# ESCUELA POLITÉCNICA NACIONAL

## ESCUELA DE INGENIERÍA

# DISEÑO Y CONSTRUCCIÓN DE UN PROGRAMADOR PARA MICROCONTROLADORES ATMEL CON INTERFAZ USB

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

FREDY GIOVANNY GUANOLIQUÍN LLUMIQUINGA

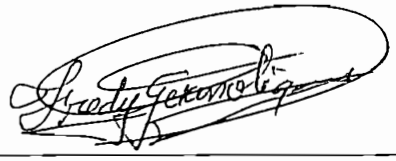
DIRECTOR: ING. JAIME VELARDE

Quito, Abril, 2004

# DECLARACIÓN

Yo Fredy Giovanny Guanoliquín Llumiquinga, declaro que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional: y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional, puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad intelectual, por su Reglamento y por la normatividad institucional vigente.



Fredy Giovanny Guanoliquín LLumiquinga

## CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Fredy Giovanny Guanoliquín Llumiquinga, bajo mi supervisión.

Ing. Jaime Velarde
DIRECTOR DEL PROYECTO

# CONTENIDO

# PRESENTACIÓN

Hoy en día resulta muy interesante observar como los avances tecnológicos nos sorprenden por la evolución tan rápida que presentan.

Los computadores y sus respectivos componentes no son una excepción a estos cambios. La necesidad de reconfigurar al PC para ir acorde con el avance tecnológico obliga a que la mayoría de personas tiendan a adquirir nuevos periféricos. Labor que hace varios años, no era tan aconsejable ya que se estaba obligado a abrir la carcaza del PC corriendo el riesgo de que se queme o deje de funcionar.

La llegada del bus USB (Universal Serial Bus ó «Bus serial universal») al mercado viene a simplificar la conexión de periféricos a los PCs. Esta actividad de añadir un periférico anteriormente lenta y costosa, se ve simplificada por las mejoras que introduce el USB respecto al puerto serie y paralelo convencional. Sus principales ventajas residen en la capacidad de soportar un verdadero *Conectar y desconectar (Plug&Play)* de los periféricos conectados al PC, no haciendo falta su configuración para que funcionen correctamente. Y que además de soportar la conexión y desconexión en caliente, sin necesidad de reiniciar el PC cada vez que lo enchufamos.

Todo lo expuesto anteriormente, nos sirve para empezar a comprender, que en estos momentos existe una nueva tecnología que facilita; como es por ejemplo: la reconfiguración de hardware, expansibilidad e interconexión del equipo con otros periféricos, etc. Este es el BUS SERIAL UNIVERSAL (USB); El que hará que en el presente trabajo se proceda a realizar un estudio amplio de gran claridad; mostrando todos los beneficios que esta tecnología esta aportando a los cambios técnicos que se producen en el campo de la computación.

# RESUMEN

Los programadores de microcontroladores facilitan a las personas dedicadas al desarrollo de software en los diferentes lenguajes como son: lenguaje assembler, lenguaje C y otros, a plasmar sus proyectos en los microcontroladores. Y como es una herramienta imprescindible surge la necesidad de construir un programador de microcontroladores con interfaz USB acorde al avance tecnológico, especialmente en lo que corresponde a computadores portátiles

El presente trabajo tiene como objetivo el diseño y construcción de un programador para microcontroladores ATMEL de la series AT89C51 y AVR que se conecte con una computadora personal mediante un interfaz USB. Como resultado de este trabajo se ha elaborado el presente documento, que se dividido en seis capítulos y seis anexos.

En el capítulo I se hace una introducción a la norma USB haciendo un análisis a la arquitectura y los protocolos de trasferencias en el bus. Dentro de la arquitectura se hace una descripción del sistema USB como son: la interconexión USB, los dispositivos USB, y el host USB. Dentro de este también se detallan las características eléctricas, mecánicas y funcionales del interfaz. En las características mecánicas se describen los diferentes componentes del interfaz como son conectores, cables, etc. Dentro de los protocolos de trasferencias en el bus se describen los tipos de paquetes, tipos transferencias, modelo lógico, clases de USB, etc.

El capítulo II describe las consideraciones necesarias para programar los microcontroladores ATMEL. Para lo cual se presentan las diferentes características de los microcontroladores como son la arquitectura, la distribución de pines para el modo de programación, voltajes de alimentación, algoritmo de programación, etc...

En el capítulo III se describen los pasos seguidos para el diseño del programador. Partiendo desde el diagrama de bloque general, hasta terminar con el diseño final del programar.

En el capítulo IV se hace una introducción a las herramientas que se disponen para elaborar el firmware y el software del programador. En el caso del desarrollo del firmware se desglosa los ejemplos proporcionados por Microchip Tecnology y de igual manera la descripción de las herramientas que proveen cada uno de estos ejemplos. En el desarrollo del software se describen las herramientas necesarias para realizar el programa.

En el capítulo V se muestran los resultados del trabajo realizado. Este capitulo es de particular importancia para aquellas personas que no estén interesadas en detalles de diseño o de construcción, si no solamente de los resultados concretos por el equipo diseñado.

Y por ultimo en el capítulo VI se presentan las conclusiones y recomendaciones necesarias después de haber elaborado el presente trabajo.

Los anexos contienen información que sirve como complemento a lo tratado en los seis capítulos del documento.

Aunque el documento luce bastante voluminoso, se ha tratado de que cada capítulo y anexo del mismo este escrito de la manera más concreta y explicita posible.

# CAPÍTULO I

# CAPÍTULO I

# LA NORMA USB (Bus Serial Universal)

## 1.1 INTRODUCCIÓN A LA NORMA USB

Cada cierto tiempo dentro del mundo de la computación se dan cambios realmente importantes, cambios que de cierta forma abren nuevos horizontes y mayores posibilidades a todos los usuarios. Es cierto que inicialmente es algo molesto tener que adecuarse a las nuevas tecnologías, pero el usuario es siempre el más beneficiado. Una de las mayores revoluciones en el mundo de la computación, ya sea por la forma de interconectar periféricos a las computadoras, la expansibilidad, la sencillez de configuración y uso del hardware es el BUS SERIAL UNIVERSAL (USB).

Desde hace varios años atrás las revistas técnicas y de investigación hablaban sobre este tipo de bus, para que luego de mucho trabajo y consenso, la especificación USB Versión 1.1 esté disponible para todas las empresas de fabricación de hardware del mundo y lo harán mucho más en el futuro próximo, diversos dispositivos que soportan esta especificación y la Versión 2.0.

El sistema USB no está patentado por ninguna empresa, lo que facilita su conversión en sistema de conexión estándar para PC. Así, empresas de reconocido prestigio se unieron para especificar las características que debía cumplir este sistema, lo que significó una buena noticia para usuarios y fabricantes, pues si una sola compañía monopolizara el bus y ofreciera licencias para su uso jamás sería soportado por los fabricantes de periféricos y evidentemente dejaría de ser un sistema de conexión estándar.

La documentación técnica relacionada a USB es realmente abundante y de gran profundidad, no solamente informática, sino también eléctrica, electrónica y mecánica. Por esta razón es altamente recomendable analizarla detalladamente, desde sus orígenes y la motivación para su actual existencia.

### 1.1.1 DEFINICIÓN DEL BUS SERIE UNIVERSAL

USB es una especificación de las empresas Compaq, Intel, Microsoft y NEC, que describe un canal de comunicación serial que soporta una gran variedad de periféricos de alta, media y baja velocidad, con soporte integral para transferencias en tiempo real como voz, audio y vídeo comprimido, y que permite mezclar dispositivos. Por lo tanto, entre los dispositivos USB más característicos se pueden citar teclados, ratones, joysticks, tabletas gráficas, monitores, módems, impresoras, escáneres, CD-ROMs, dispositivos de audio (como micrófonos o altavoces digitales), cámaras digitales y otros dispositivos multimedia. La figura 1.1 muestra una conformación entre estos dispositivos.

| Prestación | Aplicación | Atributos |
|---|---|---|
| Velocidad baja<br>Dispositivos interactivos.<br>10-100 Kbps. | Teclado, ratón.<br>Periféricos para juegos.<br>Configuración del monitor. | Bajo Costo<br>Conexión/desconexión dinámica.<br>Facilidad de uso.<br>Múltiples periféricos. |
| Velocidad media<br>Audio, vídeo comprimido, teléfono.<br>500 Kbps a 10 Mbps | RDSI (modems)<br>PBX (teléfono)<br>Audio (parlantes) | Bajo Costo<br>Conexión/desconexión dinámica.<br>Facilidad de uso.<br>Ancho de banda garantizado.<br>Latencia garantizada. |
| Velocidad alta<br>Vídeo, discos.<br>25 a 500 Mbps | Vídeo (Cámaras)<br>Discos(discos duros)<br>Red Local | Conexión/desconexión dinámica.<br>Facilidad de uso.<br>Amplio ancho de banda.<br>Latencia Garantizada. |

Figura1.1: Periféricos con interfaz USB

La motivación que ha dado origen al puerto USB, proviene de tres aspectos interrelacionados, que son:

- Conexión de periféricos con el PC.
- Fácil uso.
- Expansión del puerto

### 1.1.3 CARACTERÍSTICAS.

Realmente muchas son las características y beneficios de USB, a continuación se detallan algunas de las características de este bus:

- Todos los dispositivos USB tienen el mismo tipo de cable y el mismo tipo de conector, más allá de la función que cumplan.
- Los detalles de consumo y administración eléctrica del dispositivo son completamente transparentes para el usuario.
- El computador identifica automáticamente un dispositivo agregado mientras esta operando.

- Comparten un mismo bus tanto dispositivos que requieren unos pocos Kbps (kilo bits por segundo) como los que requieren varios Mbps (Mega bits por segundo. Ver la figura 1.1.
- Hasta 127 dispositivos diferentes pueden estar conectados simultáneamente y operando en una misma computadora sobre el USB.
- El bus permite periféricos multifunción, es decir aquellos que pueden realizar varias tareas a la vez, como lo son algunas impresoras que adicionalmente son fotocopiadoras y máquinas de fax.
- Capacidad para manejo y recuperación de errores producidos por cualquier dispositivo.
- Una distribución de alimentación desde el Controlador USB, que permite la conexión tanto de dispositivos alimentados desde el bus como autoalimentados.
- Soporte para la arquitectura Conectar y Operar (Plug & Play).
- Una arquitectura fácilmente escalable para permitir la existencia de varios Controladores USB en un mismo sistema.
- Bajo costo.

## 1.2 ARQUITECTURA DEL USB.

El Bus Serial Universal está dado esencialmente por un cable especialmente diseñado para transmisión de datos entre la computadora (sistema anfitrión o host), y diferentes periféricos, que pueden acceder simultáneamente al mismo con el fin de recibir o transmitir datos. Todos los dispositivos conectados acceden al canal o medio para transmitir sus datos de acuerdo a las normas de administración del host regido por un protocolo de 'paso de testigo' (*token*) que consecutivamente va dando la posibilidad de transmitir a cada periférico. La configuración de los dispositivos recuerda a un árbol, en el que desde cada rama pueden nacer otros tallos, los que a su vez pueden dividirse en otros más. La clave de la arquitectura se encuentra en los concentradores o Hubs, que, al igual que ocurre en las redes, representan un concentrador que dispone de una línea

de entrada y varias salidas. Cada una de ellas permite enchufar otro Hub ofreciendo la configuración típica de un bus USB.

Esta arquitectura del bus garantiza la posibilidad de que los periféricos sean conectados y desconectados del host mientras este y otros periféricos están operando normalmente.

### 1.2.1 DESCRIPCIÓN DEL SISTEMA USB

La Figura 1.2 muestra la estratificación del sistema USB. El mismo que está compuesto por tres áreas claramente demarcadas:

- La interconexión USB.
- Los dispositivos USB.
- El host USB.

#### 1.2.1.1 Interconexión USB

La interconexión del bus USB define cómo se conectan y comunican los dispositivos con el sistema anfitrión (host) a través del bus. Esto incluye: la topología del bus o el modelo de conexión entre los dispositivos USB y el host; los modelos de flujo de datos; es decir la forma en la que la información se mueve en el sistema entre los diversos elementos del mismo; la planificación USB que define la secuencia en la cual los dispositivos accederán al bus y finalmente las relaciones entre capas del modelo, y las funciones de cada capa.

##### *1.2.1.1.1 Topología del bus*

La forma física en la que los elementos se interconectan dentro del sistema USB, puede asemejarse a la topología estrella estratificada piramidalmente. El centro de cada estrella es un Hub, un dispositivo que por un lado se conecta al computador o a otro Hub y por otro lado, permite conectar varios dispositivos o en su defecto nuevos Hubs.

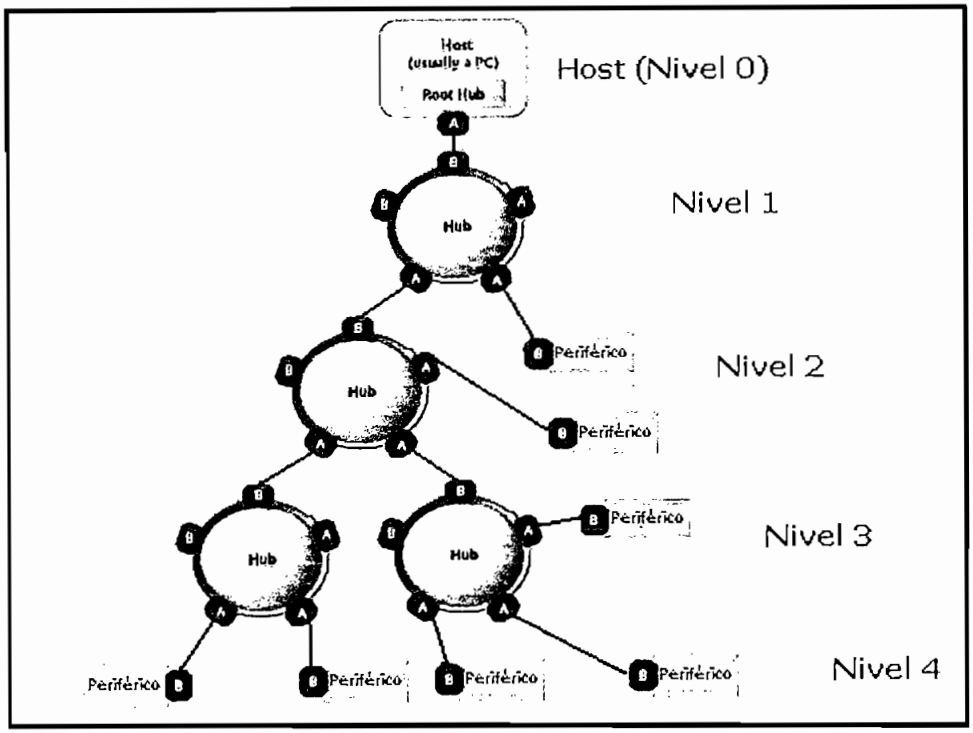



Figura1.2: Topología del bus USB

### 1.2.1.2 Dispositivos USB

Dentro del sistema USB existen dos tipos de dispositivos, los periféricos y los Hubs. Es evidente que todos estos dispositivos deben tener la capacidad de soportar la especificación USB en cuanto a protocolo de comunicación, operación USB, configuración y reseteo USB.

#### *1.2.1.2.1 Concentradores o HUBS*

Los Hubs son concentradores cableados que permiten múltiples conexiones simultáneas. Su aspecto más interesante es la concatenación, función por la que a un Hub se puede conectar otro y otro; ampliando la cantidad de puertos disponibles para periféricos.

Entre las características que destacan al Hub se tienen los siguientes:

- Genera alimentación hacia los dispositivos e incorpora la terminación de las líneas.

- Los Hubs disponen de una conexión "upstream" hacia el ordenador y una o varias conexiones "downstream" hacia dispositivos u otros Hubs (ver figura 1.3), de forma que se pueden encadenar varios Hubs para formar una topología en varios niveles. USB permite hasta 6 niveles, y en el nivel 0 (Raíz o Root) se encuentra el Controlador USB, que controla todo el tráfico de información en el bus.
- Los Hubs pueden estar integrados dentro de algún dispositivo (teclados, impresoras, monitores, etc.), y también pueden estar disponibles como elementos independientes (Ver figura 1.4).
- El Hub dispone de un Repetidor, para pasar información entre el puerto Upstream y los puertos Downstreams (Ver figura 1.3), y de un Controlador, que incorpora un juego de registros a través de los cuales el Controlador USB configura el Hub y controla y monitoriza los puertos Downstream.
- Normalmente los Hubs serán autoalimentados, aunque bajo ciertas restricciones topológicas podrían utilizarse Hubs alimentados desde el bus.

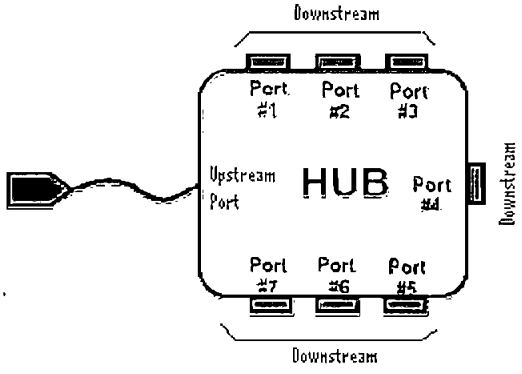



Figura 1.3: Esquema de un Hub USB

- El Hub USB tiene la capacidad de detectar si un periférico ha sido conectado a uno de sus puertos, notificando de inmediato al Controlador del Host en el computador. Proceso que desata la configuración del equipo nuevo; adicionalmente, los Hubs también son capaces de detectar la desconexión de un dispositivo, notificando al Controlador del Host que debe remover las estructuras de datos y programas de administración (drivers) del dispositivo retirado.
- Una de las funciones importantes de los Hubs es la de aislar a los puertos de baja velocidad de las transferencias a alta velocidad, proceso sin el cual

todos los dispositivos de baja velocidad conectados al bus entrarían en colapso.

- El Hub está compuesto por dos partes importantes: El Controlador del Hub y el Repetidor del Hub. El Repetidor del Hub tiene la función de analizar, corregir y retransmitir la información que llega al Hub, hacia los puertos del mismo. Mantiene una memoria consistente en varios registros de interfaz que le permiten sostener diálogos con el host y llevar adelante algunas funciones administrativas; mientras que el Controlador del Hub puede asemejarse a un pequeña CPU de supervisión de las múltiples funciones que deben desempeñar un Hub.

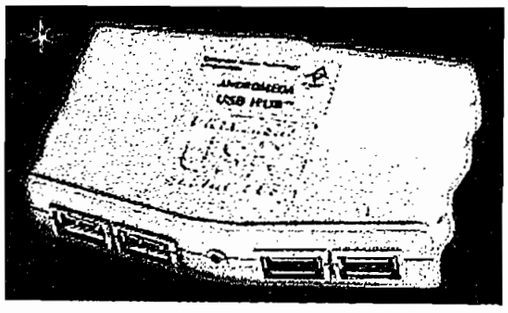


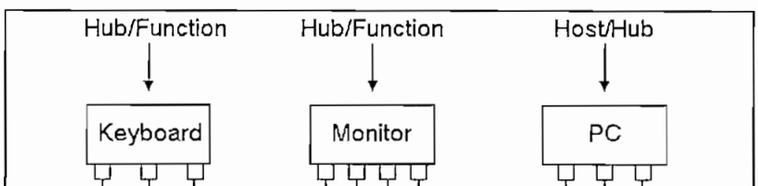
Figura 1.4: Hub USB comercial

*1.2.1.2.2 Periféricos.*

USB soporta periféricos de baja, media y alta velocidad, empleando tres velocidades para la transmisión de datos 1.5, 12 y 480 Mbps consiguiendo así una utilización más eficiente de sus recursos. Como los periféricos de baja velocidad (teclados, ratones, joysticks, etc.), no requieren de velocidades mayores que 1,5 Mbps, se puede dedicar más recursos del sistema a periféricos tales como monitores, impresoras, módems, scanner, equipos de audio, que precisan de velocidades más altas para transmitir mayor volumen de información o datos cuya dependencia temporal es más estricta.

Cabe aclarar que dentro de la terminología USB, todos los dispositivos (periféricos) que pueden ser conectados al bus USB, a excepción de los Hubs; se denominan Funciones.

Entre las funciones mas típicas están el ratón, el monitor, altoparlantes, módem, etc. Ver figura 1.5 y 1.6.



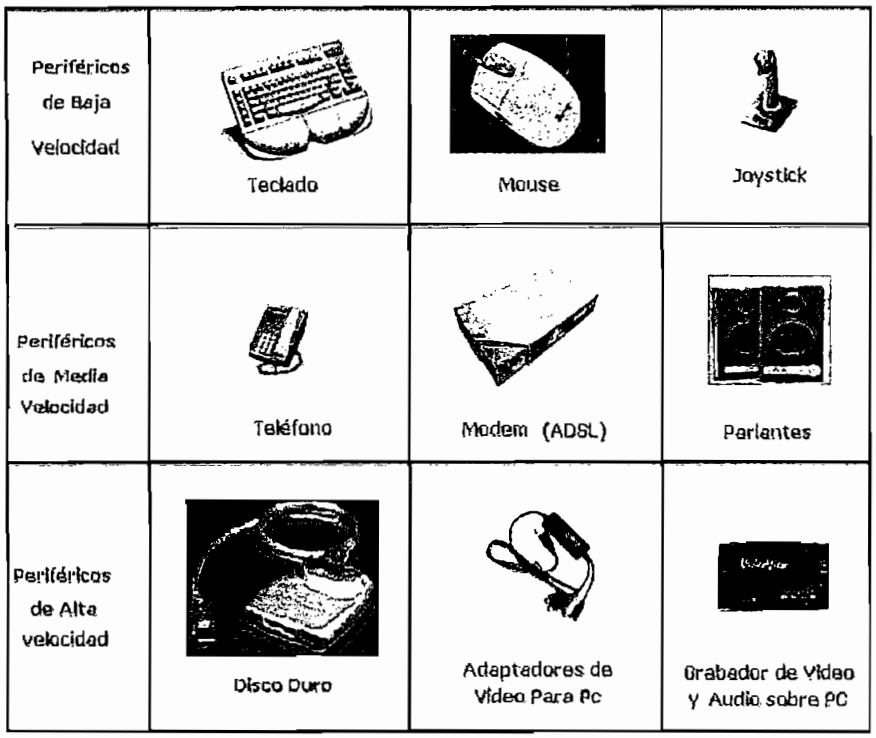



Figura 1.5: Funciones o dispositivos USB mas comunes

Cabe aclarar que dentro de la terminología USB, todos los dispositivos (periféricos) que pueden ser conectados al bus USB, a excepción de los Hubs; se denominan Funciones.

Entre las funciones mas típicas están el ratón, el monitor, altoparlantes, módem, etc. Ver figura 1.5 y 1.6.

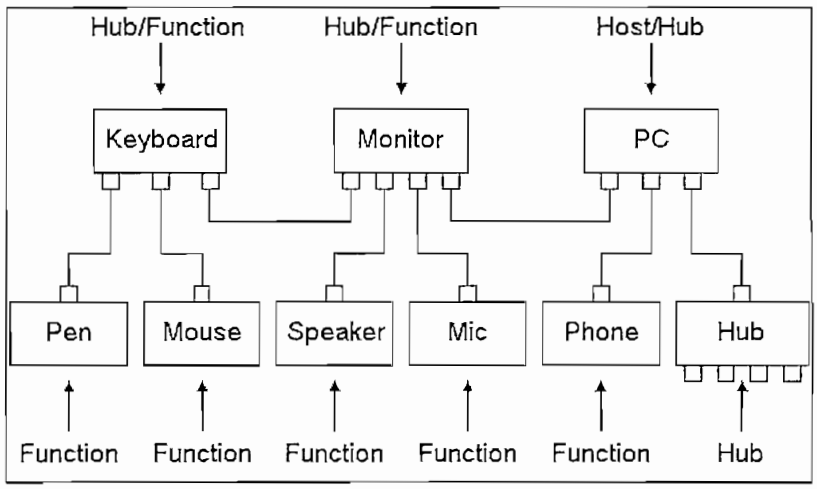



Figura 1.6: Esquema de interconexión USB

### 1.2.1.3 El host USB

A diferencia de los dispositivos y los Hubs, existe tan solo un host dentro del sistema USB, que es una parte del computador, denominado Controlador USB del Host. Este tiene la misión de hacer de interfaz entre la PC y los diferentes dispositivos. Existen algunas características respecto a este controlador. Su implementación es una combinación de hardware y software todo en uno, es decir firmware. Puede proveer de uno o dos puntos de conexión iniciales, denominados Hub Raíz; a partir de los cuales y de forma ramificada irán conectándose los periféricos.

El host es responsable al nivel de hardware, de los siguientes aspectos dentro del sistema USB:

- Detectar tanto la conexión de nuevos dispositivos USB al sistema, como la supresión de aquellos ya conectados, y por supuesto, configurarlos y ponerlos a disposición del usuario; tarea que involucra acciones por software.
- Administrar y controlar el flujo de datos entre el host y los dispositivos USB.
- Administrar y regular los flujos de control entre el host y los dispositivos USB, con el objeto de mantener el orden dentro de los elementos del sistema.
- Recolectar y resumir estadísticas de actividades y estados de los elementos del sistema.
- Proveer de una cantidad limitada de energía eléctrica para aquellos dispositivos que pueden abastecerse con tan solo la energía eléctrica proveniente desde el computador.

A nivel de software las funciones del Controlador USB del Host son las siguientes:

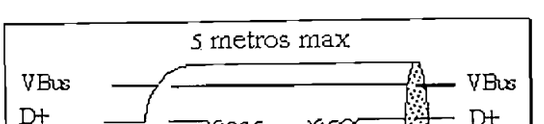
- Enumeración y configuración de los dispositivos conectados al sistema.
- Administración y control de transferencias de información.
- Administración avanzada de suministro eléctrico a los diferentes dispositivos.
- Administración de la información del bus y los dispositivos USB.

Figura 1.7: Chips Controladores USB

## 1.2.2 INTERFAZ FÍSICO

### 1.2.2.1 Especificación Eléctrica

El USB transfiere señales de información y energía eléctrica a través de 4 cables, cuya disposición se muestra en la Figura 1.8.



| Parámetro | Valor Requerido |
|---|---|
| " 1" Diferencial | (D+) – (D-) > 200mV |
| " 0" Diferencial | (D-) – (D+) > 200mV |

Tabla 1.2: Niveles o diferencial de voltajes entre D+ y D-

Los otros dos cables VBus y GND tienen la misión de llevar suministro eléctrico a los dispositivos, con un voltaje nominal de +5V para VBus. Los cables USB permiten una distancia que va desde los pocos centímetros hasta los varios metros. Los cables USB tienen protectores de voltaje a fin de evitar cualquier daño a los equipos.

Las intensidades de corrientes que pueden proporcionar el host a través del interfaz USB varían de 0 a 500 mA, según el tipo de dispositivo conectado.
El USB 2.0 permite la alimentación de dispositivos sencillos y lentos que consumen como un máximo de 2,5 W,

### 1.2.2.2 Especificación Mecánica

*1.2.2.2.1 Cables*

USB transfiere señales y energía a los periféricos utilizando un cable de 4 hilos, apantallado para transmisiones a 12 y 480 Mbps y no apantallado para transmisiones a 1.5 Mbps. En la figura 1.8 se muestra un esquema del cable, con dos conductores para alimentación y los otros dos para señal, debiendo estos últimos ser trenzados o no según la velocidad de transmisión.

USB limita la longitud de un cable entre dispositivos de alta y media velocidad a 5 metros y para dispositivos de baja velocidad el límite es de 3 metros. Dependiendo estas longitudes de sus características eléctricas.

El calibre de los conductores destinados a alimentación de los periféricos varía desde 20 a 26 AWG, mientras que el de los conductores de señal es de 28 AWG (ver tabla 1.3).

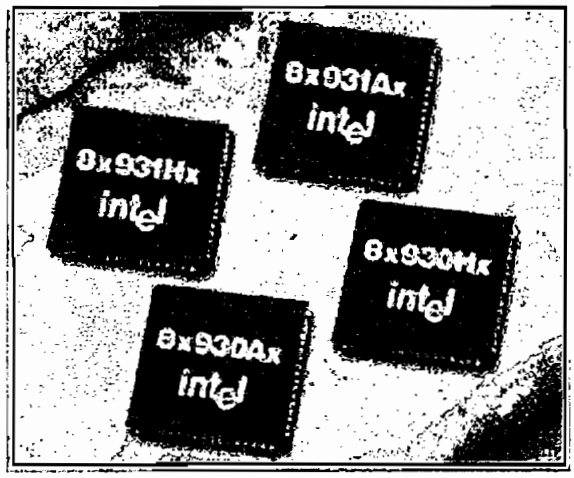



Figura 1.7: Chips Controladores USB

## 1.2.2 INTERFAZ FÍSICO

### 1.2.2.1 Especificación Eléctrica

El USB transfiere señales de información y energía eléctrica a través de 4 cables, cuya disposición se muestra en la Figura 1.8.

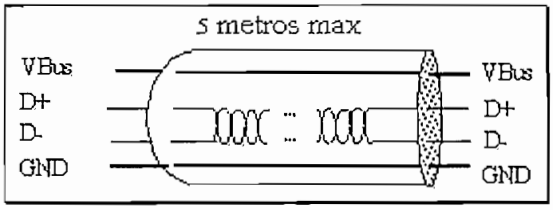



Figura1.8: Cable USB

La información es transmitida en los cables D+ y D- a través de las diferencias de tensión entre ellos ( USB usa un código de no retorno a cero invertido o NRZI) y entre segmentos comprendidos por un par de dispositivos USB, con rangos de velocidad de 480Mbps, 12Mbps y 1.5Mbps, para transmisiones de alta, media y baja velocidad respectivamente. Los tres modos de transmisión son controlados automáticamente por medio de los dispositivos USB de manera transparente al usuario. Los pulsos de reloj o sincronismo son transmitidos en la misma señal de forma codificada. Las diferencias de tensiones manejadas por el par D+ y D- son especificados en la tabla 1.2:

El desequilibrio de resistencia entre dos conductores de cualquier par no debe exceder el cinco por ciento (5%) del valor mostrado en la tabla 1.3:

| American Wire Gauge (AWG) | Ohms /100 Metro Máximo (Ohms) |
|---|---|
| 28 | 23.20 |
| 26 | 14.60 |
| 24 | 9.09 |
| 22 | 5.74 |
| 20 | 3.58 |

Tabla 1.3: Medidas Vs Resistencia

*1.2.2.2.2 Conectores USB*

Existen tres tipos de conectores del tipo ficha (conector y receptáculo) dentro del Bus Serial Universal y son:

a. **Conector Serie A:** Este conector está pensado para todos los dispositivos USB que trabajen sobre plataformas de PCs. Son bastantes comunes dentro de los dispositivos listos para ser empleados con host PCs. Es decir empleados en aquellos dispositivos en los que el cable externo, está permanentemente unido a los mismos, tales como teclados, ratones, dispositivos dedicados y Hubs (en el puerto downstream).

Presentan las cuatro patillas correspondientes a los cuatro conductores alineados en un plano (Ver figura 1.13).
El color característico es el blanco oscuro.

Los receptáculos se presentan en cuatro variantes: vertical, en ángulo recto, panel y apilado en ángulo recto así como para montaje pasamuros.

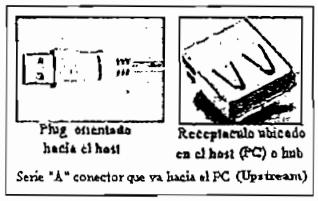



Figura 1.9: Conector Serie "A"

**b. Conector Serie B:** Este tipo de conectores presentan los contactos distribuidos en dos planos paralelos, dos en cada plano, y se emplean en los dispositivos (periféricos) que deban tener un receptáculo al que se pueda conectar un cable USB. Por ejemplo impresoras, scanner, y módems.

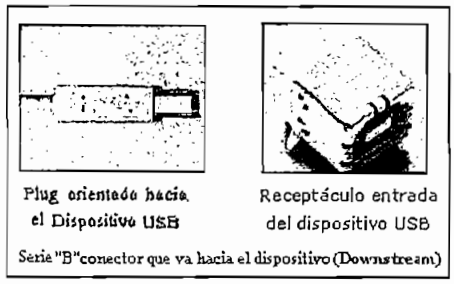



Figura 1.10: Conector Serie "B"

La descripción de medidas de construcción para los conectores USB (serie A o Serie B) se muestra en la figura 1.11.

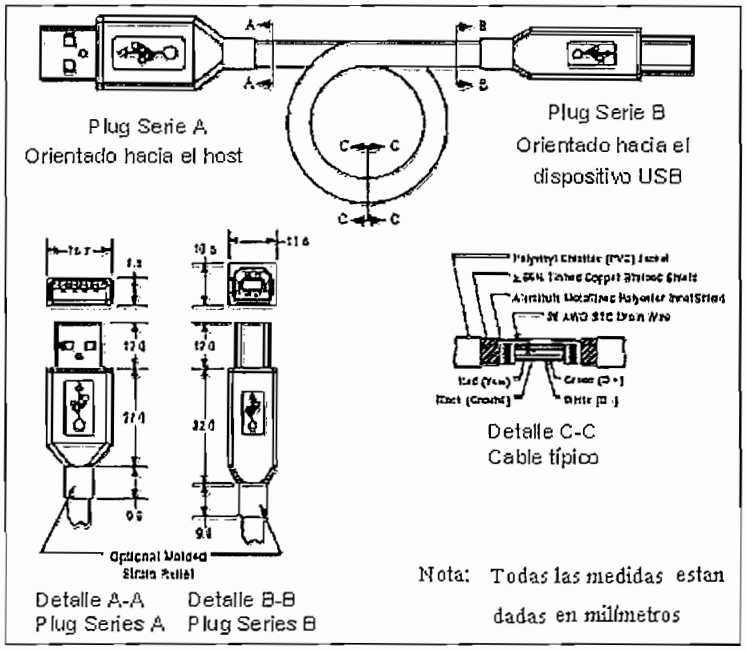



Figura 1.11: Ilustra el ensamblaje de un cable y conectores USB

**c. Conectores USB mini-B.-** La nueva familia de conectores USB mini-B ocupan tan sólo una octava parte del espacio de los conectores USB serie B estándar, son frecuentemente usadas para aplicaciones en equipos portátiles, tales como cámaras digitales, teléfonos móviles y PDAs. La nueva familia es totalmente compatible con la especificación actual USB 2.0. Estos Implementan 5 contactos, con uno de ellos reservado para aplicaciones futuras.

Las características del sistema mini-B, son: durabilidad, apantallado metálico y pines de masa para protección a las EMI (Interferencia Electromagnética) y sistema de retención con el fin de asegurar un buen acoplamiento. El conector soporta los 30V y 1.0A con referencia a los conectores USB convencionales.

Incluye conectores hembra para PC de inserción. Ambos presentan una altura de tan sólo 3.95mm es decir, aproximadamente 1/3 de la altura de los conectores hembra USB serie B convencionales para PC.
El conector aéreo USB mini-B se ofrece con una longitud de cable estándar de 1.0m, en configuración mini-B a USB-A, estando también disponible en otras longitudes.

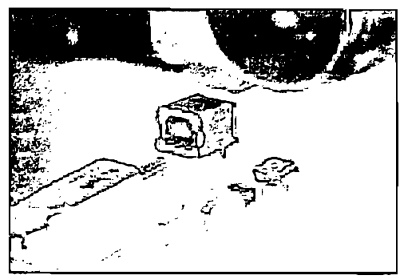


Figura 1.12: Conectores mini-B

*1.2.2.2.3 Asignación de pines del conector USB*

A continuación se detalla la asignación de pines ofrecidos al usuario a través del conector USB.

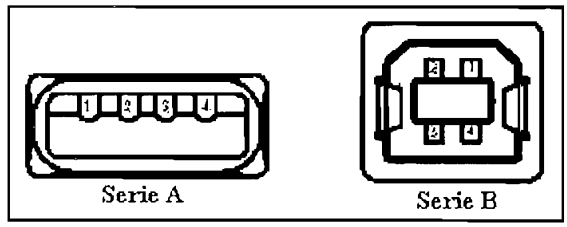



Figura 1.13: Conector USB

| PIN | Color de cable | Nombre | Función |
|---|---|---|---|
| 1 | Rojo | VBus (+5 V) | Alimentación |
| 2 | Blanco | D- | Transmisión y Recepción Diferencial - |
| 3 | Verde | D+ | Transmisión y Decepción Diferencial + |
| 4 | Negro | GND | Masa |
| | Gris (Opcional) | Escudo | Carcaza |

Tabla 1.4: Descripción de pines

### 1.2.3 MODELO LÓGICO FUNCIONAL USB

El diagrama de la Figura 1.14 ilustra el flujo de datos USB a partir de tres niveles lógicos: Capa Función (entre el Software Cliente y la Función), Capa de dispositivos USB (el Controlador USB y el dispositivo), y finalmente la capa física o capa interfase del dispositivo, donde la transmisión realmente sucede. Es importante entender que este modelo es muy parecido al modelo OSI(Modelo de Interconexión Abierto), el estándar de redes, y su comprensión radica en el hecho de que si bien existe un solo canal físico, pero los datos son manejados en cada punto por unidades homólogas, tal como si estuviesen sosteniendo una comunicación directa. Por esta razón se las denomina Capas Lógicas.

**Capa Función** Es el agente de transporte de datos que mueve la información entre el Software Cliente y el dispositivo. Existe un Software Cliente en el host, y un Software de Atención al mismo en cada una de las funciones o periféricos USB. En este nivel, el host se comunica con cada uno de los periféricos en alguna de las varias formas posibles de transmisión que soporta USB. El Software Cliente solicita a los dispositivos diversas tareas y recibe respuestas de ellos a través de esta capa.

**Capa del dispositivo USB** Es administrada por el Software del Sistema USB, y tiene la función de facilitarles las tareas particulares de comunicación a la capa superior. Administra la parte del periférico con la que la capa superior desea comunicarse, maneja la información de control y comando del dispositivo, etc. Su objetivo es permitir a la capa superior concentrarse en las tareas específicas

tendientes a satisfacer las necesidades del usuario, adicionalmente gestiona el control interno de los periféricos.

El acceso al bus es bajo la modalidad de Token, lo que involucra siempre complejidad de protocolos, especialmente si se manejan velocidades diferentes. Todos estos algoritmos y procesos son administrados por el Host USB, reduciendo la complejidad del periférico, y lo más importante, el costo final de los dispositivos USB.

**Capa física.-** Comprende los puertos físicos, el cable, los voltajes y señales, el hardware y funcionamiento del hardware. Esta capa tiene el objetivo de liberar a las capas superiores de todos los problemas relacionados a la modulación, voltajes de transmisión, saltos de fase, frecuencias y características netamente físicas de la transmisión.

Cabe indicar que en esta capa ocurre una transmisión de datos real mientras que en capas superiores la comunicación es virtual o lógica como se indica en la figura 1.14.

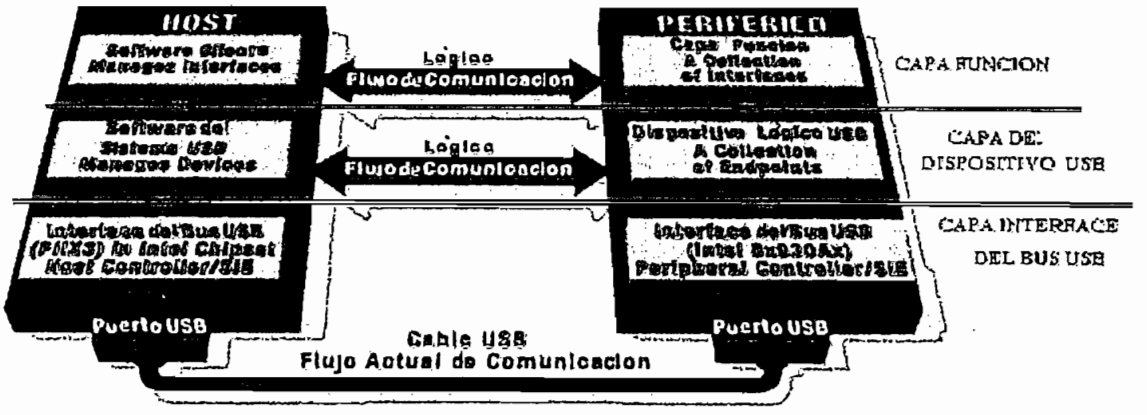



Figura 1.14: Modelo lógico de comunicaciones USB

## 1.3 PROTOCOLO DE TRANSFERENCIA DEL BUS USB

Toda transferencia de datos o transacción que emplee el bus, involucra al menos tres paquetes de datos. Cada transacción se da cuando el Controlador de Host

decide qué dispositivo hará uso del bus, para ello envía un paquete al dispositivo específico. Cada uno de los mismos tiene un número de identificación, otorgado por el Controlador del Host cuando el computador arranca o cuando un dispositivo nuevo es conectado al sistema. De esta forma, cada uno de los periféricos puede determinar si un paquete de datos (Token) es o no para sí. Una vez que el periférico solicitado recibe el permiso de transmitir, arranca la comunicación y sus tareas específicas; el mismo que informará al host con otro paquete que ya no tiene más datos que enviar y el proceso continuará con el siguiente dispositivo.

Este protocolo tiene un sistema muy eficiente de recuperación de errores, empleando uno de los modelos más seguros como es el CRC (Código de Redundancia Cíclica). Y puede estar implementado al nivel de software y/o hardware de manera configurable.

### 1.3.1 FORMATO DE PAQUETE

Los paquetes viajan en el bus como bits, los cuales son transmitidos desde el bit menos significativo (LSb) hasta el bit más significativo (MSb).

Para poder entender mejor el formato de los paquetes es necesario conocer los diferentes campos que conforma un paquete:

**1.3.1.1 Campo de sincronismo (SYNC).-** Todos los paquetes empiezan con un campo de sincronismo, el cual es una secuencia de códigos que generan un margen máximo de densidad de transmisión. El campo de sincronismo esta formado por la secuencia binaria de 8 bits "KJKJKJKK" en codificación NRZI. Este campo es usado solamente como un mecanismo de sincronización y no esta presente en los diagramas de paquetes. Los dos últimos bits en el campo de sincronismo (KK) son marcas que indican la finalización del campo de sincronismo e indica el inicio del identificador de paquete (PID).

**1.3.1.2 Campo identificador de Paquete (PID).-** El PID es una unidad de información que precede a cualquier paquete USB, consta de 8 bits (ver figura 1.16) y es empleado para indicar el tipo de paquete, formato del paquete y el

mecanismo de detección de errores empleado sobre la trama. La información aportada por PID es codificada en los cuatro primeros bits de su estructura, los restantes cuatro bits son empleados para propósitos de comprobación de posibles errores sobre la información de PID. PIDs son divididos en cuatro grupos codificados: data, hanshake, token y special con los primeros dos bits transmitidos y con los siguientes dos bits se indica el grupo al cual pertenece.

La tabla 1.5 indica en forma mas detallada los tipos de PID y la descripción de cada uno de ellos.

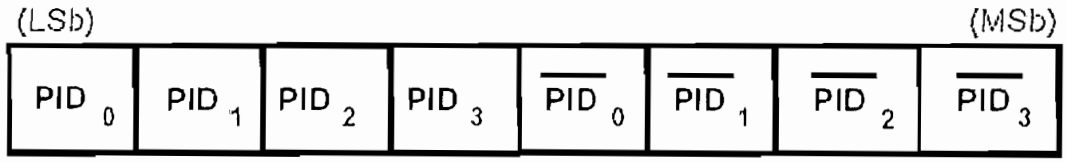



Figura 1.16: Campo PID

| Tipo de PID | Nombre PID | Codificación | Descripción |
|---|---|---|---|
| Data | DATA0 | 0011 | Paquete de datos PAR |
| | DATA1 | 1011 | Paquete de datos IMPAR |
| Handshake | ACK | 0010 | Recepción de paquete de datos libre de errores |
| | NAK | 1010 | El receptor no puede aceptar datos o el transmisor no puede enviar |
| | STALL | 1110 | Los dispositivos TX/RX detenidos o no pueden soportar un comando |
| Token | OUT | 0001 | Direcciones de dispositivo destinatario de los datos |
| | IN | 1001 | Direcciones de dispositivo transmisor de los datos |
| | SOF | 0101 | Indicador de comienzo y numeración de trama |
| | SETUP | 1101 | Direcciones de dispositivo destinatario de comandos de control |
| Special | PRE | 1100 | Preámbulo para la comunicación con dispositivos de baja velocidad |

Tabla 1.5: Tipos de identificadores de Paquetes (PID)

**1.3.1.3 Campo de Dirección.-** Es conocido como función endpoints y son direcciones que usan dos campos: el campo de dirección de función y el campo de endpoint.

