UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UNA TARJETA DE ADQUISICIÓN DE DATOS USB PARA FINES DIDÁCTICOS

TESIS

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero Mecatrónico

PRESENTA

ALEJANDRO ELÍAS RODRÍGUEZ

DIRECTOR DE TESIS: M. I. BILLY ARTURO FLORES MEDERO NAVARRO

MÉXICO, D. F. 2009

1 Introducción

En la presente tesis se trata el desarrollo de una tarjeta de adquisición de datos por puerto "*USB*". Antes que nada se describe el por qué de esta tesis. Posteriormente se toman algunos de los antecedentes necesarios para poder desarrollar esta tarjeta. Finalmente, se analiza el desarrollo y la evolución de la tarjeta.

Posteriormente, al hablar del por qué de esta tesis, nos remontamos a los temas más elementales de la propuesta, ¿del por qué de un puerto *USB*?, ¿qué es una tarjeta de adquisición de datos? y adicionalmente se tratan los alcances y objetivos de esta tesis.

1.1 ¿Por qué un puerto USB?

La evolución natural de las computadoras nos ha llevado a generar interfaces de comunicación rápidos, seguros y eficientes. Esta misma evolución es la que nos ha llevado a la creación del *USB* (*Universal Serial Bus*).

Para poder hablar sobre un por qué del *USB* necesitamos saber las ventajas y desventajas que este presenta ante otros puertos, lo cual concierne al siguiente capítulo. En este punto simplemente se mencionan algunas de las características más importantes de este. Como el nombre lo indica, el *USB* tiende a ser universal, es rápido, tiene bajo consumo de energía y, debido a su protocolo de comunicación y a su estructura física, es *Plug-n-Play*, que puede traducirse como conecta y funciona, esto permite conexiones y desconexiones dinámicas y la detección de dispositivos conectados.

Como el puerto busca ser universal, es natural que todas las computadoras modernas cuenten con al menos uno de estos; de hecho, en algunas computadoras es el único puerto de comunicación con que cuentan. Ésta es una de las principales ventajas que nos ofrece el puerto *USB*. Al existir un *USB* en cualquier computadora, lo más lógico es pensar en generar la mayor cantidad de periféricos comunicados a través de él.

Adicionalmente, al ser más rápido que los puertos más utilizados, es una buena opción para tareas que implican una gran transferencia de datos en un periodo de tiempo relativamente corto. Tareas como la transferencia de archivos, captura de video, audio o

cualquier tipo de datos en tiempo real, entre otras, son aplicaciones típicas para el *USB*, en las que además resulta óptima la selección de este puerto.

Por si esto no bastara, el *USB* es capaz de suministrar hasta 500 miliamperes de corriente, lo que nos permite desarrollar aplicaciones sin fuente adicional de energía. No debe de perderse de vista el hecho de que si nuestra aplicación requiere de más de 500 miliamperes, será necesario agregar una fuente externa de alimentación, por lo que el puerto consumirá únicamente la energía para transmisión de datos.

Por otra parte, el término *Plug-n-Play* se refiere a la capacidad de un periférico de ser configurado automáticamente por la computadora al ser conectado a la misma; si conectamos algún dispositivo en los puertos paralelo o serie usualmente es necesario reiniciar la computadora para que la configuración tenga efecto. En cambio, un *USB* permite utilizar el elemento inmediatamente después de haberlo conectado. Cabe aclarar que en algunas ocasiones es necesario proporcionar el controlador (programa que instruye a la computadora sobre cómo configurar el periférico en cuestión), sin embargo, después de haberlo proporcionado la primera vez, el periférico se puede utilizar al momento de conectarse nuevamente.

Con base en lo anterior, el *USB* ofrece varias facilidades a nivel usuario, lo que ha propiciado que se demanden más computadoras con este puerto. Adicionalmente, existe una creciente demanda de computadoras portátiles que en la actualidad carecen de los puertos paralelo y serie, que serían la alternativa más viable de expansión de la computadora después del *USB.*

El desarrollo de la tarjeta propuesta busca ayudar tanto a alumnos como a profesores en la comunicación con la computadora en los diversos proyectos escolares. Según lo mencionado anteriormente, se observa una creciente demanda de computadoras portátiles únicamente con *USB* y debido al costo de una tarjeta comercial (superiores a los 1,500 pesos), es conveniente realizar un desarrollo propio para los alumnos de la Facultad de Ingeniería.

1.2 ¿Qué es adquisición de datos?

Típicamente se entiende por adquisición de datos al proceso por el cual se toman muestras de fenómenos físicos cuantificables y se acondicionan para poder procesarlos por una computadora [9].

A manera de ejemplificar lo anterior, supóngase que se tiene un sismógrafo. Al momento en que se presenta un temblor, la aguja se mueve, manifestándose como picos en un rollo de papel. Ahora bien, si lo que queremos es estar registrando estos valores en una computadora, en lugar de tener la información en un rollo de papel, lo que necesitamos es transformar ese movimiento en una señal eléctrica cuantificable, esta señal después se codifica en un sistema binario y luego se envía a la computadora mediante un puerto. Al proceso de acondicionar la señal y codificarla de manera que una computadora la entienda es a lo que se denomina como adquisición de datos y puede ser de señales analógicas, como el valor del temblor, o digitales, como la activación de una alarma.

La adquisición de datos (figura 1.1) no es un proceso que sólo se efectúe del mundo real a la computadora. Este proceso también puede darse desde la computadora al mundo real, en cuyo caso, la señal binaria se procesa y se manifiesta tanto como una salida digital o, con un poco más de acondicionamiento, como una señal analógica.

Figura 1.1. Esquema que ejemplifica la adquisición de datos

Como se puede ver en la figura anterior, la adquisición de datos no es un proceso de una sola etapa, sino que debe de cumplir con ciertos subprocesos que acondicionen las señales a un lenguaje con el que la máquina pueda trabajar.

Para analizar paso a paso el proceso de adquisición de datos, se usa como convención que el proceso se hace del mundo real hacia la computadora, siendo el proceso inverso directamente equiparable al proceso de la computadora hacia el mundo real.

Este proceso comienza con el fenómeno objeto de investigación. Usualmente la señal a adquirir del fenómeno no será del tipo eléctrico. Por esto, es necesario utilizar un transductor (elemento capaz de transformar entre tipos de energía), que cambie la magnitud del fenómeno, ya sea fuerza, presión, temperatura, etc., en una señal eléctrica.

En ese instante se tiene una señal eléctrica que, en el mejor de los casos presentará valores digitales, es decir, un “1” o un “0” lógico. Si éste fuera el caso, la señal se puede enviar a la computadora con sólo acondicionar los niveles de voltaje. Sin embargo, muchas veces lo que se tiene es una señal analógica (que tiene valores continuos) la cual hay que discretizar y codificar para lograr que la computadora entienda dicho valor. Es aquí donde entran los convertidores analógico-digital, o digital-analógico en caso de que sea de la computadora al mundo real, por ejemplo.

Un convertidor analógico-digital, figura 1.2, o *ADC* (por su nombre en inglés *Analog to Digital Converter*) es un dispositivo que compara el voltaje de entrada con un valor de referencia y da como salida un código digital equivalente. Los procesos por los que pasa este convertidor se muestran en la misma figura. De manera análoga, un convertidor digital-analógico o *DAC* (del inglés *Digital to Analog Converter*), recibe un código digital y nos da un voltaje de salida equivalente.

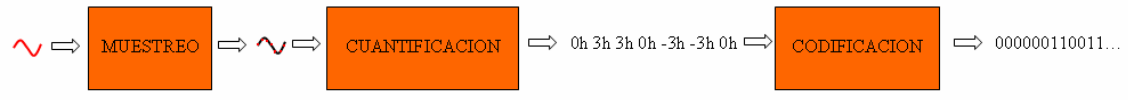


Figura 1.2. Proceso de conversión analógico-digital

Como los valores arrojados por el *ADC* son paquetes de unos o ceros lógicos se necesita un elemento que se encargue de la transmisión de los datos para que la computadora los pueda recibir de una forma que ésta entienda. Este elemento debe de ser capaz de transmitir, por medio del puerto al que se conecte, los datos en el momento en el que sean requeridos. Para esto, se requiere de un hardware de transferencia y un protocolo de transferencia, y finalmente, se necesita un software con el cual operar los datos recibidos.

Existen otras etapas de acondicionamiento, como: filtrado, amplificación, atenuación, aislamiento, etc. Sin embargo, no tiene caso profundizar en ellas al no ser siempre necesarias y al estar fuera de los alcances del presente trabajo.

1.3 Alcances y objetivos

Esta tesis, como el nombre lo indica, pretende abordar el desarrollo de una tarjeta de adquisición de datos para fines didácticos. De esta manera, la tarjeta puede servir como una herramienta en la enseñanza de materias que así lo requieran. Adicionalmente, la tarjeta puede ser utilizada por estudiantes para el desarrollo de proyectos que requieran ser controlados por computadora, o que simplemente esta interfaz facilite y mejore el desempeño y desarrollo del proyecto. Se busca que el puerto de comunicación de la tarjeta sea el puerto *USB* para lograr conectar la computadora portátil de los alumnos al proyecto y así lograr una cierta autonomía al no depender de una computadora de escritorio que cuente con los puertos serie o paralelo.

En la actualidad, existen en el mercado tarjetas de adquisición de datos conectadas mediante el puerto *USB*, sin embargo, tienen la desventaja de que presentan un costo muy elevado. Tomando como referencia la tarjeta "NI USB-6212" de *National Instruments*, cuyas principales características se citan en la tabla 1.1, que presenta un costo de $1,100.00 USD [9], muestra un precio inasequible para la mayoría de los alumnos.

General	
Conexión	USB
Entradas Analógicas	
Cantidad	16 (Multiplexadas)
Resolución	16 bits
Rango máximo de voltaje	-10 .. 10 V
Salidas Analógicas	
Cantidad	2
Resolución	16 bits
Rango máximo de voltaje	-10 .. 10 V
Entradas y Salidas Digitales	
Cantidad	32
Niveles Lógicos	TTL

Tabla 1.1. Características importantes de la tarjeta de adquisición de datos NI USB-6212 de *National Instruments*

La tarjeta a desarrollar no planea ser una innovación tecnológica, más bien planea ser una solución costeable que pese a que no iguale las características de tarjetas de adquisición de datos del mercado, sea plenamente funcional para proyectos a nivel educativo.

La solución pensada incluye entradas y salidas analógicas y digitales, 8 canales digitales y 8 analógicos, cuyo voltaje de operación se limita a un rango de 0 a 5 volts.

Debido a su facilidad de obtención, y a la familiaridad que tienen los microcontroladores PIC, el microcontrolador propuesto es un PIC18F4550, el cual es capaz de soportar comunicación *USB* y cuenta con canales multiplexados conectados a un *ADC* con una resolución de 10 bits. Existen otros microcontroladores con capacidades similares, sin embargo conseguirlos es complicado, o bien sus herramientas de desarrollo son costosas.

2 Antecedentes.

En este capítulo se trata la comparación entre puertos, analizando sus ventajas y desventajas. Adicionalmente se profundiza en el análisis de las características del puerto *USB*, pasando por el protocolo de comunicación y sus diferentes modos de operación.

Al tratarse la tesis de un desarrollo integral se debe retomar un poco sobre microcontroladores, sus especificaciones y su programación, así como la creación de circuitos impresos para dar presentación. Además, con fines de acoplamiento de circuitos, se necesita un poco de información sobre la comunicación I^2C.

2.1 Introducción a los puertos convencionales.

Debido a su popularidad, sólo se destacan los puertos *USB*, paralelo de impresión y RS-232, entendiendo que estos no son los únicos puertos existentes sino los de interés.

2.1.1 El puerto serie RS-232.

El puerto RS-232 es una interfaz serial muy popular en el uso de computadoras en la actualidad. A pesar de existir interfaces más rápidas y sofisticadas, la popularidad de la interfaz RS-232 continúa debido a su facilidad de implementación en hardware y a su bajo costo.

Figura 2.1. Conector DB 9, típicamente usado para la interfaz RS-232.

Típicamente este puerto se puede encontrar en una conexión del tipo DB de 9 (figura 2.1) o de 25 pines, siendo la primera la más común hoy en día. Ambos tipos de

conexión manejan 9 señales, que se muestran a continuación en la tabla 2.1. De las 9 señales, 6 son de control, y se puede establecer una comunicación con las otras tres señales (*RD*, *TD* y *GND*).

Señal		Pin en DB-9	Pin en DB-25
CD	*Carrier Detect*	1	8
RD	*Received Data*	2	3
TD	*Transmited Data*	3	2
DTR	*Data Terminal Ready*	4	20
GND	*Ground*	5	7
DSR	*Data Set Ready*	6	6
RTS	*Request To Send*	7	4
CTS	*Clear To Send*	8	5
RI	*Ring Indicator*	9	22

Tabla 2.1. Numeración de pines en los distintos conectores. Los pines no mencionados no se utilizan.

El cable de conexión puede tener un largo de hasta 30 metros y el número de hilos del mismo depende del conector y de la conexión que se desee realizar (con o sin señales de control).

Esta interfaz nos permite la comunicación sólo entre dos elementos con una tasa de transferencia máxima de 20 kb por segundo (115 kb agregando un poco de hardware). Los voltajes que maneja para los niveles lógicos son positivos y negativos en lugar de la lógica meramente positiva de los circuitos tipo TTL o CMOS. Para las señales de datos, en corriente directa, un voltaje positivo superior a +5 V corresponde al nivel lógico “0” mientras que un voltaje inferior a -5 V corresponde al nivel lógico “1”, es decir, lógica negativa; para las señales de control se manejan los mismos rangos pero con lógica positiva. Los rangos de voltaje utilizados son ±12 Vdc, sin embargo, los elementos soportan voltajes de hasta ±15 Vdc. Afortunadamente, existen circuitos integrados, como el MAX232, para convertir los niveles lógicos a TTL o CMOS.

2.1.2 El puerto paralelo de impresión.

Inicialmente fue pensado como un puerto exclusivo para la comunicación con una impresora, sin embargo, en la actualidad existen gran variedad de periféricos que se pueden conectar a la computadora a través de este puerto.

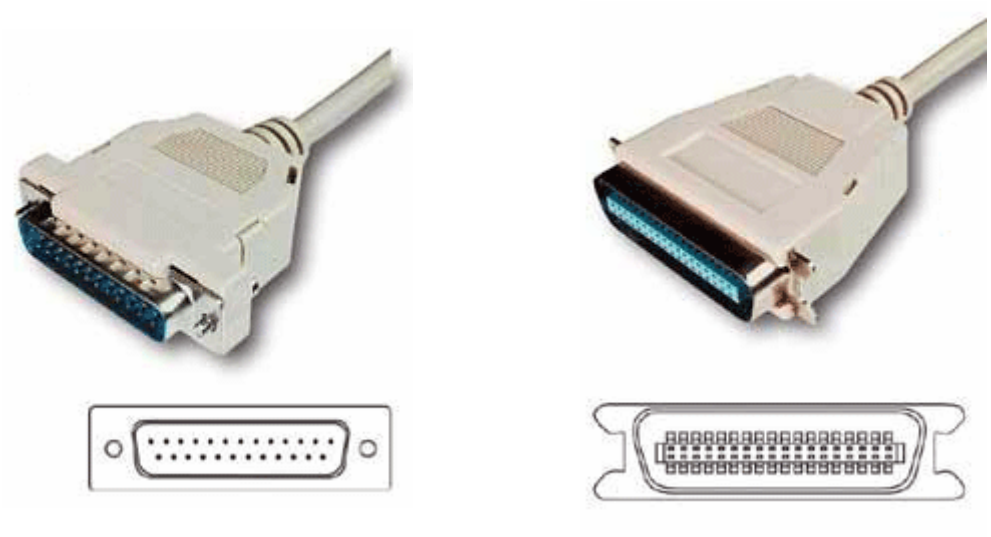


Figura 2.2. a) Conector DB 25. b) Conector Centronics.

Esta interfaz se puede presentar con conectores de 25 pines (figura 2.2.a) o de 36 pines (figura 2.2.b), de los cuales se tienen 8 pines de salida, 5 de entrada y 4 bidireccionales (lo anterior desde el punto de vista de la computadora). El cable conector que presenta la computadora es el DB-25 y los periféricos pueden presentar el DB-25 o el Centronics. Se muestra en la tabla 2.2 el listado de los pines.

Pin	Nombre	Tipo
1	*nStrobe*	Bidireccional
2	D0	Salida
3	D1	Salida
4	D2	Salida
5	D3	Salida
6	D4	Salida
7	D5	Salida
8	D6	Salida
9	D7	Salida
10	*nAck*	Entrada
11	*Busy*	Entrada
12	*PaperEnd*	Entrada
13	*Select*	Entrada

Pin	Nombre	Tipo
14	*nAutoLF*	Bidireccional
15	*nError*	Entrada
16	*nInit*	Bidireccional
17	*nSelectIn*	Bidireccional
18	GND	-
19	GND	-
20	GND	-
21	GND	-
22	GND	-
23	GND	-
24	GND	-
25	GND	-

Tabla 2.2. Listado de pines para un conector DB-25 de puerto paralelo de impresora.

A diferencia del puerto RS-232, el puerto paralelo de impresora maneja niveles lógicos TTL, lo que nos permite una fácil conexión con circuitos integrados comerciales

como microcontroladores, compuertas lógicas, contadores, etc. Esta interfaz nos permite la conexión entre dos elementos con una tasa de transferencia de hasta 8 Mbps.

Esta interfaz nos permite conectar elementos con una distancia máxima cercana a los 9 metros.

2.1.3 El puerto USB.

Debido a la sencillez de conexión de los puertos RS-232 o bien del paralelo de impresión, muchos diseñadores de periféricos optaron por comunicar sus productos a través de estas interfaces. A pesar de estas ventajas, los puertos mencionados se ven limitados en número en una computadora, por lo que otros diseñadores decidieron utilizar el *bus* PCI. Sin embargo este *bus* presenta un gran inconveniente, implica el desarrollo de su propia interfaz, obligando al usuario a instalar hardware para poder utilizar el producto.

Como respuesta a esta problemática surge el "*Universal Serial Bus*", mejor conocido como *USB*. Esta solución fue desarrollada con la colaboración varias empresas, entre ellas *Compaq*, *IBM*, *Intel* y *Microsoft*, buscando una conexión tan sencilla como enchufar una clavija a la toma de corriente. Adicionalmente se quería que el desarrollo de periféricos fuera sencillo y de bajo costo.

Esta interfaz ofrece una gran cantidad de beneficios tanto para usuarios como para desarrolladores de periféricos, entre los que destacan:

- Para el usuario:
  - Permite conectar múltiples productos a la vez.
  - El periférico se configura automáticamente la primera vez que se conecta a la computadora.
  - No hay necesidad de abrir la computadora para instalar una interfaz de conexión, solo se introduce el conector.
  - Se puede conectar en cualquier momento, sin requerir que el equipo se reinicie.

- Para el desarrollador:
  - Como se menciona más adelante, el puerto *USB* cuenta con 3 distintas velocidades y 4 tipos de transferencia, lo que lo vuelve versátil.
  - Los sistemas operativos actuales cuentan con soporte para la conexión y desconexión del dispositivo. Además, genera una relación entre los dispositivos y los "controladores" necesarios para emparejarlos al momento en que se conectan.
  - Existen microcontroladores con capacidad para *USB*, lo que facilita el desarrollo de periféricos desde cero.

El puerto *USB* presenta multiplicidad de conectores, sin embargo, la estructura de todos ellos es la misma. El cable maneja 4 hilos, dos de alimentación y dos de datos; en los conectores pequeños maneja un hilo adicional sin otra función que la de distinguir el tipo de conector. A diferencia de la interfaz serial RS-232, los canales de comunicación son bidireccionales y ambos manejan el mismo sentido de transmisión a la vez; esto con la finalidad de dar una cierta redundancia en los datos transmitidos para asegurar su fidelidad.

El cable de conexión puede presentar una longitud máxima de 5 metros. Como las velocidades de transmisión del cable son muy altas, éste se convierte en una muy buena antena [1], razón por la cual el cable debe de ser blindado, contrarrestando el efecto indeseado y elevando su costo.

La velocidad de transmisión máxima que puede presentar este puerto es de 480Mb por segundo, mostrando una notable superioridad ante los otros puertos tratados. Adicionalmente el máximo número de elementos que se pueden conectar es de 127, haciendo uso de *hubs*. Cabe aclarar que la velocidad máxima depende de diversos factores como se tratará más adelante.

2.2 Bases del puerto USB.

Como se puede observar en la sección anterior, a nivel desarrollador, el puerto *USB* no es cuestión de sólo conectar un cable especial. Para poder desarrollar un periférico conectado a través de esta interfaz es necesario saber un poco sobre cómo funciona y las tareas que se deben de ejecutar, como la enumeración y el envío de paquetes.

En ésta sección se asume una conexión *full-speed*, que corresponde a la velocidad alta en las especificaciones del *USB* 1.1 y a la velocidad media en las especificaciones del *USB* 2.0

2.2.1 Señal diferencial.

Al analizar las líneas D+ y D- del *USB*, cuando está trabajando, mediante un osciloscopio conectado directamente a ellas, se puede observar que usualmente estas líneas muestran una señal diferencial. Esto quiere decir que mientras una línea está en el nivel alto lógico la otra se encuentra en el nivel bajo.

Toda comunicación *USB* es iniciada por el hub raíz, embebido en el dispositivo "*host*" y es repetida por los *hubs* subsecuentes. La conexión *USB* es realizada punto a punto con el tipo de comunicación *half-duplex*, lo que significa que sólo existen transmisiones en un sentido a la vez. El protocolo presentado en la figura 2.3 espera que las líneas de datos cambien durante la transmisión de información. Sin embargo, entre las líneas del *USB* no se encuentra una señal de reloj, resultando en un tipo de comunicación asíncrona.

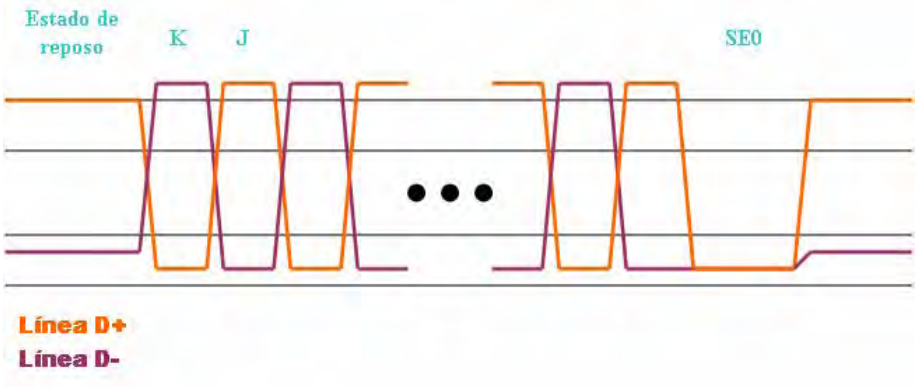


Figura 2.3. Señal del *USB*, usualmente diferencial.

2.2.2 Los paquetes.

El elemento fundamental en la comunicación USB es el paquete. Un paquete está formado por tres partes: un inicio, información y un final, como se muestra en la figura 2.4. Para explicaciones subsecuentes nombraremos al estado en el cual la línea D+ se encuentra en nivel alto y la línea D- en nivel bajo como estado "J" y como estado "K" al inverso de este.

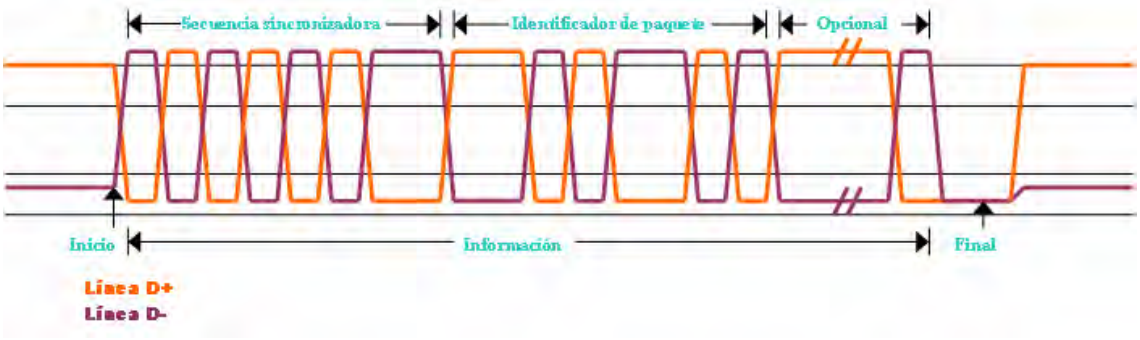


Figura 2.4. Estructura básica del paquete.

