UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

DISEÑO Y CONSTRUCCIÓN DE UNA INTERFAZ USB, PARA EL CONTROL Y MONITOREO DE INSTRUMENTACIÓN ELECTRÓNICA

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA ELÉCTRICA-SISTEMAS ELECTRÓNICOS

P R E S E N T A:

ARTURO NEVÁREZ DOMÍNGUEZ

TUTOR:
M. I. MIGUEL ANGEL BAÑUELOS SAUCEDO

2006

JURADO ASIGNADO:

Presidente: M. en I. Lauro Santiago Cruz

Secretario: M. en I. Luis Arturo Haro Ruíz

Vocal: M. en I. Miguel Angel Bañuelos Saucedo

1er. Suplente: M. en I. José Castillo Hernandez

2do. Suplente: M. en I. Sergio Quintana Thierry

Lugar donde se realizó la tesis:

CCADET, Ciudad Universitaria, México DF.

TUTOR DE TESIS:

M. en I. Miguel Angel Bañuelos Saucedo

_____

FIRMA

AGRADECIMIENTOS

Este trabajo se lo dedico a mis padres Carlota y Arturo, a mis hermanas Taty y Andre, y por supuesto a Nadia por todo el amor y apoyo que me han brindado para que este trabajo fuera posible. Por todo lo que me han dado siempre estaré agradecido.

También quiero agradecer a mis buenos amigos y colegas Roberto Molero, Nadia Launizar, Jorge Cruz, Miguel Angel Bañuelos, José Castillo, Sergio Quintana y Ricardo Damián, ya que sin su ayuda y su apoyo nunca hubiera terminado este trabajo, es más probablemente no habría entrado a la maestría.

Además quiero agradecerle a mi tutor y a mis sinodales por haberme dado parte de su valioso tiempo, para la revisión de este trabajo. También quiero agradecerles por todas sus observaciones y comentarios que enriquecieron enormemente esta tesis.

Por último, pero no por eso menos importante, quiero agradecer al CONACYT por apoyarme por medio de una beca sin la cual no podría haber cursado la maestría, a la Facultad de Ingeniería por todos los conocimientos que me dieron a lo largo de la licenciatura y la maestría, y por supuesto al CCADET por haberme ofrecido la oportunidad de crecer como ingeniero y como persona durante estos últimos ocho años.

ÍNDICE

INTRODUCCIÓN

En nuestro país tenemos un rezago considerable en el campo del desarrollo tecnológico, comparando con muchos otros países, no solo de primer mundo, sino también comparándonos con países que solían estar aún más atrasados y que actualmente se encuentran experimentando un desarrollo mayor al nuestro. Si bien no se tiene la ilusión de solucionar este problema por cuenta propia, en el Laboratorio de Electrónica del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM, se está intentando continuamente realizar contribuciones que permitan a muchos otros explorar el área del desarrollo tecnológico.

En el Laboratorio de Electrónica del CCADET, el desarrollo tecnológico se enfoca principalmente en el desarrollo de instrumentación electrónica, para las diversas áreas del conocimiento que los requieran, inclusive se cuenta con una línea de trabajo enfocada al desarrollo de dispositivos biomédicos y una a la creación de dispositivos que serán utilizados en le área de la educación.

Ahora bien, una gran parte de los proyectos desarrollados últimamente en el Laboratorio de Electrónica fueron diseñados con alguna forma de comunicación con una computadora personal, esto con el objetivo de transmitir algunas de las variables medidas por los dispositivos a un programa en la PC (Computadora Personal) para su posterior análisis. Normalmente, la implementación de esta comunicación es utilizando el puerto serial o el puerto paralelo de la computadora, las cuales son las opciones más fáciles y rápidas de implementar, sin embargo, actualmente se cuenta con puertos que presentan diversas ventajas sobre los antes mencionados.

Si bien, el puerto serial y el puerto paralelo son relativamente fáciles de incorporar a un proyecto, al compararlos con el puerto USB (*Universal Serial Bus*) se puede observar que este cuenta con varias ventajas, algunas de las cuales son:

- En primer lugar, se encuentra la velocidad de transferencia de hasta 480Mbps.
- El puerto USB permite conectar hasta 127 dispositivos.
- Al conectar un dispositivo USB a la computadora personal, esta reconoce el dispositivo de forma automática e instala y configura los *drivers* necesarios para su uso (los *drivers* pueden ser proporcionados por el fabricante del dispositivo).
- La polarización puede ser proporcionada por el mismo puerto.
- Cada vez más de las computadoras personales en el mercado, dependen del Puerto USB, lo cual poco a poco va descartando a otros puertos como son el paralelo y el serial.

Además, en la siguiente tabla podemos comparar algunas características principales del puerto USB, serial y paralelo.

	USB	***Serial***	***Paralelo***
Estándar Industrial	Si	Si	No
Ancho de banda	480Mbps	115Kbps	115KBps EPP/ECP – 3MBps
Número de dispositivos	127 dispositivos en un solo puerto	Limitado al número de puertos disponibles en la computadora	Limitado al número de puertos disponibles en la computadora
Polarización por el bus	Hasta 500mA a 5V, cada puerto	No	No
Longitud máxima del cable	5m	3m	1.8m
Plug 'n' Play	Si	No	No
Cambio en "caliente"	Si	No	No

Tabla 1. Comparación de las características del puerto USB, serial y paralelo.

Aunque actualmente se cuenta con versiones mejoradas del puerto paralelo, las cuales pueden alcanzar grandes velocidades de transferencia, debido al gran aumento en el uso del puerto USB en diversas aplicaciones comerciales, es innegable que el puerto USB es la elección predilecta de los fabricantes de hardware para computadora en el mundo, principalmente debido a que se trata de una interfaz que iguala o supera la velocidad de transferencia de la mayoría de los otro puertos pero a su vez se trata de un puerto que es extremadamente amigable al usuario.

OBJETIVO

En este trabajo se desarrollará un dispositivo que permitirá implementar una comunicación bidireccional entre los nuevos instrumentos diseñados en el laboratorio de electrónica y una computadora personal, esto por medio del puerto USB. Lo cual a su vez nos permitirá adquirir el conocimiento necesario para desarrollar otros dispositivos con tecnología USB en el futuro.

La interfaz a desarrollar debe de ser adaptable a cualquier instrumento que requiera recibir y transmitir datos a la computadora. Además de que debe de poder ser adaptada por cada diseñador que lo requiera.

El diseño se enfocará en una transmisión de baja velocidad, menor a 12Mbps, esto con el propósito de crear un diseño básico que más adelante pueda ser utilizado como escalón para diseños mejores y más rápidos, los cuales tomen plena ventaja de las capacidades del puerto USB.

El diseño estará dividido en tres partes

- **El *hardware***. Esta parte se incluirá la parte física del proyecto, es decir, la electrónica.
- **El *firmware***. En esta parte quedará incluido el programa creado para el microcontrolador que se utilizará en el hardware, el cual implementará el protocolo USB en nuestra interfaz.
- **El *software***. Esta parte estará formada por el programa para PC, que se creará como interfaz para que el usuario sea capaz de manipular la conexión por el puerto USB con el instrumento del cual se desea recibir o transmitir datos.

Por supuesto, se deberá incluir dentro del trabajo, una descripción de los pasos necesarios para realizar la adaptación de la interfaz diseñada al instrumento deseado.

CAPÍTULO I: ANTECEDENTES

El puerto USB surge ante la creciente necesidad de los usuarios de conectar cada vez más dispositivos a su PC. Originalmente el número de dispositivos que se podían conectar a una PC estaba limitado por el número de puertos disponibles en la computadora, ya fueran puertos paralelos, seriales, SCSI (*Small Computer System Interface*), PCI (*Peripheral Component Intercon*), ISA (*Industry Standard Architecture*), etc.

Con la llegada del puerto USB, llegaron también una gran variedad de beneficios para el usuario, algunos de los cuales son:

- El puerto USB permite realizar la conexión y desconexión de dispositivos en "caliente", es decir, sin tener que apagar la computadora.
- Facilidad de uso. Al conectarse un dispositivo USB en el puerto de una computadora, ésta lo reconoce, selecciona el *driver* adecuado, lo instala y configura automáticamente.
- Los conectores para el puerto son estándar y se encuentran descritos en las especificaciones USB [8].
- Puede presentar una velocidad de transmisión de datos de hasta 480 Mbps.
- Cuenta con tres velocidades de transferencia de datos (low-speed, Full-speed y Hi-speed), y el puerto se adecua a la velocidad del dispositivo conectado, de forma automática.
- Es posible conectar hasta 126 dispositivos en el mismo puerto.
- El puerto proporciona la opción de polarizar el dispositivo a través del mismo cable o por medio de una fuente externa.
- Por medio del mismo puerto los dispositivos pueden configurarse para que se desactiven automáticamente cuando no presenten actividad.
- Cuenta con detección y corrección de errores.
- El puerto es externo a la PC, esto significa que no es necesario abrir la computadora para instalar el dispositivo deseado [1], [2], [8], [10].

A pesar de las ventajas que presenta el puerto USB para el usuario, el diseño es algo más complicado. Para comenzar el diseño de una interfaz para este puerto, es necesario estudiar su protocolo, el cual nos indica como se deberá procesar la información transmitida y recibida.

La teoría detrás de la comunicación por medio del puerto USB es muy extensa. Existen manuales en donde se describe a detalle todo lo necesario para implementarla. Debido a esto, sólo expondremos algunos puntos básicos de dicha teoría, como antecedentes para poder comprender este trabajo. Si el lector lo requiere podrá consultar la fuente de dicha teoría, la cual se lista en la bibliografía [1], [2], [8], [10].

1.1. CONCEPTOS BÁSICOS

1.1.1. Arquitectura del puerto USB

El puerto USB, es una conexión por cable, la cual permite intercambiar datos entre una computadora principal (*host*) y un amplio rango de dispositivos periféricos simultáneamente accesibles.

Un sistema USB puede ser dividido en tres partes principales[1], [2]:

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

1.1.1.1. La interconexión USB

Cuando mencionamos interconexión, nos referimos a la forma en que los diversos dispositivos se conectan y comunican con el *host*.

En cada bus solo puede haber un solo *host*. Para que sea posible conectar más dispositivos en un mismo puerto, es necesario utilizar un dispositivo extra que

llamaremos *hub*, el cual tiene como función proporcionar más conectores para poder conectar otros dispositivos al puerto. Cabe mencionar, que es común que el *host* mismo tenga un *hub* interno, para proporcionar al usuario un mayor número de puntos de conexión. A dicho *hub*, se le conoce como *root hub*.

Para poder entender mejor la forma en que se conectan los dispositivos involucrados en la comunicación por el puerto USB, se muestra el siguiente diagrama.

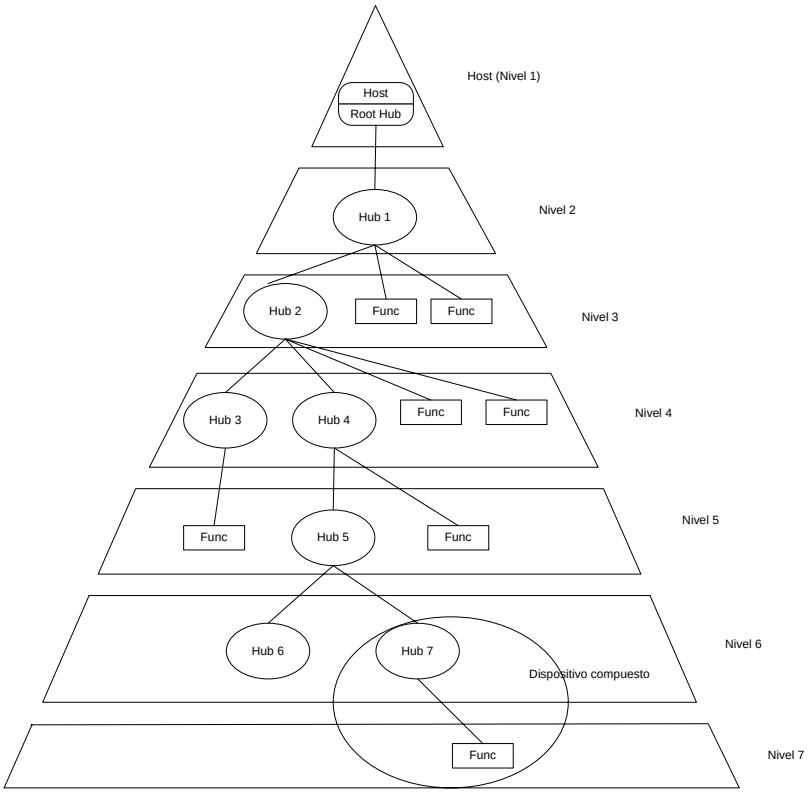


Figura 1. Topología del puerto USB.

Como se puede observar en el diagrama anterior, la topología del USB es del tipo estrella por niveles [1], [2], [8]. Un *hub* se encuentra en el centro de cada estrella. Y cada cable es una unión directa entre el *host* y un *hub*, un *hub* y otro *hub,* un *host* y una función, o entre un *hub* y una función.

1.1.1.2. Los dispositivos USB

Los dispositivos USB pueden ser uno de los siguientes:

- *Hubs.*
- Funciones, las cuales básicamente son dispositivos USB, los cuales al conectarse al *host*, directamente o por medio de un *hub*, agregan sus propias capacidades al *host [1]*, [2], [8].

1.1.1.3. El *host* USB

En cualquier sistema USB solamente existe un solo *host*. A la interfaz USB de la computadora se le conoce como controlador *host*. El controlador host puede ser implementado con una combinación de *hardware*, *firmware* ó *software*. Dentro del sistema *host*, es integrado un *hub* raiz, para proporcionar uno o más puntos de conexión.

1.1.2. Características eléctricas del puerto USB

El puerto USB transfiere la información y la polarización a través de un solo cable de cuatro alambres. Dos de los cables proveen la energía para polarizar al dispositivo que se conecte y dos transportan los datos.

Existen tres tasas de transferencia de datos para el puerto USB:

- 480 Mb/s, es la velocidad de transferencia máxima para el USB y se denomina *high-speed.*
- 12 Mb/s, es la velocidad intermedia de transferencia conocida como *full-speed.*
- 1.5 Mb/s, es la velocidad de transferencia más baja en la cual trabaja el USB, y se le denomina *low-speed.*

El puerto USB transmite una señal de sincronía, para que el receptor sintonice su reloj utilizando las transiciones de los datos recibidos. La señal de sincronía es transmitida de forma codificada junto con los datos. La codificación del mismo es NRZI (siglas en ingles para, “No retorno a cero invertido”) con rellenado de bits (*bit stuffing*) [1], [2], [8], [10].

1.1.3. Protocolo del puerto USB

Contrario a lo que se pueda pensar, el *host* no detecta transmisiones entrantes en el puerto USB, por medio de interrupciones, el *host* monitorea el puerto USB cada determinado número de microsegundos buscando datos en los dispositivos conectados al puerto. En la comunicación USB el controlador *host* inicia todas las transferencias. La mayoría de las transacciones comienza cuando el controlador *host*, de forma preprogramada, envía un paquete, el cual describe el tipo y dirección de la transacción, la dirección del dispositivo USB, y el número del punto de salida (*endpoint*). A este paquete se le conoce como paquete de muestra (*token packet*). El dispositivo USB que reconozca la dirección enviada por el *host* como la suya, se dispone a responder. En cada transacción, los datos pueden ir del *host* al dispositivo USB o viceversa, y la dirección de la transferencia se indica en el paquete de muestra. Entonces el origen de la transacción envía un paquete de datos o indica que no tiene datos para transmitir. El destino, normalmente responde con un paquete de saludo (*handshake packet*), en el cual se indica si la transferencia fue exitosa [1], [2], [8], [10].

El modelo de transferencia de datos USB entre la fuente o destino y el punto de salida en un dispositivo es conocido como conducto (*pipe*). Existen dos tipos de conductos: de flujo (*stream*) y mensaje (*message*). El flujo de datos no tiene una estructura predefinida para el USB, mientras que los datos mensaje sí. De forma adicional, los conductos tienen asociado un ancho de banda, tipo de transferencia, y características del punto de salida, como dirección y tamaño del buffer [1], [2], [8], [10].

La mayoría de los conductos son creados cuando el dispositivo USB es configurado. Solo un conducto existe desde que el dispositivo es polarizado, esto para permitir el acceso a la configuración del dispositivo, a su información de estado y de control.

1.1.4. Robustez

Hay varios atributos del USB que contribuyen a la robustez del mismo:

- Integridad de la señal, al utilizar señales diferenciales y blindaje en los cables.
- Protección CRC (siglas en ingles para, “verificación de redundancia cíclica), sobre campos de control y datos.
- Detección de la conexión y desconexión y de configuración a nivel del sistema de los recursos.
- Auto-recuperación en protocolo, usando tiempos fueras para paquetes corruptos o perdidos.
- Control de flujo para que en las transmisiones de datos se asegure la sincronía, y también para el manejo del buffer de hardware.
- Construcciones de conductos de datos y control para garantizar la independencia de interacciones adversas entre funciones [1], [2], [10].

1.1.4.1. Detección y manejo de errores

La tasa de error de *bit* basal, del medio USB, se espera que sea cercana a la del *backplane*, y por lo tanto, es muy probable que cualquier error sea de naturaleza transitoria. Para proveer protección contra dichos transitorios, cada paquete tiene campos para protección contra errores. Cuando se requiere integridad de los datos, se puede invocar a un procedimiento de recuperación de datos, ya sea por hardware o por software.

El protocolo incluye CRCs separadas para los campos de datos y control de cada paquete. Un CRC fallido, se considera como un indicador de un paquete dañado. La CRC cubre el 100% de los errores de uno o dos bits [29].

El manejo de errores incluye el reporte y reintento de transferencias fallidas. Un controlador USB reintentará hasta tres veces efectuar una transmisión fallida antes de informarle al software cliente de la falla [1], [2], [8], [10].

1.1.4.2. Configuración del sistema

El USB permite la conexión y desconexión de dispositivos USB en cualquier momento. Consecuentemente, el *software* del sistema debe contemplar cambios dinámicos en la topología física del sistema [1], [2], [8], [10].

1.1.4.3. Conexión de dispositivos USB

Todos los dispositivos USB se conectan al puerto a través de dispositivos especializados conocidos como *hubs*. Los *hubs* tienen bits de estado que son usados para reportar la conexión o desconexión de un dispositivo USB en sus puertos. El *host* le pide al *hub* que envíe estos bits. En el caso de una conexión, el *host* activa el puerto y direcciona el dispositivo a través del conducto de control en la dirección predeterminada.

El *host* asigna una dirección única al dispositivo y después determina si el dispositivo USB recientemente conectado es un *hub* o una función. El *host* establece su parte del conducto de control para el dispositivo USB utilizando la dirección asignada y el punto de salida cero.

Si el dispositivo conectado es un *hub* y se conectan a él otros dispositivos USB a sus puertos, entonces cada dispositivo debe llevar a cabo el procedimiento descrito anteriormente.

Si el dispositivo conectado es una función, entonces las notificaciones de conexión serán manejadas por algún programa del *host* que sea apropiado para la función [1], [2], [8], [10].

1.1.4.4. Enumeración del bus

La enumeración del bus es la actividad que identifica a los dispositivos conectados por medio de dos números de identificación, el PID (Identificación del producto) y el VID (Identificación del fabricante), para después asignarles una dirección única en el bus. Debido a que el USB permite a los dispositivos USB conectarse y desconectarse en cualquier momento, la enumeración es una actividad del sistema que no se detiene [1], [2], [8], [10].