*1.3.1.3.1 Campo dirección de función.-* Es un campo de 7 bits que indican el origen o el destino de los paquetes de datos dependiendo de los valores del campo PID. La figura presenta un campo de 6 bits (0-6) con los cuales se puede direccionar un total de 128 direcciones. El campo de direcciones es especificado por IN, SETUP, OUT y tokens.

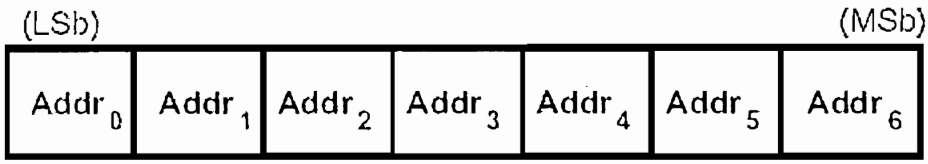



Figura 1.17: Campo de dirección

*1.3.1.3.2 Campo Endpoint.-* En la figura se presenta un campo endpoint que permite más flexibilidad de direccionamiento en las funciones en las cuales se requieran más endpoints, excepto en las transacciones en las cuales se requieren un endpoint direccionado como cero (Endpoint 0). Mas adelante se vera con mayor detalle lo que es un Endpoint.

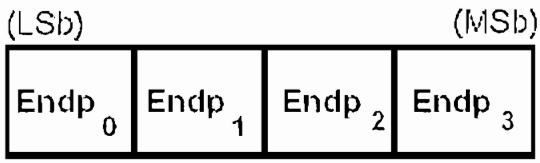



Figura 1.18: Campo Endpoind

**1.3.1.4 Campo de número de trama.-** El campo de número de trama viaja al inicio de cada trama y consiste en un campo de 11 bits que es incrementado por el host y esta en el rango de 000H hasta 7FFH.

**1.3.1.5 Campo de datos.-** El campo de los datos puede ir del cero a 1,023 bytes y debe ser un número entero de bytes. La figura muestra un formato de múltiples

bytes. El tamaño de los paquetes de datos varía con el tipo de transferencia (Ver tipo de transferencias)

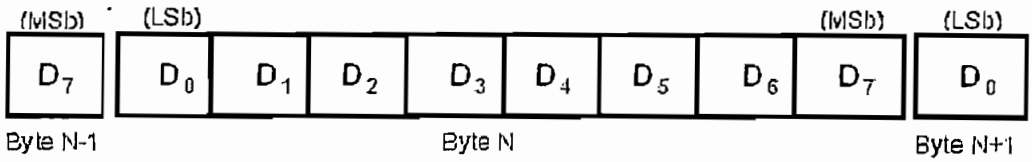



Figura 1.19: Formato del campo de datos

**1.3.1.6 Campo de Chequeo de redundancia Cíclica (CRC).-** Este campo incorporan unos mecanismos de detección de errores y de recuperación / retransmisión de datos. Este campo puede ser de 5 o 16 bits (depende del tipo de paquete) los cuales contienen los polinomios generadores $G(X) = X_5 + X_2 + 1$ y $G(X) = X_{16} + X_{15} + X_2 + 1$

### 1.3.2 TIPOS DE PAQUETES

En el dialogo entre el host y el dispositivo se emplean cuatro tipos de paquetes, el uso de cada una de ellas esta en función de tipo de comunicación: inicio de trama (SOF o start-of-frame), Data (intercambio de información), Handshake (negociación y estado de la comunicación) y Token (identificación de destinatario / origen).

#### 1.3.2.1 Paquete de inicio de trama o SOF

El paquete de inicio de trama (SOF o start-of-frame) es de 24 bits incluyendo el PID, un número de marco de 11 bits, y un CRC de 5 bits.

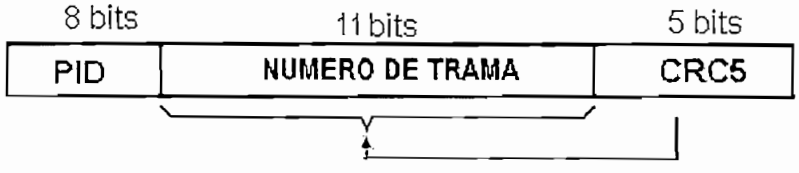



Figura 1.23: Formato de paquete Inicio de Trama

#### 1.3.2.2 Paquete de Datos

Un paquete de datos esta constituido por un PID, un campo en el cual se envían los datos y un CRC. Este último es generado sobre la porción de datos.

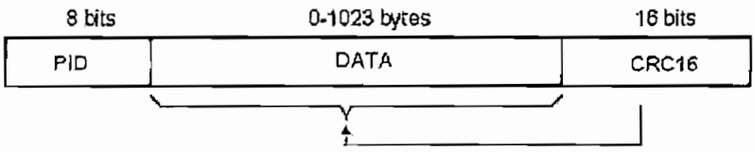



Figura 1.20: Formato del paquete de Datos

### 1.3.2.3 Paquete Handshake

Los paquetes de Handshake consisten únicamente en un PID, estos se emplean para reportar el estado de la transmisión y pueden devolver valores indicando la recepción satisfactoria de datos, comandos de aceptación o rechazo, control de flujo y situaciones de parada (ver tabla 1.5). Se identifican tres tipos de paquetes de Handshake:

**ACK:** Indica que el paquete de datos fue recibido correctamente sin errores.
**NAK:** Indica que un envío de datos no ha sido aceptado.
**STALL:** Indica que un dispositivo es incapaz de transmitir o recibir datos o que un comando de control es soportado.

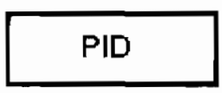



Figura 1.21: Formato de Paquete Handshake

### 1.3.2.4 Paquete Token

Un paquete Token está constituido por un PID donde se especifica el tipo de Token (IN, OUT, SETUP y STO), un campo ADDR, ENDP(numero de endpoint) y un CRC.

En transacciones OUT y SETUP los campos ADDR y ENDP identifican al destinatario que recibirá los subsiguientes paquetes de información (Data Packets).

En transacciones IN estos campos identifican al dispositivo USB que debe transmitir información.

## 1.3.3 ENDPOINT DEL DISPOSITIVO

Toda transmisión viaja desde o hacia un endpoint del dispositivo. El Endpoint es un buffer o un registro que almacena múltiples bytes. Típicamente este es un bloque de memoria distribuida en grupos de 8 bytes que se encuentra en el dispositivo. Los datos almacenados en un endpoint pueden ser datos recibidos, o datos esperando a ser trasmitidos. El host también tiene buffer para transmitir y para recibir datos, pero el host no tiene endpoint.

La especificación USB define el endpoint como "una única parte del dispositivo

Los token del tipo STO (Start-of-Frame) proporcionan información de sincronización, estos paquetes preceden a cada trama indicando su secuencia.

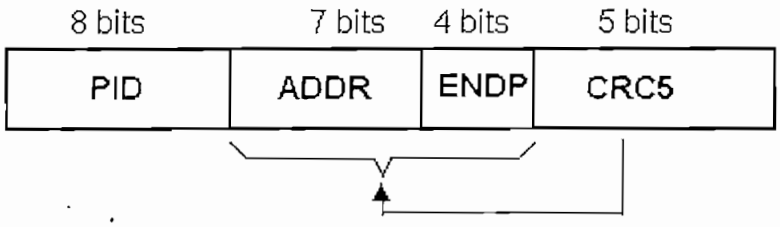



Figura 1.22: Formato de paquete Token

Para entender el protocolo encargado de la gestión de las comunicaciones del USB, es necesario conocer ciertas características de las transferencias de datos por USB como son endpoints y pipes del dispositivo.

### 1.3.3 ENDPOINT DEL DISPOSITIVO

Toda transmisión viaja desde o hacia un endpoint del dispositivo. El Endpoint es un buffer o un registro que almacena múltiples bytes. Típicamente este es un bloque de memoria distribuida en grupos de 8 bytes que se encuentra en el dispositivo. Los datos almacenados en un endpoint pueden ser datos recibidos, o datos esperando a ser trasmitidos. El host también tiene buffer para transmitir y para recibir datos, pero el host no tiene endpoint.

La especificación USB define el endpoint como "una única parte del dispositivo USB direccionable que es fuente o destino de la información en una comunicación entre el host y el dispositivo".

La única dirección requerida para cada endpoint consiste de un número endpoint y una dirección (IN y OUT). El número puede ir desde 00h hasta FFh y la dirección siempre es con respecto al host, es decir: dirección IN que representa los datos hacia el host y dirección OUT representa datos hacia el dispositivo. Cada dispositivo debe tener configurado un endpoint 0 como un endpoint de control.

En adición al Endpoint 0, un dispositivo de media velocidad puede tener hasta 30 endpoint adicionales (es decir desde 0 hasta 15, cada uno soportando direcciones IN y OUT). Un dispositivo de baja velocidad es limitado a dos endpoint adicionales con algunas combinaciones de direcciones por ejemplo: Endpoint 1 IN y Endpoint 1 OUT, o Endpoint 1 IN y Endpoint 2 OUT.

Cada transferencia en el bus contiene una dirección de dispositivo, para saber a que dispositivo va la información o bien que dispositivo a enviado datos; una dirección o número de endpoint para saber en que registro del dispositivo se debe almacenar la información enviada o en que registro esta la información que se quiere enviar y un código que indica la dirección del flujo de datos y si o no la transacción esta inicializando una transferencia de control. Los códigos son IN, OUT y Setup.

**IN:** Los datos viajan del dispositivo hacia el host y por lo tanto los datos a transmitir deberán estar dispuestos en endpoint de tipo IN.
**OUT:** Los datos viajan del host al dispositivo y los datos transmitidos serán almacenados en endpoint de tipo OUT.
**SETUP:** los datos van siempre desde el host hacia el dispositivo y contienen peticiones al dispositivo. Este tipo de transferencia será siempre atendida. Se trata de transferencias de control y siempre transmitirá por el endpoint 0.

Cuando un dispositivo recibe una transacción OUT o SETUP conteniendo la dirección del dispositivo, el hardware almacena los datos recibidos en una localidad apropiada para el endpoint y típicamente se produce una interrupción para atender a éstas. Una rutina al servicio de interrupción en el dispositivo procesa la información y hace lo que la transacción requiera. Cuando el dispositivo recibe una transacción IN conteniendo la dirección de este dispositivo, el dispositivo enviará datos al host si realmente debe hacerlo y la interrupción deberá hacer lo necesario para dejar listo al periférico para atender a la siguiente transacción IN.

### 1.3.4 PIPES

Un pipe es una asociación entre un endpoint de un dispositivo y el software en el host es decir que se crea un camino virtual entre el endpoint y el software del controlador en el host. El pipe representa la habilidad de mover los datos entre el software en el host vía un bufffer de memoria y un endpoint en un dispositivo (Ver figura 1.15).

El host establece brevemente pipes luego del encendido del sistema o cuando se adiciona un dispositivo para pedir la información de la configuración del dispositivo. Si el dispositivo es removido del bus el host elimina el pipe no utilizado.

En el proceso de enumeración (se vera más adelante), el dispositivo da información al host sobre los endpoints que tiene: número, tipos de transferencia, máximo número de datos que transmite.

Cuando se produce una transacción se informa previamente al host de las condiciones de la misma y si estas no son válidas con respecto a la configuración que tiene el host del dispositivo recogida en la enumeración, no establecerá el pipe necesario para la transmisión.

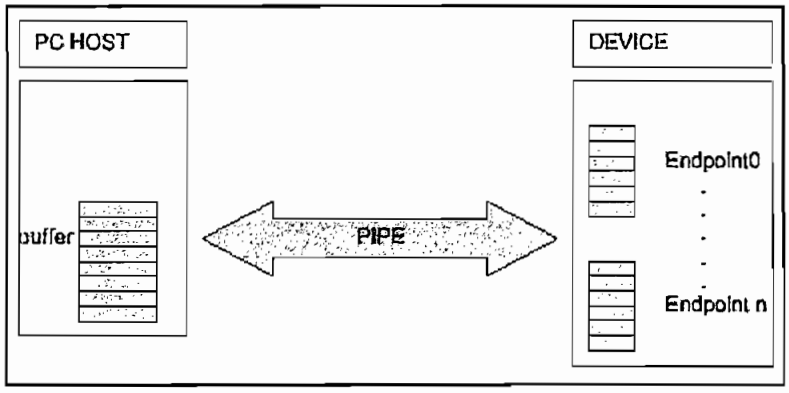



Figura 1.15: Endpoints y Pipes en el USB

### 1.3.5 FASES DEL PROTOCOLO

El protocolo se puede separar en tres fases bien diferenciadas en función de la tarea que realiza cada una. La figura siguiente muestra el esquema de estas tres fases y el orden de ejecución de cada una:

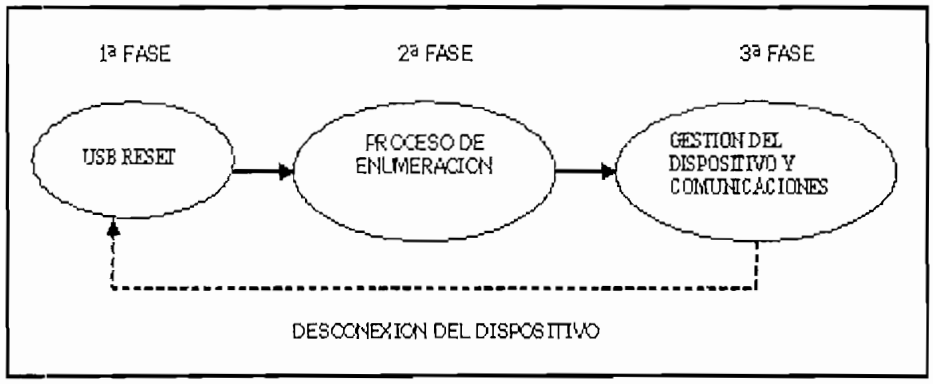



Figura 1.24: Fases del protocolo de transferencia

La descripción de cada una de estas fases es la siguiente:

#### 1.3.5.1 USB reset

Cuando el dispositivo es conectado, el host lo detecta y provoca una interrupción de reset en el dispositivo haciendo que configure los registros y punteros necesarios, para que se pueda proceder a la enumeración.

En esta fase el programa debe habilitar y configurar el endpoint 0 para recibir y contestar a transacciones de tipo *Setup* e inicializar todas las variables que posteriormente se utilizarán.

#### 1.3.5.2 Proceso de enumeración

Esta fase se produce después de la fase de USB Reset. El host debe reunir la información necesaria para que el sistema identifique al dispositivo y configure el tipo de comunicación que se producirá entre ambos y encuentre al driver que tiene que utilizar para establecer la comunicación. El proceso consiste en asignar una dirección al dispositivo y enviar una serie de peticiones para que el dispositivo mande información con el fin de establecer la comunicación. La información que debe mandar el dispositivo se estructura en registros o descriptores que

configuran al dispositivo y son transmitidos mediante transferencia de control y siempre por el endpoint 0.

La especificación USB requiere un número diferente de descriptores para proveer información necesaria para identificar un dispositivo, específicamente sus endpoints, y la función de cada endpoint. Las categorías generales de descriptores son: Dispositivo, Configuración, Interface, Endpoint, String.

**Descriptor de dispositivo:** Provee información general acerca del dispositivo como el número de fabricante, número de producto, número serial y el número de las diferentes configuraciones soportadas.

**Descriptor de configuración:** Proporciona información específica de la configuración del dispositivo como son los requerimientos de energía del dispositivo y cuantas interfaces diferentes pueden ser soportados cuando se esta en esta configuración.

**Descriptor de interface:** Detalla el número de endpoints usados en la Interface.

**Descriptor de endpoint:** Detalla los registros actuales para una función dada. Se almacena información acerca del tipo de transferencia soportada, dirección (In/Out), requerimientos de ancho de banda, e intervalo de taza de transferencia. Puede haber más de un endpoint en un dispositivo, y los endpoints pueden ser compartidos en diferentes interfaces.

**Descriptor de string:** Muchos de estos descriptores se referirán a diferentes descriptores String. Los descriptores String son usados para proporcionar información específica del vendedor o aplicación. Estos pueden ser opcionales y son codificados en un formato uni-código.
Con estos descriptores y otros, el sistema operativo, debe encontrar el driver que necesita para comunicarse con el dispositivo.

#### 1.3.5.3 Gestión del dispositivo y de las comunicaciones

Después de las dos fases anteriores en las que se configura y se establece la comunicación con el host se pasa a la tercera fase que es la que realmente se centra en gestionar la funcionalidad para la que ha sido diseñado el dispositivo.

En esta fase se controlan tanto las señales que llegan por el bus USB como las señales internas del dispositivo.

### 1.3.6 TIPOS DE TRANSFERENCIAS EN USB

La arquitectura USB comprende cuatro tipos diferentes de Transferencias de datos:

**1.3.6.1 Transferencias de Control** Es una transferencia no esperada, no se realiza periódicamente, sino que la realiza el software para inicializar una petición/respuesta de comunicación. Normalmente se utiliza para configuración y control de dispositivos y para manejo del bus.

Se desarrollan en 3 transacciones:

- La transacción de Configuración (Setup), en la que se envía al dispositivo un paquete que especifica la operación a ejecutar. Ocupa 8 bytes.
- Cero o más transacciones de datos, en las que se transfieren los paquetes de datos en el sentido indicado por la transacción de configuración. La información útil por paquete que pueden ser de 8, 16, 32 ó 64 bytes para endpoints de alta velocidad, y de 8 bytes para endpoints de baja velocidad.
- Transacción de estado, en la que el receptor informa del estado final de la operación.

Se procesan por medio de un mecanismo del "mejor esfuerzo" según el cual el controlador USB las va procesando en función del tiempo disponible en cada trama. Como mínimo se reserva el 10% del tiempo de trama, y se puede utilizar tiempo adicional siempre que las necesidades de los tráficos isócrono y de interrupción lo permitan.

Incorporan mecanismos de detección de errores (CRC) y de recuperación/retransmisión de datos.

1.3.6.2 **Transferencias Isocrónicas** Permite una comunicación periódica y continúa entre el sistema y el dispositivo. Usada en transmisión de información con ancho de banda y latencia garantizados, necesario para aplicaciones como audio, telefonía y vídeo. Este tipo de transferencia envía la señal de reloj encapsulada en los datos, mediante comunicaciones NRZI. Sólo son utilizables por dispositivos de velocidad alta y media.

La información útil por paquete puede oscilar entre 1 y 1,023 bytes.

En cada trama se transfiere un paquete por cada conexión isócrona establecida.

El sistema puede asignar como máximo el 90% del tiempo de trama para transferencias isócronas y de interrupción. Si el sistema ya tiene asignado un tiempo de trama de forma que no garantiza tiempo suficiente como para manejar una nueva conexión isócrona (transmitir un nuevo paquete por trama), simplemente no se establece la conexión.
Los posibles errores no se recuperan (la información que no llega a su tiempo, se descarta).

1.3.6.3 **Transferencias de Interrupción.-** Para transferencias de pocos datos, no periódicas, de baja frecuencia pero con unos ciertos límites de latencia.

Aseguran una transacción (paquete) dentro de un periodo máximo (los dispositivos de alta velocidad pueden solicitar entre 1 y 255 ms, y los de baja velocidad entre 10 y 255 ms de periodo máximo de servicio).

Incorpora detección de errores y retransmisión de datos.

La información útil por paquete puede oscilar entre 1 y 64 bytes para dispositivos de velocidad media y alta; y entre 1 y 8 bytes para dispositivos de baja velocidad.
El sistema puede asignar como máximo el 90% del tiempo de trama para transferencias isócronas y de interrupción. Si el sistema no puede garantizar tiempo suficiente como para manejar una nueva conexión de interrupción

(transmitir un nuevo paquete dentro del periodo máximo requerido), simplemente no se establece la conexión.

1.3.6.1 **Transferencias de Volumen (BulK)** No son transferencias periódicas. Se trata de paquetes de gran tamaño, usada en aplicaciones donde se utiliza todo el ancho de banda disponible en la comunicación. Estas transferencias pueden quedar a la espera de que el ancho de banda quede disponible. Para transferencias de grandes cantidades de datos con dispositivos asíncronos, como impresoras, escáneres, cámaras de fotos (foto fija), etc.

Sólo son utilizables por dispositivos de velocidad alta y media.

Se procesan por medio de un mecanismo del "mejor esfuerzo", en el que el sistema aprovecha cualquier ancho de banda disponible y en el momento en que esté disponible (en otras palabras, no se garantiza una latencia ni un ancho de banda mínimos). Se puede utilizar el tiempo de Trama reservado y no consumido por transferencias de Control (10%).

Incorporan mecanismos de control de errores para garantizar la entrega de datos. La información útil por paquete puede ser de 8, 16, 32 ó 64 bytes.

Estos 4 tipos de transferencias están disponibles como software que el sistema pone a disposición de los fabricantes de dispositivos, estando los drivers (manejadores) obligados a comunicarse con los dispositivos, única y exclusivamente a través de estos 4 interfaces de programación. Esto viene a significar que un manejador de dispositivo USB jamás accede directamente al hardware del dispositivo, y por otro lado significa que todos los dispositivos USB deben cumplir necesariamente unas especificaciones básicas comunes, ya que deben gestionar adecuadamente los tipos de transferencias que soportan. Adicionalmente, los dispositivos USB se agrupan en clases, de forma que todos los dispositivos de una misma clase cumplen además con las especificaciones de dicha clase, ya que la clase incide directamente en la manera en que el software interactúa con el dispositivo.

### 1.3.7 CONTROLADORES DE SOFTWARE (DRIVERS)

El driver USB de bajo nivel maneja la energía del dispositivo USB, su enumeración y varias transacciones USB. Atrás de este, el driver controlador host conversa directamente con el hardware USB en el PC. Los dos drivers son soportados por las actuales versiones de Windows y Linux, a los que no se los debe modificar.

Windows, del mismo modo como lo hace la especificación USB, segmenta los drivers en "clases", donde el hardware que está dentro de una misma clase tienen interfaces similares. Una clase define la línea de fondo de la especificación para un set de capacidades; todos los dispositivos requieren comparable tipo de soporte de software.

Un ejemplo es la Clase Human Interface Device HID (Clase para Dispositivo de Interface Humana), la cual soporta dispositivos como mouses, joysticks, teclados, y otros dispositivos dedicados. Otra es la Clase Monitor, que controla posición, espacio y alineación de video.

Las últimas versiones de los populares sistemas operativos traen una gran cantidad de controladores para una gama amplia de dispositivos USB, por lo tanto, lo más probable es que el mismo sistema operativo reconozca y configure el dispositivo de forma inmediata y transparente. Adicionalmente cada dispositivo que aparece contiene sus propios drivers para los diferentes sistemas operativos para el caso en que no exista este controlador en el sistema operativo. Por ejemplo para localizar un driver, Windows debe tener cargado un archivo especial de texto llamado INF que dice al sistema que driver debe utilizar. Una vez que Windows identifique el driver este se carga y pone al dispositivo listo para usarse.

### 1.3.8 CLASES USB

Una Clase USB es un grupo de dispositivos (o interfaces) con atributos o características similares. Las especificaciones para cada Clase permiten el desarrollo de dispositivos que pueden controlarse por medio de un manejador adaptativo, es decir, que se configura según la Clase reportada por el dispositivo.

Dos dispositivos (o interfaces) pertenecen a la misma Clase si por ejemplo utilizan una misma forma de comunicarse con el sistema, o si por ejemplo utilizan el mismo formato de datos.

Las Clases USB se usan principalmente para describir la manera en que los dispositivos (o interfaces) se comunican con el sistema, incluyendo los mecanismos de control y datos, y adicionalmente algunas Clases se usan para identificar en todo o en parte la funcionalidad del dispositivo (o interfaz). En este caso, la Clase se puede utilizar para identificar qué manejador debe controlar dicho dispositivo (o interfaz).

Adicionalmente, los dispositivos de una Clase pueden agruparse en Subclases, lo que facilita aún más el que los manejadores puedan explorar el bus y seleccionar todos aquellos dispositivos que pueda controlar.

## 1.4 LOGOTIPO UNIVERSAL USB


Figura 1.25: Logotipo universal USB

Para mayor detalle acerca de la Especificación Universal Serial Bus refiérase a la Especificación USB Rev 1.1. Que puede descargarse de la página web www.usb.org.

# CAPÍTULO II

# CAPÍTULO II

# CONSIDERACIONES PARA PROGRAMACIÓN DE LOS MICROCONTROLADORES ATMEL

## 1.1 INTRODUCCIÓN A LOS MICROCONTROLADORES.

Los microcontroladores están conquistando el mundo, están presentes en nuestra casa, en el trabajo y en nuestra vida en general. Las extensas áreas de aplicación de estos microcontroladores exigirán un gigantesco trabajo de diseño y fabricación. Aprender a manejar y aplicar microcontroladores solo se consigue desarrollando prácticamente diseños reales.

En el mercado existen un sinnúmero de microcontroladores de todo tamaño, tipo, forma, etc. Pero los microcontroladores Atmel están ganando un amplio mercado ya sea por su versatilidad, costo y lo que es mas importante provee un sinnúmero de herramientas para el desarrollo de proyectos.

La familia de microcontroladores Atmel es muy extensa, existiendo microcontroladores tanto para aplicaciones específicas como para aplicaciones personalizadas. De la amplia gama de microcontroladores fabricados por Atmel se a seleccionado los de la serie AT89C51/52, AT89C1051/2051 y los AVR del tipo PDIP (Plastic Dual in line Package) por su versatilidad y bajo precio.

## 1.2 DESCRIPCIÓN GENERAL DE LOS MICROCONTROLADORES

A continuación se hace una breve descripción de las características más importantes de los microcontroladores Atmel para los cuales se construirá el programador.

### 1.2.1 MICROCONTROLADORES ATMEL AT89C1051/2051 Y AT89C51/LV51/C52/LV52

Son microcontroladores CMOS de 8 bits, de alto rendimiento y bajo consumo de potencia. Poseen memoria FLASH borrable y reprogramable de 1, 2, 4 y 8Kbytes. Compactan un versátil CPU de 8-bit con la memoria Flash en un solo chip, haciendo que sea un microcontrolador que proporciona una solución muy-flexible y de bajo costo para muchas aplicaciones de control. Están diseñados con lógica estática para operaciones bajo frecuencia cero y soportan dos modos de ahorro de energía. En el modo inactivo detiene el CPU mientras permite que la RAM, el temporizador / contador, el puerto serie y el sistema de interrupción continúen funcionando. En el modo de apagado guardan el contenido de la RAM, detienen el oscilador y deshabilitan todas las otras funciones del chip hasta que se reinicie el hardware nuevamente.

En la tabla 2.1 se detallan las características más importantes de cada uno de estos microcontroladores.

| | Microcontrolador | | | | | |
|---|---|---|---|---|---|---|
| Descripción | AT89C1051 | AT89C2051 | AT89C51 | AT89LV51 | AT89C52 | AT89LV52 |
| Memoria Flash | 1 Kbytes | 2 Kbytes | 4 Kbytes | 4 Kbytes | 8 Kbytes | 8 Kbytes |
| Memoria RAM | 64 Bytes | 128 Bytes | 128 Bytes | 128 Bytes | 256 Bytes | 256 Bytes |
| Líneas de entrada y salida (I/O) | 15 | 15 | 32 | 32 | 32 | 32 |
| contador / temporizador de 16bit | 1 | 2 | 2 | 2 | 2 | 2 |
| comparador analógico | 1 | 1 | no | no | no | no |
| oscilador externo | si | si | si | si | si | si |
| reloj de seguridad | si | si | si | si | si | si |
| puerto serie | no | 1 | 1 | 1 | 1 | 1 |
| Voltaje de operación | 2.7 - 6V | 2.7 - 6V | 5V + 20% | 2.7 - 6V | 5V + 20% | 2.7 - 6V |
| Voltaje de habilitación programación | 11.5 - 12.5V | 11.5 - 12.5V | 11.5 - 12.5V | 11.5 - 12.5V | 11.5 - 12.5V | 11.5 - 12.5V |
| Rango de Frecuencia de trabajo | 0 - 24Mhz | 0 - 24Mhz | 0 - 24Mhz | 0 - 12Mhz | 0 - 24Mhz | 0 - 12Mhz |
| Costo local (No incluye IVA) | US$ 5 | US$ 5 | US$ 11 | US$ 11 | US$ 11 | US$ 11 |

Tabla 2.1: Características de los Microcontroladores ATMEL AT89Cxxxx

### 1.2.2 MICROCONTROLADORES ATMEL AVR

Son microcontroladores CMOS de 8 bits, basado en la arquitectura AVR RISC. Ejecuta las instrucciones en un ciclo de reloj, tienen un throughputs de 1 MIPS por MHz aproximadamente, permitiendo al diseñador del sistema optimizar el consumo de potencia versus la velocidad de procesamiento.

El AVR combina un conjunto de instrucciones con 32 registros de propósito general. Todos los 32 registros son directamente conectados a la unidad aritmética lógica (ALU), permitiendo que dos registros independientes puedan ser accesados en una simple instrucción, ejecutándose en un ciclo de reloj. La arquitectura resultante es más eficiente mientras logra un throughputs hasta 10 veces más rápida que un microcontrolador convencional CISC.

Poseen 3 modos de ahorro de energía seleccionadle por software. En el modo inactivo detiene el CPU mientras permite que la SRAM, el temporizador / contador, el puerto SPI y el sistema de interrupción continúen funcionando. En el modo de apagado guarda el contenido de los registros, detiene el oscilador y deshabilita todas las otras funciones del chip hasta que se reinicie el hardware nuevamente En el modo de ahorro de energía, el oscilador del temporizador continua trabajando, permitiéndole al usuario mantener una base del temporizador, mientras el resto del dispositivo está durmiendo.

La memoria Flash puede ser programada en el circuito a través del interfaz serie SPI o por un programador de memoria convencional.

Compactan un versátil CPU de 8-bit con una memoria Flash en un solo chip, haciendo que sea un microcontrolador que proporciona una solución muy-flexible y de bajo costo para muchas aplicaciones de control. Los microcontroladores AVR están apoyados con un soporte completo de programas y herramientas de desarrollo como: kits de evaluación, compiladores en C, ensambladores, programas de simulación y depuración.

Son varios los microcontroladores AVR pero se han seleccionado los microcontroladores más comunes en el mercado de los cuales se dan a conocer las características más importantes de cada una de ellas en la tabla 2.2.

| | Microcontrolador | | | | | | |
|---|---|---|---|---|---|---|---|
| Descripción | AT90S1200 | AT90S2313 | AT90S/LS2333 | AT90S/LS4433 | AT90S/LS4434 | AT90S8515 | AT90S/LS8535 |
| Memoria Flash | 1 Kbytes | 2 Kbytes | 2 Kbytes | 4 Kbytes | 4 Kbytes | 8 Kbytes | 8 Kbytes |
| Memoria SRAM | no | 128 Bytes | 128 Bytes | 128 Bytes | 256 Bytes | 512 Bytes | 512 Bytes |
| Memoria Eeprom | 64 bytes | 128 Bytes | 128 Bytes | 256 Bytes | 256 Bytes | 512 Bytes | 512 Bytes |
| Líneas de entrada y salida (I/O) | 15 | 15 | 20 | 20 | 32 | 32 | 32 |
| Reloj en tiempo real (RTC) | no | no | 1 | 1 | 1 | 1 | 1 |
| Contador/temporizador de 8-bit | 1 | 1 | 1 | 1 | 2 | 1 | 2 |
| Contador/temporizador de 16-bit | no | 1 | 2 | 2 | 1 | 1 | 1 |
| Canal ADC de 10-bit | 1 | 1 | 6 | 6 | 8 | 1 | 8 |
| 8,9 o 10-bit PWM | no | 1 | 1 | 1 | 1 | 1 | 1 |
| Puerto serial SPI | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| UART Programable | 1 | | 1 | 1 | 1 | 1 | 1 |
| Oscilador externo | si | si | si | si | si | si | si |
| Reloj de seguridad | si | si | si | si | si | si | si |
| Puerto serie | no | 1 | 1 | 1 | 1 | 1 | 1 |
| Voltaje de operación AT90S | 4 - 6V | 4 - 6V | 4 - 6V | 4 - 6V | 4 - 6V | 4 - 6V | 4 - 6V |
| Voltaje de operación AT90LS | 2.7 - 6V | 2.7 - 6V | 2.7 - 6V | 2.7 - 6V | 2.7 - 6V | 2.7 - 6V | 2.7 - 6V |
| Rango de Frecuencia de trabajo AT90S | 0 - 12Mhz | 0 - 10Mhz | 0 - 8Mhz | 0 - 8Mhz | 0 - 8Mhz | 0 - 8Mhz | 0 - 8Mhz |
| Rango de Frecuencia de trabajo AT90LS | 0 - 4Mhz | 0 - 4Mhz | 0 - 4Mhz | 0 - 4Mhz | 0 - 4Mhz | 0 - 4Mhz | 0 - 4Mhz |
| **Costo Local ( No Incluye IVA)** | **US$ 4** | **US$ 4** | **US$ 11** | **US$ 11** | **US$ 15** | **US$ 15** | **US$ 15** |

Tabla 2.2: Características de los Microcontroladores Atmel AVR

# 1.3. CONSIDERACIONES DE PROGRAMACIÓN DE LA MEMORIA FLASH DE LOS MICROCONTROLADORES ATMEL

La programación de los microcontroladores Atmel se puede hacer de dos formas: Programación en el circuito a través de un la interfaz serie SPI (In sytem program flash memory). O por un programador de memoria convencional que utiliza el modo de programación paralela.

En este capitulo nos centraremos en el modo de programación paralela.
La programación de algunos microcontroladores ATMEL es similar, razón por la cual se presentan la programación en grupos.

## 1.3.1 CONSIDERACIONES DE PROGRAMACIÓN DE LA MEMORIA FLASH DE LOS MICROCONTROLADORES AT89C1051 Y AT89C2051

El AT89C1051 y AT89C2051 poseen un contador interno de direcciones de la memoria el cual se carga en 000H en el borde creciente de RST y se incrementa cuando se aplica un pulso positiva en el pin XTAL 1

La interfaz de programación acepta un voltaje alto (12V) para habilitar el modo de programación. Se recomienda conectar las señales apropiadas en los diferentes pines como se especifica en las figuras 2.1 y considerar los tiempos de programación como se indica en la figura 2.2.

### 13.1.1 Algoritmo de programación

Para programar el AT89C1051 y el AT89C2051 es recomendado seguir el siguiente algoritmo.

1. Secuencia de encendido
   Aplicar 5V entre los pines Vcc y GND
   Fijar RST y XTAL1 a GND
2. Fijar Pin RST a "1"
   Fijar Pin P3.2 a "1"

3. Aplicar las combinaciones apropiadas de niveles lógicos "1" y "0" en los pines P3.3, P3.4, P3.5, P3.7 según la tabla 2.3 para seleccionar uno de los modos de operación.

Programación y verificación de las localidades de memoria.

4. Aplicar el primer dato en la localidad 000H a través de P1.0 hasta P1.7
5. Fijar RST a 12V para habilitar la programación.
6. Pulse P3.2 una vez para programar un byte en la EPROM o en el bit de bloqueo. El tiempo de escritura del byte dura aproximadamente 1.2 ms.
7. Para verificar el dato programado, hay que cambiar el voltaje del pin RST de 12V a 5V, luego poner los pines P3.3 a P3.7 de acuerdo a la tabla 2.3. Los datos de salida pueden ser leídos en el puerto P1
8. Para programar un byte de la próxima localidad de memoria poner un pulso en XTAL1. Con esto se consigue que se incremente la siguiente localidad de memoria.
9. Repetir los pasos 5 hasta el 8, cambiando datos e incrementando las direcciones hasta finalizar el cargado de datos.
10. Para terminar la grabación de debe seguir la siguiente secuencia de apagado:
    Fijar XTAL de "1" a "0"
    Fijar RST a "0"
    Quitar la alimentación (VCC).

Nota: Las siguientes señales indican si una grabación fue exitosa o no.

Data Polling: Una de las características de estos micros es el Data Polling que indica el fin de un ciclo de escritura. Al culminar la escritura del dato se tiene una señal que es el complemento del bit (dato) escrito en el Pin P1.7 con el cual se indica que el dato fue grabado satisfactoriamente.

Ready/Busy: El proceso de programación del byte también puede ser monitoreado por la señal de salida RDY/BSY (pin P3.1). El pin P3.1 se pone en bajo antes de que P3.2 se ponga en alto durante la programación indicando que esta ocupado. Y P3.1 se pone en alto cuando la programación del byte a finalizado es decir indica que la programación esta lista.

### 1.3.1.2 Verificación del Programa:

Si los bits de seguridad LB1 y LB2 no han sido programados, los datos programados pueden leerse a través de las líneas de dirección y línea de datos para la comprobación mediante los siguientes pasos:

1. Iniciar el contador interno de la localidad de memoria a 000H cambiando el estado de "0" a "1" en RST
2. Aplicar las apropiadas señales de control para seleccionar el modo de lectura (ver tabla 2.3).
3. Aplicar un pulso en XTAL1 para incrementar la dirección interna del contador.
4. Leer el próximo dato en el puerto P1.
5. Repetir los pasos 3 y 4 hasta finalizar la lectura de la memoria.

Nota: Los bits de bloqueo no pueden ser verificados directamente

### 1.3.1.4 Borrado del Chip

La memoria Flash puede ser borrado eléctricamente utilizando las adecuadas señales de control como se indica en la tabla 2.3 y con la puesta de P.2 en bajo ("0") durante 10ms. Todas las localidades de memoria son puestas en "1" en el borrado.

Nota: El Chip debe ser borrado antes de reprogramar la memoria.

| Modo | | RST/Vpp | P3.3/PROG | P2.6 | P2.7 | P3,6 | P3,7 |
|---|---|---|---|---|---|---|---|
| Escritura | | 12V | | 0 | 1 | 1 | 1 |
| Lectura | | 5V | 1 | 0 | 0 | 1 | 1 |
| Escritura de los Bits de seguridad | Bit-1 | 12V | | 1 | 1 | 1 | 1 |
| | Bit-2 | 12V | | 1 | 1 | 0 | 0 |
| Borrado | | | 1 | 1 | 0 | 1 | 1 |
| Tipo de microcontrolador | | 5V | 1 | 0 | 0 | 0 | 0 |

Tabla 2.3: Modos de programación

### 1.3.1.4 Lectura de los bytes descriptores

Los bytes descriptores se leen por el mismo procedimiento como en la comprobación normal de las localidades de memoria 000H, 001H, y 002H, excepto que P3.5 y P3.7 deben ser puestos en "0".

Al verificar estas localidades de memoria los valores retornados son los siguientes:

(000H) = 1EH Indica que fue fabricado por Atmel.

(001H) = 11H indica que es un 89C1051.

(001H) = 21H indica que es un 89C2051.

En la figura 2.1 se indican el diagrama de conexión de los diferentes pines de los microcontroladores en la fase de escritura y lectura de la memoria flash. En la figura 2.2 se muestran las formas de onda de las diferentes señales y el tiempo de duración de cada una de ellas tanto para programación y verificación de la memoria flash.

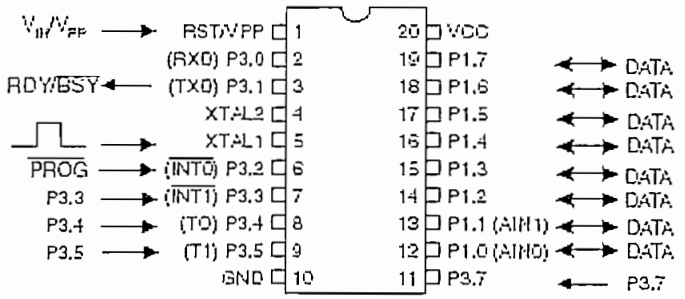



Figura 2.1: Interfaz de programación y Verificación de la memoria Flash

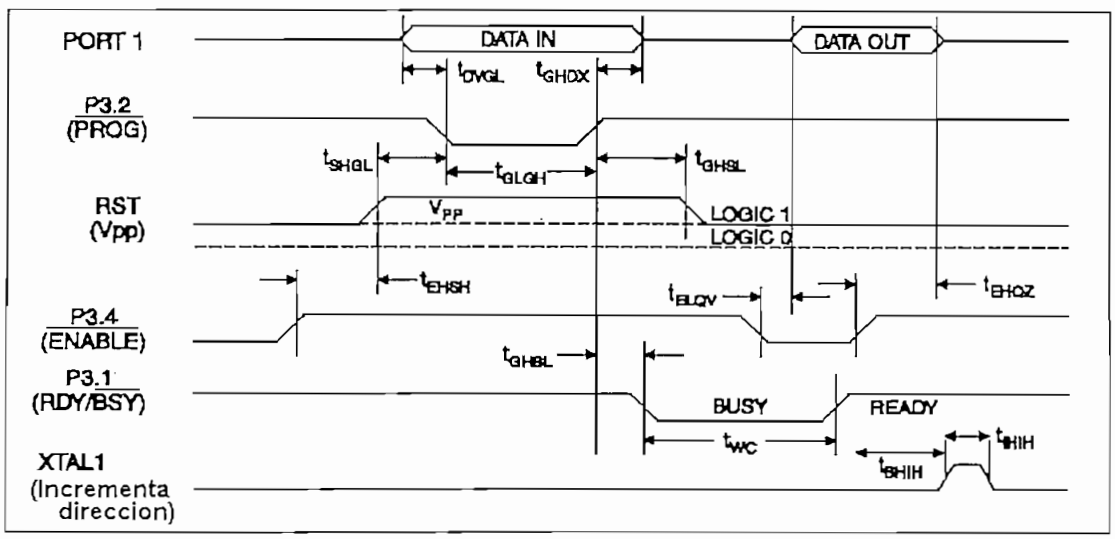



Figura 2.2: Formas de onda de la programación y verificación de la memoria Flash

## 1.3.2 CONSIDERACIONES DE PROGRAMACIÓN DE LA MEMORIA FLASH DE LOS MICROCONTROLADORES AT89C51, AT89C51, AT89LV51 Y AT89LV52

Los microcontroladores AT89C51, AT89C52, AT89LV51 y AT89LV52 son programados byte por byte en el modo de programación. En el borrado de la memoria Flash todas las localidades de memoria son puestas en 1.

Para la programación requieren de un oscilador de 3 - 24 MHZ (serie AT89CXX) y de 3 - 12 MHZ (serie AT89LVXX) debido a que el bus interno del microcontrolador esta siendo utilizada para transmitir direcciones y datos a registros apropiados.

Algunos pines no son expresados por el número de pin, si no por nombres de las señales que actúan, razón por la cual se recomienda conectar de acuerdo con la figura 2.3 y considerar los tiempos de programación como se indica en la figura 2.4.

### 1.3.2.1 Algoritmo de programación

Antes de programar tomar en cuenta que las direcciones, datos y señales de control deben estar puestos acorde a la tabla 2.4 y la figuras 2.3.

Para programar es recomendado seguir la siguiente secuencia.

1. Ingresar la dirección de la localidad deseada en las líneas de dirección.
2. Ingresar el dato en las líneas de Datos.
3. Activar correctamente las combinaciones de señales de control.
4. Pasar EA/VPP a 12V para el modo de programación.
5. Pulsar ALE/PROG una vez para programar el byte. El ciclo de escritura del byte típicamente toma un tiempo de 1.5ms.
6. Repetir el paso 1 hasta el 5 cambiando las direcciones y datos hasta completar hasta la localidad requerida.

