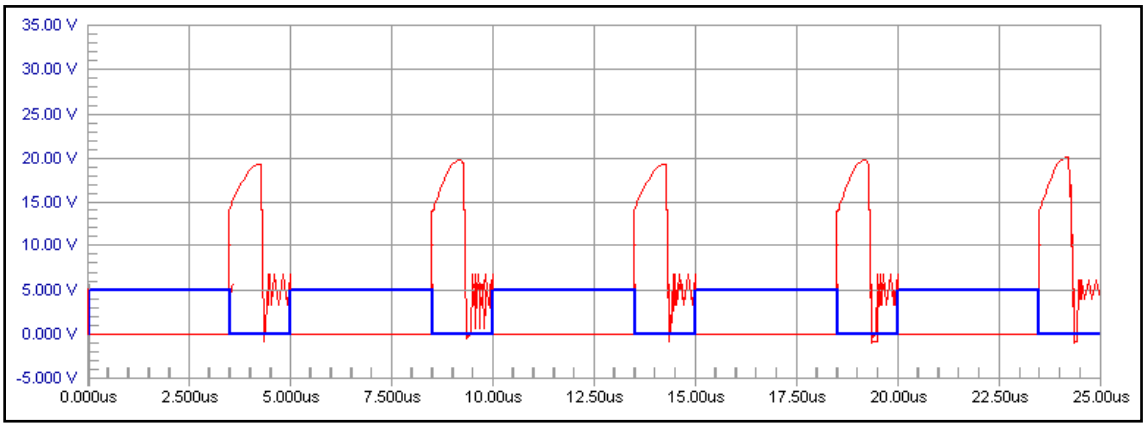
SIMULACIONES DEL CIRCUITO CONVERSOR ELEVADOR IMPLEMENTADO EN EL PROGRAMADOR EPNprog.

Para este propósito se utilizó el software CircuitMaker que está enfocado en la simulación de circuitos de Potencia, como las fuentes de switcheo forman parte de esta rama de la electrónica se decidió incursionar con esta herramienta.
El circuito simulado y las graficas obtenidas se muestran a continuación:

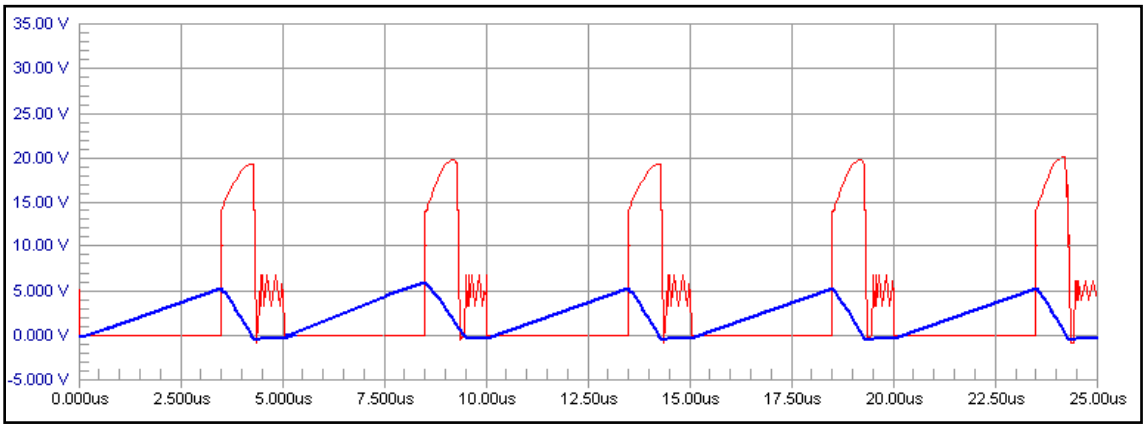


CIRCUITO SIMULADO EN CIRCUITMAKER

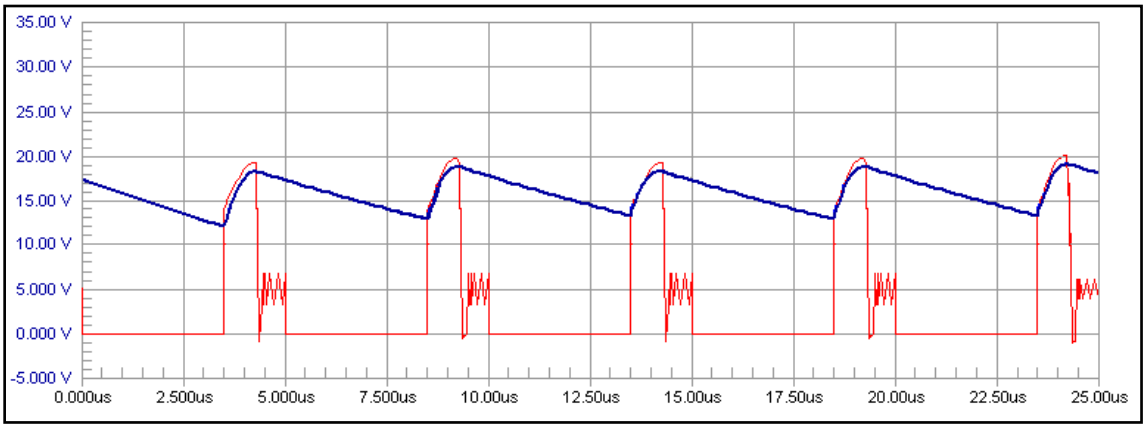
a)



b)



c)



FORMAS DE ONDA OBTENIDAS EN LA SIMULACIÓN

En las figuras anteriores, se observan las formas de onda en conducción discontinua obtenidas en la simulación. La onda de color rojo representa el voltaje en el drain del MOSFET. La onda azul varia de acuerdo a cada figura**: a)** pulsos a la entrada del gate del MOSFET(δ=0.7)**, b)** corriente que circula por la bobina**, c)** Carga y descarga del voltaje en el condensador (entrada al regulador 7812).