El inicio del paquete es señalado por una transición del estado “J”, el cual es también el estado de reposo, al estado “K”.

La información es un bloque de tamaño variable ordenado en bytes. El primer byte, como se puede ver en la figura, es una secuencia sincronizadora inmediatamente después del inicio. Esta secuencia está conformada por seis transiciones de estados, dando como resultado la secuencia KJKJKJKK. El receptor utiliza esta señal para sincronizar el reloj de recepción con las transiciones de datos, asegurando así una transmisión confiable. El segundo byte, y en algunos casos el último, es un identificador de paquetes, el cual puede verse como un comando predefinido. El identificador de paquetes está formado por 4 bits y su complemento; ofreciendo así un verificador de información basado en la redundancia del byte. Después del identificador de paquetes vienen datos opcionales los cuales se mencionan más adelante.

El final del paquete es una condición especial del *bus* en la que ambas líneas son llevadas al nivel lógico bajo durante dos tiempos. Esta señal se conoce como “*Single-Ended Zero*”.

Existen distintos tipos de paquetes:

- Paquetes de configuración, figura 2.5.
  - *SOF*: Este paquete se presenta para dividir el envío de datos en periodos de tiempo llamados recuadros.
  - *IN*: Este paquete pide al dispositivo conectado enviar información al host.
  - *OUT*: Este paquete solicita al dispositivo conectado que reciba información del host.

- *SETUP*: Este es un caso especial del paquete *OUT*, de alta prioridad. Prepara al dispositivo para recibir datos sobre su configuración.

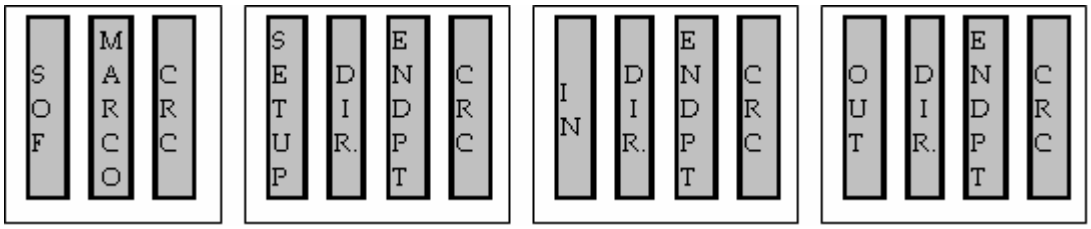


Figura 2.5. Estructura de los paquetes de configuración.

- Paquetes de datos, figura 2.6.
  - *Data0*: Paquete de dimensión variable que, alternado con *Data1*, permite la transmisión de información.
  - *Data1*: Paquete de dimensión variable que permite la transmisión de información.

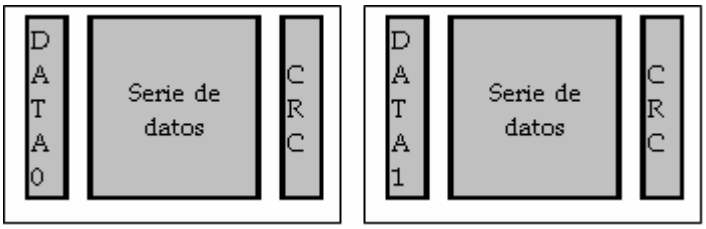


Figura 2.6. Estructura de los paquetes de datos.

- Paquetes de estado, figura 2.7.
  - *ACK*: Este paquete es enviado después de recibir correctamente la información.
  - *NACK*: Este paquete es enviado cuando no se puede recibir la información.
  - *STALL*: Este paquete es enviado cuando existe una falla en la transmisión.

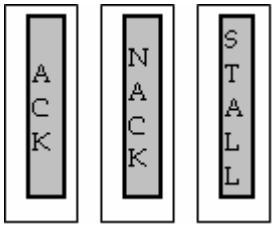


Figura 2.7. Estructura de los paquetes de estado.

- Paquetes especiales, figura 2.8.
  - *PRE*: Este paquete alerta la comunicación *low-speed*, debido a que esta no soporta todos los modos de transferencia.

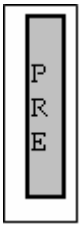


Figura 2.8. Estructura del paquete *PRE*.

2.2.3 La enumeración.

Como se ha mencionado antes, una de las ventajas del *USB* es que este presenta la característica de *hot-pluggable*. Esta característica implica que el *host* no puede tener una base de datos estática de los dispositivos conectados a ella. Aquí es donde entra la enumeración.

La enumeración es un proceso por el cual el *host* otorga nuevas direcciones a los dispositivos recién conectados para así poder comunicarse con ellos. Adicionalmente, el *host* requiere saber qué es el dispositivo recién conectado para saber cómo tratarlo, la enumeración es el proceso como esto se lleva a cabo.

Según las especificaciones del *USB* el dispositivo posee seis estados. Durante la enumeración el dispositivo pasa por 4 de ellos: encendido, *default* (por omisión), direccionado y configurado.

Los pasos a continuación son una secuencia típica de los eventos que ocurren durante la enumeración. Sin embargo, no se debe de suponer que los pasos son necesariamente ordenados, debiendo así de responder ante cada situación al momento de presentarse.

I. Conexión del dispositivo al puerto *USB*.

El hub alimenta al dispositivo poniéndolo en el estado de encendido. En este estado, aunque el dispositivo esté funcionando, todavía no presenta ningún flujo de información.

II. Detección, por parte del *hub*, del dispositivo.

El *hub* monitorea el voltaje en los puertos. El *hub* posee una resistencia *pull-down* en cada una de las líneas de señal; y el dispositivo posee una resistencia *pull-up* ya sea en la línea D+ (si es *full-speed* o *high-speed*) o en la línea D- (si es *low-speed*).

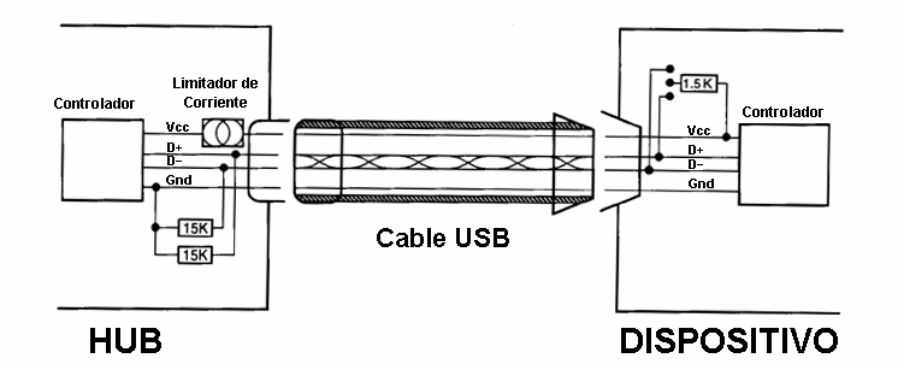


Figura 2.9. Detección del dispositivo.

Debido a la conexión del dispositivo, y su resistencia *pull-up*, figura 2.9, el *hub* sabe de la existencia del dispositivo. En esta etapa el dispositivo sigue en el estado de encendido.

III. El *host* cae en cuenta del nuevo dispositivo.

Mediante el *endpoint* de interrupción de los *hubs*, estos alertan al *host* del nuevo dispositivo

IV. El *hub* detecta la velocidad del dispositivo.

Como se había dicho, el dispositivo cuenta con una resistencia *pull-up* en alguna de las dos líneas de señal del *USB*. El hub reconoce la velocidad del dispositivo determinando cuál línea tienen el voltaje más alto.

V. El *hub* reinicia el dispositivo.

Cuando el *host* se percata de la conexión del nuevo dispositivo, manda una señal al *hub* para que éste envíe una condición de reinicio. La condición de reinicio es una condición especial en la que ambas líneas son llevadas al nivel bajo durante 10 milisegundos ó más. La señal de reinicio la envía el *hub* al que está conectado el dispositivo

y sólo al puerto en el que está conectado, haciendo invisible esta señal para otros dispositivos.

VI. El *host* detecta si el dispositivo es capaz de manejar la *high-speed*.

Si el dispositivo es capaz de manejar la *high-speed*, éste envía durante el estado de reinicio una señal de estado K, indicando así al *hub* su capacidad. Este responde enviando una serie de transiciones de estados K y J para indicar al dispositivo que quite su resistencia *pull-up* y que el resto de la comunicación se realizará a *high-speed*.

VII. Se establece el canal de comunicación entre el *hub* y el dispositivo.

El *host* espera a que el dispositivo salga del estado de reinicio. Al hacerlo, el dispositivo se encuentra en el estado *default*. Este estado le otorga la dirección 0 y le permite iniciar comunicación a través del *endpoint* 0 de control.

VIII. El *host* obtiene el tamaño máximo de los paquetes de datos a enviar.

El *host* envía una solicitud al dispositivo para que, mediante un descriptor, este regrese el número de bytes que se van a utilizar para la comunicación de control. Después de que el dispositivo responde, el *host* solicita al *hub* una señal de reinicio para asegurar que al salir del reinicio el dispositivo se encuentre en un estado conocido.

IX. El *host* otorga una dirección al dispositivo.

El *host* envía una dirección única al dispositivo, poniéndolo así en el estado direccionado. A partir de este momento, toda comunicación entre el *host* y el dispositivo se va a realizar a través de esta dirección.

X. El *host* obtiene las características del dispositivo.

El *host* envía una señal al dispositivo para que éste le regrese sus características. Toda esta información es enviada a través de descriptores, que como se verá más adelante, son estructuras muy bien definidas que presentan información sobre configuraciones, tamaños de paquetes, tipos de transferencias usadas, etc.

XI. El *host* asigna y carga el controlador.

En general, en esta etapa el *host* busca la manera de manejar la información que intercambia con el dispositivo.

En el caso de una computadora con *Windows* como *host*, el sistema operativo busca un controlador basado en los identificadores de producto y de vendedor (*PID* y *VID* respectivamente por sus siglas en ingles) en su base de datos, en caso de que el dispositivo no se encuentre en la base de datos pedirá al usuario ingresar el controlador. Sin embargo existe una clase de dispositivos (*Human Interface Device*) que no necesariamente requieren un controlador para poder funcionar.

XII. El *host* selecciona la configuración del dispositivo.

Un dispositivo puede tener más de una configuración programada. En caso de ser así, el *host* selecciona una y el dispositivo atiende al llamado operando de aquí en adelante bajo esa configuración.

Al seleccionar una configuración podemos decir que el dispositivo se encuentra en el estado configurado.

2.2.4 Los descriptores.

Un descriptor es una estructura bien definida cuya finalidad es almacenar de manera correcta las características del dispositivo. Cada descriptor contiene valores propios que lo identifican como cierto tipo de descriptor, los tipos estándar se muestran en la tabla 2.3. Adicionalmente un descriptor incluye en sus primeros bytes el tamaño (en bytes) del mismo.

ID	Descriptor	¿Es requerido?
01h	Dispositivo	Sí
02h	Configuración	Sí
03h	Cadena	No
04h	Interfaz	Sí
05h	*Endpoint*	No si sólo se usa el endpoint 0
06h	Calificador de dispositivo	Sí, si el dispositivo soporta *high-speed*
07h	Otras configuraciones de velocidad	Sí, si el dispositivo soporta *high-speed*
08h	Interfaz de alimentación	No
09h	*On The Go* (OTG)	Sólo para dispositivos OTG
0Ah	Depuración	No
0Bh	Asociaciones de interfaz	Para dispositivos compuestos

Tabla 2.3. Tipos de descriptores y su identificador.

A continuación se tratan los descriptores principales y de utilidad para esta tesis.

Descriptor de dispositivo.

Este descriptor contiene información acerca del dispositivo. Este dispositivo es el primero en ser leído por el *host*. Este descriptor tiene 14 campos listados en la tabla 2.4.

Offset	Descripción	Bytes
0	Tamaño del descriptor	1
1	ID del descriptor	1
2	Especifiación USB (BCD)	2
4	Clase	1
5	Subclase	1
6	Protocolo	1
7	Tamaño máximo del endpoint 0	1
8	ID del vendedor	2
10	ID del producto	2
12	Versión del dispositivo (BCD)	2
14	Índice del descriptor de cadena del productor	1
15	Índice del descriptor de cadena del producto	1
16	Índice del descriptor de cadena con el número de serie	1
17	Número de posibles configuraciones	1

Tabla 2.4. Campos del descriptor de dispositivo.

Descriptor de configuración.

Después de obtener el descriptor de dispositivo, el *host* busca obtener los descriptores de configuración, interfaz y *endpoints*.

Cada dispositivo tiene al menos una configuración, que a menudo es la única necesaria, sin embargo se pueden presentar más. Cada configuración requiere de su propio descriptor.

Offset	Descripción	Bytes
0	Tamaño del descriptor	1
1	ID del descriptor	1
2	Tamaño del descriptor y todos sus subordinados	2
4	Número de interfaces en la configuración	1
5	Número de configuración	1
6	Índice del descriptor de cadena de la configuración	1
7	Configuración de alimentación	1
8	Máxima corriente requerida (miliamperes/2)	1

Tabla 2.5. Campos del descriptor de configuración.

Los descriptores de configuración tienen como subordinados a los descriptores de interfaz y a los de endpoint, los cuales también pueden ser más de uno. Este descriptor cuenta con 8 campos listados en la tabla 2.5.

Descriptor de cadena.

El descriptor de cadena es un tipo de descriptor especial que nos permite ingresar texto descriptivo de la funcionalidad de cada descriptor.

Éste descriptor sólo cuenta con 3 campos listados en la tabla 2.6.

Offset	Descripción	Bytes
0	Tamaño del descriptor	1
1	ID del descriptor	1
2	Una cadena de texto en formato Unicode	Varía

Tabla 2.6. Campos del descriptor de cadena.

Descriptor de interfaz.

Un descriptor de interfaz tiene información acerca de la función o característica que implementa. El descriptor incluye la clase, subclase y protocolos para tratar la información.

Éste descriptor tiene 9 campos listados en la tabla 2.7.

Offset	Descripción	Bytes
0	Tamaño del descriptor	1
1	ID del descriptor	1
2	Número de la interfaz	1
3	Valor de una configuración alterna	1
4	Número de endpoints que soporta	1
5	Clase	1
6	Subclase	1
7	Protocolo	1
8	Número del descriptor de cadena asociado	1

Tabla 2.7. Campos del descriptor de interfaz.

Descriptor de *endpoint*.

Cada *endpoint* especificado en una interfaz debe de proporcionar un descriptor. Existe una excepción, el *endpoint* 0, ya que éste siempre debe de estar implementado por que es el que sirve para el control de la comunicación como se mencionó en la sección de enumeración.

Éste descriptor presenta 6 campos listados en la tabla 2.8.

Offset	Descripción	Bytes
0	Tamaño del descriptor	1
1	ID del descriptor	1
2	Número de endpoint y dirección	1
3	Tipo de transferencia	1
4	Tamaño máximo de paquete	2
5	Máximo tiempo sin respuesta	1

Tabla 2.8. Campos del descriptor de endpoint.

2.2.5 Tipos de transferencias.

Existen 4 tipos de transferencias para la comunicación *USB*, cada una tiene rasgos característicos que la hace mejor para ciertos tipos de aplicación. Las clases entre las que diferenciamos son las siguientes: control, interrupción, masiva e isócrona.

Control.

La transferencia de control tiene dos usos. Lleva las solicitudes definidas por la especificación *USB* usadas por el *host* para configurar los dispositivos y aprender sobre características de los mismos. También sirve para realizar transacciones definidas por el usuario con cualquier finalidad.

Masiva.

Las transferencias masivas son útiles para la transmisión de datos cuando el tiempo no es crítico. Esta transferencia puede enviar grandes cantidades de información sin saturar el *bus*, ya que en este modo se espera a que el *bus* esté disponible. Además, en caso de que el *bus* este libre, éste tipo de transferencia es el más rápido.

Interrupción.

Las transferencias de interrupción son útiles cuando la información tiene que transmitirse en tiempos específicos. Como el nombre lo sugiere, el dispositivo puede espontáneamente enviar datos que desaten una función en el *host*. Sin embargo, como todos los demás tipos, el dispositivo sólo puede enviar información si el *host* lo solicita, por lo que sólo es una simulación de interrupción, donde la información está garantizada con un retraso mínimo.

Isócrona.

Las transferencias isócronas son transferencias en tiempo real, útiles cuando mucha información debe de llegar en un tiempo definido y se pueden aceptar errores ocasionales. A diferencia de las transmisiones de interrupción, la información se envía constantemente, pero dado que se busca transmitir una gran cantidad de información no hay tiempo para verificar errores.

A continuación se muestra en la tabla 2.9 un resumen de los tipos de transferencias.

Transferencia	Atributos	Tamaño máximo [bytes]	Ejemplo
Control	Tiempo y calidad	64	Control del sistema
Interrupción	Tiempo y calidad	64	Teclado
Masiva	Calidad	64	Impresora
Isócrona	Tiempo	1024	Bocinas

Tabla 2.9. Resumen de los tipos de transferencias.

2.2.6 Clases.

La mayoría de los dispositivos *USB* tienen muchas funciones en común con otros dispositivos que realizan funciones similares. Cuando un grupo de dispositivos o interfaces comparten muchos atributos o proveen servicios similares, comienza a tener sentido hablar de clases predefinidas que faciliten el desarrollo.

De este modo, las clases son definiciones y protocolos para tratar a dispositivos con funciones similares.

El hablar de clases ofrece la ventaja de servir como guía para desarrolladores de controladores, o bien los sistemas operativos pueden incluir controladores comunes para muchas tareas.

A pesar de que un dispositivo basado en una clase presente funciones que no están contenidas en ella, el desarrollador puede crear sólo una variante o complemento del controlador de la clase para agregar la funcionalidad deseada, en lugar de desarrollar el controlador en su totalidad.

A continuación se enlistan las clases más comunes:

- Audio.
- Comunicación.
- Contenido de seguridad.
- Dispositivo de actualización de *firmware*.
- Dispositivo de interfaz humana.
- IrDA.
- Dispositivos de almacenamiento.
- Impresoras.
- Cámaras.
- Mediciones y pruebas.
- Video.

2.3 Los microcontroladores PIC®.

Los microcontroladores son circuitos integrados que ofrecen, como el nombre lo dice, un sistema completo de control configurable. Los microcontroladores están pensados para aplicaciones en las que el dispositivo debe de realizar un pequeño número de tareas al menor costo posible.

Se puede ver entonces a un microcontrolador como una solución embebida en un solo circuito integrado, capaz de resolver tareas de forma autónoma.

Siendo uno de los logros más sobresalientes en el mundo de la electrónica, los microcontroladores se encuentran presentes en la mayoría de los dispositivos electrónicos utilizados diariamente, como lo son los hornos de microondas, los televisores, los refrigeradores, etc.

2.3.1 Microcontroladores vs. Microprocesadores.

Un microprocesador es un circuito integrado que contiene una unidad central de procesamiento, o *CPU* (por sus siglas en inglés) como se manejará de aquí en adelante. El *CPU* puede considerarse como el intérprete y el ejecutor de instrucciones predefinidas.

Así, los pines de un microprocesador son el medio de comunicación de los *buses* de direcciones, datos y control del *CPU*, permitiéndole conectarse a través de ellos a distintos periféricos.

Entre los periféricos más comunes que se pueden conectar al microprocesador se encuentran la memoria de programa, la memoria de datos y los módulos de entrada-salida que permiten una interacción con el medio. Esto permite que la configuración sea completamente configurable a la aplicación.

El microcontrolador por su parte es la solución que conjunta lo antedicho, es decir, posee un microprocesador conectado internamente a los periféricos propios del sistema, figura 2.10. Se puede deducir de lo anterior que no es posible entonces modificar del todo la configuración interna del circuito. Para resolver este problema, se busca que los microcontroladores sean multifuncionales, configurando su funcionalidad mediante software.

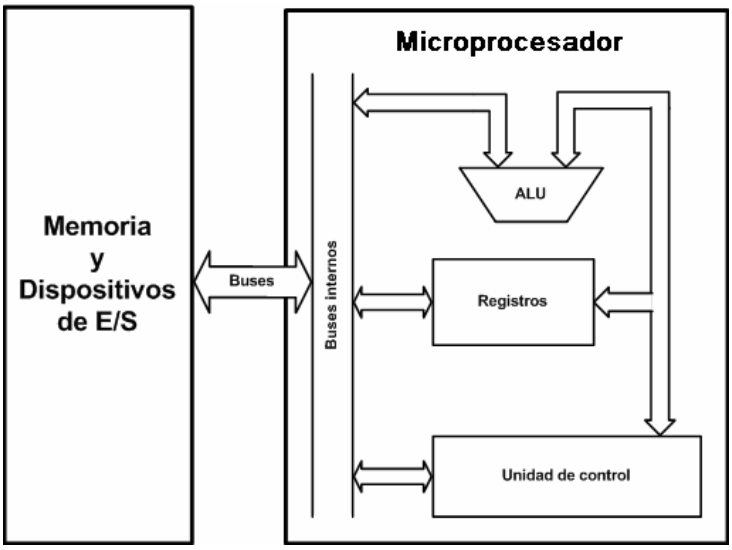


Figura 2.10. Diagrama general de un microcontrolador.

Si se visualiza lo anterior en un modelo escalado a una computadora personal, el microprocesador es el circuito encargado del procesamiento, en este caso al microprocesador se le identifica como el procesador *Pentium*, *Athlon* o símil; mientras que el microcontrolador es la computadora en su totalidad, el microprocesador conectado a la tarjeta madre que a su vez está conectada a las unidades de disco, a la memoria *RAM*, a los dispositivos de entrada como teclado y *mouse*, a los dispositivos de salida como bocinas y

monitor, etc. Sin embargo, para completar la analogía, la computadora no puede ser configurable, es decir, si suponemos una computadora con una memoria *RAM* de tamaño "X" y con "Y" número de puertos, no se deben de modificar (internamente) estas condiciones.

2.3.2 Clasificación.

Los "PIC" son una familia de microcontroladores fabricados por la empresa *Microchip Technology Inc*. Estos microcontroladores se presentan en una amplia variedad de productos.

Principalmente podemos dividir a la familia por el tamaño de palabra [1] que utilizan. Los tamaños de palabra que *Microchip* maneja son: palabra de 32 bits, palabra de 16 bits y palabra de 8 bits. Siendo esta última la de mayor popularidad para aplicaciones generales.

A su vez, se subdividen los microcontroladores con palabra de 8 bits en 3 gamas distintas, las cuales se enuncian a continuación:

- Gama baja.
  - 33 instrucciones de 12 bits.
  - Memoria de programa de hasta 2048 palabras de 12 bits.
  - No tienen interrupciones.
  - Número reducido de terminales.
  - Pueden disponer de un temporizador y un contador.
- Gama media.
  - 35 instrucciones de 14 bits.
  - Memoria de programa de hasta 8192 palabras de 14 bits.
  - En general poseen *EEPROM* de datos.

[1] El tamaño de palabra refiere al número de bits que maneja el CPU para sus operaciones.

- Poseen diversas interrupciones.
- En general cuentan con una amplia variedad de dispositivos de entrada y salida.
- Disponen de hasta 3 temporizadores, dos módulos de *CCP* [2], varios tipos de puerto serie, convertidor A/D, etc.

- Gama alta
  - 58 o 77 [3] instrucciones de 16 bits.
  - Memoria de programa de hasta 210 palabras de 16 bits.
  - Diversos modos de funcionamiento.
  - Interrupciones jerarquizadas.
  - En general mejoran todas las características de los PIC's de gama media.

2.3.3 Arquitectura.

Estos microcontroladores manejan una arquitectura *RISC*-Harvard. Esto se explica a continuación.

Arquitectura Harvard.

Las instrucciones de un programa y los datos deben de ser almacenados en una memoria. Las instrucciones deben de pasar de forma secuencial de la memoria al *CPU* para su decodificación y ejecución. Mientras tanto, los datos pueden ser leídos de o escritos a la memoria.

Con esto se puede pensar que la organización de la memoria y la conexión de esta con el *CPU* son de suma importancia y afectan al desempeño del microcontrolador; la arquitectura Harvard es una de las soluciones.