Nota: Las siguientes señales indican si una grabación fue exitosa o no

Data Polling: Una de las características de estos micros es el Data Polling que indica el fin de un ciclo de escritura. Al culminar la escritura del dato se tiene una

señal que es el complemento del bit (dato) escrito en el Pin P0.7 con el cual se indica que el dato fue grabado satisfactoriamente.

Ready/Busy: El proceso de programación del byte también puede ser monitoreado por la señal de salida RDY/BSY (pin P3.4). El pin P3.4 se pone en bajo antes de que ALE se ponga en alto durante la programación indicando que esta ocupado. P3.4 se pone en alto cuando la programación del byte a finalizado es decir indica que la programación esta lista.

### 1.3.2.2 Verificación del Programa:

1. Fijar las respectivas señales de acuerdo a la tabla 2.4.
2. Fijar ALE/PROG y EA/VPP en alto.
3. Aplicar la dirección de la localidad del dato deseado a en las líneas de dirección.
4. Copiar el dato del puerto P0.
5. Repetir los pasos 3 y 4 hasta finalizar la lectura de la memoria.

### 1.3.2.3 Borrado del Chip

La memoria Flash puede ser borrado eléctricamente utilizando las adecuadas señales de control como se indica en la tabla 2.4 y con la puesta de ALE/PROG en bajo ("0") durante 10ms. Todas las localidades de memoria son puesta en "1" en el borrado.

Nota: El Chip debe ser borrado antes de reprogramar la memoria.

| Modo | | RST | PSEN | EA/VPP | P2.6 | P2.7 | P3,6 | P3,7 |
|---|---|---|---|---|---|---|---|---|
| Escritura | | 1 | 0 | 5V/12V | 0 | 1 | 1 | 1 |
| Lectura | | 1 | 0 | 5V | 0 | 0 | 1 | 1 |
| Escritura de Bit de seguridad | Bit-1 | 1 | 0 | 5V/12V | 1 | 1 | 1 | 1 |
| | Bit-2 | 1 | 0 | 5V/12V | 1 | 1 | 0 | 0 |
| | Bit-3 | 1 | 0 | 5V/12V | 1 | 0 | 1 | 0 |
| Borrado | | 1 | 0 | 5V/12V | 1 | 0 | 0 | 0 |
| Tipo de | microcontrolador | 1 | 0 | 5V | 0 | 0 | 0 | 0 |

Tabla 2.4: Modos de programación

### 1.3.2.4 Lectura de los bytes descriptores.

Los bytes descriptores se leen por el mismo procedimiento como en la comprobación normal de las localidades de memoria 000H, 001H, y 002H, excepto que P3.6 y P3.7 deben ser puestos en "0".

Al verificar estas localidades de memoria los valores retornados son los siguientes:

(030H) = 1EH Indica que fue fabricado por Atmel.

(031H) = 51H indica que es un 89C51.

(031H) = 61H indica que es un 89LV51.

(031H) = 52H indica que es un 89C52.

(031H) = 62H indica que es un 89LV52.

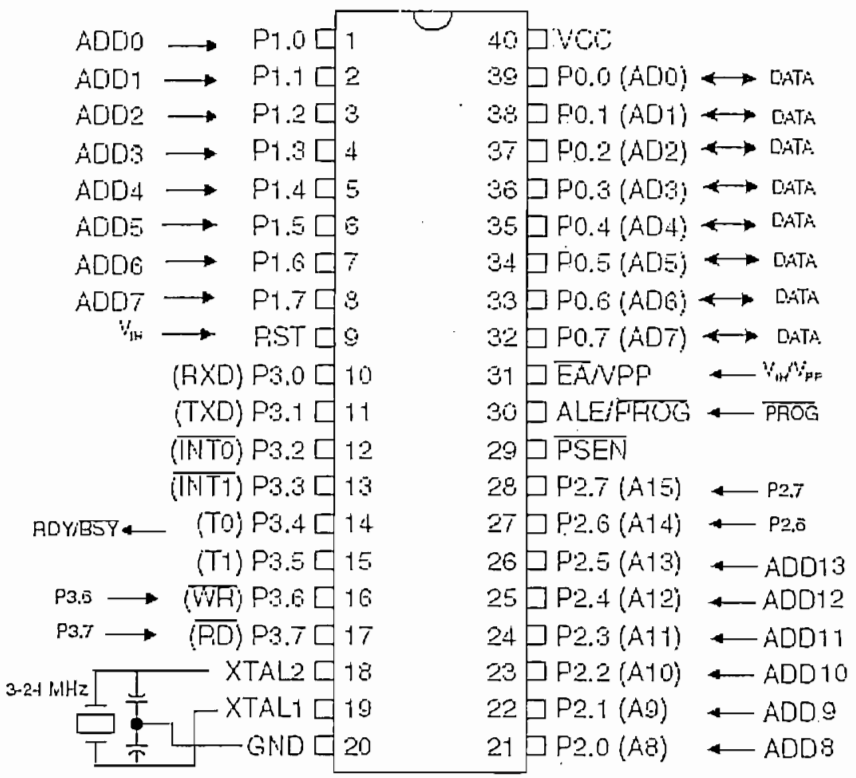



Figura 2.3: Interfaz de Programación y verificación de la memoria Flash

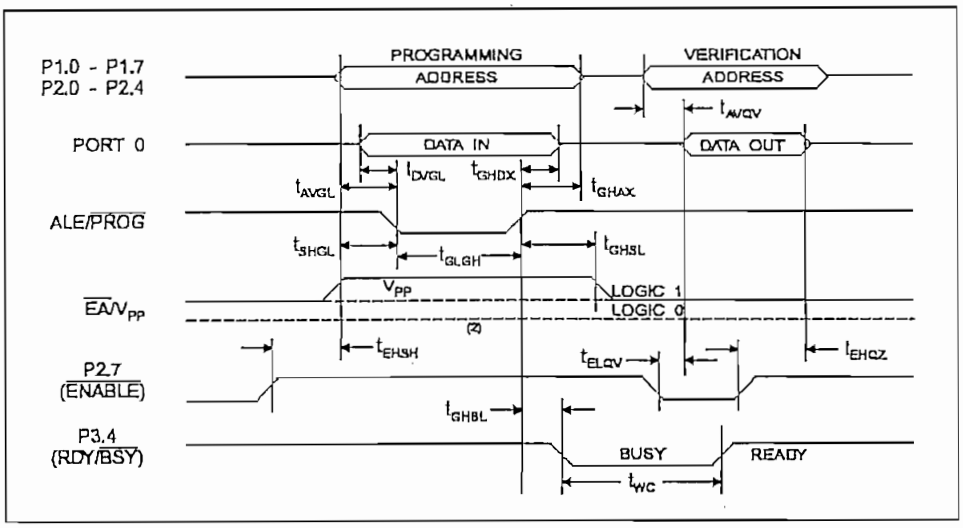



Figura 2.4: Formas de onda de la programación y verificación de la memoria Flash

## 1.3.3 CONSIDERACIONES DE PROGRAMACIÓN DE LOS MICROCONTROLADORES ATMEL AVR

### 1.3.3.1 Programación de los Bits de seguridad

Los microcontroladores AVR proporcionan dos bits de seguridad (Lock bits), estos puede ser no programados ("1") o programados ("0") para obtener los rasgos adicionales listados en la Tabla 2.5. Los bits de bloqueo sólo pueden borrarse con el comando de borrado del chip.

| Bits de seguridad de la memoria | | | Tipo de protección |
|---|---|---|---|
| Modo | LB1 | LB2 | |
| 1 | 1 | 1 | Habilitada características de no bloqueo de memoria. |
| 2 | 0 | 1 | Programación de memoria Flash y EEPROM es desactivado. |
| 3 | 0 | 0 | Igual que modo 2 y se desactiva la verificación. |

Tabla 2.5. Bit de seguridad para los diferentes modos de protección

#### *1.3.3.1.1 Fuse Bits*

El AT90S4434/8535 tiene dos Fuse bits el SPIEN y FSTRT.

Cuando el SPIEN es programado es decir en "0", se habilitada la descarga de datos y la programación serial. El valor por defecto de este bit es "0". Este bit es no accesible en el modo de programación serial.

Cuando el FSTRT es programado ("0"), se selecciona el tiempo de encendido corto. El valor por defecto de este bit es 1.

El estado del los Fuse bits no afecta en el borrado del chip.

*1.3.3.1.2 Bytes Descriptores (Signature Bytes)*

Todos los microcontroladores Atmel tienen un código de tres bytes que identifican al dispositivo. Estos bytes pueden ser leídos en los dos modos de programación, tanto en el modo serial como en el modo paralelo.

Los tres bytes residen en localidades de memoria separadas

- Para el AT90S1200 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 90H (indica 1K bytes de memoria Flash).
  3. 002H: 01H (indica que es un dispositivo AT90S2333 cuando el byte 001H es 90H).
- Para el AT90S2333 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 91H (indica 2K bytes de memoria Flash).
  3. 002H: 05H (indica que es un dispositivo AT90S2333 cuando el byte 001H es 92H).
- Para el AT90S4433 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 92H (indica 4K bytes de memoria Flash)
  3. 002H: 03H (indica que es un dispositivo AT90S4433 cuando el byte 001H es 92H).
- Para el AT90S4434 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 92H (indica 8K bytes de memoria Flash).
  3. 002H: 01H (indica que es un dispositivo AT90S4434 cuando el byte 001H es 92H).

- Para el AT90S8515 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 93H (indica 8K bytes de memoria Flash).
  3. 002H: 01H (indica que es un dispositivo AT90S8535 cuando el byte 001H es 93H).
- Para el AT90S8535 estos son:
  1. 000H: 1EH (indica fabricado por Atmel).
  2. 001H: 93H (indica 8K bytes de memoria Flash).
  3. 002H: 03H (indica que es un dispositivo AT90S8535 cuando el byte 001H es 93H).

Nota: Cuando los bits de bloque son programados en el modo 3, los bytes de descripción no pueden ser leídos en el modo serial. En el caso en que se lean estas localidades de memoria este debe devolver valores de: $00, $01 y $02.

### 1.3.3.2 Programación de las memorias Flash

Los microcontroladores Atmel AVR ofrecen 1K/2K/4K/8K bytes de memoria flash (in-system programmable Flash) y 64/128/256/512 bytes de memoria EEPROM.

Para la memoria flash y EEPROM de los AVR el estado $FF de todas las localidades de memoria significa que estas están borradas y listas para ser programados. Estos dispositivos soportan un voltaje alto (12V) en el modo de programación paralela y un voltaje bajo para el modo de programación serial.

Los +12V son usados solamente para habilitar la programación y ninguna corriente de importancia es manejado por este pin.

El programa y la memoria de datos en el AVR son programados byte-por-byte en cada modo de programación.

Durante la programación, el voltaje suministrado debe estar de acuerdo con la Tabla 2.6.

| Dispositivo | Programación paralela |
|---|---|
| AT90S2333 | 4.5 – 5.5 V |
| AT90LS2333 | 4.5 -5.5 V |
| AT90S4433 | 4.5 -5.5 V |
| AT90LS4433 | 4.5 -5.5 V |
| AT90S4434 | 4.5 -5.5 V |
| AT90LS4434 | 4.5 -5.5 V |
| AT90S8535 | 4.5 -5.5 V |
| AT90LS8535 | 4.5 -5.5 V |

Tabla 2.6. Suministro de Voltaje durante la Programación.

### 1.3.3.2 Algoritmo de programación paralela

Aquí se describe cómo programar y verificar el programa en la memoria Flash, y de igual forma los bits de seguridad y Fuse bits.

*1.3.3.3.1 Nombres de las señales (Signal Names)*

Algunos pines del AVR son referenciados por nombres de las señales que describen su función durante la programación paralela (Ver Figura 2.5 y Tabla 2.7). Los Pines no descritos en la tabla 2.7 son referenciados por el nombre del pin.

Los pines XA1/XA0 determinan la acción ejecutada cuando en el pin de XTAL1 se da un pulso positivo. Los bits codificados se muestran en la Tabla 2.8.

Al pulsar WR o OE, el comando cargado determina la acción ejecutada. El comando es un byte dónde los diferentes bits son funciones asignadas como se muestra en Tabla 2.9.

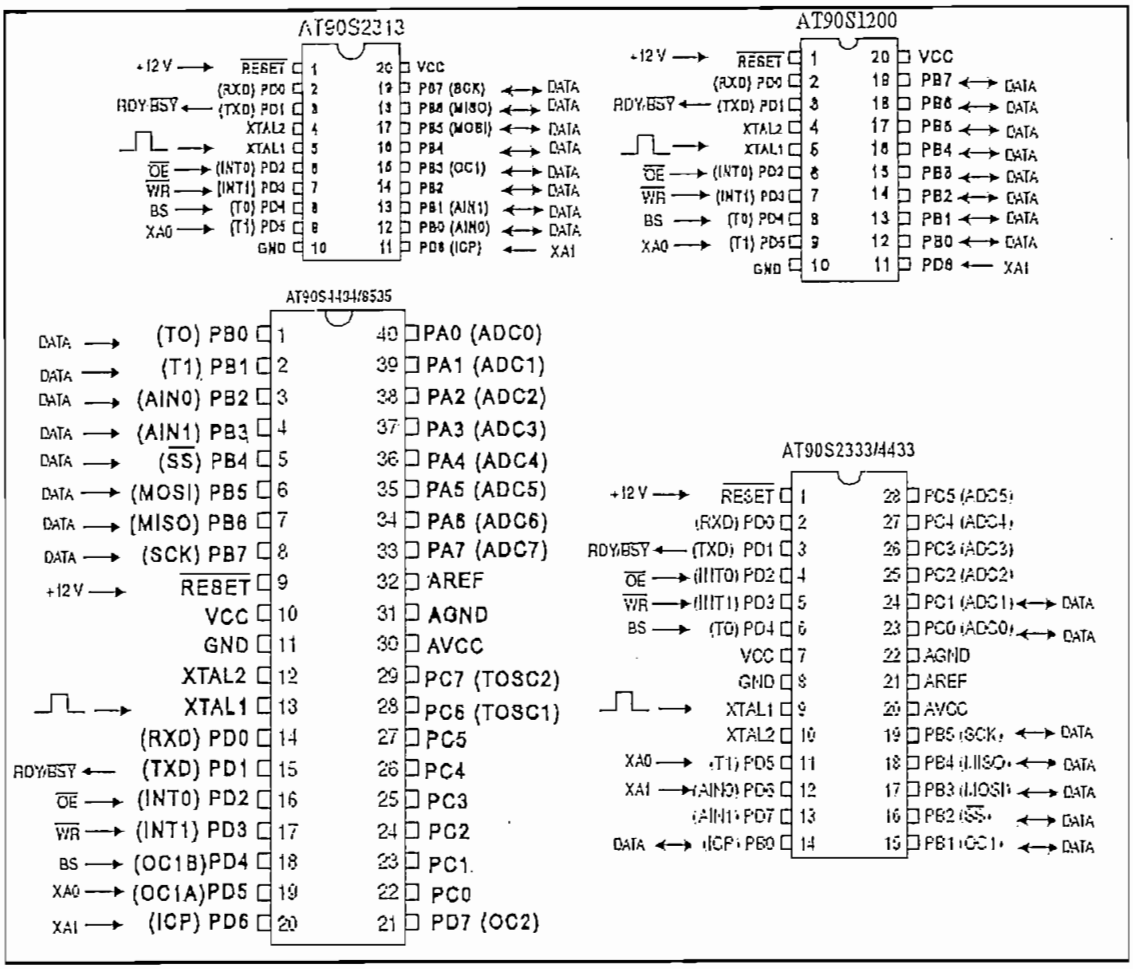



Figura 2.5. Modo de Programación Paralela

| Nombre de la señal | Nombre del Pin | I/O | Función |
|---|---|---|---|
| RDY/BSY | PD1 | O | 0: Dispositivo esta ocupado programando<br>1: Dispositivo esta listo para nuevo comando |
| OE | PD2 | I | Salida habilitada(activa en bajo) |
| WR | PD3 | I | Pulso de escritura (Activa en bajo) |
| BS | PD4 | I | Selección de byte(0: selecciona el byte bajo; 1: selecciona el byte alto) |
| XA0 | PD5 | I | XTAL acción Bit 0 |
| XA1 | PD6 | I | XTAL acción Bit 1 |
| DATA | PB7-0 | O | Bus de Datos Bi-direccional(salida cuando OE esta en bajo) |

Tabla 2.7 Asignación de Nombre a los diferentes Pines en el modo de programación.

| XA1 | XA1 | Acción cuando XTAL 1 es pulsado |
|---|---|---|
| 0 | 0 | Carga dirección en la memoria Flash o EEPROM (Dirección alta o baja es determinada por BS) |
| 0 | 1 | Carga datos (byte de datos altos o bajos para Flash, determinado por BS) |
| 1 | 0 | Carga comando |
| 1 | 1 | No acción, Inactivo |

Tabla 2.8. Codificación de los bits XA1 y XA0

| Comando byte | Comando Ejecutado |
|---|---|
| 1000 0000 | Borrado del Chip (Chip Erase) |
| 0100 0000 | Escribe Fuse Bits (Write Fuse Bits) |
| 0010 0000 | Escribe Bits de Bloqueo (Write Lock Bits) |
| 0001 0000 | Escribe Flash (Write Flash) |
| 0001 0001 | Escribe EEPROM (Write EEPROM) |
| 0000 1000 | Lee los bytes descriptores(Read Signature Bytes) |
| 0000 0100 | Lee bit de Bloqueo y Fuse Bits(Read Lock and Fuse Bits) |
| 0000 0010 | Lee la memoria Flash(Read Flash) |
| 0000 0011 | Lee la memoria EEPROM (Read EEPROM) |

Tabla 2.9. Comando Byte codificado en Bits

*1.3.3.3.2 Modo de ingreso a programación.*

El siguiente algoritmo pone al dispositivo en modo de programación paralela:

1. Aplicar el suministro de voltaje según la Tabla 2.6, entre VCC y GND.
2. Poner el RESET y BS fijo a "0" y esperar 100 ns por lo menos.
3. Aplicar el voltaje (11.5 - 12.5V) en RESET. Cualquier actividad en BS dentro de los 100 ns después de aplicar los +12V en RESET, causará que el dispositivo no ingrese en el modo de programación.

*1.3.3.3.3 Borrado del Chip.*

El comando de borrado del chip debería borrar la memoria Flash y EEPROM y los bits de bloqueo. Los bits de bloqueo no son puestos en cero hasta que la memoria Flash y EEPROM sean completamente borrados. Los Fuse bits no son

- Cargando dato en el byte bajo
    1. Fijar XA1, XA2 a "01" (esto permite cargar la dirección).
    2. Fijar DATA = Dato del byte bajo ($00 - $FF).
    3. Dar un pulso positivo en XTAL 1 (esto carga el dato en el byte bajo).
- Escritura del byte bajo
    1. Fijar BS a "0". Esto selecciona dato bajo.
    2. Fijar WR un pulso negativo. Esto inicia la programación del dato. RDY/BSY pasa a "0".
    3. Esperar hasta que RDY/BSY se ponga en alto para poder continuar con el próximo Byte.
- Cargando dato en el byte alto
    1. Fijar XA1, XA2 a "01" (esto permite cargar la dirección).
    2. Fijar DATA = Dato del byte bajo ($00 - $FF).
    3. Dar un pulso positivo en XTAL 1 (esto carga el dato en el byte bajo).
- Escritura del byte alto
    1. Fijar BS a "1". Esto selecciona dato alto.
    2. Fijar WR un pulso negativo. Esto inicia la programación del dato, RDY/BSY pasa a "0".
    3. Esperar hasta que RDY/BSY se ponga en alto para poder continuar con el próximo Byte.

Para la programación ver forma de ondas de la figura 2.6, 2.7 y 2.8

El comando cargado y la dirección son retenidos en el dispositivo durante la programación.

Para una programación eficaz, lo siguiente debe ser considerado:

- El comando sólo necesita ser cargado una sola vez cuando se escribe o lee múltiples localidades de memoria.
- La dirección del byte alto sólo necesita ser cargado una sola vez antes de programar una nueva página de las 256-palabras en la memoria Flash.
- Saltar la escritura de los datos con valores $FF.

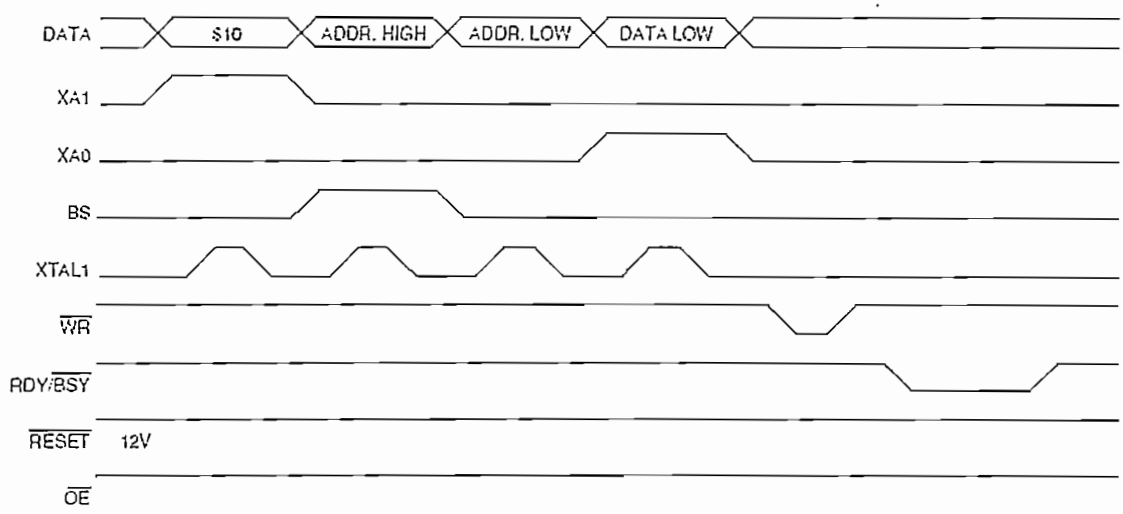



Figura 2.6: Formas de ondas en la programación de la memoria Flash

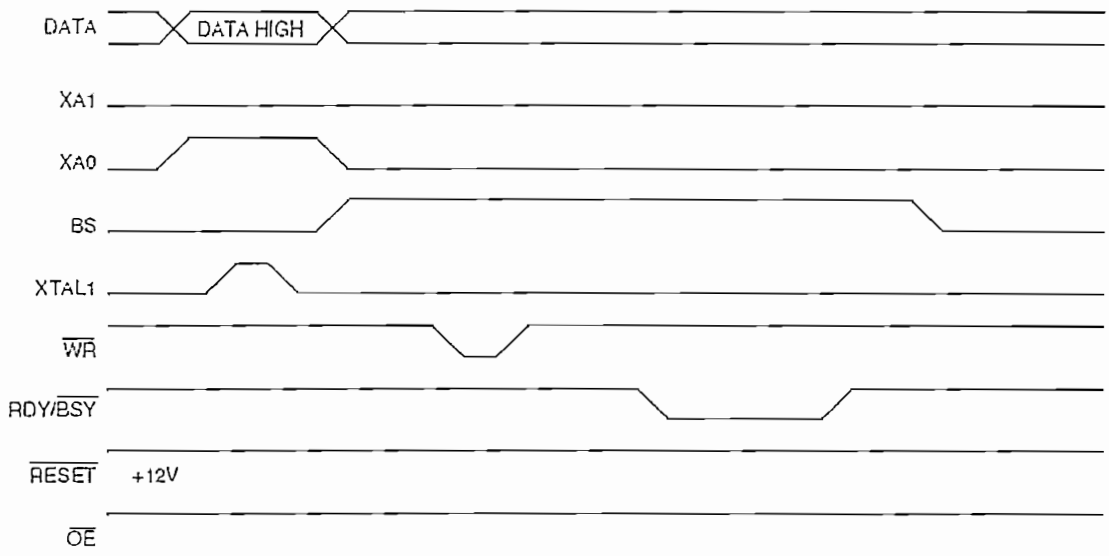



Figure 2.7: Formas de ondas en la programación de la memoria Flash (Continuación)

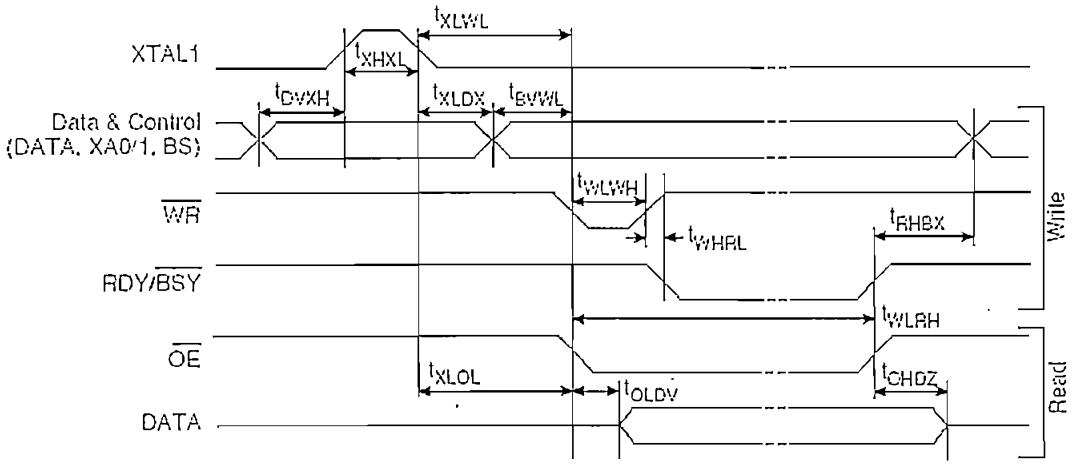



Figura 2.8: Diagramas de tiempo para la programación y verificación

*1.3.3.3.5 Verificación de la memoria Flash.*

En la verificación de la memoria Flash los pasos a seguir son similares a los de programación pero hay que realizar ciertos cambios como se muestra a continuación.

- Cargar comando de escritura de memoria Flash
    1. Fijar XA1, XA2 a "10". Esto habilita el comando de carga.
    2. Fijar BS a "0".
    3. Fijar DATA a "0000 0010". Este es el comando para escritura en el chip.
    4. Dar un pulso positivo en XTAL 1 (Esto carga el comando).
- Cargar dirección en el Byte Alto (High bit)
    1. Fijar XA1, XA2 a "00". Esto permite cargar la dirección.
    2. Fijar BS a "1". Este selecciona el byte alto.
    3. Fijar DATA = Dirección del byte alto ($00 - $07/$0F).
    4. Dar un pulso positivo en XTAL 1 (esto carga la dirección del byte alto).
- Cargando dirección el byte bajo (Low bit)
    1. Fijar XA1, XA2 a "00". Esto permite cargar la dirección.
    2. Fijar BS a "0". (Este selecciona el byte bajo).
    3. Fijar DATA = Dirección del byte bajo ($00 - $FF).
    4. Dar un pulso positivo en XTAL 1. (esto carga la dirección del byte bajo).

- Recuperando dato en el byte bajo
    1. Fijar OE a "0" y BS a "0".
    2. Copiar DATA = Dato del byte bajo ($00 - $FF).
- Recuperando dato en el byte alto
    1. Fijar OE a "0" y BS a "1".
    2. Copiar DATA = Dato del byte bajo ($00 - $FF).
- Etapa Final
    1. Fijar OE a "1". Esto selecciona dato alto.

*1.3.3.3.6 Programación de la memoria Flash.*

En la verificación de la memoria Flash los pasos a seguir son similares a los de programación y pero hay que realizar ciertos cambios como se muestra a continuación.

- Cargar comando de escritura de memoria Flash
    1. Fijar XA1, XA2 a "10". Esto habilita el comando de carga.
    2. Fijar BS a "0".
    3. Fijar DATA a "0010 0000". Este es el comando para escritura en el chip.
    4. Dar un pulso positivo en XTAL 1 (Esto carga el comando).
- Cargando dato en el byte bajo. Bit n = "0" programación de Local bit

  Bit 2 = Lock Bit2.

  Bit 1 = Lock Bit1.

  Bit 7-3,0 = "1". Estos bits son reservados y deberían no ser programados.
- Escritura del byte bajo
    1. Fijar BS a "0". Esto selecciona dato bajo.
    2. Fijar WR un pulso negativo. Esto inicia la programación del dato. RDY/BSY pasa a "0".
    3. Esperar hasta que RDY/BSY se ponga en alto para poder continuar con el próximo Byte.

Los bits de seguridad pueden ser borrados solamente ejecutando el comando de borrado del chip.

# CAPÍTULO III

# CAPITULO III
# DISEÑO DEL PROGRAMADOR

## 3.1 REQUERIMIENTOS GENERALES

Para cumplir con el objetivo propuesto se deben considerar los siguientes parámetros:

- Interfaz de comunicación entre el computador personal y el programador.
- Requerimientos de Voltajes.
- Señales de direccionamiento, datos y control.
- Software para el PC y para el microcontrolador

### 3.1.1 INTERFAZ DE COMUNICACIÓN ENTRE EL COMPUTADOR PERSONAL Y EL PROGRAMADOR.

El interfaz de comunicación entre el computador personal y el programador es el interfaz USB. La motivación para utilizar este interfaz viene de las siguientes ventajas:

- Facilidad de uso.
- Modelo simple de cables y conectores.
- Detalles eléctricos aislados del usuario.
- Fácil expansión de periféricos en la PC.
- Bajo costo para nuevas aplicaciones.
- Soporta hasta 127 dispositivos físicos.
- Soporte completo para transmisión en tiempo real de voz, audio y video.
- Flexibilidad de protocolos para transmisiones de múltiples flujos de datos y mensajes entre la PC y los diferentes dispositivos.
- El computador identifica automáticamente al dispositivo agregado y lo configura mientras opera.
- Capacidad para manejo y recuperación de errores producidos por cualquier dispositivo.

- Soporte para la arquitectura Conectar y Operar (Plug & Play).
- Soporte para diversos sistemas operativos como Windows, etc.
- Posibilitar la producción de nuevos dispositivos para aprovechar todas sus ventajas.
- Cómoda integración de dispositivos de tecnologías y fabricantes diferentes.
- Ofrece una interfaz estándar de rápida difusión entre productos para PC como por ejemplo: teclados, ratones, altavoces, impresoras, lectores externos de CDs, módems, routers, webcams, cámaras fotográficas digitales, escaners, entre otros.

### 3.1.2 REQUERIMIENTOS DE VOLTAJES.

La alimentación es uno de los elementos mas importantes de cualquier circuito, razón por la cual se debe hacer un estudio minucioso para el dimensionamiento de la fuente, considerando todos los voltajes necesarios para el correcto funcionamiento del circuito.

El voltaje requerido para el funcionamiento de los elementos del programador es de 5V, el cual puede obtenerse de una fuente regulada o del interfaz USB (tal como se indica en el Capítulo 1, subcapítulo 1.2.1.3 El host USB). Los voltajes requeridos para programar los diferentes microcontroladores son variables dependiendo del dispositivo, estos voltajes varían de 0V, 5V y 12V con una tolerancia $\pm$ 0.2 V; que se pueden obtener de una fuente programable.

### 3.1.3 SEÑALES DE DIRECCIONAMIENTO, DATOS Y CONTROL.

Las señales de direccionamiento, datos y control permiten determinar los distintos niveles de voltajes a manejar por el programador.

Dependiendo del tipo de microcontrolador se hace necesario utilizar varias líneas de control, de datos y de direcciones, como es el caso de los microcontroladores AT89CXX y AT89LVXX que requieren de líneas adicionales para el manejo de datos y para el direccionamiento de las localidades de memoria. No siendo así el

caso de los microcontroladores AVR que utilizan un solo bus de ocho bits para manejar datos y direcciones.

## 3.2 DIAGRAMA DE BLOQUE DEL PROGRAMADOR

En la figura 3.1 se presenta el diagrama general del programador, el cual esta formado por las diferentes partes:

- El computador personal en el que esta el software del programador y que sirve de interfaz gráfico para interactuar con el usuario.
- El cable USB que es el interfaz físico entre el computador personal y el programador.
- El Programador de microcontroladores que es el encargado del manejo de las señales de control, datos y direccionamiento.
- El bus de direcciones que es el interfaz físico entre los zócalos y el programador.
- Y los zócalos que sirven de soporte para la comunicación de los microcontroladores a programar y el programador.

Cabe indicar que la figura 3.1 es la base para el diseño del circuito final.

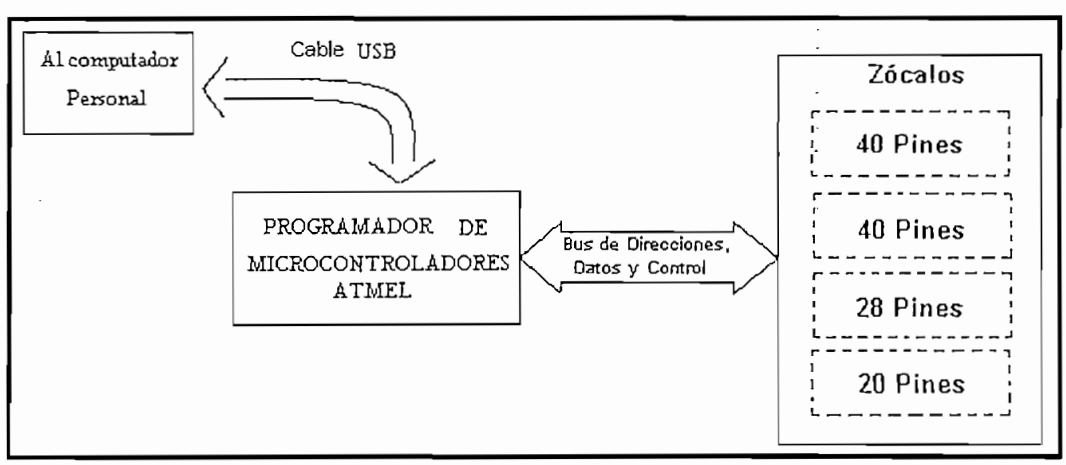



Figura 3.1: Diagrama general del programador

En la figura 3.2 se representa el programador en diagrama de bloques. Aquí se puede observar todas las etapas que permiten describir de una manera mas detallada el diseño del circuito. De acuerdo a la figura 3.2 el programador se divide en las siguientes partes:

- Conector USB en el cual se inserta el cable USB
- Microcontrolador con USB (USB uC) que es el encargado de manejar las señales de direccionamiento, control y datos
- LEDs Indicadores que indican de una forma visual que el programador esta encendido, y que además indican las diferentes actividades que el programador esta realizando como por ejemplo: inicio de lectura y escritura de la memoria flash de los diferentes microcontroladores, etc.
- Latches que son los encargados de almacenar las direcciones de las localidades de memoria para los microcontroladores AT89CXX y AT89LVXX.
- Bus de datos que son los encargados de llevar las diferentes señales desde el microprocesador USB a los diferentes dispositivos a programar.
- Cristales que generan las señales de reloj para el funcionamiento del microcontrolador con USB y los microcontroladores AT89CXX y AT89LVXX.
- Fuente de alimentación fija y programable que generan el voltaje necesario para alimentar a los diferentes elementos del programador y que generan los diferentes voltajes de programación.
- Zócalos de 20, 28 y 40 pines, que sirven de soporte para conectar los diferentes microcontroladores a programar.
-

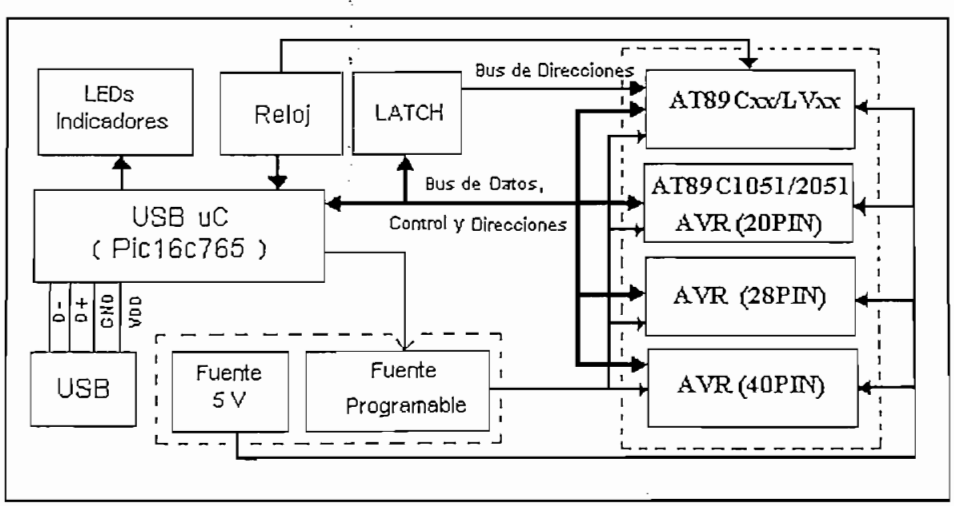



Figura 3.2: Diagrama de bloques del programador

## 3.3 DISEÑO CIRCUITAL

El diseño circuital del programador esta dividido en varias etapas, las cuales se describen a continuación:

### 3.3.1 DISEÑO DE LA ETAPA DE CONTROL

El programador tiene la capacidad de realizar operaciones directamente en la memoria Flash de los diferentes microcontroladores a programar, por lo cual es necesario optar por un microcontrolador debido a que estos son capaces de realizar operaciones aritméticas y lógicas mediante un programa almacenado en una memoria flash interna.

Luego de una búsqueda minuciosa de todos los microcontroladores USB comerciales se eligió el microcontrolador USB de la Microchip Technology Inc, el PIC16C765 cuyas características se describen a continuación:

- El microcontrolador PIC16C765, es un dispositivo con USB de baja velocidad que cumple con las especificaciones USB Rev 1.1, es de bajo costo (11 dólares el tipo OTP-Programable solo una vez y 18 dólares el tipo JW-Con ventana para borrar), con tecnología CMOS, full-static, microcontrolador de 8 bits en la familia media de los PIC, basado en la arquitectura RISC.
- Tiene 256 Bytes de memoria RAM y 64 bytes de memoria RAM para un puerto dual USB.
- Tiene características centrales reforzadas, una pila de ocho niveles y muchas fuentes de interrupción interna y externa.
- Todas las instrucciones son ejecutadas en un solo ciclo, excepto para saltos de programa que requiere 2 ciclos. Tiene un total de 35 instrucciones, y un largo set de registros.
- El dispositivo 16C765 consta de 33 pines de E/S (Entrada Salida) divididos en 5 puertos (RA, RB, RC, RD, RE). De las cuales el puerto RD es un puerto paralelo esclavo de 8 bits.

- Tiene 256 bytes de memoria RAM. Además incluye 3 temporizadores (contadores), 8 canales conversores análogo/digital de 8 bits, 2 módulos Capture Compare PWM y 2 puertos seriales, posee un modo de almacenamiento de energía (modo SLEEP) y un Watchdog Timer (WDT) con un oscilador dedicado on-chip RC.
- Con respecto a su oscilación, existen 4 configuraciones para el oscilador, de los cuales:
    1. EC es para una fuente de reloj regulada externa de 24 MHz.
    2. E4 es para una fuente de reloj regulada externa de 6 MHz con el PLL interno habilitado.
    3. HS es para cristales y resonadores de alta velocidad de 24 MHz y
    4. H4 es para cristales y resonadores alta velocidad de 6 MHz con el PLL interno habilitado, que multiplica la frecuencia de oscilación ($F_{osc}$) por 4.

Para mayor detalle de las especificaciones técnicas del microcontrolador refiérase al Anexo 1 o al archivo pdf completo de las especificaciones técnicas del PIC16C765 que se lo encuentra en: http://www.microchip.com.

Para el programador se utilizo los puertos RA (5 pines), RB (1 pin), RC (3 pines) y RD (8 pines) que son necesarios en el manejo de la información, direcciones, señales de control y datos de entrada y de salida (E/S).

El direccionamiento de las señales de control y de dirección desde el PIC16C765 a los diferentes microcontroladores a programar es directo. Ya que los puertos del PIC16C765 tienen salidas CMOS y entradas TTL. Los microcontroladores a programar poseen puertos de colector abierto facilitando la comunicación con el PIC

En la tabla 3.1 se muestra la relación de los pines del PIC16C765; con los pines de los diferentes dispositivos a programar. En los microcontroladores de la serie

AT89CXX y AT89LVXX se requieren de líneas adicionales para el direccionamiento de las localidades de memoria a través de un solo bus de datos del PIC; razón por la cual se adecuan dos latchs conectados al puerto RD que permite manejar hasta 16 líneas de direcciones adicionales. Ver figura 3.3.