1.1.4.5. Desconexión de dispositivos USB

Cuando un dispositivo USB es desconectado de uno de los puertos de un *hub*, el *hub* deshabilita el puerto y provee una indicación de la desconexión del dispositivo al *host*. La indicación de desconexión es manejada entonces por un programa del sistema apropiado. Si el dispositivo USB removido es un *hub*, el programa USB del sistema debe manejar la desconexión del *hub* y de todos los dispositivos previamente conectados a él [1], [2], [8], [10].

1.2. TIPO DE FLUJOS DE DATOS

El USB soporta intercambios de datos funcionales y de control, entre el *host* USB y un dispositivo USB, como un conjunto de conductos tanto unidireccionales como bidireccionales. La transferencia de datos se realiza entre el *software* del *host* y un punto de salida de algún dispositivo USB. A tal asociación entre el *software* del *host* y un dispositivo USB se le denomina conducto. Algunos dispositivos USB pueden tener varios conductos.

La arquitectura USB soporta cuatro tipos básicos de transferencias:

- Transferencias de control: Es usada para configurar un dispositivo en el momento de la conexión y se puede usar para otros propósitos específicos, incluyendo control de otros conductos del dispositivo.
- Transferencia de bloque de datos: Generados o consumidos en un tiempo y cantidad relativamente grandes y cuentan con una gran libertad en cuanto a las restricciones de la transmisión.
- Transferencia de datos por interrupción: Es usada para realizar entregas de datos puntuales y consistentes, por ejemplo, caracteres o coordenadas con eco humanamente perceptible o características de retroalimentación.
- Transferencia de datos isocrónica: Ocupa una cantidad de ancho de banda prenegociada, del puerto USB, con latencia de entrega prenegociada. (También conocida como transferencia de flujo en tiempo real).

Una conducto soporta solamente un tipo de transferencia de las descritas arriba, para una cierta configuración del dispositivo [1], [2], [8], [10].

CAPÍTULO II: DISEÑO DEL *FIRMWARE*

2.1. INTRODUCCIÓN

Para comenzar con el diseño de un dispositivo que tenga conectividad USB, no existe una regla que especifique el orden en el que se debe ir desarrollando el sistema. Sin embargo, aquellos que en alguna ocasión hayan diseñado un dispositivo con puerto USB sugieren un cierto orden, en el cual se debe realizar todo el diseño [1], [2].

Para empezar, es recomendable escoger el dispositivo programable que se utilizará para desarrollar la interfaz USB del hardware a diseñar [1], [2].

Antes de escoger un dispositivo electrónico como plataforma para nuestro diseño, es necesario señalar las especificaciones que éste debe cumplir. No sólo en cuanto a las especificaciones relacionadas con el puerto USB, sino también con el aspecto económico y de disponibilidad, así como también de las herramientas disponibles para su desarrollo [1], [2].

Los parámetros fundamentales que se deben de tomar en cuenta, para la selección del dispositivo programable a utilizar en la implementación de la interfaz USB son:

- Disponibilidad: El dispositivo seleccionado debe de poderse conseguir en nuestro país, esto principalmente porque reduce los costos del diseño. Además debe de ser un componente que tenga perspectivas de permanecer en el mercado por algún tiempo, de tal forma que no tengamos que emigrar a otra plataforma demasiado pronto [1], [2].
- Herramientas de desarrollo: Es deseable que la plataforma a escoger, tenga el soporte necesario para facilitar el desarrollo, esto con el objeto de acortar los tiempos de diseño y construcción [1], [2].

- Precio: Este punto se encuentra ligado al primero, y aunque no es definitivo, siempre se busca disminuir los costos en el diseño [1], [2].

2.1.1. Dispositivos programables considerados para el diseño

Todos los dispositivos programables que incluyen en su arquitectura el hardware necesario para implementar la comunicación USB, deben de tener una serie de elementos básicos para ello, tal como se muestra en la Figura 2.

El transceptor (*Tanscever*), se encarga de convertir entre las señales diferenciales bidireccionales del USB, y las señales referenciadas a tierra de nivel TTL/CMOS del motor de la interfaz serial (SIE).

El motor de la interfaz serial o SIE (*Serial Interface Engine*), se encarga de recibir bits, o bytes del tranceptor, para validarlos, y entonces enviar datos validos a la interfaz del SIE. Además el SIE es el que tiene que recuperar el reloj del bus, a través de las transiciones en el bus, para así poder sincronizarse a el.

Ahora bien, la interfaz del SIE, sirve como puente entre el controlador de protocolo y el propio SIE y puede tratarse de una comunicación paralela o incluso serial con un protocolo I^2C. Esto depende principalmente de la velocidad con la que se quiera transmitir datos. Cabe mencionar que esta parte junto con el controlador de protocolo y el módulo de E/S, pueden todas estar incluidas en un microcontrolador.

El módulo controlador del protocolo, es el que se encarga de implementar el protocolo USB, de tal forma que controla a los otros módulos para poder establecer la comunicación, además de que procesa los datos recibidos del lado del bus y del módulo de entradas/salidas.

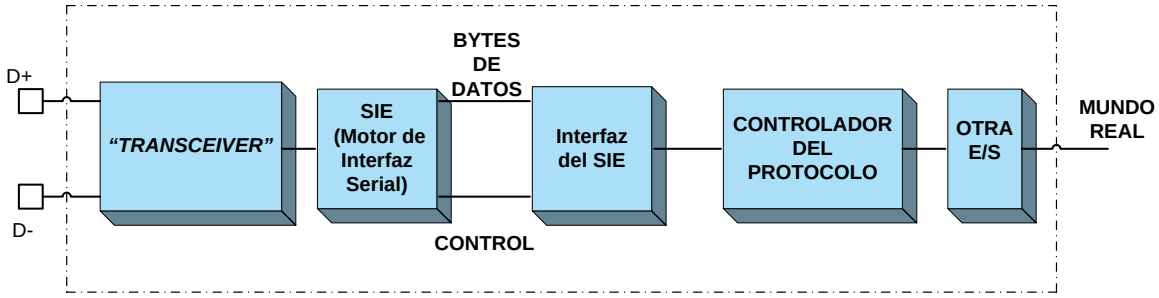


Figura 2. Diagrama de bloques de un dispositivo de E/S.

Existen una gran cantidad de dispositivos los cuales pueden ser usados para desarrollar una interfaz USB, sin embargo, en este capítulo solamente se incluyen aquellos que pueden ser conseguidos de forma relativamente fácil. Los dispositivos a escoger deberán incluir el hardware necesario para implementar el puerto USB en nuestra interfaz, ya sea separado o integrado a un microcontrolador.

A continuación se presenta una breve descripción de los dispositivos considerados.

Dispositivo	***Fabricante***	***Versión USB***	***Distribuido en México***	***MCU integrado***	***Velocidad máxima de transferencia***
TUSB3210	Texas Instruments	V. 2.0	Si	Si (8052)	*Full speed*
USBN9603	National Semiconductors	V. 2.0	Si	No	*Full speed*
PIC18F4550	Microchip	V. 2.0	Si	Si	*Full speed*
PIC16C765	Microchip	V. 1.1	Si	Si	*Low speed*

Tabla 2. Dispositivos USB considerados.

TUSB3210

El circuito integrado TUSB3210 es un controlador USB diseñado teniendo como base un microcontrolador 8052, con puertos de entrada y salida de propósito general "*GPIO*". El TUSB3210 cuenta con memoria de acceso aleatorio de 8K x 8, para el desarrollo de aplicaciones. Además, la programabilidad del TUSB3210 lo

hace lo suficientemente flexible como para realizar varias otras aplicaciones de entrada/salida (I/O: *Input/Output*) de propósito general. Utilizando un cristal de 12MHz, el oscilador interno genera el reloj del sistema de 49 MHz, necesario para establecer comunicación USB a *full-speed*. El dispositivo puede ser programado vía una interfaz I^2C durante el encendido desde una EEPROM, u opcionalmente el *firmware* puede ser descargado al dispositivo desde una PC *host* vía USB. Este microcontrolador, basado en el circuito MCU 8052, permite utilizar algunas herramientas de desarrollo de terceros para la creación de nuevas aplicaciones, como pueden ser compiladores de C [11].

USBN9603

El USBN9603 es un controlador de nodo USB integrado. El dispositivo provee soporte para DMA mejorado, con muchas características automáticas de manejo de datos. Es compatible con las especificaciones USB 1.0 y 1.1, y es una versión avanzada del USBN9602.

El dispositivo integra el *transceiver* USB necesario, con un regulador de 3.3V, un motor de interfaz serial (SIE), FIFOs para el punto de salida del USB, una interfaz paralela muy versátil de 8 bit, un generador de reloj y una interfaz MICROWIRE/PLUS™. Soporta siete puntos de salida, uno es el punto de salida de control (el cual es bidireccional) y los otros 6 para manejar transferencias isocrónicas, de bloque de datos y por interrupciones. Cada conducto de un punto de salida tiene su propio FIFO, de 8 bytes para el punto de salida de control y 64 bytes para cada uno de los otros seis [12].

Para poder utilizar este dispositivo, es necesario incluir en el diseño un microcontrolador para poder implementar el módulo del controlador

PIC16C765

Los dispositivos PIC16C745/765 son microcontroladores CMOS de alto desempeño y bajo costo de 8 bits, de la familia de rango medio. El PIC16C765 tiene 33 terminales de entrada/salida. Cada dispositivo cuenta con 256 bytes de

RAM. Además, tiene incluidos varios periféricos, como: tres contadores/ temporizadores, dos módulos de captura/comparación/PWM y dos puertos seriales. Contiene además el periférico USB (USB1.1), que provee comunicación por el bus, y el Transmisor Receptor Síncrono Asíncrono Universal (USART) también conocida como Interfaz Serial de Comunicación (SCI). También tiene 8 canales A/D de alta velocidad de 8 bits de resolución. Además, este dispositivo, cuenta con una memoria EPROM de programa.

El módulo periférico USB soporta transferencias de control y de interrupción (entrada y salida). La implementación soporta 3 puntos de salida (0, 1, 2) para un total de 6 puntos [26].

PIC18F4550

Este es un nuevo dispositivo de Microchip, el cual cuenta con el hardware necesario para implementar la comunicación por el puerto USB.

La familia de dispositivos a la que pertenece el PIC18F4550 ofrece las ventajas de la familia de microcontroladores PIC18, es decir, alto rendimiento computacional a un precio económico, con la ventaja extra de una memoria FLASH mejorada de alta duración. Además de estas características, esta familia de MCUs introduce mejoras de diseño, que hacen de este dispositivo una opción lógica para muchas aplicaciones de alto rendimiento y sensibles al consumo de energía. Estos dispositivos incorporan un módulo de comunicaciones USB completo, el cual es compatible con la especificación de USB 2.0, soportando comunicaciones, tanto *low-speed* como *full-speed*, en todos los tipos de transferencia. También incorpora su propio tranceptor, además de su propia fuente de 3.3V [24].

2.2. SELECCIÓN DE HARDWARE

De los dispositivos mencionados anteriormente, se seleccionaron inicialmente dos, para servir como plataforma de diseño inicial, el USBN9603 de Nacional Semiconductors y el PIC16C765 de Microchip. Este último presenta limitaciones

en cuanto a la velocidad soportada, así como, en el hecho de que solo soporta el protocolo USB 1.1. Sin embargo, se decidió utilizar estos integrados por que ambos forman plataformas que integran microcontroladores de Microchip, lo cual representa una ventaja debido a la experiencia con este fabricante de MCU, además de las herramientas de desarrollo las cuales incluyen un compilador de C.

La plataforma a utilizar se recomienda que sea una tarjeta de evaluación, ya que esto nos permite descartar errores de hardware en las etapas iniciales de diseño, en las cuales se desarrolla el *firmware* del proyecto. Debido a esto, se escogió inicialmente la tarjeta de evaluación del PIC16C765, sin embargo, rápidamente quedó claro que este dispositivo se encontraba limitado en recursos, debido principalmente a el tipo de memoria interna (EPROM) y la velocidad de transmisión USB máxima que podía alcanzar, por lo que se decidió probar con el nuevo microcontrolador de Microchip PIC18F4550, el cual presentaba numerosas mejoras, además de ser mucho más poderoso en general [1], [2].

2.3. DESARROLLO DEL *FIRMWARE* PARA EL PIC184550

El desarrollo del programa para la interfaz comenzó con el programa que Microchip llama USB *Framework*, el cual es un marco que permite generar aplicaciones que incorporen comunicación USB. Inicialmente dicho marco se incluía como una aplicación que emulaba a un "*mouse*" o un teclado, con la documentación de la tarjeta de evaluación. Sin embargo, la autora del libro "USB Complete" Jan Axelson, modificó este programa para generar un dispositivo HID (dispositivo de interfaz humana) general. El programa resultante al igual que el programa original fue desarrollado en el lenguaje C, por medio de un compilador de C para los microcontroladores de Microchip [1], [2], [22], [25].

Inicialmente, el programa de Axelson estaba diseñado para convertir al microcontrolador en un dispositivo HID que recibía dos bytes de datos de un programa en la PC y después los regresaba, a esta versión del programa se le nombró HID2.

El diagrama de flujo simplificado del programa de Jan Axelson se muestra en la figura 3.

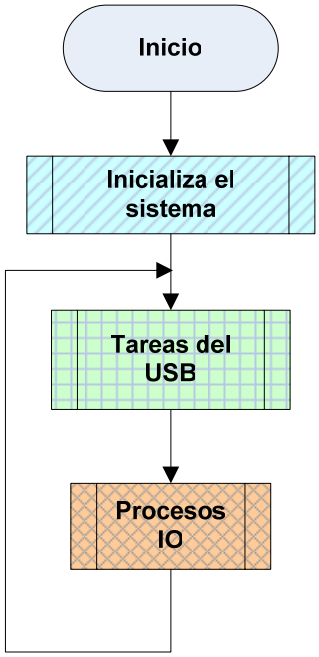


Figura 3. Diagrama del programa HID2.

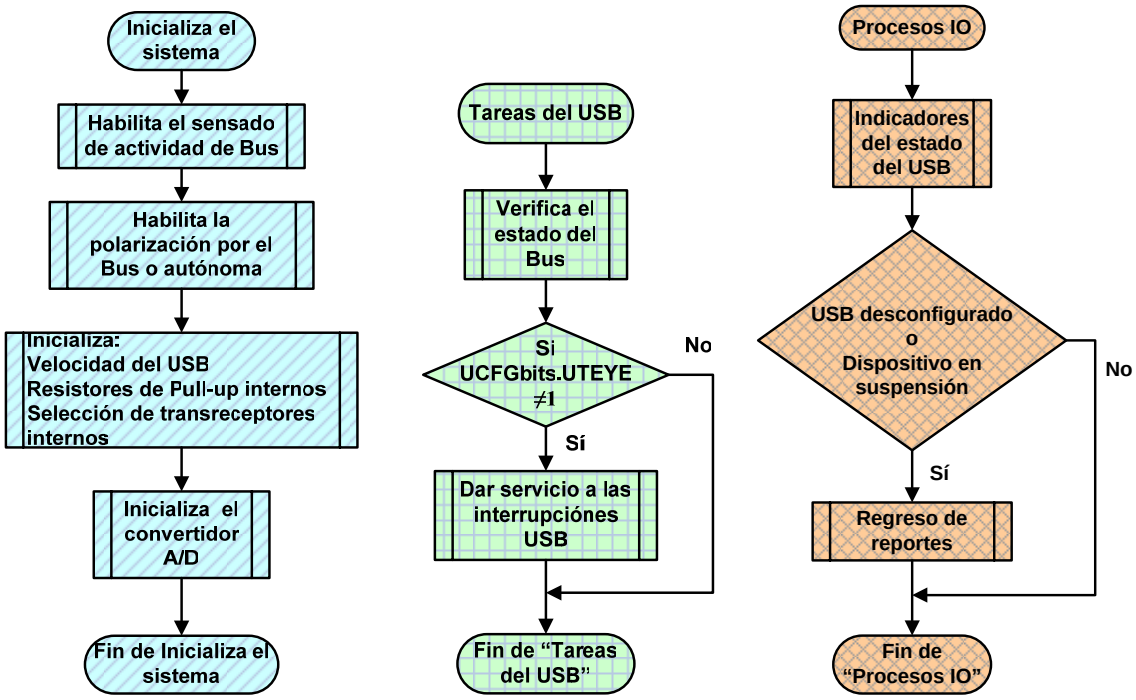
El diseñador debe trabajar dentro y alrededor del *framework*, para introducir sus propias funciones en el programa, dejando así que éste se encargue del funcionamiento del puerto. Aunque es posible y en muchas aplicaciones necesario modificar el *framework* para adaptarlo a las necesidades del problema a resolver.

Otro aspecto importante, es que el diseñador debe trabajar al mismo tiempo en el software para la computadora, para en determinado momento probar los cambios realizados al *firmware*. Como punto de partida se utilizó un proyecto de Visual Basic .NET, creado por Jan Axelson para probar el *firmware*. Se trataba de un pequeño programa que enviaba dos *bytes* al dispositivo USB y después recibía los mismos dos *bytes* de vuelta. Por simple que parezca, esta es una parte medular del proyecto, ya que el lograr la transmisión y recepción de datos por el puerto USB, permite avanzar de forma más rápida en el desarrollo del proyecto [22].

2.4. El FRAMEWORK DE MICROCHIP

Antes de proseguir, debemos entender más a fondo la estructura del *framework* de Microchip, el cual controla la comunicación por el puerto USB [25].

Anteriormente en la figura 3, se mostró el diagrama de flujo simplificado del framework, mostrando sólo las funciones de manera superficial. En la figura 4 se puede observar con un poco más de profundidad la estructura de este programa.



(a) (b) (c)

Figura 4. (a) Subrutina "Inicializa el sistema"
(b) Subrutina "Tareas del USB"
(c) Subrutina "Procesos IO".

El *framework* tiene dentro de su estructura funciones que se encargan de todas las interrupciones del microcontrolador generadas por la actividad en el puerto USB.

Como se mencionó anteriormente, el programa de Microchip, se encuentra escrito en lenguaje C, para su propio compilador. A pesar de que el compilador de

Microchip, Mplab C18 está diseñado de acuerdo al estándar ANSI C, cabe recordar que una parte está compuesta por instrucciones específicas para los microcontroladores PIC, las cuales varían de compilador a compilador [17], [18], [19], [20].

Para aprovechar el programa de Axelson, que anteriormente nombramos HID2, se decidió basar las primeras versiones del *firmware* en el código en C, incluyendo la parte en la que envía y recibe dos *bytes* a la vez.