---

[2] Módulo de comparación, captura y modulación por ancho de pulso (CCP por sus siglas en inglés).

[3] La serie 17 maneja hasta 58 instrucciones mientras la serie 18 maneja hasta 77.

Esta arquitectura consiste en separar la memoria de datos de la memoria de programa (figura 2.11). De este modo, mediante una conexión paralela, el *CPU* puede acceder de manera simultánea a las dos memorias. Se asignan *buses* de datos, de direcciones y de control independientes para ambas memorias.

Mientras que la memoria de programa es, al menos en principio, de solo lectura; en la memoria de datos se puede leer y escribir.

A pesar de la complejidad de conexión de la arquitectura Harvard, es evidente que ofrece una gran ventaja en cuanto a velocidad de operación.

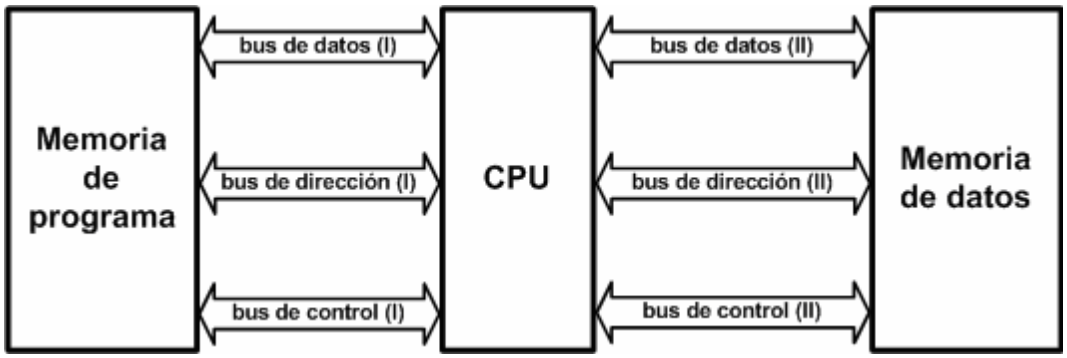


Figura 2.11. Esquematización de la arquitectura Harvard.

Arquitectura *RISC*.

En la arquitectura *RISC* (*Reduced Instruction Set Computer*), el *CPU* dispone de un repertorio de instrucciones sencillas. Cada instrucción puede realizar una operación muy simple pero a una alta velocidad.

Se descartó el uso de instrucciones complejas después de observar que las operaciones avanzadas, de propósito dedicado, eran escasamente utilizadas y además se podía lograr lo mismo haciendo uso de instrucciones simples. Adicionalmente, los *CPU* comenzaron a trabajar a velocidades mayores a las de la memoria con la que se comunicaban, haciendo inútil el desarrollo de diversos métodos de direccionamiento.

La arquitectura *RISC*, al utilizar un conjunto pequeño de instrucciones simples, requiere una menor complejidad del *CPU* que las decodifica, ahorrando espacio y permitiendo aumentar la frecuencia del oscilador (aumentando la velocidad de operación).

2.3.4 Características generales.

Registro de trabajo (registro *W*) y la unidad aritmética y lógica (*ALU*)

Uno de los componentes fundamentales de la *CPU* es la *ALU*, la cual se encarga de las operaciones aritméticas y lógicas previstas por las instrucciones. La *ALU* tiene asociado un registro (registro *W*), en el cual se almacena el resultado de las operaciones, o bien es un receptáculo temporal de uno de los datos que intervienen en la operación.

Adicionalmente, existen ciertos bits que indican estados especiales de la *ALU* como acarreos, desbordamientos, préstamo, entre otros, los cuales son almacenados en el registro de estado (registro *STATUS*).

Ejecución de las instrucciones.

Como todos los microcontroladores, los PIC tienen un oscilador que dicta la frecuencia en que se desenvuelven las operaciones internas. En el caso de los PIC, los pulsos generados son divididos internamente para generar 4 señales que sincronizan el trabajo del microcontrolador, es decir, cada 4 pulsos el procesador ejecuta un ciclo de máquina.

En el primer pulso, el contador de programa se incrementa en una unidad, apuntando a la instrucción que ha de realizarse. Paralelo a este ciclo de máquina se está ejecutando la instrucción anterior (esto es conocido como *pipeline*).

La ejecución de cualquier instrucción se realiza de la siguiente forma: se busca la instrucción que se debe de ejecutar, se decodifica la instrucción y por último se ejecuta.

Para propiamente ejecutar una instrucción se requieren dos ciclos de máquina, en el primer ciclo se busca la instrucción y en el segundo ciclo se decodifica y se ejecuta. Es aquí donde se ve la utilidad del *pipeline*, donde a pesar de que una instrucción requiere de dos ciclos de máquina para ejecutarse, al hacer en paralelo la búsqueda de la siguiente operación y la decodificación y ejecución de la actual, se obtiene en promedio una ejecución por ciclo de máquina.

Los osciladores.

En general los microcontroladores PIC disponen de las siguientes opciones para escoger el oscilador principal:

- Oscilador externo.
- Oscilador interno.
- Oscilador RC (Resistencia-Capacitor).
- Oscilador de cristal:
  - Aplicaciones de baja potencia.
  - Aplicaciones a frecuencias medias.
  - Aplicaciones de alta frecuencia.

Reset.

El *reset* de un microcontrolador hace que éste vaya a un estado conocido, en particular para los PIC, el contador de programa toma el valor de 0.

El *reset* puede ser provocado por diversas fuentes, algunas de ellas configurables por software. Las fuentes de *reset* son las siguientes:

- *Reset* por código.
- *Reset* externo.
- *Reset* por encendido.
- *Reset* por desbordamiento del *WDT* (*Watchdog Timer*).
- *Reset* por *Brown-Out* (Voltaje de operación estable).

***WDT* (*Watchdog Timer*).**

El *WDT* es un temporizador que garantiza que el programa no caiga en un ciclo infinito. Partiendo de un oscilador interno independiente del oscilador principal, el *WDT* se incrementa de forma proporcional a la frecuencia de este oscilador. Al desbordarse el contador del *WDT* se produce un reset indicando que pasó un determinado tiempo sin que

el procesador hiciera lo que debía. Para evitar que se desborde el contador es necesario eventualmente reiniciar el contador desde código.

Modo de bajo consumo (*SLEEP*).

El modo de bajo consumo es una función que tienen los microcontroladores para el ahorro de energía. En caso de no ser necesario el procesamiento de información, se puede poner al microcontrolador en este modo, suspendiendo casi todas sus funciones.

Para poder salir de este modo es necesaria una de las siguientes condiciones:

- Un *reset*.
- Desbordamiento del *WDT*.
- Una interrupción externa o procedente de alguno de los módulos internos que aún operen.

Al salir del estado de reposo el microcontrolador ejecuta la instrucción en la que se quedó, es decir, la que sigue a la instrucción *sleep*.

Interrupciones.

Una interrupción es un evento asíncrono que hace que el microcontrolador interrumpa el flujo normal de operación para ejecutar una línea de programa distinto. Usualmente, al terminar el programa de interrupción el microcontrolador regresa al curso natural justo con la instrucción en la que terminó.

Para esto, al proceder una interrupción, se guarda el contador del programa en una pila conocida como *Stack Pointer*, ejecuta la subrutina y al terminarla llama al valor guardado para proceder con la siguiente línea de código normal.

Como en un microcontrolador suelen existir diversas fuentes de interrupción la subrutina suele verificar cuál fue la fuente de interrupción y actuar en consecuencia.

Existen interrupciones tanto internas como externas. Las interrupciones externas tienen lugar en los puertos de comunicación, mientras que las interrupciones internas son generadas por eventos especiales de los módulos periféricos.

Entre las posibles fuentes de interrupción están las siguientes:

- Interrupción externa por el pin INT del microcontrolador.
- Interrupción por cambio en el nivel lógico de las entradas RB4:RB7.
- Interrupción por desbordamiento de los temporizadores.
- Interrupción por algún evento en el modulo *CCP*.
- Interrupción por el puerto *USART*.
- Interrupción por el convertidor A/D.

Bits de configuración.

Todos los PIC disponen de un cierto número de palabras para configurar el modo de operación y funciones especiales del microcontrolador. Estos bits están almacenados en una memoria volátil, sin embargo no son accesibles durante el funcionamiento. Los valores de estos bits son cambiados sólo durante la programación.

La finalidad de estos bits es darle al usuario la flexibilidad de configuración que necesita para personalizar las características y adaptarlo a sus necesidades.

Las características más destacadas que se pueden programar con estos bits son:

- El tipo de oscilador.
- La habilitación del *WDT*.
- La protección de memoria del programa.
- La protección de memoria de datos.
- Las características de *reset* del dispositivo.

Funciones especiales.

Existe una gran variedad de características adicionales propias de cada PIC. Algunas de las más comunes se citan a continuación:

- Módulos de comunicación.
  - *USART* (Receptor-Transmisor Serial Asíncrono Universal).
  - *SPI* (Interfaz de Periféricos Serial).

- I^2C (Circuitos Inter-Integrados).
- *CAN* (Red Controladora de Área).
- *USB* (*Bus* Serial Universal).
- *LIN* (Red Local Interconectada).
- Radio frecuencia.
- Ethernet.

- Modulo de Comparación, Captura y *PWM* (modulación por ancho de pulso).
- Contadores.
- Temporizadores.
- Convertidor analógico-digital.
- Controlador de *LCD*.

2.3.5 Configuración.

Como se mencionó en la sección 2.3.1 la configuración de las funciones de un microcontrolador se hace mediante software. Los microcontroladores PIC cuentan con registros especiales dedicados a esto.

Para cada una de las funciones existen uno o más registros y, a su vez, existen registros compartidos por varias funciones. Para mayor información sobre qué configura cada bit de estos registros, y cómo configurar en su totalidad la funcionalidad del microcontrolador, se debe referir a la hoja de especificaciones propia del circuito.

2.4 PIC18F4550.

Como se había mencionado anteriormente, el PIC18F4550 cumple con lo necesario para poder implementar la solución planteada.

El siguiente listado muestra las características más sobresalientes del circuito:

- Arquitectura de 8 bits.
- Memoria de programa tipo *FLASH* de 32 kB.

- Memoria de datos *EEPROM* de 256 B.
- Memoria *RAM* de 2 kB.
- Encapsulado de 40 ó 44 pines, de los cuales 35 son de entrada-salida divididos en 5 puertos.
- 75 instrucciones, 83 si se activan las instrucciones extendidas.
- Frecuencia máxima del oscilador principal de 48 MHz, proporcionando un máximo de 12 MIPS (Mega [4] Instrucciones Por Segundo).
- Oscilador interno configurable entre 32 kHz y 8MHz.
- 12 configuraciones para el oscilador.
- Tecnología *nanoWatt* incluida.
- 2 comparadores.
- Posee un convertidor analógico-digital, multiplexado a 13 canales de conversión.
- Módulo *USART*.
- Módulo *MSSP* para comunicación I^2C ó *SPI*.
- *USB* 2.0 para full-speed o low-speed.
- Módulo *LIN*.
- Posee un módulo de *CCP* normal y uno mejorado, con resolución del *PWM* de 10 bits.
- Un temporizador de 8 bits y tres de 16 bits compartidos con dos contadores.
- Puerto paralelo esclavo.
- 20 fuentes de interrupción.
- Prioridad en interrupciones.

---

[4] En términos informáticos, el prefijo “Mega” refiere a 2^{20} mientras que el prefijo “kilo” refiere a 2^{10}.

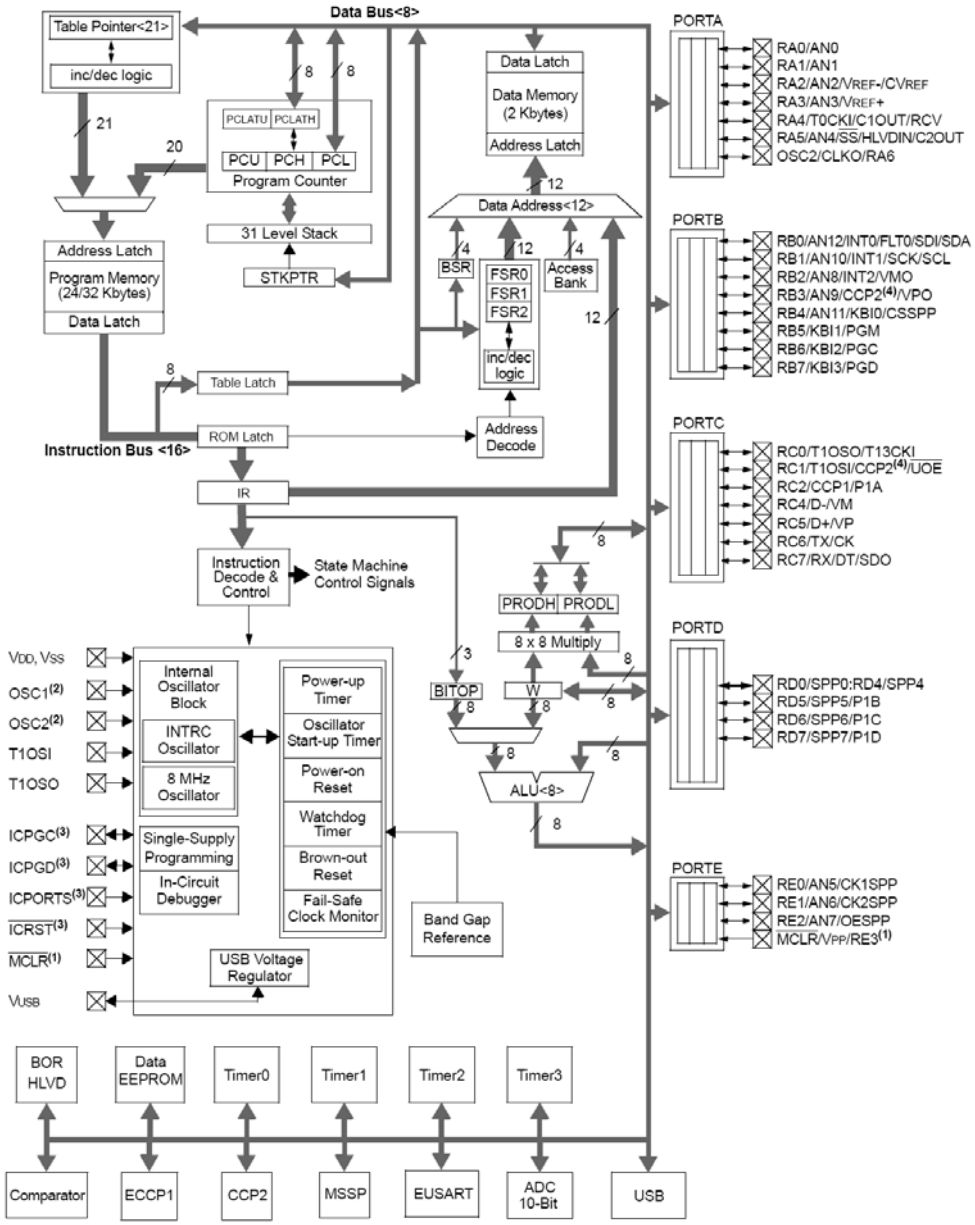


Figura 2.12. Arquitectura del microcontrolador PIC18F4550. Tomada de la hoja de especificaciones.

- *ICSP.*
- *ICD.*
- *Power on Reset.*
- *Brown out Reset.*
- *Watchdog Timer.*

- *Reset* por desbordamiento de pila.
- *High/Low Voltage Detect.*
- Rango de voltajes de operación entre 2 y 5.5 volts.

Como se había mencionado anteriormente, este PIC está basado en la arquitectura Harvard, como se puede ver en la figura 2.12 de la estructura interna.

Como se puede observar, hablar sobre este microcontrolador da para muchas páginas, es por esto que sólo se detalla lo más importante para el desarrollo de esta tesis.

2.4.1 Oscilador.

Como se ha mencionado, el oscilador es un elemento fundamental para el funcionamiento correcto del microcontrolador. Este microcontrolador cuenta con un oscilador interno, útil para aplicaciones en las que se busca minimizar los componentes a usar.

El microcontrolador cuenta con un módulo *PLL* que permite multiplicar la frecuencia de operación. Debido a los estándares de comunicación del *USB*, si se quiere operar en modo *low-speed* se necesita una frecuencia de operación de 6 MHz, en cuyo caso la frecuencia natural del procesador debe de ser de 24 MHz. Del mismo modo, si queremos utilizar el PIC para la conexión *USB full-speed* requerimos de una frecuencia de 48 MHz.

Las configuraciones que posee el PIC se enlistan a continuación:

- **XT** Oscilador de cristal para frecuencias medias.
- **XTPLL** Oscilador de cristal para frecuencias medias con el módulo *PLL* activado.
- **HS** Oscilador de cristal para frecuencias altas.
- **HSPLL** Oscilador de cristal para frecuencias altas con el módulo *PLL* activado.
- **EC** Reloj externo. Adicionalmente esta configuración brinda una salida con un cuarto de la frecuencia de entrada.
- **ECIO** Reloj externo habilitando el pin 14.

- **ECPLL** Reloj externo con el módulo *PLL* activado. Adicionalmente esta configuración brinda una salida con un cuarto de la frecuencia de entrada.
- **ECPIO** Reloj externo habilitando el pin 14 con el módulo *PLL* activado.
- **INTHS** Oscilador interno utilizado como fuente del microcontrolador. Oscilador de cristal de frecuencias altas usado para controlar el módulo *USB.*
- **INTXT** Oscilador interno utilizado como fuente del microcontrolador. Oscilador de cristal de frecuencias medias usado para controlar el módulo *USB.*
- **INTIO** Oscilador interno utilizado como fuente del microcontrolador. Oscilador externo usado para controlar el módulo *USB.* Habilita el pin 14.
- **INTCK0** Oscilador interno utilizado como fuente del microcontrolador. Oscilador externo usado para controlar el módulo *USB.* Salida con un cuarto de la frecuencia de entrada.

Toda la configuración necesaria se realiza, como en otros modelos de PIC, por medio de la modificación de registros dedicados.

2.4.2 Puertos.

Para hablar de los puertos atendemos a la nomenclatura de *Microchip*, mostrada en el patigrama del microcontrolador (figura 2.13). Este PIC cuenta con 5 puertos nombrados como A, B, C, D y E respectivamente con la distribución mostrada en la tabla 2.10.

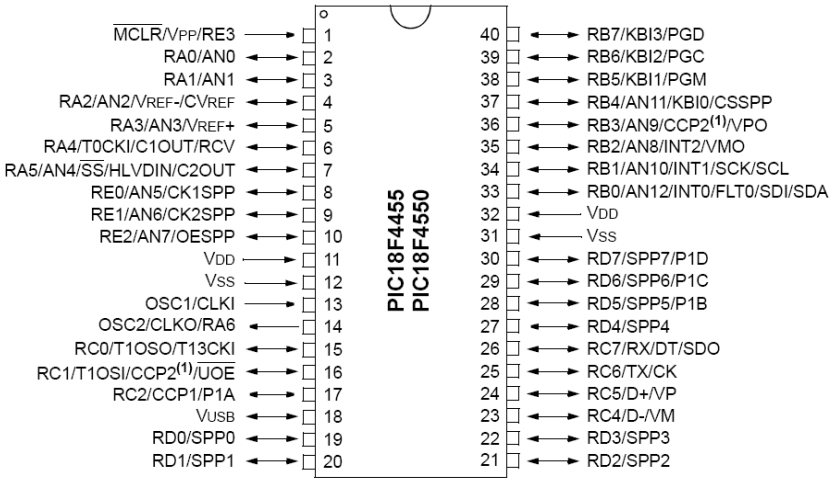


Figura 2.13. Patigrama del PIC18F4550. Tomada de la hoja de especificaciones.

Función	Pin	Sentido	Tipo de entrada	Descripción
RA0/AN0	2			
RA0		E/S	TTL	E/S Digital
AN0		E	Analógica	Entrada analógica 0
RA1/AN1	3			
RA1		E/S	TTL	E/S Digital
AN1		E	Analógica	Entrada analógica 1
RA2/AN2/ V_{REF-}/CV_{REF}	4			
RA2		E/S	TTL	E/S Digital
AN2		E	Analógica	Entrada analógica 2
V_{REF-}		E	Analógica	Referencia del ADC
CV_{REF}		O	Analógica	Referencia del comparador
RA3/AN3/V_{REF+}	5			
RA3		E/S	TTL	E/S Digital
AN3		E	Analógica	Entrada analógica 3
V_{REF+}		E	Analógica	Referencia del ADC
RA4/T0CKI/ C1OUT/RCV	6			
RA4		E/S	Schmitt Triger	E/S Digital
T0CKI		E	Schmitt Triger	Señal de reloj del Timer0
C1OUT		S	–	.
RCV		E	TTL	Entrada RCV del transceptor USB externo
RA5/AN4/SS/ HLVDIN/C2OUT	7			
RA5		E/S	TTL	E/S Digital
AN4		E	Analógica	Entrada analógica 4
#SS		E	TTL	Selector de esclavo para SPI
HLVDIN		E	Analógica	.
C2OUT		S	Analógica	Salida del comparador 2
OSC2/CLK0/RA6	14			
RA6		E/S	TTL	E/S Digital

Tabla 2.10. Descripción de las funciones de los puertos.

Función	Pin	Sentido	Tipo de entrada	Descripción
RB0/AN12/INT0/ FLT0/SDI/SDA	33			
RB0		E/S	TTL	E/S Digital
AN12		E	Analógica	Entrada analógica 12
INT0		E	Schmitt Triger	Interrupción externa 0
FLT0		E	Schmitt Triger	.
SDI		E	Schmitt Triger	Entrada de datos SPI
SDA		E/S	Schmitt Triger	Canal SDA de la I²C
RB1/AN10/INT1/ SCK/SCL	34			
RB1		E/S	TTL	E/S Digital
AN10		E	Analógica	Entrada analógica 10
INT1		E	Schmitt Triger	Interrupción externa 1
SCK		E/S	Schmitt Triger	Línea de reloj de SPI
SCL		E/S	Schmitt Triger	Línea de reloj de I²C
RB2/AN8/INT2/ VMO	35			
RB2		E/S	TTL	E/S Digital
AN8		E	Analógica	Entrada analógica 8
INT2		E	Schmitt Triger	Interrupción externa 2
VMO		S	—	Salida VMO del transceptor USB externo
RB3/AN9/CCP2/ VPO	36			
RB3		E/S	TTL	E/S Digital
AN9		E	Analógica	Entrada analógica 9
CCP2		E/S	Schmitt Triger	Canal del módulo CCP2
VPO		S	—	Salida VPO del transceptor USB externo

Tabla 2.10. (Continuación). Descripción de las funciones de los puertos.