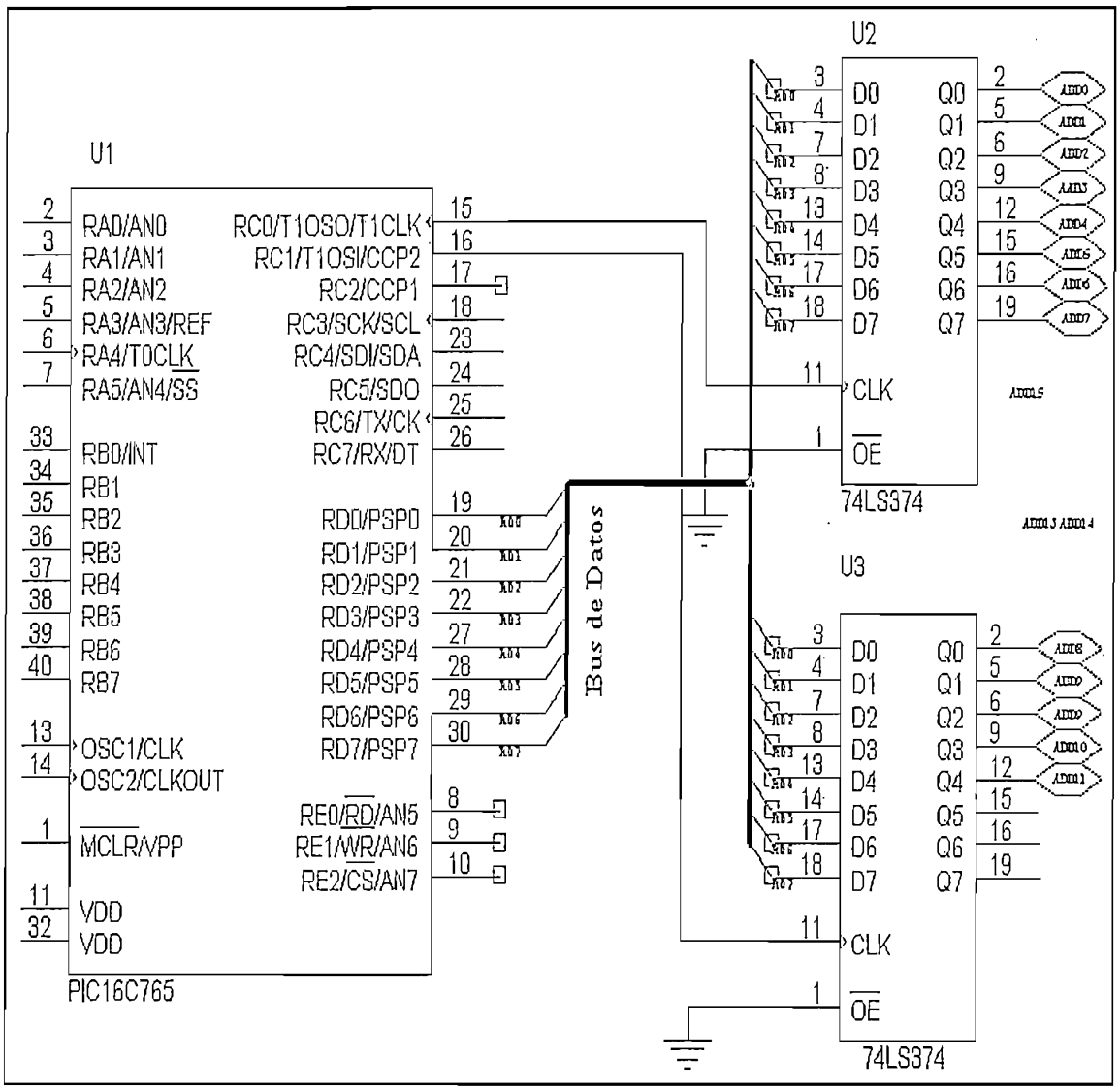



Figura 3.3: Latchs conectados al puerto RD del PIC16C765

| Puertos PIC16C765 | AT89C1051/C2051 | AT89Cxx/LVxx | AT90S1200/S2313 | AT90S/LS2333/ S/LS4433 | AT90S/LS4434/ S8515 / S/LS8535 |
|---|---|---|---|---|---|
| RA0 | P3.3 | P2.6 | PD.3 / WR | PD.3 / WR | PD.3 / WR |
| RA1 | P3.4 / ENABLE | P2.7 / ENABLE | PD.4 / BS | PD.4 / BS | PD.4 / BS |
| RA2 | P3.5 | P3.6 | PD.5 / XA0 | PD.5 / XA0 | PD.5 / XA0 |
| RA3 | P3.7 | P3.7 | PD.6 / XA1 | PD.6 / XA1 | PD.6 / XA1 |
| RA4 | Control 0/5/12V | Control 0/5/12V | Control 0/5/12V | Control 0/5/12V | Control 0/5/12V |
| RA5 | P3.2 / PROG | P3.2 / ALE/PROG | PD.2 / OE | PD.2 / OE | PD.2 / OE |
| RB0 | P3.1 / RDY/BSY | P3.4 / RDY/BSY | PD.1 / RDY/BSY | PD.1 / RDY/BSY | PD.1 / RDY/BSY |
| RC0 | ********** | Carga Buffer1 | ********** | ********** | ********** |
| RC1 | ********** | Carga Buffer2 | ********** | ********** | ********** |
| RC6 | Control 5/12V | Control 5/12V | Control 5/12V | Control 5/12V | Control 5/12V |
| RC7 | XTAL1 | ********** | XTAL1 | XTAL1 | XTAL1 |
| RD0 | P1.0 / Dato.0 (E/S) | P0.0 / Dato.0 (E/S/ADD) | PB.0 / Dato.0 (E/S/ADD) | PB.0 / Dato.0 (E/S/ADD) | PB.0 / Dato.0 (E/S/ADD) |
| RD1 | P1.1 / Dato.1 (E/S) | P0.1 / Dato.1 (E/S/ADD) | PB.1 / Dato.1 (E/S/ADD) | PB.1 / Dato.1 (E/S/ADD) | PB.1 / Dato.1 (E/S/ADD) |
| RD2 | P1.2 / Dato.2 (E/S) | P0.2 / Dato.2 (E/S/ADD) | PB.2 / Dato.2 (E/S/ADD) | PB.2 / Dato.2 (E/S/ADD) | PB.2 / Dato.2 (E/S/ADD) |
| RD3 | P1.3 / Dato.3 (E/S) | P0.3 / Dato.3 (E/S/ADD) | PB.3 / Dato.3 (E/S/ADD) | PB.3 / Dato.3 (E/S/ADD) | PB.3 / Dato.3 (E/S/ADD) |
| RD4 | P1.4 / Dato.4 (E/S) | P0.4 / Dato.4 (E/S/ADD) | PB.4 / Dato.4 (E/S/ADD) | PB.4 / Dato.4 (E/S/ADD) | PB.4 / Dato.4 (E/S/ADD) |
| RD5 | P1.5 / Dato.5 (E/S) | P0.5 / Dato.5 (E/S/ADD) | PB.5 / Dato.5 (E/S/ADD) | PB.5 / Dato.5 (E/S/ADD) | PB.5 / Dato.5 (E/S/ADD) |
| RD6 | P1.6 / Dato.6 (E/S) | P0.6 / Dato.6 (E/S/ADD) | PB.6 / Dato.6 (E/S/ADD) | PB.6 / Dato.6 (E/S/ADD) | PB.6 / Dato.6 (E/S/ADD) |
| RD7 | P1.7 / Dato.7 (E/S) | P0.7 / Dato.7 (E/S/ADD) | PB.7 / Dato.7 (E/S/ADD) | PB.7 / Dato.7 (E/S/ADD) | PB.7 / Dato.7 (E/S/ADD) |

Tabla 3.1: Relación de los pines del PIC16C765 con los pines de los diferentes dispositivos a programar.

### 3.3.2 DISEÑO DE LA FUENTE

Como se estableció anteriormente, los voltajes requeridos por el programador son de 5V (fuente fija) y 0V / 5V / 12V (fuente programable). Por lo cual se dividió en dos casos:

#### 3.3.2.1 Diseño de la fuente fija

Para el diseño de la fuente fija hay que considerar la máxima corriente de consumo de todos los componentes del programador, como son los microcontroladores a programar, resistencias, latchs 74LS374 y leds.

El microcontrolador PIC16C765 funciona con un voltaje de 5V el cual puede obtenerse de la fuente fija o del interfaz USB. En el diseño se opto por utilizar este voltaje para el manejo del PIC ya que una de las características del interfaz USB es que puede obtenerse del PC un voltaje de 5V con una corriente de hasta 100mA.

En la tabla 3.2 se detalla la corriente máxima que se debe obtener del interfaz USB.

| Item | Elemento | Máxima corriente de consumo(mA) |
|---|---|---|
| 1 | PIC16C765 (U1) | 25 |
| 2 | Resistencia 4,7Kohm (R3) | 1 |
| 3 | Resistencia 4,7Kohm (R7) | 1 |
| 4 | Resistencia 1,5Kohm (R9) | 3 |
| | **Total :** | **30** |

Tabla 3.2: Corriente máxima requerida del Interfaz USB

En la tabla 3.3 se hace un desglose de las corrientes máximas requeridas por los elementos del programador.

| Item | Elemento | Máxima corriente de consumo(mA) |
|---|---|---|
| 1 | 74LS374 (U2) | 40 |
| 2 | 74LS374 (U3) | 40 |
| 3 | 74LS07 (U10A) | 45 |
| 4 | AT89C1051 / C2051 | 15 |
| 5 | AT89Cxx / LVxx | 25 |
| 6 | AT90S1200 / S2313 | 3 |
| 7 | AT90S/LS2333 / S/LS4433 | 5 |
| 8 | AT90S/LS4434 / S8515 / S/LS8535 | 6,4 |
| | **Total :** | **171.4** |

Tabla 3.3: Consumo de corriente máxima por elemento

Como se puede observar en la tabla 3.3 la máxima corriente que debe entregar la fuente es de 171.4 mA a un voltaje de 5V. Por lo tanto se necesita un regulador de 5V, el mismo que sea capaz de entregar una corriente máxima de 172 mA. Para esto se utiliza un regulador LM7805 que puede dar una corriente máxima de 1A.
El voltaje de entrada característico del regulador LM7805 es de 9 a 12V razón por el cual es necesario adecuar la señal de entrada a un regulador LM7812 que es el encargado de entregar la corriente necesaria y los 12 voltios requeridos por el regulador.
En la figura 3.4 se indica el conexionado del regulador con los distintos elementos necesarios para su funcionamiento.

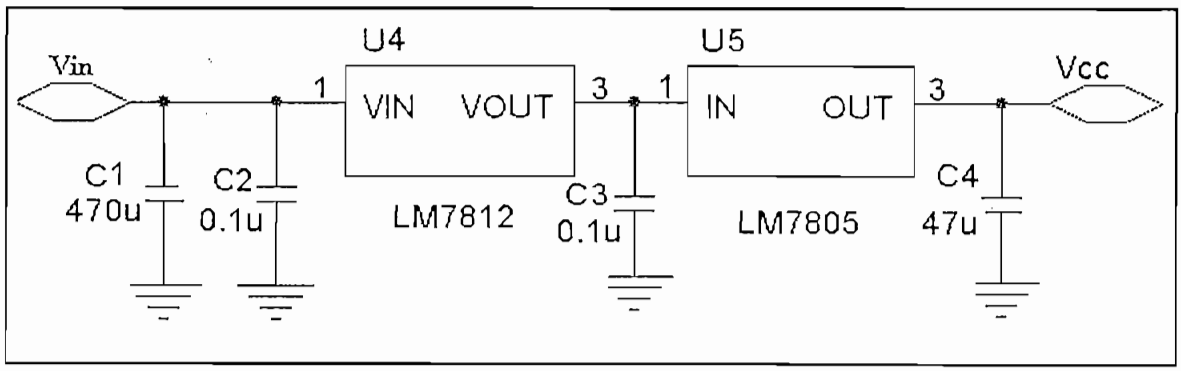



Figura 3.4: Fuente regulada Vcc de 5V

### 3.3.2.2 Diseño de la fuente programable

Los microcontroladores ATMEL requieren de fuentes reguladas programables para manejar las señales de EA, PROG, RESET. Estas señales requieren de voltajes de 0V, 5V o 12V con un consumo máximo de corriente de 40 mA. El regulador variable LM317 cubre esta necesidad ya que dependiendo del voltaje de entrada y de los valores de dos resistencias, puede entregar voltajes comprendidos entre 1.5 a 30V y una corriente máxima de 1.5A.
El voltaje de salida para este regulador depende de la relación

$$V_{OUT} = 1.25 * \left(1 + \frac{R_{V1}}{R}\right)$$

Donde R es la resistencia fija que va del pin de voltaje salida ($V_{out}$) al pin de ajuste (Adj) y $R_{V1}$ es la resistencia variable que va desde el pin de ajuste hacia tierra.

Por lo tanto despejando la ecuación anterior: $R_{v1} = R_2 * \left( \frac{V_{out}}{1.25} - 1 \right)$

Para obtener el voltaje de 12V calculamos mediante la relación anterior obteniendo para una resistencia $R_2$ = 220 ohm el valor de la resistencia variable de $R_{V1}$ = 1892 ohm. El valor obtenido no es un valor estándar por lo que se utilizo un potenciómetro de precisión de 2Kohm para obtener el valor calculado.

Para conmutar la fuente de 12 a 5V o viceversa se incorpora una resistencia $R_{v2}$ en paralelo con $R_{v1}$ controlada por un transistor NPN (2N3904). El voltaje de salida del regulador depende del estado del transistor, es decir si esta trabajando en corte o saturación. Este estado depende de la señal que envía el PIC16C765 por el pin RC6 a la resistencia de 4.7 Kohm conectada en la base del transistor.
En base a la ecuación de $V_{out}$ y para un voltaje de salida de 5V se obtiene la siguiente ecuación que nos permite el cálculo de la resistencia $R_{v2}$:

$$R_{v2} = \frac{3 * R_2 * R_{v1}}{R_{v1} - 3 * R_2}$$

Reemplazando el valor de $R_2$ =220 ohm y el de Rv1=1892 ohm se obtiene el valor de $R_{v2}$ = 1013 ohm
Al igual que el caso anterior el valor obtenido para $R_{v2}$ no es un valor estándar por lo que se utilizo un potenciómetro de precisión de 2 Kohm.
La figura 3.5 muestra la fuente programable con todos los componentes necesarios para su correcto funcionamiento.

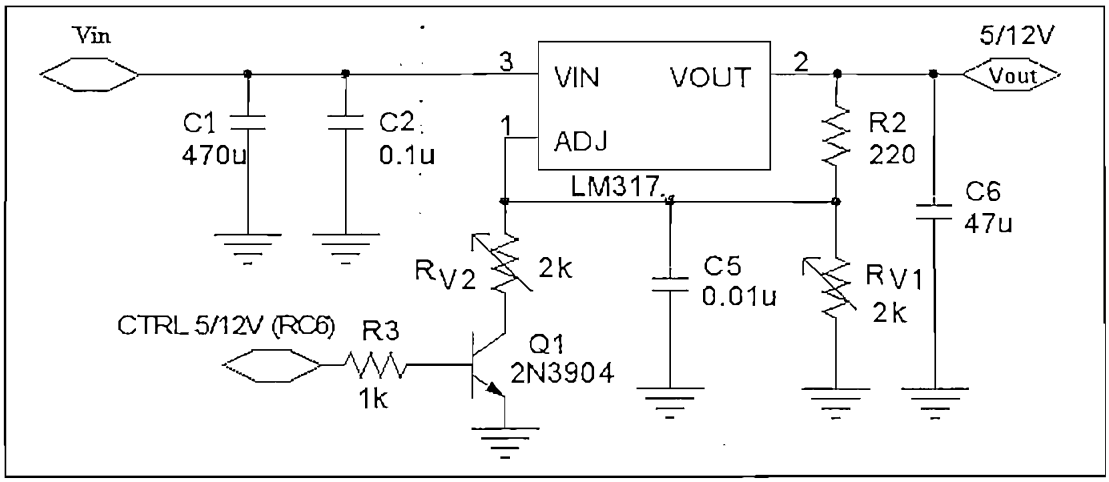



Figura 3.5: Fuente programable 5V/12V

Para obtener la conmutación completa que comprende el cambio de 5V a 0V o de 12 a 0V o viceversa es necesario implementar un circuito que se pueda controlar desde uno de los pines del PIC16C765. El pin seleccionado es RA4 ya que tiene la característica de ser a colector abierto por lo que se aprovecha esta característica para conectar a una de las entradas del buffer/driver 74LS07 que permite manejar voltajes de salida de hasta 30V. Manejar directamente los voltajes de 5 o 12V desde el pin RA4 puede ocasionar un daño permanente al PIC. A la salida del pin RA4 es necesario utilizar una resistencia R7 de 4.7Kohm conectada a $V_{DD}$ y este a su vez a una de las entradas del 74LS07.

En la figura 3.6 se puede observar que el voltaje de salida de la figura 3.5 esta conectado a una resistencia R4 de 10Kohm y al emisor del transistor PNP. En la base del transistor se tiene conectado el otro terminal de la resistencia con otra resistencia de 1Kohm (R5) conectada a la salida del 74LS07. En el colector del transistor se tiene conectada una resistencia R6 de 4.7Kohm que limita la corriente para el encendido del led y de igual manera aquí se obtiene la salida de voltaje Vpp, que sirven para los diferentes voltajes de programación.

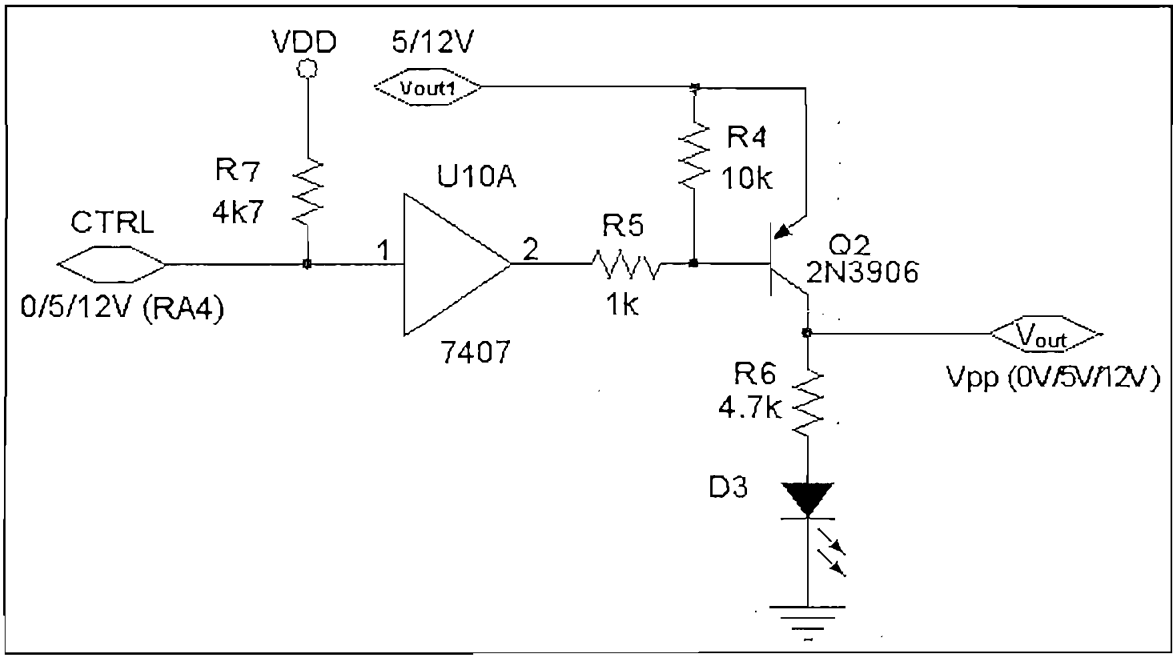



Figura 3.6: Salida de Voltaje de Programación Vpp

Como se indicó anteriormente los pines del PIC16C765 asignados al manejo de la conmutación de voltajes son los pines RA4 y RC6. En la tabla 3.4 se tienen los voltajes de salida al realizar las diferentes combinaciones en los pines RA4 y RC6

| RA4 | RC6 | Voltaje de salida |
|---|---|---|
| 0 | 0 | 12V |
| 0 | 1 | 5V |
| 1 | 0 | 0V |
| 1 | 1 | 0V |

Tabla 3.4: Voltajes de salida vs Combinación de señales en RA4 y RC6

Para la señal de entrada se utiliza una fuente DC externa de 15 a 18V.

Una medida para evitar que se produzcan daños en los elementos (programador) por polaridad negativa de la fuente DC es utilizar un puente de diodos de un amperio. En la figura 3.7 se muestra la etapa de entrada con los elementos mencionados.

### 3.3.3 DISEÑO DEL MODULO DE INSERCIÓN DE MICROCONTROLADORES

En el diseño del modulo de Inserción de microcontroladores es indispensable un modulo en el cual se puedan insertar los dispositivos a programar.

Para el diseño de estos módulos se consideraron las líneas tanto de control como de direccionamiento y de igual manera la superposición de los pines correspondientes a los diferentes microcontroladores a programar

Debido a que los pines asignados para la programación de los diferentes microcontroladores no coinciden se llego a una solución bastante económica como es utilizar cuatro zócalos de los cuales dos son de 40 pines, uno de 28 pines y uno de 20 pines.

Para la programación de los microcontroladores de la serie AT89CXX y AT89LVXX se requiere de un oscilador de 3 a 24 MHZ (serie AT89CXX) y de 3 a 12 MHZ (serie AT89LVXX) debido a que el bus interno del microcontrolador esta siendo utilizada para transmitir direcciones y datos a registros apropiados. Por el cual se selecciona un cristal de 4MHz con dos condensadores de 33 pF.

### 3.3.5 DISEÑO DE LA ETAPA DE SEÑALIZACIÓN

Al hablar de señalización nos referimos a la parte visual es decir a la etapa que nos muestra los diferentes procesos o eventos que realiza el programador.

### 3.3.4 DISEÑO DE LA ETAPA DE RELOJ

Para que cualquier microcontrolador funcione, además de la fuente de alimentación es necesaria una fuente que genere las señales de reloj. Estas señales pueden ser generadas por una fuente de reloj regulada externa o por un arreglo de cristales o resonadores con condensadores.

Para generar la señal de reloj para el microcontrolador PIC16C765 se utiliza un cristal de 6MHz con condensadores de 33pF, para trabajar en el modo de operación H4 como se vio anteriormente en las características técnicas.

Para la programación de los microcontroladores de la serie AT89CXX y AT89LVXX se requiere de un oscilador de 3 a 24 MHZ (serie AT89CXX) y de 3 a 12 MHZ (serie AT89LVXX) debido a que el bus interno del microcontrolador esta siendo utilizada para transmitir direcciones y datos a registros apropiados. Por el cual se selecciona un cristal de 4MHz con dos condensadores de 33 pF.

### 3.3.5 DISEÑO DE LA ETAPA DE SEÑALIZACIÓN

Al hablar de señalización nos referimos a la parte visual es decir a la etapa que nos muestra los diferentes procesos o eventos que realiza el programador.

Por lo cual se implementa un arreglo de una resistencia con un led rojo a la entrada de la fuente para indicar que el programador esta encendido. Esta resistencia R1 es de 1Kohm y limita la corriente para el Led.

Se realiza otro arreglo de resistencia con led verde en la salida de VPP para indicar que el dispositivo esta verificando, programando, etc.

En la siguiente figura se detalla el programador con todas sus etapas.

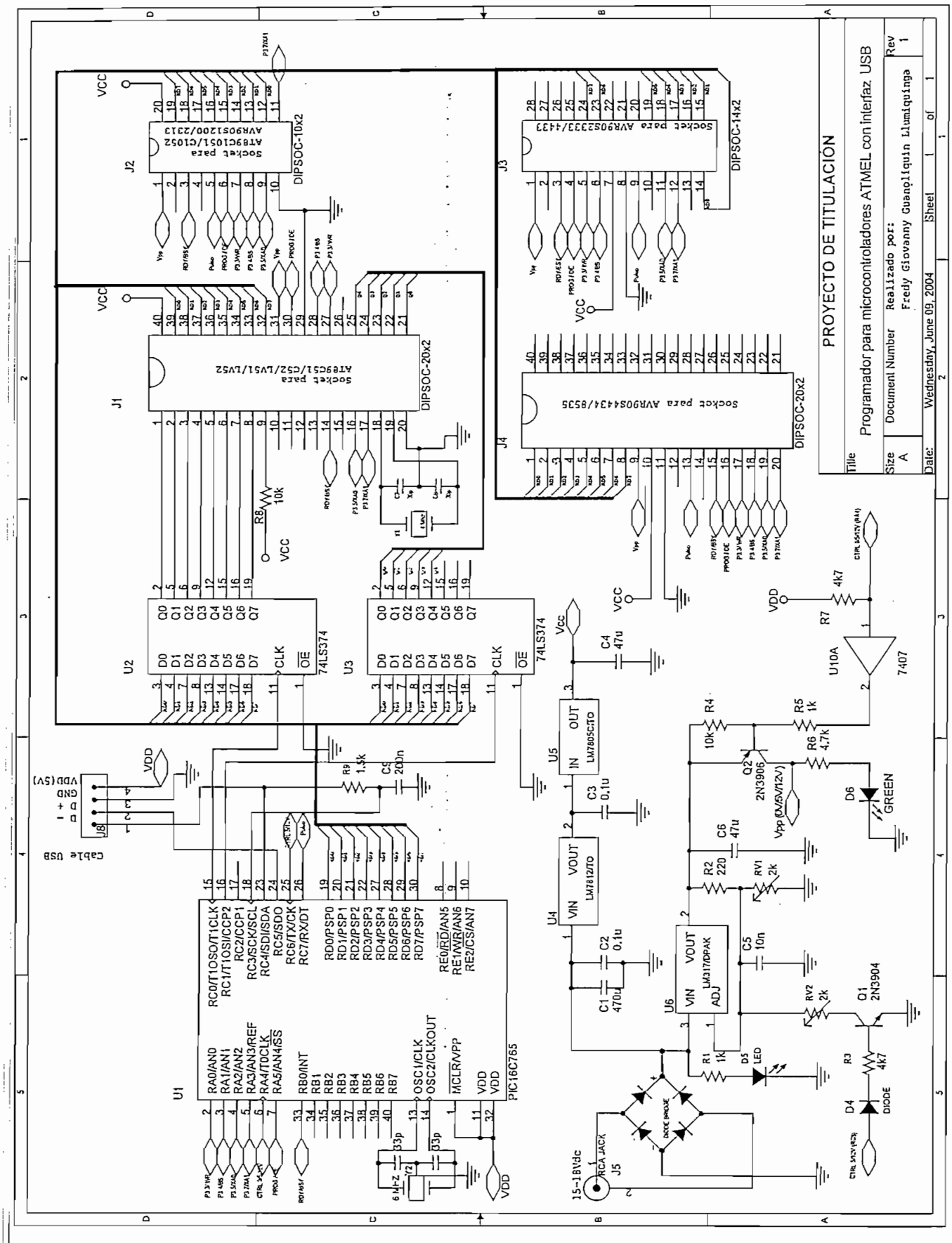



Figura 3.7: Diagrama del Programador para microcontroladores ATMEL con interfaz USB

# CAPITULO IV
# DESARROLLO DEL SOFTWARE

El desarrollo del software se divide en dos partes: desarrollo del firmware y desarrollo del software para el PC

## 4.1 DESARROLLO DEL FIRMWARE

Para desarrollar el firmware se hace necesario algunas definiciones, las cuales se obtuvieron de la hoja de datos del PIC16C765 (USB Firmware users guide).

### 4.1.1 USB FIRMWARE PIC16C765

Microchip Technology Inc provee un Firmware en lenguaje ensamblador (Microchip Assembly) para el microcontrolador 16C765, que permite dar un punto de partida en el desarrollo de un dispositivo USB. Este Firmware implementa la funcionalidad del Capítulo 9 de la Especificación USB 1.1 y la definición del Dispositivo de Interfaz Humana HID (Human Interface Device).

Usando una librería pre-definida de Interfaz para la Programación de Aplicaciones (APIs), este firmware permite a diseñadores con poca o ningúna experiencia en USB llevar a cabo sus aplicaciones rápidamente con el microcontrolador PIC16C765. Microchip provee un software de capas que manejan el interfaz de bajo nivel. Este provee un simple interface Put/Get para la comunicación.

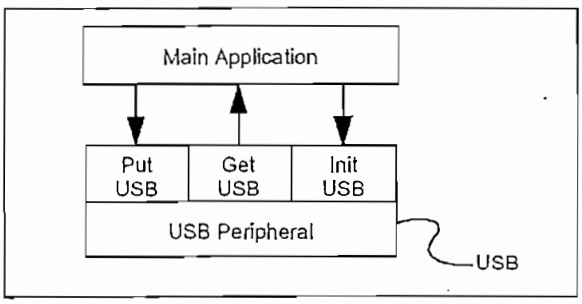



Figura 4.1: Interface del software USB

Este firmware puede ser descargado de la página web www.microchip.com, que constantemente se está proveyendo actualizaciones de la aplicación USB inicial.

### 4.1.1.1 Conceptos básicos de USB en el PIC16C765

#### *4.1.1.1.1 Memoria RAM.*

En esta sección no se va ha definir que es memoria RAM, pero si es importante hacer una introducción a la memoria RAM que posee este microcontrolador.

El PIC16C765 Tiene 256 Bytes de memoria RAM para propósito general y 64 bytes de memoria RAM para el puerto dual USB.

La memoria RAM para el puerto dual USB se distribuye en: 24 bytes para el buffer de la tabla de los descriptores (BDT) y 40 bytes distribuidos en cinco buffers de 8 bytes dedicados para los Endpoints.

#### *4.1.1.1.2 Endpoints*

Los Endpoints son buffers, en donde los datos aguardan para ser puestos en el bus USB, o desde donde los datos son removidos del bus. Los Endpoints consisten de un número y una dirección. Todos los dispositivos USB utilizan el EP0 (*Endpoint cero*) para administrar la comunicación entre el dispositivo y el host. Esto significa que EP0 es bi-direccional es decir se tienen dos endpoints, uno de entrada EP0 IN, y uno de salida EP0 OUT.

Estos utilizan 2 buffers de los 40 bytes dedicados para endpoints. De este modo quedan tres buffers para posibles endpoints. De estos, solo dos pueden ser usados porque la especificación USB 1.1 permite a los dispositivos de baja velocidad utilizar solo dos endpoints adicionales al endpoint 0.

La configuración por defecto asigna buffers individuales a EP0 OUT, EP0 IN, EP1 OUT y EP1 IN. El último buffer es compartido entre EP2 IN y EP2 OUT. Nuevamente

la especificación dice que dispositivos de baja velocidad pueden usar solo dos endpoints a parte de EP0. Esta configuración soporta muchas de las posibles combinaciones de endpoints (EP1 IN y EP1 OUT, EP1 OUT y EP2 IN, EP1 OUT y EP2 OUT, EP1 IN y EP2 OUT, EP1 IN y EP2 IN). La única combinación que no es soportada por esta configuración es EP2 IN y EP2 OUT.

*4.1.1.1.3 Ancho de banda*

La velocidad de transmisión de datos para un endpoint es limitada a 800 bytes/segundo. Como se indico anteriormente el PIC16C765 al ser un dispositivo de baja velocidad esta limitado a dos endpoints adicionales aparte del EP0 para trasmitir datos del usuario, garantizando así una taza de transmisión uni-direccional de 1600 bytes/segundo.

Si la dirección deseada de comunicación es desde el dispositivo hacia el host, se usarían los endpoints EP1 IN y EP2 IN.

La Especificación de Clase HID hace posible la petición de Get_Report y Set_Report. Estas peticiones pueden incrementar el ancho de banda de un dispositivo a 10 veces. Estas funciones pueden hacer esto posible porque permiten transferir los datos vía EP0, puesto que EP0 tiene una taza de transmisión de 1 ms, entonces el incremento en el ancho de banda es de 8000 bytes/segundo en cada dirección.

*4.1.1.1.4 Tipo de datos*

La especificación USB define cuatro tipos de transferencia: control, interrupción, isócrona y en volumen. De estos, solo la transferencia de control e interrupción son soportadas por dispositivos de baja velocidad.

La transferencia de control soporta la enumeración y comunicación vía EP0. El host envía información de petición y setup al microcontrolador vía EP0 OUT. El microcontrolador responde a esta petición vía EP0 IN. El usuario especifica la máxima taza de transferencia en el descriptor de endpoint. Si se especifican 10 ms, significa que el host envía el endpoint particular al menos cada 10 ms, donde 10 ms es el intervalo que puede ser pedido de acuerdo a la especificación USB 1.1.

En definitiva, los dispositivos de baja velocidad usan transferencia de control por EP0, mientras que EP1 y EP2 usan solo transferencia de interrupción.

#### 4.1.1.2 Referencia de las funciones del firmware

Este firmware implementa funciones básicas como la inicialización de la comunicación USB, funciones para obtener datos del bus y poner datos en el bus, otras funciones más avanzadas como modo de ahorro de energía y reinicialización de la enumeración. Estas funciones básicas se describen a continuación:

- ***InitUSB:*** Esta función debe ser llamada por la rutina principal después de energizarse. Se debe llamar con un retardo de 16 μs para dar tiempo al periférico USB que realice el RESET antes de comenzar con la enumeración. Es decir InitUSB habilita al dispositivo y la interrupción Reset USB, de este modo incita al host a que enumere al dispositivo.

- ***ConfiguredUSB:*** Obtiene los bits de estado de la enumeración para ver si el dispositivo ha sido configurado por el host. ConfiguredUSB retorna un 1 en la bandera Z si el dispositivo es configurado, o un 0 si no. Este macro debe ser usado después de llamar a InitUSB para determinar si los Endpoints 1 y 2 pueden ser usados (vía GetEPn o PutEPn). Los Endpoints 1 y 2 pueden ser usados solo si el dispositivo es configurado.

- ***DeinitUSB:*** Deshabilita al periférico USB, es decir hace que el host ignore al dispositivo. Todas las interrupciones USB son deshabilitadas.

- ***ServiceUSB:*** Se llama a esta función desde el lazo principal del programa para procesar las transacciones USB. Las transacciones desde el host señalan un TokenDone cuando se a completado el TOKEN que se está procesando actualmente.

- *PutEPn:* Envía datos al host vía endpoint n.

- *GetEPn:* recibe datos del host vía endpoint n.

Las funciones avanzadas son:

- *SoftDetachUSB:* Re-enumera al dispositivo. Este proceso toma aproximadamente 50 ms para asegurar que el host ha detectado su desconexión y reconexión al bus.

- *RemoteWakeup:* Incita al host a que reinicie la comunicación con el microcontrolador.

- *StallUSBEP/UnstallUSBEP:* Encera o pone en uno el bit stall en el registro de control de endpoint. El bit stall le indica al host que hay una demanda de intervención de usuario, y hasta que esta intervención sea realizada, la comunicación con el endpoint no será satisfactoria. Una vez que la intervención del usuario haya sido realizada, UnstallUSBEP encerará el bit permitiendo la comunicación nuevamente. Estas llamadas son útiles para indicar al host que el usuario va ha intervenir. Un ejemplo de esto podría ser que la impresora indique que no tiene papel.

#### 4.1.1.3 Archivos empaquetados versión en lenguaje ensamblador

Microchip provee soporte mediante varios ejemplos que utilizan firmware USB, los cuales pueden ser modificados de tal forma que sirvan para el desarrollo de nuevas aplicaciones.

Como ejemplo vamos a desglosar el ejemplo del mouse que causa el giro en círculo del cursor en la pantalla.

La versión en lenguaje ensamblador proporciona siete archivos necesarios para construir el archivo hexadecimal. Estos archivos son:

- MOVECURS.pjt
- 16C765.lkr
- usb_defs.inc
- usb_main.asm
- usb_ch9.asm
- hidclass.asm
- descript.asm

Los cinco últimos archivos pertenecen a la implementación del USB en el PIC16C765. La descripción de estos archivos es la siguiente:

- usb_defs.inc – Define las variables USB y macros usadas en los siguientes archivos ensambladores.
- usb_main.asm – implementa ejemplo del cursor en un círculo. Los componentes de este ejemplo muestran la manera apropiada de implementar una Rutina de Servicio de Interrupción (ISR), inicialización de la comunicación USB, obtención de la rutina Service USB, y usa la Interface Put.
- usb_ch9.asm – Implementa las funciones que se encuentran en el Capítulo 9 de la Especificación USB Rev 1.1.
- hidclass.asm – Implementa las funciones encontradas en la Definición de la Clase de Dispositivo de Interface Humana.
- descript.asm – Contiene los descriptores para el ejemplo del cursor en un círculo.

Esta aplicación se enumera como un mouse y tiene el efecto de movimiento del cursor en un círculo en la pantalla. Proporciona propiamente descriptores setup, un ejemplo de ISR y de como la enumeración es iniciada.

Los pasos para correr el firmware en el PIC16C765 son:

1. Descomprimir el archivo usbxxxx.zip en una carpeta de proyecto
2. Programar el PIC16C765 con el archivo hex compilado en cualquier versión de MPLAB. Verificar que los bits de configuración estén establecidos de la siguiente manera:
    - Oscillator: H4
    - Watchdog Timer: Off
    - Power-up Timer: Off
    - Code Protect: Off

3. Implemente el circuito de la figura 3.1.

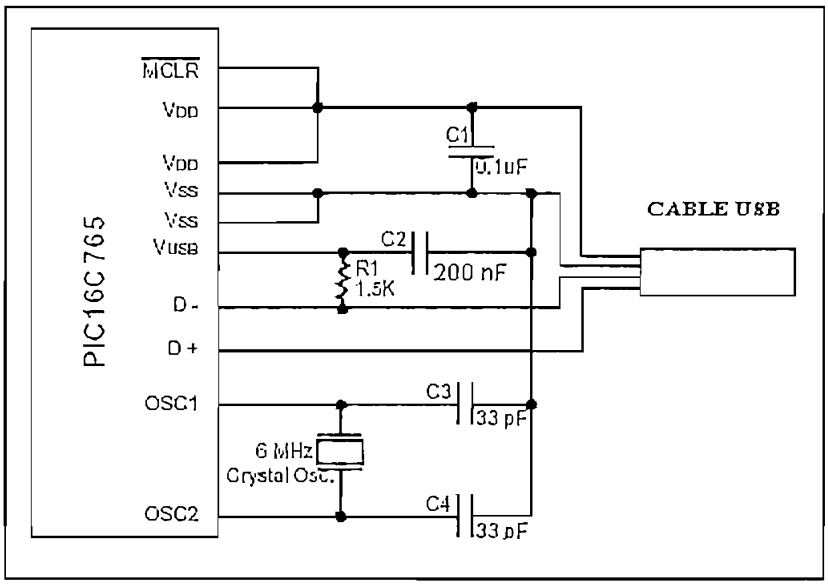



Figura 3.1: Circuito del PIC16C765

4. Coloque el cable USB al circuito y conéctelo al puerto USB de la PC. En máquinas con Windows 98, o mas avanzados, o Mac OS X, el sistema operativo detectará un dispositivo nuevo e instalará los drivers necesarios automáticamente. Después de ello, el cursor rotará en un pequeño círculo en la pantalla. Para parar la rotación el cursor, desconecte el cable USB.

Microchip proporciona varios ejemplos en paquete de archivos en versión en ensamblador y en C.

### 4.1.2 FIRMWARE DEL PROGRAMADOR USB

El firmware del programador se basa en optimizar el ejemplo del mouse, para lo cual se modifico el programa principal de tal manera que se elabore el archivo HEX que va ha implementarse en el PIC16C765.

Una de las modificaciones incluye el cambio de los descriptores de tal manera que windows reconozca al dispositivo como un programador e Interfaz Humana general no como un mouse como en el ejemplo.

Los archivos que incluye el firmware del programador son los siguientes:

- TesisProgAtmel.mcp
- TesisProgAtmel.hex
- 16C765.lkr
- USB_DEFS.inc
- ProgAtmel.asm
- USB_CH9.asm
- HIDCLASS.asm
- DESCRIPT.asm

El archivo TesisProgAtmel.mcp es el proyecto que se construye al compilar los archivos ASM en MPLAB ICE V6.3 o V6.4.

El archivo TesisProgAtmel.hex es el programa del programador compilado en MPLAB ICE V6.4, listo para programar en el PIC16C765 con cualquiera de los programadores comerciales de Pics.

El archivo 16C765.lkr es un archivo fuente no modificable que contiene recursos USB para este microcontrolador y que se debe incluirlo al construir el proyecto en MPLAB ICE.

Los cinco últimos archivos pertenecen a la implementación del USB en el PIC16C765, y realizan la comunicación Host – Programador.

La descripción de estos archivos es la siguiente:

- USB_DEFS.inc – Define las variables USB y macros usadas en los siguientes archivos ensambladores.
- ProgAtmel.asm – Implementa la comunicación USB Host-Programador. Es el programa principal que llama a las rutinas de inicialización del USB, configuración del USB, y obtiene los datos del bus con la ayuda de GetEP1(), a la vez que envía datos hacia el host vía PutEP1().
- USB_CH9.asm – Implementa las funciones que se encuentran en el Capítulo 9 de la Especificación USB Rev 1.1.
- HIDCLASS.asm – Implementa las funciones encontradas en la Definición de la Clase de Dispositivo de Interface Humana.
- DESCRIPT.asm – Contiene los descriptores para el programador de dispositivo de interfaz humana

#### 4.1.2.1 Diagrama de flujo de ProgAtmel.asm

El archivo ProgAtmel.asm es el archivo en lenguaje Ensamblador (Assembler) que implementa el programa principal para la comunicación USB del programador con el host. El programa principal se resume en los diagramas de flujo de las Figuras 4.1 hasta la Figura 4.13. A continuación se representan las Figuras y las descripciones de cada uno de ellos:

##### *4.1.2.1.1 Inicialización del programa principal*

Los diagramas de flujo de las Figura 4.1, 4.2, 4.3 y 4.4, muestran la secuencia que sigue el dispositivo al iniciar el programa. Primero se requiere de un retardo de 16 us, necesarios desde el instante en que el dispositivo es conectado en el host hasta que este lo detecta, provocando una interrupción de reset en el dispositivo,

permitiendo así que se configuren los registros y punteros necesarios, para que se pueda proceder a la enumeración.

A continuación se habilitan los puertos RA, RC y RD como salidas para que manejen las señales de direccionamiento, control y datos. El puerto RB es habilitado como entrada para manejar la señal de BSY/RDY.

Luego se hace un llamado a la función externa **InitUSB** ubicada en el archivo usb_ch9.asm, esta función inicializa el firmware y hardware USB, habilitando la interrupción reset USB. Como paso siguiente entra en un ciclo o lazo principal. En este lazo se llama a la función **ServiceUSB** en el cual se monitorea si hay algún token en proceso. Posterior a esto se llama a la función **ConfiguredUSB** implementada también en el usb_ch9.asm, espera que el proceso de enumeración sea completado, retornando un 1 lógico en el bit z del registro STATUS.

Realizados los pasos anteriores se llama a la función externa **GetEP1** para obtener los datos que ha enviado el host. Si el estado de la comunicación mantiene el carry en un valor de 0, quiere decir que aún no se ha obtenido toda la información del bus, y continúa intentando hasta conseguirla. Los datos que se espera en el dispositivo son ocho registros a las cuales se las va ha interpretar como tramas. Estos registros son RcvBuffer(0), RcvBuffer(1), RcvBuffer(3) RcvBuffer(4), RcvBuffer(5), RcvBuffer(6) y RcvBuffer(7).

Dependiendo del contenido de cada uno de estos registros las tramas recibidas por el PIC16C765 pueden ser comandos o datos. En la tabla 4.1 se resumen los diferentes comandos que puede recibir el PIC16C765 en base a la trama que se envía desde la PC.

| | RcvBuffer( i ) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Comando | i = 0 | i = 1 | i = 2 | i = 3 | i = 4 | i = 5 | i = 6 | i = 7 |
| Borra dispositivo AT89Cxx/LVxx | 1 | 2 | 2 | 1 | X | X | X | X |
| Borra dispositivo AT89C1051/LV2051 | 1 | 2 | 2 | 2 | X | X | X | X |
| Borra dispositivo AVR | 1 | 2 | 2 | 4 | X | X | X | X |
| Inicializa Modo de Lectura AT89Cxx/LVxx | 1 | 2 | 2 | 0 | 1 | X | X | X |
| Inicializa Modo de Lectura AT89C10/2051 | 1 | 2 | 2 | 0 | 2 | X | X | X |
| Inicializa Modo de Lectura AVR | 1 | 2 | 2 | 0 | 4 | X | X | X |
| Inicialización Programación AT89Cxx/LVxx | 1 | 2 | 2 | 0 | 8 | X | X | X |
| Inicializa Modo de Programación AT89C10/2051 | 1 | 2 | 2 | 0 | 16 | X | X | X |
| Inicializa Modo de Programación AVR | 1 | 2 | 2 | 0 | 32 | X | X | X |
| Verifica byte descriptores de AT89Cxx/LVxx | 1 | 2 | 8 | X | X | 1 | X | X |
| Verifica byte descriptores de AT89C1051/LV2051 | 1 | 2 | 8 | X | X | 2 | X | X |
| Verifica byte descriptores de AVR | 1 | 2 | 8 | X | X | 4 | X | X |
| Finalización | 1 | 2 | 4 | X | X | X | X | X |

Tabla 4.1: Comandos que llegan hacia el PIC16C765 desde el PC

En base a cada uno de estos comandos el programa llama a las diferentes subrutinas que se muestran en los diagramas de flujo. En el Figura 4.2 se observa que se hace uso de un registro temporal TempWord que es la encargada de almacenar el comando para los casos de programación y Lectura de datos de los diferentes dispositivos a programar.