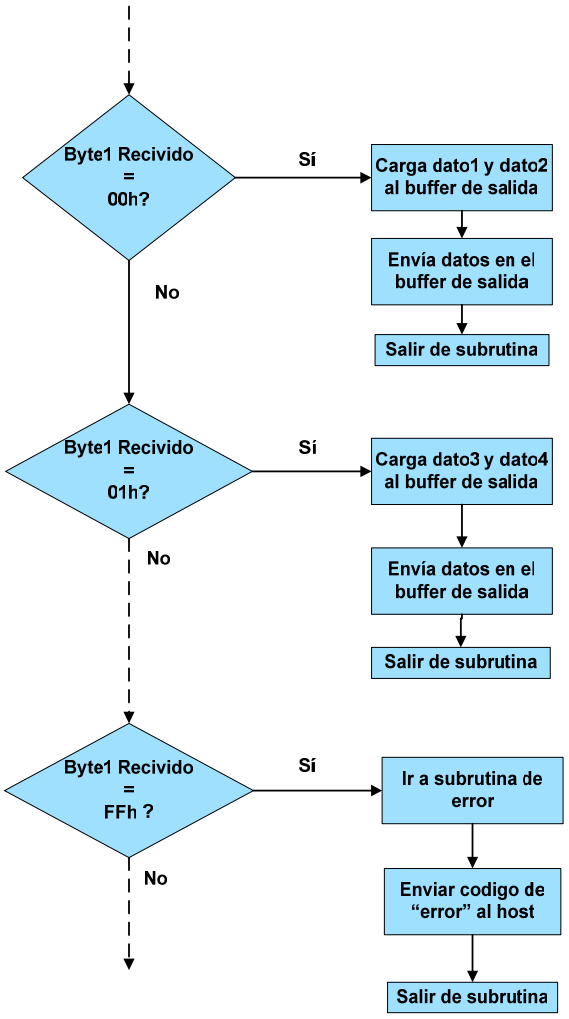


Figura 5. Diagrama simplificado de secuencia de comandos en el microcontrolador.

En primer lugar, se decidió utilizar los dos *bytes* que enviaba el *host*, para poder implementar un pequeño grupo de comandos, de tal forma que el microcontrolador en vez de regresarlos, los interpretara y ejecutara una función específica de acuerdo al comando recibido. Por ejemplo, el primer comando a realizar sería ejecutado al recibir dos ceros del *host*, y sería enviar un número X por el puerto USB. La idea no es ejecutar una función compleja con el MCU, sino probar que al ingresar más código en el *framework*, éste no colapsara. Este pequeño ejemplo se describe más fácilmente con el diagrama de flujo que se muestra en la figura 5.

Es necesario, al realizar modificaciones al *framework*, que no se interrumpa el flujo del mismo, ya que esto provocaría errores en las rutinas de mantenimiento del puerto USB, que se encargan de manejar las interrupciones generadas por el mismo en el microcontrolador PIC18F4550.

A pesar de que uno de los puntos fuertes de la comunicación por el puerto USB es que el formato de los datos enviados y recibidos pueden adecuarse a las necesidades del proyecto, tanto en forma como en tamaño, se optó por dejarlo como estaba, de tal forma que los datos a enviar serían adaptados para enviarlos y recibirlos dos *bytes* a la vez. Sin embargo, una vez que el proyecto funcione correctamente, manteniendo el mismo formato de los datos, se puede pensar en modificar el formato de los mismos.

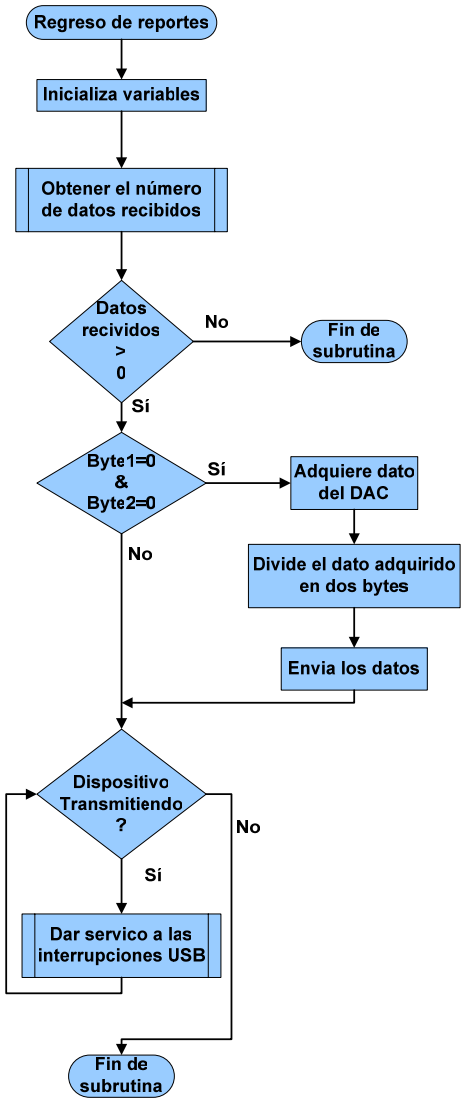


Figura 6. Subrutina “Regreso de reportes” del programa de prueba HID3.

Entonces, ya planteada la idea básica del funcionamiento del *firmware* modificado, se realizó el siguiente diagrama de flujo, el cual representa el código que sería integrado al *framework* [17], [18], [19], [20], [25].

El diagrama que se presenta en la figura 6 es una versión modificada de la función *ReportLoopback* (Regreso de reportes), la cual originalmente contenía el código que tomaba los datos recibidos y los enviaba de regreso, y que se denominó como HID3. En esta

modificación, en realidad se está implementando una versión básica del ejemplo que se mostró en la figura 5, con el objetivo de probar su funcionamiento [17], [18], [19], [20], [25].

La subrutina “Regreso de reportes” básicamente verifica si se ha recibido algún dato del *host*, de ser así, el microcontrolador toma los datos recibidos y los compara con una serie de opciones, las cuales representan diferentes comandos. En el caso de este primer ejemplo solamente se cuenta con un comando, el cual se ejecuta cuando los dos *bytes* recibidos son iguales a cero. En el caso de que el único comando se active, éste adquiere un dato del convertidor A/D de 10 bits, lo divide en dos *bytes* y lo envía al *host*.

Hay que recordar que esta parte es de gran importancia, ya que el diseño que se busca es de un dispositivo que permita a los diseñadores que quieran utilizarlo, programarlo para realizar diversas tareas con el microcontrolador, por medio de comandos recibidos a través del puerto USB. Si alguien quisiera que se ejecutaran diferentes tareas, lo que tendría que hacer es redefinir los comandos definidos en la subrutina adecuada, tal y como se muestra en la figura 6, además de que el diseñador podría diseñar sus propias subrutinas, las cuales podría llamar en cuanto se detectara un evento en específico, incluyendo los comandos mencionados. Cabe también mencionar que la comunicación es en dos sentidos, y así como el microcontrolador puede ejecutar tareas después de recibir algún comando del *host*, también puede transmitir datos al *host*, después de detectar algún evento externo al USB.

El adquirir datos por el convertidor no fue escogido al azar para el ejemplo, en realidad se tiene pensado unir la interfaz a varios instrumentos que se están diseñando en paralelo, y de los cuales varios necesitarán enviar muestras de señales analógicas a la computadora por medio del convertidor A/D del microcontrolador y por el puerto USB. Un buen ejemplo sería un sistema de adquisición de datos (de temperatura por ejemplo) el cual podría muestrear señales analógicas de varios de sus canales, para después enviar esas muestras

a la computadora, donde serían procesadas en la interfaz de usuario en la PC, o simplemente guardadas en la computadora para futuro análisis.

2.5. LA ENUMERACIÓN

Ya se habló brevemente en los antecedentes de este trabajo sobre el proceso inicial por el que pasan el *host* y el dispositivo, la enumeración. Ahora, veremos con un poco más de detalle en esta sección, como es que se implementó este proceso, del lado del microcontrolador, en la parte del *framework*. En esta parte es necesario adentrarse un poco más en los registros del microcontrolador PIC18F4550, por lo que se explica junto con un diagrama de flujo, pero aunque puede ser un poco complejo, es necesario para entender el proceso por el que tiene que pasar el microcontrolador para poder configurarse hasta el punto de estar listo para la comunicación [1], [2], [8], [10], [25].

En la parte inicial, el hardware debe estar configurado, para que el *hub* raíz, el cual es controlado directamente por el *host*, detecte cuando algún dispositivo se conecte al bus, esto se logra monitoreando los voltajes en los cables de datos.

Después de que el MCU PIC18F4550 se conecta al puerto, éste se polariza y empieza a inicializar variables y registros, para empezar la comunicación. Entonces el microcontrolador inicia la subrutina "*USBTasks*" (Tareas del USB), la cual se muestra en el diagrama de la figura 7 y que se explica a continuación. Lo primero que hace el microcontrolador es verificar si se encuentra conectado al puerto, de ser así entonces enciende el módulo USB (que se encuentra integrado en el PIC18F4550), si es que no había sido habilitado, esto hace que el estado actual de esta máquina de estados pase al estado_conectado (*ATTACHED_STATE*). En el caso de que el microcontrolador se encuentre desconectado del puerto, entonces apaga el modulo USB, siempre y cuando éste no estuviera ya apagado, esto hace que el estado actual cambie a estado_desconectado (*DETACHED_STATE*) [25].

En seguida, el microcontrolador verifica si se encuentra en estado_conectado (*ATTACHED_STATE*), de ser así, se verifica la bandera SE0 en el registro UCON (UCONbits.SEO en la figura 7), para dar tiempo a que los voltajes en las terminales diferenciales del USB, D+ y D-, alcancen el voltaje necesario después de que se habilita el modulo USB, para evitar que se malinterprete como una señal de reset del sistema. Ya que la bandera SE0 se ha limpiado después del encendido del módulo USB, se procede a limpiar las interrupciones del mismo, a enmascarar las interrupciones del módulo USB, desenmascarar las interrupciones de *Reset* y de *IDLE*. Ya que se realizó lo anterior, se cambia el estado actual a estado_polarizado (*POWERED_STATE*).

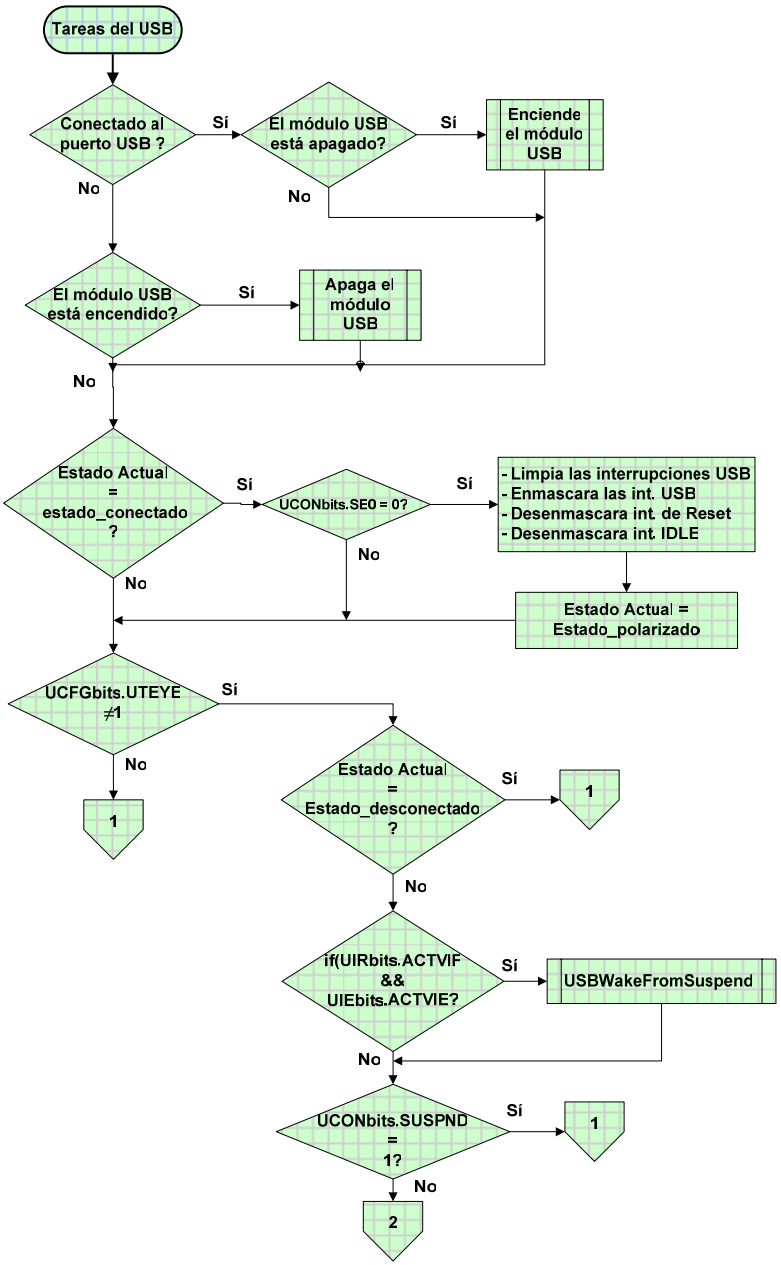


Figura 7. Primera parte del diagrama de la función "Tareas del USB" (*USBTasks).*

En seguida, el microcontrolador verifica que el bit UTEYE del registro UCFG no esté activado, ya que éste controla un modo de prueba el cual no se debe activar mientras el modulo USB se está utilizando [25].

Después, se verifica si el estado actual es el estado_desconectado (DETACHED_STATE), en el

caso de que no lo sea se verifican si hay actividad en el bus, asegurándose de que los bits ACTIVF del registro UIR (UIRbits.ACTIVF) y ACTIVE del registro UIE se encuentran activos (en estado lógico “1”). En el caso de que no hubiese actividad en el bus, el microcontrolador entra en la función USBWakeFromSuspend, la cual lo saca de modo de suspensión. Entonces si el puerto USB se encuentra activo, el siguiente paso, que se muestra en la figura 8, es verificar si el *hub* raíz ha mandado una señal de reset. De ser así, el microcontrolador entra a su subrutina de atención a un *reset* del sistema, entonces el dispositivo reinicia su dirección a cero, deshabilita todos los *endpoints* excepto el cero (EP0), inicializando el EP0 para las transmisiones iniciales, reinicia las interrupciones del puerto, desenmascara las interrupciones a usar y reinicia las variables internas de la máquina de estado. En seguida se verifica si el bus está en espera (*IDLE*), de ser así, el microcontrolador suspende toda actividad en el puerto y pone al módulo USB en suspensión [24], [25].

En seguida, el microcontrolador verifica si se ha recibido un SOF (o inicio de marco), los cuales son enviados por el *host* para los dispositivos USB *full speed* cada 1ms. Si se detecta, el microcontrolador ejecuta la función *USB_SOF_Handler*, en la cual el diseñador puede introducir las acciones que requiera ante tal evento. Después se verifica si algún *endpoint* en el microcontrolador está en *stall*, es decir, si se ha detenido. De ser así, el microcontrolador entra a la función *USBStallHandler*, la cual se encarga de esta situación. También, monitorea a los bits UERRIE del registro UIE y UERRIF del registro UIF; en caso de que ambos estén activos, significa que el microcontrolador detectó un error en el puerto USB, entonces entra en la función *USBErrorHandler* para atender dicho problema. Esto se usa principalmente para depurar el programa durante la etapa de diseño [1], [2], [8], [24], [25]. En seguida se verifica si el estado actual es menor al estado DEFAULT, esto porque un estado menor significaría que el microcontrolador no está listo para iniciar la comunicación en el puerto. En el caso de que el microcontrolador se encuentre listo, se verifican los bits TRNIF del registro UIR y TRNIE del registro UIE, para detectar si hay alguna transacción pendiente; de ser así, se verifica el *USB Status*

Register (USTAT), el cual registra el estado de las transacciones, para saber si se recibió alguna comunicación del EP0; si es el caso, se verifica si la identificación del paquete recibido (*Packet Identifier*) corresponde con un paquete de configuración (*Setup Token*) mandado por el *host*. En el caso de que sean iguales, se maneja en la función USBCtrlTrfSetupHandler, la cual se encarga de verificar si puede manejar la petición del *host* a través del EP0 y después realiza las acciones necesarias de acuerdo a lo requerido por el *host*. En el caso contrario, la transacción se maneja por la función USBCtrlTrfOutHandler, para manejar las transacciones de salida [24], [25].

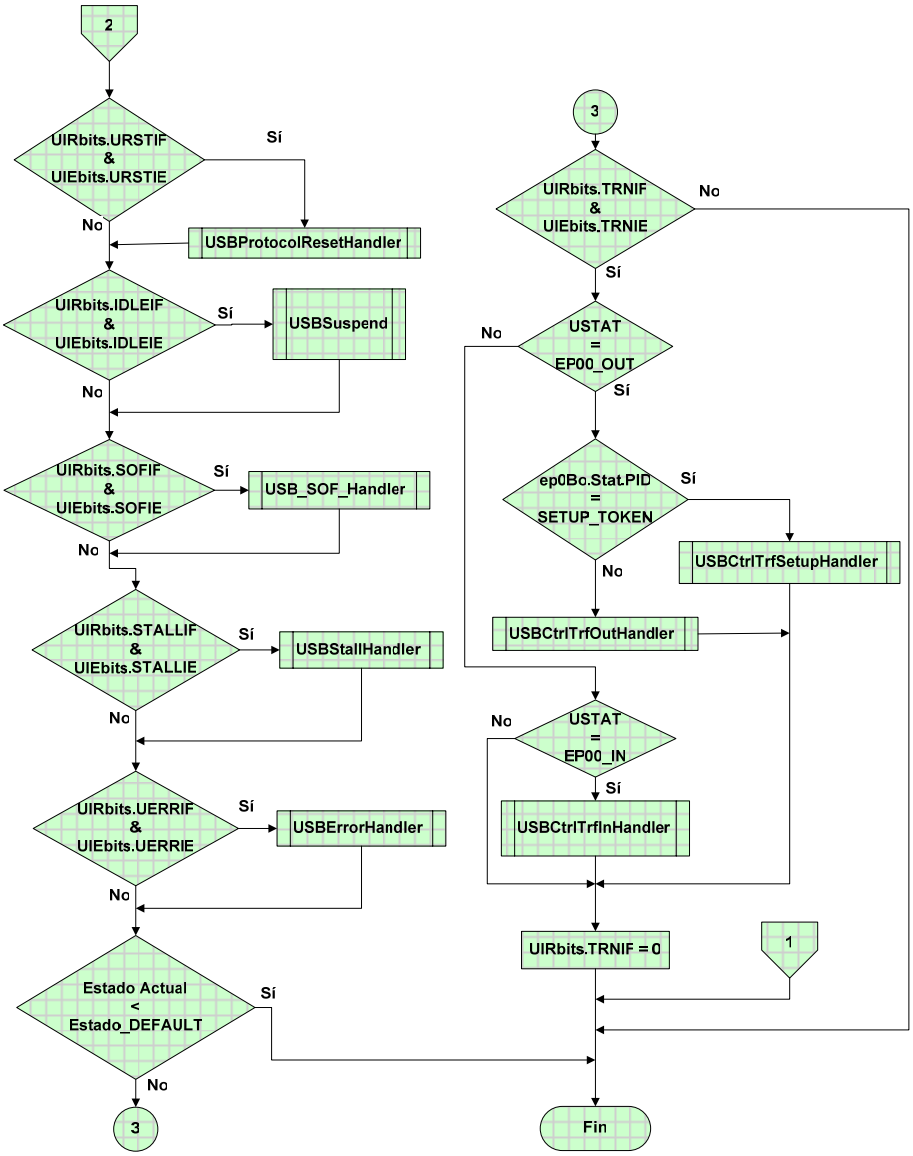


Figura 8. Segunda parte del diagrama de la función *USBTasks.*

Regresando un poco, si el microcontrolador verifica que no se recibió una comunicación del EP0, entonces procede a ver si hay en progreso una transmisión hacia el *host*, esto nuevamente revisando el USTAT, de ser así, se entra a la subrutina que maneja las transacciones hacia el *host* por medio del EP0, *USBCtrlTrfInHandler*.