Función	Pin	Sentido	Tipo de entrada	Descripción
RB4/AN11/KBI0/ CSSPP	37			
RB4		E/S	TTL	E/S Digital
AN11		E	Analógica	Entrada analógica 11
KBI0		E	TTL	Pin de interrupción por cambio de estado
CSSPP		S	–	.
RB5/KBI1/PGM	38			
RB5		E/S	TTL	E/S Digital
KBI1		E	TTL	Pin de interrupción por cambio de estado
PGM		E/S	Schmitt Triger	Pin de selección de LVP para ICSP
RB6/KBI2/PGC	39			
RB6		E/S	TTL	E/S Digital
KBI2		E	TTL	Pin de interrupción por cambio de estado
PGC		E/S	Schmitt Triger	Línea de reloj para ICSP
RB7/KBI3/PGD	40			
RB7		E/S	TTL	E/S Digital
KBI3		E	TTL	Pin de interrupción por cambio de estado
PGD		E/S	Schmitt Triger	Línea de datos para ICSP
RC0/T1OSO/ TI3CKI	15			
RC0		E/S	Schmitt Triger	E/S Digital
T1OSO		S	–	Salida de oscilador del Timer1
T13CKI		E	Schmitt Triger	Entrada de reloj del Timer1/Timer3
RC1/T1OSI/CCP2/ UOE	16			
RC1		E/S	Schmitt Triger	E/S Digital
T1OSI		E	CMOS	Entrada de oscilador del Timer1
CCP2		E/S	Schmitt Triger	Canal del módulo CCP2
#UOE		S	–	Salida #OE del transceptor USB externo

Tabla 2.10. (Continuación). Descripción de las funciones de los puertos

Función	Pin	Sentido	Tipo de entrada	Descripción
RC2/CCP1/P1A	17			
RC2		E/S	Schmitt Triger	E/S Digital
CCP1		E/S	Schmitt Triger	Canal del módulo CCP1
P1A		S	TTL	Canal A del PWM del módulo ECCP
RC4/D-/VM	23			
RC4		E	TTL	E/S Digital
D-		E/S	—	Línea de comunicación D- del USB
VM		E	TTL	Entrada VM del transceptor USB externo
RC5/D+/VP	24			
RC5		E	TTL	E/S Digital
D+		E/S	—	Línea de comunicación D+ del USB
VP		E	TTL	Entrada VP del transceptor USB externo
RC6/TX/CK	25			
RC6		E/S	Schmitt Triger	E/S Digital
TX		S	—	Línea de transmisión del módulo EUSART
CK		E/S	Schmitt Triger	Línea de reloj del módulo EUSART
RC7/RX/DT/SDO	26			
RC7		E/S	Schmitt Triger	E/S Digital
RX		E	Schmitt Triger	Línea de recepción del módulo EUSART
DT		E/S	Schmitt Triger	Línea de datos del módulo EUSART
SDO		S	—	Salida de datos de SPI
RD0/SPP0	19			
RD0		E/S	Schmitt Triger	E/S Digital
SPP0		E/S	TTL	Canal del SPP
RD1/SPP1	20			
RD1		E/S	Schmitt Triger	E/S Digital
SPP1		E/S	TTL	Canal del SPP

Tabla 2.10. (Continuación). Descripción de las funciones de los puertos

Función	Pin	Sentido	Tipo de entrada	Descripción
RD2/SPP2	21			
RD2		E/S	Schmitt Triger	E/S Digital
SPP2		E/S	TTL	Canal del SPP
RD3/SPP3	22			
RD3		E/S	Schmitt Triger	E/S Digital
SPP3		E/S	TTL	Canal del SPP
RD4/SPP4	27			
RD4		E/S	Schmitt Triger	E/S Digital
SPP4		E/S	TTL	Canal del SPP
RD5/SPP5/P1B	28			
RD5		E/S	Schmitt Triger	E/S Digital
SPP5		E/S	TTL	Canal del SPP
P1B		S	—	Canal B del PWM del módulo ECCP
RD6/SPP6/P1C	29			
RD6		E/S	Schmitt Triger	E/S Digital
SPP6		E/S	TTL	Canal del SPP
P1C		S	—	Canal C del PWM del módulo ECCP
RD7/SPP7/P1D	30			
RD7		E/S	Schmitt Triger	E/S Digital
SPP7		E/S	TTL	Canal del SPP
P1D		S	—	Canal D del PWM del módulo ECCP
RE0/AN5/CK1SPP	8			
RE0		E/S	Schmitt Triger	E/S Digital
AN5		E	Analógica	Entrada analógica 5
CK1SPP		S	—	Salida de reloj del SPP
RE1/AN6/CKSPP	9			
RE1		E/S	Schmitt Triger	E/S Digital
AN6		E	Analógica	Entrada analógica 6
CK2SPP		S	—	Salida de reloj del SPP

Tabla 2.10. (Continuación). Descripción de las funciones de los puertos

Función	Pin	Sentido	Tipo de entrada	Descripción
RE2/AN7/OESPP	10			
RE2		E/S	Schmitt Triger	E/S Digital
AN7		E	Analógica	Entrada analógica 7
OESPP		S	–	.
MCLR/V_{PP}/RE3	1			
RE3		E	Schmitt Triger	Entrada Digital

Tabla 2.10. (Continuación). Descripción de las funciones de los puertos.

Además, el puerto B cuenta con unas resistencias *pull-up* internas configurables por software.

2.4.3 USB.

El PIC18F4550 contiene una interfaz serial compatible con *full-speed* o *low-speed USB*, mostrado en la figura 2.14. Esta interfaz puede conectarse directamente a un *host USB* a través del transceptor embebido o a través de uno externo. Adicionalmente, el microcontrolador cuenta con un regulador de 3.3 V incluido para las aplicaciones que lo requieran.

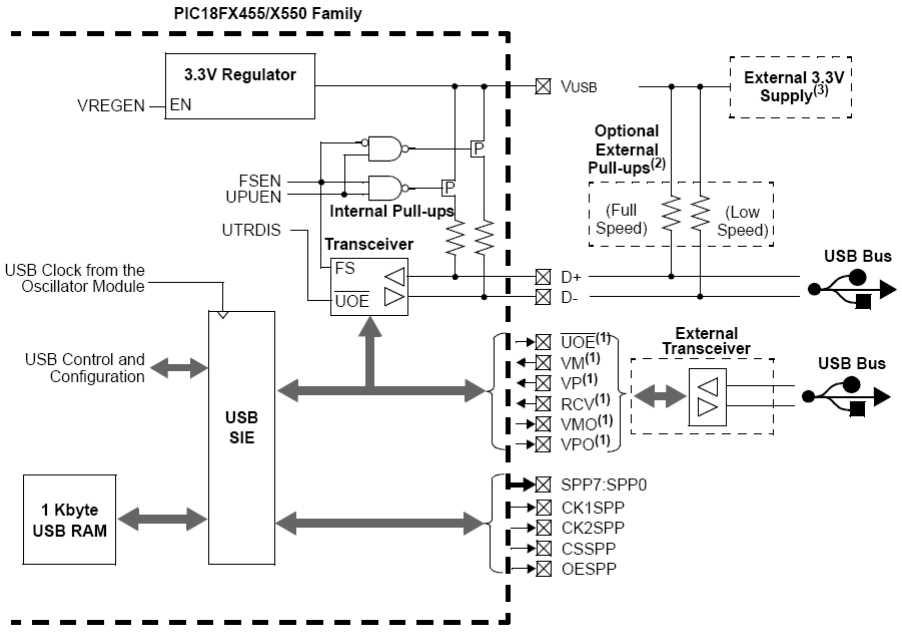


Figura 2.14. Conexión interna del módulo USB. Tomada de la hoja de especificaciones.

El microcontrolador cuenta con una memoria reservada para el uso de *USB* (*USBRAM*), mostrada en la figura 2.15. Esta memoria esta personalizada para que el usuario pueda programar directamente sobre la *RAM* y se refleje en el *USB*.

El microcontrolador cuenta también con resistencias *pull-up* internas para indicar la velocidad del puerto.

El oscilador debe de ser configurado de una manera especial para poder proveer la frecuencia que exige el estándar de *USB*.

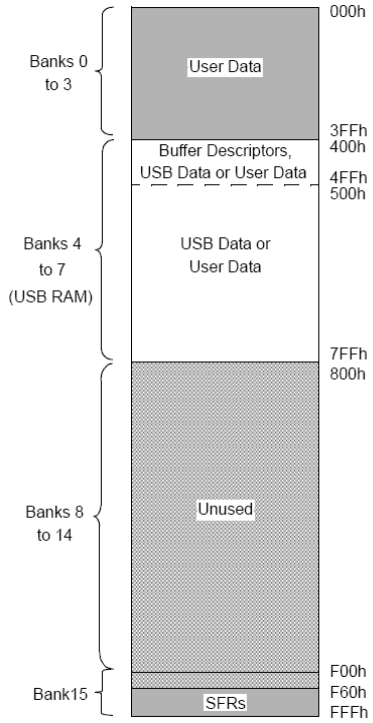


Figura 2.15. Diagrama de la memoria *RAM* de *USB*. Tomada de la hoja de especificaciones.

2.4.4 Convertidor A/D.

Un convertidor analógico digital (*ADC*), como su nombre lo indica, es un periférico que nos permite convertir una señal analógica a un valor binario entendible por la máquina cuya función de transferencia se muestra en la figura 2.16. Este proceso nos permite interpretar señales del mundo real.

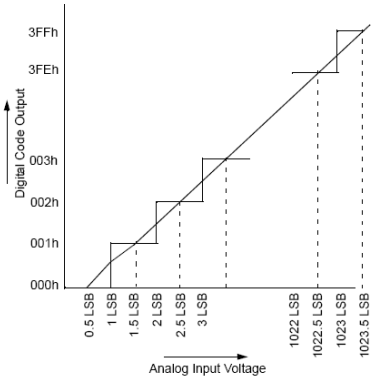


Figura 2.16. Función de transferencia del *ADC*. Tomada de la hoja de especificaciones.

Este microcontrolador cuenta con un *ADC* de 10 bits de resolución, lo que significa que una señal analógica podemos interpretarla como un rango de valores entre 0 y 1023 donde este último equivale al 100%. Sin embargo, pese a que el microcontrolador solo cuenta con un convertidor, se puede disponer de 13 canales distintos mediante un multiplexor, figura 2.17. La selección del canal de conversión se define mediante software, modificando los bits apropiados de los registros de control del *ADC*.

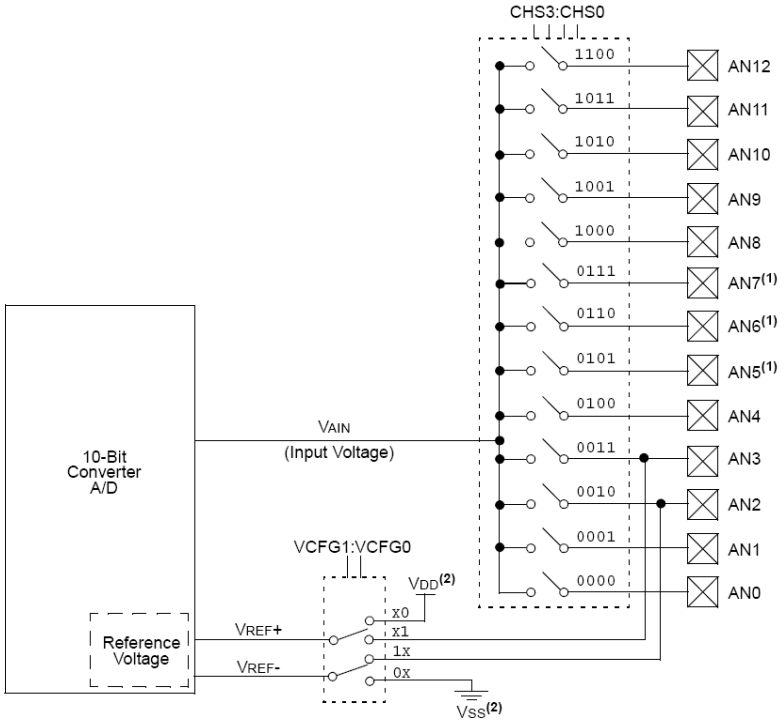


Figura 2.17. Conexión interna del módulo *ADC*. Tomada de la hoja de especificaciones.

Adicionalmente, se definen por software cuántos de estos canales van a ser implementados.

El proceso de conversión requiere de un cierto tiempo, función de la frecuencia del oscilador del PIC, el cual da una frecuencia máxima de muestreo. Sin embargo cabe aclarar que esta frecuencia debe de ser dividida por el número de canales que estén utilizando el convertidor.

Este *ADC* puede utilizar como voltaje de referencia tanto el voltaje de alimentación del microcontrolador, como un voltaje de referencia externo. Cualquier voltaje de referencia conectado al PIC debe de estar en el rango de alimentación del PIC y, adicionalmente, por ningún motivo el voltaje a convertir deberá de sobrepasar el voltaje de referencia.

2.4.5 I^2C.

El *I^2C* es un protocolo de comunicación muy usado para comunicar periféricos a microcontroladores. Esto da una clara idea del por qué es de interés esta función del PIC. El *I^2C* se verá a detalle en la siguiente sección, limitándose a ver las generalidades de como es utilizada por el microcontrolador.

El puerto *MSSP* (figura 2.18) puede funcionar como protocolo *I^2C*, ya sea como maestro o como esclavo. Este modo de operación permite conectar directamente las líneas *SDA* y *SCL* a las terminales del PIC.

El PIC cuenta con un registro de desplazamiento denominado SSPSR por el cual transita toda la información del *bus*. Este registro no es accesible directamente, en su lugar, si se desea leer o escribir al *bus*, la información debe de ser ingresada en otro registro (SSPBUF) que está directamente comunicado. Esta segmentación busca garantizar que la información se ponga en el *bus* sólo en tiempos correctos.

Adicionalmente se cuenta con un registro para funciones especiales que, en caso de utilizar al dispositivo como maestro permite dar la velocidad del reloj, o en caso de utilizarlo como esclavo permite almacenar la dirección.

El microcontrolador trae integrados circuitos CMOS que permiten generar señales especiales utilizadas por el protocolo.

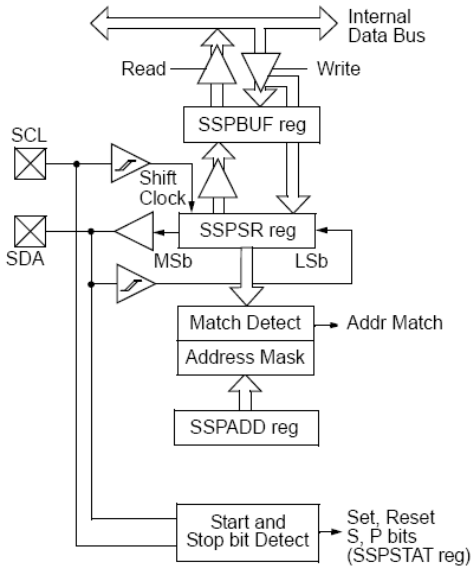


Figura 2.18. Conexión interna del módulo *MSSP* para la I^2C. Tomada de hoja de especificaciones.

2.5 El estándar de comunicaciones I^2C.

El I^2C es un bus de comunicación del tipo *master-slave* (maestro-esclavo) [5] desarrollado por la empresa *Philips Semiconductors*. Este *bus* nos permite controlar diferentes dispositivos conectados con un mismo cable mediante direccionamiento de señales.

Aunque inicialmente la I^2C tenía definida una velocidad de transmisión de 100kb por segundo, en la actualidad el estándar nos permite una velocidad de hasta 400kb por segundo en modo normal de operación, y hasta 3.4 Mb por segundo en su modo de alta velocidad[6]. A pesar de que los estándares de velocidad están bien definidos en las especificaciones del *bus*, estos son sólo valores máximos, la tasa de transferencia no necesita ser garantizada para un correcto funcionamiento.

---

[5] Una comunicación maestro-esclavo implica que un dispositivo envía órdenes por un bus a otro dispositivo.

[6] Para poder utilizar la alta velocidad se debe de enviar un comando especial que indique este modo de operación mientras se está en velocidad normal.

Este *bus* se conecta mediante dos cables, uno de transmisión de datos (*SDA*), y uno de señal de reloj (*SCL*), ambos de colector abierto requiriendo así ser conectados mediante resistencias *pull-up* (figura 2.19). La transmisión de datos es bidireccional, siendo el maestro aquél que se encarga de coordinar las transacciones para evitar colisión de datos. La señal de reloj es generada y determinada por el maestro; sin embargo hay ocasiones en las que el esclavo puede ser incapaz de trabajar a la velocidad dictada, en cuyo caso retiene la señal de reloj. Es por esto que el maestro debe de ser capaz de verificar el estado de la línea.

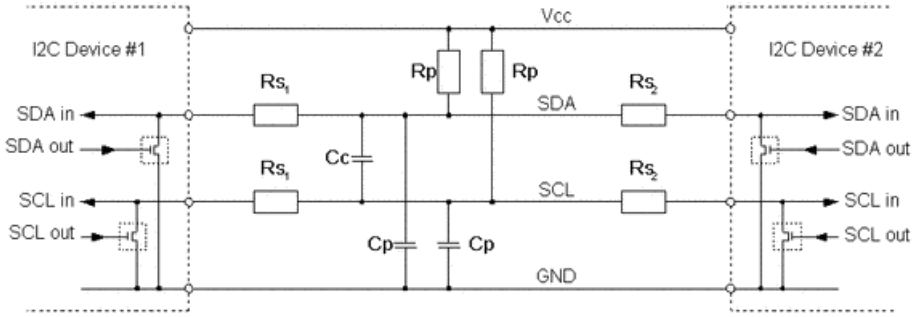


Figura 2.19. Construcción de la I^2C. Tomada de la página de *Philips*.

Uno de los principales beneficios de este bus de datos, es la capacidad de ser *multi-master* (tiene más de un maestro). Esta característica permite tener varios elementos maestros capaces de interactuar, y en caso de presentarse una situación de dos maestros solicitando el *bus* al mismo tiempo, los elementos tienen un orden para actuar y evitar escenarios impredecibles. A su vez, permite que los elementos maestros puedan ser esclavos.

Finalmente, como se había comentado, este *bus* funciona mediante direccionamiento. El direccionamiento permite indicar a cuál de los dispositivos conectados va la señal. Existen dos tipos de direccionamiento, uno con direcciones de 7 bits y otro con direcciones de 10 bits. Típicamente se utiliza el sistema de direccionamiento de 7 bits, el otro fue implementado debido a la demanda de conexión, ya que, a pesar de que 7 bits proporcionan un límite teórico de 128 direcciones, el actual número de direcciones permitidas es de 112, las direcciones faltantes son utilizadas para comandos especiales.

La comunicación por I^2C presenta la siguiente forma, figura 2.20, donde los dispositivos pueden ser maestros, esclavos o ambos, presentando al menos un maestro.

Figura 2.20. Conexión de dispositivos en el I^2C.

La comunicación entre los dispositivos conectados al I^2C es iniciada y terminada por el dispositivo maestro. La comunicación inicia cuando el maestro envía una señal de inicio (S), y es detenida por una señal de paro (P). Ambas señales se identifican por un cambio de nivel lógico de la línea *SDA* mientras la línea *SCL* tiene un nivel lógico "1". La diferencia entre ambas es que la señal de inicio consiste en una transición de un "1" a un "0", mientras que la señal de paro es la transición del "0" al "1".

Una vez generada la condición S, el dispositivo maestro envía por la línea la dirección del dispositivo con el que debe de comunicarse. Utilizando un direccionamiento de 7 bits, se envía un byte con los bits más significativos como la dirección del dispositivo y el último bit indicando si la operación va a ser de escritura ("0") o de lectura ("1"). Hecha la indicación, comienza la comunicación entre estos dispositivos según esté programada. Cada dato enviado, en el sentido que sea, espera la respuesta de reconocimiento. Finalizando son la condición de P.

2.6 Circuitos impresos.

Cuando hablamos de circuitos eléctricos, sobre todo de circuitos que han de perdurar, es usual que se requiera la producción de circuitos impresos. Los circuitos impresos o *PCB* (del inglés *Printed-Circuit Board*) son un medio para sostener mecánicamente los componentes y a su vez hacer las conexiones eléctricas necesarias para el correcto funcionamiento del diseño.

La producción de las *PCB*'s consiste en el grabado de pistas en un material conductor. Para esto se parte de placas consistentes en hojas de cobre laminadas sobre un material no conductor, baquelita o fibra de vidrio, y posteriormente se delimitan las pistas

mediante algún método, ya sea químico o mecánico. En este caso se trata una de las formas de ataque químico.

Para la producción de una tarjeta es necesario contar con el diseño de la misma. Este puede ser realizado mediante software especializado o bien diseñado a mano, considerando la distribución de los componentes.

Figura 2.21. Ejemplo de circuito impreso.

Se parte del esquemático del circuito reproduciendo las conexiones de nodo mostradas en el mismo. En caso de diseños sencillos la ubicación de los componentes puede no importar, sin embargo, al incrementar el número de éstos, pueden imposibilitarse las conexiones si los componentes se encuentran en cierta posición, en cuyo caso convendría utilizar la paquetería especializada, figura 2.21.

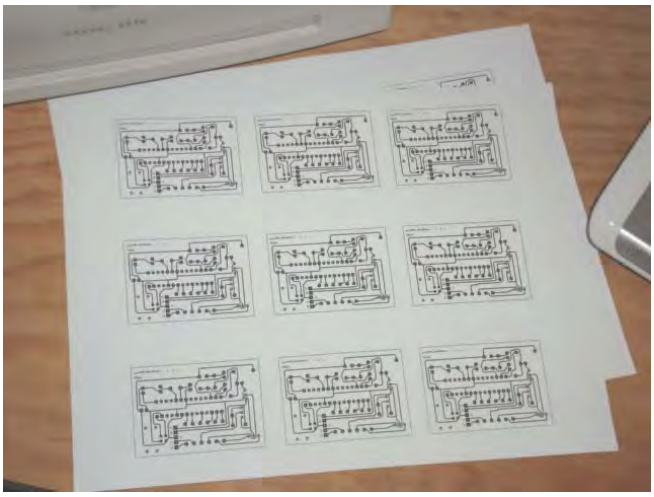


Figura 2.22. Papel Couche impreso por tóner.

Partiendo del diseño digital, se imprime sobre papel couche con una impresora láser, figura 2.22. La impresión preferentemente deberá de realizarse utilizando la mejor calidad de impresión y quitando el ahorro de tóner. La selección del papel se hizo mediante prueba y error, siendo éste el que ofrece el mejor resultado.

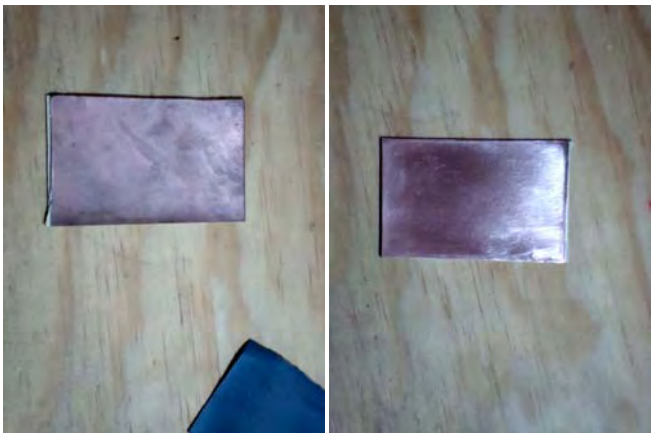


Figura 2.23. Tarjeta antes (izquierda) y después (derecha) de la limpieza.

A continuación preparamos la tarjeta. Para esto con una lija fina limpiamos la superficie hasta eliminar toda la suciedad y las impurezas que puedan afectar nuestro proceso. Una comparación entre una tarjeta sucia y una limpia se muestra en la figura 2.23. También se puede recortar la tarjeta a un tamaño adecuado para aprovechar al máximo el material.

Figura 2.24. Procedimiento del planchado.

Cuidadosamente se coloca la hoja de papel, con la cara de tóner contra la lámina de cobre, asegurando de que todo el diseño quede dentro del área de la tarjeta. De ser necesario, se puede sujetar la hoja a la tarjeta mediante cinta adhesiva para evitar movimientos accidentales. Una vez colocada la hoja sobre la tarjeta, se procede a planchar la hoja contra la tarjeta hasta que el tóner pase de la primera a la segunda. Para esto, es conveniente observar el estado de la hoja, y se sabrá que esto ha ocurrido si en toda la superficie de la hoja se muestran indicios de quemado uniforme.

Inmediatamente después de retirar la plancha, conviene sumergir la tarjeta en agua para facilitar el retiro de la hoja. Se busca retirar completamente la hoja de papel, si queda un poco de papel se refleja en un mal revelado del circuito. Hay que asegurarse que al término de este proceso todas las pistas presenten un estado deseado, y en caso de no ser así, se pueden reparar las mismas con un plumón indeleble. Fotos del proceso son mostradas en la figura 2.24.

Teniendo la placa impresa y en condiciones favorables, se procede a revelar la tarjeta mediante un ataque químico, figura 2.25. En este caso, se hace uso de cloruro férrico ($FeCl_3$), cuyo efecto es remover todo el cobre que no haya sido cubierto por el tóner y/o el plumón. Se sumerge la placa en su totalidad en el ácido y se evalúa su evolución periódicamente, retirándola cuando todo el cobre sea removido.

Figura 2.25. Revelado del circuito.

Se procede a barrenar la tarjeta con brocas del diámetro adecuado para que quepan los componentes, figura 2.26. También se limpia el tóner, ya sea lijando la tarjeta o bien haciendo uso de acetona. Esto permite soldar los componentes a la tarjeta dejándola

funcional. Adicionalmente se pueden aplicar diversas capas, como la mascarilla antisoldante para mejorar el desempeño o la mascarilla de componentes para facilitar el entendimiento de la tarjeta.