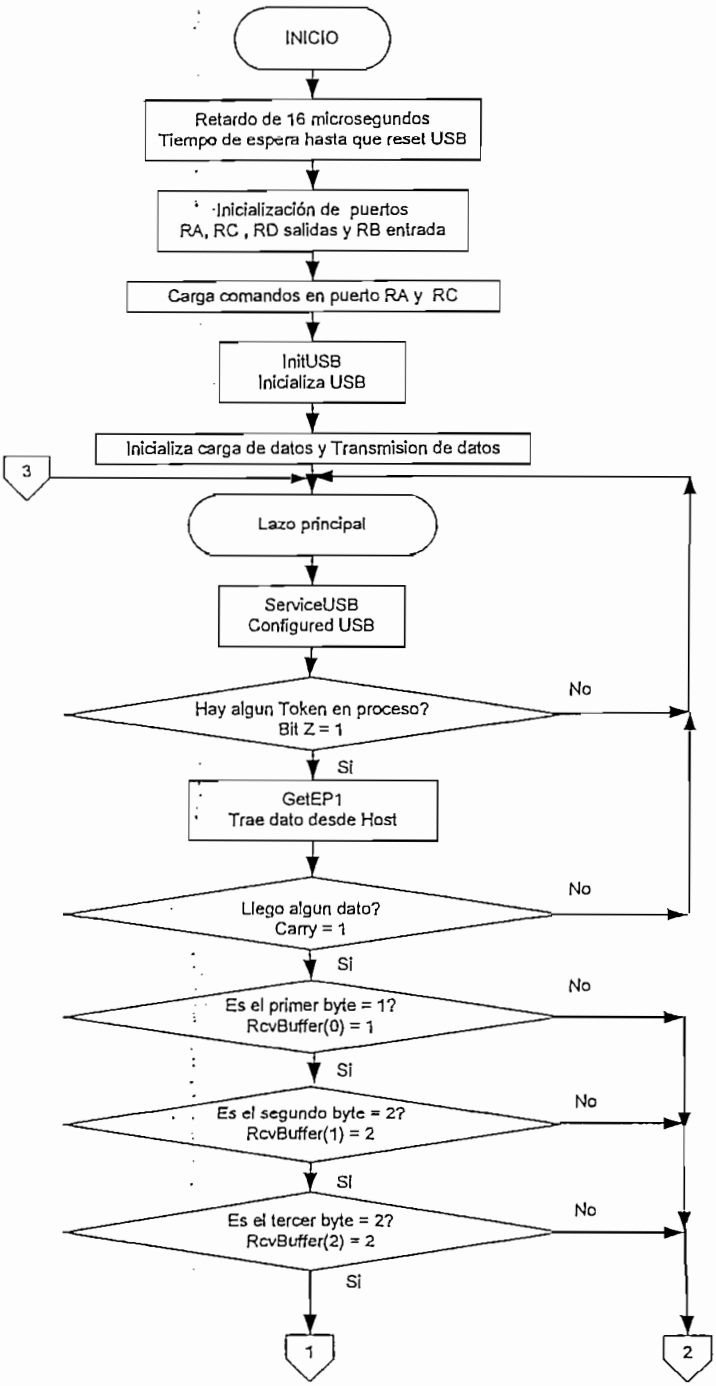



Figura 4.1: Diagrama de flujo del inicio de ProgAtmel.asm

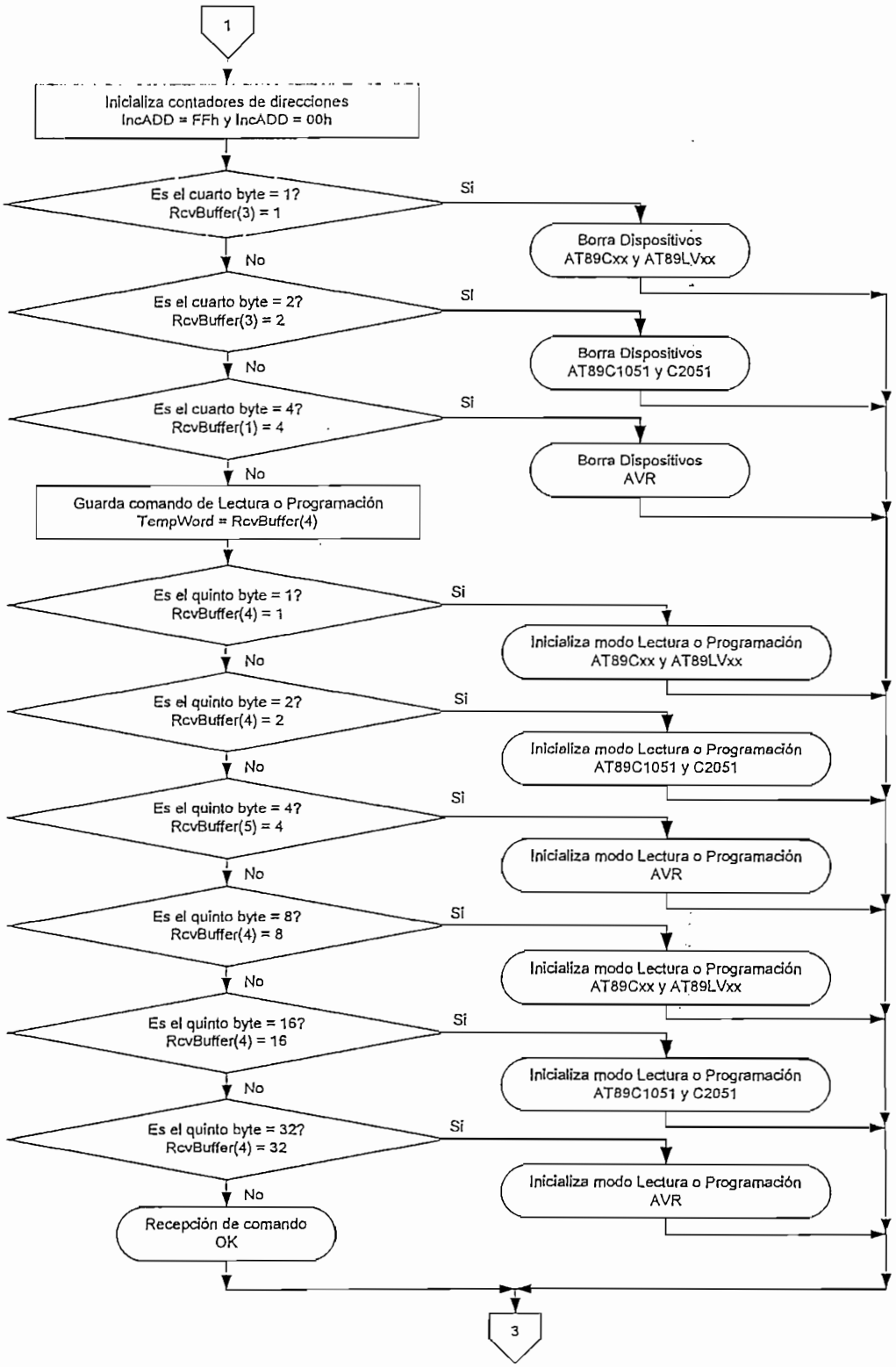



Figura 4.2: Diagrama de flujo continuación de inicio ProgAtmel.asm

Una vez que se ha inicializado cada uno de los dispositivos a programar tanto en el caso de programación o en Lectura de datos, el registro temporal Tempword es la variable que indica cual es el proceso que debe seguir el PIC16C765, es decir si la penúltima trama recibida fue comando de inicialización de programación o Lectura, esto indica que el próximo evento que debe realizar el PIC16C765 es programar o verificar el dispositivo seleccionado con los datos que le lleguen en la tarma recibida. Este proceso continua hasta que llegue una trama con el comando de fin de trama. Si la penúltima trama recibida fue de borrado o de fin de trama, la variable Tempword no tiene ningún comando por lo cual salta a la subrutina de que no ha recibido ningún comando.

En el Figura 4.3 se muestra la subrutina en la que se verifica si el comando recibido es de lectura de los bytes descriptores. Dependiendo del contenido del sexto registro se produce el salto hacia las subrutinas seleccionada en la trama de comando. Si el sexto registro contiene un dato diferente de 1, 2 o 4 se producen un salto a la subrutina de indicación de que no se ha recibido ninguna trama de comando.

En el diagrama de flujo de la figura 4.3 se muestra los casos en los que se cumplen los comandos de lectura de byte descriptores del dispositivo a programar y el comando de finalización de trama. En cada uno de los casos se hace el salto a cada una de las subrutinas que se verán más delante.

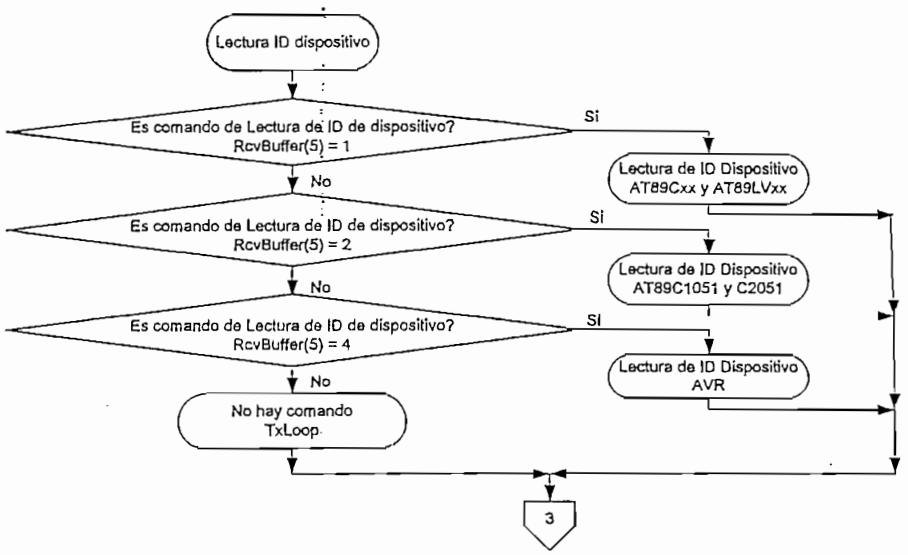



Figura 4.3: Diagrama de flujo continuación de inicio ProgAtmel.asm

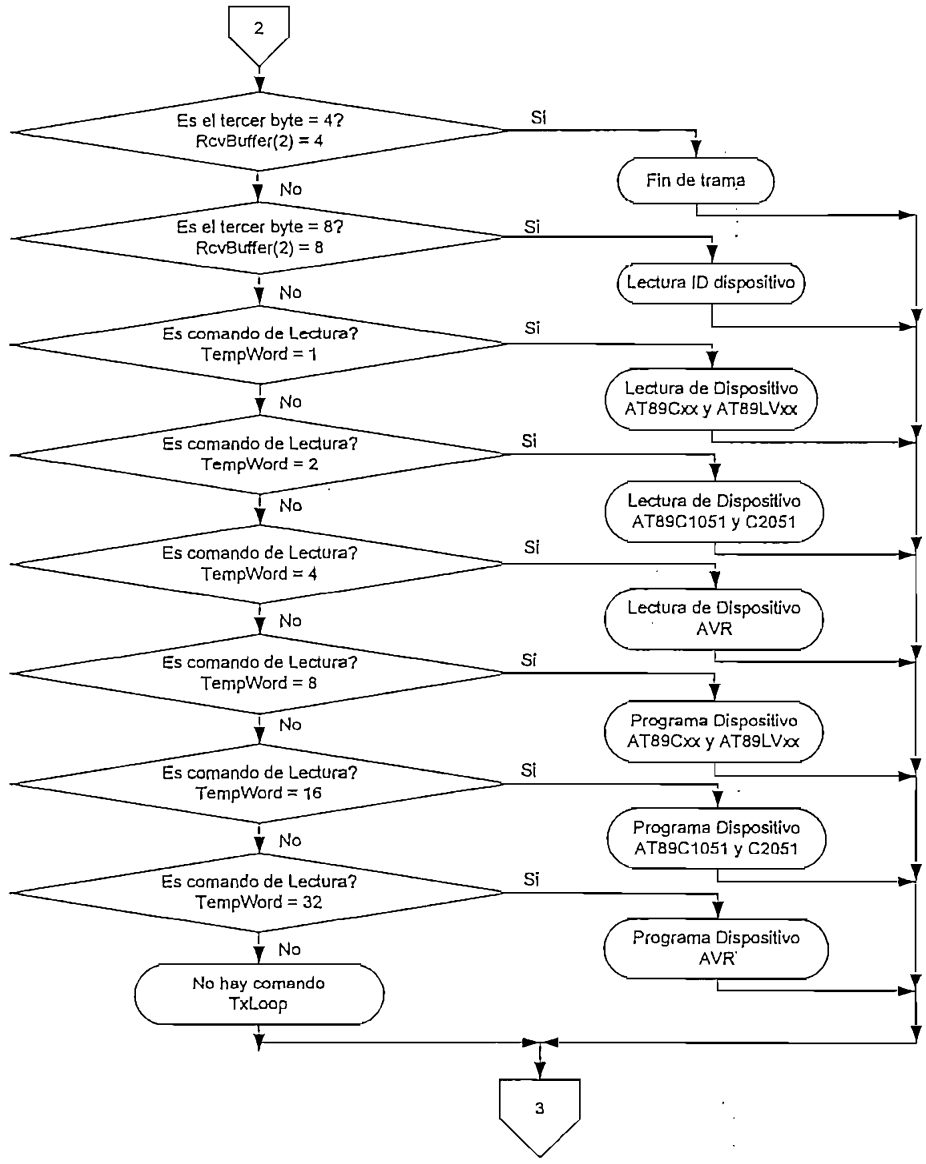



Figura 4.4: Diagrama de flujo continuación de inicio ProgAtmel.asm

*4.1.2.1.2 Subrutinas de borrado.*

En el Figura 4.5 se muestran las subrutinas de borrado para los diferentes microcontroladores. A continuación se describen los pasos que se deben seguir en cada una de las subrutinas de borrado:

El procedimiento para el borrado de los microcontroladores AT89Cxx y AT89LVxx::

- Carga comando de borrado en el puerto RA

- Habilita EA/VPP a 5V
- Espera 12 microsegundos
- Habilita EA/VPP a 12 V
- Espera 12 microsegundos
- Pone en bajo ALE/PROG
- Espera 10 milisegundos
- Pone en alto ALE/PROG
- Habilita fuente a 5V
- Deshabilita EA/VPP
- Salta a subrutina de Borrado OK

El procedimiento para el borrado de los microcontroladores AT89C1051 y AT89 C2051:

- Carga comando de borrado en el puerto RA
- Pone RST/VPP en 0V.
- Mantiene XTAL1 en bajo.
- Espera 12 microsegundos.
- Habilita RST/VPP a 12 V.
- Espera 12 microsegundos.
- Pone en bajo PROG.
- Espera 10 milisegundos.
- Pone en alto PROG.
- Habilita fuente a 5V.
- Deshabilita RST/VPP
- Salta a subrutina de Borrado OK

El procedimiento para el borrado de los microcontroladores AVR:

- Inicializa al puerto RD como salida
- Inicializa modo de programación
- Mantiene RESET en 0V.
- Espera de 2 microsegundos.
- Habilita RST/VPP a 12 V.

- Pone en alto XA1
- Carga comando de borrado AVR (10000000) en el Puerto RD
- Manda un pulso positivo en XTAL1.
- Pone en bajo WR.
- Espera 10 milisegundos.
- Pone en alto WR.
- Habilita fuente a 5V.
- Deshabilita RST/VPP
- Salta a subrutina de Borrado de dispositivo OK

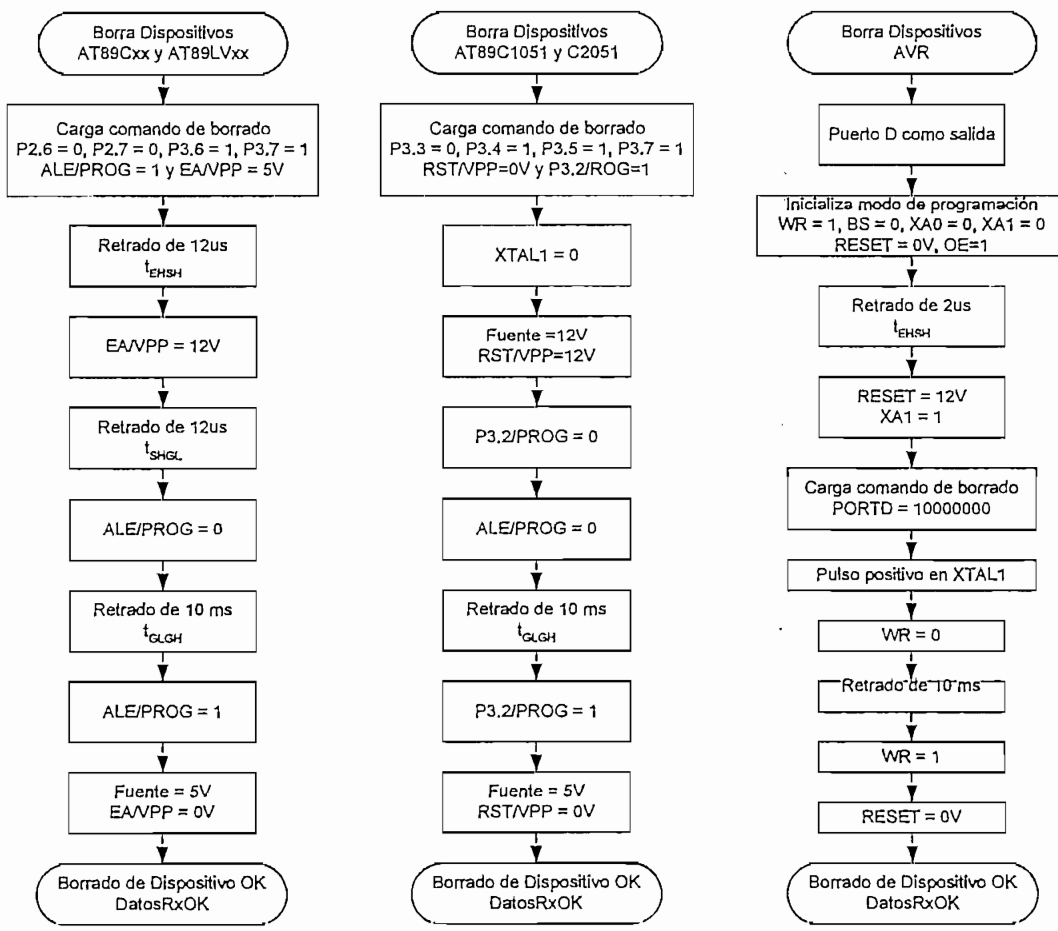



Figura 4.5: Diagramas de flujos de las subrutinas de borrado.

*4.1.2.1.3 Subrutinas de Inicialización de los modos de Programación o Lectura y de Finalización de los diferentes modos.*

En las subrutinas del Figura 4.6 se indican los pasos a seguir en las subrutinas de los diferentes modos de inicialización tanto para la programación como para la Lectura de un dispositivo. Para el caso de finalización de los diferentes modos se tiene una subrutina que se la nombro como FIN.

El procedimiento para inicializar el modo de programación y Lectura de los microcontroladores AT89Cxx y AT89LVxx:

- Inicializa al puerto RD como salida
- Habilita fuente a 5V
- Carga comando de Lectura en el puerto RA
- Habilita EA/VPP a 5V
- Salta a subrutina de Inicialización OK

El procedimiento para inicializar el modo de programación y Lectura de los microcontroladores AT89C1051 y AT89C2051:

- Inicializa al puerto RD como salida.
- Habilita fuente a 5V.
- Carga comando de Lectura en el puerto RA
- Mantiene RST/VPP en 0V.
- Salta a subrutina de Inicialización OK

El procedimiento para inicializar el modo de programación y Lectura de los microcontroladores AVR:

- Inicializa al puerto RD como salida.
- Habilita fuente a 12V.
- Inicializa modo de programación
- Mantiene RESET en 0V.
- Carga en registro de dirección alta con FFh.
- Salta a subrutina de Borrado de dispositivo OK

Procedimiento para la finalización de cualquiera de los modos ya sea de programación, Lectura, Lectura de byte descriptores e incluso del modo de borrado:

- Enceramiento de todos los registros como por ejemplo el registro TempWod, los registros de incremento de direcciones, etc.
- Habilita la fuente a 5V
- La señal Vpp en 0V
- Salta a subrutina de finalización OK

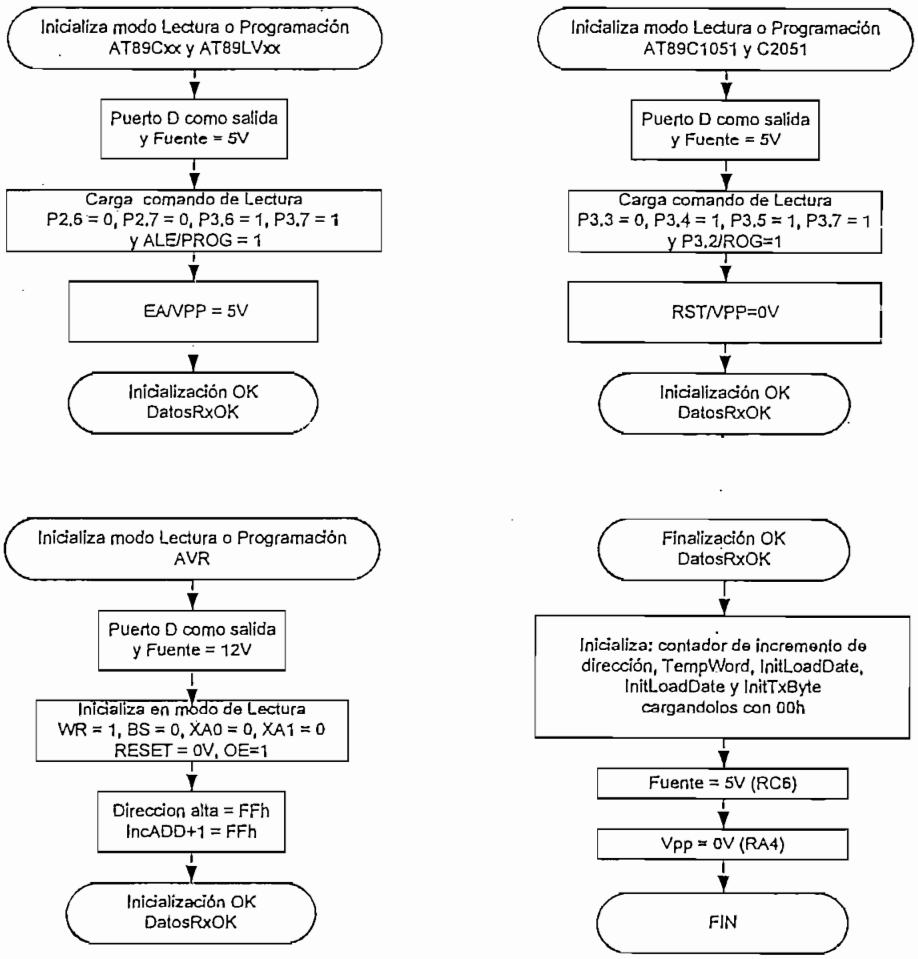



Figura 4.6: Diagramas de flujos de las subrutinas de modos de inicialización de Lectura de datos o programación y de fin de modos.

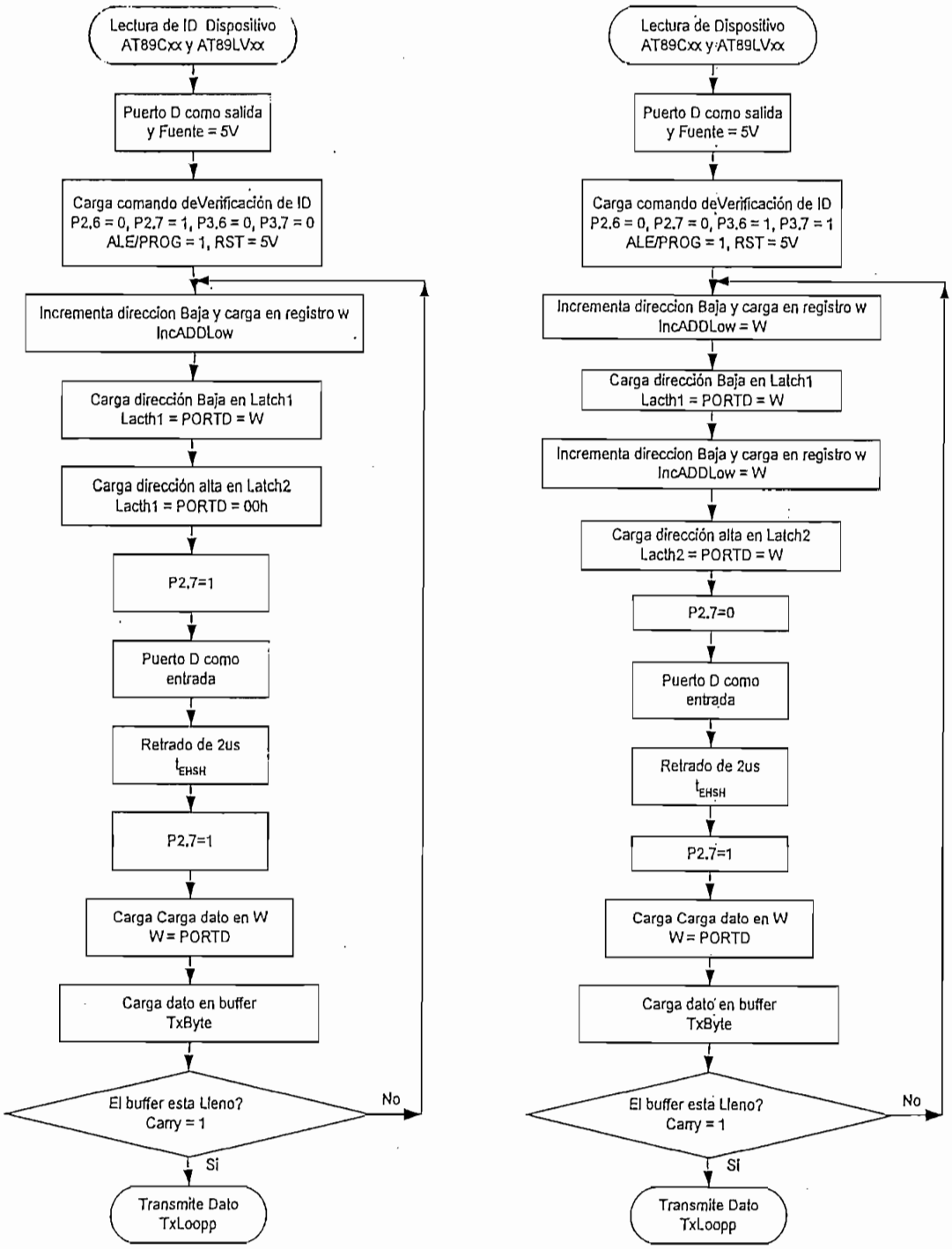



Figura 4.7: Diagrama de flujo de las subrutinas de Lectura de datos y Lectura de bytes descriptores para los microcontroladores AT89Cxx y AT89LVxx

*4.1.2.1.5 Subrutinas de Lectura de datos y Lectura de los bytes descriptores para los microcontroladores AT89C1051 y AT89C2051.*

Al igual que el caso anterior las verificaciones de los datos y de los bytes descriptores siguen el mismo proceso con variantes en la carga de comandos.

El procedimiento a seguir en la Lectura se presenta en el Figura 4.8 los cuales se describen a continuación:

- Inicializa al puerto RD como entrada.
- Habilita fuente a 5V.
- Carga comando de Lectura de datos o Lectura de bytes descriptores en el puerto RA
- Mantiene RST/VPP en 0V. Solo para el caso de Lectura de bytes descriptores.
- Espera 12 microsegundos. Solo para el caso de Lectura de bytes descriptores.
- Habilita RST/VPP a 5V
- Espera 12 microsegundos.
- Carga dato en registro W
- Carga dato en buffer de transmisión de datos
- Espera 2 microsegundos
- Verifica si el buffer esta lleno. En el caso que aun no este lleno repite el proceso, caso contrario salta a subrutina de envío de datos.

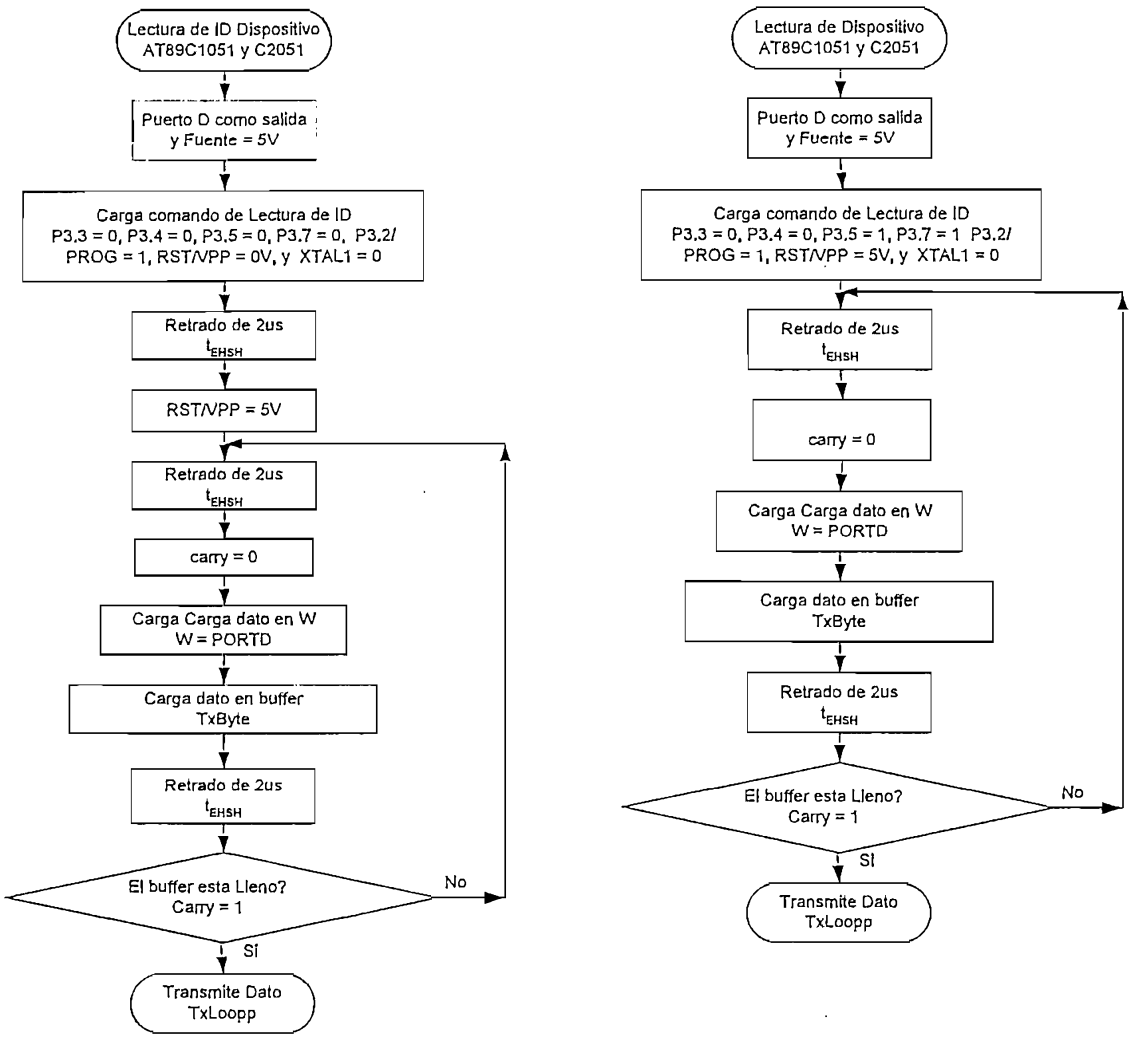



Figura 4.8: Diagrama de flujo de las subrutinas de Lectura de datos y Lectura de bytes descriptores para los microcontroladores AT89C1051 y AT89C2051

*4.1.2.1.6 Subrutinas de Lectura de datos y Lectura de los bytes descriptores para los microcontroladores AVR.*

Al igual que el caso anterior las verificaciones de los datos y de los bytes descriptores siguen el mismo proceso con variantes en la carga de comandos.

El procedimiento a seguir en la Lectura se presenta en el Figura 4.9 los cuales se describen a continuación:

- Inicializa al puerto RD como salida
- Inicializa modo de programación

- Mantiene RESET en 0V.
- Espera de 2 microsegundos.
- Habilita RESET a 12 V.
- Poner en alto XA1
- Carga comando de lectura de datos AVR (00001000) o lectura de bytes descriptores (00000010) en el Puerto RD según el caso
- Envía un pulso positivo en XTAL1.
- Incrementa registro de incremento de dirección alta y carga el contenido en el registro W y a su vez al puerto RD. Este paso es solamente para el modo de lectura de datos.
- Envía un pulso positivo en XTAL1. Este paso es solamente para el modo de lectura de datos
- Pone en bajo BS y XA1.
- Incrementa registro de incremento de dirección baja y carga el contenido en el registro W y a su vez al puerto RD.
- Envía un pulso positivo en XTAL1
- Pone en bajo OE.
- Esperar 550 microsegundos.
- Inicializa puerto RD como entrada
- Cargar dato en registro W y carga dato en buffer de transmisión de datos.
- Pone en alto BS.
- Cargar dato en registro W y carga dato en buffer de transmisión de datos.
- Pone en alto OE.
- Verificar si el buffer esta lleno. En el caso que aun no este lleno repite el proceso, caso contrario salta a subrutina de envío de datos.

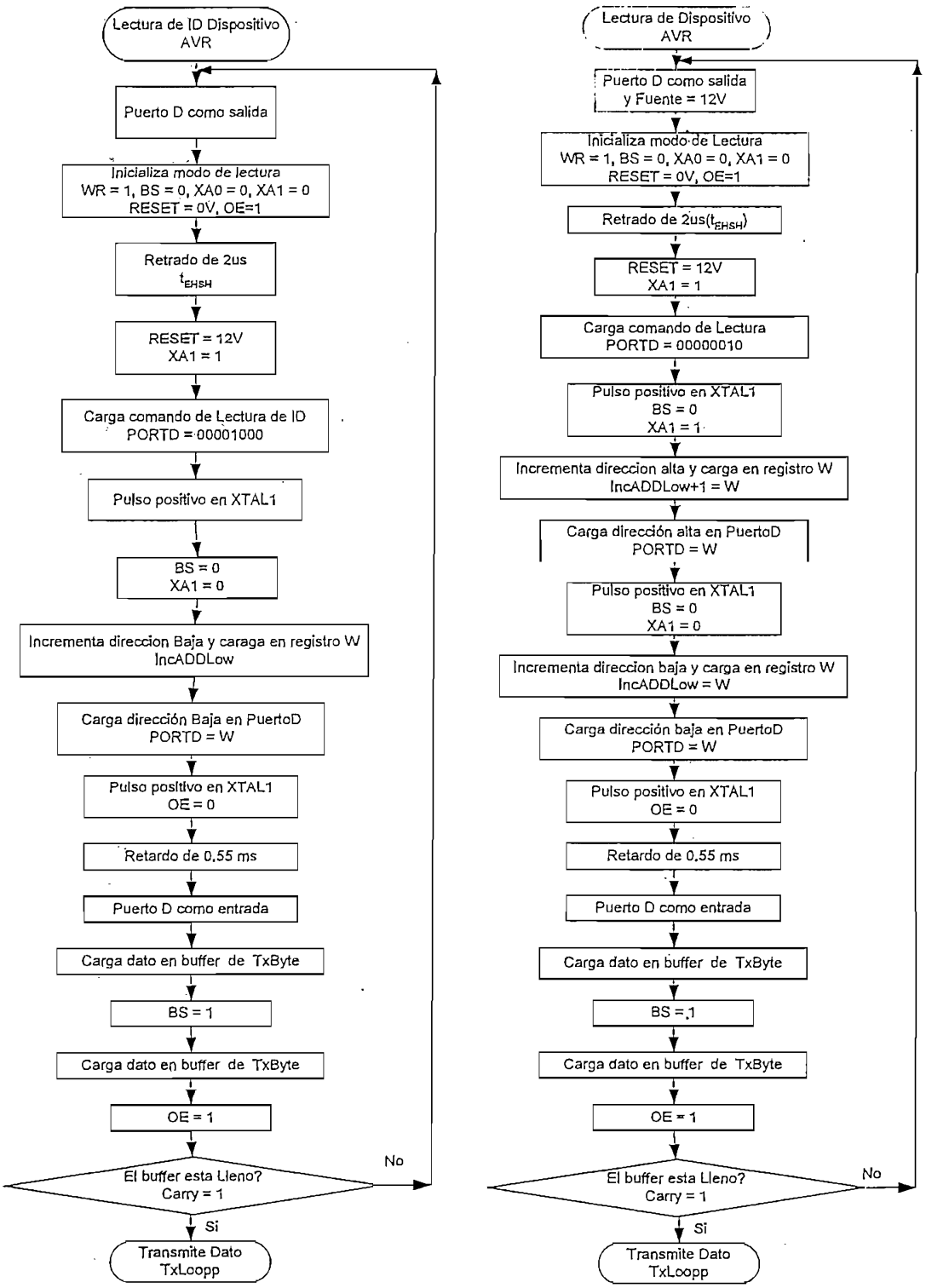



Figura 4.9: Diagrama de flujo de las subrutinas de Lectura de datos y Lectura de bytes descriptores para los microcontroladores AVR.

*4.1.2.1.7 Subrutinas de programación de datos y programación de los bits de seguridad para los microcontroladores AT89Cxx y AT89LVxx.*

La programación de los datos y de los bits de seguridad utilizan la misma rutina teniéndose las variantes en el instante de cargar el comando.

El procedimiento a seguir en la programación se presenta en el Figura 4.10 los cuales se describen a continuación:

- Inicializar al puerto RD como salida.
- Habilitar fuente a 5V.
- Cargar comando de programación de datos o programación de bits de seguridad en el puerto RA
- Habilita EA/VPP a 5V
- Chequea si el buffer de salida esta vacío. Si esta vacío salta a subrutina de programación OK, caso contrario sigue con el proceso.
- Esperar 12 microsegundos
- Habilita EA/VPP a 12V
- Incrementa registro de incremento de dirección baja y carga el contenido en el registro W y este a su vez en el Lacth1.
- Incrementa registro de incremento de dirección alta y carga el contenido en el registro W y este a su vez en el Lacth2.
- Llama a subrutina de recepción de dato y pone dato en W y a su vez este carga dato en el puerto RD
- Espera 12 microsegundos.
- Poner en bajo la señal ALE/PROG.
- Espera 12 microsegundos.
- Poner en alto la señal ALE/PROG.
- Espera 12 microsegundos.
- Inicializar al puerto RD como entrada.
- Habilita EA/VPP a 5V

- Espera hasta que RDY/BSY pase del estado al estado alto. Esta señal indica que el dato fue grabado correctamente. Si cumple la condición sigue el proceso.
- Verificar si el buffer esta vacío. En el caso que aun no este vacío repite el proceso, caso contrario salta a subrutina de programación OK.

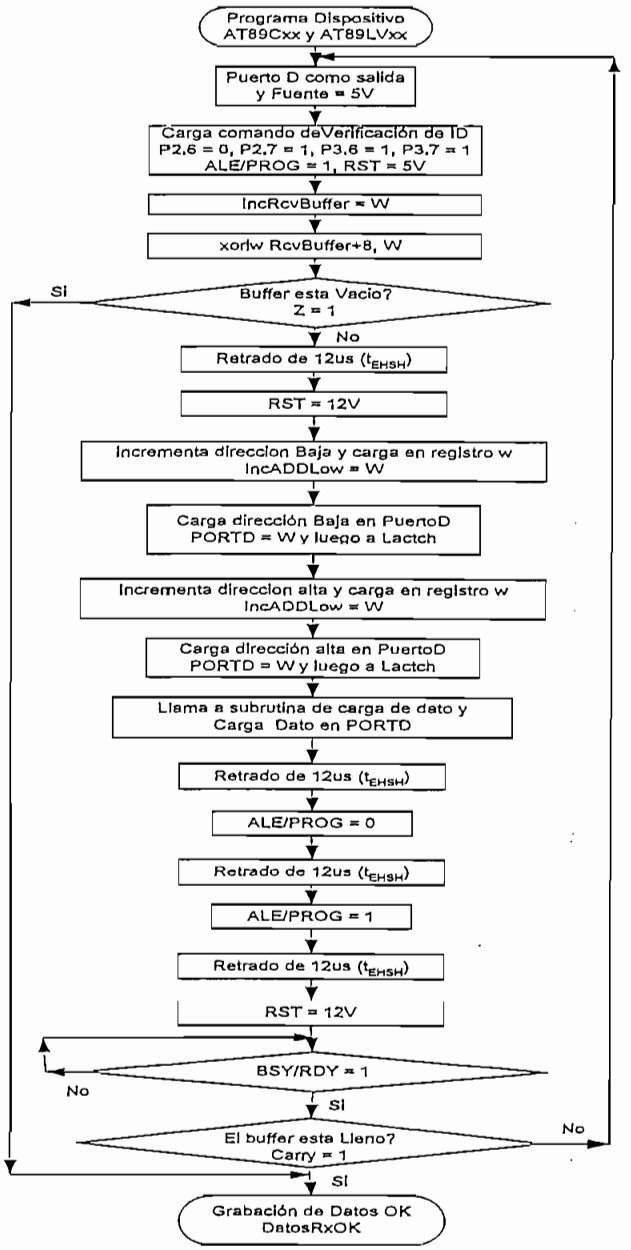



Figura 4.10 Diagrama de flujo de la subrutina de programación de los microcontroladores AT89Cxx y AT89LVxx.

*4.1.2.1.8 Subrutinas de programación de datos y programación de bits de seguridad para los microcontroladores AT89C1051 y AT89LV2051.*

Al igual que el caso anterior la programación de los datos y de los bits de seguridad siguen el mismo proceso con variantes en la carga de comandos.

El procedimiento a seguir en la programación se presenta en el Figura 4.10 los cuales se describen a continuación:

- Inicializar al puerto RD como salida
- Habilitar fuente a 5V.
- Cargar comando de programación de datos o programación de bits de seguridad en el puerto RA.
- Esperar 2 microsegundos.
- Pone RST/VPP en 12V
- Esperar 2 microsegundos.
- Llama a subrutina de recepción de dato y pone dato en W y a su vez este carga dato en el puerto RD.
- Chequea si el buffer de entrada esta vacío. Si esta vacío salta a subrutina de programación OK, caso contrario sigue con el proceso.
- Esperar 2 microsegundos.
- Pone en bajo ALE/PROG.
- Esperar 2 microsegundos.
- Pone en alto ALE/PROG.
- Esperar 2 microsegundos.
- Cargar dato en registro W.
- Cargar dato en buffer de transmisión de datos.
- Esperar 2 microsegundos.
- Pone RST/VPP en 5V.
- Espera hasta que RDY/BSY pase del estado al estado alto. Esta señal indica que el dato fue grabado correctamente. Si cumple la condición sigue el proceso.

- Verifica si el buffer esta vacío. En el caso que aun no este vacío repite el proceso, caso contrario salta a subrutina de programación OK.

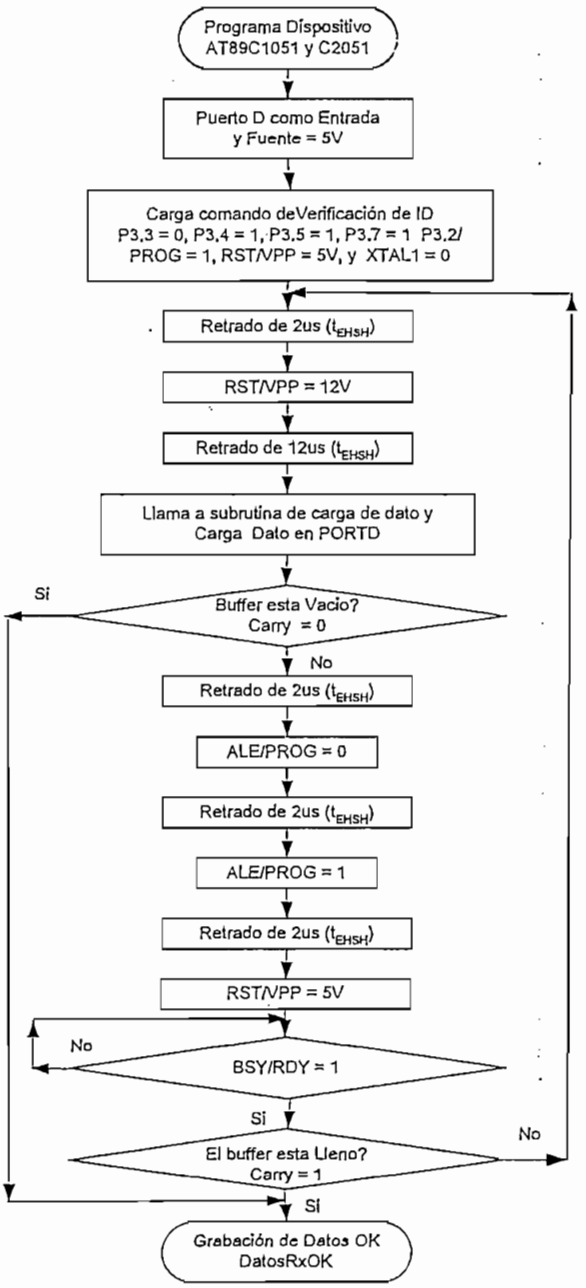



Figura 4.11: Diagrama de flujo de la subrutina de programación de los microcontroladores AT89C1051 y AT89C2051.

