Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Kit y Cargador para Microcontroladores PIC

Autor: ***Marcos Michel Orta Hernández.***

Tutores: ***Ing. Arnaldo Moreno Montes de Oca.***

Msc. Carlos Alberto Bazán Prieto.

Santa Clara

2009

"Año 51 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

KIT Y CARGADOR PARA MICROCONTROLADORES PIC

Autor: Marcos Michel Orta Hernández

E-mail: mmorta@uclv.edu.cu

Tutores : Ing. Arnaldo Moreno Montes de Oca.

Profesor, Dpto. de Electrónica y Telecomunicaciones

Facultad de Ing. Eléctrica. UCLV.

E-mail: arnaldomm@uclv.edu.cu

Msc. Carlos Alberto Bazán Prieto.

Profesor, Dpto. de Electrónica y Telecomunicaciones

Facultad de Ing. Eléctrica. UCLV.

E-mail: cabazan@uclv.edu.cu

Santa Clara

2009

"Año 51 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Tutor

Firma del Jefe de Departamento
donde se defiende el trabajo

Firma del Responsable de
Información Científico-Técnica

PENSAMIENTO

El trabajo es la condición inevitable de la vida humana,

la verdadera fuente de bienestar humano.

Leon Tolstoi

DEDICATORIA

A mis padres que han sabido encaminarme a lo largo de esta dura faena.

A mis abuelos que me han alentado en todo momento

A mis hermanos y mi sobrina que siempre me han apoyado.

A los demás familiares.

A esos amigos que han hecho realidad uno de mis sueños.

AGRADECIMIENTOS

A mi tutor Arnaldo Moreno Montes de Oca, por su activa participación en este proyecto.

A mi tutor Carlos A. Bazán Prieto por su ayuda.

Al colectivo docente de la Facultad de Ingeniería Eléctrica de la Universidad Central "Marta Abreu" de Las Villas, por haber puesto todo su empeño en hacer de nosotros excelentes profesionales.

A Mario y Migdalia, sin ellos este sueño no se hubiera materializado.

* * *

A mis padres, hermanos y abuelos, aunque algunos no estén presente físicamente estarán guiándome a lo largo de toda mi vida

* * *

Al resto de mis familiares y amigos.

TAREA TÉCNICA

1. Revisión crítica de la bibliografía del tema.
2. Estudio del subsistema de comunicación USB en los PICs y la PC.
3. Estudio del software de ayuda al diseño.
4. Elaborar un Kit que permita cargar aplicaciones empleando los subsistemas analizados.
5. Elaboración de aplicaciones que permitan ejemplificar la utilización del Kit.
6. Confección y presentación del informe.

Firma del Autor

Firma del Tutor

RESUMEN

En el presente trabajo se describe el diseño, construcción, pruebas y resultados de un conjunto de herramientas, las cuales constituyen un Kit de desarrollo de aplicaciones basado en el microcontrolador PIC 18F2550. Además se describen los software empleados en el diseño así como el por qué de su elección. Este kit está constituido por una aplicación desarrollada utilizando LabVIEW la cual se ejecuta en la PC y se comunica a través del puerto USB con el microcontrolador, con el objetivo de enviarle un programa previamente desarrollado que se quiera ejecutar en el mismo. La aplicación que se ejecuta en el microcontrolador, constituye un cargador que recibirá el programa a través de su módulo USB y lo grabará en su memoria de programas, para su posterior ejecución. Esta aplicación se desarrolló utilizando el compilador de C PCHW. Para la simulación, diseño del circuito esquemático y circuito impreso, se utilizaron las herramientas que brinda el Proteus. Finalmente se incluye un manual de usuario que facilita la utilización del kit. Este kit facilitará la creación y puesta a puntos de programas basados en su arquitectura, lo cual será de gran utilidad en las asignaturas de microcontroladores.

Palabras claves: Microcontrolador PIC, kit, cargador, LabVIEW, PCHW, Proteus, USB.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Los microcontroladores han ido evolucionado a través de los años. Muchas de las aplicaciones de la microelectrónica tienen como núcleo a los microcontroladores, y los PICs son los más empleados y difundidos en el mundo.

Hoy existen millones de microchips de alguna clase en uso y el número de productos que funcionan en base a uno o varios microcontroladores aumenta de forma exponencial. El mayor atributo del microcontrolador es que puede integrar inteligencia casi a cualquier artefacto. La presencia de subsistemas avanzados capaces de dotar a los microcontroladores de prestaciones cada vez más valiosas hace necesario el dominio y empleo de estos subsistemas en el desarrollo de aplicaciones de creciente complejidad.

De lo anteriormente expuesto se desprenden las siguientes interrogantes científicas:

¿Cómo diseñar un Kit de entrenamiento para microcontroladores PICs?

¿Cómo desarrollar el ensamblaje, compilación y puesta a punto de programas utilizando el Kit de entrenamiento diseñado?

Entre los objetivos del presente trabajo están, el diseño, programación y construcción de un Kit para microcontroladores PICs, que permita poner a punto programas. Hacer un estudio teórico del microcontrolador PIC 18F2550. Diseñar un Kit de entrenamiento con el microcontrolador PIC 18F2550 que permita la puesta a punto de programas. Para poder lograr el vencimiento de los objetivos planteados se ejecutaron las siguientes tareas: primero, búsqueda bibliográfica de las herramientas utilizadas y del microcontrolador PIC 18F2550; segundo, estudio del protocolo USB y del formado del fichero Intel-Hex.; tercero, diseño y desarrollo del Kit; cuarto, confección de un programa que permita

ejemplificar los pasos necesarios para la utilización del Kit, y destaque las ventajas y facilidades del mismo.

Organización del informe.

Este informe se ha estructurado en tres capítulos que tratan las siguientes temáticas:

CAPITULO I*:* Análisis Teórico.

Se dedica a la teoría relacionada a la aplicación. Se analizan: la función de un cargador, el microcontrolador PIC 18F2550, el protocolo USB, el formato del fichero Intel Hexadecimal, y los software empleados en el diseño y construcción de las herramientas.

CAPITULO II: Diseño y Construcción de las Herramientas.

Se describe como se diseñaron y construyeron las herramientas, así como el funcionamiento de las mismas. Se realiza además el montaje real, auxiliándose de los resultados obtenidos anteriormente.

CAPITULO III: Desarrollo de Aplicaciones.

Se dedica a los resultados finales, para lo cual se prueba el Kit con diversas aplicaciones. Además se incluye un manual de usuario.

CAPÍTULO 1. ANALISIS TEÓRICO.

En este capítulo se aborda el marco teórico necesario para la realización de la aplicación, lo cual implica conocer que es un cargador y su utilidad, el conocimiento del protocolo USB y del fichero con formato Intel Hexadecimal. Además se describen las potencialidades de las herramientas de diseño así como el porque de su elección. Finalmente se dan a conocer las posibilidades del microcontrolador 18F2550 y de su módulo USB.

1.1 Partes y Función del Kit.

El kit está compuesto por dos partes fundamentales, una aplicación que se ejecuta en la PC y un programa que se ejecuta en el microcontrolador PIC 18F2550, esté último montado en un hardware diseñado en este trabajo. Para la comunicación de estas dos partes se utilizó el estándar USB (*Universal Serial Bus*), y sobre este se creó un protocolo sencillo el cual permite la transferencia del fichero con formato Intel Hexadecimal, y algunos comandos. El kit tiene como función, enviar desde la PC el fichero Intel Hexadecimal resultante de la compilación de un código desarrollado para el PIC 18F2550. Luego este fichero será recibido y cargado en memoria de programas dentro de del PIC 18F2550, para su posterior ejecución. De esta forma la ejecución real de un programa desarrollado para este PIC se hace de forma muy sencilla, solo conectando el hardware al puerto USB de la PC e interactuando con la aplicación que se ejecuta en la misma. Además si se utiliza el esquema del hardware diseñado, se facilita aún más el trabajo.

1.2 Estándar USB.

El USB presenta las ventajas de expansibilidad, sencillez de configuración y uso del hardware, este protocolo presenta beneficios tangibles para el usuario como son:

- Fácil expansión de periféricos en la PC, no debe hacer falta más que conectar el periférico y emplearlo.
- Bajo costo para aplicaciones que demandan velocidades de transferencia altas.
- Cómoda integración de dispositivos de tecnologías y fabricantes diferentes.
- Soporte para plataformas diversas de la línea de las PC compatibles.
- Posibilitar la producción de nuevos dispositivos capaces de aprovechar sus ventajas.

USB corrigió tres de los defectos de las interfaces que le precedían. Para mejorar funcionamiento, se diseñó una tarifa de datos 12Mbps (con un índice de poca velocidad alternativo de 1.5Mbps). En abril 27 de 2000, un nuevo grupo, conducido por Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, y Philips, publicó un estándar revisado USB, versión 2.0. El cambio fundamental era un mejoramiento en el funcionamiento, subiendo la velocidad de 12Mbps a 480Mbps. El nuevo sistema incorpora todos los protocolos del viejo y es compatible completamente con el anterior. Los dispositivos negociarán la velocidad común más alta y la utilizarán para sus transferencias. Los conectores y los cables permanecieron iguales (Rosch, 2003). Para permitir que un solo *socket* de la computadora maneje tantos periféricos como sean necesarios, diseñaron el sistema para acomodar hasta 127 dispositivos por puerto.

Presenta soporte *plug-and-play* de modo que cada conexión pudiera configurar por si sola. Usted podría incluso conectar en caliente los nuevos dispositivos y utilizarlos inmediatamente

1.2.1 Topología de Bus.

La forma física en la que los elementos se interconectan dentro del sistema USB, puede asemejarse a la topología estrella piramidal. 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 al mismo varios dispositivos o en su defecto nuevos *hubs*. Esta disposición significa que los computadores con soporte para USB han de tener tan solo uno o dos conectores USB, pero ello no representa poder contar con tan solo dos dispositivos de esta clase, como un ratón y un teclado (Axelson, 2001). Muchos dispositivos USB han de traer conectores USB adicionales incorporados, por ejemplo un monitor puede tener tres o cuatro conectores USB donde pueden ir el teclado, el ratón, y algún otro dispositivo. Por su parte el teclado puede tener otros más, y así sucesivamente hasta tener 127 dispositivos, todos funcionando simultáneamente. Aún así, existirán dispositivos específicos destinados a ampliar la cantidad de conectores, estos se denominan *hubs*, y su funcionamiento como apariencia física está muy cercana a la de los *hubs* de redes *Ethernet*. Un *hub* de ocho puertos o conectores, puede ser acoplado a uno de los puertos USB del computador, ampliando la cantidad de dispositivos que se pueden emplear (Anderson, 2001, Implementers, 2003).

A diferencia de los dispositivos y los *hubs*, existe tan solo un *host* dentro del sistema USB, que es el computador mismo, particularmente una porción del mismo denominado controlador USB del *host*. Éste tiene la misión de hacer de interfaz entre el computador mismo y los diferentes dispositivos. Existen algunas particularidades respecto a este controlador. Su implementación es una combinación de *hardware* y *software* todo en uno. Puede proveer de uno o dos puntos de conexión iniciales, denominados *hub* raíz o (*root hub*), a partir de los cuales y de forma ramificada irán conectándose los periféricos (Philips, 2000, Mueller, 2006).

Dentro de la terminología USB, todos los dispositivos que pueden ser conectados al bus USB se denominan funciones. Son funciones típicas el ratón, el monitor, altoparlantes, *modem*, etc.

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 (Rosch, 2003, Fujitsu, 2003).

1.2.2 Interfaz Eléctrica y Mecánica.

Para transmitir en modo *high-speed*, el transmisor activa una fuente de corriente interna, derivada a partir de su fuente de alimentación positiva, y dirige dicha corriente hacia una de las dos líneas de datos por medio de un conmutador de corriente de alta velocidad. De esta manera, el transmisor genera los estados J y K *high-speed* en el cable (Philips, 2000).

Esta conmutación dinámica de corriente sobre ambas líneas de datos D+ y D-, sigue las mismas reglas de codificación NRZI y de inserción de *bit* (*bit-stuffing*) ya utilizadas en los modos *full-speed* y *low-speed*. El estado J se obtiene dirigiendo la corriente sobre la línea D+, mientras que el estado K se obtiene dirigiendo la corriente sobre la línea D-.

En modo *high-speed*, tanto el transmisor como el receptor activan unas resistencias de terminación entre cada línea y masa (45 Ohms +/- 10 %), de forma que el valor nominal de la corriente (17,78 mA) produce un voltaje nominal en la línea de 400 mV. El voltaje diferencial nominal (D+ - D-) es, por lo tanto, de +400 mV para el estado J y de – 400 mV para el estado K (Peacock, 2007).

El sistema del USB implica cuatro diversos estilos de conectores, dos *sockets* y dos conectores. Cada *socket* y enchufe viene en dos variedades: A y B (Anderson, 2001).

En esta aplicación se utilizó el conector tipo A, permitiendo la conexión sin adaptadores a los puertos de la PC.

1.2.3 Transferencias de Datos.

El USB tiene su modelo particular de capas. En el cual existen 3 de ellas. Por lo general, las 2 capas superiores se implementan en *software* y la inferior en *hardware*, tanto en el esclavo como en el *host*.

El *software* en el *host* se comunica con un dispositivo lógico vía un sistema de flujos de comunicación. Los flujos de la comunicación son transportados sobre los conductos

lógicos de comunicación (*pipes*), entre los *endpoint* y los *buffers* de memoria del lado del *host*.

Un *endpoint* es una porción de memoria singularmente identificable de una función de USB la cual provee la terminación del flujo de comunicación entre el *host* y la función. Cada *endpoint* posee un único identificador, la combinación de la dirección de la función más el número del *endpoint* permite que cada uno sea diferenciado singularmente (Philips, 2000, Fujitsu, 2003).

Los *endpoints* poseen las características que determinan el tipo de servicio de transferencia requerida entre el *host* y la función o dispositivo.

Los *pipes* entran en existencia cuando un dispositivo es configurado.

USB 1.x dividía el tiempo en tramas de un milisegundo. Adicionalmente, USB 2.0 define un tiempo de micro trama de ciento veinticinco microsegundos. Al igual que en USB 1.x se reservan ciertos porcentajes del tiempo de trama, para dar servicio a las distintas transacciones de control, interrupción e isócronas *full/low-speed*, en USB 2.0 también se reservan ciertos porcentajes del tiempo de micro trama, para dar servicio a los distintos tipos de transacciones *high-speed* (Fujitsu, 2003).