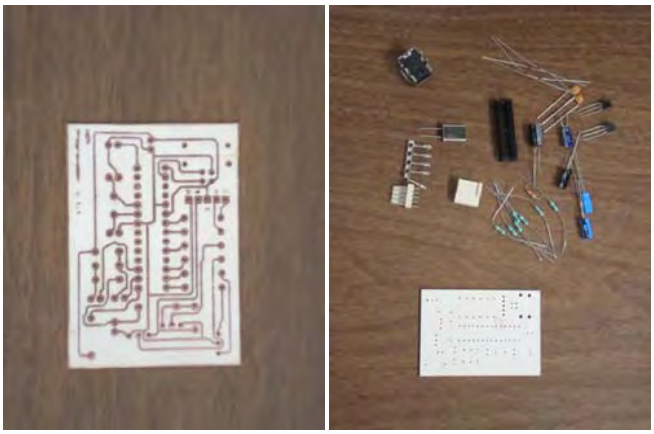


Figura 2.26. Circuito listo para soldarse.

2.7 Desarrollo de una tarjeta de programación para PIC.

Era de esperarse que al trabajar con microcontroladores se contara con algún medio para poder cargarles el programa generado. Además, generalmente los programas no quedan a la primera por lo que debemos de entrar en un ciclo iterativo de programar y depurar, que termina sólo con el comportamiento satisfactorio del sistema implementado.

Con todo esto, se puede ver la importancia de tener un programador al alcance de las manos.

Debido a la creciente popularidad de los microcontroladores PIC, existe una infinidad de programadores disponibles, algunos a la venta y otros tantos libres de producción. Muchos modelos libres de producción pueden ser encontrados en Internet, permitiéndonos obtener diseños a distintas escalas, algunos sólo presentan esquemáticos, mientras que otros proveen la lista de materiales y el diseño de los *PCB*'s que requieren.

El diseño a tratar en ésta sección está basado en el "GTP-USB Lite" y tiene las siguientes características (figura 2.27):

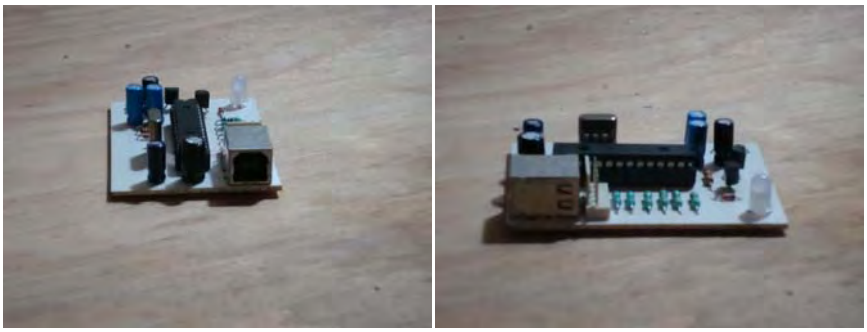


Figura 2.27. Vistas del programador.

- Interfaz de comunicación *USB.*
- Alimentado por *USB.*
- *Firmware* gratuito.
- Bajo costo de construcción.
- Conexión *ICSP* que permite la programación directamente en el circuito destino.
- Tamaño compacto (65 x 45 mm).
- Mejoras ante el GTP-USB Lite respecto al voltaje de operación.
- Capaz de programar una gran variedad de microcontroladores.
- Capaz de programar una gran variedad de memorias seriales.

Para consulta del *PCB* y de la lista de materiales del diseño propuesto véase el apéndice A.

El software para su utilización en una PC es el WinPic800 V3.55 (figura 2.28) y puede ser descargado junto con el firmware del microcontrolador necesario gratuitamente en Internet.

Figura 2.28. Ventana del programa.

3 Desarrollo.

El proceso de desarrollo integral exige la inclusión de diversos temas y a distintos niveles. Antes que nada, se ha dicho que se va a utilizar el microcontrolador PIC18F4550, desconociendo el porqué de la elección. Además, entre la información necesaria para la creación del dispositivo se encuentra el cómo programar el dispositivo para atender al protocolo USB. Del mismo modo debe de existir su contraparte en la computadora, este pequeño código es el controlador. Es muy sabido que una buena manera de probar la funcionalidad de un elemento es probar sus partes y, al resultar estas pruebas positivas, probarlo en su totalidad; de este modo conviene realizar un prototipo no tan ambicioso y de ahí evolucionar hasta llegar al dispositivo planeado. Para lograr lo anterior, se requiere pasar por distintos niveles de desarrollo, el primero de estos, el del hardware. Partiendo de lo anterior, a sabiendas de cómo está estructurado nuestro dispositivo, se procede a la programación del mismo. Así, una vez establecido el comportamiento del dispositivo, se puede programar lo necesario para operar correctamente desde la computadora. Es conveniente, antes de realizar pruebas a un prototipo energizado, realizar las simulaciones del mismo a fin de evitar mayores problemas.

El desarrollo de esta tesis se realiza con el microcontrolador mencionado anteriormente, sin mencionar mas nada sobre esta selección. Sin embargo, la elección del mismo es un proceso iterativo basado en el análisis de las características propias de los diferentes microcontroladores. En este capítulo desglosaremos la selección del microcontrolador yendo de lo general a lo particular, mediante la comparación con sus similares.

La programación del microcontrolador, en particular la parte concerniente al protocolo USB, se va a realizar con herramientas provistas por Microchip. La funcionalidad del dispositivo final se puede analizar como un todo conformado por distintas subfunciones. Cada una de estas desarrolladas bajo distintas herramientas. Hablando en específico de la funcionalidad USB, se hará uso del *framework "MCHPFSUSB"*.

En adición al control del protocolo USB llevado a cabo por el microcontrolador, es necesario el controlador del dispositivo, que sirve como intérprete para la computadora.

Este código almacena diversa información, entre ella nombre e identificador del fabricante y permite relacionarlo con el identificador incluido en el *firmware*. Además permite identificar el tipo de dispositivo del que se trata al momento de conectarlo al puerto.

Hablando a nivel de la programación específica del USB, el primer paso es la comunicación entre la computadora y la tarjeta. Es conveniente realizar esta prueba antes de avanzar en la programación de cualquiera de las partes, debido a la necesidad de esta para el correcto funcionamiento de la solución planteada. De este modo, se obtiene un primer acercamiento a la tarjeta que consta únicamente de un dispositivo USB cuya única función programada es lograr la enumeración. Para poder probar el software hacemos uso de simulaciones mediante la utilería *ISIS* del programa *Proteus v7.2 SP2*.

Con lo anterior procedemos al desarrollo propio de la tarjeta, comenzando por la parte del *hardware*. Para esto hay que ubicar cada una de las funciones en un pin del microcontrolador que pueda realizarla. Se buscan también los periféricos necesarios para la operación del mismo.

Una vez distribuidas las funciones se puede desarrollar el *firmware* direccionando los tipos de señal a su canal correspondiente. Implementando las funciones necesarias para mejorar obtener un mejor desempeño.

El desarrollo del *software* no es conveniente sino hasta que se ha definido en el *firmware* la estructura de la información a manejar, que una vez implementada no ha de cambiar. Este programa permite interactuar con la tarjeta mediante una librería en *Visual Basic .NET*.

3.1 Seleccionando el microcontrolador.

Antes que nada el microcontrolador seleccionado es un PIC, pero ¿por qué seleccionar un PIC en lugar de otra clase de microcontroladores?

Los microcontroladores PIC son los que presentan la mayor cantidad de herramientas y literatura, sobre todo en Internet, para su desarrollo. Esto es fácilmente comprobable con búsquedas en Internet. La tabla 3.1 muestra una relación de resultados con el respectivo tipo de microcontrolador; se busca que la prueba sea en igualdad de condiciones, así que utilizamos como búsqueda "microcontroladores X", donde X es el

microcontrolador que buscamos[1]. Además se puede observar una columna que muestra la relación de productos por fabricantes de la comercializadora de dispositivos electrónicos *AG Electrónica*.

Fabricante	Busqueda	Resultados	
		Google	AG Electrónica
National	"*microcontroladores cop8*"	97	43
Atmel	"*microcontroladores avr*"	3490	-
Philips	"*microcontroladores philips*"	107	-
Microchip	"*microcontroladores pic*"	93400	105
Motorola	"*microcontroladores motorola*"	8440	35
Dallas	"*microcontroladores dallas*"	112	1
Texas	"*microcontroladores texas*"	223	3
Intel	"*microcontroladores intel*"	1040	23

Tabla 3.1. Consulta en Internet de los diversos microcontroladores.

Como puede observarse de los resultados obtenidos, los microcontroladores PIC muestran una gran superioridad en recursos de información y, adicionalmente, son los más fáciles de obtener al menos en nuestro país.

Con esta notable ventaja es claro el porqué la selección de los PIC's, pero ¿cual PIC seleccionar?

En la actualidad existen once PIC's distintos con capacidad *USB*, con tamaño de palabra de 8 bits y empaquetado *DIP* (del tipo que se puede poner en una *protoboard*). Estos se diferencian principalmente por el número de pines y por el tamaño de las memorias de datos y de programa. Los once PIC's están agrupados en 4 subfamilias:

- PIC18F14K50
- PIC18F2450/4450
- PIC18F2455/2550/4455/4550
- PIC18F2458/2553/4458/4553

[1] Fecha de consulta 25 de diciembre del 2008.

Recordemos que como objetivo buscamos un total de 32 canales de entradas y salidas, 16 digitales y 16 analógicos, operando entre 0 y 5 volts. Si deseamos aprovechar el *ADC* integrado del microcontrolador y la I^2C necesitamos un mínimo de 26 pines de entrada-salida (8 para las DI, 8 para las DO, 8 para las AI y 2 para la comunicación I^2C), más aquellos necesarios para el funcionamiento del *USB* y del PIC (alimentación, canales de comunicación USB, puertos de conexión del oscilador, etcétera).

El microcontrolador del primer grupo citado tiene únicamente 20 pines, por lo tanto no sirve, los otros grupos poseen miembros tanto de 28 pines, como de 40, donde los 28 pines no cubren la necesidad, dejando los de 40 como los de utilidad. En el orden listado, la segunda familia carece del módulo *MSSP* lo que permite acoplar circuitos integrados a través del protocolo I^2C o *SPI*.

Las dos familias restantes solo difieren en la resolución del *ADC*, la tercera es de 10 bits y la cuarta de 12 bits. Debido a que la tarjeta es para fines didácticos podemos prescindir de la mayor resolución de la cuarta subfamilia, resultando en un menor costo del microcontrolador y por lo tanto de la tarjeta.

Ahora bien, al tener elegida una subfamilia falta distinguir entre sus miembros. Los PIC18F2455/2550 son de 28 pines, mientras que los PIC18F4455/4550 son de 40, pudiendo escoger tanto el 4450 como el 4550. La única diferencia entre estos dos PIC es el tamaño de memoria de programa, por lo tanto seleccionamos el último que tiene mayor memoria, por si se busca mejorar el *firmware*.

3.2 Programación del protocolo USB para el PIC.

Para programar el microcontrolador se hace uso de las herramientas que provee *Microchip*: el *MPLAB IDE v8.10*, el *C18 v3.20* y el *MCHPFSUSB v1.3*.

El *MPLAB* es un entorno de desarrollo integrado para *Windows* utilizado para desarrollar aplicaciones para los microcontroladores de *Microchip*. Este entorno es gratuito, fácil de usar e incluye herramientas para depurar los programas creados.

Adicionalmente, *Microchip* ofrece herramientas que facilitan el desarrollo de aplicaciones mencionadas anteriormente, el *C18* y el *MCHPFSUSB*.

El *C18* es un compilador "*C*" para la serie de microcontroladores PIC18, existen otros compiladores para las demás series. Este compilador, aparte de permitirnos desarrollar nuestras funciones bajo el lenguaje de programación "*C*", cuenta con funciones especializadas para el manejo de los módulos con los que cuentan estos microcontroladores.

El *MCHPFSUSB* es un *framework* (estructura definida para el soporte de la programación *USB*), que nos facilita la programación del protocolo que debe de seguir el microcontrolador para interactuar con el *host*. Así mismo, nos provee de un controlador y una *dll* para el desarrollo de aplicaciones para el *host*.

Dicho *framework* incluye unos ejemplos de aplicación. De estos destacan tres programas, cada uno desarrollado bajo distintas clases: *HID*, *CDC* y uno genérico. Estos incluyen distintas funciones y, por lo mismo, distintas librerías para cumplir con la comunicación. Para fines de desarrollo se toma como referencia la aplicación de USB genérica.

El uso del framework para la presente tesis se limita a facilitar el desarrollo de la misma. La herramienta es capaz de enumerar el dispositivo y llevar a cabo las respuestas necesarias para la tolerancia a fallas de la comunicación implementada en el protocolo USB; adicionalmente cuenta con distintas funciones para la transmisión y recepción de datos.

3.3 Edición del controlador y sus efectos

Un controlador es un elemento de programación que ayuda al sistema operativo a comunicarse con dispositivos periféricos. En el caso particular de los controladores para los dispositivos USB, este contiene información que lo relaciona con un *VID* y un *PID* específicos, esto con la finalidad de asegurar una correcta comunicación con el dispositivo.

Al instalarse por primera vez un controlador este es almacenado por el sistema operativo en sus carpetas, haciendo innecesario instalarlo cada vez que sea conectado el dispositivo. Como se acaba de mencionar, para los dispositivos USB, el controlador incluye la información del *VID* y el *PID*; esta información es almacenada por el sistema operativo y al conectarse un dispositivo USB busca en su base de datos el *VID* y el *PID* del dispositivo

para ver si lo tiene instalado, y en caso de ser así solo busca los archivos previamente guardados.

El controlador incluye información adicional sobre el dispositivo, esta es: la clase a la que pertenece, la versión del controlador (incluyendo la fecha), el nombre y datos del creador, y el nombre y tipo del dispositivo. Esta información se utiliza para la administración de los controladores y adicionalmente es la que se muestra en el globo cuando el dispositivo es conectado por primera vez, figura 3.1.

Figura 3.1. Globo de notificación de dispositivos encontrados.

El controlador provisto por el *framework* consta de 3 archivos: el *mchpusb.inf*, el *mchpusb.sys* y el *wdmstub.sys*. Cada uno de estos con una función específica. El primero, funciona como un encabezado, en el que se incluye la información sobre el controlador, su autor y versión, y además da aviso de la existencia y necesidad de los demás archivos necesarios. Los archivos restantes tienen propiamente el código desarrollado por Microchip para que, por medio de la *dll*, se permita la comunicación con el dispositivo. De este modo, el único archivo que hay que editar para acoplarlo a la aplicación final es el *.inf*.

Los efectos de acondicionar el archivo a la aplicación se pueden ver en la siguiente imagen (figura 3.2).

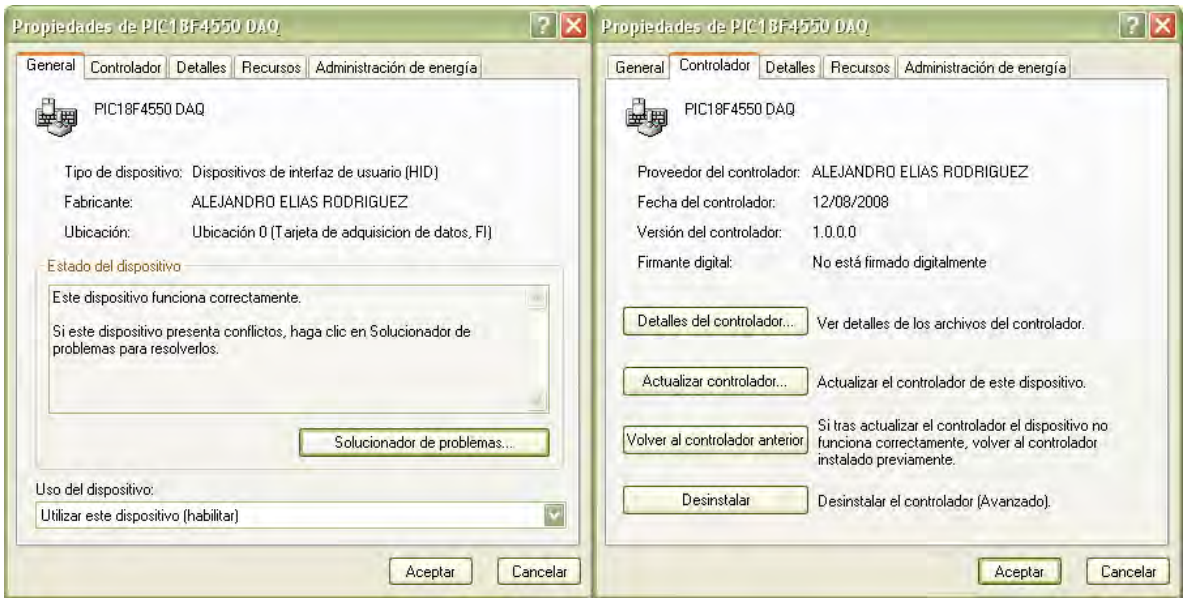


Figura 3.2. Propiedades de la tarjeta de adquisición de datos.

3.4 Probando la enumeración, circuito simulado con Proteus

ISIS es una utilería, parte de *Proteus*, que permite simular circuitos. Este programa ofrece además la capacidad de simular un dispositivo conectado mediante un puerto *USB* virtual. De este modo todo el desarrollo a continuación es probado paso a paso evitando los percances del prototipado físico.

Haciendo uso de esta utilería somos ahora capaces de probar paso a paso el desarrollo del dispositivo mientras este es programado y depurado. Por si esto fuera poco, *MPLAB* es capaz de utilizar a *ISIS* como su depurador, y así podemos ver reflejados los efectos de los cambios hechos y hacer barridos paso a paso del código en tiempo real.

El primer paso para el desarrollo de un dispositivo es lograr la enumeración del mismo y con esto una prueba de comunicación sencilla.

Como se menciono en la sección 3.2 el *framework* nos facilita este trabajo. Partiendo del ejemplo de comunicación genérica llevamos a nuestra aplicación al punto de comunicación. Para esto es necesario inicializar la configuración del microcontrolador, llamar a las funciones requeridas para iniciar la enumeración y por último esperar la exitosa enumeración del dispositivo. Al completarse la enumeración el sistema operativo para el

que está siendo desarrollada la presente tesis, *Windows XP*, emite un sonido característico y, en caso de que el controlador no haya sido instalado, muestra un globo de notificaciones con los datos del dispositivo conectado.

Si el controlador del dispositivo ha sido instalado el dispositivo continúa con la comunicación de manera normal. En caso contrario el sistema operativo suspende momentáneamente la comunicación para pedir la instalación del mismo.

Figura 3.3. Globo de notificación desplegado al conectar por primera vez la tarjeta de adquisición de datos.

Esta instalación se realiza de la siguiente manera:

Lo primero por hacer es conectar la tarjeta de adquisición de datos en el puerto USB. De esta manera, el sistema operativo detecta el nuevo hardware conectado, tal como lo muestra la figura 3.3.

El sistema operativo ha de buscar el dispositivo en su base de datos y, al no encontrarlo, despliega el asistente de instalación de controladores. En la primer ventana (figura 3.4) se selecciona "No por el momento" y se da clic en siguiente.

Figura 3.4. Ventana del asistente de instalación para nuevos dispositivos.

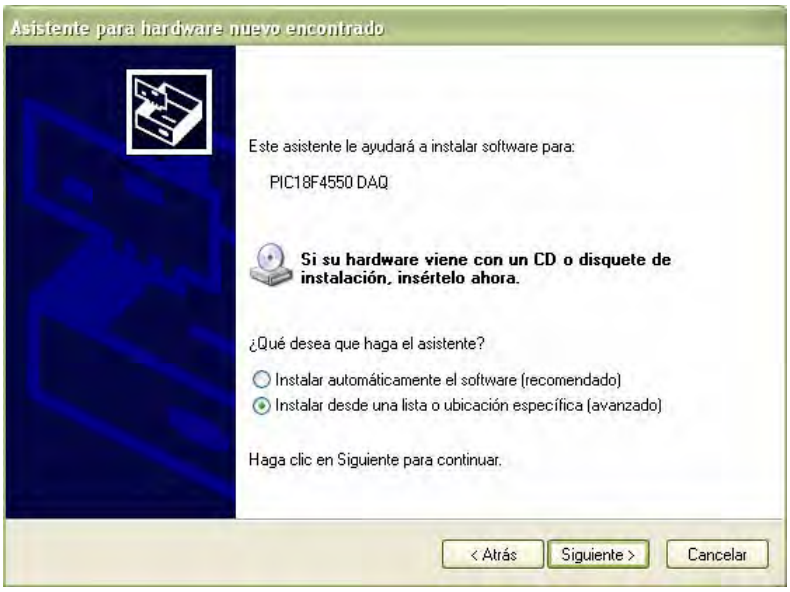


Figura 3.5. Ventana que permite seleccionar la manera de instalar el controlador.

La siguiente ventana, figura 3.5, ofrece buscar el software del dispositivo o instalarlo manualmente. Seleccionando "Instalar desde una lista o ubicación específica (avanzado)" se hace clic en siguiente.

Con las opciones seleccionadas, se requiere la búsqueda personalizada del controlador como se muestra en la figura 3.6. Incluyéndola desde una ubicación específica, se busca el directorio en el cual se tienen guardados los archivos del controlador.

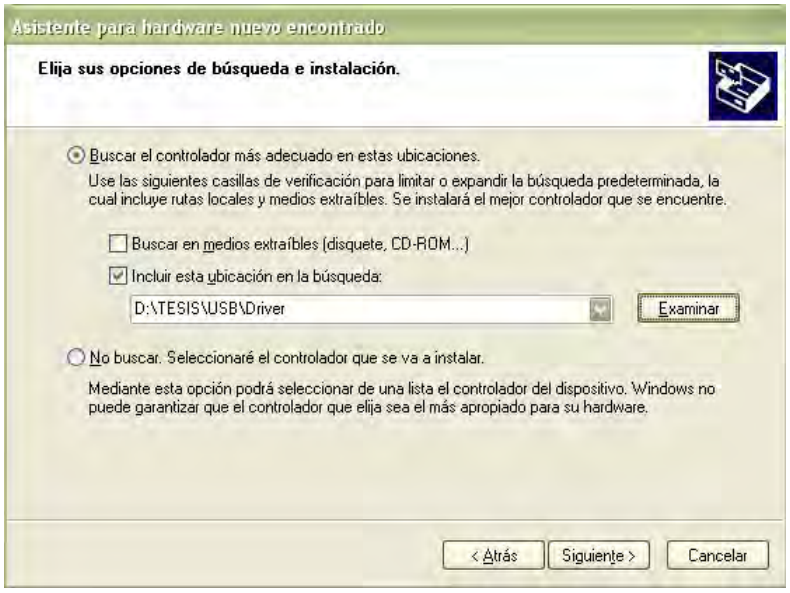


Figura 3.6. Ventana que permite seleccionar la ruta del archivo “.inf”.

El controlador a instalar no posee una firma digital registrada con Microsoft, por lo que el sistema operativo muestra la siguiente alerta (figura 3.7). Simplemente se da clic en continuar y el sistema operativo reanudará la instalación.

Figura 3.7. Ventana de notificación de la prueba del controlador.

A continuación se muestra una ventana (figura 3.8) que pide la ruta del resto de los archivos del controlador. Simplemente se busca la ruta en la que los archivos del controlador están guardados y se da clic en aceptar, permitiendo finalizar con el proceso de instalación.

Figura 3.8. Ventana que pide la ruta de los archivos adicionales del controlador.

Ahora bien, para poder simular el dispositivo USB son necesarias dos cosas, que pueden ser desarrolladas a la par:

- El programa que ha de ser cargado en el microcontrolador.
- El esquemático en *ISIS* necesario para mostrar el funcionamiento adecuado del circuito.

El programa a cargar en el microcontrolador no es otra cosa que el resultado de compilar la aplicación desarrollada en *MPLAB*. Este programa presenta en formato hexadecimal el total de la aplicación traducida a lenguaje ensamblador. Para poder generar este archivo, el compilador de *MPLAB* requiere que la aplicación no presente errores. Este archivo tiene el nombre de la aplicación con la extensión *.hex*.

Por otro lado, crear el esquemático completo presenta un poco más de dificultad. Al momento de simular un circuito eléctrico en *ISIS*, se requiere que los dispositivos involucrados, al menos los elementales, tengan programada la manera de comportarse. La mayoría de los dispositivos incluidos por *default* ya tienen este código, sin embargo, en el caso de los microcontroladores es necesario relacionarlo con su programa hexadecimal y su configuración. Esto se hace a través de la ventana de propiedades del dispositivo (figura 3.9).