*4.1.2.1.9 Subrutinas de programación de datos y programación de bits de seguridad para los microcontroladores AVR.*

En los microcontroladores AVR la programación varia en relación a la Lectura ya que aquí se consideran varios pasos adicionales los cuales son descritos en el diagrama de flujo de la figura 4.12 cuyos pasos se describen a continuación:

- Inicializar al puerto RD como salida
- Habilita fuente a 12V
- Inicializar modo de programación
- Habilita RESET en 12V.
- Chequea si el buffer de entrada esta vacío. Si esta vacío salta a subrutina de programación OK, caso contrario sigue con el proceso.
- Cargar comando de programación de datos AVR (00010000) o programación de bits de seguridad (00100000) en el Puerto RD según el caso
- Envía un pulso positivo en XTAL1.
- Pone en alto BS y en bajo XA1.
- Incrementa registro de incremento de dirección alta y carga el contenido en el registro W y a su vez al puerto RD. Este paso es solamente para el modo de programación de datos.
- Envía un pulso positivo en XTAL1. Este paso es solamente para el modo de programación de datos.
- Pone en bajo BS.
- Incrementa registro de incremento de dirección baja y carga el contenido en el registro W y a su vez al puerto RD.
- Envía un pulso positivo en XTAL1
- Pone en bajo alto XA1
- Llama a subrutina de recepción de dato y pone dato en W y a su vez este carga dato en el puerto RD.

- Chequea si el buffer de entrada esta vacío. Si esta vacío salta a subrutina de programación OK, caso contrario sigue con el proceso.
- Envía un pulso positivo en XTAL1.
- Pone en bajo WR.
- Espera 220 nanosegundos.
- Pone en alto WR.
- Espera hasta que RDY/BSY pase del estado al estado alto. Esta señal indica que el dato fue grabado correctamente. Si cumple la condición sigue el proceso.
- Llama a subrutina de recepción de dato y pone dato en W y a su vez este carga dato en el puerto RD.
- Envía un pulso positivo en XTAL1.
- Pone en bajo WR y en alto BS
- Espera 200 microsegundos.
- Pone en alto WR
- Espera hasta que RDY/BSY pase del estado al estado alto. Esta señal indica que el dato fue grabado correctamente. Si cumple la condición sigue el proceso.
- Verifica si el buffer esta vacío. En el caso que aun no este vacío repite el proceso, caso contrario salta a subrutina de programación OK.

*4.1.2.1.10 Subrutinas de Recepción de datos OK, borrado de datos OK, No hay datos y transmisión de datos.*

El objetivo de estas subrutinas es enviar respuesta al PC en el registro del buffer de salida (RCVBuffer(0) ), como indicación de que el dato recibido o el comando recibido llegó con éxito o en caso contrario el dato no fue interpretado. Todas estas subrutinas apuntan hacia una subrutina de transmisión de datos. Esta subrutina es una implementación de la función PutEpn que se encarga de poner el dato en el buffer USB de salida. La bandera del carry indica si la transmisión de datos hacia el host terminó.

En la figura 4.13 se puede ver con más detalle, lo expuesto anteriormente.

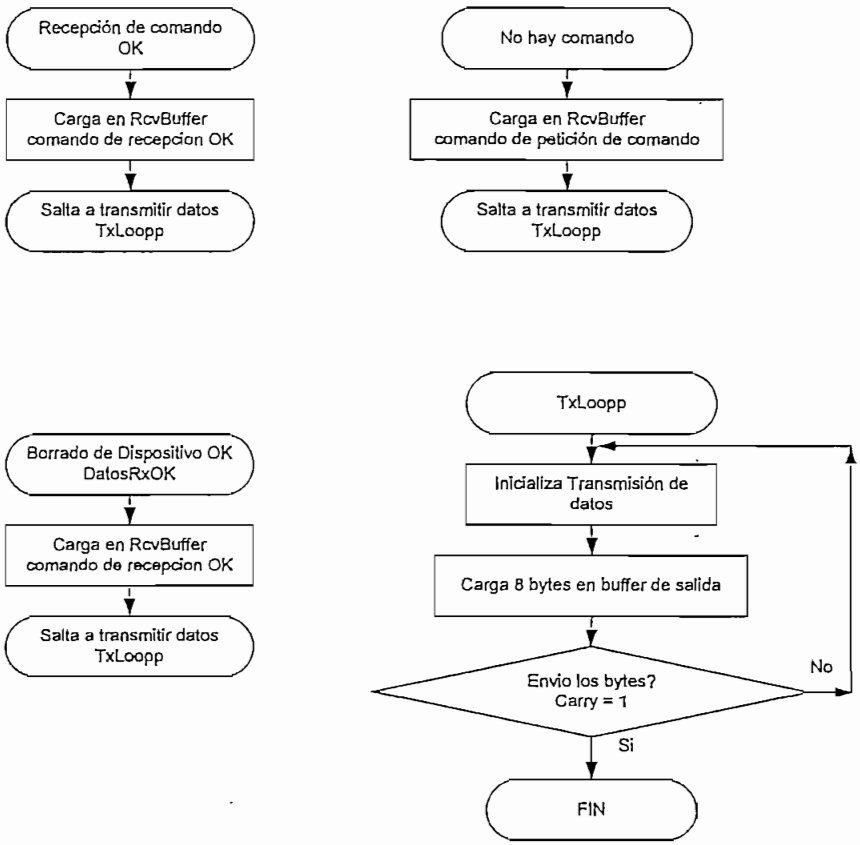



Figura 4.13 Diagrama de flujo de la subrutina de Recepción de datos OK. Borrado de datos OK, No hay datos y transmisión de datos

#### 4.1.2.2 Código fuente del programa ProgAtmel.asm

Este programa implementa la comunicación USB entre el programador y el host, tiene como base el firmware en ASM proporcionado por la Microchip Technology Inc. El código del firmware para el dispositivo USB en versión Assembler está integrado en el Anexo 2.

## 4.2 DESARROLLO DEL SOFTWARE EN EL PC

### 4.2.1 CRITERIOS GENERALES.

Cualquier aplicación que maneje un dispositivo USB en general, debe tener un Interfaz con el usuario a través de la PC y además debe permitir enviar o recibir datos del dispositivo USB con el fin de manejarlos.

USB es un estándar complejo que requiere una enorme cantidad de soporte de software tanto en dispositivos USB como en el host.

En al actualidad muchas aplicaciones de host que son corridas bajo el sistema operativo Microsoft, Windows 2000, Windows XP o posteriores tienen soporte USB. Windows 2000 y XP incluyen una gran cantidad de drivers para aplicaciones USB.

El driver USB de bajo nivel maneja la energía del dispositivo USB, su enumeración y varias transacciones USB. A través de este, el driver controlador del host conversa directamente con el hardware USB en el PC. Los dos drivers son soportados por las actuales versiones de Windows, a los que no se los debe modificar.

Windows, del mismo modo como lo hace la especificación USB, segmenta los drivers en *"clases"*, donde el hardware que está dentro de una misma clase poseen atributos o características similares. Una clase define la línea de fondo de la especificación para un conjunto de capacidades; todos los dispositivos requieren un comparable tipo de soporte de software. Un ejemplo es la Clase Human Interface

Device HID (Clase para Dispositivo de Interface Humana), la cual soporta dispositivos como mouses, joysticks, teclados, y otros dispositivos dedicados. Otra es la Clase Monitor, que controla posición de la imagen, espacio y alineación de video. El USB implementa este agrupamiento definiendo *Clases de dispositivos* y estableciendo para cada una de ellas la forma de accionar del driver y del dispositivo. Algunas de las clases definidas son: Human Interface Device (HID), Comm, Printer, Image, Mass Storage, Audio, etc.

Esta característica es una de las principales ventajas del USB, ya que evita que el usuario deba instalar distintos drivers y configurar el hardware de los distintos periféricos con las dificultades que esto representa. A pesar de esto, cada fabricante puede construir y ofrecer su propio driver para diferenciar su producto del de la competencia.

La definición de la clase para dispositivo de HID se creó para trabajar con la especificación USB y proporciona HID´s con suficiente información para construir drivers compatibles con dispositivos USB. La especificación completa de HID (Specification HID Rev 1.1) la puede encontrar en la página web www.usb.org.

Una vez establecida la comunicación entre el dispositivo y la PC, es necesario desarrollar herramientas para permitir la depuración del software desarrollado, lo cual se realizó en Visual BASIC 6.0. Y utilizando el soporte brindado por el Driver Development Kit ( DDK ) de Microsoft.

El Microsoft Windows Driver Development Kit (DDK) es un conjunto consolidado de desarrollo de drivers que proporciona información, ejemplos de drivers, cabeceras de drivers específicos, librerías, fuentes, ambiente de construcción y herramientas de soporte para el desarrollo de drivers en múltiples versiones del sistema operativo Windows. Windows DDK asiste a los desarrolladores en la elaboración de productos que interactúen con los sistemas operativos Windows. Para obtener el soporte brindado por Windows DDK en Windows 2000 es a través del ServicePack 3 o

superior y en el caso de Windows XP Profesional o Home Edition, es a través del ServicePack 1 o superior.
Los service Pack pueden obtenerse de la página web de Driver Development Kit www.microsoft.com/ddk/W2kDDK.asp.

Con la ayuda de Windows DDK y la interacción con Visual C++ o Visual Basic se pueden desarrollar HID's para manejar dispositivos externos USB desde el host. Así, se pueden diseñar interfaces de comunicación USB entre el host y el dispositivo USB para envío y recepción de paquetes de datos.

### 4.2.2 MICROSOFT VISUAL BASIC Y ACTIVE X

Microsoft Visual Studio es un paquete de desarrollo con herramientas de cabeceras, librerías y archivos fuente para el manejo de dispositivos externos conectados a puertos seriales y paralelos. La propiedad de Visual Basic hace posible que se puedan desarrollar software de comunicación utilizando controladores ActiveX.

Microchip provee el Controlador ActiveX HIDComm para el manejo del microcontrolador PIC16765 u otro dispositivo HID desde host. Este controlador puede ser usado en Visual Basic para seleccionar un dispositivo de Interface humana HID, y establecer su comunicación.

El control ActiveX HIDComm está diseñado para facilitar el proceso de comunicación USB con un dispositivo HID. Sin el control ActiveX HIDComm, los usuarios necesitan usar archivos Win32 API para comunicarse con los dispositivos HID.

Con HIDComm se pueden desarrollar Lectura y escritura de Reportes con el PIC16C765, sin embargo este control ActiveX se encuentra aun en una etapa de desarrollo.

El control ActiveX HIDComm viene empaquetado en el archivo setupex2.zip y puede ser bajado de la pagina Web www.microchip.com

El control ActiveX HIDComm es empaquetado con la tecnología InstallShield. Para instalarlo, simplemente se debe correr el archivo ejecutable "HIDCOMM_b.exe". El programa de instalación debe crear algunos atajos sobre el menú de inicio para este control. Para usar el control ActiveX simplemente inicie Visual Basic. Los detalles de cómo utilizar el control ActiveX HIDComm se encuentra en el Anexo 3. Dentro de las propiedades del control ActiveX HIDComm se tienen eventos y métodos los cuales son descritos a continuación.

#### 4.2.2.1 Métodos del control ActiveX HIDComm.

*Función Browse() tipo Long*

Busca a todos los dispositivos conectados al bus USB. El usuario puede escoger un dispositivo de entre todos los dispositivos conectados al bus USB.

*Función Connect() tipo Long*

Conecta a un dispositivo HID. El dispositivo HID a conectarse es determinado por las propiedades o criterios MatchXXX. El control ActiveX HIDComm debe estar conectado a un dispositivo HID para hacer alguna operación de entrada/salida. Si no se encuentra ningún dispositivo con las propiedades establecidas en el control ActiveX, la función Connect() falla y un código de estado es retornado para indicar la razón de la falla o del éxito de este método.

*Función ConnectToPath() tipo Long*

Conecta a un dispositivo HID dado una ruta. El control ActiveX HIDComm debe estar conectado a un dispositivo HID para hacer alguna operación de entrada/salida. Un código de estado es retornado para indicar la razón de la falla o del éxito de este método.

*Función Disconnect() tipo Long*

Desconexión de un dispositivo USB. Después de una desconexión, ninguna operación de entrada/salida puede ser llevada antes de establecer otra conexión.

*Función GetFeature(ByVal Size As Long) tipo Byte()*

Obtiene las características de reporte del dispositivo HID. Size es el número de bytes a leer. El reporte de características es retornado como un arreglo tipo byte. La variable ReportID determina que característica es direccionada por esta transacción.

*Función ReadFrom(ByRef Size As Long) tipo Byte()*

Lee de dispositivos HID. Size es el tamaño del buffer que almacena los datos. Retorna un arreglo de datos tipo byte que corresponden a los que el dispositivo HID entrega a host.

*Sub WriteTo(ByRef Buffer() As Byte, ByRef Size As Long)*

Escribe en el dispositivo HID, donde Buffer contiene los datos a ser enviados y Size es el número de bytes a enviar.

*Función SetFeature(ByRef Buffer() As Byte, ByVal Size As Long) tipo Boleana*

Coloca el reporte de característica del HID a un valor especificado del Buffer que contiene los datos a ser enviados y Size es el número de bytes a ser enviados. Si la operación es satisfactoria, la función retorna un valor verdadero (TRUE), caso contrario retorna Falso (FALSE).

*Sub Uninit()*

Esta función descarga las DLLs usadas por el control ActiveX. Se debe llamar a esta función al terminar un evento o para terminar el programa.

#### 4.2.2.2 Eventos del control ActiveX HIDComm

*Evento ConnectSuccess(ByVal Status As Long)*

Este evento se produce si el control ActiveX HIDComm conecta al dispositivo HID exitosamente.

### 4.2.3 DIAGRAMA DE PROCESOS DEL SOFTWARE EN EL PC

El software de cliente host ha sido desarrollado bajo la plataforma de Microsoft Visual Basic 6.0 Edición Empresarial y con la ayuda de las librerías de Windows DDK Setup y el Controlador ActiveX HIDComm.

El software que se lo denomino "USBFlashProgram" sigue los procesos que se describen a continuación:

En el instante de ejecutar el programa "USBFlashProgram" aparecerá la pantalla principal con el que se inicia el lazo principal. En el siguiente diagrama (figura 4.14) se observa el proceso desde el inicio del programa hasta finalizarlo. Dentro del programa se pueden seleccionar cualquiera de los procesos por medio de botones o desde el menú principal. El programa termina con la opción salir.

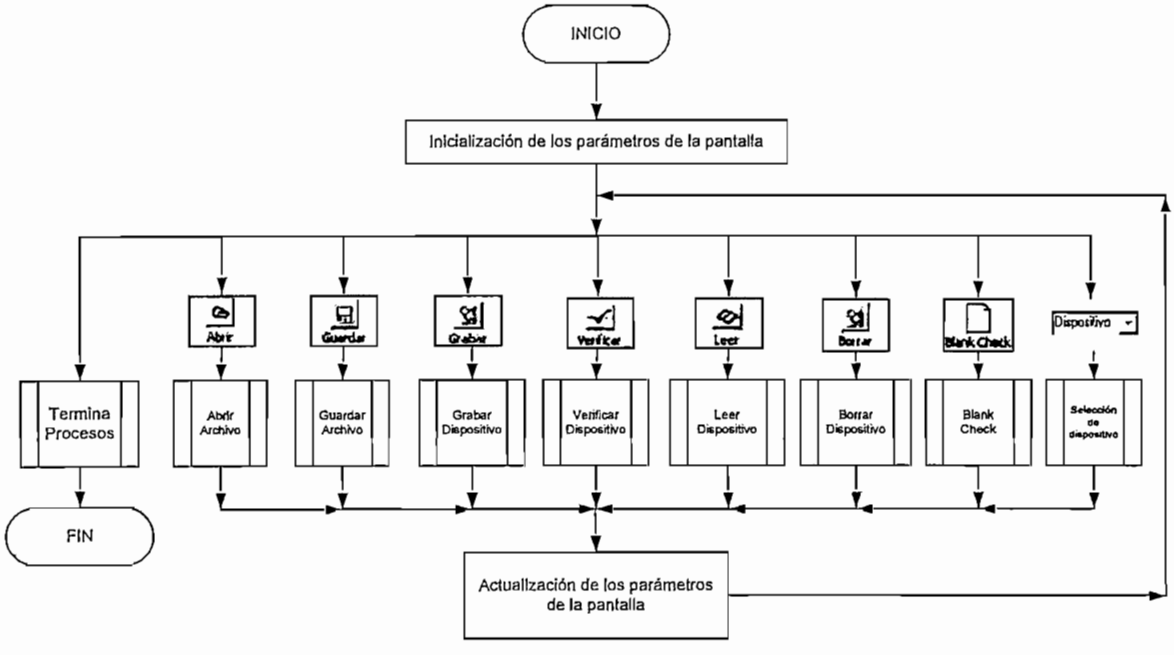



Figura 4.14: Diagrama de procesos del programa principal

Como una acotación en el momento en el que se mencione la palabra dispositivo se referirá al microcontrolador a programar y en el caso de mencionar microcontrolador se refiere al PIC16C765. De igual manera para Tx_Trama y Rx_Trama se indica que son tramas de 8 bytes que dependiendo de los casos van a ser comandos o datos.

#### 4.2.3.1 Diagrama de proceso de verifica_ID.

Al mencionar Verifica_ID nos estamos refiriendo al proceso de verificación de conexión del dispositivo, es decir que se hace una verificación inicial primero considerando si el usuario selecciono el dispositivo y luego comprobando si el dispositivo seleccionado es igual al dispositivo insertado en el programador.

Si el usuario no selecciono el dispositivo, en el menú se presenta el mensaje1 indicando que seleccione el dispositivo.

En el proceso de lectura de ID de dispositivo se envía Tx_trama que es un comando de lectura de ID de dispositivo. Como paso seguido el microcontrolador envía RX_Trama con los datos de identificación del dispositivo insertado. Con los datos recibidos se verifica si el dispositivo seleccionado corresponde al insertado. En el caso que estos no correspondan se presenta el mensaje2 que indica al usuario que debe revisar si el dispositivo esta bien insertado y luego termina el proceso. En caso de cumplirse la condición termina el proceso. En la figura 4.14 se presenta el proceso.

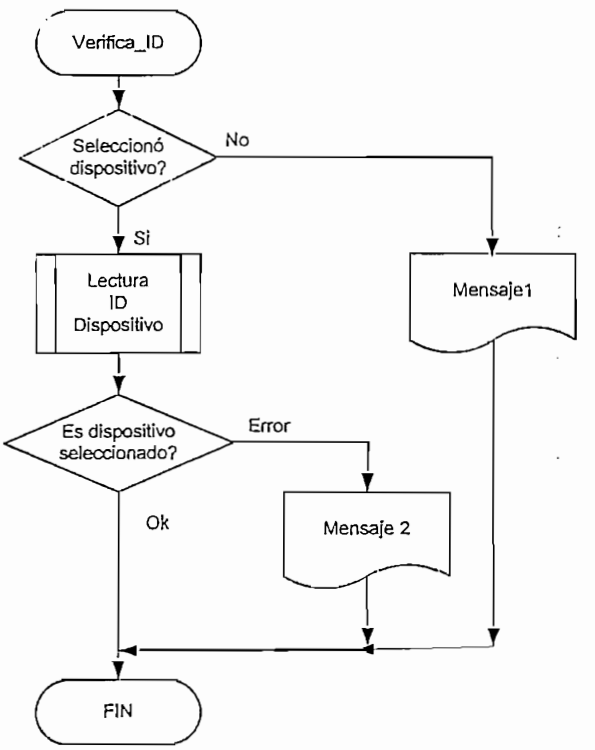



Figura 4.14: Diagrama de proceso de Verifica_ID.

4.2.3.2 Diagrama de proceso de borrado.

En el proceso de borrado tenemos los siguientes pasos:

Verificación de Identidad de dispositivo. Si el dispositivo seleccionado no corresponde al insertado se muestra el mensaje1 que indica al usuario que debe revisar si el dispositivo esta bien insertado y luego termina el proceso. Si cumple la condición entra en el proceso de borrado de dispositivo.

En este proceso se envía al microcontrolador en Tx_Trama el comando de borrado mientras que este retorna Rx_Trama indicando que el borrado fue exitoso o fue erróneo. Si el borrado fue correcto en Tx_Trama se envía el comando de finalización de borrado y luego se presenta el mensaje3 indicando que el dispositivo fue borrado exitosamente, caso contrario se presenta el mensaje2 indicando que hubo un error de borrado. En cualquiera de los dos casos se termina el proceso.

En la figura 4.15 se indica el diagrama de proceso para el borrado de los diferentes dispositivos.

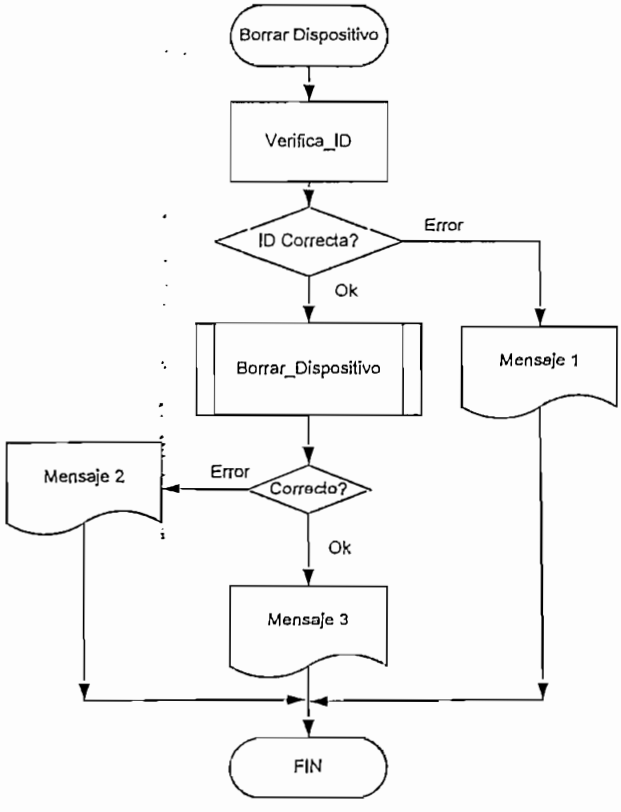



Figura 4.14: Diagrama de proceso de borrado

#### 4.2.3.3 Diagrama de proceso de BlankCheck.

En el proceso de blankCheck la verificación de ID del dispositivo es similar al anterior. Si el dispositivo seleccionado corresponde al insertado se continúa con el proceso de black_Check. Este consiste en enviar un comando de lectura de dispositivo en Tx_Trama, como paso siguiente el microcontrolador envía los datos en RX_Trama hasta que reciba una trama de finalización. Si todos los datos llegaron correctamente pasa al siguiente proceso que consiste en comparar los datos recibidos: Si los datos recibidos son todos FF indica que el dispositivo esta en blanco caso contrario el dispositivo no esta en blanco.

En la figura 4.15 se representa el proceso de BlankCheck.

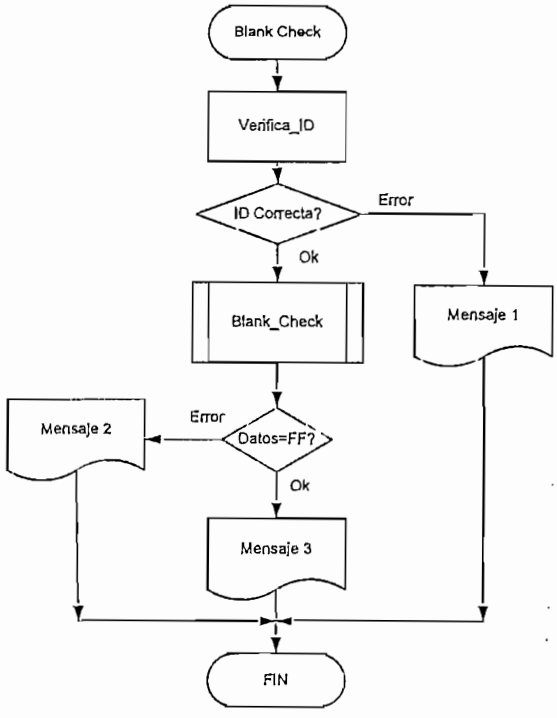



Figura 4.15: Diagrama de proceso de BlankCheck

#### 4.2.3.4 Diagrama de proceso de Lectura del dispositivo.

El proceso de lectura de dispositivo es similar al caso anterior con la diferencia que en lugar de comparar los resultados ahora se presentan en la pantalla los datos extraídos del dispositivo. El proceso de lectura de dispositivo es presentado en la figura 4.16.

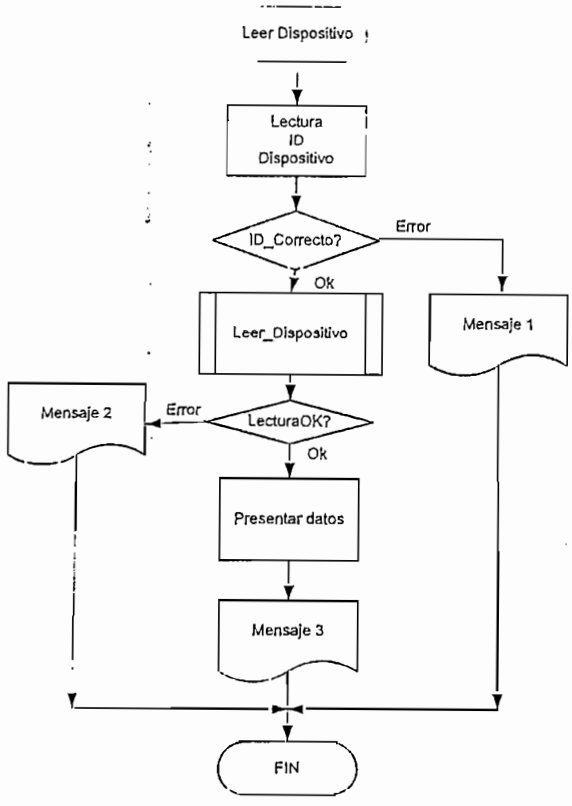



Figura 4.16: Diagrama de proceso de Lectura de dispositivo

#### 4.2.3.5 Proceso de Abrir archivo.hex.

Para extraer los datos del archivo.hex, se hace uso de Microsoft common Dialog 6.0 (SP3) que se encuentran ubicado en el directorio c:\windows\sistem32\comdlg32.ocxs. Los datos extraídos son almacenados en un vector. Como paso siguiente se muestra estos datos en la ventana de datos. El archivo.hex es en formato Intel. El detalle de este formato se presenta en el anexo 6.

#### Proceso de Guardar archivo.hex.

Para guardar los datos del archivo.hex se hace uso de la misma herramienta que en el caso anterior y se sigue el proceso inverso. Los datos a guardar pueden ser datos del archivo.hex abierto o de los datos extraídos del dispositivo.

### 4.2.3.6 Diagrama de proceso de verificación del dispositivo.

El proceso de verificación consiste en extraer los datos del dispositivo y compararlos con los datos del archivo.hex extraído. Si los datos extraídos no coinciden se presenta un mensaje3 indicando que la verificación no fue exitosa. Caso contrario presenta un mensaje4 indicando que la verificación fue exitosa y termina el proceso.

En la condición en la cual no se abrió el archivo.hex se presenta el mensaje1 indicando que no se encuentra abierto el archivo y termina el proceso. El diagrama de procesos para la de verificación datos del dispositivo se muestra en la figura 4.17.

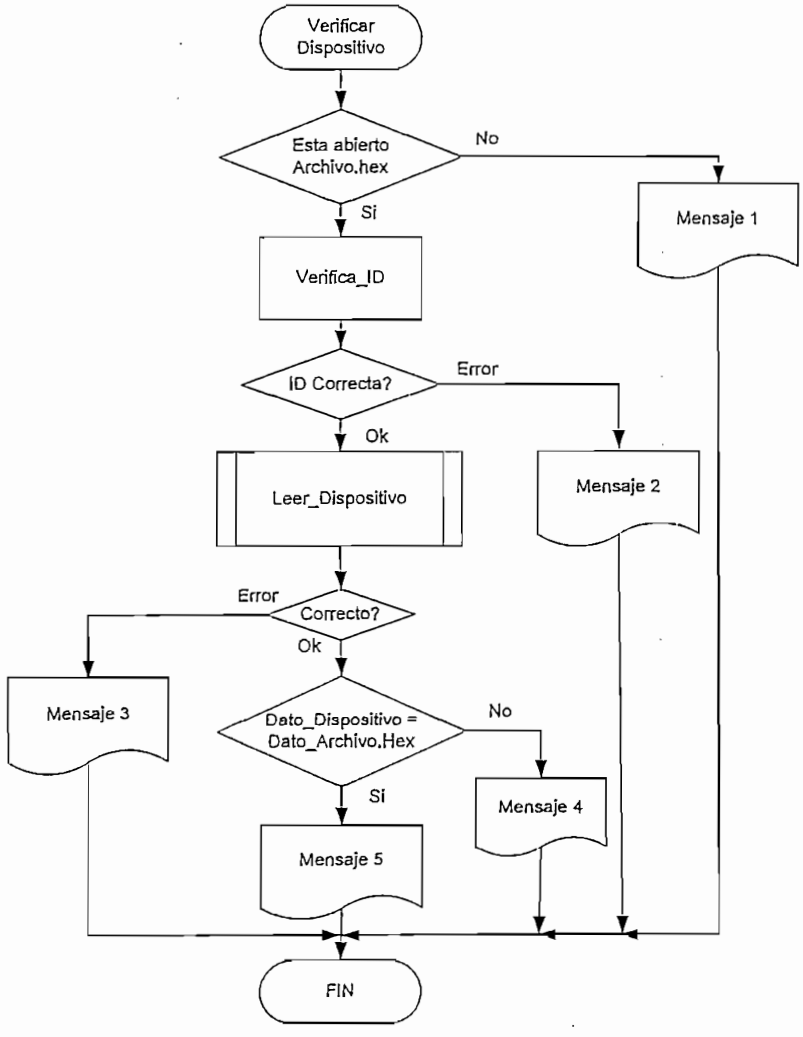



Figura 4.17: Diagrama de proceso de Verificación de dispositivo

#### 4.2.3.7 Diagrama de proceso de Grabación del dispositivo.

En el proceso de grabación se utilizan cada uno de los procesos vistos anteriormente. La primera condición que debe cumplir el proceso es verificar si el archivo.hex esta abierto. El siguiente paso es revisar si el dispositivo esta vacío. Para cualquiera de los dos casos anteriores si no se cumplen se presentan mensajes de error.

El proceso de grabar_Dispositivo consiste en enviar un comando de grabación en Tx_Trama al inicio del proceso y luego enviar todos los datos del archivo.hex al dispositivo. Una vez terminado el envió de datos se finaliza enviando el comando de finalización en Tx_trama. Luego de este proceso se verifica los datos grabados, si la verificación falla se presenta el mensaje que no se grabo y termina el proceso. Caso contrario presenta un mensaje de programación exitosa y termina el proceso.

El diagrama de procesos para la de grabación del dispositivo se muestra en la figura 4.18.

### 4.2.4 CÓDIGO DEL SOFTWARE "USB ATMEL FLASH PROGRAMADOR"

Este programa sirve de interfaz entre el usuario y el programador ATMEL USB, permitiendo el control desde una ventana amigable.

El código fuente del software "USB ATMEL Flash programador" se presenta en el Anexo 4.

# CAPÍTULO V

## PRUEBAS Y RESULTADOS

Las pruebas realizadas para determinar la efectividad del programador fueron muchas, dichas pruebas se realizaron programando, verificando, leyendo y borrando los microcontroladores AT89C51/52, AT89C1051/2051 y AT90S1200. Adicionalmente se verificó la programación de los microcontroladores implementando el circuito real, obteniéndose resultados satisfactorios. Como por ejemplo se programó el microcontrolador ATMEL89C2051 con el programa Manejapuertos.Hex el cual hace que se enciendan Leds conectados al puerto P0 en forma secuencial.

Las pruebas realizadas fueron en base a varios archivos hex. De las cuales como ejemplos presentamos las diferentes pruebas en base al archivo ManejaPuertos.hex, el cual se muestra a continuación en su formato Intel original.

```
:050000000201000832BE
:02000B000932B8
:020013000A32AF
:02001B000B32A6
:0C0023000C0C0C0C00DCFD7598000C327D
:1001000075A89F75B811758902758C0FD288D28C2D
:1001100020B726D2F37401F58023D5F0FAD2F37517
:100120008000F59023D5F0FAD2F3759000F5A02366
:10013000D5F0FA75A000020139D2F37480F5A0035E
:10014000D5F0FA75A000D2F3F59003D5F0FA7590CA
:0E01500000D2F3F58003D5F0FA75800080B57B
:00000001FF
```

El mismo archivo extraído por el software del programador se representa en la figura 5.1.

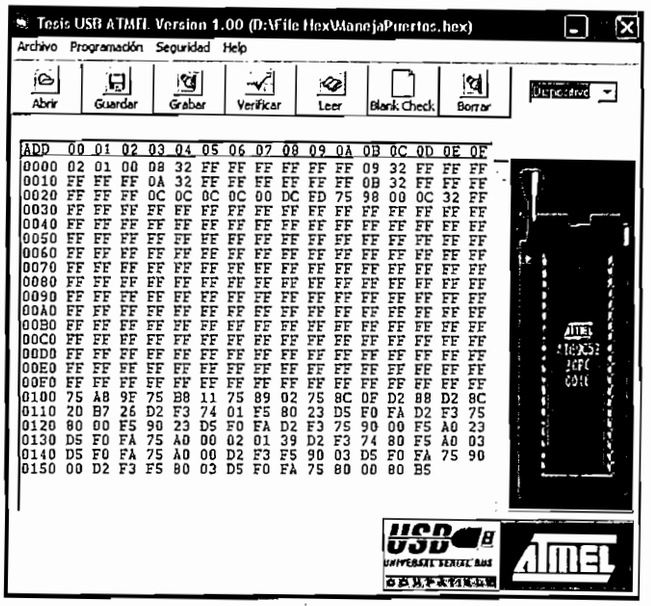



Figura 5.1: Archivo ManejaPuertos.hex extraído por el software del programador

Puesto que las pruebas efectuadas para los microcontroladores fueran las mismas y al ser los resultados similares se presentan tres ejemplos. Cada ejemplo seleccionado representa a la serie de los microcontroladores al que corresponde, es decir en el caso de los microcontroladores de la serie AT89Cxx/LVxx se selecciono como ejemplo al microcontrolador AT89C51, para el caso de los microcontroladores de la serie AT89C1051/2051 se seleccionó al microcontrolador AT89C1051 y de igual manera se realizó para los microcontroladores AVR.

## 5.1 PRUEBAS REALIZADAS CON EL MICROCONTROLADOR AT89C51.

Para realizar la prueba de grabación primero es necesario verificar si el dispositivo esta vacío. Razón por la cual se realizo el chequeo de la memoria flash con la opción leer. Esta opción puede ser ejecutada directamente desde la pantalla principal o desde el menú principal en la opción "Programación. La pantalla presentada luego de este evento se muestra en la figura 5.2.

La grabación puede ser comprobada de igual manera con la opción de verificación, La figura 5.5 presenta la pantalla mostrada luego de la verificación.

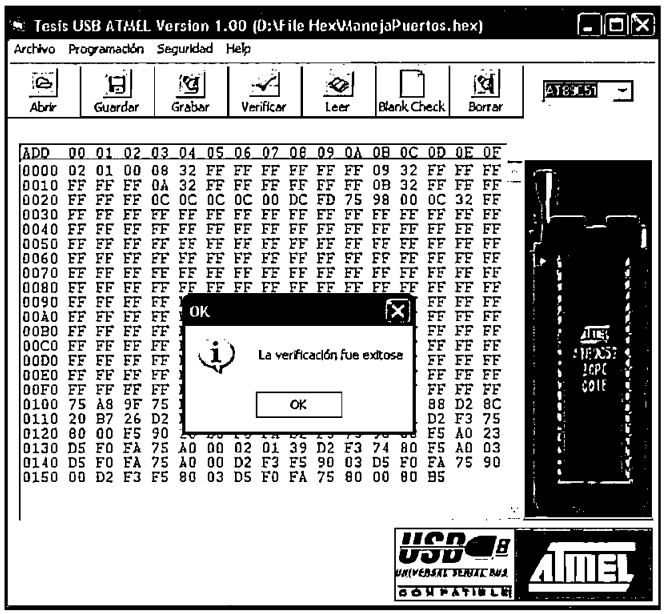



Figura 5.5: Pantalla de verificación exitosa.

## 5.2 PRUEBAS REALIZADAS CON EL MICROCONTROLADOR AT89C1051.

Los pasos seguidos en las pruebas para este dispositivo son similares a los del AT89C51. Los resultados obtenidos en las pruebas de borrado, Blank Check, verificación y grabación, para el microcontrolador AT89C1051 se muestran en las siguientes figuras.

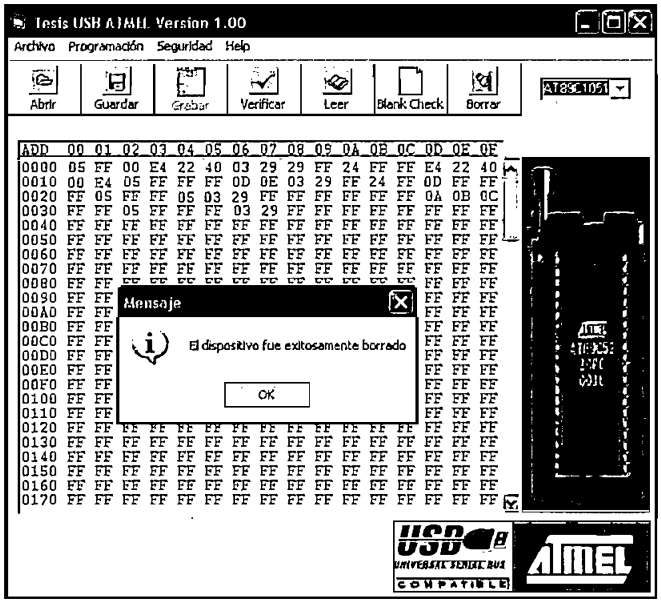



Figura 5.6: Pantalla luego del evento de borrado

La figura 5.8 muestra el resultado luego de haber ejecutado la opción de lectura de la memoria Flash después de la grabación.

## 5.3 PRUEBAS REALIZADAS CON EL MICROCONTROLADOR AT90S1200.

La secuencia seguida para la programación del dispositivo es similar a los casos anteriores. A continuación se describen los pasos seguidos.

Para revisar si la memoria flash del microcontrolador se encuentra vacío se ejecuto la opción de blank Check. En este caso la memoria se encontraba vacía, por lo cual no fue necesario borrar el dispositivo. El detalla de este evento se muestra en la figura 5.8.

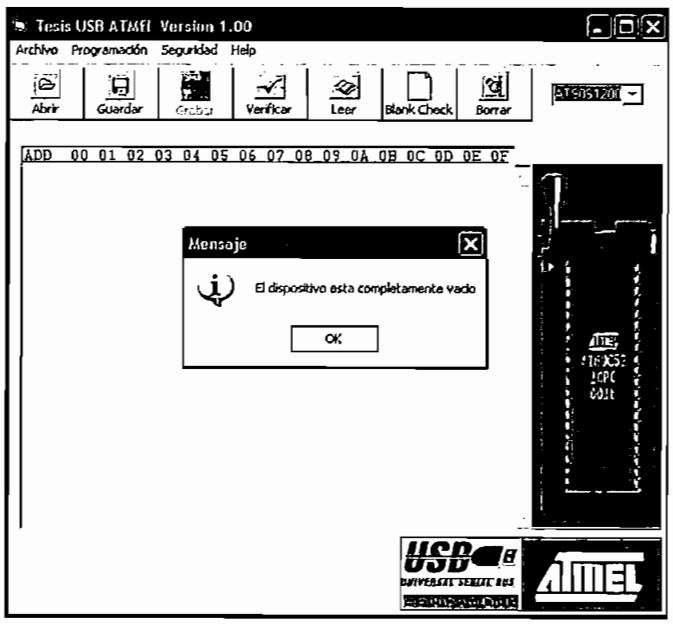



Figura 5.9: Pantalla de verificación de memoria Flash en blanco.

Luego del paso anterior se importo el archivo hex a la pantalla de datos y como paso seguido se grabo el dispositivo. La ventana obtenida es la siguiente.

Si se hace un clic en OK el mensaje que se presenta en pantalla es "Programador USB ATMEL no se ha encontrado". Ver figura 5.12


Figura 5.12: Ventana de error al hacer clic en OK

En este caso se recomienda que verifique que el conector serie A del programador este conectado correctamente.

El segundo caso se presenta cuando el programa ya ha sido ejecutado y se desconecta el programador. La ventana que aparece al ejecutar cualquiera de las opciones del programa (Grabar, Borrar, etc) es "Error al intentar leer el dispositivo Revise si el dispositivo esta bien conectado".

La solución es conectar el programador nuevamente.

**b. Se intenta grabar, borrar, leer o verificar el microcontrolador sin haber seleccionado el dispositivo en el menú de dispositivos.**

En este caso se presenta la siguiente pantalla con el mensaje: "Seleccione Dispositivo".Ver figura 5.13.

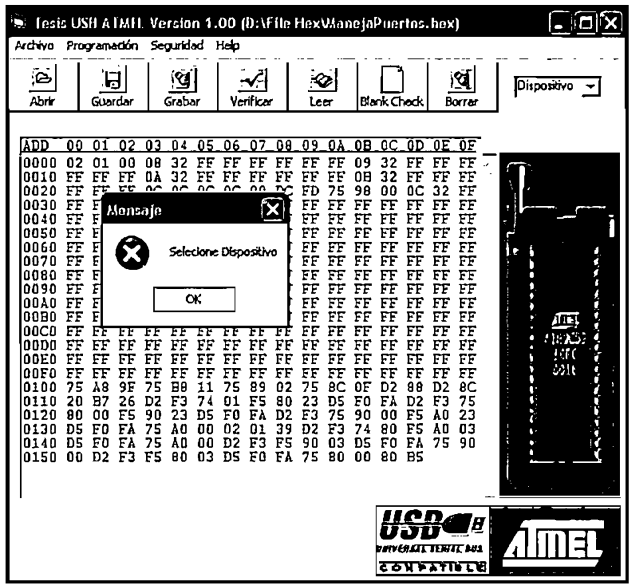



Figura 5.13: Pantalla de no selección de dispositivo

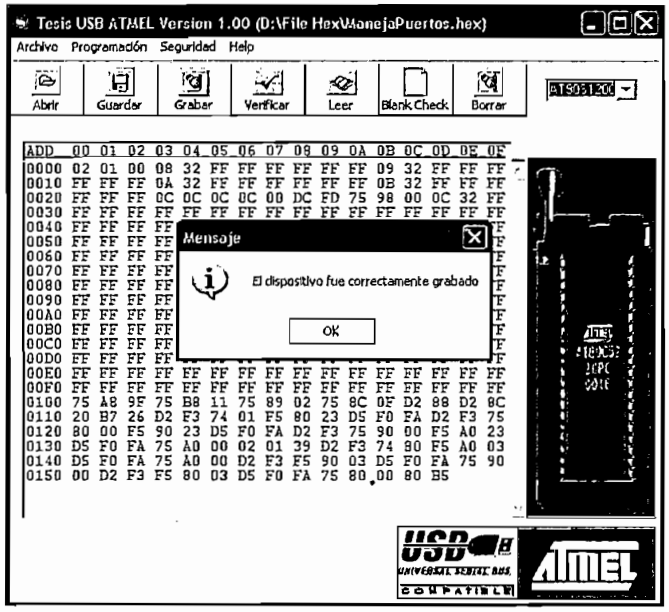



Figura 5.10: Pantalla de grabación en la memoria flash del microcontrolador AT90S1200

## 5.3 PRUEBAS QUE PRESENTAN MENSAJES DE ERROR.

A continuación se detallan los posibles mensajes de error que pueden presentarse al realizar alguna acción no permitida y de igual manera se presentan las soluciones a cada una de ellas.