La transferencia de información entre el *host* y los *endpoints* de un dispositivo es distribuida a través de una serie de transacciones y sobre una serie de *frames*. Cada transacción incluye un número determinado de paquetes y en un *frame* están incluidas un número determinado de transacciones. Las transacciones se agrupan formando transferencias o transmisiones, las que poseen sentido propio, y cada transmisión obedece a un tipo predeterminado como lo son los modos: *Interrupt*, *Control*, *Bulk* e *Isochronous*.

El modo que se utiliza en esta aplicación es el *Bulk*.

Una transferencia es un bloque de datos que conforma una estructura comprensible para el *host* o el esclavo. Existen 2 tipos básicos de transferencias, que estén directamente relacionadas a los tipos de *endpoints*: las de control y las de datos. Dentro de las transferencias de datos existen 3 tipos: Interrupción, *bulk*, isocrónicas (Anderson, 2001).

Las transferencias *bulk* están diseñadas para soportar aquellos dispositivos que precisan enviar o recibir grandes cantidades de datos, con latencias que pueden tener amplias variaciones, y en que las transacciones pueden utilizar cualquier ancho de banda disponible (Axelson, 2001, Philips, 2000).

1.2.4 Esclavos y Descriptores.

Un esclavo USB tiene una cantidad finita de estados disponibles: (Aguirre, 2005).

- *ATTACHED*: Es el primer estado en que se encuentra el esclavo apenas es conectado físicamente al *bus* USB.

- *POWERED*: Se considera que un esclavo USB está en este estado cuando se ha alimentado.

- *DEFAULT*: Un esclavo USB está en este estado después de que ha recibido una señal de *reset*, esto le permitirá comunicarse con el *host* mediante el *endpoint0*.

- *ADDRESSED*: Un esclavo USB se encuentra en éste estado una vez que le ha sido asignada una dirección USB por el *host*.

- *CONFIGURED*: Antes de que el esclavo USB pueda prestar su utilidad, éste debe se configurado. Desde el punto de vista del dispositivo, éste estará configurado una vez que haya podido procesar correctamente un *request* del tipo *SetConfiguration ()*.

- *SUSPENDED*: Este estado es usado para ahorrar energía. Un esclavo entra en este estado cuando no ha recibido un paquete EOP durante 3 MS. Un dispositivo debe salir de este estado cuando detecta actividad en el *bus*. Durante este estado se mantiene la dirección USB que fue asignada al dispositivo USB.

Los descriptores son tablas en las que los esclavos almacenan información sobre sus características. Dichas tablas son no modificables por el *host* (grabadas en memoria de solo lectura). Los descriptores son jerárquicos (figura 1.1). Algunos pueden contener información relativa a *string descriptors*, que tienen información para que el *host* muestre al usuario (Peacock, 2007).

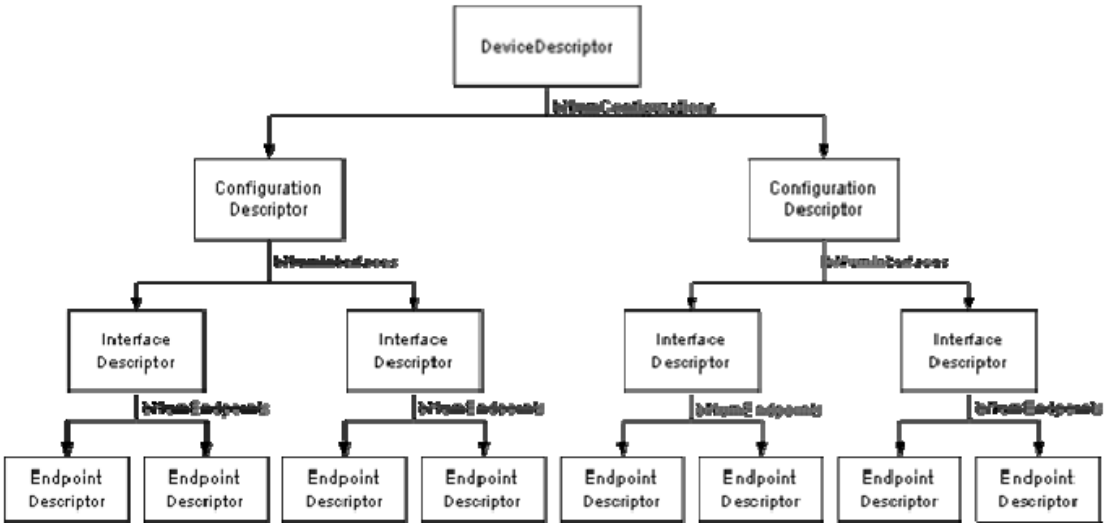

Figura 1.1. Jerarquía de los descriptores.

A continuación se describen los descriptores comunes a todos los tipos de esclavos USB (Aguirre, 2005):

- *Device Descriptor*: Contiene información sobre el máximo tamaño de paquete que soporta el *endpoint0,* cuántas configuraciones soporta el esclavo, y otras informaciones. Es el primero que lee el *host.*

- *Configuration Descriptor*: Existe uno por cada posible forma de operar del esclavo (solo puede haber una configuración activa en un determinado momento). Tiene información sobre cuántas interfaces existen por configuración.

- *Interface Descriptor*: Tiene información sobre el número de *endpoints* (excepto el *endpoint0*) que utiliza la Interfaz y sobre la clase a la que pertenece.
- *Endpoint Descriptor*: Existe uno por cada *endpoint* de una *interfaz*. Tiene información sobre el número del *endpoint*. También sobre el tipo de transferencia que realiza (*Control*, *Interrupt*, *Bulk* o *Isochronous*). Y también tiene información sobre el tamaño máximo de paquete del *endpoint*.

1.3 Formato Intel Hexadecimal.

Intel Hexadecimal Format (Formato Intel Hexadecimal), es un estándar para guardar los programas de idioma de máquina en un formato imprimible. Un archivo INTEL (.hex.) es

una serie de líneas o “registros de grabación” que contiene los campos mostrados en la tabla 1.1

Tabla 1.1 Campos de las líneas de archivo INTEL.

Campo	Bytes	Descripción
Marca de Registro	1	": " indica comienzo del registro.
Longitud de Registro	1	Número de bytes de datos en el registro (en hexadecimal).
Dirección de carga	2	Dirección de carga para los bytes de datos.
Tipo de registro	1	00=datos 01=fin registro.
Bytes de datos	0-16	datos
Suma de Comprobación	1	Suma de comprobación de todos los bytes.

En la figura 1.2 de muestran todos los campos de este fichero en un segmento del mismo.

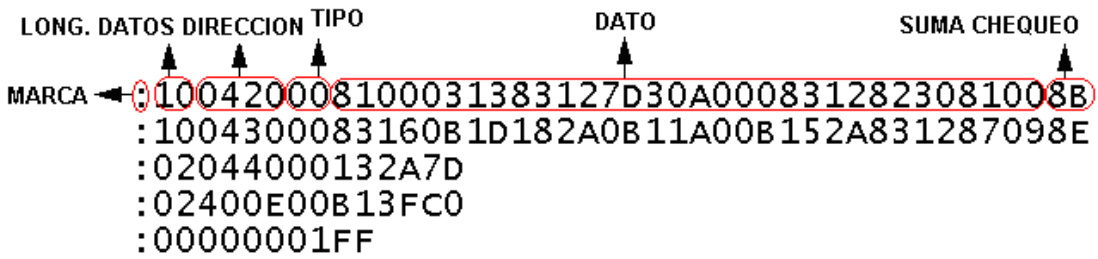

Figura 1.2. Campos del fichero INTEL.

- El campo Marca de Registro de 1 byte indica el comienzo de la transmisión del registro, dentro de este byte se almacena el código ASCII del carácter ‘:’(3AH).
- En el byte de Longitud de Registro se guarda el número de bytes de datos que serán transmitidos en el registro (en hexadecimal).
- Los bytes de Dirección de Carga contienen la dirección de arranque (en memoria de programa) para los bytes de los datos.

- El byte Tipo de Registro indica el contenido de lo que será transmitido a continuación, 00=datos; 01=fin registro.
- El próximo campo que puede ser de 0 a 16 bytes y contiene los datos del programa.
- El byte Suma de Comprobación se calcula de manera que la suma de todos los bytes del campo más el propio *checksum* sea 0x00 (00 en hexadecimal). Sólo se considera el byte menos significativo de la suma.

Para calcular la suma de comprobación como primer paso se suman todos los bytes del registro con excepción del carácter correspondiente a los dos puntos, se trunca el resultado y se toma el byte menos significativo. A este número se le halla el complemento A2, para ello, se deben cambiar los 1 por 0 y viceversa.

1.4 Microcontrolador PIC 18F2550.

Este microcontrolador es de gama alta dentro de la familia de los PICs. Posee amplias memorias de datos y programas Tabla 1.2.

Tabla 1.2. Memoria interna del PIC 18F2550.

Device	Program Memory		Data Memory	
	FLASH (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)
PIC18F2550	32 K	16384	2048	256

Algunos de los otros aspectos de este microcontrolador son:

- Posee un amplio *set* de instrucciones y opcionalmente un *set* extendido.
- Arquitectura optimizada para la utilización de compiladores de C.
- Auto programable controlado mediante el software.
- Niveles de prioridades para las interrupciones.
- Tres interrupciones externas.

- Cuatro módulos temporizadores.
- Un módulo EUSART (*Enhanced Universal Synchronous Asynchronous Receiver Transmitter*).
- Un módulo conversor análogo/digital de diez bits.
- Un módulo USB.

Dentro de los módulos que posee este microcontrolador, el módulo USB es de gran importancia en este trabajo. Sus principales características son:

- Compatible con la versión USB 2.0.
- Puede operar en los modos *Low Speed* (1.5 Mb/s) y *Full Speed* (12 Mb/s).
- Soporta los modos de transferencia de datos: *Control*, *Interrupt*, *Isochronous* y *Bulk*.
- Soporte de hasta 32 *endpoints* (16 bidireccionales).
- Dispone de un KB de Memoria RAM (*Random Access Memory*), que puede ser accedida por el módulo USB o por el núcleo del microcontrolador.
- Integra un *transceiver* USB con el regulador de voltaje de 3.3V necesario para que funcione el mismo.
- Dispone de una interfaz para la utilización de un *transceiver* externo en caso de que no se utilice el interno.

La distribución de terminales del microcontrolador se muestra en la figura 1.3.

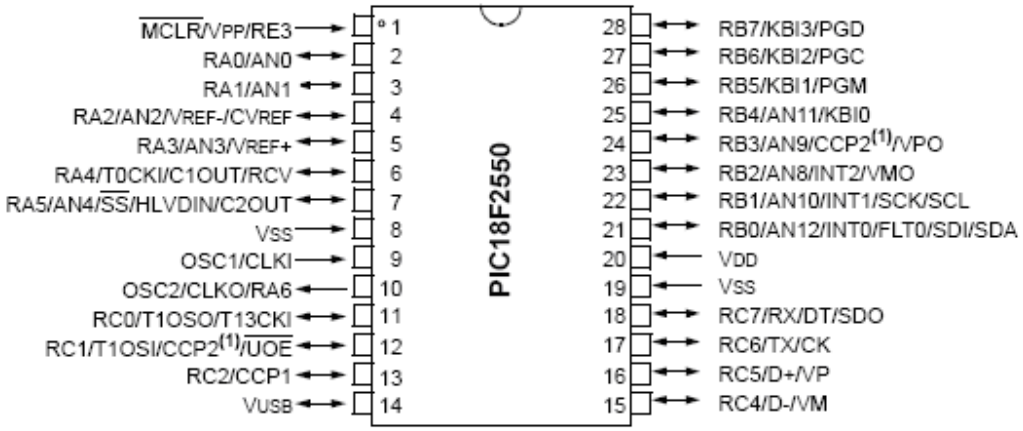

Figura 1.3. Distribución de terminales del PIC 18F2550.

1.5 Herramientas Utilizadas.

En este epígrafe se describen las herramientas utilizadas y además el porque de su selección.

1.5.1 LabVIEW.

Arquitectura de *software* para instrumentos virtuales (VISA) es una librería de interfaz simple para controlar VXI, GPIB, RS-232 y otros tipos de instrumentos en las plataformas de LabVIEW. VISA es el estándar utilizado por la Alianza de Sistemas de VXI *plug & play*, que incluye más de 35 de las compañías más grandes en la industria de instrumentación. El estándar VISA unifica la industria para hacer *software* que pueda ser interpretado y rehusado por más tiempo, sin importar el tipo de operación de su instrumento. NI-VISA es un *software* controlador que está disponible la página Web de la *National Instrument* (Instrument, 2008).

LabVIEW (*Laboratory Virtual Instrument Engineering Workbench*) es un lenguaje de programación gráfico para el diseño de sistemas de adquisición de datos, instrumentación y control. LabVIEW permite diseñar interfaces de usuario mediante una consola interactiva basada en *software*. Se puede diseñar especificando su sistema funcional, su diagrama de bloques o una notación de diseño de ingeniería.

Éste tiene diferentes funciones implementadas a través de VISA para la comunicación por USB las cuales facilitan el diseño de las aplicaciones.
Algunas de las funciones se muestran a continuación: (Instrument, 2007)

- VISA *Open*
- VISA *Close*
- VISA *Read*
- VISA *Write*
- VISA *Read To File*
- VISA *Write From File*

Y otra gran variedad de funciones provistas por VISA para un control eficiente del dispositivo.

1.5.2 Compilador de C PCHW.

El compilador C PCWH versión 4.033 de la empresa CCS (*Custom Computer Services*), es una de las herramientas compatible con MPLAB para la programación en lenguaje de alto nivel de la familia de microcontroladores PIC de Microchip.

Este es un entorno de desarrollo integrado, que incluye los compiladores en lenguaje C, PCB (para PICs de 12 bits), PCM (para PICs de 14 bits) y PCH (para PICs de 16 bits). Se integra con el MPLAB automáticamente, usándose el ambiente del MPLAB.

Este compilador cuenta con diversas directivas de preprocesador, con objetivos como: inclusión de ficheros cabecera, definición de la palabra de configuración, definición de la frecuencia del oscilador, habilitación de diferentes módulos pertenecientes al microcontrolador, definición de subrutinas de atención a las interrupciones, inserción de código ensamblador, etc.

Otro aspecto, es la existencia de diversas funciones incorporadas (*built-in functions*) las cuales permiten un fácil manejo de los diferentes módulos del microcontrolador, haciendo ciertos procedimientos muy sencillos. (Custom Computer Services, Inc., 2007). Las funciones para el trabajo con la comunicación USB, fue uno de los motivos por el que se seleccionó este compilador.