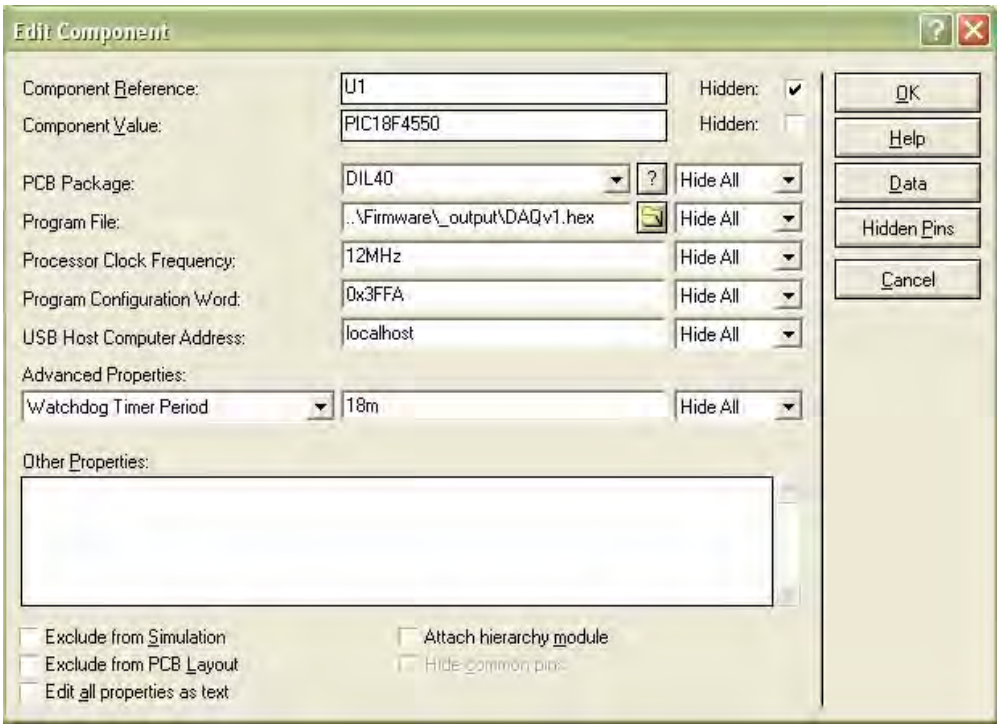


Figura 3.9. Ventana de propiedades del microcontrolador.

Para probar la enumeración solo es necesario incluir, en el esquemático, el microcontrolador, ya que la interacción de los pines D+ y D- de este con los del puerto virtual depende directamente del código del microcontrolador. Tampoco es necesario incluir en el esquemático el modelo del reloj de cuarzo, este ni siquiera presenta código

para simulación y la frecuencia de oscilación de los pines asignados a este fin es simulada internamente por *ISIS* mediante la configuración del parámetro en la ventana de propiedades del microcontrolador. Debido a esto no tiene sentido mostrar el esquemático para probar la enumeración. Solo cabe mencionar que en la ventana de propiedades del PIC es necesario asignar la ruta del archivo hexadecimal.

Teniendo el programa del microcontrolador y el esquemático, es posible probar el circuito. Para probarlo, basta con iniciar el simulador pulsando el botón *Play* de la ventana de *ISIS* que contiene el esquemático.

ISIS cuenta con un analizador de la comunicación USB, donde podemos observar el proceso de enumeración satisfactorio, figura 3.10.

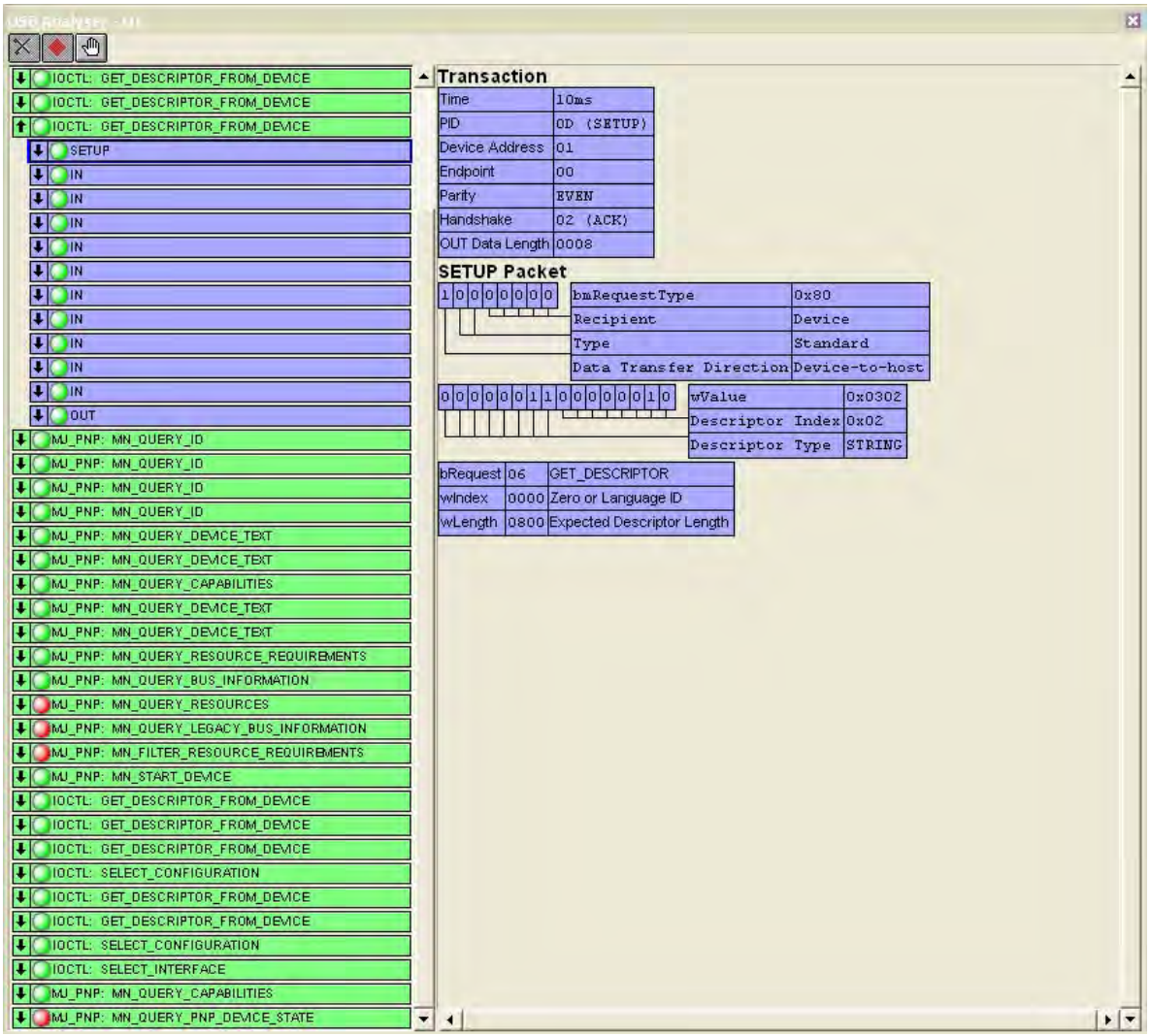


Figura 3.10. Ventana del analizador USB para la simulación de la tarjeta de adquisición de datos.

3.5 Desarrollo del hardware

Para el desarrollo del hardware lo primero que se debe hacer es la distribución de las funciones acorde a la capacidad del PIC.

Antes que nada, verificando en la hoja de especificaciones, se selecciona cada una de las funciones. Comenzando con las funciones únicas, el módulo *MSSP* para la comunicación I^2C se encuentra alambrado a los pines 33 y 34, para los canales *SDA* y *SCL* respectivamente. Los canales de entrada analógicos se encuentran ubicados de los pines 2 al 5, del 7 al 10 y del 33 al 37; sin embargo, como se buscan solo 8 canales de este tipo, nos basta con los primeros 8 (pines 2, 3, 4, 5, 7, 8, 9 y 10). Los pines 23 (D-) y 24 (D+) son requeridos para la comunicación USB.

Con todo lo anterior, solo nos queda un puerto de E/S digital de 8 bits completo (el puerto D), y diversas E/S dispersas en los otros puertos. De este modo, asignamos las entradas digitales al puerto D para facilitar la lectura; adicionalmente la configuración de las entradas de este tipo es Schmitt Trigger, mejorando el desempeño de la lectura.

Por último, nos queda asignar las salidas digitales. Estas, como se concluye del párrafo anterior, han de estar dispersas en los canales de los puertos restantes, principalmente del puerto C. Así pues, la asignación de las salidas queda de la siguiente manera: el canal 0 se conecta al pin 15, el canal 1 se conecta al pin 16, el canal 2 se conecta al pin 17, el canal 3 se conecta al pin 36, el canal 4 se conecta al pin 37, el canal 5 se conecta al pin 38, el canal 6 se conecta al pin 25 y, por último, el canal 7 se conecta al pin 26.

Para las salidas analógicas se escogió un circuito integrado de Microchip, el MCP4725. Este circuito presenta una resolución de 12 bits para la conversión Digital-Analógico. Este chip se presenta en un encapsulado del tipo SOT-23 de 6 pines, esto implica que el circuito es de montaje superficial y presenta un volumen reducido. Cuenta con 8 direcciones, justo las necesarias, posibles. La dirección del encapsulado depende de 2 bits configurados en fábrica y un bit de direccionamiento como pin del circuito.

Parte	Descripción
C1, C2	Capacitor cerámico 20 pF
C3	Capacitor electrolítico de 47 uF
D1	Led bicolor
R1,R2	Resistencia de 10kΩ
R3,R4	Resistencia de 330 Ω
U1	PIC18F4550
U2, U3	MCP4725-A0
U4, U5	MCP4725-A1
U6, U7	MCP4725-A2
U8, U9	MCP4725-A3
USB	Conector USB-B
XTAL	Crystal de cuarzo de 12 MHz

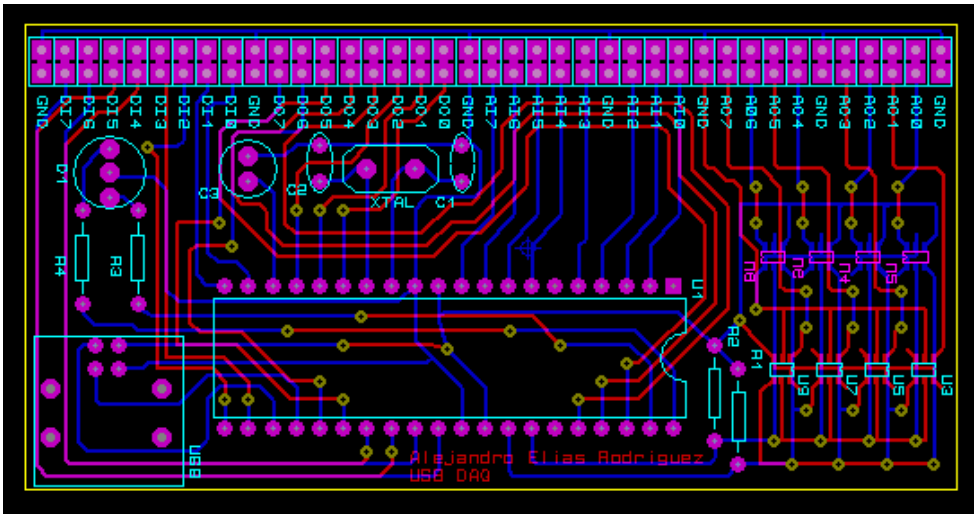
Tabla 3.2. Listado de componentes para la tarjeta.

Una vez colocadas las funciones, se procede a diseñar el circuito de la tarjeta de adquisición de datos.

Proteus es un programa que integra *ISIS*, tratado en la sección 3.3, y *ARES*, una utilería que nos permite el diseño de *PCB*'s. De este modo, con el uso de *ARES*, se procede al desarrollo del circuito impreso.

Antes que nada se coloca la totalidad de los componentes a usar (tabla 3.2.) de una manera que se crea conveniente, en un área propuesta de trabajo. Esta área corresponde al tamaño de la placa fenólica que se ha de usar por lo que se busca el menor tamaño posible.

Mediante un proceso iterativo se llega a la solución que se muestra en la figura 3.11. Esta solución tiene un tamaño de 5 x 10 cm. y requiere de una placa fenólica de doble cara.



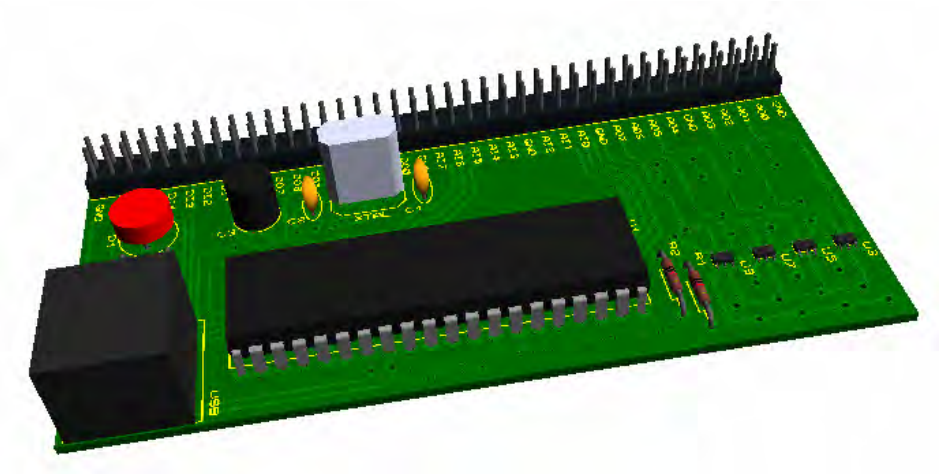


Figura 3.11. Vistas de la tarjeta.

Para la producción de esta tarjeta de forma manual es necesario realizar la conexión de las pistas entre caras. Esto se puede realizar con un trozo de metal que atraviese la tarjeta soldado por ambos lados. Adicionalmente se debe de realizar soldado de montaje superficial para los *DAC*'s. La impresión de las pistas para el ataque químico y la aplicación de las mascarillas de componentes se realizan mediante serigrafía.

3.6 Desarrollo del firmware

Ya asignada la distribución de los pines, de acuerdo a su funcionalidad. La distribución de los pines se muestra en la siguiente figura.

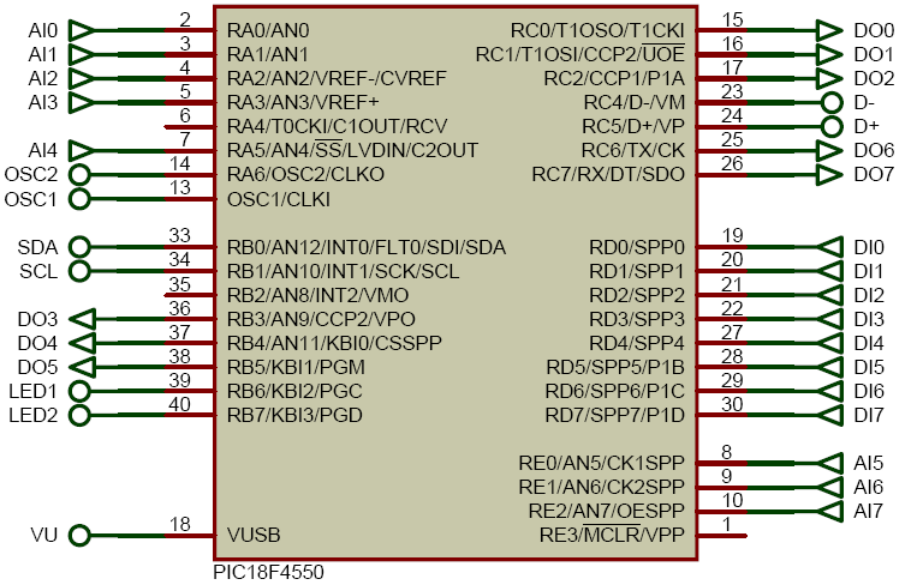


Figura 3.12. Distribución de las funciones.

3.6.1 Configuración de los canales de entrada analógicos.

Para configurar los canales analógicos de entrada lo primero por hacer es configurar los puertos como entradas. Según las especificaciones del PIC para hacer que un pin sea de entrada o de salida se pone el bit correspondiente a su registro “TRIS” en “1” o en “0” respectivamente. De este modo, se llevan a 1 los bits de los registros TRISA y TRISE que son los puertos asociados a la conversión analógico-digital según la distribución de la figura 3.12.

Existen tres registros asociados al control de la conversión (ADCON0, ADCON1 y ADCON2).

El registro ADCON0 es el encargado de la dinámica de la conversión. Este registro permite seleccionar cual de los canales multiplexados es el que ha de ser leído. Adicionalmente, tiene un bit que sirve como bandera para saber si la conversión ha terminado y otro que sirve para habilitar o deshabilitar la conversión. Como este registro es dinámico no se analiza su configuración.

El registro ADCON1 nos permite configurar el número de canales a utilizar y las referencias para la conversión. Es importante analizar el número de canales a utilizar debido a que la selección de canales como analógicos bloquea la función de estos para otra modalidad de funcionamiento; por esto seleccionamos únicamente los ocho canales que se buscan. En cuanto a las referencias nos permite elegir si los rangos de voltaje para la conversión son externos o internos; para los fines de esta tesis los internos cumplen (rango de valores entre 0 y 5 V).

El registro ADCON2 es el encargado del tiempo de muestreo y de cómo se han de guardar los valores. Es importante configurar de manera adecuada el tiempo de muestreo para garantizar una correcta conversión.

Existen otros dos registros asociados a la conversión, ADRESH y ADRESL, receptáculos del valor resultado. El orden de los bits en cada uno de los registros depende directamente de la configuración del bit más significativo del ADCON2.

3.6.2 Configuración de los canales digitales.

La configuración de los canales digitales no tiene mayor complicación. Los microcontroladores PIC tienen, por *default*, configurados los canales como entradas digitales. Esto nos indica que todos los puertos funcionan como canales digitales.

Como se mencionó en la sección anterior, para indicar el sentido del canal basta con modificar el valor del registro TRIS asociado a este. De este modo, el registro TRISD debe de tener sus valores en "1", y los respectivos bits de los registros TRISB y TRISC deben de ser llevados a "0".

3.6.3 Configuración de la I^2C para los canales analógicos de salida.

Antes que nada, los canales *SCL* y *SDA* deben de ser configurados como entradas, esto para poner las entradas como drenador abierto. Sin embargo estos canales han de ser dinámicos debido a que el dispositivo debe ser utilizado como maestro.

Existen 5 registros asociados directamente al uso del módulo I^2C del PIC. Estos son: SSPCON1, SSPCON2, SSPSTAT, SSPBUF y SSPADD.

El registro SSPSTAT sirve para conocer el estado del *bus*.

Los registros SSPCON1 y SSPCON2 son registros de control. Del primero, por el momento, solo son de interés los 3 bits menos significativos, los cuales nos permiten indicar el modo de operación del microcontrolador (esclavo o maestro). Los otros bits del registro y el registro SSPCON2 son dinámicos.

El registro SSPBUF corresponde al buffer de datos. Como se había mencionado en el capítulo anterior, este registro está directamente relacionado con el registro SSPSR que es el que se comunica con el exterior.

Por último, el registro SSPADD, funcionando el PIC como maestro, es un escalador del oscilador interno para dar la frecuencia de funcionamiento adecuada. El valor guardado en el registro presenta la siguiente relación con la frecuencia de comunicación:

$$f_{I^2C} = \frac{f_{osc}}{2(SSPADD + 1)}$$

3.6.4 Selección de los bits de configuración.

Para los bits de configuración buscamos las siguientes características:

- Tipo de oscilador: *HSPLL*.
- Divisor del *PLL* por 3 (Oscilador de 12 MHz).
- Fuente del oscilador del *USB*: Oscilador *PLL* entre dos.
- Regulador *USB* habilitado.
- *BOR* deshabilitado.
- *MCLR* deshabilitado.
- *WDT* deshabilitado.
- *LVP* deshabilitado.

Si suponemos que se usa un cristal de cuarzo de 12 MHz para generar los pulsos del oscilador, el tipo de oscilador debe de ser *HS*, adicionalmente, para poder generar la frecuencia de 48 MHz necesaria para el funcionamiento del *USB* a *full-speed* utilizamos el *PLL*. La configuración de los bits requiere saber precisamente la frecuencia del oscilador, en la hoja de especificaciones se pueden consultar los números preestablecidos para los correspondientes valores de frecuencia acorde a los cristales utilizados, en este caso el valor es de 3.

3.6.5 Programa del microcontrolador.

El *firmware* del controlador es sumamente sencillo como se puede ver en la figura 3.13. Al conectarse el microcontrolador al puerto *USB*, este obtiene el voltaje necesario para operar iniciando el programa desde el estado de *RESET*. Después de que el contador de programa es reiniciado, el programa comienza su ejecución. La estructura del programa es la siguiente: El sistema se inicializa con valores preestablecidos para asegurar una operación adecuada, después, en un ciclo infinito, se ejecutan de manera simultánea las tareas que requiere el protocolo del *USB* y los procesos para que la tarjeta opere de la manera deseada.

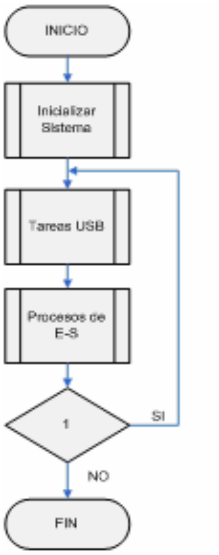


Figura 3.13. Diagrama de flujo del programa.

La rutina que inicializa el sistema (figura 3.14) no hace más que poner el microcontrolador en un estado conocido. Antes que nada, activa el módulo *USB* del PIC e inicia el estado del *USB* como “DESCONECTADO”. Enseguida se configura el puerto de entrada digital, mediante el registro TRIS correspondiente. Después se configura el puerto de salida digital, mediante el registro TRIS correspondiente e inicializando su respectivo registro LATCH a “0”, con la finalidad de asegurar que las salidas comiencen sin voltaje, evitando accionar accidentalmente elementos conectados a estas. La siguiente rutina prepara el puerto de entrada analógica al configurar el módulo *ADC* del PIC, escribiendo el número de canales a utilizar y seleccionando el primero antes de empezar la rutina de escaneo de lecturas. Por último, se presenta la rutina que nos permite configurar el módulo I^2C que comunica con el puerto de salida analógica.

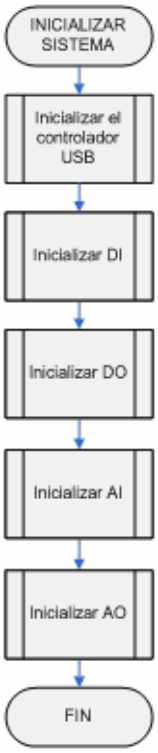


Figura 3.14. Diagrama de flujo de la subrutina Inicializar sistema.

La rutina de las tareas del *USB* se encarga de los procesos propios del protocolo de comunicación. Pasa por la detección del *bus*, la enumeración y las tareas periódicas que mantienen “viva” la comunicación.

Finalmente, la rutina de los procesos de entrada salida se encarga de verificar si existe algún comando enviado por el *host* y, en caso afirmativo, de ejecutar la respuesta esperada. El esquema general de esta subrutina se muestra en la figura 3.15.

Figura 3.15. Diagrama de flujo de los procesos de entrada-salida.

Al iniciar la rutina de procesos se verifica que el dispositivo *USB* este configurado y no se encuentre en estado de suspensión, en caso de ser así procede a ejecutar sus deberes; en caso contrario, termina con esta rutina para permitir que el dispositivo se configure. Una vez configurado y trabajando, el dispositivo atiende a los comandos del *host*, en caso de que los haya, y se presentan de la siguiente forma: identificador de comando, selección del canal y, opcionalmente, el valor del canal.

Después de verificar la existencia del comando, y, en caso de que exista, haberlo atendido, se presenta una subrutina que atiende al proceso de conversión analógico-digital y guarda el valor de cada uno de los canales en un arreglo. Esto permite crear un buffer de

---

* El microcontrolador verifica el estado de la comunicación, en caso de que el dispositivo ya se encuentre configurado (capaz de realizar las comunicaciones), regresa verdadero.

** El sistema cuenta con una cola para agendar las conversiones digital analógico que le sean encomendadas. Si existen conversiones pendientes se entra a la rutina de conversión.

lectura que retiene el último valor leído de cada uno de los canales, solucionando el pequeño inconveniente del ADC multiplexado que contiene el PIC; brindando, hasta cierto punto, una lectura en tiempo real.