**a. Cuando el programador no esta conectado.**

Cuando el programador no esta conectado se pueden tener dos posibilidades:

El primer caso ocurre si se ejecuta por primera vez el programa "USBFlashProgram.exe" y el programador no esta conectado. La ventana en aparecer es la siguiente.

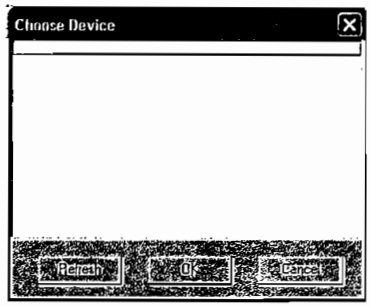



Figura 5.11: Ventana que aparece luego de ejecutar "USBFlashProgram.exe"

Si se hace un clic en OK el mensaje que se presenta en pantalla es "Programador USB ATMEL no se ha encontrado". Ver figura 5.12

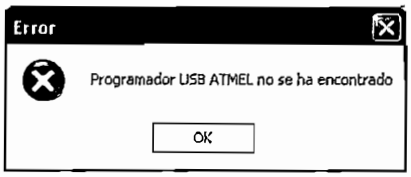



Figura 5.12: Ventana de error al hacer clic en OK

En este caso se recomienda que verifique que el conector serie A del programador este conectado correctamente.

El segundo caso se presenta cuando el programa ya ha sido ejecutado y se desconecta el programador. La ventana que aparece al ejecutar cualquiera de las opciones del programa (Grabar, Borrar, etc) es "Error al intentar leer el dispositivo Revise si el dispositivo esta bien conectado".

La solución es conectar el programador nuevamente.

**b. Se intenta grabar, borrar, leer o verificar el microcontrolador sin haber seleccionado el dispositivo en el menú de dispositivos.**

En este caso se presenta la siguiente pantalla con el mensaje: "Seleccione Dispositivo".Ver figura 5.13.

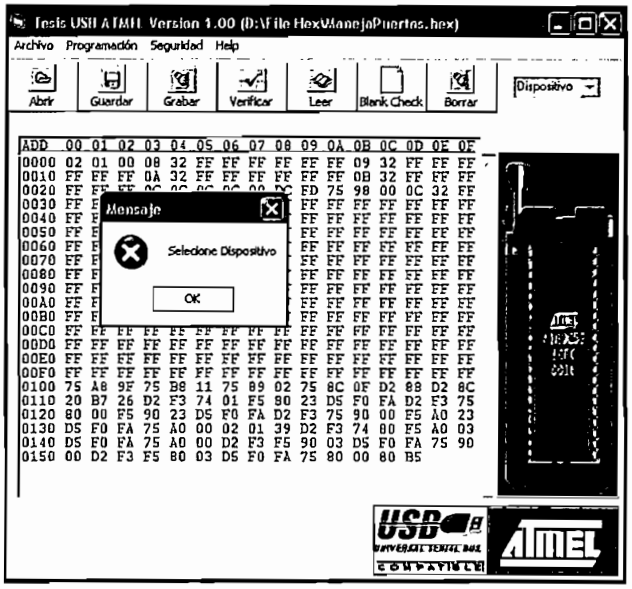



Figura 5.13: Pantalla de no selección de dispositivo

La solución a este caso es seleccionar el microcontrolador a programar en el menú de dispositivos.

**c. El microcontrolador a programar este dañado o no esta conectado apropiadamente.**

Para este caso el mensaje que se presenta en la pantalla es: "Error al intentar leer el dispositivo Verifique si el dispositivo esta bien conectado". Esta pantalla es presentada en la figura 5.14.

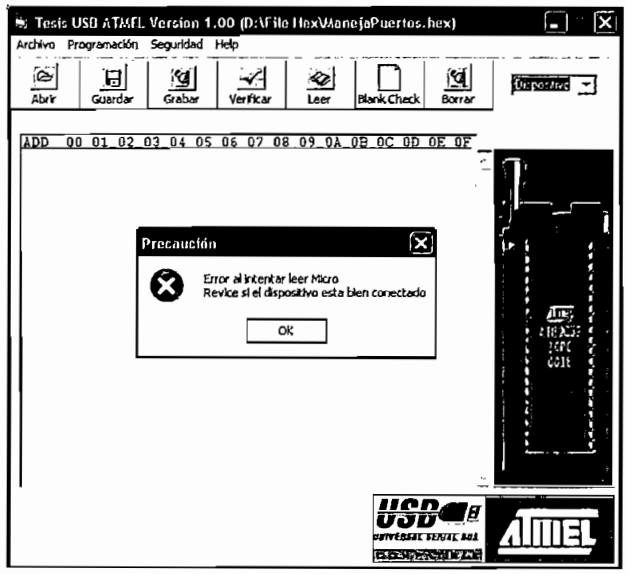



Figura 5.15: Pantalla de error de conexión

La solución a esta acción es verificar que el microcontrolador a programar este bien conectado o caso contrario reemplazarlo por otro.

**d. Si se hace la verificación sin haber importado el archivo.hex.**

La ventana con mensaje de error que aparece es el siguiente:

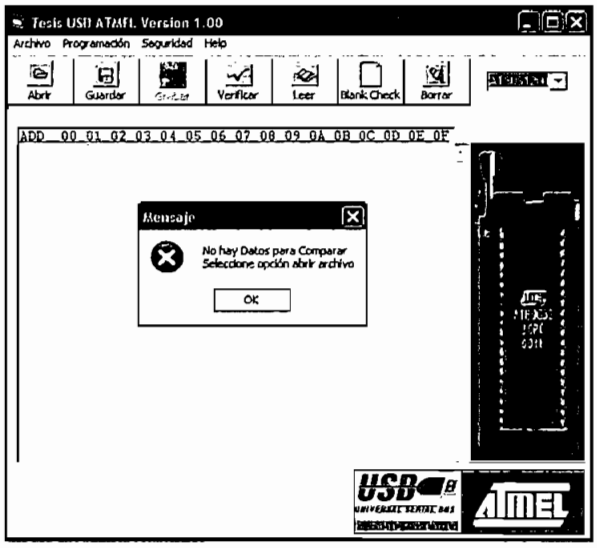



Figura 5.14: Mensaje de error si se intento verificar el dispositivo sin haber importado el archivo.hex

**d. Si se hace la verificación sin haber grabado la memoria flash del microcontrolador con el archivo.hex importado.**

El siguiente mensaje de error puede presentarse en el caso en que el usuario intente verificar la memoria flash del microcontrolador sin haber grabado el archivo importado, el mensaje presentado es "El dato de la dirección XXXX esta erróneo". Ver figura 5.15

Para este caso se debe grabar la memoria flash del microcontrolador a programar con el archivo.hex importado si se desea. Caso contrario no realizar ninguna acción.

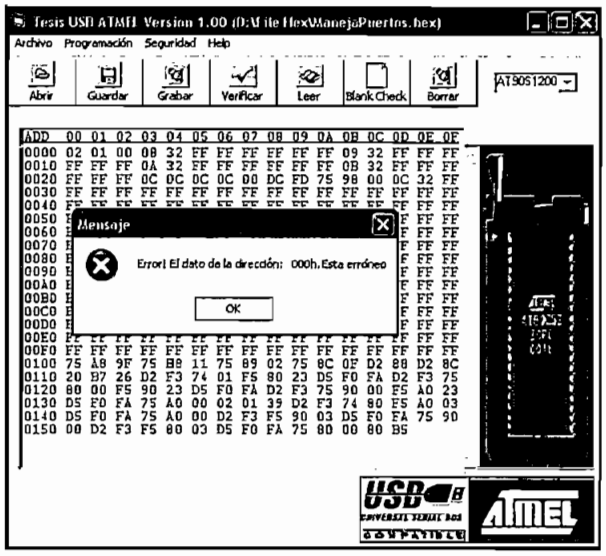



Figura 5.15: Mensaje de error si se intentó verificar el dispositivo sin haber importado el archivo.hex

**e. Si se abre un archivo.hex que no tiene la estructura del formato INTEL.**

Los mensajes presentados en la pantalla son los siguientes. Ver figura 5.16 y 5.17.

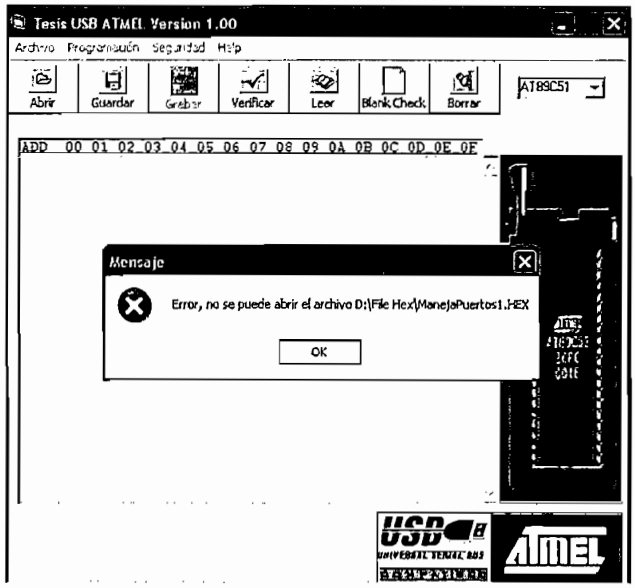



Figura 5.16: Ventana con mensaje de error al abrir archivo sin estructura de formato Intel

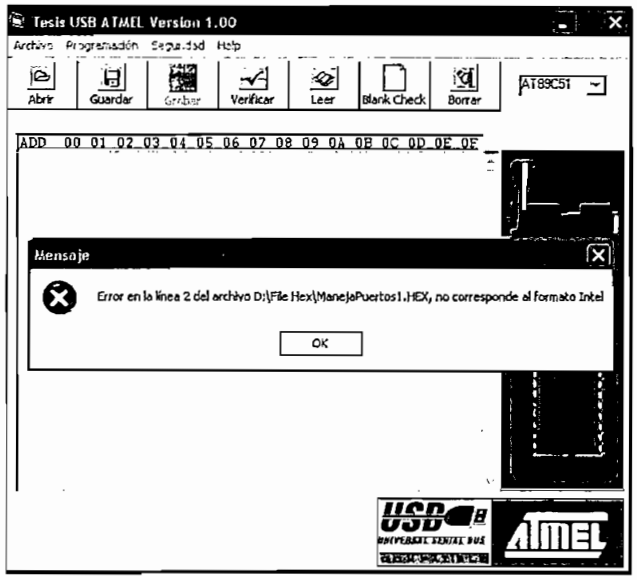



Figura 5.17: Siguiente ventana con mensaje de error al abrir archivo sin estructura de formato Intel

La solución es verificar que el archivo.hex tenga el formato intel para lo cual se recomienda que se compile el programa en un ensamblador.

# CAPÍTULO VI

# CAPÍTULO VI

# CONCLUSIONES Y RECOMENDACIONES

## 6.1 CONCLUSIONES

- Las especificaciones técnicas acerca del interfaz USB están dirigidas principalmente al desarrollo de nuevos periféricos, además de proveer información para plataformas Windows como también para otros sistemas operativos. Estas especificaciones pueden ser usadas para el desarrollo de nuevos productos y software asociado.
- La limitación de distancia que tiene el interfaz USB hace que todavía no sea un interfaz cien por ciento universal.
- El interfaz USB se va implantando como bus de comunicación dominante, en el mercado de periféricos para PCs, facilitando a los diseñadores a no pensar en otro tipo de interfaz.
- En el diseño del programador se puede utilizar un zócalo en lugar de los cuatro zócalos, esto se puede lograr adicionando más elementos.
- El programador esta entre los dispositivos de la clase HID (Human Interface Device). Esta característica hace que los drivers para el funcionamiento del mismo estén incluidos en el sistema operativo.
- La agrupación de los distintos dispositivos en clases, hace que cada una de ellas cumpla una función y un conjunto de requerimientos determinados. Permitiendo utilizar un driver genérico para controlar los dispositivos de cada clase; que puede ser incluido dentro del sistema operativo. Los desarrolladores de nuevos dispositivos no necesariamente están obligados a utilizar los drivers del sistema operativo si no que puede crear sus propios drivers acorde a la necesidad de su aplicación.
- Una de las ventajas del interfaz USB es que no requiere más de una IRQs (interrupt request) para reconocer a un gran número de periféricos.

- El funcionamiento ideal de un producto USB siguiendo la filosofía de conexión dinámica de periféricos se sustenta en su reconocimiento automático por parte del sistema operativo o, en su defecto, en la solicitud del driver correspondiente. De este modo, al establecer la conexión y cargar el controlador adecuado el periférico debería comenzar a funcionar sin problemas. Por desgracia, no siempre se observa este comportamiento y en muchas ocasiones es necesario reiniciar la PC para que reconozca el dispositivo.
- El bus USB permite modificar el aspecto externo del sistema de conexiones de periféricos al mismo equipo. Permite que cualquier ordenador, por pequeño que sea, acepte tantos periféricos como sea preciso (hasta 128 periféricos, incluido el sistema anfitrión).
- El sistema de conexiones USB es de bajo costo y su funcionamiento es muy simple, el usuario no necesita instalar ni configurar tarjetas de expansión y le permite la conexión y desconexión de dispositivos con el PC en funcionamiento. También permite controlar de forma remota la alimentación de los periféricos alimentados íntegramente a través del bus.
- El utilizar latch para expandir los puertos, facilita la programación en el PIC16C765, en el caso de no disponer de estos; seria necesario manejar 13 líneas adicionales a las de control y datos, cuyo requerimiento no podría ser cubierto con los pines sobrantes del PIC16C765.
- Sin la ayuda de Visual Basic y la librería HIDcomm para la comunicación entre un dispositivo que utilice el PIC16C765 y la PC, la construcción del HID es mas complicado, ya que en este caso se hace necesario emplear las funciones APIs que se encuentran en las librerías y cabeceras proporcionadas por Windows DDK. La herramienta apropiada para manejar este tipo de funciones es Visual C++.
- Con la construcción del programador quedan las bases para realizar en un futuro nuevos dispositivos que incorporen el interfaz USB.
- El programador "USB Atmel flash Programador" al ser un dispositivo de clase HID es reconocido en los sistemas operativo Windows 98 y Milenium. Pero el software no funcionara, ya que esta realizado para los sistemas operativos Windows 2000 y XP.

## 6.2 RECOMENDACIONES

- A los desarrolladores intrépidos que deseen empaparse más acerca del interfaz USB se recomienda revisar la especificación USB1.1 que da un detalle más amplio acerca de este interfaz.
- En el desarrollo de nuevas aplicaciones con el PIC16765 hay que tomar siempre en cuenta que el puerto RA por defecto viene habilitado como un puerto analógico (conversor análogo/digital), razón por la cual hay que deshabilitar al inicio del programa. Para que el puerto RA trabaje como entradas y salidas digitales hay que habilitar los tres primeros bits del registro de control ADCON1 en uno lógico.
- En el mercado hay varios fabricantes de microcontroladores con USB, facilitando al desarrollador la posibilidad de elegir con cual de ellos desea trabajar. Los análisis que se recomiendan para la selección del microcontrolador mas apropiado es en base al número de EndPoints que desea manejar, los costos, la distancia a la cual va ha trabajar el dispositivo y lo mas importante es que tenga una herramienta de desarrollo compacta.
- Los futuros desarrolladores de dispositivos USB que no quieran utilizar fuente externa, podrían aprovechar la fuente regulada que proporciona el interfaz USB, siempre y cuando consideren la máxima intensidad que pueden extraer del mismo.
- Una desventaja de tener dispositivos USB es que no funcionan en MS-DOS, ni en versiones antiguas de Windows, Linux con núcleos viejos. Se debe tener en cuenta en periféricos esencialmente como en teclados, ratones, monitores, impresoras, etc.
- En el caso que alguien desee mejorar el presente software y el firmware del programador de microcontroladores ATMEL debe tomar en cuenta las formas de onda con tiempos más de los mencionados en las especificaciones técnicas de cada microcontrolador a programar.
- Hay que tener en cuenta que no se deben realizar instalaciones de windows DDK sobre previas versiones u otro DDK. La instalación

inapropiada puede causar que archivos viejos e incorrectos causen problemas en los drivers que se desarrollen.

- El usuario debe tener mucho cuidado en conectar los microcontroladores a programar, pues si no toma en cuenta los pasos recomendados en el manual de usuario (ver Anexo5), puede causar serios daños al microcontrolador.
- Para actualizar el programador "USB Atmel flash Programador" de dispositivo de baja velocidad (USB 1.1) a un dispositivo de alta velocidad (USB 2.0) es necesario cambiar el chip con puerto USB que soporte alta velocidad. Por lo cual se debe realizar un nuevo software y firmware.
- El costo del programador en relación al costo de los programadores con interfaz USB comerciales, en cierto modo puede resultar mucho más económico, como ejemplo se tiene el programador comercial Leaper 5E que tiene un costo en el mercado de 229 dolares; en comparación al programador "USB Atmel flash Programador" en el cual se hizo una inversión de 48 dólares y los elementos para la construcción se los puede encontrar fácilmente en el mercado. A continuación se presenta el detalle de cada uno de los elementos utilizados con sus respectivos costos, no se incluyen costos del software ni de la mano de obra.

| Item | Elementos | Unidades | Valor Unitario | Costo |
|---|---|---|---|---|
| 1 | Resistencias | 9 | 0,05 | 0,45 |
| 2 | Puente de diodos | 1 | 0,15 | 0,15 |
| 3 | Led | 2 | 0,15 | 0,3 |
| 4 | Diodo | 1 | 0,1 | 0,1 |
| 5 | Capacitores | 10 | 0,12 | 1,2 |
| 6 | Regulador LM7805 | 1 | 0,6 | 0,6 |
| 7 | Regulador LM7812 | 1 | 1,2 | 1,2 |
| 8 | Regulador LM317 | 1 | 0,6 | 0,6 |
| 9 | Jack | 2 | 0,1 | 0,2 |
| 10 | Cristales | 2 | 0,1 | 0,2 |
| 11 | Lactch 74ls374 | 2 | 0,69 | 1,38 |
| 12 | Buffer 74ls07 | 1 | 0,7 | 0,7 |
| 13 | PIC16C765 | 1 | 18 | 18 |
| 14 | Circuito Impreso | 1 | 14 | 14 |
| 15 | Potenciómetro | 2 | 0,3 | 0,6 |
| 16 | Zócalo | 8 | 0,25 | 2 |
| 17 | Cable USB | 1 | 1 | 1 |
| 18 | Carcaza | 1 | 5 | 5 |
| | | | **Total** | **47,68** |

# BIBLIOGRAFÍA

# BIBLIOGRAFÍA


- CUENCA CISNEROS MARCELO IVAN, Programador de memorias y microcontroladores INTEL, Tesis, 1995.
- MICROCHIP INC, PIC16C765 Data Sheet, ©Microchip Technology Inc, 2000.
- MICROCHIP INC, USB FIRMWARE USER'S GUIDE, ©Microchip Technology Inc, 2002
- MICROSOFT, Windows DDK Documentation, ©Microsoft Corporation, 2000
- MICROCHIP INC, ACTIVEX HIDComm Help Documentation, ©Microchip Technology Inc, 2001. Tutorial.
- COMPAQ, INTEL, MICROSOFT, NEC, Universal Serial Bus Specification, Revision 1.1.
- John Hyde, "USB Design by Example", John Wiley & Sons, Inc. 1999.
- USB Implementers' Forum, "Device Class Definition for Human Interface Devices (HID) – Firmware Specification", Version 1.1, 4/7/99.
- AXELSON JAN, USB Complete, Lakeiew Research, 2001
- ATMEL INC, ATMEL Data Sheet.
- http://www.atmel.com
- http://www.microsoft.com
- http://www.microchip.com
- http://www.usb.org
- http://www.lvr.com
- http://www.clubedohardware.com.br/usb1.html
- http://www.refly.com/
- http://www.conozcasuhardware.com/
- http://www.kingston.com/
- http://www.monografias.com
- http://www.apple.com/
- http://www.intel com/
- http://users.ece.gatech.edu/~shalan/mmc_usb_web.htm

- http://pages.sbcglobal.net/without_land/(buenisima)
- http://sipan.inictel.gob.pe/internet/alex/publi28.html
- http://www.casadomo.com/revista_domotica_redes.asp?TextType=1302
- http://www.usbdeveloper.com/UnderstandUSB/understandusb.htm
- http://www.kmitl.ac.th/~kswichit/Pgm89v3_WEB/Pgm89v3.html
- http://www.ftdichip.com/FTModule.htm1
- http://www.futurlec.com/DS89C420Controller.shtml
- http://www.futurlec.com/USB.shtml
- http://sipan.inictel.gob.pe/internet/alex/publi28.html
- http://sipan.inictel.gob.pe/internet/alex/publi21.html

# ANEXO 1

## ESPECIFICACIONES TÉCNICAS DEL MICROCONTROLADOR PIC16C765

MICROCHIP

# PIC16C745/765

## 8-Bit CMOS Microcontrollers with USB

**Devices included in this data sheet:**

- PIC16C745
- PIC16C765

**Microcontroller Core Features:**

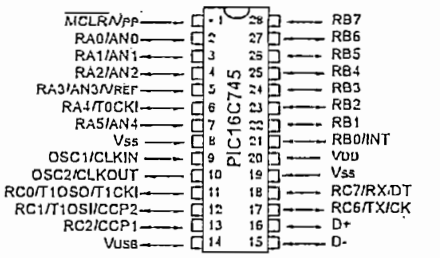
- High-performance RISC CPU
- Only 35 single word instructions

| Device | Memory | | Pins | A/D Resolution | A/D Channels |
|---|---|---|---|---|---|
| | Program x14 | Data x8 | | | |
| PIC16C745 | 8K | 256 | 28 | 8 | 5 |
| PIC16C765 | 8K | 256 | 40 | 8 | 8 |

- All single cycle instructions except for program branches which are two cycle
- Interrupt capability (up to 12 internal/external interrupt sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Brown-out detection circuitry for Brown-out Reset (BOR)
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
  - EC - External clock (24 MHz)
  - E4 - External clock with PLL (6 MHz)
  - HS - Crystal/Resonator (24 MHz)
  - H4 - Crystal/Resonator with PLL (6 MHz)
- Processor clock of 24 MHz derived from 6 MHz crystal or resonator
- Fully static low-power, high-speed CMOS
- In-Circuit Serial Programming™ (ICSP)
- Operating voltage range
  - 4.35 to 5.25V
- High Sink/Source Current 25/25 mA
- Wide temperature range
  - Industrial (-40°C - 85°C)
- Low-power consumption:
  - ~ 16 mA @ 5V, 24 MHz
  - 100 µA typical standby current

**Pin Diagrams**

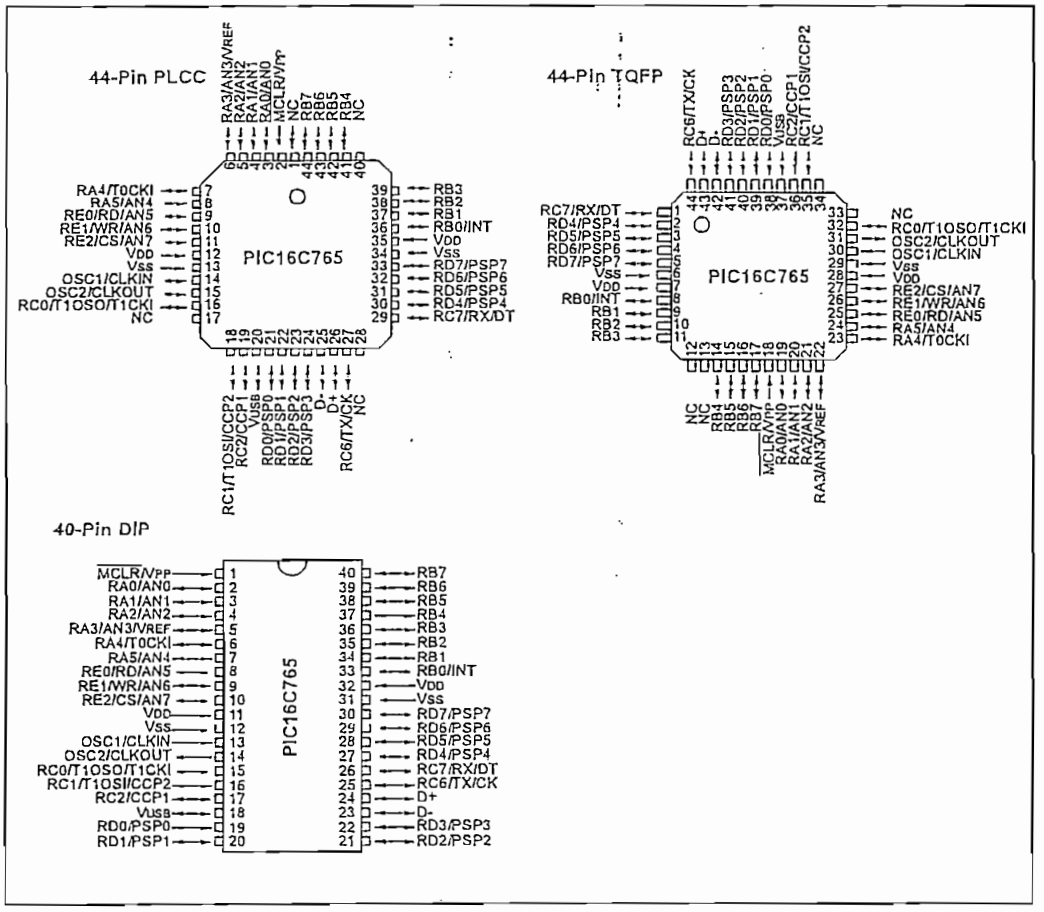
28-Pin DIP, SOIC




**Peripheral Features:**

- Universal Serial Bus (USB 1.1)
  - Soft attach/detach
- 64 bytes of USB dual port RAM
- 22 (PIC16C745) or 33 (PIC16C765) I/O pins
  - Individual direction control
  - 1 high voltage open drain (RA4)
  - 8 PORTB pins with:
    - Interrupt-on-change control (RB<7:4> only)
    - Weak pull-up control
  - 3 pins dedicated to USB
- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- 2 Capture, Compare and PWM modules
  - Capture is 16-bit, max. resolution is 10.4 ns
  - Compare is 16-bit, max. resolution is 167 ns
  - PWM maximum resolution is 10-bit
- 8-bit multi-channel Analog-to-Digital converter
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI)
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (PIC16C765 only)

# PIC16C745/765




| Key Features<br>PICmicro™ Mid-Range Reference Manual<br>(DS33023) | PIC16C745 | PIC16C765 |
|---|---|---|
| Operating Frequency | 6 MHz or 24 MHz | 6 MHz or 24 MHz |
| Resets (and Delays) | POR, BOR (PWRT, OST) | POR, BOR (PWRT, OST) |
| Program Memory (14-bit words) | 8K | 8K |
| Data Memory (bytes) | 256 | 256 |
| Dual Port Ram | 64 | 64 |
| Interrupt Sources | 11 | 12 |
| I/O Ports | 22 (Ports A, B, C) | 33 (Ports A, B, C, D, E) |
| Timers | 3 | 3 |
| Capture/Compare/PWM modules | 2 | 2 |
| Analog-to-Digital Converter Module | 5 channel x 8 bit | 8 channel x 8 bit |
| Parallel Slave Port | — | Yes |
| Serial Communication | USB, USART/SCI | USB, USART/SCI |
| Brown-out Detect Reset | Yes | Yes |

## 3.0 ARCHITECTURAL OVERVIEW

The high performance of the PIC16C745/765 family can be attributed to a number of architectural features commonly found in RISC microprocessors. To begin with, the PIC16C745/765 uses a Harvard architecture, in which program and data are accessed from separate memories using separate buses. This improves bandwidth over traditional von Neumann architecture in which program and data are fetched from the same memory using the same bus. Separating program and data buses further allows instructions to be sized differently than the 8-bit wide data word. Instruction opcodes are 14-bits wide making it possible to have all single word instructions. A 14-bit wide program memory access bus fetches a 14-bit instruction in a single cycle. A two-stage pipeline overlaps fetch and execution of instructions (Example 3-1). Consequently, most instructions execute in a single cycle (166.6667 ns @ 24 MHz) except for program branches.

| Device | Memory | | Pins | A/D Resolution | A/D Channels |
|---|---|---|---|---|---|
| | Program x14 | Data x8 | | | |
| PIC16C745 | 8K | 256 | 28 | 8 | 5 |
| PIC16C765 | 8K | 256 | 40 | 8 | 8 |

The PIC16C745/765 can directly or indirectly address its register files or data memory. All special function registers, including the program counter, are mapped in the data memory. The PIC16C745/765 has an orthogonal (symmetrical) instruction set that makes it possible to carry out any operation on any register using any addressing mode. This symmetrical nature and lack of 'special optimal situations' make programming with the PIC16C745/765 simple yet efficient. In addition, the learning curve is reduced significantly.

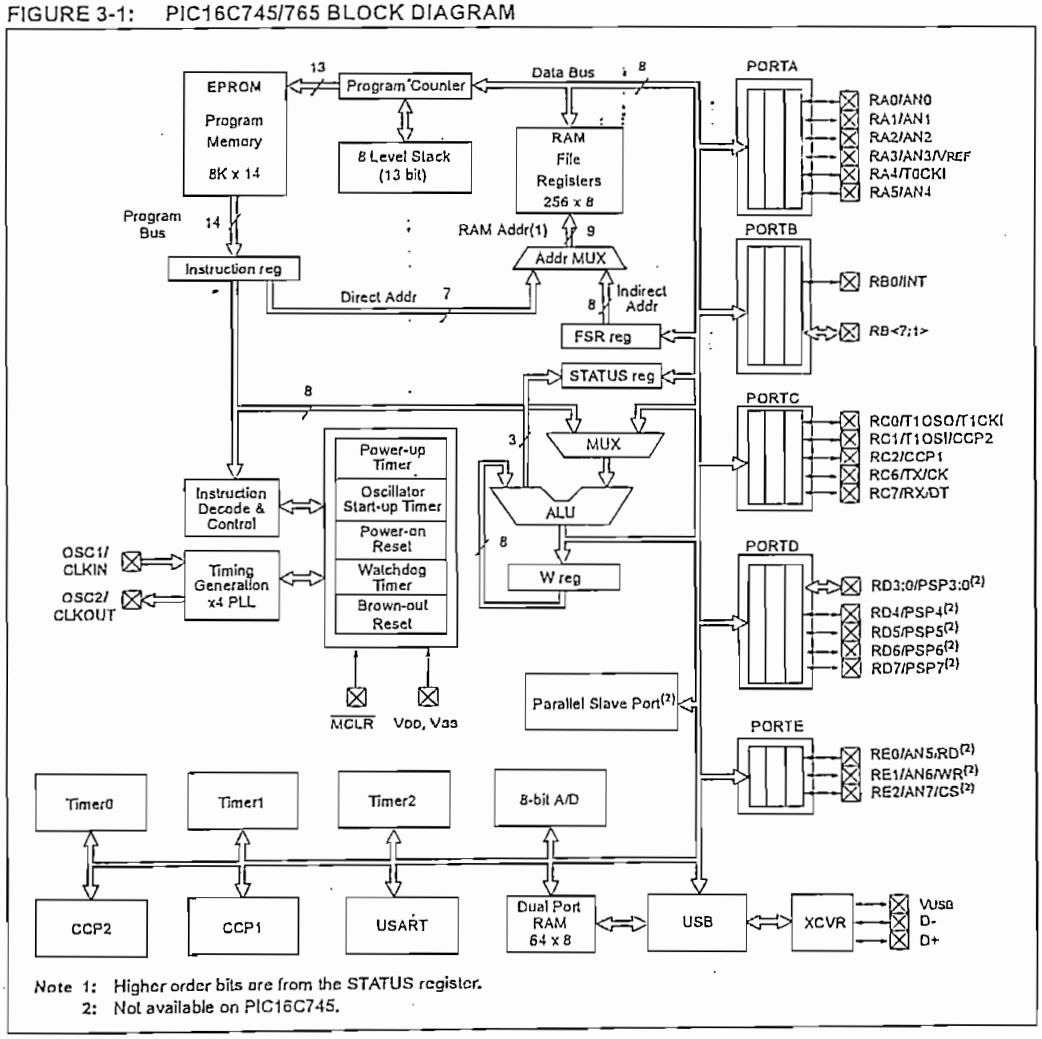
PIC16C745/765 devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between the data in the working register and any register file.

The ALU is 8-bits wide and capable of addition, subtraction, shift and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the working register (W register). The other operand is a file register or an immediate constant. In single operand instructions, the operand is either the W register or a file register.

The W register is an 8-bit working register used for ALU operations. It is not an addressable register.

Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a borrow bit and a digit borrow out bit, respectively, in subtraction. See the SUBLW and SUBWF instructions for examples.

FIGURE 3-1: PIC16C745/765 BLOCK DIAGRAM




Note 1: Higher order bits are from the STATUS register.
2: Not available on PIC16C745.

TABLE 3-1: PIC16C745/765 PINOUT DESCRIPTION

| Name | Function | Input Type | Output Type | Description |
|---|---|---|---|---|
| MCLR/VPP | MCLR | ST | — | Master Clear |
| | VPP | Power | — | Programming Voltage |
| OSC1/CLKIN | OSC1 | Xtal | — | Crystal/Resonator |
| | CLKIN | ST | — | External Clock Input |
| OSC2/CLKOUT | OSC2 | — | Xtal | Crystal/Resonator |
| | CLKOUT | — | CMOS | Internal Clock (FINT/4) Output |
| | | | | |
| RA0/AN0 | RA0 | ST | CMOS | Bi-directional I/O |
| | AN0 | AN | — | A/D Input |
| RA1/AN1 | RA1 | ST | CMOS | Bi-directional I/O |
| | AN1 | AN | — | A/D Input |
| RA2/AN2 | RA2 | ST | CMOS | Bi-directional I/O |
| | AN2 | AN | — | A/D Input |
| RA3/AN3/VREF | RA3 | ST | CMOS | Bi-directional I/O |
| | AN3 | AN | — | A/D Input |
| | VREF | AN | — | A/D Positive Reference |
| RA4/T0CKI | RA4 | ST | OD | Bi-directional I/O |
| | T0CKI | ST | — | Timer 0 Clock Input |
| RA5/AN4 | RA5 | ST | | Bi-directional I/O |
| | AN4 | AN | — | A/D Input |
| | | | | |
| RB0/INT | RB0 | TTL | CMOS | Bi-directional I/O(1) |
| | INT | ST | — | Interrupt |
| RB1 | RB1 | TTL | CMOS | Bi-directional I/O(1) |
| RB2 | RB2 | TTL | CMOS | Bi-directional I/O(1) |
| RB3 | RB3 | TTL | CMOS | Bi-directional I/O(1) |
| RB4 | RB4 | TTL | CMOS | Bi-directional I/O with Interrupt-on-Change(1) |
| RB5 | RB5 | TTL | CMOS | Bi-directional I/O with Interrupt-on-Change(1) |
| RB6/ICSPC | RB6 | TTL | CMOS | Bi-directional I/O with Interrupt-on-Change(1) |
| | ICSPC | ST | | In-Circuit Serial Programming Clock Input |
| RB7/ICSPD | RB7 | TTL | CMOS | Bi-directional I/O with Interrupt-on-Change(1) |
| | ICSPD | ST | CMOS | In-Circuit Serial Programming Data I/O |
| | | | | |
| RC0/T1OSO/T1CKI | RC0 | ST | CMOS | Bi-directional I/O |
| | T1OSO | — | Xtal | T1 Oscillator Output |
| | T1CKI | ST | — | T1 Clock Input |
| RC1/T1OSI/CCP2(1) | RC1 | ST | CMOS | Bi-directional I/O |
| | T1OSI | Xtal | — | T1 Oscillator Input |
| | CCP2 | | | Capture In/Compare Out/PWM Out 2 |
| RC2/CCP1 | RC2 | ST | CMOS | Bi-directional I/O |
| | CCP1 | | | Capture In/Compare Out/PWM Out 1 |
| VUSB | VUSB | | Power | Regulator Output Voltage |
| D- | D- | USB | USB | USB Differential Bus |
| D+ | D+ | USB | USB | USB Differential Bus |

Legend: OD = open drain, ST = Schmitt Trigger

Note 1: Weak pull-ups. PORT B pull-ups are byte wide programmable.

2: PIC16C765 only.

TABLE 3-1: PIC16C745/765 PINOUT DESCRIPTION (CONTINUED)

| Name | Function | Input Type | Output Type | Description |
|---|---|---|---|---|
| RC6/TX/CK | RC6 | ST | CMOS | Bi-directional I/O |
| | TX | — | CMOS | USART Async Transmit |
| | CK | ST | CMOS | USART Master Out/Slave In Clock |
| RC7/RX/DT | RC7 | ST | CMOS | Bi-directional I/O |
| | RX | ST | — | USART Async Receive |
| | DT | ST | CMOS | USART Data I/O |
| | | | | |
| RD0/PSP0 | RD0 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP0 | TTL | — | Parallel Slave Port Data Input(2) |
| RD1/PSP1 | RD1 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP1 | TTL | — | Parallel Slave Port Data Input(2) |
| RD2/PSP2 | RD2 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP2 | TTL | — | Parallel Slave Port Data Input(2) |
| RD3/PSP3 | RD3 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP3 | TTL | — | Parallel Slave Port Data Input(2) |
| RD4/PSP4 | RD4 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP4 | TTL | — | Parallel Slave Port Data Input(2) |
| RD5/PSP5 | RD5 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP5 | TTL | — | Parallel Slave Port Data Input(2) |
| RD6/PSP6 | RD6 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP6 | TTL | — | Parallel Slave Port Data Input(2) |
| RD7/PSP7 | RD7 | TTL | CMOS | Bi-directional I/O(2) |
| | PSP7 | TTL | — | Parallel Slave Port Data Input(2) |
| | | | | |
| RE0/$\overline{RD}$/AN5 | RE0 | ST | CMOS | Bi-directional I/O(2) |
| | $\overline{RD}$ | TTL | — | Parallel Slave Port Control Input(2) |
| | AN5 | AN | — | A/D Input(2) |
| RE1/$\overline{WR}$/AN6 | RE1 | ST | CMOS | Bi-directional I/O(2) |
| | $\overline{WR}$ | TTL | — | Parallel Slave Port Control Input(2) |
| | AN6 | AN | — | A/D Input(2) |
| RE2/$\overline{CS}$/AN7 | RE2 | ST | CMOS | Bi-directional I/O(2) |
| | $\overline{CS}$ | TTL | — | Parallel Slave Port Data Input(2) |
| | AN7 | AN | — | A/D Input(2) |
| | | | | |
| VDD | VDD | Power | — | Power |
| Vss | Vss | Power | — | Ground |

Legend: OD = open drain, ST = Schmitt Trigger

Note 1: Weak pull-ups. PORT B pull-ups are byte wide programmable.

2: PIC16C765 only.

FIGURE 4-2: DATA MEMORY MAP FOR PIC16C745/765

| Bank 0 | File Address | Bank 1 | File Address | Bank 2 | File Address | Bank 3 | File Address |
|---|---|---|---|---|---|---|---|
| Indirect addr.(*) | 00h | Indirect addr.(*) | 80h | Indirect addr.(*) | 100h | Indirect addr.(*) | 180h |
| TMR0 | 01h | OPTION_REG | 81h | TMR0 | 101h | OPTION_REG | 181h |
| PCL | 02h | PCL | 82h | PCL | 102h | PCL | 182h |
| STATUS | 03h | STATUS | 83h | STATUS | 103h | STATUS | 183h |
| FSR | 04h | FSR | 84h | FSR | 104h | FSR | 184h |
| PORTA | 05h | TRISA | 85h | | 105h | | 185h |
| PORTB | 06h | TRISB | 86h | PORTB | 106h | TRISB | 186h |
| PORTC | 07h | TRISC | 87h | | 107h | | 187h |
| PORTD(2) | 08h | TRISD(2) | 88h | | 108h | | 188h |
| PORTE(2) | 09h | TRISE(2) | 89h | | 109h | | 189h |
| PCLATH | 0Ah | PCLATH | 8Ah | PCLATH | 10Ah | PCLATH | 18Ah |
| INTCON | 0Bh | INTCON | 8Bh | INTCON | 10Bh | INTCON | 18Bh |
| PIR1 | 0Ch | PIE1 | 8Ch | | 10Ch | | 18Ch |
| PIR2 | 0Dh | PIE2 | 8Dh | | 10Dh | | 18Dh |
| TMR1L | 0Eh | PCON | 8Eh | | 10Eh | | 18Eh |
| TMR1H | 0Fh | | 8Fh | | 10Fh | | 18Fh |
| T1CON | 10h | | 90h | | 110h | UIR | 190h |
| TMR2 | 11h | | 91h | | 111h | UIE | 191h |
| T2CON | 12h | PR2 | 92h | | 112h | UEIR | 192h |
| | 13h | | 93h | | 113h | UEIE | 193h |
| | 14h | | 94h | | 114h | USTAT | 194h |
| CCPR1L | 15h | | 95h | | 115h | UCTRL | 195h |
| CCPR1H | 16h | | 96h | | 116h | UADDR | 196h |
| CCP1CON | 17h | | 97h | | 117h | USWSTAT(1) | 197h |
| RCSTA | 18h | TXSTA | 98h | | 118h | UEP0 | 198h |
| TXREG | 19h | SPBRG | 99h | | 119h | UEP1 | 199h |
| RCREG | 1Ah | | 9Ah | | 11Ah | UEP2 | 19Ah |
| CCPR2L | 1Bh | | 9Bh | | 11Bh | | 19Bh(1) |
| CCPR2H | 1Ch | | 9Ch | | 11Ch | | 19Ch(1) |
| CCP2CON | 1Dh | | 9Dh | | 11Dh | | 19Dh(1) |
| ADRES | 1Eh | | 9Eh | | 11Eh | | 19Eh(1) |
| ADCON0 | 1Fh | ADCON1 | 9Fh | | 11Fh | | 19Fh(1) |
| General Purpose Register 96 Bytes | 20h | General Purpose Register 80 Bytes | A0h | General Purpose Register 80 Bytes | 120h | USB Dual Port Memory 64 Bytes | 1A0h |
| | | | | | | | 1DFh |
| | | | EFh | | 16Fh | | 1E0h 1EFh |
| | 7Fh | accesses 70h-7Fh | F0h FFh | accesses 70h-7Fh | 170h 17Fh | accesses 70h-7Fh | 1F0h 1FFh |

Unimplemented data memory locations, read as '0'.
*Not a physical register.
Note 1: Reserved registers may contain USB state information.
2: Parallel slave ports (PORTD and PORTE) not implemented on PIC16C745; always maintain these bits clear.

### 4.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and Peripheral Modules for controlling the desired operation of the device. These registers are implemented as static RAM.

The Special Function Registers can be classified into two sets (core and peripheral). Those registers associated with the "core" functions are described in this section, and those related to the operation of the peripheral features are described in the section of that peripheral feature.

