ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

"DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA EXPERIMENTAL PARA EL ANÁLISIS COMPUTACIONAL DEL MOVIMIENTO RECTILÍNEO MEDIANTE ADQUISICIÓN DE DATOS POR EL PÓRTICO USB."

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

JORGE RODRIGO MERA QUILLUPANGUI
MARIO JAVIER SAAVEDRA MANZANILLAS

DIRECTOR: DR. LUIS CORRALES

Quito, Octubre 2004

DECLARACIÓN

Nosotros, Jorge Rodrigo Mera Quillupangui y Mario Javier Saavedra Manzanillas, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

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

Jorge Rodrigo Mera Quillupangui

Mario Javier Saavedra Manzanillas

CERTIFICACIÓN

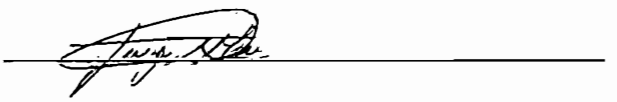
Certifico que el presente trabajo fue desarrollado por Jorge Rodrigo Mera Quillupangui y Mario Javier Saavedra Manzanillas, bajo mi supervisión.

Dr. Luis Corrales
DIRECTOR DEL PROYECTO

DEDICATORIA

Dedico el presente trabajo a mis Padres Carlos y Margarita por todo su amor, ejemplo y sacrificio. A mis hermanos Carmen, Luis, Lidia, Patricia, Martha y a Nancy de quienes siempre tuve una palabra de aliento.

Este trabajo también lo dedico a todos mis sobrinos.



Jorge Rodrigo Mera Quillupangui

DEDICATORIA

Dedico especialmente este trabajo a mi Madre Melva, a mi Padre Mario y a Jehová DIOS que solo por ellos he luchado tanto y he podido culminar este proyecto.

Dedico también este trabajo con mucho cariño a mis tres queridos hermanos Jhonny, Pato, Luchín y a mi sobrinita Abigail porque siempre se preocuparon en saber como adelantaba en mi carrera y en la conclusión de este proyecto.

Dedico también este trabajo con mucho cariño a todos mis tíos, tías y primos que siempre me apoyaron para culminar mi carrera.

Mario Javier Saavedra Manzanillas

CONTENIDO

RESUMEN.....Vii

PRESENTACIÓN.....viii

CAPÍTULO 1: FUNDAMENTO TEÓRICO DEL PUERTO USB

1.1 OBJETIVO GENERAL.....1
1.2 FUNDAMENTOS FÍSICOS DEL MOVIMIENTO RECTÍLINEO.....1
1.2.1 POSICIÓN.....2
1.2.2 VELOCIDAD.....3
1.2.3 ACELERACIÓN.....3
1.2.4 SEGUNDA LEY DE NEWTON.....4
1.2.5 DESCRIPCIÓN DE LA PLANTA.....5
1.4 ULTRASONIDO.....6
1.3.1 PROPAGACIÓN DEL ULTRASONIDO.....6
1.3.2 IMPEDANCIA ACÚSTICA.....7
1.3.3 SENSORES DE ULTRASONIDO.....8
1.3.3.1 Efecto Piezoeléctrico.....8
1.3.3.2 Efecto Electrostático.....8
1.3.4 PARÁMETROS A CONSIDERAR EN UN SISTEMA ULTRASÓN!CO.....9
1.3.5 DETERMINACIÓN DE LA DISTANCIA MEDIANTE EL ULTRASONIDO.....10
1.4 ASPECTOS GENERALES DEL PUERTO USB.....11
1.4.1 BENEFICIOS DEL USB.....12
1.4.2 DESVENTAJAS DEL USB.....14
1.5 TOPOLOGÍA DEL BUS.....15
1.5.1 TOPOLOGÍA FÍSICA DE BUS.....15
1.5.2 TOPOLOGÍA LÓGICA DEL BUS.....16
1.5.3 RELACIÓN SOFTWARE CLIENTE -A- FUNCIÓN.....17
1.6 ESPECIFICACIÓN ELÉCTRICA DEL USB.....17
1.6.1 VOLTAJES.....18
1.6.2 DISPOSITIVOS ALIMENTADOS POR EL BUS (BUS – POWERED).....18
1.6.3 DISPOSITIVOS AUTO – ALIMENTADOS (SELF – POWERED).....18