1.5.3 Proteus.

El Proteus es un software que permite el diseño y desarrollo de circuitos eléctricos con gran variedad de elementos, incluyendo microcontroladores de diversas familias. Brinda también la posibilidad de interactuar durante la simulación, donde son incluidos diversos componentes tales como pantallas LCD, LEDs, interruptores, botones, teclados matriciales, etc. Esta es una cualidad que lo hace particularmente novedoso y muy útil. El Proteus está compuesto por dos elementos fundamentales: ISIS (*Schematic Capture Program*) y ARES (*Advanced Routing and Editing Software*). Acompañan a este producto una gran cantidad de materiales complementarios tales como ayudas, tutoriales y ejemplos que facilitan su aprendizaje y utilización. (Labcenter Electronics, 2004).

Entre las principales posibilidades ofrecidas por ISIS se encuentran:

- Gráficos en pantalla, los cuales son colocados en el diseño como cualquier otro elemento, pueden ser maximizados a pantalla completa para análisis y mediciones.
- Análisis basados en gráficos (*Graph Based Análisis*) tales como transient, frecuencia, ruido, distorsión, componentes AC y DC y transformada de Fourier.
- Soporte directo para componentes análogos en formato SPICE.
- Arquitectura abierta para la inserción de modelos creados en C++ u otros lenguajes. Estos pueden ser eléctricos, gráficos o combinaciones de ambos.
- Generación de listado de componentes a usar en el PBC.

Entre las principales propiedades ofrecidas por ARES se encuentran:

- Una base de datos de alta precisión de 32 bit que aporta una resolución lineal de 10nm, y angular de 0.1° y un tamaño máximo de PBC de 10m.
- ARES soporta 16 placas de cobre, dos pantallas de seda, cuatro placas mecánicas más la de soldadura y la de pegado de las máscaras.
- Listado de componentes basado en la integración con el ISIS.
- Reportes de chequeo de reglas físicas y de conectividad.
- Una poderosa edición de rutas incluyendo opciones de líneas curvas y auto-trazado.
- Trazado 2D con una biblioteca de símbolos.
- Salida a una amplia gama de impresoras y *plotters*. Formatos de salida **.**DXF, .EPS, .WMF y .BMP a teclado o archivo

Después de una amplia busqueda por la red, se llegó a la conclución de que esta herramienta es la más indicada para simular circuitos basados en microcontroladores, ya que posee una amplia gama de diversos modelos y de distintos fabricantes. Además permite la simulación introduciendo puntos de ruptura en el programa que se ejecuta en el

microcontrolador, así como la visualización del código. Es posible también incorporarlo al MPLAP IDE.

CAPÍTULO 2. DISEÑO Y CONSTRUCCION DE HERRAMIENTAS.

Este capítulo estará dedicado a la explicación de cómo se diseñaron y construyeron las partes fundamentales que componen el Kit. Para esto se divide en tres partes. Primero la aplicación que se ejecuta en la PC, la cual se diseñó utilizando LabVIEW. Luego el programa que se ejecuta en el PIC, para el cual se utilizó el ambiente integrado entre el MPLAB IDE y el compilador de C PCHW. Y finalmente los circuitos esquemático e impreso del hardware, lo cual se realizó utilizando las herramientas que brinda el Proteus.

2.1 Funcionamiento del Kit.

La aplicación que se ejecuta en la PC es la encargada de enviar comandos y datos hacia el microcontrolador. El primer byte de cada trama constituye un comando que interpretará el microcontrolador para tomar decisiones, como pueden ser: almacenar la trama en memoria de datos, escribir las tramas almacenadas en memoria de programas, o ejecutar el programa cargado previamente. Además esta aplicación abre el fichero .hex., lo divide en líneas y sustituye el primer byte de cada línea por el comando que indica que esa trama contiene datos y deberá ser guardada. Por otra parte el microcontrolador recibirá estas tramas y chequeará el primer byte para tomar una de las decisiones ante explicadas.

Toda esta comunicación se ejecuta utilizando el protocolo USB. La PC actúa como *host* y el microcontrolador como una función.

2.2 Aplicación Interfaz en la PC.

Esta aplicación requiere del envió y recibo de datos a través del puerto USB.

LabVIEW posee varias funciones implementadas a través del VISA (*Virtual Instruments Software Architecture*) para el manejo de estos dispositivos. (Instrument, 2007, Instrument, 2006a)

VISA *Open*, VISA *Close*, VISA *Read*, VISA *Write*, VISA *Read To File*, VISA *Write From File*, VISA USB *Control In*, VISA USB *Control Out*, VISA *Get* USB *Interrupt Data*, etc.

Para el desarrollo del *software* lo primero será abrir la sesión (VISA *Open*) después ejecutar todas las acciones que se deseen y al final es necesario cerrar la sesión. Para esto la parte principal estará en una estructura de eventos donde inhabilitando los demás botones mediante un nodo de propiedad (*property node*) solo se podrá ejecutar la selección del dispositivo y *open* VISA.

Después se podrá ejecutar todos los eventos deseados y al final se ejecutará *close* VISA. Todo esto estará dentro de un ciclo *while* el cual mantendrá la continuidad de todos los parámetros, hasta que se oprima la tecla stop o haya algún error en el programa.

En la figura 2.1 se muestra el caso donde se abre la sesión (*open* VISA)

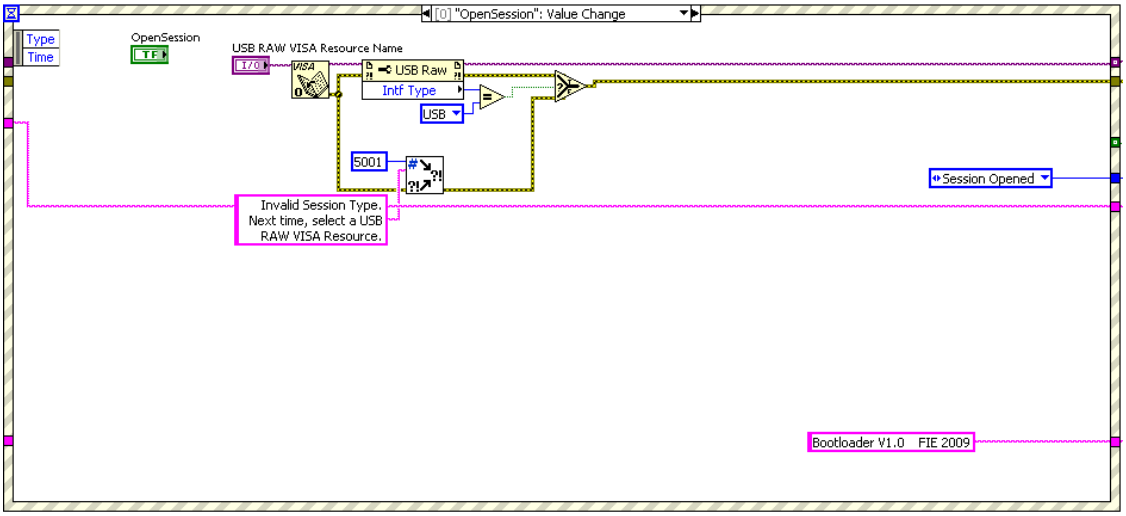

Figura 2.1 Abrir sesión.

En el siguiente caso (fig. 2.2) se envía una trama cuyo primer y único byte tiene el valor 0x01.Esto tiene como objetivo que el microcontrolador entre en el programa cargador a la espera de futuros comandos.

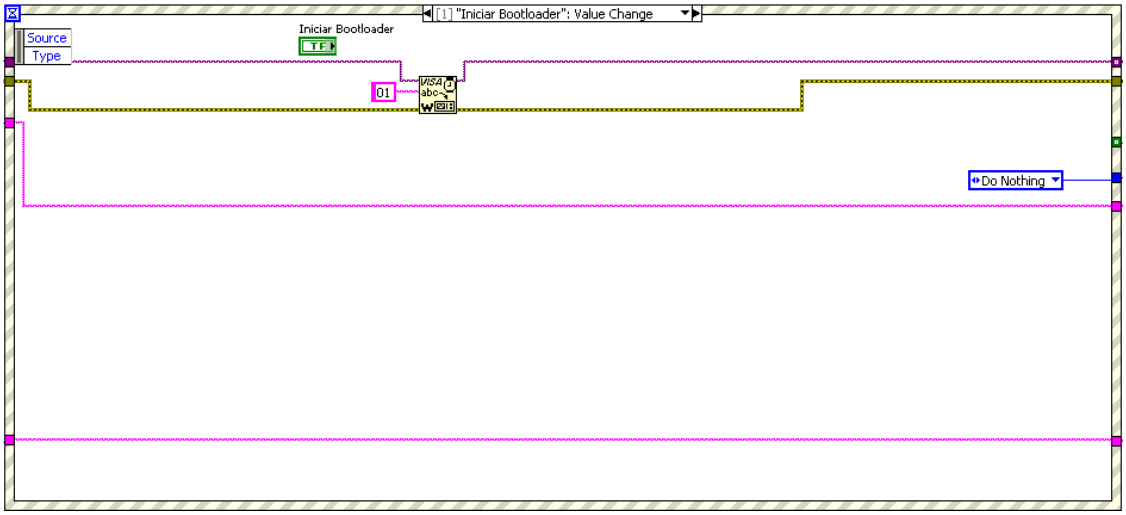

Figura 2.2 Iniciar el programa Cargador.

En el siguiente caso se abre el fichero .hex. (fig.2.3) y se muestra en el campo *Intel Hexadecimal File* del panel frontal de la aplicación.

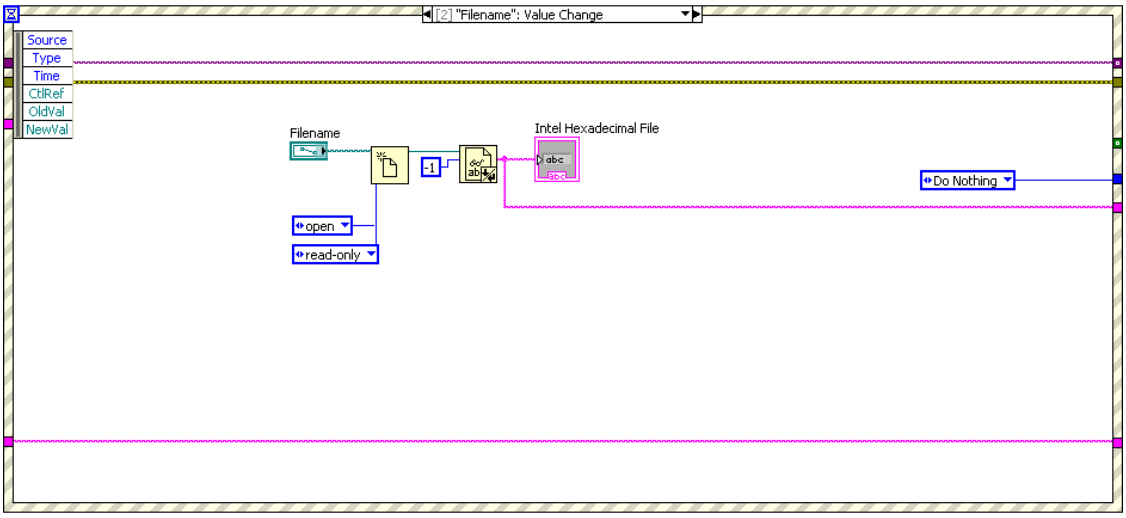

Figura 2.3 Abrir Fichero.

En el siguiente caso (fig. 2.4) se divide el fichero .HEX en líneas. Cada línea va a conformar una trama que será enviada. El primer byte de cada línea el cual tiene el valor 0x3A (código ASCII del símbolo ´:´), se cambiará por el número 0x02, el cual indica que la trama contiene datos útiles, y que una vez recibida en el microcontrolador, deberá ser guardada en memoria de datos para posteriormente escribirla en la memoria de programas. Dentro del microcontrolador existe un arreglo el cual puede almacenar 30

tramas de datos, por lo cual cuando se haya llenado el mismo se espera por la respuesta que indique que no ha ocurrido ningún error escribiendo estas tramas en la memoria de programas y que se puede seguir enviando el fichero .HEX. De esta forma el arreglo mencionado actuará como un *buffer*. Después del envío del la última trama, se enviará una trama cuyo primer y único byte tiene el valor 0x03, el cual indica que las tramas almacenadas deberán ser escritas en memoria de programas. En cada caso se chequea la ocurrencia de errores y se indica en el panel frontal de la aplicación. Las causas de los posibles errores se explicarán más adelante.

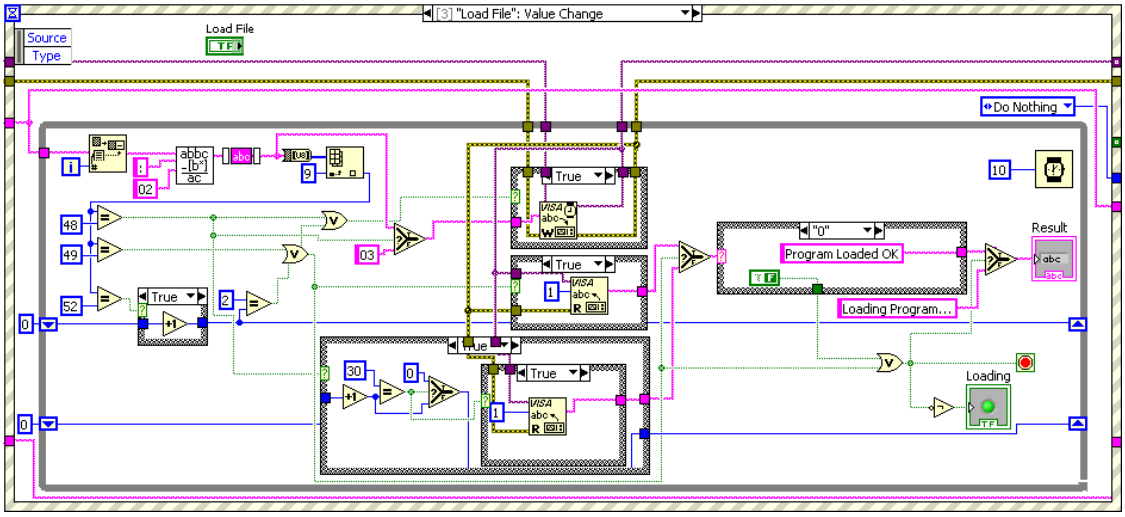

Figura 2.4 Cargar archivo en el microcontrolador.

En el siguiente caso (fig. 2.5), se envía una trama, la cual también cuenta con un solo byte. Este byte cuyo valor es 0x04, le indica al microcontrolador, que saldrá del programa cargador, y ejecutará el programa previamente recibido y cargado en su memoria de programas.