TABLE 4-1: SPECIAL FUNCTION REGISTER SUMMARY

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other resets (2) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 0 | | | | | | | | | | | |
| 00h | INDF(3) | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 0000 0000 |
| 01h | TMR0 | Timer0 module's register | | | | | | | | xxxx xxxx | uuuu uuuu |
| 02h | PCL(3) | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 0000 0000 |
| 03h | STATUS(3) | IRP(2) | RP1(2) | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 000q quuu |
| 04h | FSR(3) | Indirect data memory address pointer | | | | | | | | xxxx xxxx | uuuu uuuu |
| 05h | PORTA | — | — | PORTA Data Latch when written: PORTA pins when read | | | | | | --0x 0000 | --0u 0000 |
| 06h | PORTB | PORTB Data Latch when written: PORTB pins when read | | | | | | | | xxxx xxxx | uuuu uuuu |
| 07h | PORTC | RC7 | RC6 | — | — | — | RC2 | RC1 | RC0 | xx-- -xxx | uu-- -uuu |
| 08h | PORTD(4) | PORTD Data Latch when written: PORTD pins when read | | | | | | | | xxxx xxxx | uuuu uuuu |
| 09h | PORTE(4) | — | — | — | — | — | RE2 | RE1 | RE0 | ---- -xxx | ---- -uuu |
| 0Ah | PCLATH(1,3) | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | ---0 0000 |
| 0Bh | INTCON(3) | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 0000 000u |
| 0Ch | PIR1 | PSPIF(4) | ADIF | RCIF | TXIF | USBIF | CCP1IF | TMR2IF | TMR1IF | 0000 0000 | 0000 0000 |
| 0Dh | PIR2 | — | — | — | — | — | — | — | CCP2IF | ---- ---0 | ---- ---0 |
| 0Eh | TMR1L | Holding register for the Least Significant Byte of the 16-bit TMR1 register | | | | | | | | xxxx xxxx | uuuu uuuu |
| 0Fh | TMR1H | Holding register for the Most Significant Byte of the 16-bit TMR1 register | | | | | | | | xxxx xxxx | uuuu uuuu |
| 10h | T1CON | — | — | T1CKPS1 | T1CKPS0 | T1OSCEN | $\overline{T1SYNC}$ | TMR1CS | TMR1ON | --00 0000 | --uu uuuu |
| 11h | TMR2 | Timer2 module's register | | | | | | | | 0000 0000 | 0000 0000 |
| 12h | T2CON | — | T0UTPS3 | T0UTPS2 | T0UTPS1 | T0UTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 | -000 0000 | -000 0000 |
| 13h | — | Unimplemented | | | | | | | | — | — |
| 14h | — | Unimplemented | | | | | | | | — | — |
| 15h | CCPR1L | Capture/Compare/PWM Register1 (LSB) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 16h | CCPR1H | Capture/Compare/PWM Register1 (MSB) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 17h | CCP1CON | — | — | DC1B1 | DC1B0 | CCP1M3 | CCP1M2 | CCP1M1 | CCP1M0 | --00 0000 | --00 0000 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | — | FERR | OERR | RX9D | 0000 -00x | 0000 -00x |
| 19h | TXREG | USART Transmit Data Register | | | | | | | | 0000 0000 | 0000 0000 |
| 1Ah | RCREG | USART Receive Data Register | | | | | | | | 0000 0000 | 0000 0000 |
| 1Bh | CCPR2L | Capture/Compare/PWM Register2 (LSB) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 1Ch | CCPR2H | Capture/Compare/PWM Register2 (MSB) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 1Dh | CCP2CON | — | — | DC2B1 | DC2B1 | CCP2M3 | CCP2M2 | CCP2M1 | CCP2M0 | --00 0000 | --00 0000 |
| 1Eh | ADRES | A/D Result Register | | | | | | | | xxxx xxxx | uuuu uuuu |
| 1Fh | ADCON0 | ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{DONE}$ | — | ADON | 0000 00-0 | 0000 00-0 |

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0'.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) RESETS include external RESET through $\overline{MCLR}$ and Watchdog Timer Reset.

3: These registers can be addressed from any bank.

4: The Parallel Slave Port (PORTD and PORTE) is not implemented on the PIC16C745, always maintain these bits clear.

TABLE 4-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other resets (2) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 1 | | | | | | | | | | | |
| 80h | INDF(3) | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 0000 0000 |
| 81h | OPTION | $\overline{RBPU}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 1111 1111 |
| 82h | PCL(3) | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 0000 0000 |
| 83h | STATUS(3) | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 000q quuu |
| 84h | FSR(3) | Indirect data memory address pointer | | | | | | | | xxxx xxxx | uuuu uuuu |
| 85h | TRISA | — | — | PORTA Data Direction Register | | | | | | --11 1111 | --11 1111 |
| 86h | TRISB | PORTB Data Direction Register | | | | | | | | 1111 1111 | 1111 1111 |
| 87h | TRISC | TRISC7 | TRISC6 | — | — | — | TRISC2 | TRISC1 | TRISC0 | 11-- -111 | 11-- -111 |
| 88h | TRISD(4) | PORTD Data Direction Register | | | | | | | | 1111 1111 | 1111 1111 |
| 89h | TRISE(4) | IBF | OBF | IBOV | PSPMODE | — | PORTE Data Direction Bits | | | 0000 -111 | 0000 -111 |
| 8Ah | PCLATH(1,3) | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | ---0 0000 |
| 8Bh | INTCON(3) | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 0000 000u |
| 8Ch | PIE1 | PSPIE(4) | ADIE | RCIE | TXIE | USBIE | CCP1IE | TMR2IE | TMR1IE | 0000 0000 | 0000 0000 |
| 8Dh | PIE2 | — | — | — | — | — | — | — | CCP2IE | ---- ---0 | ---- ---0 |
| 8Eh | PCON | — | — | — | — | — | — | $\overline{POR}$ | $\overline{BOR}$ | ---- --qq | ---- --uu |
| 8Fh | — | Unimplemented | | | | | | | | — | — |
| 90h | — | Unimplemented | | | | | | | | — | — |
| 91h | | Unimplemented | | | | | | | | | |
| 92h | PR2 | Timer2 Period Register | | | | | | | | 1111 1111 | 1111 1111 |
| 93h | — | Unimplemented | | | | | | | | — | — |
| 94h | — | Unimplemented | | | | | | | | — | — |
| 95h | — | Unimplemented | | | | | | | | — | — |
| 96h | — | Unimplemented | | | | | | | | — | — |
| 97h | — | Unimplemented | | | | | | | | — | — |
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 0000 -010 |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 0000 0000 |
| 9Ah | — | Unimplemented | | | | | | | | — | — |
| 9Bh | — | Unimplemented | | | | | | | | — | — |
| 9Ch | — | Unimplemented | | | | | | | | — | — |
| 9Dh | — | Unimplemented | | | | | | | | — | — |
| 9Eh | — | Unimplemented | | | | | | | | — | — |
| 9Fh | ADCON1 | — | — | — | — | — | PCFG2 | PCFG1 | PCFG0 | ---- -000 | ---- -000 |

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0'.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) RESETS include external RESET through $\overline{MCLR}$ and Watchdog Timer Reset.

3: These registers can be addressed from any bank.

4: The Parallel Slave Port (PORTD and PORTE) is not implemented on the PIC16C745, always maintain these bits clear.

TABLE 4-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other resets (2) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 2 | | | | | | | | | | | |
| 100h | INDF(3) | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 0000 0000 |
| 101h | TMR0 | Timer0 module's register | | | | | | | | xxxx xxxx | uuuu uuuu |
| 102h | PCL(3) | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 0000 0000 |
| 103h | STATUS(3) | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 000q quuu |
| 104h | FSR(3) | Indirect data memory address pointer | | | | | | | | xxxx xxxx | uuuu uuuu |
| 105h | — | Unimplemented | | | | | | | | — | — |
| 106h | PORTB | PORTB Data Latch when written: PORTB pins when read | | | | | | | | xxxx xxxx | uuuu uuuu |
| 107h | — | Unimplemented | | | | | | | | — | — |
| 108h | — | Unimplemented | | | | | | | | — | — |
| 109h | — | Unimplemented | | | | | | | | — | — |
| 10Ah | PCLATH(1,3) | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | ---0 0000 |
| 10Bh | INTCON(3) | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 0000 000u |
| 10Ch-11Fh | — | Unimplemented | | | | | | | | — | — |

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0'.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) RESETS include external RESET through $\overline{MCLR}$ and Watchdog Timer Reset.

3: These registers can be addressed from any bank.

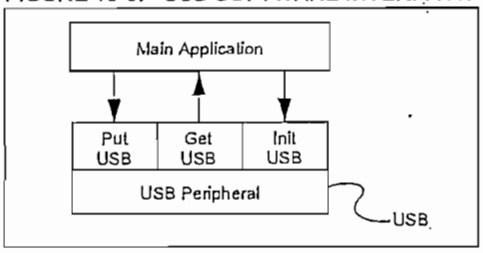
4: The Parallel Slave Port (PORTD and PORTE) is not implemented on the PIC16C745, always maintain these bits clear.

## 10.9 USB Firmware Users Guide

### 10.9.1 INTRODUCING THE USB SOFTWARE INTERFACE

Microchip provides a layer of software that handles the lowest level interface so your application won't have to. This provides a simple Put/Get interface for communication. Most of the USB processing takes place in the background through the Interrupt Service Routine. From the application viewpoint, the enumeration process and data communication takes place without further interaction. However, substantial setup is required in the form of generating appropriate descriptors.

FIGURE 10-3: USB SOFTWARE INTERFACE




### 10.9.2 INTEGRATING USB INTO YOUR APPLICATION

The latest version of the USB interface software is available on Microchip's website (see http://www.microchip.com/).

The interface to the application is packaged in 3 functions: **InitUSB**, **PutUSB** and **GetUSB**. **InitUSB** initializes the USB peripheral, allowing the host to enumerate the device. Then, for normal data communications, function **PutUSB** sends data to the host and **GetUSB** receives data from the host.

However, there's a fair amount of setup work that must be completed. USB depends heavily on the descriptors. These are the software parameters that are communicated to the host to let it know what the device is, and how to communicate with it. See USB V1.1 spec section 9.5 for more details.

Also, code must be added to give meaning to the **SetConfiguration** command. The Chapter 9 commands call **SetConfiguration** when it receives the command. Both the descriptors and **SetConfiguration** are in DESCRIPT.ASM.

**InitUSB** enables the USB interrupt so enumeration can begin. The actual enumeration process occurs in the background, driven by the host and the Interrupt Service Routine. Macro **ConfiguredUSB** waits until the device is in the CONFIGURED state. The time required to enumerate is completely dependent on the host and bus loading.

### 10.9.3 INTERRUPT STRUCTURE CONCERNS

#### 10.9.3.1 Processor Resources

Most of the USB processing occurs via the interrupt and thus is invisible to application. However, it still consumes processor resources. These include ROM, RAM, Common RAM and Stack Levels. This section attempts to quantify the impact on each of these resources, and shows ways to avoid conflicts.

If you write your own Interrupt Service Routine: W, Status, FSR and PCLATH may be corrupted by servicing the USB interrupt and must be saved.

USB_MAIN.ASM provides a skeleton ISR which does this for you, and includes tests for each of the possible interrupt bits. This provides a good starting point if you haven't already written your own.

#### 10.9.3.2 Stack Levels

The hardware stack on the PICmicro® MCU is only 8 levels deep. So the worst case call between the application and ISR can only be 8 levels. The enumeration process requires 4 levels, so it's best if the main application holds off on any processing until enumeration is complete. **ConfiguredUSB** is a macro that waits until the enumeration process is complete for exactly this purpose, by testing the lower two bits of USWSTAT (0x197).

#### 10.9.3.3 ROM

The code required to support the USB interrupt, including the chapter 9 interface calls, but not including the descriptor tables, is about 1kW. The descriptor and string descriptor tables can each take up to an additional 256W. The location of these parts is not restricted.

#### 10.9.3.4 RAM

With the exception of Common RAM discussed below, servicing the USB interrupt requires ~40 bytes of RAM in Bank 2. That leaves all the General Purpose RAM in banks zero and one, plus half of bank two, available for your application to use.

#### 10.9.3.5 Common RAM Usage

The PIC16C745/765 has 16 bytes of common RAM. These are the last 16 addresses in each bank and all refer to the same 16 bytes of memory, without regard to which register bank is currently addressed by the RP0, RP1 and IRP bits.

These are particularly useful when responding to interrupts. When an interrupt occurs, the ISR doesn't immediately know which bank is addressed. With devices that don't support common RAM, the W register must be provided for in each bank. The 16C745/765 can save the appropriate registers in Common RAM and not have to waste a byte in each bank for W register.

10.9.3.6 Buffer Allocation

The PIC16C745/765 has 64 bytes of Dual Port RAM. 24 are used for the Buffer Descriptor Table (BDT), leaving 40 bytes for buffers.

Endpoints 0 IN and OUT need dedicated buffers since a setup transaction can never be NAKed. That leaves three buffers for four possible Endpoints, but the USB spec requires that low speed devices are only allowed 2 endpoints (USB 1.1 paragraph 5.3.1.2), where an endpoint is a simplex connection that is defined by the combination of Endpoint number and direction.

10.9.3.7 Vendor Specific Commands

Vendor specific commands are defined by the vendor. These are parsed out, but are not processed. Instead, control is passed to function **CheckVendor** where they can be processed.

10.9.4 FILE PACKAGING

The software interface is packaged into four files, designed to simplify the integration with your application.

File **USB_CH9.ASM** contains the interface and core functions needed to enumerate the bus. **DESCRIPT.ASM** contains the device, config, interface, endpoint and string descriptors. Both of these files must be linked in with your application.

**HIDCLASS.ASM** provides some HID Class specific functions. Currently only **GetReportDescriptor** is supported. Other class specific functions can be implemented in a similar fashion. When a token done interrupt determines that it's a class specific command on the basis that ReportType bit 6 is set, control is passed to function **ClassSpecific**. If you're working with a different class, this is your interface between the core functions and the class specific functions.

**USB_MAIN.ASM** is useful as a starting point on a new application and as an example of how an existing application needs to service the USB interrupt and communicate with the core functions.

10.9.5 FUNCTION CALL REFERENCE

Interface between the Application and Protocol layer takes place in three main functions: **InitUSB**, **PutUSB** and **GetUSB**.

**InitUSB** should be called by the main program immediately upon power-up. It enables the USB peripheral and USB Reset interrupt, and transitions the part to the powered state to prepare the device for enumeration. See Section 10.9.6 "Behind the Scenes" for details on the enumeration process.

**DeInitUSB** disables the USB peripheral, removing the device from the bus. An application might call **DeInitUSB** if it was finished communicating to the host and didn't want to be polled any more.

**PutUSB** (Buffer pointer, Buffer size, Endpoint) sends data up to the host. The pointer to the block of data to transmit is in the FSR/IRP, and the block size and endpoint is passed in W register. If the IN buffer is available for that endpoint, **PutUSB** copies the buffer, flips the Data 0/1 bit and sets the OWNS bit. A buffer not available would occur when it has been previously loaded and the host has not requested that the USB peripheral transmit it. In this case, a failure code would be returned so the application can try again later.

**GetUSB** (Buffer Pointer, Endpoint) returns data sent from the host. If the out buffer pointed to by the endpoint number is ready, as indicated by the OWNS bit, the buffer is copied from dual port RAM to the locations pointed to by the buffer pointer, and resets the endpoint for the next out transaction from the host. If no data is available, it returns a failure code. Thus the functions of polling for buffer ready and copying the data are combined into the one function.

**ServiceUSBInt** handles all interrupts generated by the USB peripheral. First, it copies the active buffer to common RAM, which provides a quick turn around on the buffer in dual port RAM and also avoids having to switch banks during processing of the buffer. File **USB_MAIN.ASM** gives an example of how **ServiceUSBInt** would be invoked.

**StallUSBEP/UnstallUSBEP** sets or clears the stall bit in the endpoint control register. The stall bit indicates to the host that user intervention is required and until such intervention is made, further attempts to communicate with the endpoint will not be successful. Once the user intervention has been made, **UnstallUSBEP** clears the bit allowing communication to take place. These calls are useful to signal to the host that user intervention is required. An example of this might be a printer out of paper.

**SoftDetachUSB** clears the DEV_ATT bit, electrically disconnecting the device from the bus, then reconnecting, so it can be re-enumerated by the host. This process takes approximately 50 mS, to ensure that the host has seen the device disconnect and reattach to the bus.

**CheckSleep** tests the UCTRL.UIDLE bit if set, indicating that there has been no activity on the bus for 3 mS. If set, the device can be put to SLEEP, which puts the part into a low power standby mode, until wakened by bus activity. This has to be handled outside the ISR because we need the interrupt to wake us from SLEEP, and also because the application may not be ready to SLEEP when the interrupt occurs. Instead, the application should periodically call this function to poll the bit, when the device is in a good place to SLEEP.

Prior to putting the device to SLEEP, it enables the activity interrupt so the device will be awakened by the first transition on the bus. The PICmicro device will immediately jump to the ISR, recognize the activity interrupt, which then disables the interrupt and resumes processing with the instruction following the **CheckSleep** call.

**ConfiguredUSB** (Macro) continuously polls the enumeration status bits and waits until the device has been configured by the host. This should be used after the call to **InitUSB** and prior to the first time your application attempts to communicate on the bus.

**SetConfiguration** is a callback function that allows your application to associate some meaning to a Set Configuration command from the host. The CH9 software stores the value in USB_Curr_Config so it can be reported back on a **Get Configuration** call. This function is also called, passing the new configuration in W. This function is called from within the ISR, so it should be kept as short as possible.

10.9.6 BEHIND THE SCENES

**InitUSB** clears the error counters and enables the 3.3V regulator and the USB Reset interrupt. This implements the requirement to prevent the PICmicro device from responding to commands until the device has been RESET.

The host sees the device and resets the device, to begin the enumeration process. The RESET then initializes the Buffer Descriptor Table (BDT), EndPoint Control Registers and enables the remaining USB interrupt sources.

The Interrupt transfers control to the interrupt vector (address 0x0004). Any Interrupt Service Routine must preserve the processor state by saving the FSRs that might change during interrupt processing. We recommend saving W, STATUS, PCLATH and FSR. W can be stored in unbanked RAM to avoid banking issues. Then it starts polling the Interrupt flags to see what triggered the Interrupt. The USB interrupts are serviced by calling **ServiceUSBInt** which further tests the USB interrupt sources to determine how to process the interrupt.

Then, the host sends a setup token requesting the device descriptor. The USB Peripheral receives the Setup transaction, places the data portion in the EP0 OUT buffer, loads the USTAT register to indicate which endpoint received the data and triggers the Token Done (TOK_DNE) interrupt. The Chapter 9 commands then interpret the Setup token and sets up the data to respond to the request in the EP0 IN buffer, then sets the UOWN bit to tell the SIE there is data available.

Then, the host sends an IN transaction to receive the data from the setup transaction. The SIE sends the data from the EP0 IN buffer and then sets the Token Done interrupt to notify us that the data has been sent. If there is additional data, the next buffer is setup in EP0 IN buffer.

This token processing sequence holds true for the entire enumeration sequence, which walks through the flow chart starting chapter 9 of the USB spec. The device starts off in the powered state, transitions to default via the Reset interrupt, transitions to ADDRESSED via the **SetAddress** command, and transitions to CONFIGURED via a **SetConfiguration** command.

The USB peripheral detects several different errors and handles most internally. The USB_ERR interrupt notifies the PICmicro device that an error has occurred. No action is required by the device when an error occurs. Instead, the errors are simply acknowledged and counted. There is no mechanism to pull the device off the bus if there are too many errors. If this behavior is desired, it must be implemented in the application.

The Activity interrupt is left disabled until the USB peripheral detects no bus activity for 3 mS. Then it suspends the USB peripheral and enables the activity interrupt. The activity interrupt then reactivates the USB peripheral when bus activity resumes, so processing may continue.

**CheckSleep** is a separate call that takes the bus idle one step further and puts the PICmicro device to SLEEP, if the USB peripheral has detected no activity on the bus. This powers down most of the device to minimal current draw. This call should be made at a point in the main loop where all other processing is complete.

## 14.0 INSTRUCTION SET SUMMARY

Each PIC16CXX instruction is a 14-bit word divided into an OPCODE, which specifies the instruction type and one or more operands, which further specify the operation of the instruction. The PIC16CXX instruction set summary in Table 14-2 lists byte-oriented, bit-oriented, and literal and control operations. Table 14-1 shows the opcode field descriptions.

For byte-oriented instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For bit-oriented instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For literal and control operations, 'k' represents an eight or eleven bit constant or literal value.

TABLE 14-1: OPCODE FIELD DESCRIPTIONS

| Field | Description |
|---|---|
| f | Register file address (0x00 to 0x7F) |
| W | Working register (accumulator) |
| b | Bit address within an 8-bit file register |
| k | Literal field, constant data or label |
| x | Don't care location (= 0 or 1)<br>The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools. |
| d | Destination select; d = 0: store result in W,<br>d = 1: store result in file register f. Default is d = 1 |
| label | Label name |
| TOS | Top of Stack |
| PC | Program Counter |
| PCLATH | Program Counter High Latch |
| GIE | Global Interrupt Enable bit |
| WDT | Watchdog Timer/Counter |
| $\overline{TO}$ | Time-out bit |
| $\overline{PD}$ | Power-down bit |
| dest | Destination either the W register or the specified register file location |
| [ ] | Options |
| ( ) | Contents |
| → | Assigned to |
| < > | Register bit field |
| ∈ | In the set of |
| *italics* | User defined term (font is courier) |

The instruction set is highly orthogonal and is grouped into three basic categories:

- Byte-oriented operations
- Bit-oriented operations
- Literal and control operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 14-2 lists the instructions recognized by the MPASM assembler.

Figure 14-1 shows the general formats that the instructions can have.

**Note:** To maintain upward compatibility with future PIC16CXX products, do not use the OPTION and TRIS instructions.

All examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

FIGURE 14-1: GENERAL FORMAT FOR INSTRUCTIONS

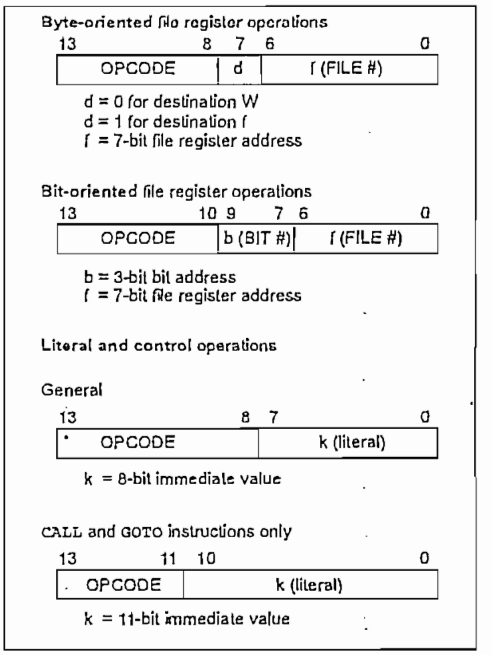


TABLE 14-2: PIC16CXX INSTRUCTION SET

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

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

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

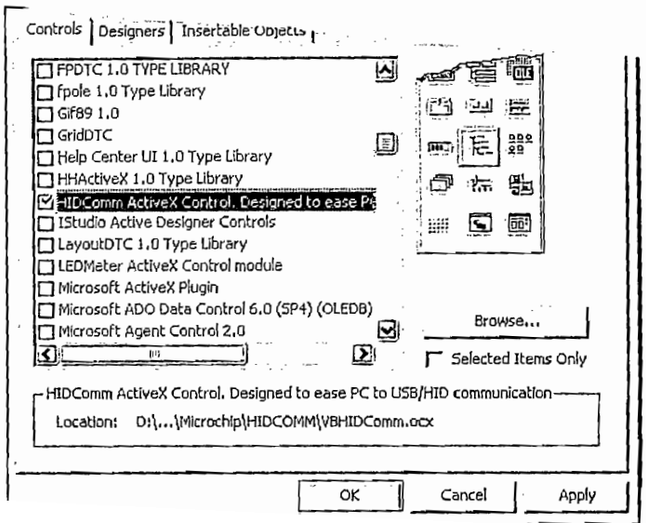
3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

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

# ANEXO 3

# CONTROL ACTIVEX HIDComm



Ponga una copia del control HIDComm en el formulario del proyecto. Las propiedades de este control se muestran en la ventana de propiedades de elementos de Visual Basic como en la figura A2.3.

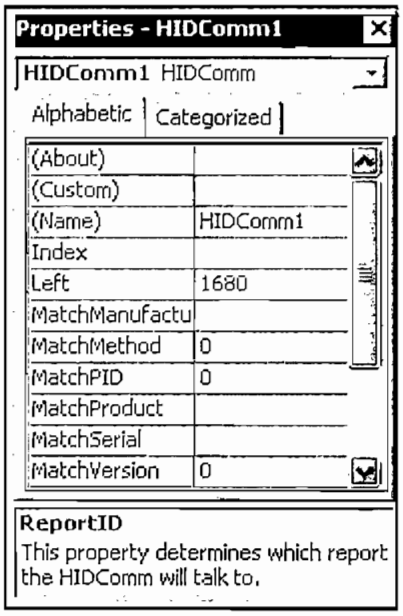



Figura A3.3 Propiedades del control HIDComm

A partir de cada una de las propiedades MatchXXX se puede acceder a una ventana amigable de interfaz para el usuario como el mostrado en la figura A3.4. En esta ventana se pueden modificar los diferentes criterios del dispositivo USB conectado como por ejemplo el número serial, fabricante, etc.

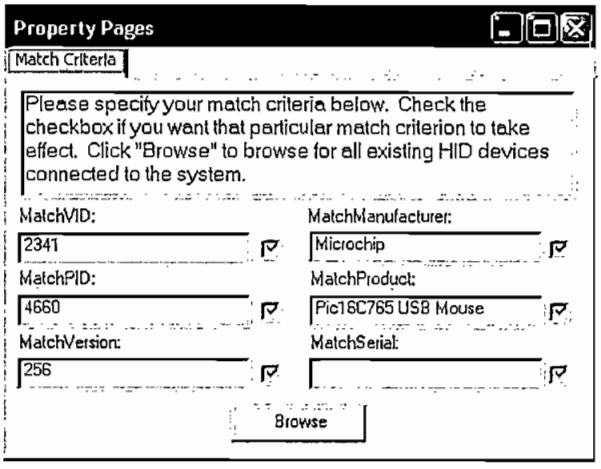



Figura A3.4: Criterios del dispositivo HID

Haciendo un click en el botón Browse el PC busca automáticamente todos los dispositivos USB conectados y los presenta en una pantalla como la mostrada en la figura A3.5.

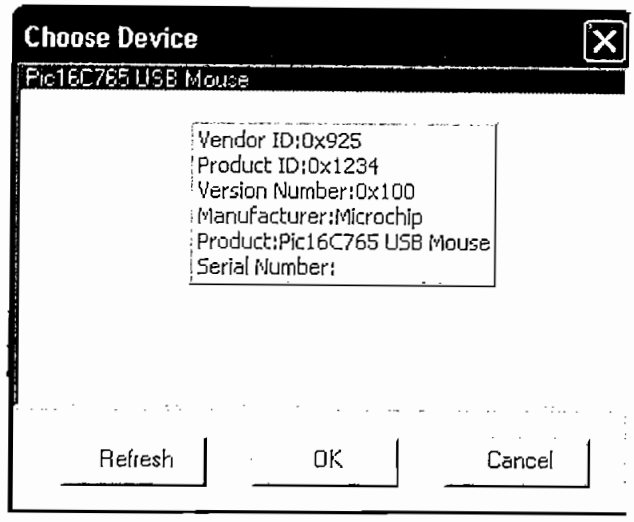



Figura A3.5 Lista de dispositivos HID conectados al bus USB

En esta nueva ventana se pueden observar las características del dispositivo HID como el ProductID, VendorID, Fabricante, Número de Versión y Número Serial.
En el caso que existan dispositivos conectados se puede seleccionar a cualquiera de los dispositivos y al presionar OK automáticamente se cargan los criterios en la página de criterios.
Una vez cargados los criterios del dispositivo HID dentro del control HIDComm se puede comenzar a establecer su comunicación mediante funciones de Conexión del dispositivo, Escritura y Lectura de reportes, etc.
Hay que tomar en cuenta que el HIDComm funciona solamente con dispositivos que tengan incorporados microcontroladores PIC16C745 o PIC16C765.

# ANEXO5

## MANUAL DE USUARIO

# MANUAL DE USUARIO USB ATMEL FLASH PROGRAMADOR V1.0

## INTRODUCCION

En este documento proporciona información acerca de los siguientes temas:

- Requerimientos del sistema.
- Instalación del software.
- Instalación del Hardware.
- Iniciando USB ATMEL Flash Programador.
- Interfaz de usuario en USB ATMEL Flash Programador.
- Manejo del programa USB ATMEL Flash Programador.

### Requerimientos del sistema

**Hardware:**

Para ejecutar USB ATMEL Flash Programador el PC debe tener las siguientes características.

- Memoria RAM de 128Mbyte como mínimo.
- Procesador Pentium celeron o superior.
- Espacio mínimo en Disco duro de 20 MegaBytes.
- Un Puerto USB.

**Software:**

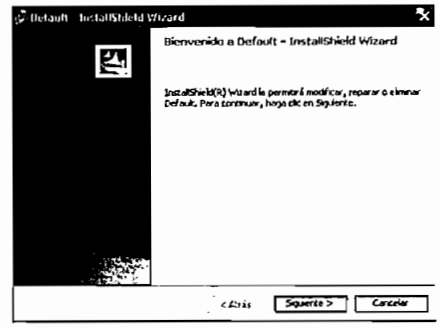
- Sistema Operativo: Windows 2000 Service Pack 3 o Windows XP Service Pack 1.

### Instalación del software

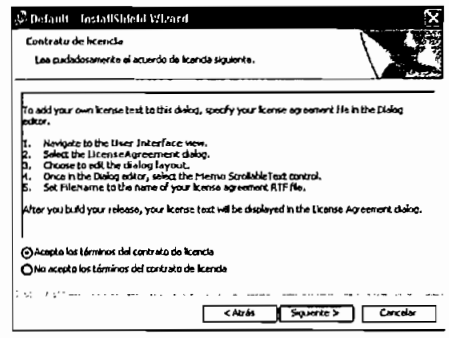
Para instalar USB ATMEL Flash Programador en el PC, deberá tener el archivo ejecutable Setup.exe generado a través de Visual Studio Installer 1.1. Este archivo lo encuentra en la carpeta USBAtmFlaProg.

Los pasos a seguir en la instalación es la siguiente:

1. Inicie la instalación haciendo doble clic en el archivo Setup.exe. La ventana que aparece es el siguiente:




2. Haga clic en el botón Siguiente de la pantalla anterior y se visualizara la ventana siguiente:




3. En esta ventana se presentan las condiciones bajo las cuales el usuario puede usar el software.

   Las condiciones son las siguientes:

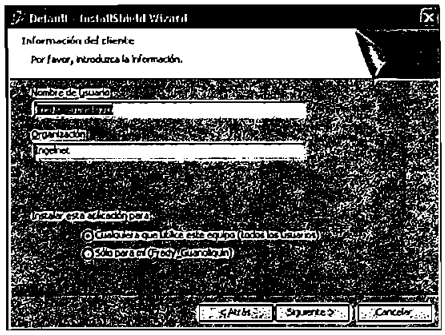
   El software es propiedad única y exclusivamente de la Escuela Politécnica Nacional, y es protegido bajo las leyes de propiedad Intelectual.

   Todos los derechos son reservados.

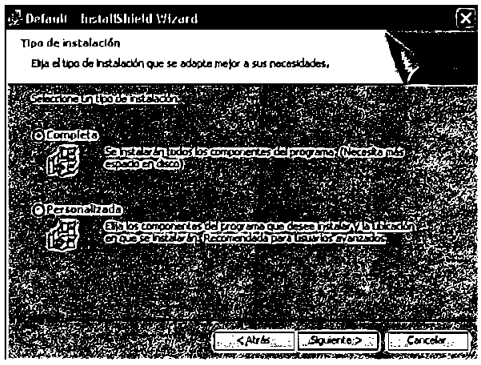
   Cualquier uso en la violación de las restricciones anteriores puede sujetar al usuario a las sanciones delictivas bajo las leyes aplicables, así como a la obligación civil para las condiciones de esta licencia.

   Si esta de acuerdo seleccione la opción acepto término del contrato y hacer un clic en botón Siguiente.

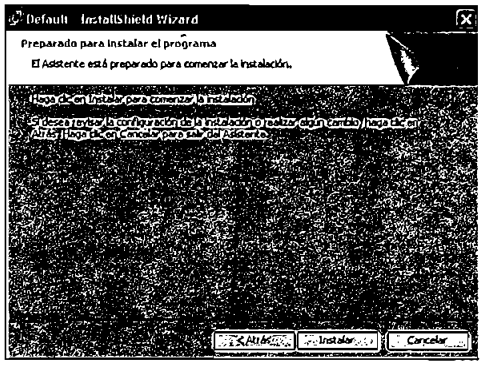
   Y si no esta de acuerdo escoger la opción de no ha aceptar los términos del contrato o hacer clic en cancelar:



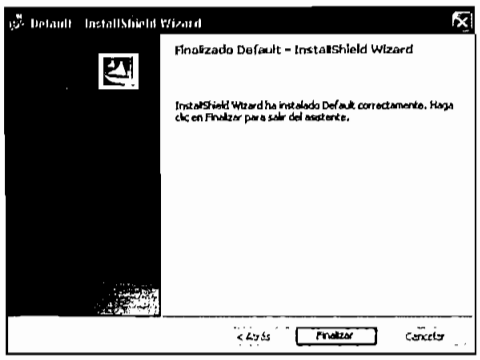

4. Seleccione cualquiera de las opciones y hacer clic en siguiente.




5. Escoja cualquiera de las dos opciones y haga un clic en siguiente.




6. Como siguiente paso hacer un clic en instalar.




7. Como último paso hacer un clic en finalizar y termina la instalación

**Instalación del Hardware**

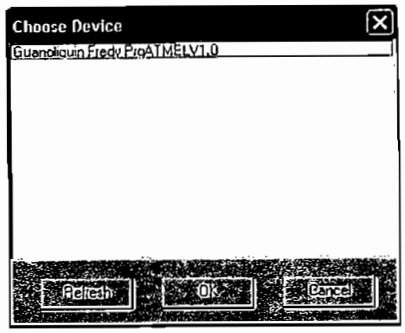
Los siguientes pasos son necesarios para conectar el USB ATMEL Flash Programador a su PC.

- El programador debe estar desconectado para poder conectar el conector serie A del cable USB al puerto USB de la PC.
- Asegurese que el adaptador de voltaje este desconectado antes de insertar el plug de poder al conector del Programador
- Conecte el adaptador. Nota: El led de **ON** se enciende.

**Iniciando USB ATMEL Flash Programador**

Inicie USB ATMEL Flash Programador haciendo doble click en Programs > USB Flash Program V1.0 > USBFlashProgram.

La ventana que aparece por primera vez pide al usuario que seleccione el dispositivo, para nuestro caso debemos seleccionar Guanoliquin Fredy PrgATMELV1.0 y hacer un clic en el boton OK. Nota: El hardware debe estar conectado al puerto USB de la PC.




Como siguiente paso se despliega la ventana del USB ATMEL Flash programador

**Interfaz de usuario en USB ATMEL Flash Programador.**

La Figura A6.8 muestra la ventana principal que estará interactuando directamente con el usuario, la cual está conformada por 4 componentes principales que son los siguientes:

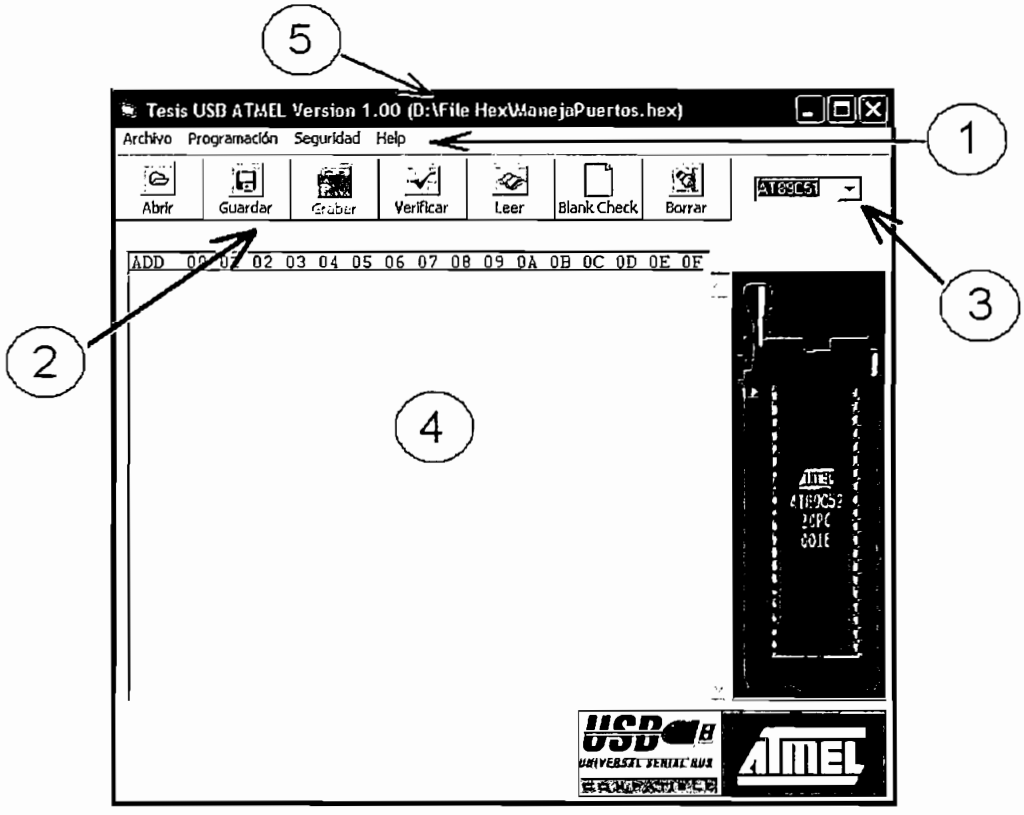



Figura A6.8: Ventana principal de USB ATMEL Flash Programador

1) Barra de menú Principal.

2) Botones de Control.

3) Menú de selección de dispositivo a programar.

4) Ventana de visualización de los contenidos.

5) Ventana de visualización del nombre del archivo importado.

A continuación se describen estos ítems:

**1) Barra de menú Principal**

Hay cuatro tipos de menús donde se encuentran las principales opciones de los comandos que se utilizan en de USB ATMEL Flash Programador. Dentro los menús se tienen las opciones que se describen a continuación:

***Archivo:*** En este menú el usuario tiene las siguientes opciones: Abrir archivo.hex, Guardar archivo.hex y salir.

***Programación:*** Dentro de este se tienen las propiedades de Leer dispositivo, Grabar dispositivo, Verificar dispositivo, Blank Check y Borrar

***Seguridad:*** En este menú se tiene tres opciones con los cuales se pueden seleccionar tres tipos de seguridades.

***Ayuda:*** Dentro de este menú se puede obtener información sobre el manejo el programa e información general del mismo.

**2) Botones de Control.**

Los botones de control son opciones visuales del menú principal. La descripción de cada una de ellas es el siguiente:


Abre el archivo.hex seleccionado. Nota: Este archivo debe ser en formato Intel.

con el que desee trabajar.

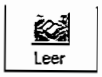
**4) Ventana de visualización.**

En este cuadro de texto se visualiza el contenido de la memoria flash o el archivo.hex importado.

Guarda el archivo abierto con el control anterior o los datos extraídos de la memoria flash del microcontrolador seleccionado.

Graba los datos extraídos del archivo.hex en la memoria flash del microcontrolador

Verifica que los datos extraídos del archivo.hex estén correctamente grabados en la memoria flash del microcontrolador seleccionado.



Lee la memoria flash del microcontrolador seleccionado.

Verifica que la memoria flash del microcontrolador seleccionado este vacía.

Borra la memoria flash del microcontrolador seleccionado.

Menú de selección de microcontrolador a grabar, verificar, etc.

**3) Menú de selección de microcontrolador a programar.**

En este menú se encuentra la lista de todos los microcontroladores con los que se pueden trabajar, Aquí el usuario debe seleccionar el microcontralador con el que desee trabajar.

**4) Ventana de visualización.**

En este cuadro de texto se visualiza el contenido de la memoria flash o el archivo.hex importado.

**Manejo del programa USB ATMEL Flash Programador**

Para el manejo de cualquiera de las opciones que ofrece el programa USB ATMEL Flash Programador se recomienda seguir los siguientes pasos que garantizan el correcto funcionamiento del mismo.

- Conecte el hardware del programador al PC de acuerdo a los pasos que se indicaron anteriormente e inicie el programa USB ATMEL Flash Programador.
- Inserte el microcontrolador con el que se desee trabajar en el zócalo que corresponda. Nota: Conecte apropiadamente el microcontrolador.
- Dentro del programa USB ATMEL Flash Programador seleccione el dispositivo en el menú de dispositivos.
- Utilice cualquiera de las opciones.

# ANEXO 6

## FORMATO INTEL

# FORMATO INTEL

El formato usado en los archivos objetos (Hex) para la programación de los microcontroladores ATMEL es el formato Intel.

Este formato esta organizado por una sucesión de registros que van escritos en una línea seguidos por los códigos ASCII de retorno de carro y avance de línea. Dentro de un registro el dato binario es representado por dígitos ASCII hexadecimales, dos dígitos por byte binario. Por ejemplo el valor de 255 es representado por "FF" en el archivo.

El detalle de los registros se describe a continuación:

| Designación del formato | Número de Caracteres | Descripción |
|---|---|---|
| Cabecera | 1 | Representado siempre por ":" |
| Bytes a Grabar | 2 | Indica el número de datos a ser grabados que contiene el registro. |
| Dirección Inicio | 4 | Indica la dirección de la localidad de memoria donde se va a ubicar el primer dato del registro. Para el caso de Fin de registro el valor es "0000". |
| Identificador de tipo de línea | 2 | Puede indicar dos posibilidades:<br>"00": Indica que es una línea intermedia.<br>"01": Indica línea final. |
| Datos | Variable | Representan los datos a ser grabados. La longitud o el número de datos es indicada por los "Bytes a Grabar". |
| Checksum | 2 | Representa el complemento de dos de la suma de los Bytes precedentes. |

En las siguientes figuras se representan la disposición de los caracteres en los registros.

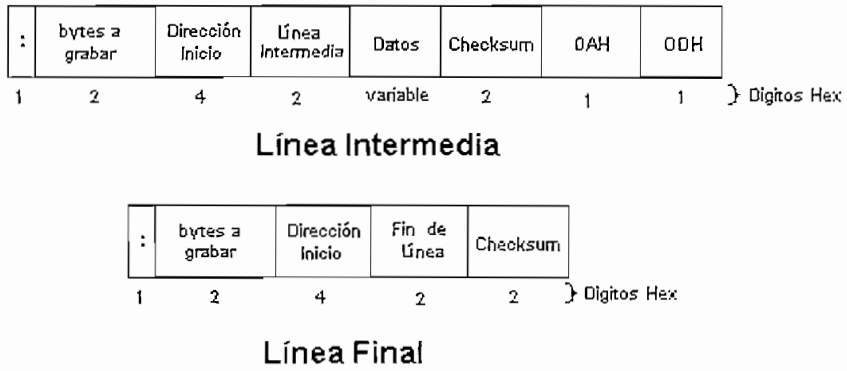



Figura A6.1: Disposición de caracteres en los registros.

A continuación se indica con un ejemplo el formato Intel.

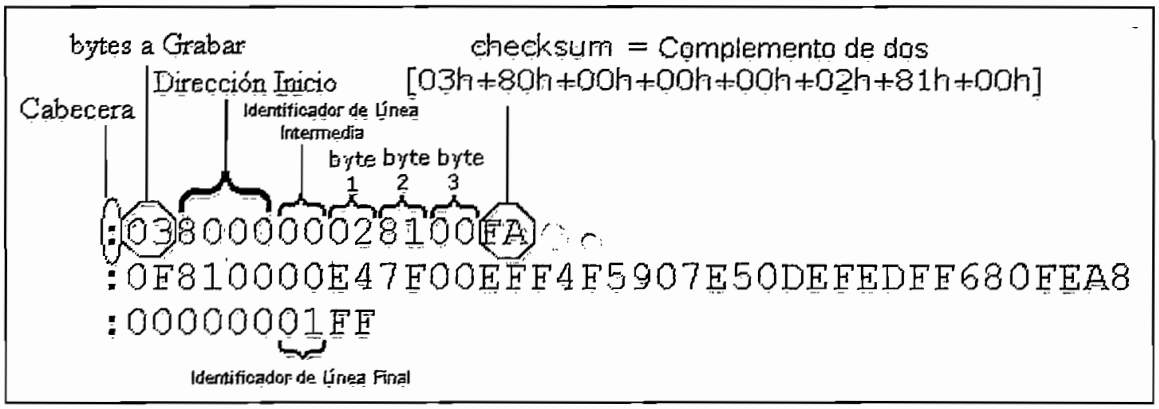



Figura A6.2: Ejemplo de Formato INTEL.