En el caso de que en la rutina que atiende a los comandos se haya presentado una escritura analógica, esta es almacenada en una cola. La cola recién mencionada es evaluada, y en caso de tener conversiones pendientes, realiza una conversión digital-analógica. Concluyendo con el ciclo.

La codificación antedicha se realiza por medio de la lectura de 2 o 3 bytes, para lectura o escritura respectivamente, y la respuesta, en caso de tratarse de una lectura, es un byte con el valor del canal. De manera adicional, se implementó en el firmware la capacidad de leer o escribir toda la palabra (los 8 canales) de las entradas y salidas digitales.

Confiando en todos los mecanismos de seguridad de transmisión implementados en el protocolo USB, no se añadió tolerancia a fallas implementada en software.

3.7 Desarrollo del módulo en VB .NET.

Para poder desarrollar aplicaciones de manera sencilla es conveniente generar funciones predeterminadas para manejar la comunicación con la tarjeta de adquisición. Estas funciones pueden ser agrupadas en librerías y así darles portabilidad.

Como se mencionó en la sección que habla del *framework* de *Microchip*, este nos proporciona una *dll* que nos permite interactuar con la tarjeta. Esta *dll* (mpusbapi.dll) debe de ser instalada en la carpeta */Windows/system32* para poder utilizarla.

Una *dll* es una librería con funciones comunes a varios programas pensando en usarla por todos estos programas a la vez, en lugar de que cada programa cargue su propia librería. Para poder acceder a una *dll* desde *VB .NET* debemos de declarar el nombre de la función que se desea utilizar, la librería donde esta función se localiza y el nombre de la función propio de la librería y por último los argumentos necesarios.

Para el desarrollo del módulo que se desea, son necesarias 5 de las funciones de la *dll* estas son: *MPUSBGetDeviceCount*, *MPUSBOpen*, *MPUSBClose*, *MPUSBRead* y *MPUSBWrite*.

La primera función nos permite saber si existen dispositivos conectados, esto lo hace relacionando los identificadores de producto y de vendedor de la tarjeta con los dispositivos *USB* conectados. Las funciones *MPUSBOpen* y *MPUSBClose* nos permiten habilitar o deshabilitar los canales de comunicación respectivamente. Por último, las funciones *MPUSBRead* y *MPUSBWrite*, como su nombre lo dice, son las que permiten escribir o leer de los canales.

Función	Recive	Regresa	Descripción
IniciaUSB	-	-	Prepara las variables del programa
DigitalInput	PIN	VALOR	Lee la entrada digital
DigitalOutput	PIN, VALOR	-	Escribe la salida digital
AnalogInput	PIN	VALOR	Lee la entrada analógica
AnalogOutput	PIN,VALOR	-	Escribe la salida analógica
RESET	-	-	Reinicia el microcontrolador

Tabla 3.3. Descripción de las funciones del módulo "USBctrl.vb".

Mezclando estas funciones se desarrolla el módulo que permite una fácil interacción con el dispositivo. Para esto, se crean las funciones y subrutinas siguientes: *IniciaUSB*, *DigitalInput*, *DigitalOutput*, *AnalogInput*, *AnalogOutput* y *RESET*, cuya descripción se muestra en la tabla 3.3. Estas funciones y subrutinas cumplen con los requisitos de cifrado del microcontrolador.

Una vez terminado el módulo (USBctrl.vb) basta con agregarlo al proyecto en el que se desea comunicar con la tarjeta y se puede hacer uso de las funciones y subrutinas directamente. Se adjunta la programación del módulo en el apéndice B.

3.8 Probando la simulación

Para poder probar el desarrollo completo, es necesario crear un esquemático con el microcontrolador seleccionado y cargarle el programa final. También, es necesario desarrollar un programa de prueba con el modulo desarrollado para tener la comunicación esperada. Adicionalmente hay que agregar al diagrama los periféricos que permitan simular las condiciones de operación normal del circuito.

Para el caso de las entradas digitales simplemente se realiza la conexión del tipo mostrado en la figura 3.16. Esta conexión nos garantiza un nivel lógico de "1" cuando el interruptor se encuentra abierto, o de "0" en caso contrario.

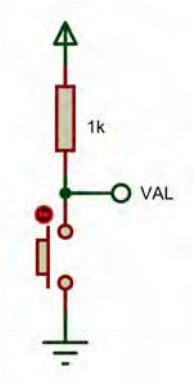


Figura 3.16. Diagrama de conexión de los interruptores.

La prueba de las salidas digitales es más sencilla. Estas se prueban directamente conectando un LED que encienda si el estado de la salida es “1”. Es necesario adicionar una resistencia para optimizar el consumo de energía y proteger al microcontrolador contra sobrecargas.

Conectando un potenciómetro lineal bajo la configuración de divisor de voltaje se puede conseguir una variación de voltaje haciendo variar el valor de la resistencia. Este voltaje puede ser directamente leído por el ADC del PIC e interpretado como un valor digital para su manipulación.

Por último, *ISIS* cuenta con un depurador I^2C incluido. Este depurador nos sirve para verificar que el PIC envíe la información correcta al momento de recibir el comando de escritura analógica.

El resultado de este diagrama esquemático de prueba se puede observar en la figura 3.17. Esta plataforma de prueba es útil también para la depuración del firmware mediante el uso combinado de *ISIS* y *MPLAB*.

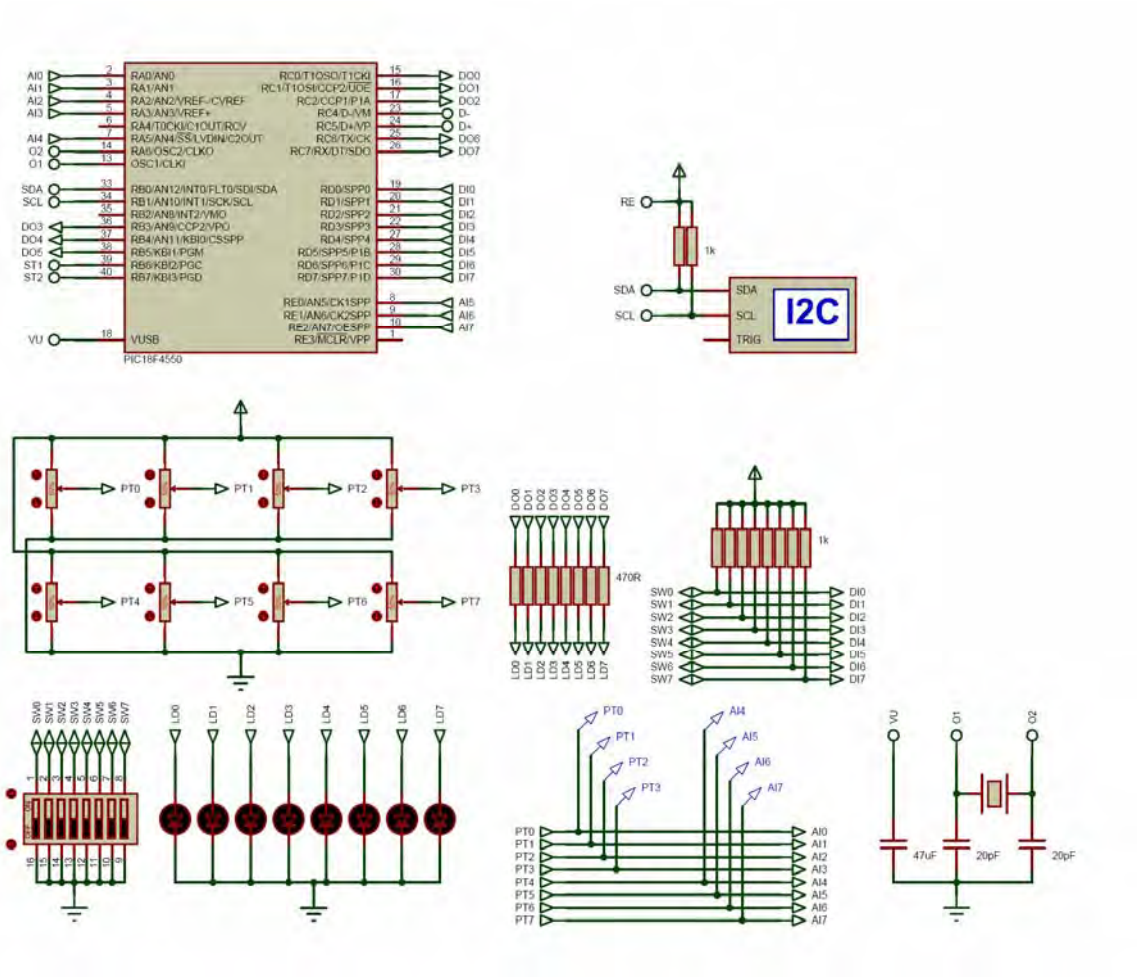


Figura 3.17. Diagrama utilizado en ISIS para probar el dispositivo.

4 Ejemplo de aplicación: Control de velocidad crucero en motores eléctricos.

Habiendo terminado el desarrollo de la tarjeta de adquisición de datos, el siguiente paso lógico a realizar es desarrollar una aplicación en la que se pueda poner a prueba su capacidad. En esta sección podrá verse la facilidad de desarrollo de software que interactúe con la tarjeta.

Para ejemplificar la funcionalidad de la tarjeta se elabora una maqueta en la que se puede observar físicamente el resultado de la interacción con la tarjeta y con una computadora.

4.1 Planteamiento general.

Para poder simular la aplicación se hace uso de un automóvil de juguete, donde lo importante es el eje en el que se ha de controlar la velocidad. Este automóvil se fija a una plataforma y mediante un tornillo, como se muestra en la figura 4.1, que simula la variación de fuerzas necesaria para mover el automóvil en diferentes circunstancias (i.e. más fuerza para subidas, menos fuerza en bajadas).

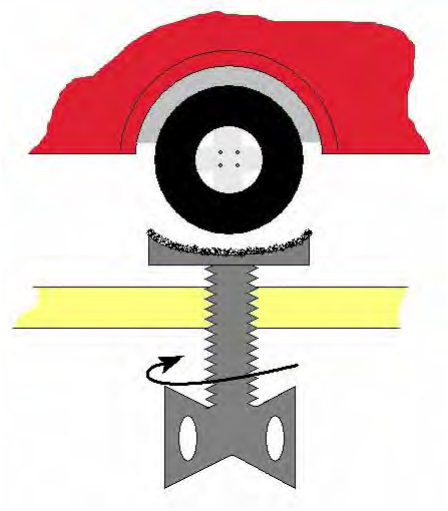


Figura 4.1.Esquema del modelo para el ejemplo de aplicación.

Se escogió un automóvil de juguete para aprovechar su motor eléctrico y su engranaje.

Para poder controlar el sistema requerimos la velocidad del automóvil como entrada y la energía para mover el motor como salida. Se busca un control sencillo, que muestre la facilidad de implementación. Es por esto que se presenta un controlador proporcional al error entre la velocidad buscada y la velocidad real. Este controlador tiene la siguiente forma:

$$y[k+1] = e + y[k]$$

Donde:

- y[k+1] es la salida a calcular.
- e es el error de control calculado como:

$$e = v_{deseada} - v_{real}$$

- y[k] es la última salida conocida.

4.2 Desarrollo de la interfaz eléctrica.

Para obtener la velocidad del automóvil se requiere de un sensor de velocidad. Esto se puede lograr utilizando un motor eléctrico como generador, donde éste nos brinda un voltaje proporcional a la velocidad de giro de su eje.

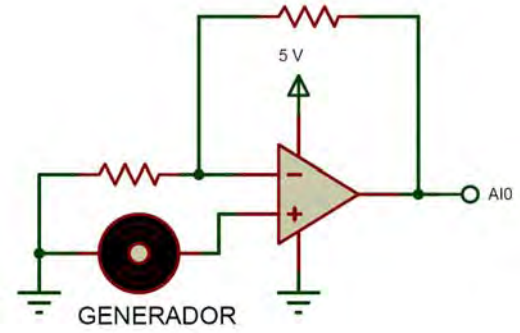


Figura 4.2. Circuito para acondicionar la señal de entrada.

Sin embargo, es necesario acondicionar la señal del motor para que no salga de los rangos de operación de la tarjeta de adquisición. El diagrama eléctrico mostrado en la figura 4.2 muestra el circuito capaz de acondicionar la señal, donde se utiliza un amplificador operacional. Este amplificador operacional se alimenta de 0 y 5 volts, bajo la configuración de amplificador no inversor, para que el valor del motor no pueda exceder el

rango deseado y adicionalmente se le pueda asignar una ganancia al voltaje del generador para escalar la señal.

Por otra parte, se requiere una interfaz de potencia, entre la tarjeta de adquisición y el modelo, para generar una corriente tal que logre mover el motor a la velocidad deseada.

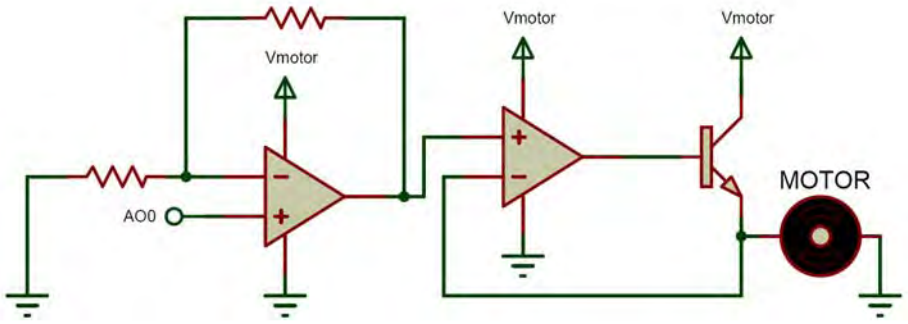


Figura 4.3. Interfaz de potencia para el motor.

El desarrollo de esta interfaz de potencia (figura 4.3) es un tanto más compleja que el acondicionamiento de la señal de entrada. Antes que nada, necesitamos que el voltaje de salida este en el rango de operación del motor eléctrico utilizado (0-voltaje nominal). Esto requiere de una etapa de amplificación de la señal que consta de un amplificador operacional con configuración de amplificador no inversor. Adicionalmente, se requiere propiamente dicho de la parte de potencia, la cual está formada por un amplificador y un transistor. Al combinarse estos elementos en la configuración propuesta, se puede variar el voltaje de operación del motor brindándole la corriente que requiera para la operación adecuada.

De esta manera se puede conectar la tarjeta de adquisición de datos al modelo por un lado y a la computadora por el otro, brindando la capacidad de controlar al modelo a través del puerto *USB*.

4.3 Desarrollo del software.

Para el desarrollo de la aplicación en el host lo primero que hacemos es crear una interfaz gráfica para el usuario (figura 4.4). Gracias a las facilidades que brinda *Visual Basic* lo único que se necesita hacer es colocar los controles prediseñados en una forma natural.

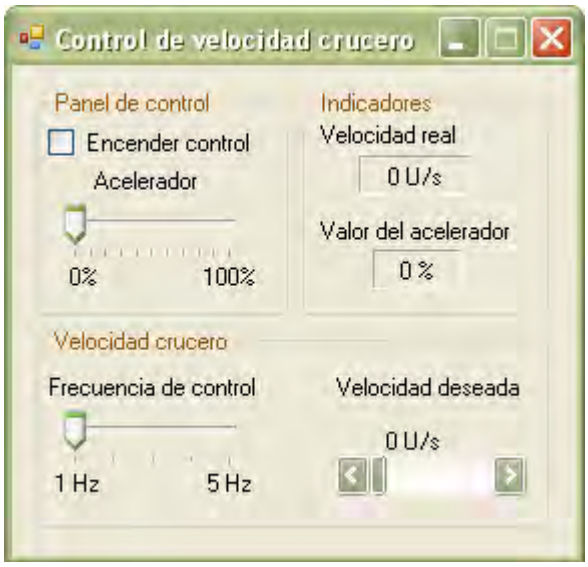


Figura 4.4. Vista de la interfaz gráfica del software.

Una vez desarrollada la interfaz, sólo queda programar las acciones de cada uno de los controles. Sin embargo, sólo rescatamos el siguiente extracto que incluye la parte medular del control y la interacción con la tarjeta:

```
Private Sub Control_Crucero()
 Dim vel_real As UShort, salida As Short
 vel_real = AnalogInput(0)
 salida = hsbVelCrucero.Value - vel_real + trbAcelerador.Value
 If salida > 4095 Then
     trbAcelerador.Value = 4095
 ElseIf salida < 0 Then
     trbAcelerador.Value = 0
 Else
     trbAcelerador.Value = salida
 End If
 AnalogOutput(0, trbAcelerador.Value)
 lblVelReal.Text = vel_real & " U/s"
End Sub
```

De este extracto se puede observar la sencillez para obtener la velocidad real y para generar la salida. El código completo se puede consultar en el apéndice C.

Cabe mencionar que para que funcione el programa hay que agregar al proyecto el módulo “USBctrl.vb” cuyo desarrollo se trató en el capítulo anterior.

5 Resultados y conclusiones.

Como se puede observar de los capítulos anteriores, esta tarjeta de adquisición de datos es de fácil creación y uso. Basta con tener acceso a *Visual Basic .Net* e instalar la *dll* para poder desarrollar aplicaciones que utilicen este circuito.

En los objetivos se busca que un alumno pueda desarrollar proyectos que interactúen con la computadora a través del puerto *USB*. El sistema fue desarrollado de manera que así fuera, sin embargo presenta algunas limitantes.

Al haber sido desarrollado en una computadora cuyo sistema operativo es *Windows XP*, se desconoce la portabilidad a *Windows* de versión superior, o incluso a sistemas operativos de otros fabricantes. Sin embargo, aunque fuera portable a sistemas operativos de otros fabricantes, *Visual Basic .Net* es de uso exclusivo de *Windows*; esto impide que el módulo desarrollado sea de utilidad. Como solución a este pequeño inconveniente se podría generar un módulo similar en lenguajes universales, como *C* o bien *Java* que suponen portabilidad.

La velocidad de transmisión no es tan alta como la de tarjetas de adquisición comerciales. Sin embargo, se podría acercar aún más mediante mejoras en distintos niveles.

El primer nivel de mejora es directamente el cambio de la programación del microcontrolador para hacer rutinas más eficientes. Del mismo modo, el software desarrollado para el host debería de optimizarse, sin embargo esto requiere de una cultura de programación que sale del alcance de esta tesis.

Otro nivel de mejora es el desarrollo completo del *driver*. Esto se debe a que el *driver* que proporciona *Microchip* no es tan rápido como se esperaría. Existen *drivers* de otros fabricantes, inclusive *Windows* proporciona un *driver* genérico (*WINUSB*) que podría ser más rápido.

Por último, un tercer nivel es aquel en el que se busca migrar a las prestaciones más altas del *USB* 2.0 haciendo uso del *high-speed*, o inclusive el *super-speed* tratado en el *USB* 3.0 [8]. Sin embargo, esto implica el cambio de microcontroladores, y así, de toda la circuitería planteada posiblemente encareciendo el proyecto.

Sin embargo, a pesar de lo mencionado, la tesis puede funcionar de manera satisfactoria para todos los proyectos que no requieran velocidades altas de transmisión de datos (del orden de cientos por segundo).

6 Apéndice A: Tarjeta programadora de PIC’s.

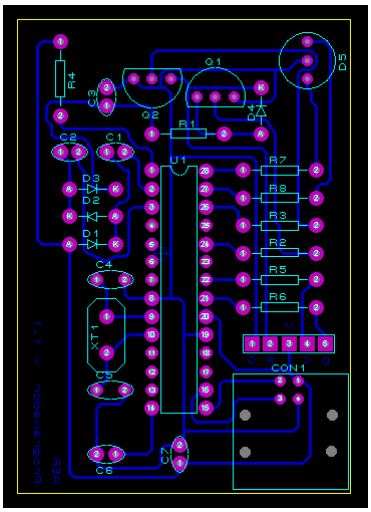


Figura 6.1. Vista general de la tarjeta en “ARES”.

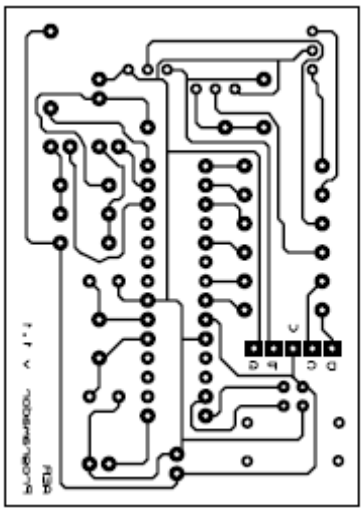


Figura 6.2. PCB de la tarjeta de programación.

REFERENCIA	DESCRIPCIÓN
C1, C2	Capacitor electrolítico de 1 uF
C3	Capacitor electrolítico de 10 uF
C4, C5	Capacitor cerámico de 15 a 30 pF
C6	Capacitor electrolítico de 47 uF
C7	Capacitor electrolítico de 100 uF
CON1	Conector USB-B hembra
D1 - D4	Diodo 1N4148
D5	LED Bicolor
Q1, Q2	Transistor BC548
R1	Resistencia de 2.2 kΩ @ 1/4W
R2, R3	Resistencia de 4.7 kΩ @ 1/4W
R4	Resistencia de 10 kΩ @ 1/4W
R5, R6	Resistencia de 100 Ω @ 1/4W
R7, R8	Resistencia de 1 kΩ @ 1/4W
U1	PIC18F2550
XT1	Crystal de cuarzo de 12MHz

Tabla 6.1. Lista de material de la tarjeta de programación.