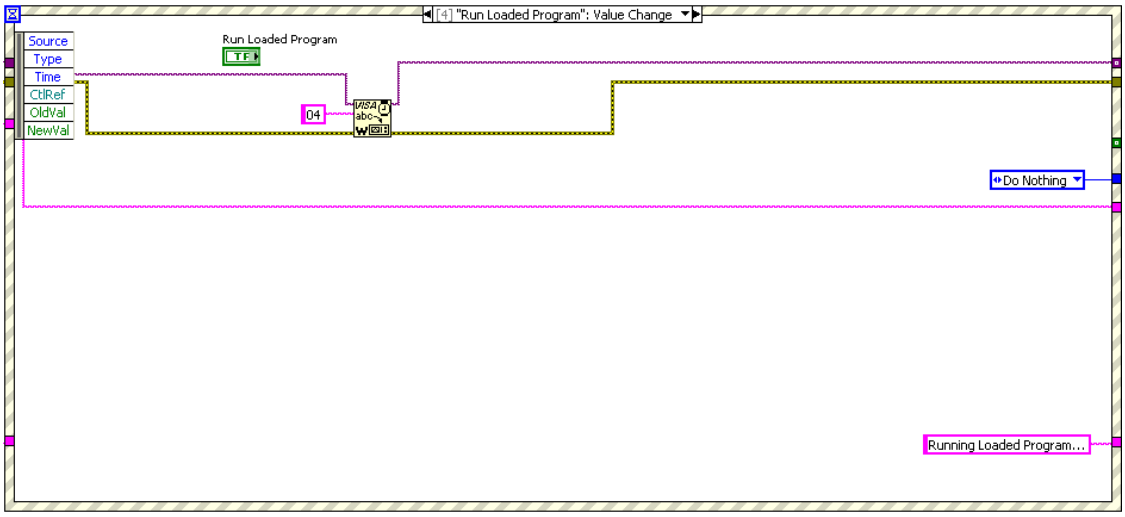

Figura 2.5. Correr programa cargado.

Al finalizar la comunicación es necesario utilizar la función VISA *close* (fig. 2.6) para que la aplicación libere al dispositivo permitiendo que sea usado por otro *software*.

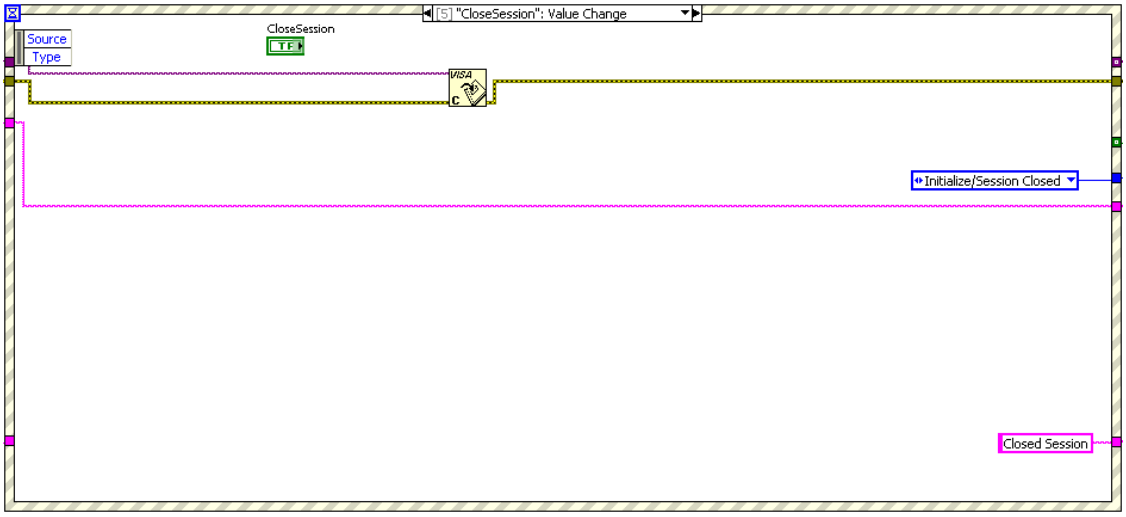

Figura 2.6 Cerrar sesión.

En el último caso, se detiene la ejecución del programa. Además se utiliza la función VISA *close* con el mismo objetivo que en la sesión anterior (fig. 2.7).

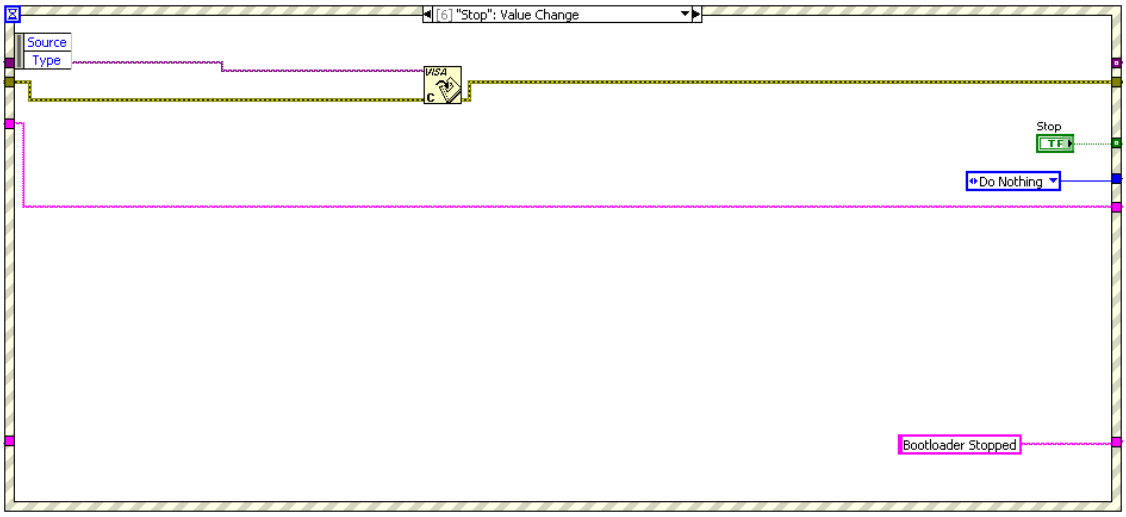

Figura 2.7 Detener el programa.

La estructura *case* (fig. 2.8) es la encargada de habilitar o no a los diferentes botones mediante los nodos de propiedad para que no existan violaciones en los pasos de la comunicación.

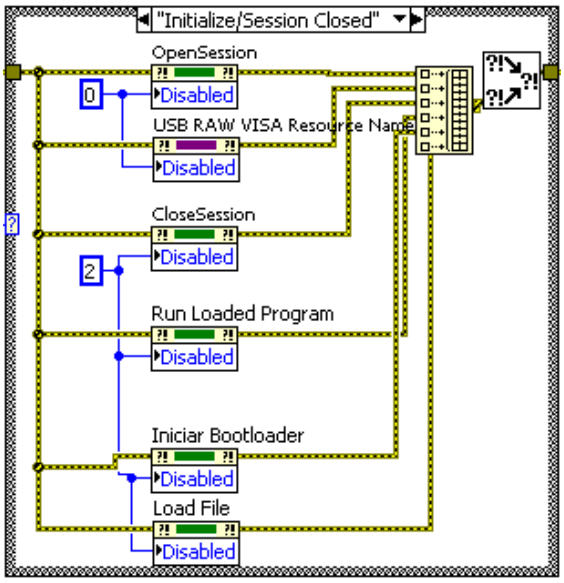

Figura 2.8 Estructura *case* con los *property nodes*.

Cuando se tiene el instrumento virtual funcionando correctamente y salvado. Es necesario conocer como crear un instalador y un ejecutable.

Lo primero es crear un nuevo proyecto. Dentro del proyecto se oprime clic derecho en *My Computer* y se selecciona adicionar fichero (*.VI).

Después en *build specification* se puede crear una aplicación, un instalador, una librería compartida, etc. En cada caso se puede especificar todos sus campos. Si todo se hizo correctamente debe aparecer como en la figura 2.9 (Uvais, 2006).

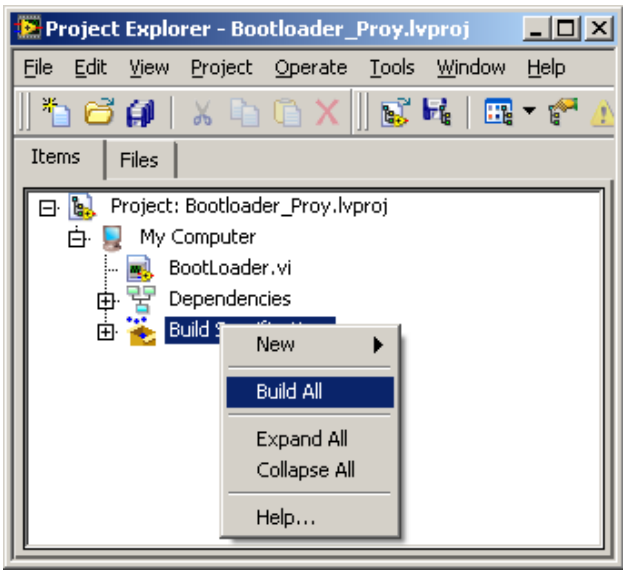

Figura 2.9 Proyecto en LabVIEW.

2.3 Edición del *Driver* USB.

En el momento que se tienen las aplicaciones, en el PIC y en la PC solo queda la edición del *driver* para que el software tenga acceso al dispositivo USB.

Éste lo proveerá el VISA y en particular su herramienta para la edición de *drivers* (*Driver Development Wizard*). Este *software* se instala junto con el LabVIEW y se puede ejecutar en *start\nacional instrument\VISA\VISA Driver Development Wizard.*

Al ejecutarlo se optará por la edición del *driver* USB, entonces saldrá la ventana mostrada en la figura 2.10 en la cual debemos llenar los campos pedidos. El identificador del vendedor (VID) y el identificador del producto (PID) son números de 2 bytes cada uno que identifican el dispositivo. El nombre del productor y del modelo son cadenas que nos darán información sobre éste. Estos datos están también localizados en el *Device Descriptor* de la función (el microcontrolador).

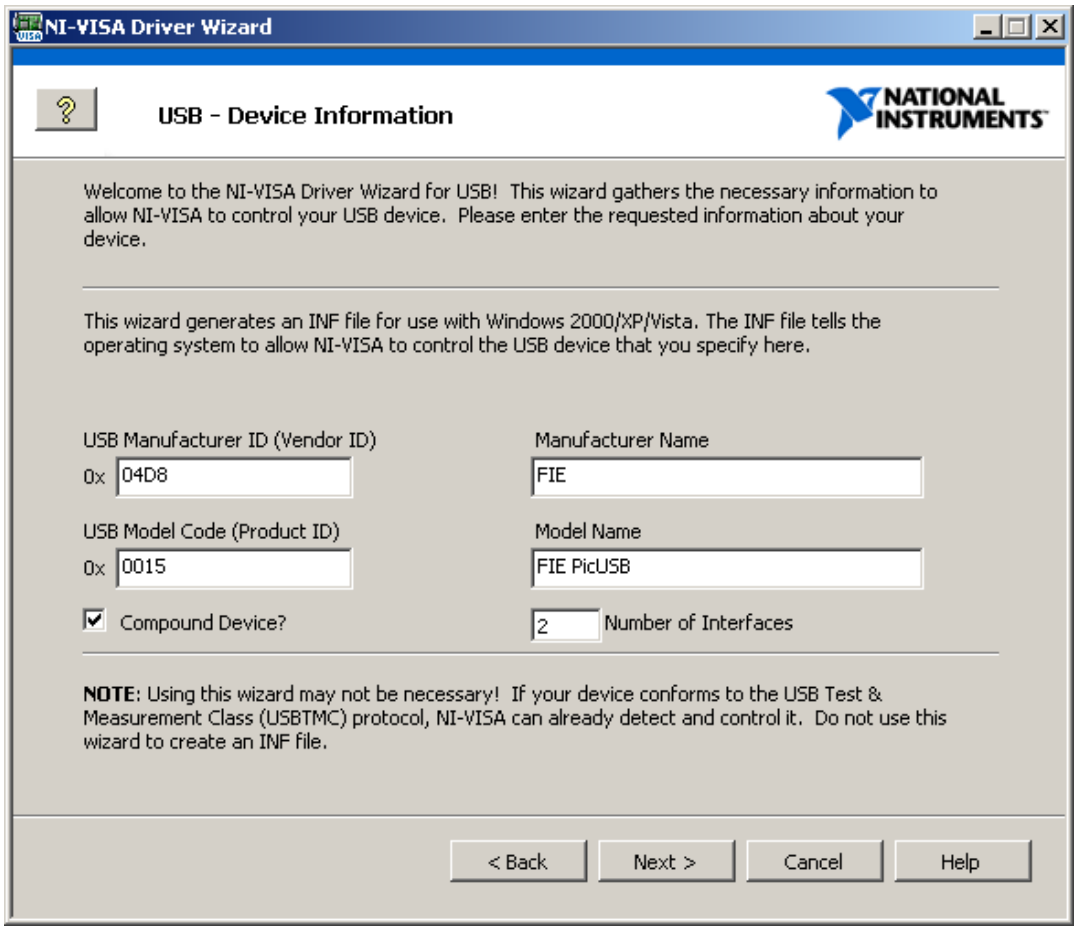

Figura 2.10 Edición del *driver*.

Con los campos correctamente llenados entonces se terminará la edición del *driver*, el cual permitirá el acceso del LabVIEW al dispositivo por USB (Instrument, 2008).

Ahora el PIC estará listo para ser conectado a la PC por USB. Cuando el Sistema Operativo lo detecte recibirá su VID&PID y buscará entre sus *drivers* instalados para encontrar el que corresponde a esa identificación, si no la encuentra preguntará sobre donde ha de buscar un *driver* adecuado y entonces se le indicará su ubicación, como se ve en la figura 2.11 (Peacock, 2007b, Márquez, 2007).

Figura 2.11 Asignación del *driver*.

En el siguiente paso se selecciona instalar desde una lista o ubicación específica para después darle la ubicación de la carpeta donde fue creado previamente por el VISA *Driver Development Wizard*. Y si todo ha ido correctamente *Windows* debe reconocer el dispositivo, como se observa en la figura 2.12.