Finalmente, se limpia la interrupción TRNIF en el registro UIR, indicando así que no hay transacciones pendientes [24], [25].

Aunque parezca algo compleja la descripción que se dio de los diagramas 7 y 8, hay que recordar que son simplificaciones de las funciones que realiza el microcontrolador, pero son necesarias debido a que nos describe un panorama de cómo el microcontrolador maneja las etapas iniciales de la comunicación con el *host*, a través del puerto USB.

2.6. ADAPTACIÓN DEL FIRMWARE PARA UN PROYECTO ESPECÍFICO

2.6.1. Introducción

Para este trabajo, fue necesario escoger un ejemplo para mostrar la forma en que se puede integrar el diseño realizado, a un proyecto desarrollado por otra persona.

Como estudio de caso o aplicación se decidió utilizar un grupo de termómetros para adquirir datos con el microcontrolador para después ser enviados por el puerto USB a la computadora, donde serán procesados.

En la figura 9 se muestra el diagrama de bloques del sistema al que se pretende implementar.

Este ejemplo por si solo es de gran utilidad, ya que el monitoreo de temperatura tiene un amplia variedad de aplicaciones, como puede ser el control ambiental, control de temperatura de procesos como el termoformado o incluso en prácticas

de laboratorio. Sin embargo, lo más importante de esta aplicación es que sirve de ejemplo para aquellos que deseen implementar su propio sistema utilizando la plataforma mostrada en este trabajo.

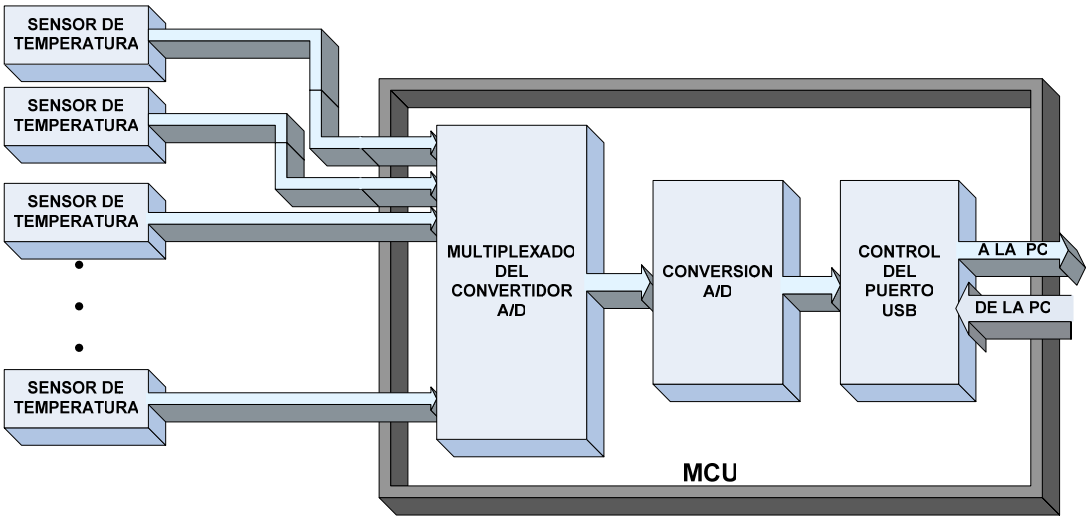


Figura 9. Sistema de monitoreo de temperatura.

2.6.2. Especificaciones del sistema de aplicación

Como lo que se desea es un sistema de ejemplo, y no se pretende resolver un problema específico, este se diseñó de la forma más general posible. Para la implementación de los termómetros se optó por utilizar sensores semiconductores. El circuito integrado escogido fue el TMP35 de *Analog Devices*, el cual entrega a la salida $10\frac{mV}{^{\circ}C}$ [13], [28].

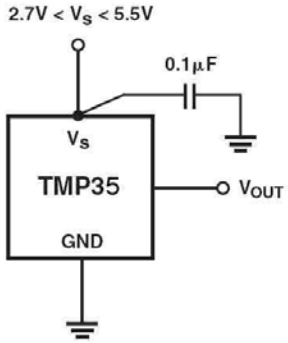


Figura 10. Aplicación típica.

El sistema contará con 6 sensores de temperatura, los cuales serán conectados al microcontrolador en los puertos de su convertidor A/D, de tal forma que al obtener un comando del software de la computadora, éste comenzará a adquirir 6 lecturas de cada sensor, multiplexando los canales de cada uno. Además se puede diseñar el sistema, de tal forma que junto con el comando de inicio, se envíe el periodo de muestreo que se seleccionó en la interfaz de usuario, para que el microcontrolador modifique la frecuencia con la que adquiere datos de los sensores, para así asegurar una mejor precisión.

2.6.3. Diseño del *firmware* del sistema de aplicación

Para iniciar el diseño del *firmware*, se decidió partir del programa HID3, ya que este ya se había probado. Para poder llegar a la solución final, fue necesario hacer varias versiones del *firmware*, donde en cada una de ellas se le añadían algunas características, hasta que se llegó a la versión HID9, la cual ya tiene varias modificaciones del framework. En la figura 11 se muestra el diagrama de flujo de la función *ReportLoopback* de la versión HID9 del *firmware*. Si se compara esta versión con la que se muestra en la figura 6, se puede observar que la adquisición de datos ya no se hace en esta función, en vez de esto, la única función que se realiza es la de recibir comandos del *host*, interpretarlos, y después ejecutarlos junto con cualquier dato adicional que se halla

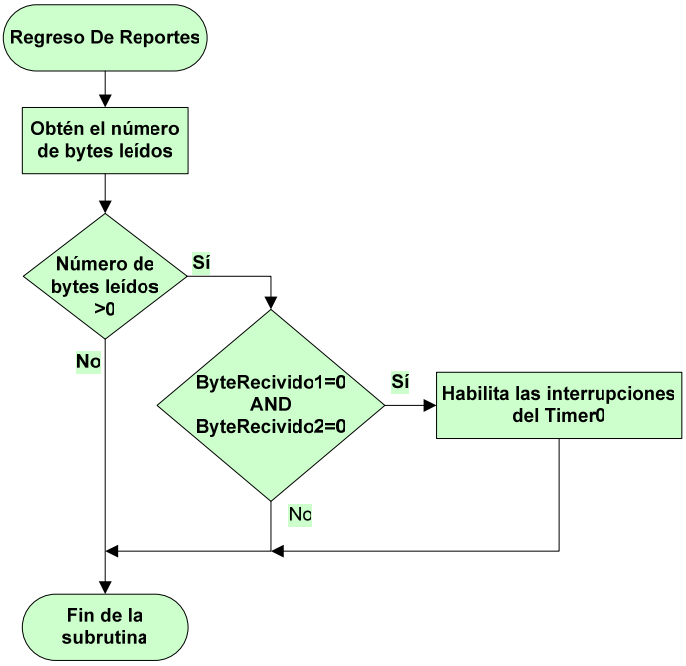


Figura 11. Diagrama de la función “Regreso De Reportes”, de la versión HID9.

transmitido. En este caso, el comando a ejecutar solamente es habilitar la interrupción del *Timer0*, para que el microcontrolador comience a obtener datos con cada desbordamiento del reloj [24].

Debido a que el sistema adquirirá temperatura, se decidió hacer el tiempo mínimo entre muestra y muestra de 1s. Esto debido a que las frecuencias con las que varía un sistema térmico son bajas. Sin embargo, de ser necesario el sistema puede transmitir datos con una velocidad máxima de 64,000 $\frac{Bytes}{s}$, que representa el cuello de botella del sistema [1], [2].

Una vez que en la función “Regreso De Reportes” se activó la adquisición de datos, la subrutina de atención a interrupciones del Timer0 se encarga de adquirir datos y programar su envío por el puerto USB. Dicha rutina se muestra en el diagrama de flujo de la figura 12 [24].

Debido a que se escogió como tiempo mínimo entre adquisiciones 1 segundo, el *Timer0* se programa de tal forma que se desborde cada segundo, esto por medio de los pre-escaladores del mismo contador, que nos permite dividir el reloj que usa por potencias de 2, hasta 256, además de el control del número de cuentas del mismo [24].

Ante la presencia de un desbordamiento, el microcontrolador pasa el control a la subrutina de atención a interrupciones “Timer_isr”, la cual se muestra en la figura 12. Esta subrutina inicialmente utiliza un contador que llamamos “basesT”, el cual representa el número de segundos que se desea como tiempo entre adquisiciones. Después, el microcontrolador verifica si el Timer0 se ha desbordado “X” número de veces, de ser así, el microcontrolador procede a adquirir los datos, de lo contrario se limpia la bandera de interrupción y se sale de la subrutina. Una vez que se alcanza el número requerido de desbordamientos, la subrutina multiplexa entre los N canales del convertidor A/D que se quieran usar, adquiriendo datos de 10 bits, dividiéndolos en dos *bytes* y guardándolos en el

buffer de transmisión. Después sólo es cuestión de transmitir los datos del *buffer*, por medio de las funciones ya implementadas en el *framework* [24].

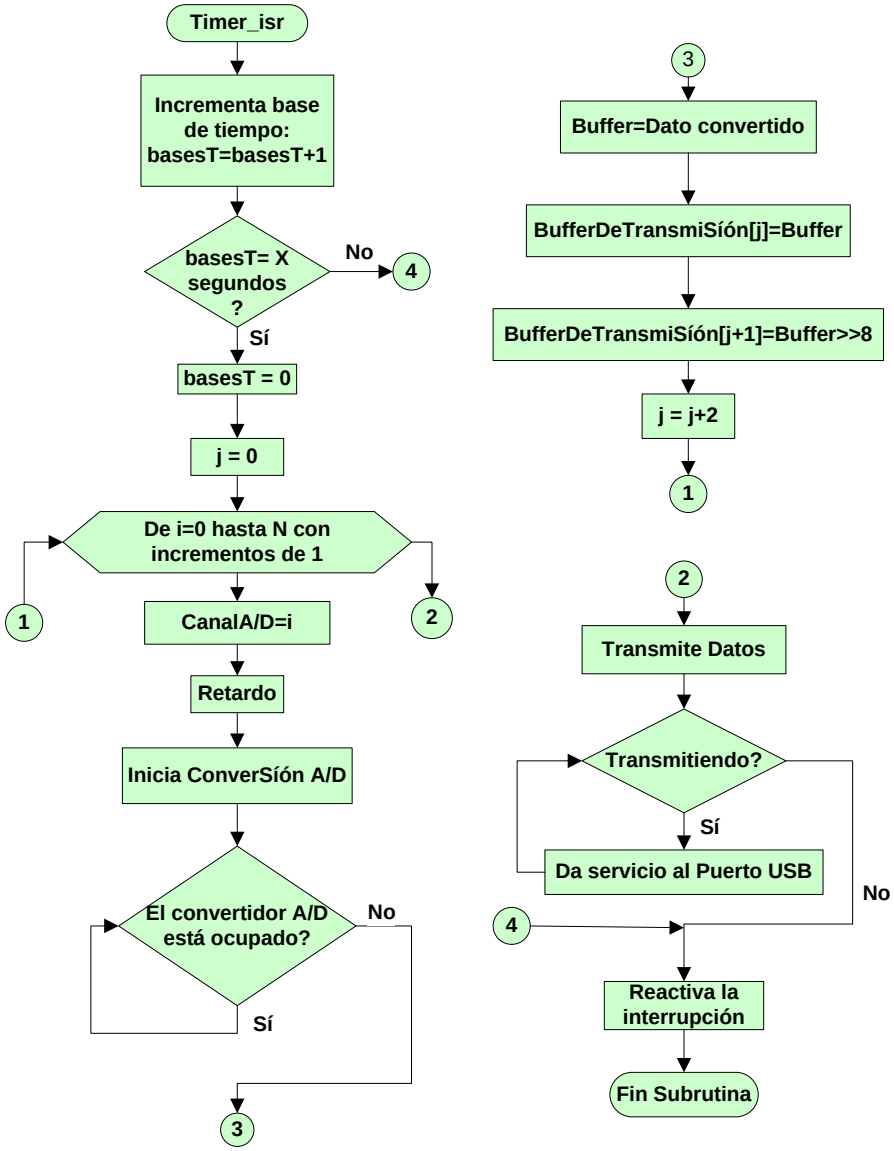


Figura 12. Subrutina de atención a la interrupción del Timer0.

Con esta versión del *firmware* funcionando, se implementó un sistema de adquisición de temperatura, de múltiples canales, por lo que sólo restaba terminar de implementar la interfaz de usuario, específica de esta aplicación.

CAPÍTULO III: DISEÑO DEL *SOFTWARE* PARA LA PC

Para este proyecto fue necesario desarrollar prácticamente de forma simultánea tanto al *software* como al *firmware* de la interfaz, ya que para cada avance en el extremo del microcontrolador era necesario rediseñar la interfaz para poder probar el estado de la comunicación USB.

3.1. DISEÑO BASE

La aproximación que se decidió usar para este proyecto, fue la de utilizar un ejemplo como base para el desarrollo de la interfaz. Una de las ventajas de esto, es que al no partir desde cero, el tiempo de desarrollo de esta etapa se ve reducido.

Para empezar el desarrollo de la interfaz era necesario en primer lugar escoger la plataforma y el lenguaje de programación a usar. Para esto se debía tener en mente que la interfaz a desarrollar debiera operar bajo el sistema operativo *Windows*, ya que se trata del sistema utilizado en el Laboratorio de Electrónica. Esto, a pesar de que existen otros sistemas operativos, los cuales podrían reducir los costos de desarrollo e incluso más adelante el costo para el usuario, como por ejemplo LINUX.

El escoger el lenguaje de programación fue una tarea relativamente rápida, debido a que se requería un lenguaje que fuera fácil de usar, con el cual el diseñador tuviera experiencia y que permitiera diseñar aplicaciones basadas en *Windows*. La opción obvia fue *Visual Basic .NET*, no sólo por las razones anteriores, sino porque en le Laboratorio de Electrónica se cuenta con experiencia en el desarrollo de interfaces con este software, así como una amplia cantidad de información relacionada al mismo.

A pesar de que el lenguaje de programación escogido cuenta con un amplio número de herramientas integradas en el mismo ambiente de programación, inmediatamente se notó una dificultad, que el Visual Basic no es capaz de acceder directamente a los puertos USB. Las única forma de acceder a los puertos utilizando las herramientas incluidas en el paquete de Microsoft, es mediante llamadas directas a funciones API (*Application Programmer's Interface*), o por medio de un control de ActiveX, el cual facilita el proceso, pero que resulta menos flexible [2], [22], [6].

3.2. DISEÑO BASADO EN CONTROLES *ACTIVEX*

Existen varios tipos de controles de *ActiveX* en el mercado, que tienen como propósito facilitar el desarrollo de ciertas aplicaciones, como por ejemplo el acceso de datos a través de los puertos de la computadora. Un ejemplo de este tipo de objetos es el conocido MSCOMM, que hasta el Visual Basic 6 era la forma más fácil para establecer comunicación por medio del puerto serie de una PC [4], [6], [7], [22], [27], [30].

Con el auge de la comunicación por puerto USB, los distintos fabricantes de soluciones para esta área desarrollaron controles del tipo del MSCOMM, para el uso con el puerto USB. Sin embargo, debido a la gran diversidad de tipos de dispositivos que se pueden comunicar por este medio y considerando que la gran mayoría de ellos requieren de un *driver* específicamente diseñado para la aplicación, es prácticamente imposible desarrollar un control para todo tipo de comunicaciones por puerto USB. Lo que si lograron crear fueron controles para aplicaciones que utilizaran los *drivers* incluidos con el sistema operativo, como es el de HID. Un ejemplo de estos controles creados para dispositivos de interfaz humana, es el HIDComm, el cual fue desarrollado por Microchip para ser usado en aplicaciones de Visual Basic 6, de preferencia con su primer microcontrolador USB el PIC16C765 [22], [27].

Debido a que el control HIDComm era distribuido de forma gratuita, se decidió importarlo a un proyecto de Visual Basic .NET, para ser usado como una posible solución para la interfaz de usuario. Esto permitiría desarrollar una aplicación funcional en poco tiempo [4], [7], [30].

3.2.1. Desarrollo de la interfaz basada en el control HIDComm de ActiveX

3.2.1.1. Versión HIDComm1

Inicialmente se necesitaba un programa que permitiera detectar el dispositivo *HID* diseñado por medio de su *VID* (*Vendor Identification*) y su *PID* (*Product Identification*), para después enviar dos bytes y recibir dos bytes. Esto para poder desarrollar a la par de la primera versión del *firmware* de Jan Axelson, el cual ya había sido probado [22].

Para esto se desarrolló una interfaz sencilla, la cual se muestra en la figura 13. Se puede observar que la primera versión de la interfaz basada en el control de *ActiveX* contaba con dos casillas de texto para poner los datos a enviar, el control *HIDComm* para manejar la comunicación con dispositivos *HID*, un botón para enviar los datos y una etiqueta para desplegar los datos recibidos.

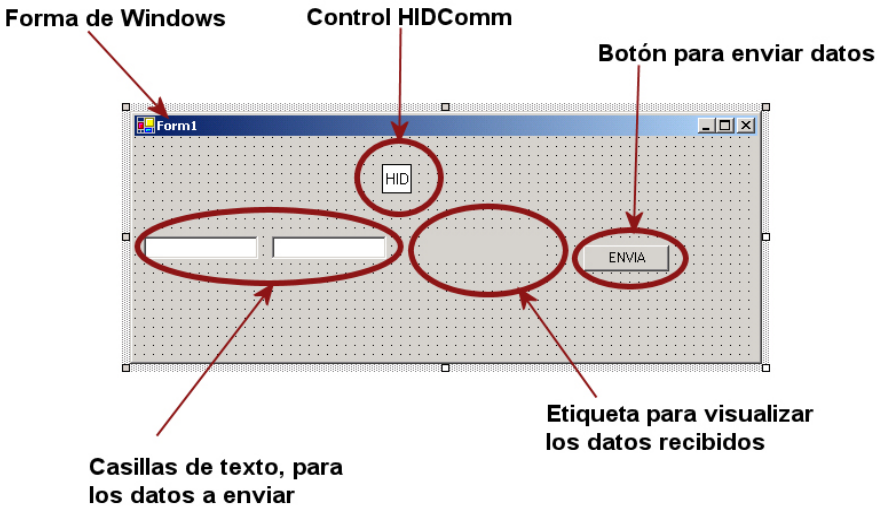


Figura 13. Diseño de la interfaz HIDComm1.

La interfaz diseñada HIDComm1 se utilizó con un firmware ya probado, el cual se mencionó anteriormente, y se trata de la versión HID2 proporcionada por Jan Axelson.

La configuración de la interfaz de la figura 13 se inicia con la definición de las propiedades del control *HIDComm*, esto desde el Visual Studio, a través de las propiedades del control, como se muestra en la figura 14 [4], [7], [30].