7 Apéndice B: Código del módulo USBCtrl.vb.

```
Option Strict Off
Option Explicit On
'================================================================================
'           Código en VB.NET para MPUSBAPI.DLL
'           Probado en Windows XP
'================================================================================

'================================================================================
'=========================    Descripción de Funciones   ========================
'================================================================================
'
'
'IniciaUSB() - Limpia los valores de los EP de comunicación.
'
'ConectaUSB() - Inicia la comunicación entre el dispositivo y la computadora.
'
'DesconectaUSB() - Finaliza la comunicación entre el dispositivo y la computadora.
'
'DigitalInput(PIN)- Lee los canales digitales de entrada
'       PIN (Byte)- Permite seleccionar el pin del que se busca el valor (0-7) o bien
'                   lee todos los canales simultaneamente (8)
'
'DigitalOutput(PIN, VAL)- Escribe los canales digitales de salida
'       PIN (Byte)- Permite seleccionar el pin del que se busca escribir el valor (0-7)
'                   o bien escribe todos los canales simultaneamente (8)
'       VAL (Byte)- Para la escritura por canal recibe un 0 o 1, para la escritura
'                   simultanea recibe el valor final del puerto (0-255)
'
'AnalogInput(PIN)- Lee los canales analógicos de entrada
'       PIN (Byte)- Permite seleccionar el pin del que se busca el valor (0-7)
'
'AnalogOutput(PIN, VALOR)- Escribe los canales analógicos de salida
'       PIN (Byte)- Permite seleccionar el pin del que se busca escribir el valor (0-7)
'       VAL (UShort)-  Recibe el valor con el que se va a escribir el canal (0-4095)
'
'RESET()- Reinicia el microcontrolador
'

Imports System.Runtime.InteropServices.Marshal

Module USBctrl

    Public Declare Function MPUSBGetDLLVersion Lib "mpusbapi.dll" Alias "_MPUSBGetDLLVersion" ()
As Integer
    Public Declare Function MPUSBGetDeviceCount Lib "mpusbapi.dll" Alias "_MPUSBGetDeviceCount"
(ByVal pVID_PID As String) As Integer
    Public Declare Function MPUSBOpen Lib "mpusbapi.dll" Alias "_MPUSBOpen" (ByVal instance As
Integer, ByVal pVID_PID As String, ByVal pEP As String, ByVal dwDir As Integer, ByVal dwReserved
As Integer) As Integer
    Public Declare Function MPUSBClose Lib "mpusbapi.dll" Alias "_MPUSBClose" (ByVal handle As
Integer) As Integer
    Public Declare Function MPUSBRead Lib "mpusbapi.dll" Alias "_MPUSBRead" (ByVal handle As
Integer, ByVal pData As Integer, ByVal dwLen As Integer, ByRef pLength As Integer, ByVal
dwMilliseconds As Integer) As Integer
    Public Declare Function MPUSBWrite Lib "mpusbapi.dll" Alias "_MPUSBWrite" (ByVal handle As
Integer, ByVal pData As Integer, ByVal dwLen As Integer, ByRef pLength As Integer, ByVal
dwMilliseconds As Integer) As Integer
    Public Declare Function MPUSBReadInt Lib "mpusbapi.dll" Alias "_MPUSBReadInt" (ByVal handle
As Integer, ByRef pData() As Byte, ByVal dwLen As Integer, ByRef pLength As Integer, ByVal
dwMilliseconds As Integer) As Integer

    Public Const INVALID_HANDLE_VALUE As Short = -1
    Public Const ERROR_INVALID_HANDLE As Short = 6

    Public Declare Function GetLastError Lib "kernel32" () As Integer

    'Constantes de Microchip
```

```
Public Const vid_pid As String = "vid_04d8&pid_000c"
Public Const out_pipe As String = "\MCHP_EP1"
Public Const in_pipe As String = "\MCHP_EP1"

Public Const MPUSB_FAIL As Short = 0
Public Const MPUSB_SUCCESS As Short = 1

Public Const MP_WRITE As Short = 0
Public Const MP_READ As Short = 1

Public myInPipe, myOutPipe As Integer

Sub IniciaUSB()
    myInPipe = INVALID_HANDLE_VALUE
    myOutPipe = INVALID_HANDLE_VALUE

End Sub

Sub ConectaUSB()
    Dim tempPipe As Integer
    Dim count As Integer

    tempPipe = INVALID_HANDLE_VALUE
    count = MPUSBGetDeviceCount(vid_pid)

    If count > 0 Then

        myOutPipe = MPUSBOpen(0, vid_pid, out_pipe, MP_WRITE, 0)
        myInPipe = MPUSBOpen(0, vid_pid, in_pipe, MP_READ, 0)

        If myOutPipe = INVALID_HANDLE_VALUE Or myInPipe = INVALID_HANDLE_VALUE Then

            MsgBox("Fallo la configuración del dispositivo.")
            myOutPipe = myInPipe = INVALID_HANDLE_VALUE

        End If
    Else
        MsgBox("No hay dispositivos conectados.")
    End If

End Sub

Sub DesconectaUSB()
    If myOutPipe <> INVALID_HANDLE_VALUE Then
        MPUSBClose(myOutPipe)
        myOutPipe = INVALID_HANDLE_VALUE
    End If
    If myInPipe <> INVALID_HANDLE_VALUE Then
        MPUSBClose(myInPipe)
        myInPipe = INVALID_HANDLE_VALUE
    End If

End Sub

Function DigitalInput(ByVal pin As Byte) As Byte
    Dim send_buf(8) As Byte
    Dim receive_buf(8) As Byte

    If pin < 9 Then

        ConectaUSB()

        If myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE Then

            send_buf(0) = 1
            send_buf(1) = pin

            If (SendReceivePacket(send_buf, 2, receive_buf, 3, 500, 500) = 1) Then

                DigitalInput = receive_buf(2)
```

```
            End If

        End If

        DesconectaUSB()

    End If

End Function

Sub DigitalOutput(ByVal pin As Byte, ByVal val As Byte)
    Dim send_buf(8) As Byte
    Dim receive_buf(8) As Byte

    If (pin < 8 And val < 2) Or (pin = 8 And val < 255) Then

        ConectaUSB()

        If myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE Then

            send_buf(0) = 2
            send_buf(1) = pin
            send_buf(2) = val

            SendReceivePacket(send_buf, 3, receive_buf, 3, 500, 500)

        End If

        DesconectaUSB()

    End If

End Sub

Function AnalogInput(ByVal pin As Byte) As UShort
    Dim send_buf(8) As Byte
    Dim receive_buf(8) As Byte

    If pin < 8 Then

        ConectaUSB()

        If myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE Then

            send_buf(0) = 3
            send_buf(1) = pin

            If (SendReceivePacket(send_buf, 2, receive_buf, 3, 500, 500) = 1) Then

                AnalogInput = receive_buf(1) + (receive_buf(2) * 256)

            End If

        End If

        DesconectaUSB()

    End If

End Function

Sub AnalogOutput(ByVal pin As Byte, ByVal val As UShort)
    Dim send_buf(8) As Byte
    Dim receive_buf(8) As Byte
    Dim val1 As Byte
    Dim val2 As Byte

    If (pin < 8) And (val < 4096) Then

        ConectaUSB()
```

```
            val2 = val Mod 256
            val1 = (val - val2) / 256

            pin = (pin << 1) Or 192

            If myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE Then

                send_buf(0) = 4
                send_buf(1) = pin
                send_buf(2) = val1
                send_buf(3) = val2

                SendReceivePacket(send_buf, 4, receive_buf, 3, 500, 500)

            End If

            DesconectaUSB()

        End If

    End Sub

    Sub RESET()
        Dim send_buf(8) As Byte
        Dim receive_buf(8) As Byte

        ConectaUSB()

        If myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE Then

            send_buf(0) = 0

            SendReceivePacket(send_buf, 1, receive_buf, 3, 500, 500)

        End If

        DesconectaUSB()

    End Sub

    Function SendReceivePacket(ByRef SendData() As Byte, ByRef SendLength As Integer, ByRef
ReceiveData() As Byte, ByRef ReceiveLength As Integer, ByVal SendDelay As Integer, ByVal
ReceiveDelay As Integer) As Integer


        Dim SentDataLength As Integer
        Dim ExpectedReceiveLength As Integer

        ExpectedReceiveLength = ReceiveLength

        If (myOutPipe <> INVALID_HANDLE_VALUE And myInPipe <> INVALID_HANDLE_VALUE) Then

            If (MPUSBWrite(myOutPipe,
Runtime.InteropServices.Marshal.UnsafeAddrOfPinnedArrayElement(SendData, 0).ToInt32(),
SendLength, SentDataLength, SendDelay) = MPUSB_SUCCESS) Then

                If (MPUSBRead(myInPipe,
Runtime.InteropServices.Marshal.UnsafeAddrOfPinnedArrayElement(ReceiveData, 0).ToInt32(),
ExpectedReceiveLength, ReceiveLength, ReceiveDelay) = MPUSB_SUCCESS) Then

                    If (ReceiveLength = ExpectedReceiveLength) Then
                        SendReceivePacket = 1 '// Exito
                        Exit Function
                    ElseIf (ReceiveLength < ExpectedReceiveLength) Then
                        SendReceivePacket = 2 '// Recepción incompleta
                        Exit Function
                    End If

                Else
                    CheckInvalidHandle()
                End If
```

```
            Else
                CheckInvalidHandle()
            End If
        End If

        SendReceivePacket = 0 '// Fallo

    End Function

    Sub CheckInvalidHandle()
        If (GetLastError() = ERROR_INVALID_HANDLE) Then
            DesconectaUSB()
        Else
            MsgBox("Error Code : " & Str(GetLastError()))
            DesconectaUSB()
        End If
    End Sub

End Module
```

8 Apéndice C: Código del ejemplo de aplicación.

```
Public Class frmAplicación

    Private Sub chkCrucero_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles chkCrucero.CheckedChanged
        grpCrucero.Enabled = chkCrucero.Checked
        tmrControl.Enabled = chkCrucero.Checked
        trbAcelerador.Enabled = Not chkCrucero.Checked
        tmrAcelerador.Enabled = Not chkCrucero.Checked
    End Sub

    Private Sub trbAcelerador_ValueChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles trbAcelerador.ValueChanged
        Dim temp As Short
        temp = (trbAcelerador.Value / 4095 * 100)
        lblValorAcel.Text = temp & " %"
    End Sub

    Private Sub hsbVelCrucero_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles hsbVelCrucero.Scroll
        lblVelDeseada.Text = hsbVelCrucero.Value & " U/s"
    End Sub

    Private Sub trbFrecuencia_Scroll(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles trbFrecuencia.Scroll
        tmrControl.Interval = 1000 / trbFrecuencia.Value
    End Sub

    Private Sub tmrControl_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
tmrControl.Tick
        tmrControl.Enabled = False
        Control_Crucero()
        tmrControl.Enabled = True
    End Sub

    Private Sub tmrAcelerador_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
tmrAcelerador.Tick
        tmrAcelerador.Enabled = False
        AnalogOutput(0, trbAcelerador.Value)
        lblVelReal.Text = AnalogInput(0) & " U/s"
        tmrAcelerador.Enabled = True
    End Sub

    Private Sub frmAplicación_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        IniciaUSB()
    End Sub

    Private Sub Control_Crucero()
        Dim vel_real As UShort, salida As Short
        vel_real = AnalogInput(0)
        salida = hsbVelCrucero.Value - vel_real + trbAcelerador.Value
        If salida > 4095 Then
            trbAcelerador.Value = 4095
        ElseIf salida < 0 Then
            trbAcelerador.Value = 0
        Else
            trbAcelerador.Value = salida
        End If
        AnalogOutput(0, trbAcelerador.Value)
        lblVelReal.Text = vel_real & " U/s"
    End Sub
       End Class
```

```
        <Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
        Partial Class frmAplicación
            Inherits System.Windows.Forms.Form

            'Form overrides dispose to clean up the component list.
            <System.Diagnostics.DebuggerNonUserCode()> _
            Protected Overrides Sub Dispose(ByVal disposing As Boolean)
                If disposing AndAlso components IsNot Nothing Then
                    components.Dispose()
                End If
                MyBase.Dispose(disposing)
            End Sub

            'Required by the Windows Form Designer
            Private components As System.ComponentModel.IContainer

            'NOTE: The following procedure is required by the Windows Form Designer
            'It can be modified using the Windows Form Designer.
            'Do not modify it using the code editor.
            <System.Diagnostics.DebuggerStepThrough()> _
            Private Sub InitializeComponent()
                Me.components = New System.ComponentModel.Container
                Me.lbl5 = New System.Windows.Forms.Label
                Me.hsbVelCrucero = New System.Windows.Forms.HScrollBar
                Me.trbFrecuencia = New System.Windows.Forms.TrackBar
                Me.grpCrucero = New System.Windows.Forms.GroupBox
                Me.lblVelDeseada = New System.Windows.Forms.Label
                Me.lbl7 = New System.Windows.Forms.Label
                Me.lbl6 = New System.Windows.Forms.Label
                Me.lbl8 = New System.Windows.Forms.Label
                Me.chkCrucero = New System.Windows.Forms.CheckBox
                Me.tmrControl = New System.Windows.Forms.Timer(Me.components)
                Me.grpPanel = New System.Windows.Forms.GroupBox
                Me.lbl2 = New System.Windows.Forms.Label
                Me.lbl1 = New System.Windows.Forms.Label
                Me.lbl0 = New System.Windows.Forms.Label
                Me.trbAcelerador = New System.Windows.Forms.TrackBar
                Me.grpIndicadores = New System.Windows.Forms.GroupBox
                Me.lbl3 = New System.Windows.Forms.Label
                Me.lblVelReal = New System.Windows.Forms.Label
                Me.lblValorAcel = New System.Windows.Forms.Label
                Me.lbl4 = New System.Windows.Forms.Label
                Me.tmrAcelerador = New System.Windows.Forms.Timer(Me.components)
                CType(Me.trbFrecuencia,
System.ComponentModel.ISupportInitialize).BeginInit()
                Me.grpCrucero.SuspendLayout()
                Me.grpPanel.SuspendLayout()
                CType(Me.trbAcelerador,
System.ComponentModel.ISupportInitialize).BeginInit()
                Me.grpIndicadores.SuspendLayout()
                Me.SuspendLayout()
                '
                'lbl5
                '
                Me.lbl5.AutoSize = True
                Me.lbl5.Location = New System.Drawing.Point(3, 23)
                Me.lbl5.Name = "lbl5"
                Me.lbl5.Size = New System.Drawing.Size(110, 13)
                Me.lbl5.TabIndex = 0
                Me.lbl5.Text = "Frecuencia de control"
                '
                'hsbVelCrucero
                '
                Me.hsbVelCrucero.LargeChange = 1
                Me.hsbVelCrucero.Location = New System.Drawing.Point(150, 63)
                Me.hsbVelCrucero.Maximum = 1000
                Me.hsbVelCrucero.Name = "hsbVelCrucero"
                Me.hsbVelCrucero.Size = New System.Drawing.Size(95, 21)
                Me.hsbVelCrucero.TabIndex = 4
                '
                'trbFrecuencia
```

```
'
Me.trbFrecuencia.BackColor = System.Drawing.SystemColors.Control
Me.trbFrecuencia.LargeChange = 10
Me.trbFrecuencia.Location = New System.Drawing.Point(6, 39)
Me.trbFrecuencia.Maximum = 5
Me.trbFrecuencia.Minimum = 1
Me.trbFrecuencia.Name = "trbFrecuencia"
Me.trbFrecuencia.Size = New System.Drawing.Size(104, 45)
Me.trbFrecuencia.TabIndex = 3
Me.trbFrecuencia.TabStop = False
Me.trbFrecuencia.Value = 1
'
'grpCrucero
'
Me.grpCrucero.Controls.Add(Me.lblVelDeseada)
Me.grpCrucero.Controls.Add(Me.lbl7)
Me.grpCrucero.Controls.Add(Me.lbl6)
Me.grpCrucero.Controls.Add(Me.trbFrecuencia)
Me.grpCrucero.Controls.Add(Me.lbl8)
Me.grpCrucero.Controls.Add(Me.lbl5)
Me.grpCrucero.Controls.Add(Me.hsbVelCrucero)
Me.grpCrucero.Enabled = False
Me.grpCrucero.Location = New System.Drawing.Point(12, 131)
Me.grpCrucero.Name = "grpCrucero"
Me.grpCrucero.Size = New System.Drawing.Size(256, 100)
Me.grpCrucero.TabIndex = 3
Me.grpCrucero.TabStop = False
Me.grpCrucero.Text = "Velocidad crucero"
'
'lblVelDeseada
'
Me.lblVelDeseada.Location = New System.Drawing.Point(158, 50)
Me.lblVelDeseada.Name = "lblVelDeseada"
Me.lblVelDeseada.Size = New System.Drawing.Size(60, 13)
Me.lblVelDeseada.TabIndex = 3
Me.lblVelDeseada.Text = "0 U/s"
Me.lblVelDeseada.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
'
'lbl7
'
Me.lbl7.AutoSize = True
Me.lbl7.Location = New System.Drawing.Point(82, 71)
Me.lbl7.Name = "lbl7"
Me.lbl7.Size = New System.Drawing.Size(29, 13)
Me.lbl7.TabIndex = 3
Me.lbl7.Text = "5 Hz"
'
'lbl6
'
Me.lbl6.AutoSize = True
Me.lbl6.Location = New System.Drawing.Point(6, 71)
Me.lbl6.Name = "lbl6"
Me.lbl6.Size = New System.Drawing.Size(29, 13)
Me.lbl6.TabIndex = 3
Me.lbl6.Text = "1 Hz"
'
'lbl8
'
Me.lbl8.AutoSize = True
Me.lbl8.Location = New System.Drawing.Point(147, 23)
Me.lbl8.Name = "lbl8"
Me.lbl8.Size = New System.Drawing.Size(98, 13)
Me.lbl8.TabIndex = 0
Me.lbl8.Text = "Velocidad deseada"
'
'chkCrucero
'
Me.chkCrucero.AutoSize = True
Me.chkCrucero.Location = New System.Drawing.Point(6, 19)
Me.chkCrucero.Name = "chkCrucero"
Me.chkCrucero.Size = New System.Drawing.Size(107, 17)
```

```
Me.chkCrucero.TabIndex = 0
Me.chkCrucero.Text = "Encender control"
Me.chkCrucero.UseVisualStyleBackColor = True
'
'tmrControl
'
Me.tmrControl.Interval = 1000
'
'grpPanel
'
Me.grpPanel.Controls.Add(Me.lbl2)
Me.grpPanel.Controls.Add(Me.lbl1)
Me.grpPanel.Controls.Add(Me.lbl0)
Me.grpPanel.Controls.Add(Me.trbAcelerador)
Me.grpPanel.Controls.Add(Me.chkCrucero)
Me.grpPanel.Location = New System.Drawing.Point(12, 12)
Me.grpPanel.Name = "grpPanel"
Me.grpPanel.Size = New System.Drawing.Size(126, 113)
Me.grpPanel.TabIndex = 4
Me.grpPanel.TabStop = False
Me.grpPanel.Text = "Panel de control"
'
'lbl2
'
Me.lbl2.AutoSize = True
Me.lbl2.Location = New System.Drawing.Point(80, 87)
Me.lbl2.Name = "lbl2"
Me.lbl2.Size = New System.Drawing.Size(33, 13)
Me.lbl2.TabIndex = 3
Me.lbl2.Text = "100%"
'
'lbl1
'
Me.lbl1.AutoSize = True
Me.lbl1.Location = New System.Drawing.Point(13, 87)
Me.lbl1.Name = "lbl1"
Me.lbl1.Size = New System.Drawing.Size(21, 13)
Me.lbl1.TabIndex = 3
Me.lbl1.Text = "0%"
'
'lbl0
'
Me.lbl0.AutoSize = True
Me.lbl0.Location = New System.Drawing.Point(26, 39)
Me.lbl0.Name = "lbl0"
Me.lbl0.Size = New System.Drawing.Size(58, 13)
Me.lbl0.TabIndex = 3
Me.lbl0.Text = "Acelerador"
'
'trbAcelerador
'
Me.trbAcelerador.LargeChange = 400
Me.trbAcelerador.Location = New System.Drawing.Point(6, 55)
Me.trbAcelerador.Maximum = 4095
Me.trbAcelerador.Name = "trbAcelerador"
Me.trbAcelerador.Size = New System.Drawing.Size(104, 45)
Me.trbAcelerador.SmallChange = 40
Me.trbAcelerador.TabIndex = 1
Me.trbAcelerador.TabStop = False
Me.trbAcelerador.TickFrequency = 409
'
'grpIndicadores
'
Me.grpIndicadores.Controls.Add(Me.lbl3)
Me.grpIndicadores.Controls.Add(Me.lblVelReal)
Me.grpIndicadores.Controls.Add(Me.lblValorAcel)
Me.grpIndicadores.Controls.Add(Me.lbl4)
Me.grpIndicadores.Location = New System.Drawing.Point(145, 12)
Me.grpIndicadores.Name = "grpIndicadores"
Me.grpIndicadores.Size = New System.Drawing.Size(123, 113)
Me.grpIndicadores.TabIndex = 5
```

```
        Me.grpIndicadores.TabStop = False
        Me.grpIndicadores.Text = "Indicadores"
        '
        'lbl3
        '
        Me.lbl3.AutoSize = True
        Me.lbl3.Location = New System.Drawing.Point(6, 16)
        Me.lbl3.Name = "lbl3"
        Me.lbl3.Size = New System.Drawing.Size(74, 13)
        Me.lbl3.TabIndex = 0
        Me.lbl3.Text = "Velocidad real"
        '
        'lblVelReal
        '
        Me.lblVelReal.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.lblVelReal.Location = New System.Drawing.Point(28, 33)
        Me.lblVelReal.Name = "lblVelReal"
        Me.lblVelReal.Size = New System.Drawing.Size(57, 20)
        Me.lblVelReal.TabIndex = 3
        Me.lblVelReal.Text = "0 U/s"
        Me.lblVelReal.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'lblValorAcel
        '
        Me.lblValorAcel.BackColor = System.Drawing.SystemColors.Control
        Me.lblValorAcel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.lblValorAcel.Location = New System.Drawing.Point(35, 80)
        Me.lblValorAcel.Name = "lblValorAcel"
        Me.lblValorAcel.Size = New System.Drawing.Size(45, 20)
        Me.lblValorAcel.TabIndex = 3
        Me.lblValorAcel.Text = "0 %"
        Me.lblValorAcel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'lbl4
        '
        Me.lbl4.AutoSize = True
        Me.lbl4.Location = New System.Drawing.Point(6, 64)
        Me.lbl4.Name = "lbl4"
        Me.lbl4.Size = New System.Drawing.Size(101, 13)
        Me.lbl4.TabIndex = 3
        Me.lbl4.Text = "Valor del acelerador"
        '
        'tmrAcelerador
        '
        Me.tmrAcelerador.Enabled = True
        Me.tmrAcelerador.Interval = 200
        '
        'frmAplicación
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(282, 245)
        Me.Controls.Add(Me.grpIndicadores)
        Me.Controls.Add(Me.grpPanel)
        Me.Controls.Add(Me.grpCrucero)
        Me.MaximizeBox = False
        Me.Name = "frmAplicación"
        Me.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen
        Me.Text = "Control de velocidad crucero"
        CType(Me.trbFrecuencia, System.ComponentModel.ISupportInitialize).EndInit()
        Me.grpCrucero.ResumeLayout(False)
        Me.grpCrucero.PerformLayout()
        Me.grpPanel.ResumeLayout(False)
        Me.grpPanel.PerformLayout()
        CType(Me.trbAcelerador, System.ComponentModel.ISupportInitialize).EndInit()
        Me.grpIndicadores.ResumeLayout(False)
        Me.grpIndicadores.PerformLayout()
        Me.ResumeLayout(False)

    End Sub
    Friend WithEvents lbl5 As System.Windows.Forms.Label
```

```
    Friend WithEvents hsbVelCrucero As System.Windows.Forms.HScrollBar
    Friend WithEvents trbFrecuencia As System.Windows.Forms.TrackBar
    Friend WithEvents grpCrucero As System.Windows.Forms.GroupBox
    Friend WithEvents chkCrucero As System.Windows.Forms.CheckBox
    Friend WithEvents lbl6 As System.Windows.Forms.Label
    Friend WithEvents lbl7 As System.Windows.Forms.Label
    Friend WithEvents lblVelDeseada As System.Windows.Forms.Label
    Friend WithEvents lbl8 As System.Windows.Forms.Label
    Friend WithEvents tmrControl As System.Windows.Forms.Timer
    Friend WithEvents grpPanel As System.Windows.Forms.GroupBox
    Friend WithEvents lbl0 As System.Windows.Forms.Label
    Friend WithEvents trbAcelerador As System.Windows.Forms.TrackBar
    Friend WithEvents lbl2 As System.Windows.Forms.Label
    Friend WithEvents lbl1 As System.Windows.Forms.Label
    Friend WithEvents grpIndicadores As System.Windows.Forms.GroupBox
    Friend WithEvents lbl3 As System.Windows.Forms.Label
    Friend WithEvents lblValorAcel As System.Windows.Forms.Label
    Friend WithEvents lbl4 As System.Windows.Forms.Label
    Friend WithEvents lblVelReal As System.Windows.Forms.Label
    Friend WithEvents tmrAcelerador As System.Windows.Forms.Timer

End Class
```

9 Mesografía.

[1]. **Hyde, John**. "USB Design by Example. A Practical Guide to Building I/O Devices". Segunda Edición, Intel Press, USA, 2001.

[2]. **Axelson, Jan**. *"USB Complete: Everything You Need to Develop USB Peripherals"*. Tercera Edición, Lakeview Research, USA, 2005.

[3]. **Axelson, Jan**. "Parallel Port Complete: Programming, Interfacing, & Using the PC's Parallel Printer Port". Primera Edición, Lakeview Research, USA, 1998.

[4]. **Gadre, Dhananjay**. "Programming the Parallel Port: Interfacing the PC for Data Acquisition and Process Control". Primera Edición, R&D Books, USA, 1998.

[5]. **Axelson, Jan**. "Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks". Primera Edición, Lakeview Research, USA, 2000.

[6]. **Kernighan, Brian; Ritchie, Dennis**. *"The C Programming Language"*. Segunda Edición, Prentice-Hall, USA, 1991

[7]. **Wakefield, Cameron**. *"VB.NET Developer's Guide"*. Primera Edición, Syngress Publishing, USA, 2001.

[8]. **http://www.usb.org/** *"USB.org – Welcome"*. [Recuperado en Diciembre 2008].

[9]. **http://www.ni.com/** *"National Instruments – Test and Measurement"*. [Recuperado en Diciembre 2008].

[10]. **http://www.microchip.com/** "Microchip Technology Inc. is a Leading Provider of Microcontroller and Analog Semiconductors, providing low-risk product development, lower total system cost and faster time to market for thousands of diverse customer applications worldwide". [Recuperado en Diciembre 2008].

[11]. **http://www.lvr.com/** *"Jan Axelson's Lakeview Research"*. [Recuperado en Diciembre 2008].

[12]. **http://www.i2c-bus.org/** *"I2C-Bus: What's that?"*. [Recuperado en Diciembre 2008].