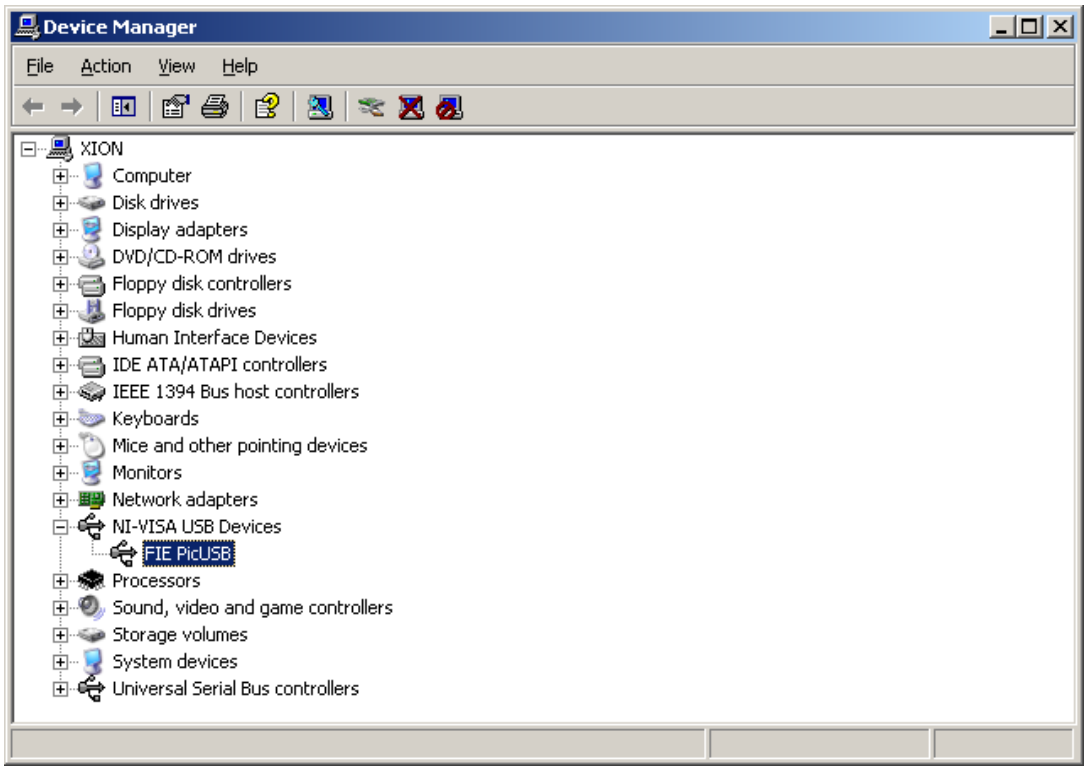

Figura 2.12 Dispositivo reconocido por *Windows*.

En este momento estará listo para recibir y trasmitir información por USB. La aplicación correctamente funcionando debe quedar como se muestra en la figura 2.13.

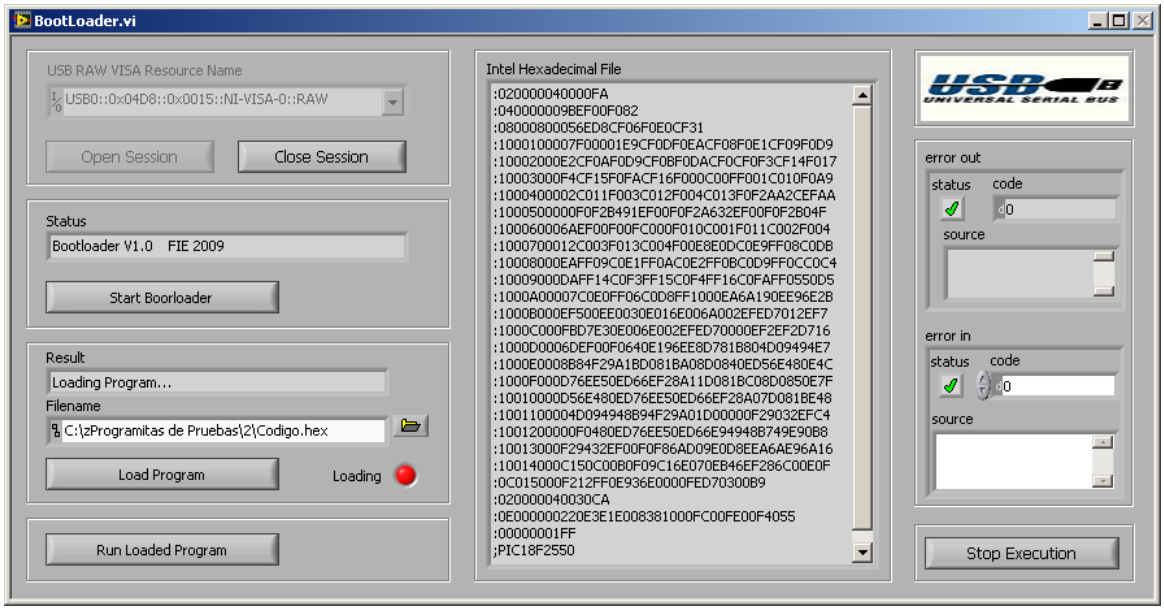

Figura 2.13 Aplicación programada en LabVIEW.

2.4 Programa Cargador.

El programa cargador está compuesto por tres partes fundamentales. Una maneja la comunicación USB, otra que se encarga de interpretar lo comandos y tomar las acciones correspondientes y la última maneja los vectores de interrupción.

2.4.1 Comunicación USB en el microcontrolador.

Para la comunicación USB se emplearon las funciones que brinda el compilador de C PCHW. De estas se utilizaron las siguientes:

- *usb_init()* inicializa el módulo USB.
- *usb_task()* habilita el periférico USB y la interrupción asociada.
- *usb_wait_for_enumeration()* espera que el microcontrolador sea configurado por el *host*.
- *usb_enumerated()* retorna *TRUE* si el microcontrolador ha sido configurado correctamente.
- *usb_kbhit(endpoint)* retorna *TRUE* si el *endpoint* especificado contiene datos del *host* en su *buffer* de recepción.
- *usb_get_packet(endpoint, ptr, max)* lee un paquete de datos desde el *endpoint* especificado. Devuelve el número de *bytes* recibidos.
- *usb_put_packet(endpoint, data, len, tgl)* pone el paquete de datos en el *buffer* del *endpoint* especificado. Devuelve *TRUE* si la operación se completó correctamente y *FALSE* si el buffer está aun lleno con datos del paquete anterior.

Además de utilizar estas funciones (Anexo I) se definieron los descriptores en el fichero cabecera *PicUSB.h* (Anexo II), y se incluyó el fichero que brinda la Microchip *pic18_usb.h*, para trabajar con el módulo USB del PIC

El diagrama de flujo relacionado a la comunicación USB se muestra en la figura 2.14

Figura 2.14 Comunicación USB.

2.4.2 Intérprete de Comandos.

Las tramas son enviadas desde la PC por la aplicación interfaz. Una vez recibida la trama mediante las funciones de manejo del módulo USB en el microcontrolador, el primer *byte* de la misma constituye el comando a interpretar. El intérprete de comandos no es más que una estructura *switch* (Anexo I) la cual se pregunta por dicho primer *byte*, cuyo diagrama de flujo se muestra en la figura 2.15. En dependencia del resultado se realizan diferentes acciones la cuales son:

- Almacenar trama recibida en la memoria de datos (MD).
- Escribir las tramas previamente guardadas en memoria de datos hacia la memoria de programas (MP).
- Chequear la ocurrencia de errores.
- Responder indicando que se ha escrito ya en la memoria de programas y si ha o no ocurrido algún error.

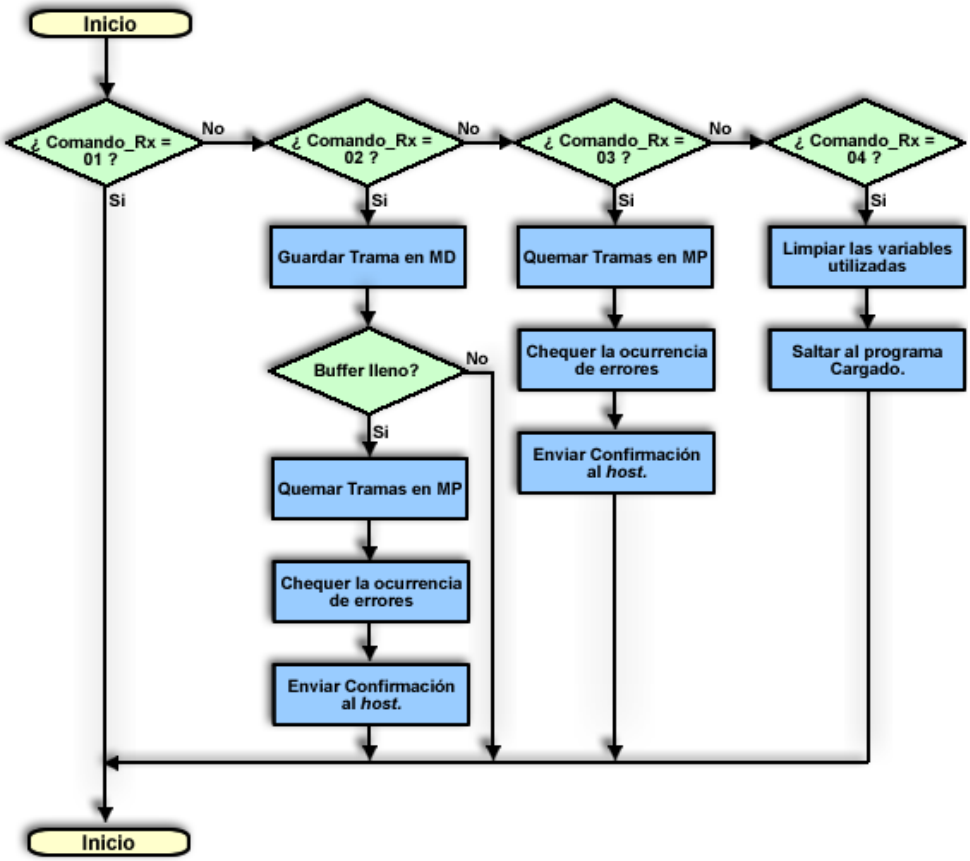

Figura 2.15 Interprete de comandos.

2.4.3 Tratamiento de las Interrupciones.

Para la vectorización de las interrupciones, se utilizó la función *default_isr ()* precedida por la directiva de preprocesador *#int_default* (Anexo I). Esta función será llamada cada vez que ocurra una interrupción que no tenga asignada ninguna función para su atención. Por lo tanto será llamada cada vez que ocurra una interrupción mientras esta no sea provocada por el módulo USB, el cual si se utiliza en el programa cargador.

Una vez dentro de esta función se determina que interrupción es la que ha ocurrido y que nivel de prioridad tiene, para luego saltar al vector de interrupciones de la aplicación. El diagrama de la memoria de programas se muestra en la figura 2.16

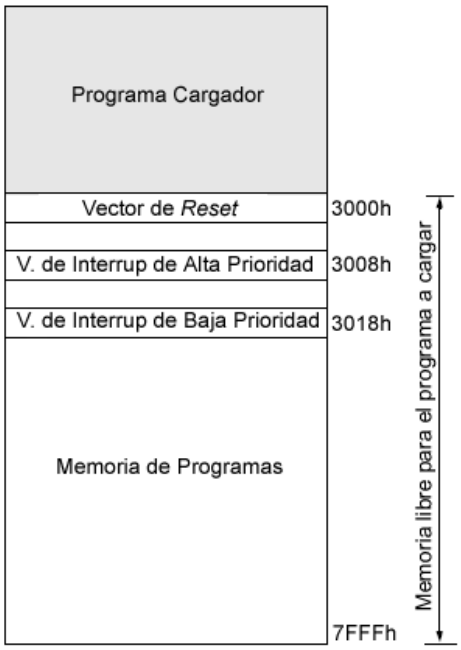

Figura 2.16 Estructura de la memoria de programas.

2.4.4 Escritura en la Memoria de Programas.

Uno de los aspectos del programa cargador es que sea transparente para el usuario, por lo que un programa el cual ocupe las primeras direcciones de la memoria de programas deberá ser cargado sin problemas.

Para resolver esto se realizan algunas operaciones antes de proceder a escribir el programa. Primero se busca el código de las instrucciones de salto absoluto dentro de las tramas recibidas, las cuales son producto del fichero .HEX. Luego se le añade un desplazamiento equivalente al tamaño del programa cargador a la parte de este código que indica hacia donde se va a saltar. Y después se le añade un desplazamiento a las direcciones que aparecen también en las tramas recibidas.

Producto de estos desplazamientos es posible que aparezcan los siguientes errores:

- Una instrucción de salto absoluto intenta saltar a una dirección de la memoria de programas superior a la 7FFFh.
- Se intenta escribir en una dirección de la memoria de programas posterior a la 7FFFh.

Para evitar estos errores, el programa a cargar deberá estar contenido en los limites 0000h a 4FFFh. De todas formas, el programa cargador envía un código asociado al tipo de error, de forma tal que dicho error sea mostrado por la aplicación interfaz.

Además se chequea por si aparece un error producto de la transmisión, a través de la suma de chequeo que está contenida en el fichero .HEX. Este error también se le informa al usuario (Anexo I).

Finalmente se escribe la memoria de programas utilizando la función que incluye el compilador de C PCHW, *write_program_memory (address, dataptr, count*). Esta función recibe como parámetros:

- Dirección donde se va a escribir la trama.
- Puntero a los datos a escribir
- Cantidad de *bytes* a escribir.

2.5 Circuitos Esquemático e Impreso.

Tanto el circuito esquemático como el Impreso se realizaron utilizando las herramientas de Proteus ISIS y ARES respectivamente.

2.5.1 Circuito Esquemático.

Para la realización de este circuito (fig. 2.17) se tuvo en cuenta la utilidad de los botones y de un *display* de cristal líquido para aplicaciones generales. Además se incluye un LED para indicar la alimentación y otro que puede ser utilizado como propósito general. Dejando libre los terminales correspondientes al conversor análogo digital y a las interrupciones externas.