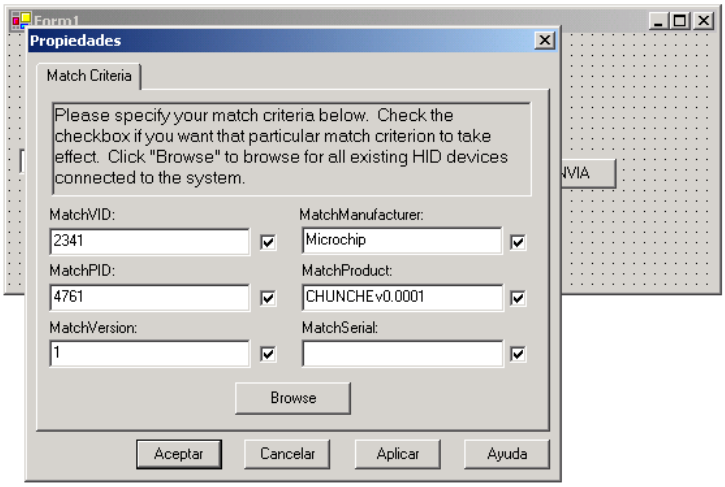


Figura 14. Propiedades del control HIDComm.

Una vez configurado el control HIDComm, se pueden utilizar las diferentes funciones del control, para detectar el estado del dispositivo HID y para enviar y recibir datos del mismo.

El código necesario para que la interfaz se conecte, envíe y reciba los datos del dispositivo HID diseñado se muestra en la figura 15, donde se puede ver la función que responde al evento generado al apretar el botón “Enviar”. Este código es mucho más reducido que el necesario para controlar un dispositivo HID por medio de llamadas a funciones API, como se mostrará más adelante. Al apretar el botón enviar, se ejecuta la función mostrada en la figura 15, lo primero es declarar variables para después conectarse con el dispositivo HID por medio de la función

"*AxHIDComm1.Connect()*", donde *AxHIDComm1* es el nombre del control de *HIDComm* dentro del programa.

```
Private Sub btEnvia_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles btEnvia.Click
    Dim Buffer() As Byte
    Dim BufferSize As Long
    ReDim Buffer(16)
    AxHIDComm1.Connect()
    Buffer(0) = CByte(txtByte1.Text)
    Buffer(1) = CByte(txtByte2.Text)
    BufferSize = 2
    AxHIDComm1.WriteTo(Buffer, BufferSize)
    Buffer = AxHIDComm1.ReadFrom(BufferSize)
    If BufferSize < 2 Then Exit Sub
    Label1.Text = Buffer(0) & Buffer(1)

End Sub
```

Figura 15. Código de la interfaz HIDComm1.

Dicha función es parte de un grupo de comandos con los que cuenta el control. En seguida, se cargan los valores que el usuario introducirá a las casillas de texto, en el buffer de datos de salida "*Buffer()*", dichos datos deben ser números enteros de 0 a 255, y considerando que no se implementó código para asegurarse que el usuario no introduzca caracteres inválidos, es importante tener cuidado en este punto. A continuación, se define el número de datos a transmitir "*BufferSize* = 2", para después transmitirlos utilizando el comando del *HIDComm* "*AxHIDComm1.WriteTo(Buffer, BufferSize)*". Por último, se intenta leer dos bytes del microcontrolador, por medio del comando "Buffer = AxHIDComm1.ReadFrom(BufferSize)", para después desplegarlos en la etiqueta de esta versión de la interfaz [4], [7], [30].

El siguiente paso es adquirir datos de forma más continua, pensando en aplicaciones donde es necesario recolectar datos de forma periódica.

3.2.1.2. Versión HIDComm2

En la versión HIDComm2, como se puede observar en la figura 16, se conserva el control HIDComm; sin embargo, esta interfaz de prueba se diseñó para poder

mandar comandos y recibir datos de forma continua. Para poder realizar esta actividad se añadió un reloj, el cual se encarga de ejecutar una subrutina cada X milisegundos, de esta forma, el usuario puede indicar por medio de la casilla numérica, el tiempo transcurrido entre muestra y muestra, es decir, el periodo de adquisición.

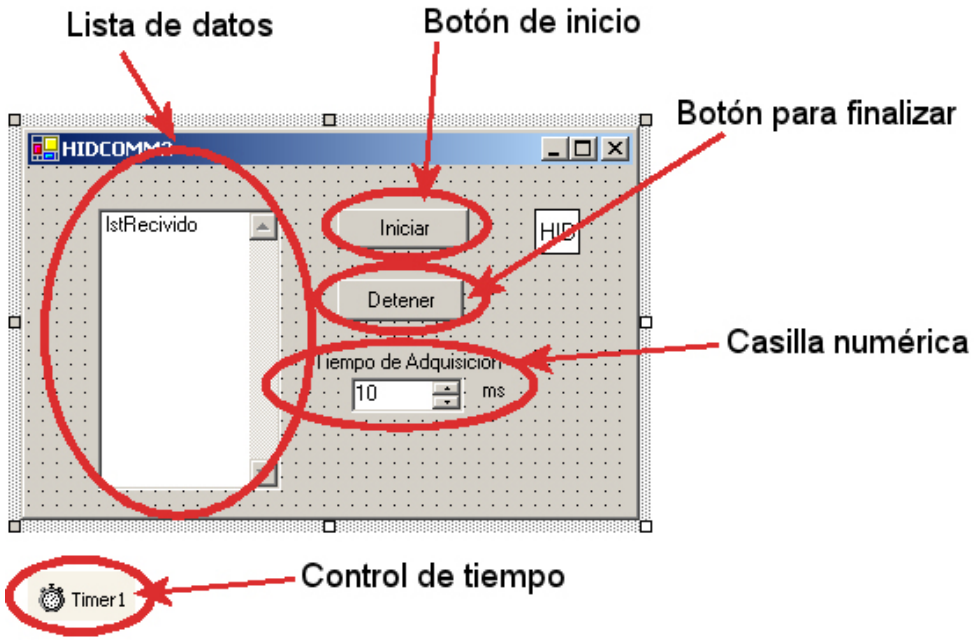


Figura 16. Interfaz HIDCOMM2.

La interfaz HIDComm2 se diseño a partir de HIDCOMM1, casi a la par del desarrollo de los programas para el microcontrolador. Se desarrolló para que interactuara con el *firmware* HID3, del cual se habló en el capítulo 2, y que cuenta con una rutina que acepta un par de *bytes* del *host* y que son interpretados como comandos.

En la figura 17 se muestra el diagrama de flujo de la función que se ejecuta al oprimir el botón “Iniciar”, de la interfaz que se muestra en la figura 16. La función que se ejecuta se llamada *btIniciar_Click*, así como, el diagrama de flujo de la función que se ejecuta cada X milisegundos, Timer1, llamada *Timer1_Tick*. Al ejecutarse la función btInicia_Click, se asigna el intervalo de tiempo para el control Timer1 en milisegundos, a partir del tiempo de adquisición indicado por el usuario,

en la casilla numérica. Después de indicar el tiempo de repetición del evento, se habilita el Timer1 para que se empiece a ejecutar la función *Timer_Tick*. Y por último, se deshabilita el botón “Iniciar” y se habilita el botón “Detener”.

La función Timer1_Tick se ejecuta cada X milisegundos, en cuanto se habilita, tal como se mencionó anteriormente. Lo primero que hace después de inicializar algunas variables locales, es conectarse al dispositivo HID, entonces carga dos ceros a las variables a enviar *Buffer(0)* y *Buffer(1)*, los cuales serán interpretados como comandos por el microcontrolador, para que inicie la adquisición y envío de datos. Después de enviar los dos ceros, se procede a leer dos *bytes* del HID, los cuales representan un dato de 10 bits, por lo cual antes de desplegarlos en la lista de la interfaz, se deben de unir, para recuperar el dato original. Esto último, finaliza la subrutina del Timer1.

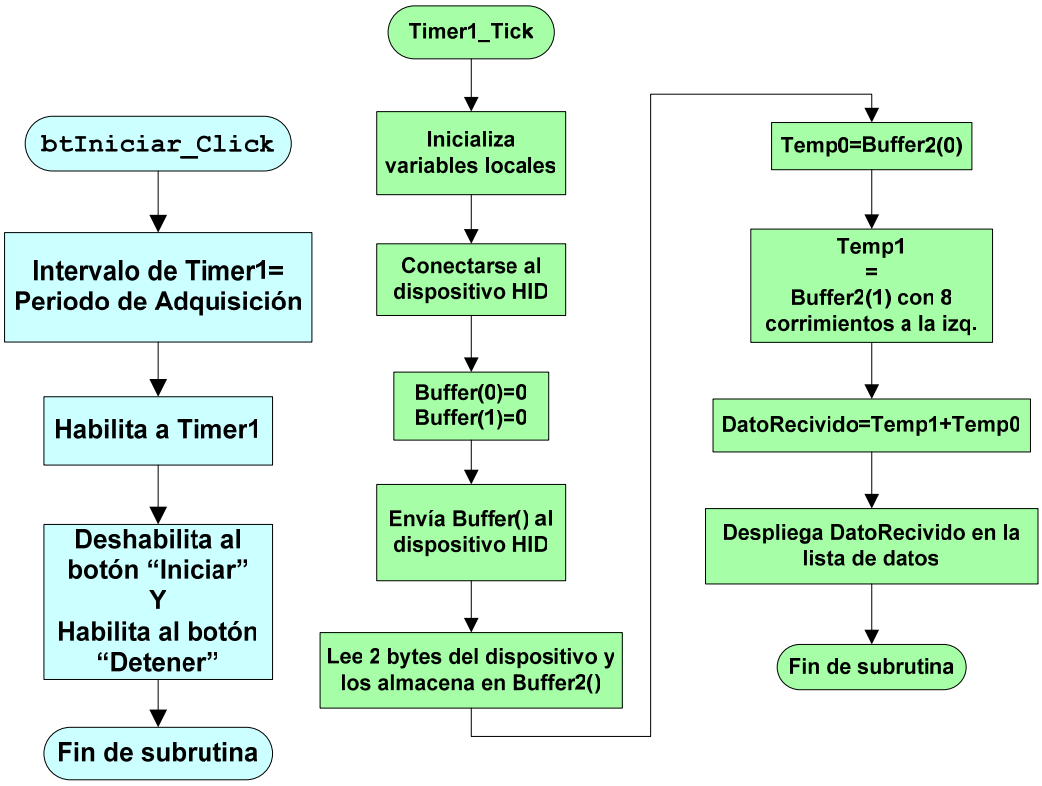


Figura 17. Funciones *btIniciar_Click* y *Timer1_Tick de HIDCOMM2.*

Las dos subrutinas mostradas en la figura 17, son las principales de la interfaz HIDComm2, y aunque si se adquirían datos, la velocidad de transmisión de datos apenas alcanzaba los 200 $\frac{bytes}{s}$. Debido a que era una velocidad baja, este problema se solucionó analizando el protocolo por medio de un programa para la PC, llamado "*HHD USB monitor*", el cual se encarga de leer y traducir todas las transacciones que ocurren en el puerto. En la siguiente figura, se muestra un análisis del protocolo hecho a esta versión de la interfaz [1], [2].

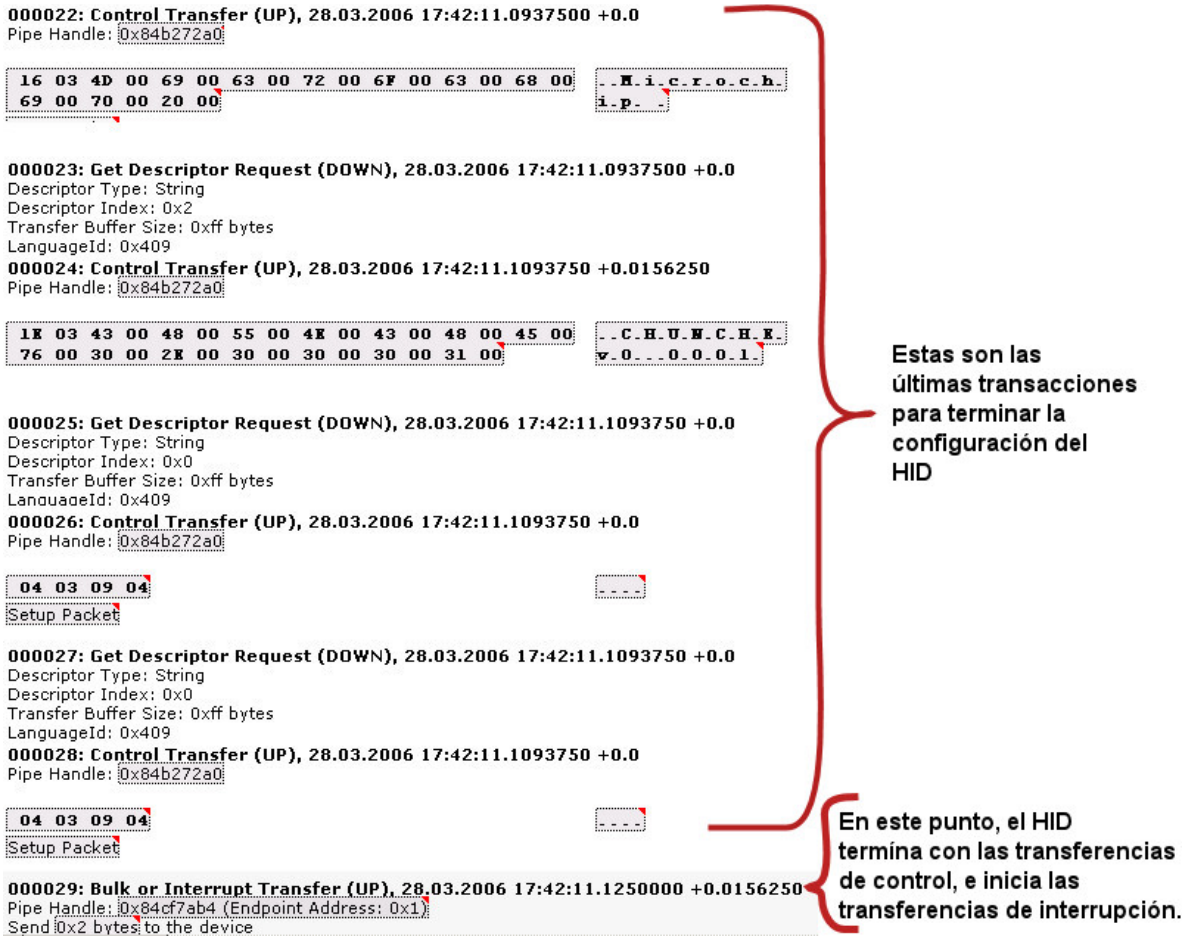


Figura 18. Análisis de la comunicación entre el *software* HIDCOMM2 y el *firmware* HID3.

En la figura 18, se muestran las transacciones de la 22 a la 29, en éstas se muestra la parte final de la enumeración de nuestro HID. De la transacción 29 en

adelante, el diseño supuestamente debería manejar únicamente transacciones de interrupción; sin embargo, como se aprecia en la figura 19, que es la continuación del análisis de la figura 18, después de recibir los datos del HID en la transacción 30, el dispositivo comienza a reenviar parte de sus descriptores. Esto consume parte del ancho de banda del bus, lo cual explicaba la inhabilidad del diseño de acercarse a las tasas máximas de transferencia para un HID [1], [2], [8].

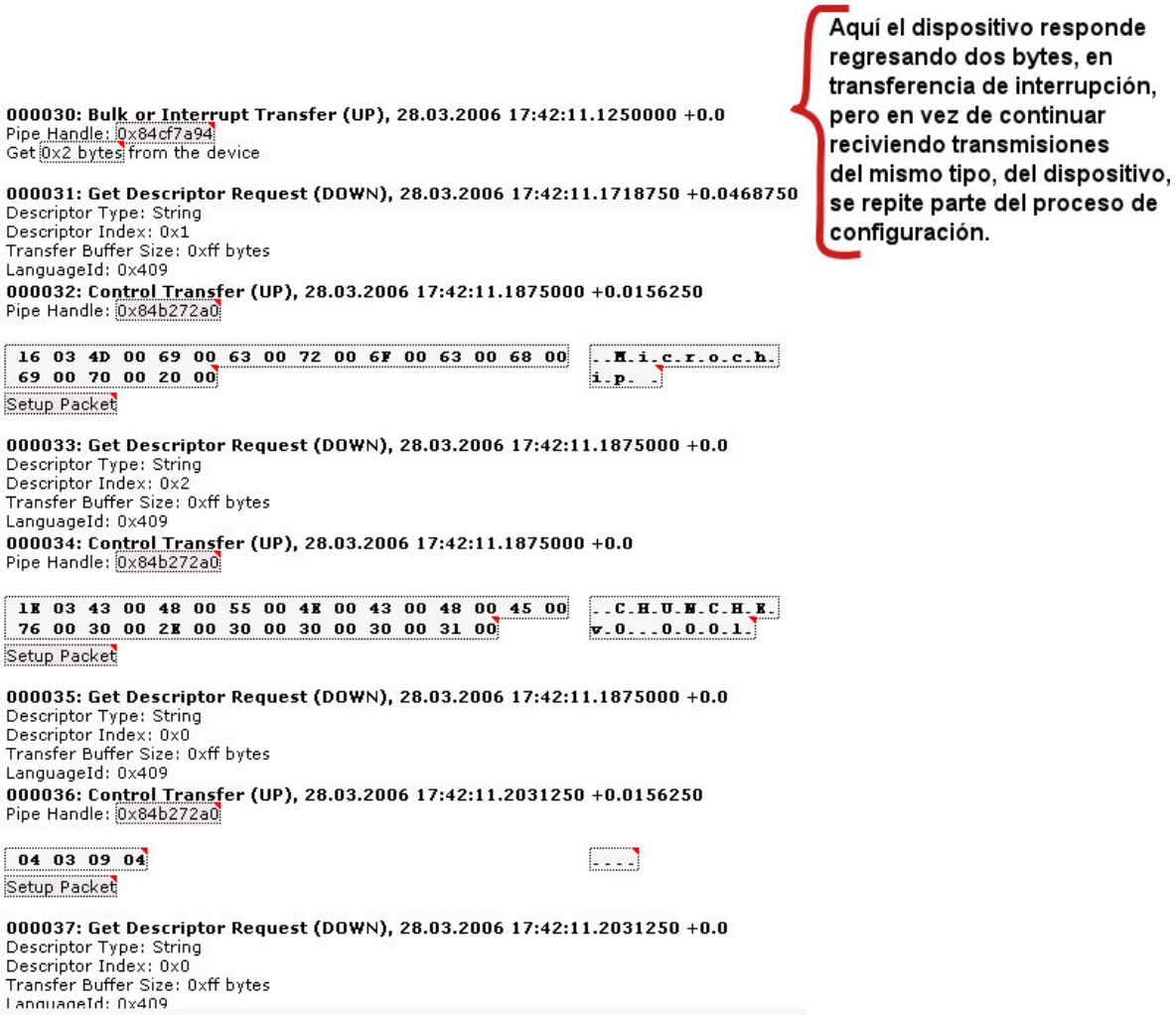


Figura 19. Análisis de la comunicación entre el *software* HIDCOMM2 y el *firmware* HID3 (segunda parte).

Analizando el problema se descubrió un error en la lógica del programa interfaz. Revisando el diagrama de flujo de la figura 17, podemos observar que en la función *Timer1_Tick*, se colocaron las funciones necesarias para conectarse al HID y las funciones para enviar los dos *bytes* igual a cero. Esto resultó ser la falla

que no permitía aumentar la velocidad de transferencia de datos, ya que con cada ciclo del Timer1, el dispositivo recibía de parte del *host* la petición de sus descriptores, como si el HID acabara de conectarse. Además, con cada ciclo, el *host* mandaba los mismos dos bytes en ceros, lo cual era innecesario, ya que la idea de mandar estos datos al dispositivo era para que iniciara la transmisión de datos continua.

Cabe mencionar que antes de descubrir la falla en el software HIDCOMM2, ya se había desarrollado una siguiente versión HIDCOMM3, que tenía la misma falla, pero que fue creada para desarrollar la capacidad de guardar datos en una hoja de cálculo de Excel.

Una vez detectados los errores en el software, se decidió desarrollar una siguiente versión para corregir los mismos, por lo que pasamos a la versión HIDCOMM4.

3.2.1.3. Versión HIDComm4

Partiendo de la versión HIDCOMM3, se decidió que además de corregir los errores ya detectados, se incorporaría al diseño una forma de desplegar los datos que fuese visualmente más amigable. Debido a esto se incorporó un control graficador diseñado por *National Instruments*, el cual forma parte de un software llamado *Measurement Studio*. En la figura 20 se muestra el diseño de la interfaz con los controles que la forman. Se puede observar que los controles son prácticamente los mismos a la versión anterior, lo único que cambia en el diseño es la forma en que se despliegan los datos; sin embargo, eso no es lo más importante, ya que cada diseñador debe decidir como desplegar o guardar los datos con los que está trabajando. La diferencia viene en realidad en el código, en donde se corrige el primer error y en realidad el más significativo de los mencionados para la versión HIDCOMM2, en el cual el *host* pide repetidamente al HID que se conecte.

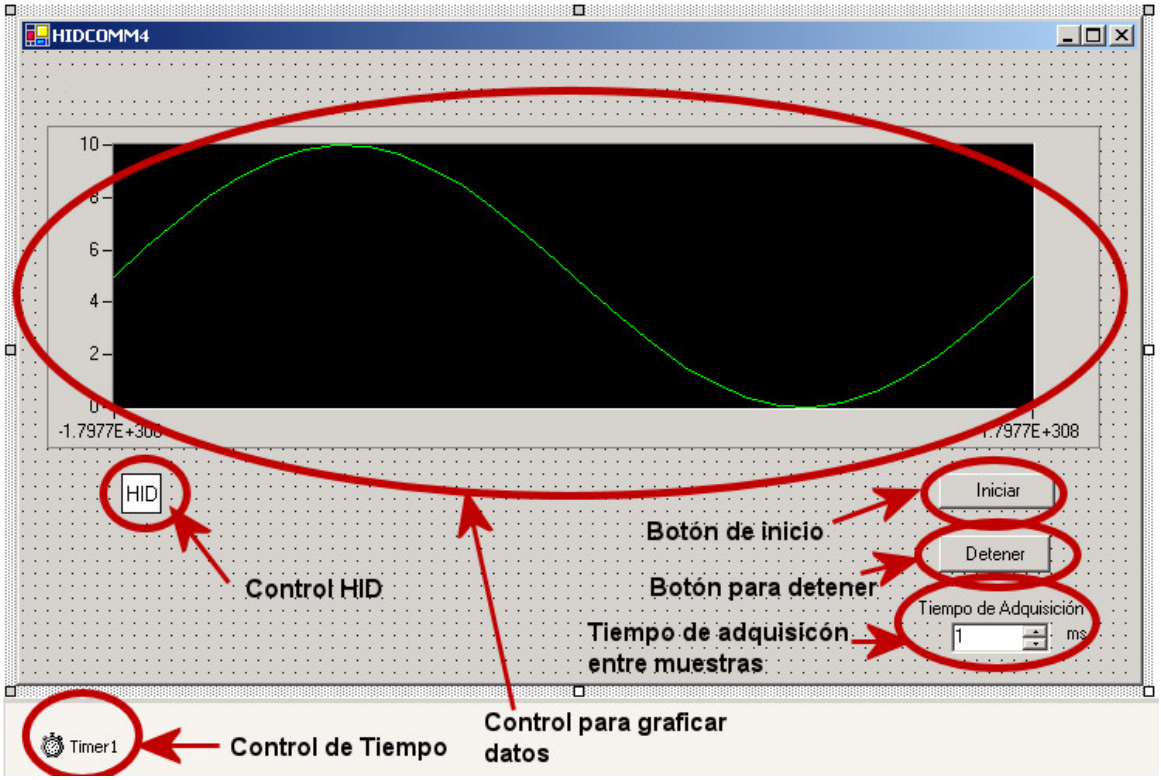


Figura 20. Interfaz HIDCOMM4.

Si comparamos las mismas funciones de la versión HIDCOMM2 y HIDCOMM4, en las figuras 17 y 21 respectivamente, podemos ver que la diferencia radica en la instrucción de conectarse al HID. Ahora en la figura 21 se puede ver que la conexión al HID se realiza en el evento generado por el botón “Iniciar”, en vez de en la función *“Timer1_Tick”*.

Observando el análisis del bus en la figura 22, con el mismo programa que se mencionó anteriormente, se puede observar que la transmisión se vuelve puramente de interrupción, en vez de mezclarse con la de control, tal y como pasaba con la versión anterior. Entonces, a pesar de que no se modificó el error en el que se envían los dos bytes con cada evento del Timer1, lo más importante es que la transmisión y recepción de datos funciona de forma adecuada.

A pesar de que hasta este punto ya se cuenta con una comunicación adecuada, por medio del programa “*HHD USB Monitor”*, se descubrió que la velocidad de

transferencia máxima continua, a la que el HID se comunicaba, era de 2 *bytes* por cada 10ms, es decir, 200 $\frac{bytes}{s}$, cuando en las especificaciones para el puerto USB, se dice que un HID es capaz de transmitir a un máximo de 64,000 $\frac{bytes}{s}$.

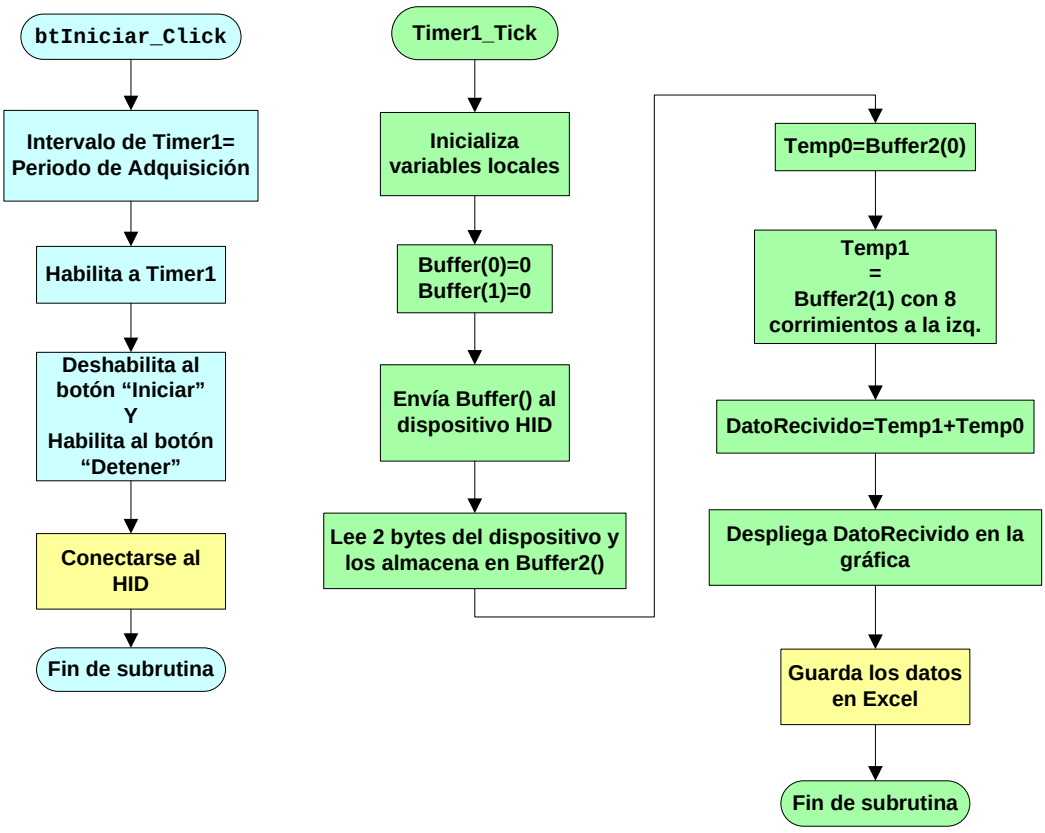


Figura 21. Funciones *btIniciar_Click* y *Timer1_Tick de HIDCOMM4.*

Con base en lo anteriormente descrito, se decidió que, además de probar con controles de ActiveX, se debería utilizar un programa para PC que fuese basado en llamadas a funciones API. De esta forma podríamos descartar al control de ActiveX como la causa de la limitada velocidad de comunicación por el puerto USB. Sin embargo, es importante mencionar que sin importar si el control HIDComm es el responsable por la baja velocidad del bus, aún así presenta una buena alternativa para aplicaciones que requieran bajas frecuencias de

transmisión, como el ejemplo que planteamos en este trabajo, principalmente por la facilidad de uso.

Figura 22. Análisis de la comunicación entre el HID con el firmware HID3 y el programa interfaz HIDCOMM4.

3.3. DISEÑO BASADO EN LLAMADAS A FUNCIONES API

3.3.1. Desarrollo de la interfaz basada en llamadas a funciones API

Para explorar esta alternativa, al uso de controles pre-diseñados de *ActiveX*, fue necesario encontrar un programa base, tal como se hizo con la serie de versiones HIDCOMMx.

Como se mencionó brevemente en la sección 2.3, Jan Axelson puso a disposición del público en general, un ejemplo de una interfaz que se comunicaba con un HID, por medio de llamadas directas a funciones de interfaz a aplicaciones para programadores o API. El programa original se encargaba de conectarse a un dispositivo de interfaz humana, detectándolo por medio de su PID y su VID, para de esta forma poder enviar un par de *bytes* al HID para que éste regresara otro par.

Lo importante del programa mencionado es que tenía implementadas las funciones necesarias para establecer una comunicación continúa.

En la siguiente sección se describe la primera versión de la interfaz basada en el programa de Jan Axelson.

3.3.2. Versión USBHIDIO

El desarrollo de las versiones de esta interfaz fue muy similar al del programa basado en el control HIDCOMM, sin embargo, éstas están diseñadas con base en llamadas a funciones API, como se mencionó anteriormente.

En la figura 23, se muestra la interfaz diseñada. En ella se puede observar que es muy parecida a la hecha con el control de ActiveX. Esto es porque el resultado deseado por ambos caminos es el mismo, por lo que si queremos hacer una comparación válida, es importante mencionar que la única diferencia entre las versiones HIDCOMM y las USBHIDIO es la forma en que están implementadas las funciones que manejan la comunicación con el HID [22].

En la figura 23 se puede observar que la mayoría de los controles son los ya antes usados en las versiones del software HICOMM, la principal diferencia en este caso es la lista de texto *lstResultados*, en la cual se despliega el estado de las transmisiones hechas por el *host*.

En realidad, ambas soluciones implementan el mismo diseño, lo único que cambia son las funciones para el control y monitoreo de los HID. A pesar de este hecho, se pudo comprobar que el programa basado en llamadas a funciones API, podía alcanzar velocidades de transmisión sostenidas de hasta $20\frac{bytes}{ms}$, o $20{,}000\frac{bytes}{s}$.

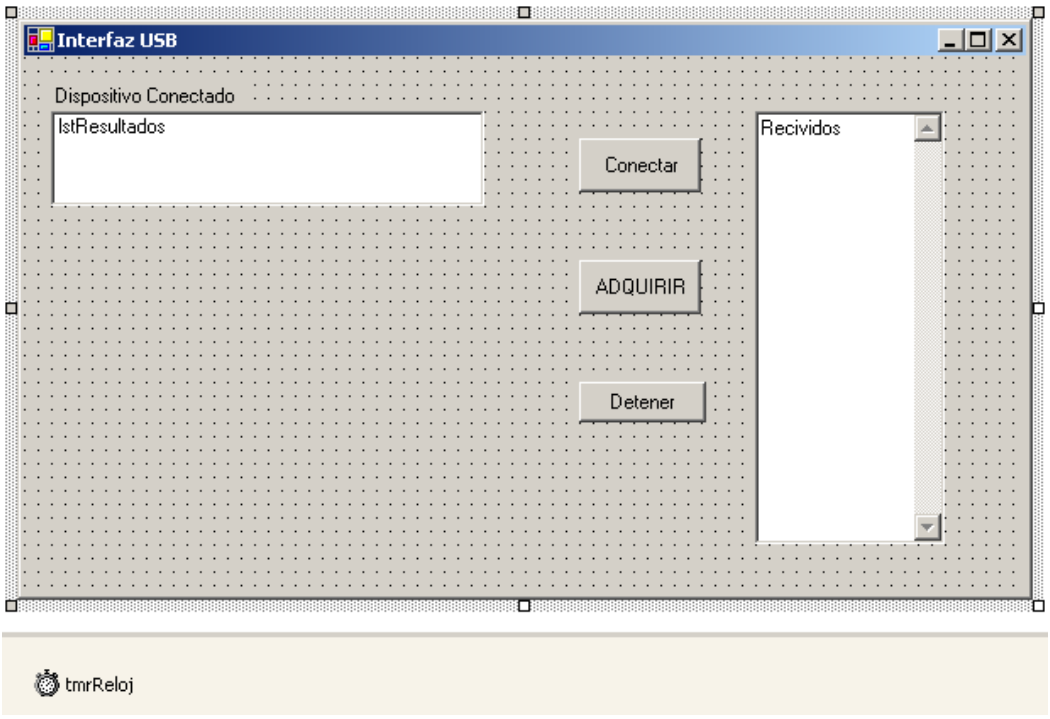


Figura 23. Diseño de la interfaz USBHIDIO.

Y a pesar de que esto no alcanza el máximo indicado en las especificaciones USB, es mucho mayor al obtenido con el MSCOMM. Para nuestro estudio de caso, la adquisición de temperatura, no es de mucha preocupación la velocidad de transmisión de datos, por lo que cualquiera de las dos soluciones son válidas.

En el análisis de la figura 24 se muestra un fragmento de la comunicación entre el USBHIDIO5 y el microcontrolador, en la cual se puede observar como se mantiene una velocidad constante de $20{,}000\frac{bytes}{s}$.

Tomando en cuenta que la velocidad en la comunicación de datos, a partir del software diseñado con el manejo de funciones API, es superior, se decidió terminar la interfaz del sistema de adquisición de temperatura, con base en las versiones USBHIDIO del software desarrollado.

Figura 24. Análisis de la comunicación entre el dispositivo de interfaz humana con el *firmware* HID9 y el programa interfaz USBHIDIO5.

3.4. Diseño final de la interfaz de usuario

Tomando como base USBHIDIO5, se desarrolló una interfaz de muestra para la aplicación de estudio. Para explicar dicho desarrollo, podemos empezar por mostrar la interfaz en el área de diseño, en la figura 25.

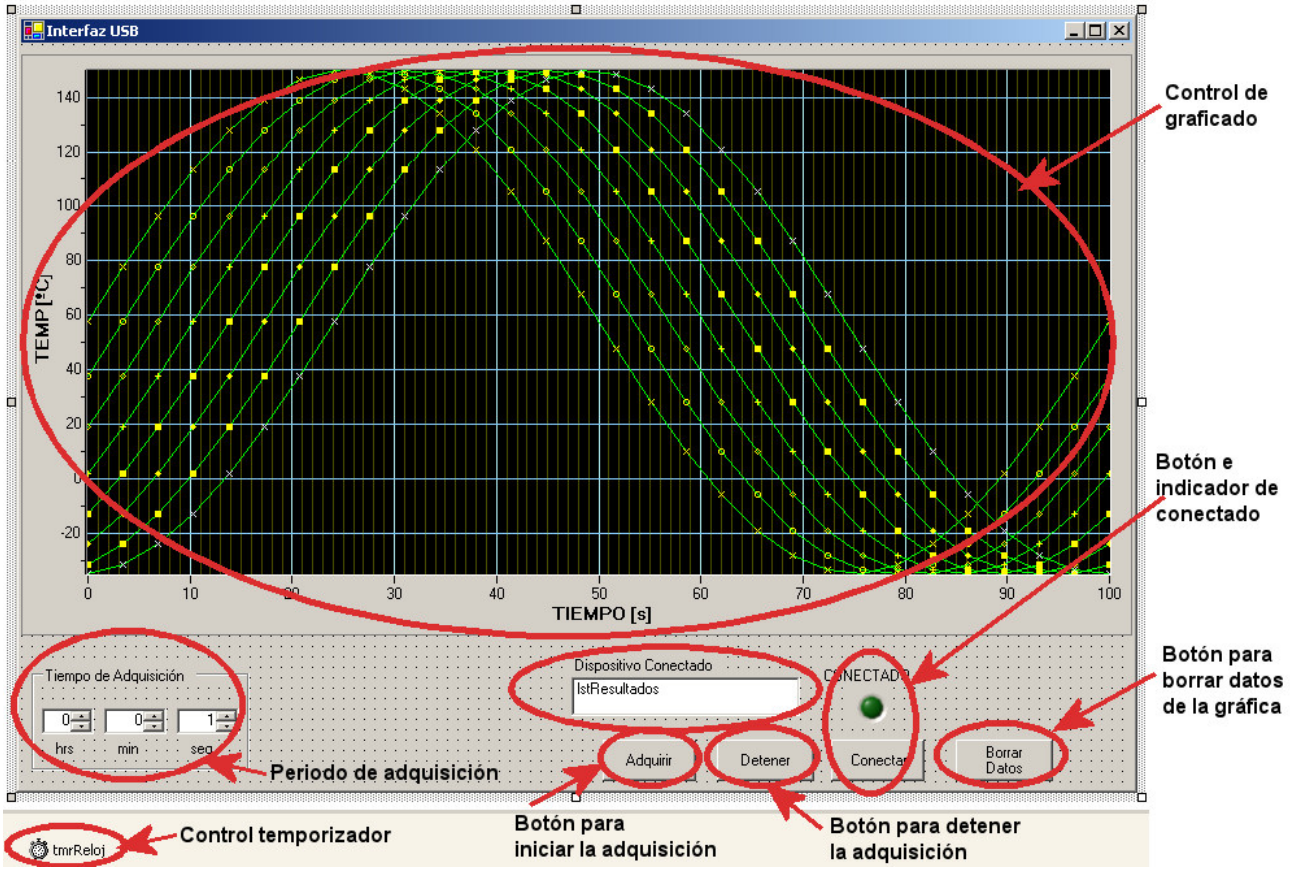


Figura 25. Interfaz final para el sistema de adquisición de temperatura.

Algo que se puede observar en el diseño, es el uso del control de tiempo de Windows, el cual en todas las versiones fue usado para buscar periódicamente datos en el HID. Esto fue por su facilidad de uso, sin embargo, presenta serias limitaciones al tratarse de señales menores a 10ms, ya que no garantiza poder ejecutar los comandos que se coloquen en su subrutina periódica. Sin embargo, para la aplicación que estamos desarrollando, en la cual la velocidad de

adquisición no es muy alta, a pesar de que se trata de múltiples canales, es adecuado.

Para el diseño de esta interfaz se crearon diagramas de flujo de cada función que se ejecuta al activar o modificar los controles en ella.