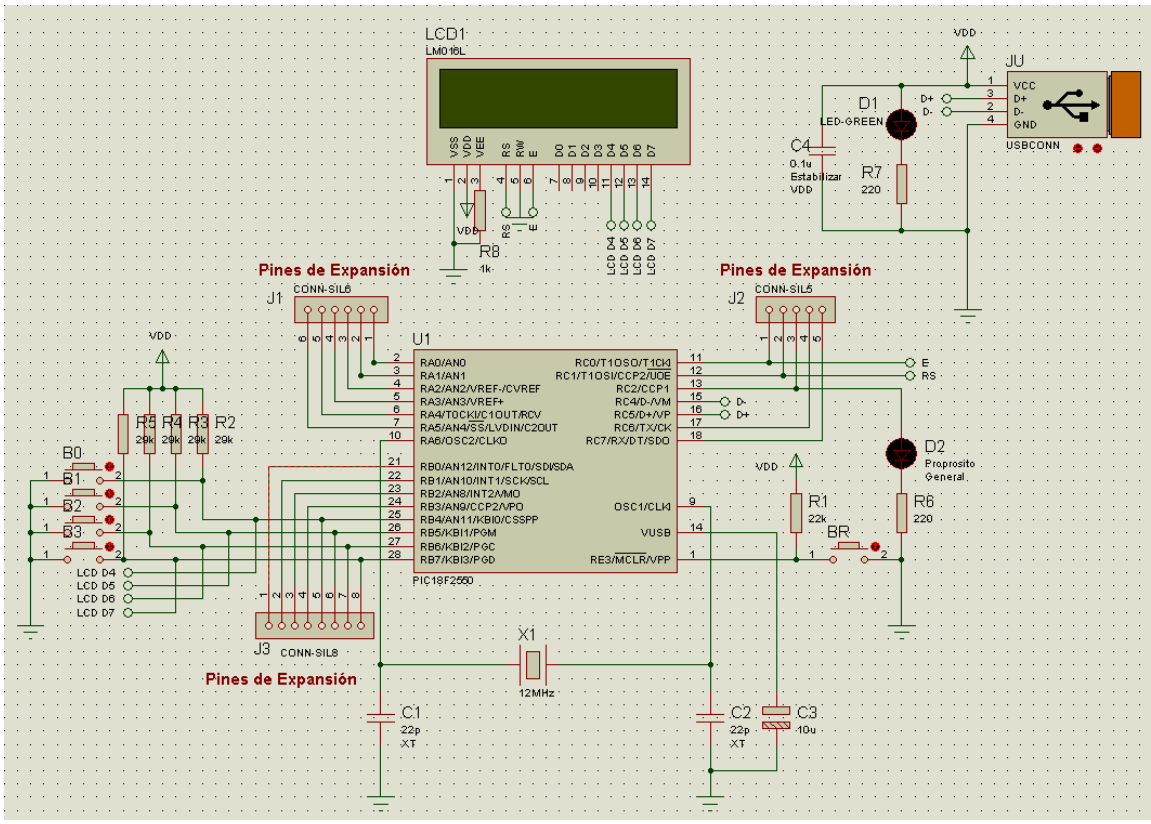

Figura 2.17 Circuito esquemático.

2.5.2 Circuito Impreso.

El circuito impreso (fig. 2.18) se diseñó teniendo en cuenta que los botones de propósito general quedaran al frente, así como que el botón de *reset* quedara cómodamente accesible. Otro aspecto es que el *display* ocupa un área sobre la placa y con la parte inferior de las letras hacia el usuario. Además los pines de expansión al igual que el conector USB quedaron perfectamente manipulables.

La figura 2.19 muestra una vista previa que brinda el ARES de cómo quedaría el circuito una vez terminado y con todos los componentes montados. En la figura 2.20 se muestra una imagen del circuito real, donde se puede apreciar que no difiere mucho de la vista previa.

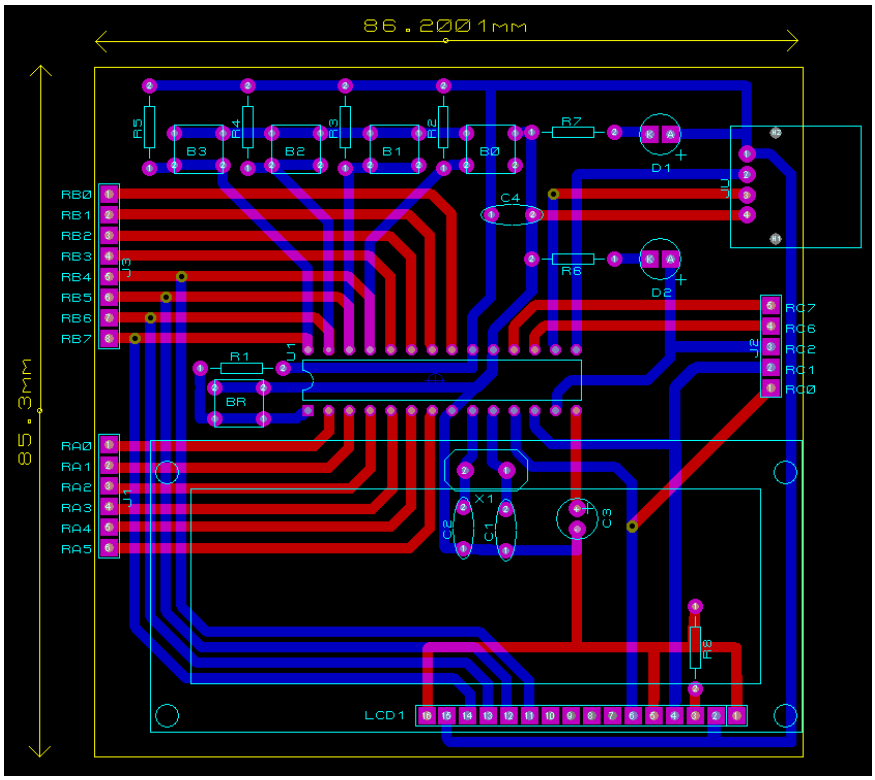

Figura 2.18 Circuito impreso.

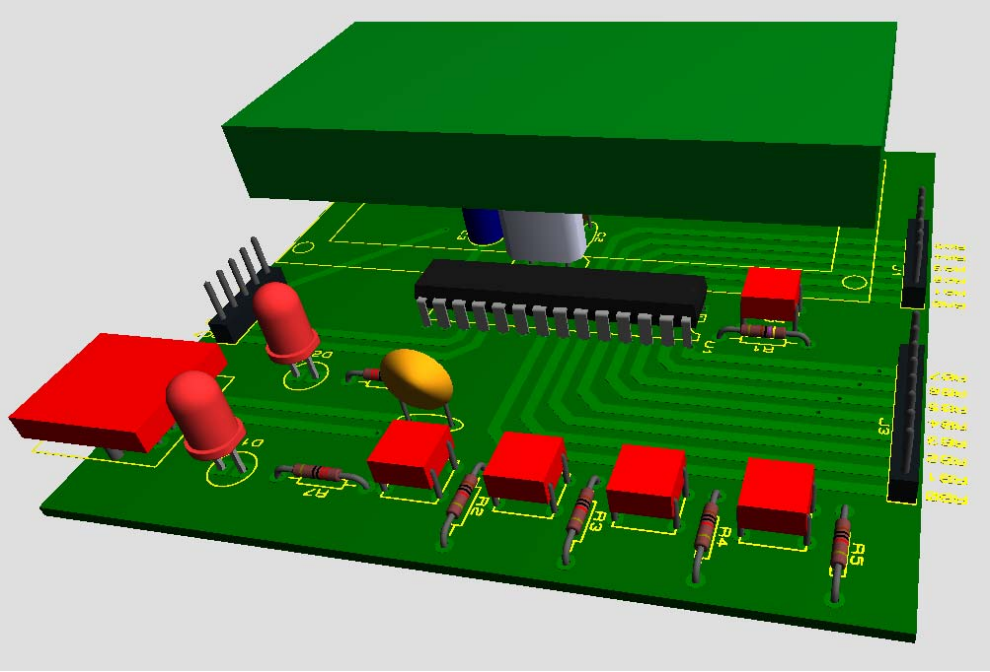

Figura 2.19 Imagen en 3D del circuito Impreso.

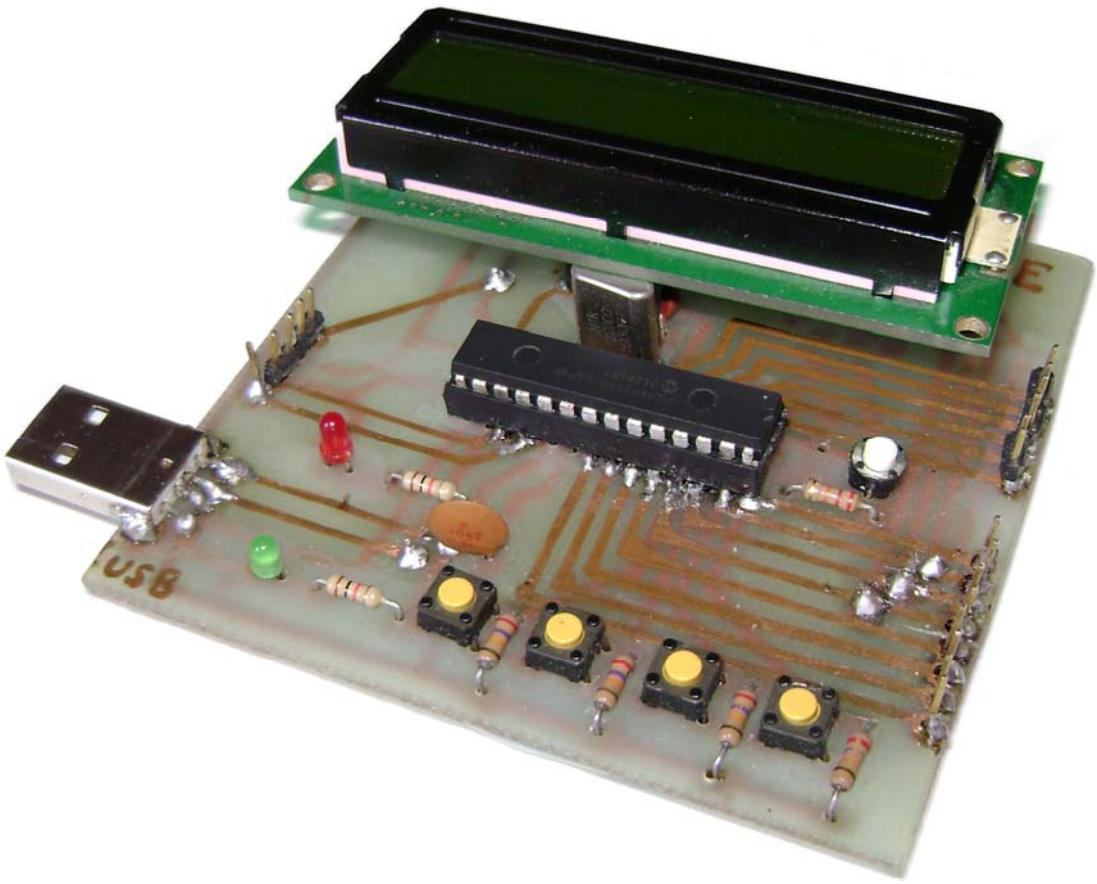

Figura 2.20 Circuito Real.

CAPÍTULO 3. DESARROLLO DE APLICACIONES.

En el presente capítulo se incluye un manual de usuario, se hace un análisis de los resultados alcanzados, realizándose comparaciones con los objetivos planteados, con otros trabajos similares realizados en el centro y finalmente se prueba la funcionalidad del Kit cargándole una aplicación.

3.1 Manual de Usuario

El trabajo con el Kit se maneja desde la aplicación interfaz que se ejecuta en la PC. A partir de este punto se describe como se deberá ser utilizado el Kit. La aplicación interfaz una vez que comienza a ejecutarse se muestra como en la figura 3.1

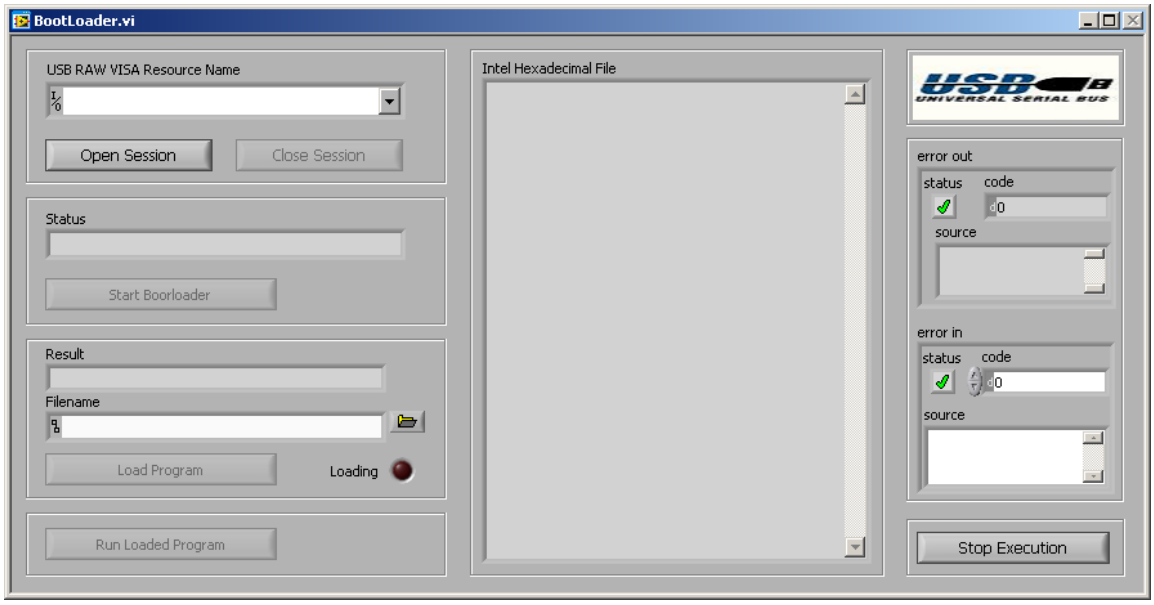

Fig. 3.1 Interfaz gráfica

A partir de este punto se deberán seguir los siguientes pasos:

1. Una vez conectado el dispositivo, se seleccionará mediante los números de sus descriptores (fig.3.2) a través del recuadro *USB RAW VISA Recursce Name*. Luego se hace clic en el botón "*Open Session*" con lo que la aplicación responde habilitando una serie de botones que estaban deshabilitados y colocando un mensaje en el campo *Status*.

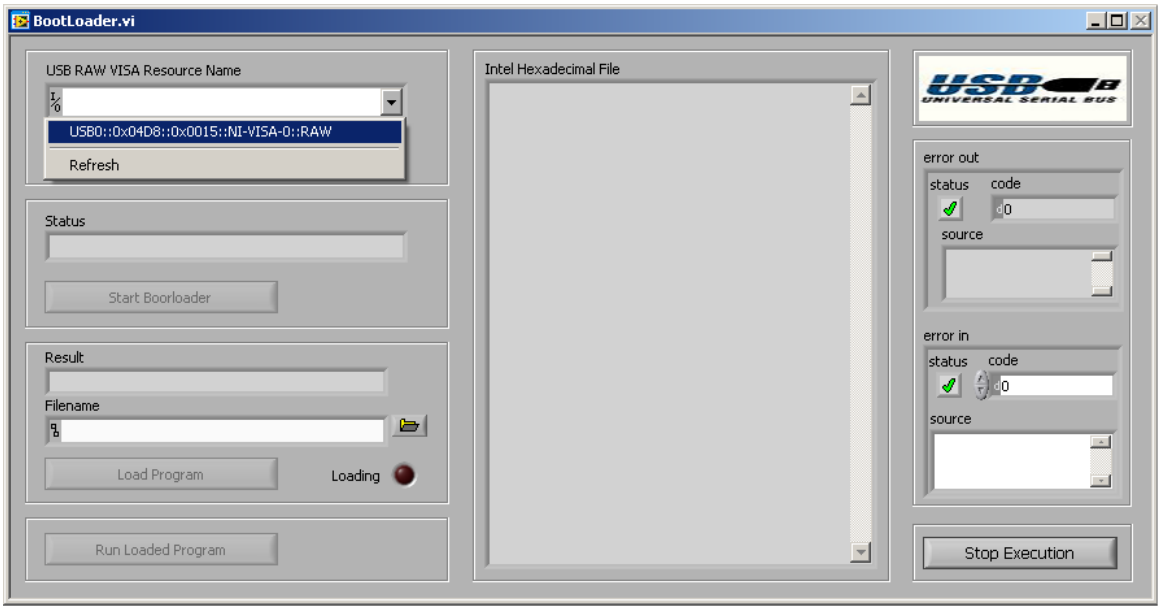

Figura 3.2 Buscar el dispositivo.

2. Luego se entra al programa cargador a través del botón "*Star Bootloader*" (fig. 3.3).

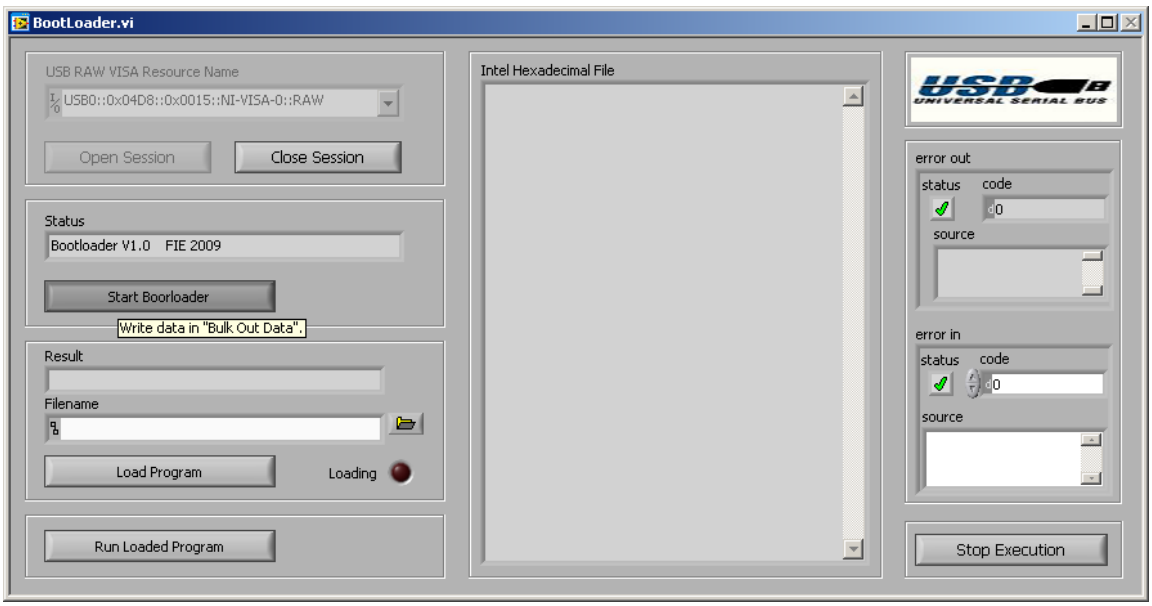

Figura 3.3 Inicializar el programa Cargador.

3. Luego se localiza la dirección del fichero .HEX generado por la aplicación que se desea cargar en el microcontrolador a través del campo "*Filename*". El fichero abierto será mostrado en el campo "*Intel Hexadecimal File*" (fig. 3.4).

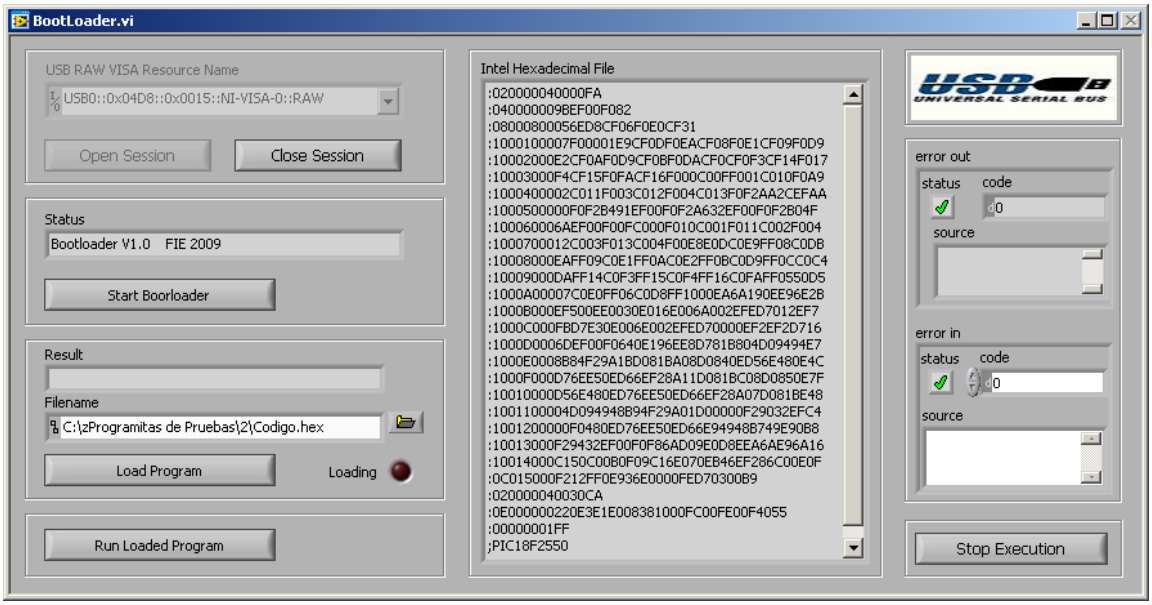

Figura 3.4 Fichero Intel Hexadecimal cargado.

4. Una vez el fichero a cargar haya sido mostrado, para cargarlo se hace clic en el botón "*Load Program*". Mientras el fichero se esté cargando, aparecerá un mensaje correspondiente en el campo "*Result*" y estará encendido el LED del panel frontal y el LED rojo montado en la placa del circuito impreso (fig. 3.5). Una vez este proceso haya terminado, en el campo "*Result*" se indicará si todo ha salido correctamente o si ha ocurrido algún error mientras se cargaba el programa. En caso de ocurrir algún error este se especifica.

   Posibles mensajes:

   - *Program Loaded OK*, indica que el proceso se completó correctamente.
   - *ERROR 1: Salto Fuera de la Memoria*, indica que ocurrió un error debido a que una instrucción de salto absoluto intenta saltar fuera del área de memoria de programas permitida (0000h – 4FFFh).

- *ERROR 2: Programa muy largo*, indica que el programa cargado ocupa posiciones en la memoria de programas fuera del rango permitido (0000h – 4FFFh).

- *ERROR 3: Suma de Chequeo*, indica que ocurrió algún cambio en los datos durante la transmisión.

En cada caso en que ocurra un error deberá cargarse el programa nuevamente en el microcontrolador.

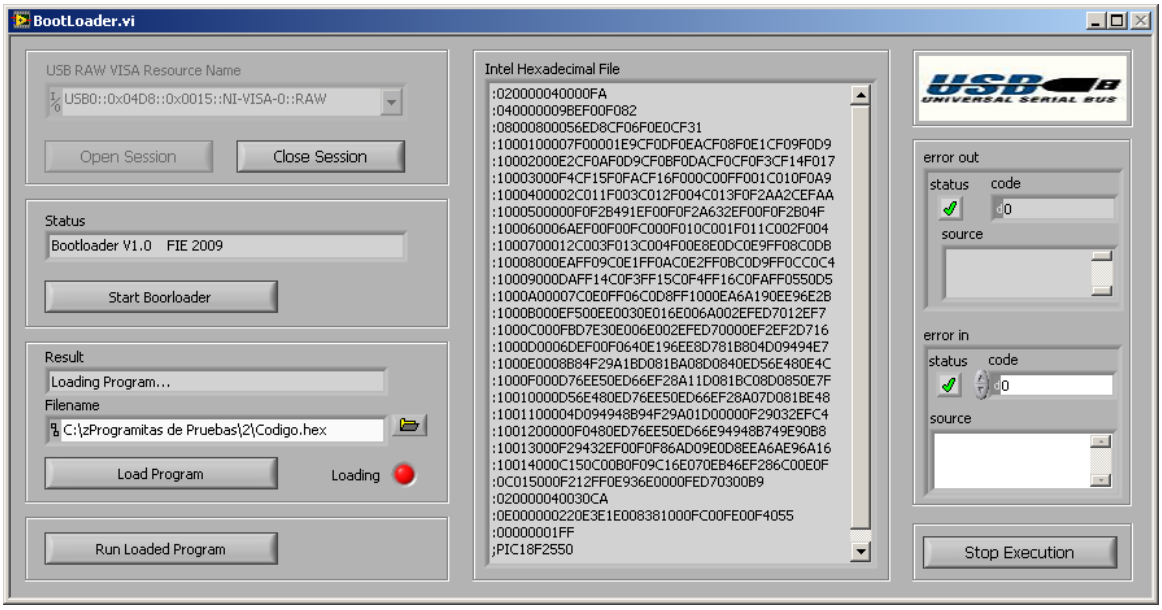

Figura 3.5. Cargando programa en el microcontrolador.

5. Una vez ejecutados los pasos anteriores todo queda listo para correr la aplicación que se desea a través del botón "*Run Loaded Program*". Es válida la aclaración de que se puede correr un programa que haya sido previamente cargado en el microcontrolador a través de este botón directamente.

6. Como último paso es necesario liberar el dispositivo, lo cual se hace con el botón "*Close Seccion*", y si además se desea detener la aplicación interfaz, esto se hace con el botón de "*Stop*". En cada caso se muestra un mensaje indicando el estado del sistema en el campo "*Status*". Si se desea repetir todo el proceso no es necesario desconectar el dispositivo. Para esto habrá que reiniciar el microcontrolador mediante el botón blanco que se muestra en la figura 2.19, luego se comenzará de nuevo por el primer paso.

3.2 Resultados Alcanzados.

Para probar los resultados alcanzados se realizó un programa teniendo en cuenta el diseño del circuito esquemático del Kit. El programa consiste en una toma de decisiones generada por las interrupciones externas al accionar uno u otro botón de los 4 conectados al Puerto B del microcontrolador. La toma de decisiones consiste en encender el LED conectado en el pin C3 de forma permanente en caso de apretar el botón conectado al in B4, en otros dos casos que encienda y apague a distintas frecuencias a través de los botones conectados a los pines B5 y B6 y en el otro caso que se apague el LED apretando el botón conectado al pin B7. El apagado y encendido del LED a distintas frecuencias se logra mediante el *TIMER0*.

Con esta aplicación se comprobó el correcto funcionamiento de la re-vectorización de las interrupciones, ya que se generan dos tipos de interrupciones, la externa y la del temporizador 0, las cuales a su vez se le asignó diferentes niveles de prioridad. Además de esto se puedo ver como el programa fue cargado íntegramente a partir de la posición 3000h en memoria de programas, con lo cual comprobó también el cambió que se le hizo a las instrucciones de salto absoluto antes de que su código fuera cargado.

3.3 Prestaciones del Kit.

El Kit permite que una aplicación que haya sido diseñada sin tenerlo en cuenta pueda funcionar en el siempre y cuando no influya la diferencia entre la palabra de configuración del programa a cargar y la del programa cargador (Anexo I) y el rango de direcciones de la memoria de programas utilizada. Además el proceso de carga es muy sencillo y rápido en comparación a otros cargadores previamente diseñados los cuales se comunicaban mediante el puerto serie de la PC. Este proceso es extremadamente seguro, y permite en caso de que ocurra algún error que el usuario tenga conocimiento del mismo. Todo esto lo convierte en una potente herramienta de prueba de programas en tiempo real.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1 Se logró el diseño y construcción de varias herramientas las cuales conforman un Kit basado en el microcontrolador 18F2550, el cual permite la corrida en tiempo real de programas.

2 Se cargaron diversos programas en al Kit, demostrando que la utilización del protocolo USB lo hace muy seguro, rápido y de fácil utilización.

Recomendaciones.

1 Continuar el desarrollo de herramientas más potentes como emuladores a partir de la base que brinda este trabajo.

2 Introducir en la docencia la utilización de este Kit.

REFERENCIAS BIBLIOGRÁFICAS

Aguirre, E. G. (2005) Firmware para dispositivo esclavo USB de Clase HID. San Francisco. [En línea]. Disponible en: http://www.cneisi.frc.utn.edu.ar/papers/3642e51321ca65a8eeeaad6a302e.pdf [Consultado el 2 de abril de 2009]

Anderson, D. (2001) *USB System Architecture (USB 2.0).*

Axelson, J. (2001) *USB Complete*. [En línea] Disponible en: http://www.lvr.com [Consultado el 16 de abril de 2009]

Custom Computer Services, INC. (2007). *C Compiler Reference Manual*. Brookfield, WI, E.U.

Delgado, D. (2008). Aplicación de comunicación vía USB del microcontrolador PIC con la PC. Trabajo de Diploma . Santa Clara,Facultad de Ingeniería Eléctrica, Universidad Central "Marta Abreu" de las Villas.