Los diagramas de la figura 26 muestran las funciones que se ejecutan al cambiar el usuario el valor del periodo de adquisición, se puede observar que los tres diagramas son iguales y corresponden a los tres campos de ajuste del tiempo de adquisición, que pueden observarse en la figura 25. Las funciones descritas se encargan de recalcular el periodo de adquisición en segundos, a partir de los valores en los tres campos numéricos marcados como “hrs”, “min” y “seg”.

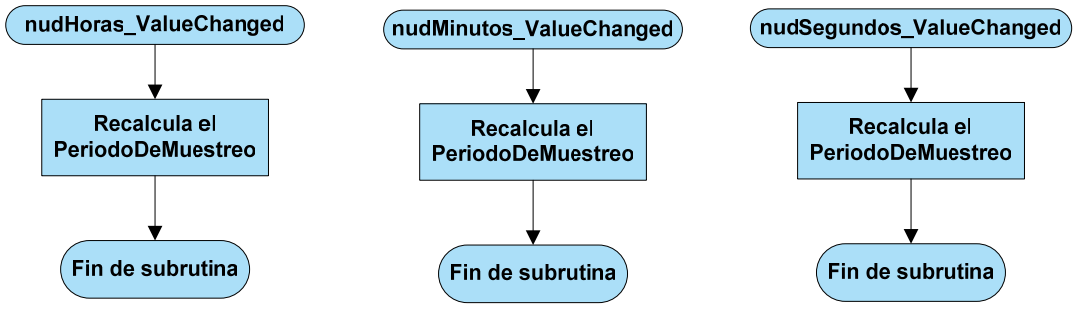


Figura 26. Funciones de los controles de segundos, minutos y horas del periodo de adquisición.

En la figura 27 se muestra el diagrama de la función que se ejecuta al oprimir el botón “Conectar”, en la interfaz de usuario. Lo primero que hace la función es buscar el HID a partir de sus números de identificación. En el caso de que se encuentre el dispositivo deseado, habilita los controles de la interfaz, excepto el botón “Detener”, enciende el led “Conectado” y deshabilita el botón “Conectar”. En el caso de que el HID no se encuentre, se despliega un mensaje que dice “Dispositivo no encontrado”.

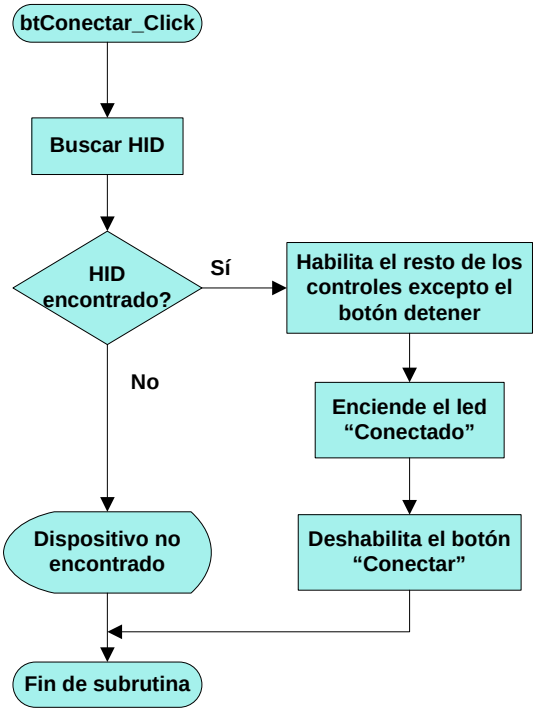


Figura 27. Función del botón "Conectar".

Después tenemos la función que se ejecuta al oprimir el botón "Adquirir", de la cual se puede observar su diagrama de flujo en la figura 28. Al comenzar a trabajar esta función, se deshabilitan los botones "Adquirir" y "Borrar Datos", así como los controles para fijar el periodo de adquisición, también se habilita el botón "Detener". En seguida, se fija el tiempo para el control del temporizador a 1 segundo y se activa el temporizador, para que el control ejecute una función periódicamente. A continuación se inicializa la variable ContadorDeTiempo, la cual servirá para contar el número de eventos del temporizador que se ejecuten (de un segundo cada uno). Por último, se envían dos *bytes*, como comando para que el microcontrolador comience la adquisición.

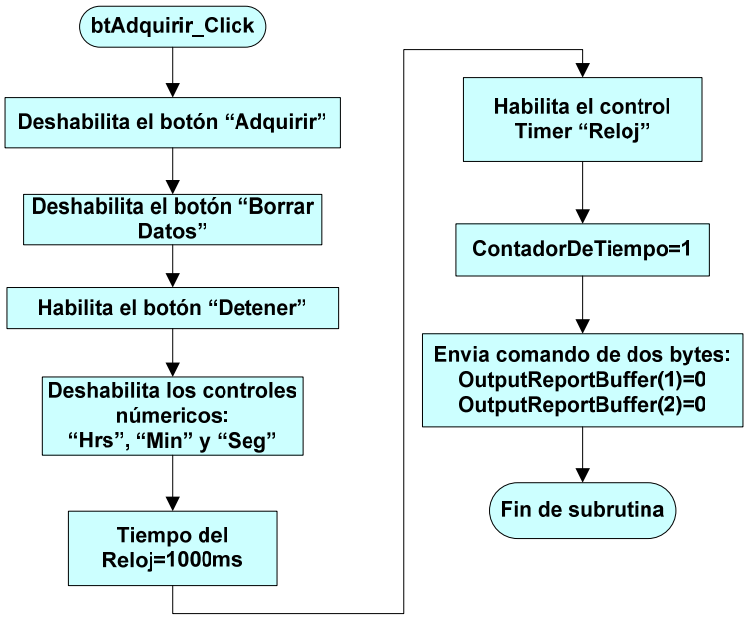


Figura 28. Función del botón "Adquirir".

En la figura 29 se muestra el diagrama de la función "tmrReloj_Tick", la cual se ejecuta periódicamente cada x segundos, pero como se vio en la figura 28, se programa a 1 segundo para nuestra aplicación. Comienza verificando si nuestra variable "ContadorDeTiempo", es igual al número de segundos del periodo de muestreo que introdujo el usuario. De ser así, se solicita al HID que envíe un grupo de datos y se reinicia el "ContadorDeTiempo". Después se incrementa la misma variable en uno y se sale de la subrutina [4], [7], [30].

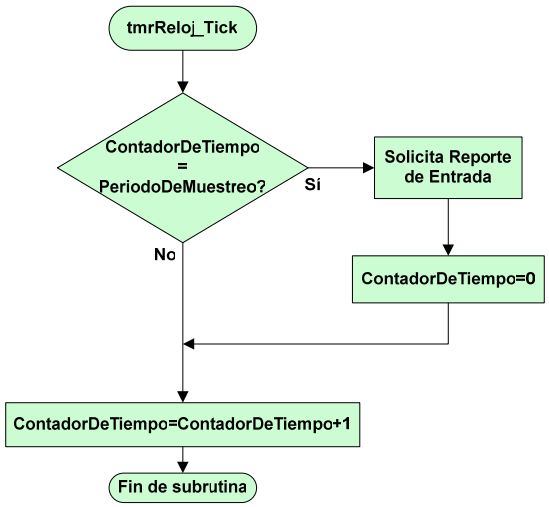


Figura 29. Función del control temporizador.

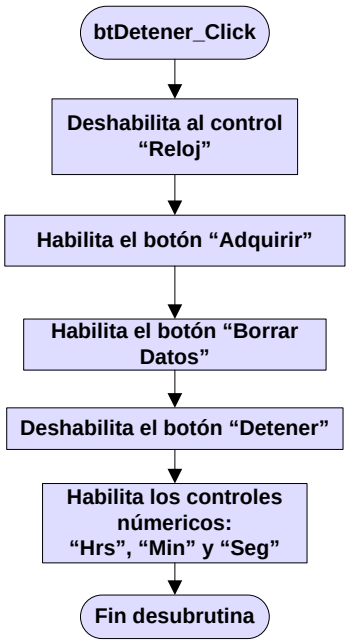


Figura 30. Función del botón "Detener".

Por último, tenemos la función que se ejecuta con el evento en que se presiona el botón "Detener", y en la figura 30 se muestra su diagrama de flujo. En esta función se habilitan los botones "Adquirir", "Borrar Datos" y los campos para cambiar el periodo de adquisición. También se deshabilitan el control temporizador y el botón "Detener".

En la figura 31 podemos observar al software mientras adquiere 6 canales y los grafica de manera simultánea.

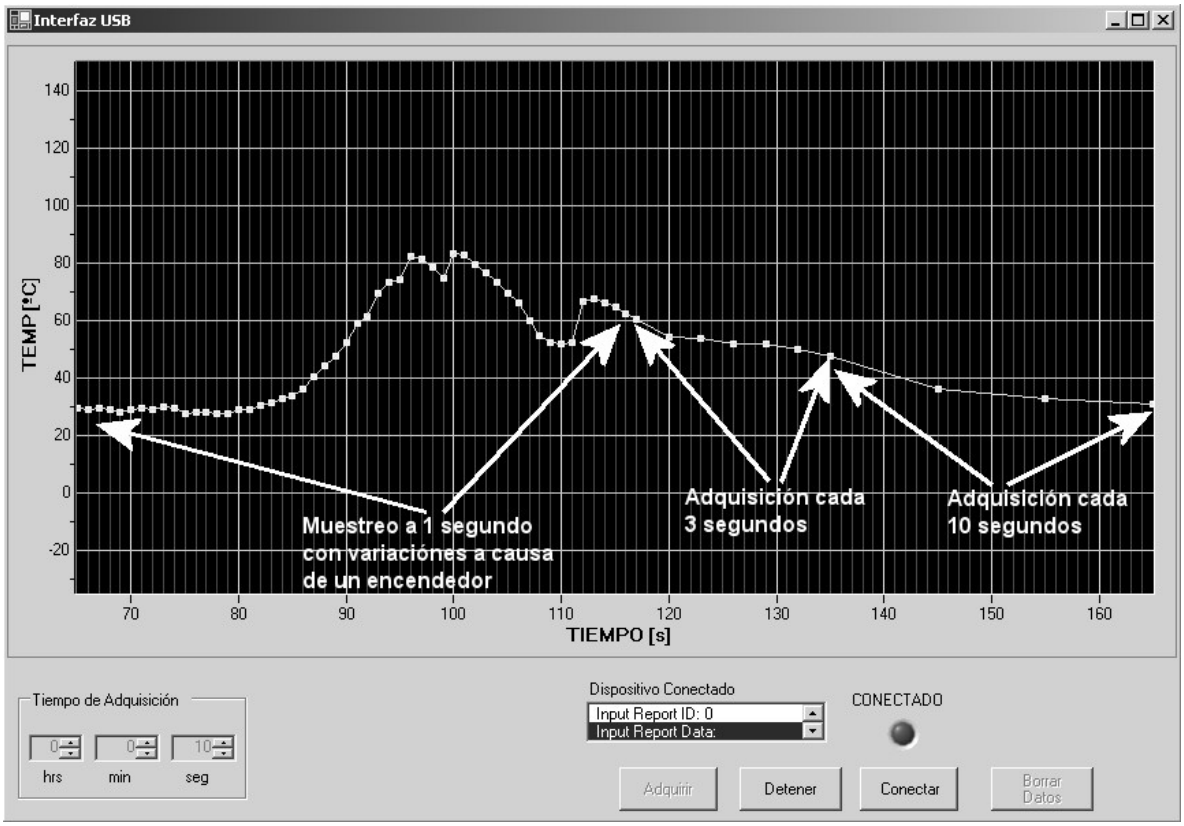


Figura 31. Corrida de la interfaz final, conectada al HID.

Es importante mencionar que el microcontrolador está funcionando con el *firmware* HID9, y también que para poder transmitir los 6 canales de forma simultánea se modificaron los descriptores para que cada reporte contuviese 16 bytes de datos, debido a que el convertidor A/D es de 10 bits y cada muestra debe enviarse al *host* en dos *bytes*.

En la figura 32 se muestra el análisis del USB, y se puede observar como el HID diseñado puede enviar 16 bytes por cada 10ms, es decir, $1600\frac{\text{bytes}}{\text{s}}$ en cada reporte de entrada. A pesar de que esta tasa de transferencia es menor a la que se alcanzó cuando sólo se transmitían dos *bytes* a la vez, el manejo de los hasta 8 canales de esta forma, permite mantener a la par la adquisición de todos las señales de entrada a nuestro HID.

Figura 32. Análisis del microcontrolador con el *firmware* HID9 y el *host* con el software Interfaz1.

En la figura 33 se muestra la tarjeta de evaluación de *Microchip*, “PICDEM FS USB”, en la cual se desarrollo este trabajo.

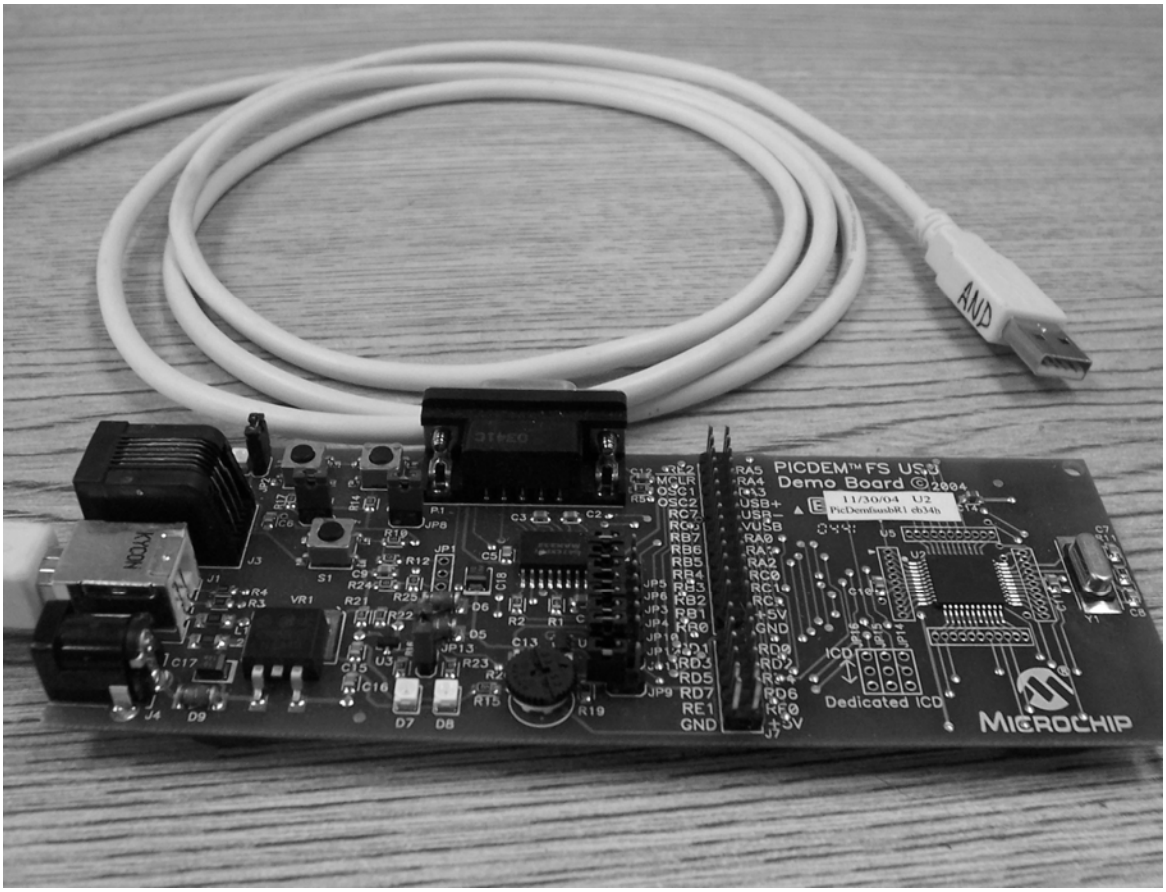


Figura 33. Tarjeta de evaluación de Microchip “PICDEM FS USB”, para el PIC184550.

RESULTADOS Y CONCLUSIONES

Al principio de este trabajo, se planteó la necesidad de poder sustituir el puerto de transmisión de datos, de los diseños del Laboratorio de Electrónica del CCADET, de RS-232 a USB.

En los objetivos se planteó que se diseñaría una interfaz USB que solamente trabajara a baja velocidad (*low-speed*), que fuese más fácil de usar que las interfaces usadas hasta el momento y que se adaptara a las necesidades de los equipos diseñados en el Laboratorio de Electrónica del CCADET.

Después de realizarle pruebas a nuestro diseño, con el ejemplo del sistema de adquisición de temperatura, se puede observar que la facilidad de uso, aunque podría ser comparable a la del puerto serie o paralelo en algunas aplicaciones, es mucho mayor en otras, como por ejemplo aquellas que requieran la conexión de múltiples equipos, ya que no requiere la instalación de software o hardware adicional.

Además, el protocolo de comunicación USB, es extremadamente flexible y permite que se pueda transmitir todo tipo de información a través del puerto. A veces sólo es necesario hacer pequeñas modificaciones para ajustar la interfaz a las especificaciones del diseño; sin embargo, de no ser suficiente, el protocolo para el puerto USB nos ofrece un gran número de clases de dispositivos, que pueden ser implementadas con un poco más de esfuerzo, pero usando como base la interfaz diseñada en este trabajo.

En los objetivos se planteó un límite de velocidad (*low-speed*) para la interfaz a desarrollar, con la idea de que más adelante, con base en este diseño, fuera posible rebasar por mucho las velocidades del puerto serie. Sin embargo, los resultados obtenidos son superiores a los esperados (*low-speed*) ya que en vez de

alcanzar los 800 $\frac{bytes}{s}$ máximos de la transmisión en *low-speed*, se logró transmitir a full-speed con una taza constante de 16 bytes por cada 10ms, es decir 1,600 $\frac{bytes}{s}$. Esto esta lejos del máximo teórico para esta velocidad que es 1.2 $\frac{MB}{s}$, sin embargo, el diseño puede ser mejorado.

También, cabe mencionar que el USB posee otra ventaja sobre el puerto serie, esto es que los dispositivos USB tienen la opción de funcionar con energía propia del puerto o con una fuente propia. De querer funcionar con energía del puerto, como el ejemplo del sistema de adquisición de temperatura, el bus es capaz de proveer hasta 500 mA, siempre y cuando el dispositivo los solicite durante la enumeración.

Es importante resaltar el hecho de que este trabajo fue hecho con la idea de explorar el mundo del diseño de dispositivos USB, con el objetivo de adquirir el conocimiento necesario para su implementación y desarrollo. Además, de esta forma, dejar de depender por completo de los fabricantes de soluciones prefabricadas, para poder incursionar nosotros mismos, en un campo que no ha sido explotado.

En cuanto a la implementación física de la interfaz, el primer prototipo se puede observar junto con su circuito impreso y su diagrama esquemático, en el apéndice A. En estos diagramas, se puede apreciar que el prototipo es aparentemente simple, sin embargo la parte más compleja se encuentra dentro del microcontrolador, es decir, el *firmware*. A partir del prototipo presentado, los diseñadores pueden hacer uso de los puertos del microcontrolador para realizar diversas tareas.

Debido a lo mencionado anteriormente, podemos concluir que se cumplieron los objetivos planteados al inicio de este trabajo de forma satisfactoria, e incluso en algunos puntos, se excedieron las especificaciones esperadas.