Domínguez, Y. (2008). Kit y cargador para microcontroladores PIC. Trabajo de Diploma . Santa Clara,Facultad de Ingeniería Eléctrica, Universidad Central "Marta Abreu" de las Villas

Fujitsu (2003) Un paseo por USB 2.0. [En línea] Disponible en: http://www.fujitsu.com/downloads/EU/es/soporte/discosduros/UnpaseoporUSB-2.pdf [Consultado el 4 de mayo de 2009]

Implementers, U. F. (2003) Universal Serial Bus Mass Storage Class Specification Overview. Revisión 1.2 ed., USB Implementers Forum. [En línea] Disponible en: http://www.usb.org/developers/devclass_docs [Consultado el 19 de mayo de 2009]

Instrument, N. (2006) USB Instrument Control Tutorial. [En línea]. National Instruments, disponible en:

http://zone.ni.com/devzone/cda/tut/p/id/4478 [Consultado el 4 de mayo de 2009]

Instrument, N. (2007) How to control USB modules using the LabVIEW package. [En línea] Disponible en:

http://www.ar.com.au/~softmark/page53.html [Consultado el 21 de abril de 2009]

Instrument, N. (2008) VPP-4.3.3: VISA Implementation Specification for the G Language. Revisión 4.1 ed. [En línea] Disponible en: http://www.ivifoundation.org/docs/vpp433.doc [Consultado el 23 de abril de 2009]

Márquez, D. (2007) EL USB DESENCADENADO: BULK USB. [En línea] Disponible en: http://picmania.garcia-cuervo.com [Consultado el 4 de mayo de 2009]

Labcenter Electronics. (2004). *About PROTEUS Demo, PROTEUS VSM CO-SIMULATION*. Disponible en: http://www.labcenter.co.uk, consultado en marzo de 2008.

Microchip Technology Inc. (2002). A FLASH Bootloader for PIC16 and PIC18 Devices. Disponible en: http://www.microchip.com, consultado en febrero de 2009.

Microchip Technology Inc. (2006c). PIC18FXX2 Data Sheet. Disponible en: http://www.microchip.com, consultado en febrero de 2008.

Moreno, A. (2008). Diseño avanzado de aplicaciones con PICs. Trabajo de Diploma . Santa Clara,Facultad de Ingeniería Eléctrica, Universidad Central "Marta Abreu" de las Villas.

Mueller, S. (2006) *Upgrading and Repairing PICs*. NEC, C. I. M. (1998) Universal Serial Bus Specification. Revision 1.1 ed., Compaq Intel Microsoft NEC

Peacock, C. (2007) USB in a NutShell Making sense of the USB standard [En línea] Disponible en: http://www.beyondlogic.org/usbnutshell/usb1.htm [Consultado el 15 de abril de 2009]

Philips, C. H.-P. I. L. M. N. (2000) Universal Serial Bus Specification. Revisión 2.0, Compaq Hewlett-Packard Intel Lucent Microsoft NEC Philips. [En línea] Disponible en: http:// www.usb.org/developers/docs/ [Consultado el 15 de abril de 2009]

Roca, R. ET AL., (2008). "Conversión de Ficheros a Intel Hex. PIC 16F84" [En línea]. Santa Clara, Disponible en: \\neumann\Asignaturas\Telecom y Electronica\Microprocesadores2\CURSO06-07\tareaextraclase\tareas\Conversion Intel Hex. [Consultado el 9 de marzo de 2009]

Rosch, W. L. (2003) Hardware Bible.6th ed

UVAIS, A. (2006) LabVIEW & Instrumentation Training. Doha. [En línea] Disponible en: http://www.unicrom.com/link.asp?TOPIC_ID=5337&view=lasttopic [Consultado el 4 de mayo de 2009]

ANEXOS

Anexo I Código Fuente del Programa Cargador (PicUSB.c).

```
#include <18F2550.h>
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL3,CPUDIV1,VREGEN
#use delay(clock=12000000)

#define USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK transfers
#define USB_EP1_TX_SIZE    64
#define USB_EP1_RX_SIZE    64

#include <pic18_usb.h>
#include <PicUSB.h>
#include <usb.c>
#include <LCDM.h>

#define     LOADER          0x3000
#define     TRAMA_LONG      22
#define     TRAMA_STORE     30

#define     PIE1        0xF9D
#define     PIR1        0xF9E
#define     IPR1        0xF9F
#define     PIE2        0xFA0
#define     PIR2        0xFA1
#define     IPR2        0xFA2
#define     RCON        0xFD0
#define     INTCON3     0xFF0
#define     INTCON2     0xFF1
#define     INTCON      0xFF2

#define     IPEN    7
#define     TMR0IP      2
#define     TMR1IP      0
#define     TMR2IP      1
#define     TMR3IP      1
#define     INT1IP 6
#define     INT2IP 7
```

```
#define	RBIP	0
#define	ADIP	6
#define	RCIP	5
#define	TXIP	4
#define	SSPIP	3
#define	CCP1IP	2
#define	CCP2IP	0
#define	BCLIP	3
#define	LVDIP	2
#define	CMIP	6
#define	EEIP	4
#define	OSCFIP	7
#define	USBIP	5

char recibe[2 * (TRAMA_LONG)];
Int DATA [TRAMA_STORE] [TRAMA_LONG - 2] = {0};
int HEX[TRAMA_LONG];
int1 modif_trama = 0;

#int_default
void default_isr (void){

	if( !bit_test((*RCON),IPEN) ){
		jump_to_isr (LOADER+0x0008);
	}
	else{
		if(interrupt_active(INT_TIMER0)){		//Timer 0 (RTCC) overflow
			if( bit_test((*INTCON2),TMR0IP) ){
				goto_address(LOADER+0x0008);
			}
			else{
				goto_address(LOADER+0x0018);
			}
		}
		else if(interrupt_active(INT_TIMER1)){		//Timer 1 overflow
			if( bit_test((*IPR1),TMR1IP) ){
				goto_address(LOADER+0x0008);
			}
			else{
				goto_address(LOADER+0x0018);
			}
		}
		else if(interrupt_active(INT_TIMER2)){		//Timer 2 overflow
			if( bit_test((*IPR1),TMR2IP) ){
				goto_address(LOADER+0x0008);
			}
			else{
				goto_address(LOADER+0x0018);
			}
		}
		else if(interrupt_active(INT_TIMER3)){		//Timer 3 overflow
			if( bit_test((*IPR2),TMR3IP) ){
				goto_address(LOADER+0x0008);
```

```
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_EXT)){                     //External interrupt
                goto_address(LOADER+0x0008);
}
else if(interrupt_active(INT_EXT1)){                    //External interrupt #1
        if( bit_test((*INTCON3),INT1IP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_EXT2)){                    //External interrupt #2
        if( bit_test((*INTCON3),INT2IP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_RB)){                      //Port B any change on B4-B7
        if( bit_test((*INTCON2),RBIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_AD)){                      //Analog to digital conversion complete
        if( bit_test((*IPR1),ADIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_RDA)){                     //RS232 receive data available
        if( bit_test((*IPR1), RCIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_TBE)){                     //RS232 transmit buffer empty
        if( bit_test((*IPR1), TXIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
```

```
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_SSP)){                     //SPI or I2C activity
        if( bit_test((*IPR1),SSPIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_CCP1)){                    //Capture or Compare on unit 1
        if( bit_test((*IPR1),CCP1IP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_CCP2)){                    //Capture or Compare on unit 2
        if( bit_test((*IPR2),CCP2IP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_BUSCOL)){                  //Bus collision
        if( bit_test((*IPR2),BCLIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_LOWVOLT)){                 //Low voltage detected
        if( bit_test((*IPR2),LVDIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_COMP)){                    //Comparator detect
        if( bit_test((*IPR2),CMIP) ){
                goto_address(LOADER+0x0008);
        }
        else{
                goto_address(LOADER+0x0018);
        }
}
else if(interrupt_active(INT_EEPROM)){                  //Write complete
        if( bit_test((*IPR2),EEIP) ){
```

```
                        goto_address(LOADER+0x0008);
                }
                else{
                        goto_address(LOADER+0x0018);
                }
        }
        else if(interrupt_active(INT_OSCF)){            //System oscillator failed
                if( bit_test((*IPR2),OSCFIP) ){
                        goto_address(LOADER+0x0008);
                }
                else{
                        goto_address(LOADER+0x0018);
                }
        }

    }
}

void ASCII_HEX(void);
int Quemar_tramas(int);
void Limpiar_variables(void);

void main(void){
int saved_tramas = 0;
int i, resultado, bytes, checksum, checksum_f;
    usb_init();
    usb_task();
    usb_wait_for_enumeration();
    saved_tramas = 0;
    checksum_f = 0;
  while (TRUE){
   if(usb_enumerated()){
     if (usb_kbhit(1)){
                bytes = usb_get_packet(1, recibe, 2*(TRAMA_LONG));
                switch (recibe[1]){
                        case '2':
                                output_high(PIN_C2);
                                ASCII_HEX();
                                checksum = 0;
                                for(i=0; i < ((bytes/2)-1); i++)
                                        checksum += HEX[i + 1];
                                checksum_f |=checksum;
                                for(i=0; i<TRAMA_LONG-2; i++)
                                        DATA[saved_tramas][i] = HEX[i + 1];
                                saved_tramas ++;
                                if(saved_tramas == TRAMA_STORE){
                                        if(checksum_f){
                                                checksum_f = 0;
                                                resultado = '3';
                                        }
                                        else{
                                                resultado = Quemar_tramas(saved_tramas);
                                        }
```

```
                                        while(!usb_put_packet(1,&resultado, 1, USB_DTS_TOGGLE));
                                        if(resultado){
                                                output_low(PIN_C2);
                                        }
                                        saved_tramas = 0;
                                }
                                break;
                        case '3':
                                if(checksum_f){
                                        checksum_f = 0;
                                        resultado = '3';
                                }
                                else{
                                        resultado = Quemar_tramas(saved_tramas);
                                }
                                while(!usb_put_packet(1,&resultado, 1, USB_DTS_TOGGLE));
                                saved_tramas = 0;
                                output_low(PIN_C2);
                                break;
                        case '4':
                                ClearLCD();
                                limpiar_variables();
                                for(i=0; i<=10; i++)
                                        Array[i]=0x00;
                                for(i=0; i<=8; i++)
                                        Array2[i]=0x00;
                                saved_tramas=0;
                                i=0;
                                goto_address(LOADER);
                                break;
                        default:
                                break;
                }
        }
    }
  }
}

void ASCII_HEX(){
int i;
char temp [2 * TRAMA_LONG];
        for(i = 0; i < 2 * TRAMA_LONG; i++){
                if(recibe[i] >= '0' && recibe[i] <= '9')
                        temp[i] = recibe[i] - '0';
                else if(recibe[i] >= 'A' && recibe[i] <= 'F')
                        temp[i] = recibe[i] - 55;
                else
                        temp[i] = 0x0F;
        }
        for (i = 0; i < TRAMA_LONG; i++){
                HEX[i] = temp[2 * i];
                HEX[i] <<= 4;
                HEX[i] |= temp[2 * i + 1];
```

```
	}
	for(i = 0; i < 2 * TRAMA_LONG; i++)
		recibe[i] = 0x00;

	for(i=0; i < 2*TRAMA_LONG; i++){
		temp[i]=0x00;
	}
	i=0;
return;
}

int Quemar_tramas(int cant_tramas){
long int direccion;
int i,j,checksum;

	if(modif_trama){
		DATA[0][4]+=((LOADER/2)/0x100);
		modif_trama=0;
		if(DATA[0][4]>0x3F){
			return '1';
		}
	}

	for (i=0; i<cant_tramas; i++){
		for (j=1; j<DATA[i][0]; j+=2){
			if((DATA[i][4+j]==0xEC)||(DATA[i][4+j]==0xED)||(DATA[i][4+j]==0xEF)){
				if(j!=DATA[i][0]-1){
					DATA[i][4+j+1]+=((LOADER/2)/0x100);
					if(DATA[i][4+j+1]>0x3F){
						return '1';
					}
				}
				else{
					if(i!=(cant_tramas-1)){
						DATA[i+1][4]+=((LOADER/2)/0x100);
						if(DATA[i+1][4]>0x3F){
							return '1';
						}
					}
					else{
						modif_trama=1;
					}
				}
			}
		}
	}
	for (i=0; i<cant_tramas; i++){
		direccion = 0;
		direccion = DATA[i][1];
		direccion <<= 8;
		direccion |= DATA[i][2];
		if(direccion<0x8000-LOADER){
			direccion += LOADER;
```

```
                        write_program_memory(direccion, DATA[i]+4, DATA[i][0]);
                }
                else{
                        return '2';
                }
        }

i=0;
j=0;
direccion=0;
return '0';
 }

void limpiar_variables(){
int i, j;
        for(j=0; j<TRAMA_STORE; j++){
                for(i=0; i<TRAMA_LONG-1; i++){
                        DATA[i][j]=0x00;
                }
        }
        for(i=0; i<TRAMA_LONG; i++){
                recibe[2*i] = 0x00;
                HEX[i] = 0x00;
        }
        output_a(0x00);
        output_b(0x00);
        output_c(0x00);
        set_tris_a (0xFF);
        set_tris_b (0xFF);
        set_tris_c (0xFF);

return;
}
```

Anexo II Código Fuente del Programa Cargador (PicUSB.h).

```
#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

#DEFINE USB_TOTAL_CONFIG_LEN     32

    char const USB_CONFIG_DESC[] = {
      USB_DESC_CONFIG_LEN,
      USB_DESC_CONFIG_TYPE
      USB_TOTAL_CONFIG_LEN,0,
      1,
      0x01,
      0x00,
      0xC0,
      0x32,
```

```
    USB_DESC_INTERFACE_LEN,
    USB_DESC_INTERFACE_TYPE,
    0x00,
    0x00,
    2,
    0xFF,
    0xFF,
    0xFF,
    0x00,


    USB_DESC_ENDPOINT_LEN,
    USB_DESC_ENDPOINT_TYPE
    0x81,
    0x02,
    USB_EP1_TX_SIZE,0x00,
    0x01,


    USB_DESC_ENDPOINT_LEN,
    USB_DESC_ENDPOINT_TYPE,
    0x01,
    0x02,
    USB_EP1_RX_SIZE,0x00,
    0x01,
};

#define USB_NUM_HID_INTERFACES   0
#define USB_MAX_NUM_INTERFACES   1

const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

#if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
  #error USB_TOTAL_CONFIG_LEN not defined correctly
#endif

char const USB_DEVICE_DESC[] ={
    USB_DESC_DEVICE_LEN,
    0x01,
    0x10,0x01,
    0x00,
    0x00,
    0x00,
    USB_MAX_EP0_PACKET_LENGTH,
    0xD8,0x04,
    0x15,0x00,
    0x01,0x00,
    0x01,
    0x02,
    0x00,
    USB_NUM_CONFIGURATIONS
};
```

```
const char USB_STRING_DESC_OFFSET[]={0,4,12};

#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
	4,
	USB_DESC_STRING_TYPE,
	0x09,0x04,

	8,
	USB_DESC_STRING_TYPE,
	'F',0,
	'I',0,
	'E',0,

	22,
	USB_DESC_STRING_TYPE,
	'F',0,
	'I',0,
	'E',0,
	' ',0,
	'P',0,
	'i',0,
	'c',0,
	'U',0,
	'S',0,
	'B',0
};
#ENDIF
```