APÉNDICE A: DIAGRAMAS DEL PROTOTIPO

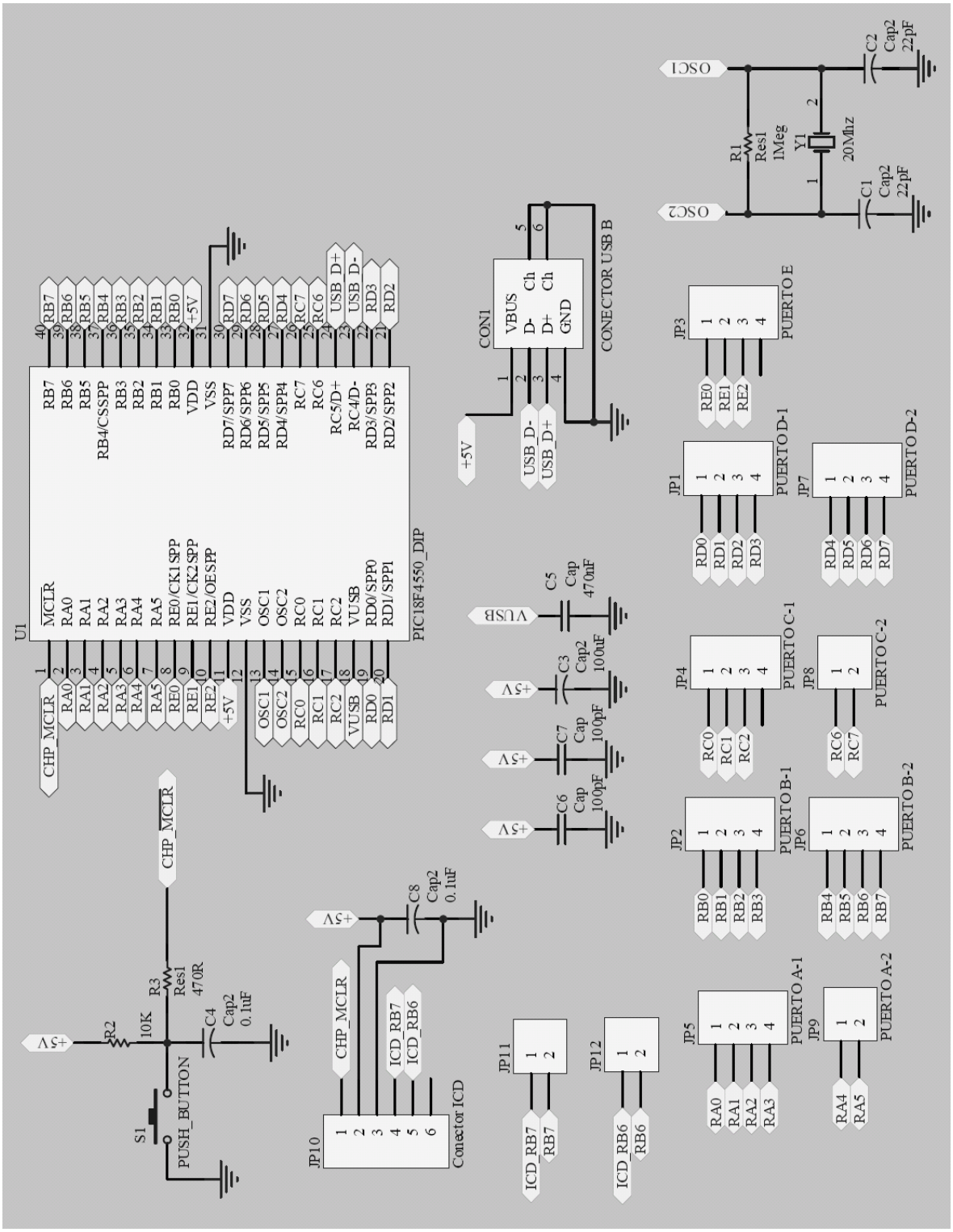


Figura 34. Diagrama de la interfaz USB.

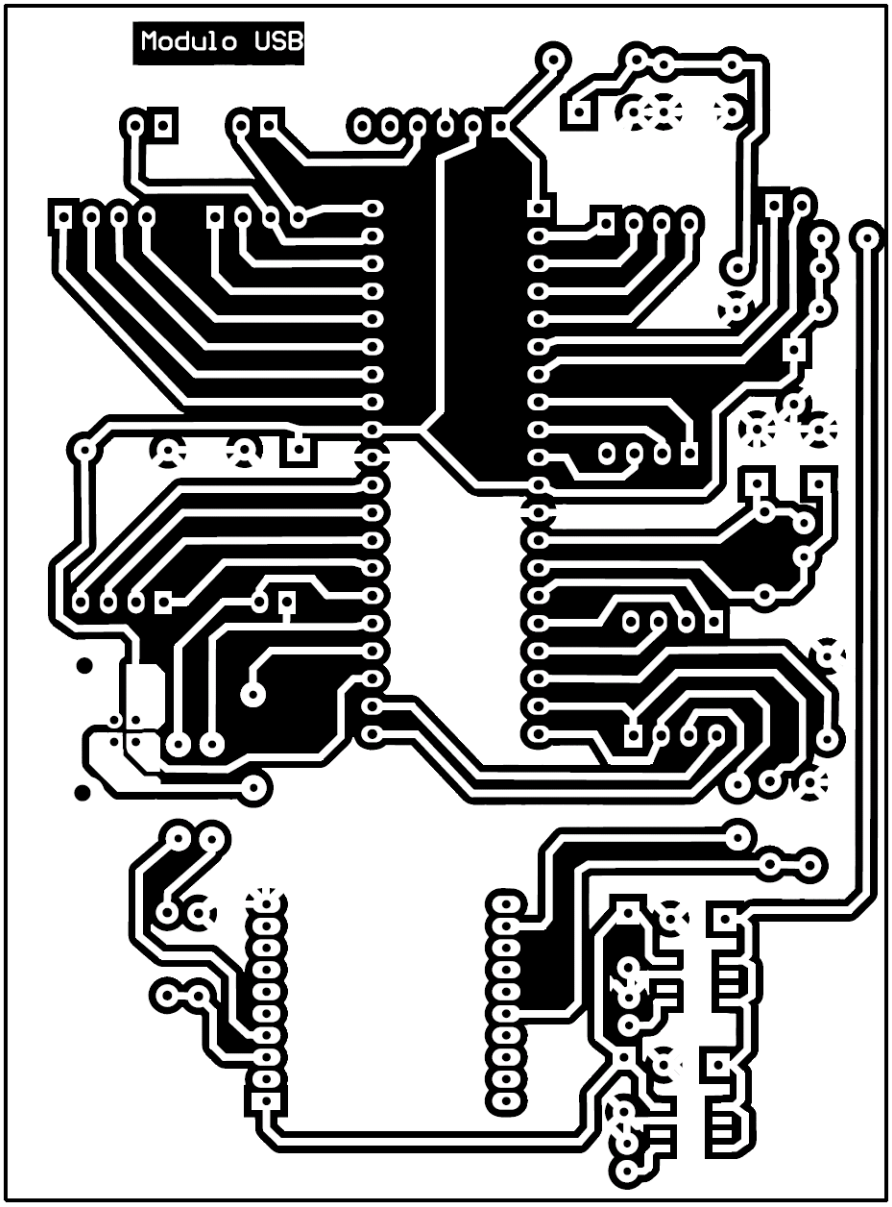


Figura 35. Diagrama del circuito impreso de la interfaz USB.

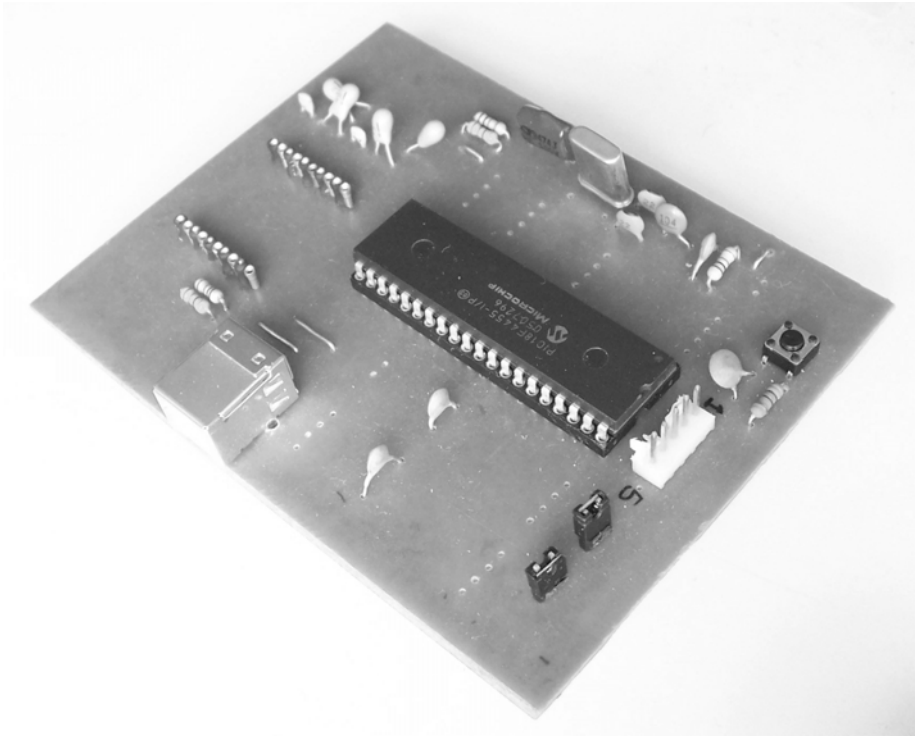


Figura 36. Primer prototipo de la interfaz USB (vista superior).

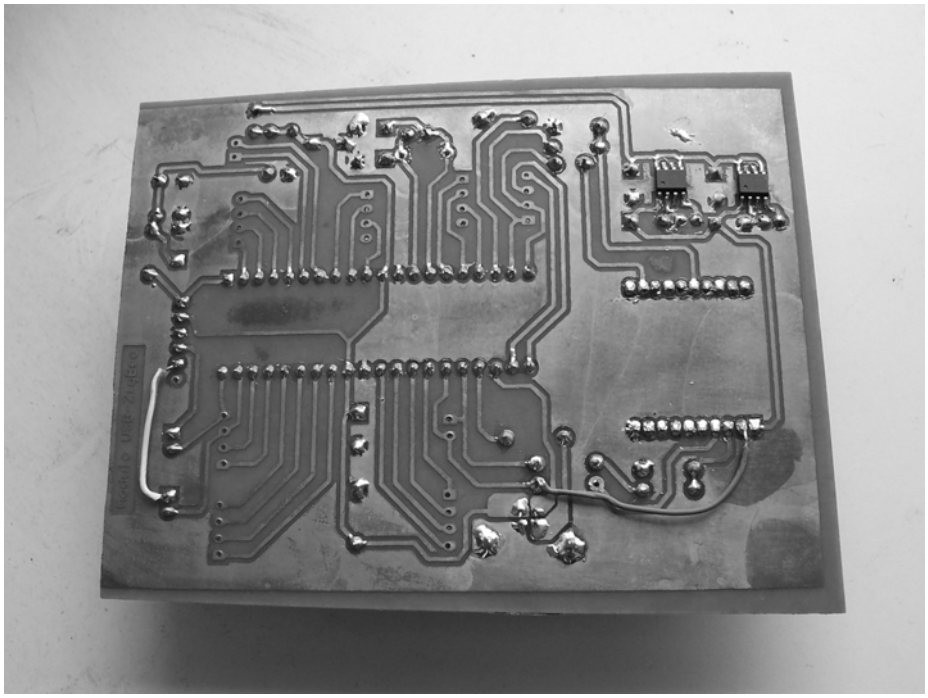


Figura 37. Primer prototipo de la interfaz USB (vista inferior).

APÉNDICE B: CÓDIGO DEL *FIRMWARE*

En este apéndice se incluye el código en C, de las principales funciones de la versión HID9, para el compilador de Microchip, C18. Para mayor referencia consúltese las referencias [22][25].

```
void timer_isr (void)                    //Rutina de servicio a la interrupción del timer
        {
        basesT++;
        nuevo_dato=0;
        if(basesT == 183)          // 183 desbordamientos de un contador de 16 bits a
                                   //una frec de 48MHz/4, es 1 segundo
         {
                basesT =0;
                i=0;
                j=0;
                for (i=0, j=0; j<=15; i++, j=j+2)
                {
                ASCON0bits= (i<<2)|( 1);          //Multiplexa entre 8 canales
                Delay100TCYx(2);                  //Un retardo para que el convertidor esté listo
                ConvertADC();
                while(BusyADC());
                buff_adq=ReadADC();
                transmit_buffer[j] = buff_adq;
                transmit_buffer[j+1] = buff_adq>>8;
                }

                HIDTxReport(transmit_buffer, HID_INPUT_REPORT_BYTES);
                 // The report will be sent in the next interrupt IN transfer.

                while(mHIDTxIsBusy())  // Espera hasta que deja de transmitir
                {
                USBDriverService();       // Da servicio a las interrupciones del puerto USB.
                }
        }

        INTCONbits.TMR0IF=0;
        INTCONbits.TMR0IE=1;

        }
```

Figura 38. Código de la función de atención a interrupciones del Timer0, del *firmware* HID9.

```
void ReportLoopback(void)
{
	byte count;
	int i;		// Find out if an Output report has been received from the host.
	number_of_bytes_read = HIDRxReport(receive_buffer, HID_OUTPUT_REPORT_BYTES);

	if (number_of_bytes_read > 0)
	{		// Se recibió un paquete de salida

		if(receive_buffer[0]==0 & receive_buffer[1]==0)	// Verifica si se recivieron dos ceros.
		{
			INTCONbits.GIE=1;		//Activa las Interrupciones globales
			INTCONbits.TMR0IE=1;
		}
	}


}//Fin de ReportLoopback
```

Figura 39. Código de la función *ReportLoopback*, del *firmware* HID9.

REFERENCIAS

[1]. Hyde, John. (2001). USB Design by Example. A Practical Guide to Building I/O Devices. Second Edition. Edit. Intel Press: USA

[2]. Axelson, Jan. USB Complete: Everything You Need to Develop Custom USB Peripherals. Segunda Edición. Edit. Lakeview Research: USA

[3]. Schildt, Herbert. (2000). Manual de referencia C. Cuarta Edición. Osborne McGraw-Hill: México.

[4]. Petroutsos, Evangelos. (2002). Mastering Visual Basic .NET. Edit. Sybex: USA.

[5]. W. Kernighan, Brian, M. Ritchie, Dennis. (1988). THE C PROGRAMMING LANGUAGE. Second Edition. Edit. Prentice Hall: USA.

[6]. T. Roff, Jason. (2001). ADO: ActiveX Data Objects. First Edition. Edit. O'Reilly: USA.

[7]. Grundgeiger, Dave. (2002). Programming Visual Basic .NET. First Edition. Edit. O'Reilly: USA

Material electrónico

[8]. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft: NEC, Phillips. (2000) "Universal Serial Bus Specification". [electrónico]. Universal Serial Bus. http://www.usb.org [Recuperado en Febrero 2005].

[9]. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Phillips. (2000). Alpis, Cybernet, DEC, Intel, Key Tronic Corporation, LCS/Telegrphics, Logitech, Microsoft Corporation, NCR, Sun Microsystems, ThrustMaster. (2001). "Universal Serial Bus (USB). Device Class Definition for Human interface Devices (HID)". [electrónico]. Universal Serial Bus. http://www.usb.org [Recuperado en Febrero 2005

[10]. Peacock, Craig. (2002). "USB in a Nutshell. Making Sense of the USB Satandard". [electrónico]. Beyond Logic. http://www.beyondlogic.org/usbnutshell [Recuperado en Agosto 2004].

[11]. Texas Instruments. "TUSB3210 Universal Serial Bus General-Purpose Device Controller. Data Manual". [electrónico]. Texas Instruments:

Technology for Innovators. http://focus.ti.com/docs/prod/folders/print/tusb3210.html [Recuperado Septiembre 2004].

[12]. National Semiconductors. “USBN9603/USBN9604 Universal Serial Bus Full Speed Node Controller with Enhanced DMA Support”. [electrónico]. National Semiconductors: The Sight & Sound of information. http://www.national.com/pf/US/USBN9603.html [Recuperado Agosto 2004].

[13]. National Semiconductors. “LM134/LM234/LM334 3-Terminal Adjustable Current Sources”. [electrónico]. National Semiconductors: The Sight & Sound of information. http://www.national.com/ [Recuperado Marzo 2006].

[14]. Tetradyne Software, INC. “DriverX: Hardware Control for 32-bit Windows Applications”. [electrónico]. http://www.tetradyne.com/driverx.htm [Recuperado Septiembre 2005]

[15]. Custom Computer Services Incorporated. (2005). “C Compiler Reference Manual November 2005”. [electrónico]. C Compiler for Microchip picmicro MCUs. http://www.ccsinfo.com/picc.shtml [Recuperado Septiembre 2005]

[16]. Custom Computer Services Incorporated. (2005). “PICmicro MCU C: An Introduction to Programming The Microchip PIC in CCS C”. [electrónico]. C Compiler for Microchip picmicro MCUs. http://www.ccsinfo.com/picc.shtml [Recuperado Septiembre 2005].

[17]. Microchip. “MPLAB C18 C COMPILER GETTING STARTED”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Abril 2005].

[18]. Microchip. “MPLAB C18 C COMPILER USER’S GUIDE”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Abril 2005].

[19]. Microchip. “MPLAB C18 C COMPILER LIBRARIES”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Abril 2005].

[20]. Microchip. “MPLAB C18 C COMPILER ADDENDUM”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Abril 2005].

[21]. Future Technology Devices International Ltd. “FTDI Chip: FT2232C Dual USB UART / FIFO I.C”. [electrónico]. FTDI Chip Future Technology

Devices International Ltd. http://www.ftdichip.com/Products/FT2232C.htm [Recuperado Agosto 2004].

[22]. Lakeview Research home. “Jan Axelson´s Lakeview Research”. [electrónico]. Lakeview Research home. http://www.lvr.com/ [Recuperado Febrero 2005].

[23]. ELO Touchsystems, LCS/Telegraphics, Microsoft Corporation, Symbol Technologies, Inc., Intel, TV Interactive, Logitech International. “Universal Serial Bus HID Usage Tables”. [electrónico]. Universal Serial Bus. http://www.usb.org/developers [Recuperado Agosto 2004]

[24]. Microchip. “Microchip PIC18F2455/2550/4455/4550 Data Sheet. 28/48/44-Pin High-Performance, Enhanced Flash USB Microcontrollers with nano Watt Technology”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Noviembre 2004]

[25]. Microchip. “Microchip PICDEM FS USB DEMOSTRATION BOARD USER’S GUIDE”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Noviembre 2004]

[26]. Microchip. “Microchip PIC16C745/765 8-bit CMOS Microcontrollers with USB”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Agosto 2000]

[27]. Microchip. “Microchip PICDEM USB User’s Guide”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Agosto 2000].

[28]. Analog Devices. “Low Voltage Temperature Sensors TMP35/TMP36/TMP37”. [electrónico]. Analog Devices. http://www.analog.com/en/prod/0,,764_811_TMP35,00.html [Recuperado Marzo 2006].

[29]. Microchip. “AN730: CRC Genarating and Checking”. [electrónico]. Products.Microchip. http://www.microchip.com [Recuperado Noviembre 2004]

[30]. Microsoft. “MSDN”. [electrónico]. Microsoft Designers Network. http://www.msdn.com