1.7 COMPONENTES FÍSICOS DEL SISTEMA USB....19
1.7.1 DISPOSITIVOS....19
1.7.2 CABLES....19
1.7.3 CONECTORES....20
1.8 ANÁLISIS BÁSICO DE LAS TRANSFERENCIAS USB....21
1.8.1 COMUNICACIONES DE CONFIGURACIÓN....22
1.8.2 COMUNICACIONES DE APLICACIÓN....22
1.8.3 MANEJO DE DATOS EN EL BUS....22
1.8.3.1 Codificación de la señal....24
1.8.4 ELEMENTOS DE UNA TRANSFERENCIA....25
1.8.4.1 Endpoints de dispositivos....25
1.8.4.2 Pipes....27
1.8.4.2.1 Pipes de flujo (stream) y de mensaje (message)....27
1.8.4.3 Inicializando una transferencia....28
1.8.4.4 Bloques de una transferencia....29
1.8.4.5 Fases de una transacción....30
1.8.5 CODIGOS DE HANDSHAKE....31
1.8.5.1 Ack....31
1.8.5.2 Nak....31
1.8.5.3 Stall....32
1.8.5.4 NYet (not yet)....32
1.8.5.5 Sin respuesta....32
1.9 TIPOS DE TRANSFERENCIAS USB....32
1.9.1 TRANSFERENCIAS DE CONTROL....33
1.9.1.1 Disponibilidad....33
1.9.1.2 Estructura....33
1.9.1.3 Tamaño de los datos....34
1.9.1.4 Velocidad....34
1.9.2 TRANSFERENCIAS BULK....35
1.9.2.1 Disponibilidad....35
1.9.2.2 Estructura....35
1.9.2.3 Tamaño de los datos....35
1.9.2.4 Velocidad....36

1.9.3 TRANSFERENCIA DE INTERRUPCIÓN....36
1.9.3.1 Disponibilidad....37
1.9.3.2 Estructura....37
1.9.3.3 Tamaño de los datos....37
1.9.3.4 Velocidad....38
1.9.4 TRANSFERENCIAS ISOCRÓNICAS....38
1.9.4.1 Disponibilidad....39
1.9.4.2 Estructura....39
1.9.4.3 Tamaño de los datos....39
1.9.4.4 Velocidad....39
1.10 PROCESO DE ENUMERACIÓN....40
1.10.1 PASOS DE ENUMERACIÓN....42
1.10.2 REMOCIÓN DEL DISPOSITIVO....45
1.10.3 TIPOS DE DESCRIPTORES....45
1.10.3.1 Tipos....46
1.11 COMO SE COMUNICA EL HOST CON EL DISPOSITIVO.... 48
1.11.1 CONCEPTOS BÁSICOS SOBRE EL DRIVER DE DISPOSITIVO..... 48
1.11.2 OPCIONES PARA DRIVERS DEL DISPOSITIVO.... 49
1.11.2.1 Tipos de dispositivos estándar.... 49
1.11.2.2 Dispositivos personalizados.... 49
1.11.3 LENGUAJES DE PROGRAMACIÓN.... 50
1.11.4 PROCESO DE SELECCIÓN DE UN DRIVER BAJO WINDOWS..... 50
1.11.5 CLASES DE DISPOSITIVOS.... 51
1.11.6 DISPOSITIVO DE INTERFAZ HUMANA (HID).... ... 51
1.11.6.1 Requerimientos de hardware.... 52
1.11.6.1.1 Requerimientos de endpoints.... 52
1.11.6.1.2 Requerimiento del "pipe" de control.... 52
1.11.6.1.3 Requerimientos de transferencias de interrupción.... 53
1.11.6.2 Requerimientos de firmware.... 53
1.11.6.3 Identificando un dispositivo como HID.... 53
1.11.6.4 Contenido de los descriptores.... 54
1.11.6.4.1 Descriptor de clase HID.... 54
1.11.6.4.2 Descriptor de reporte.... 54

1.12 PROPUESTA PARA EL SISTEMA DE ADQUISICIÓN DE DATOS.....55
1.12.1 MÓDULO EXPERIMENTAL..... 55
1.12.2 SENSORES..... 56
1.12.3 UNIDAD DE ADQUISICIÓN DE DATOS..... 56
1.12.4 PC CON INTERFAZ USB..... 56

CAPÍTULO 2: CONSTRUCCIÓN DEL HARDWARE DEL SISTEMA

2.1 INTRODUCCIÓN.....57
2.2MÓDULO EXPERIMENTAL.....57
2.3 MÓDULO ELECTRÓNICO.....58
2.3.1 SELECCIÓN DEL SENSOR ULTRASÓNICO.....60
2.3.2 DISEÑO Y CONTRUCCIÓN DEL CIRCUITO ACONDICIONADOR DE SEÑAL PARA LA EMISIÓN Y RECEPCIÓN DE LA SEÑAL DE ULTRASONIDO 61
2.3.2.1 Diseño del circuito de emisión de ultrasonido..... 62
2.3.2.2 Diseño del circuito receptor de ultrasonido..... 64
2.3.2.2.1 Circuito de acoplamiento.....67
2.3.2.2.2 Circuito de amplificación.....68
2.3.2.2.3 Circuito de comparación.....73

CAPÍTULO 3: DISEÑO DEL SOFTWARE DEL SISTEMA

3.1 INTRODUCCIÓN.....77
3.2 DESARROLLO DEL PROGRAMA PARA EL MICROCONTROLADOR.....77
3.2.1 ATENCIÓN A LA INTERRUPCIÓN.....78
3.2.1.1 RUTINAS DE INTERRUPCIÓN.....79
3.2.2 INICIALIZAR USB.....83
3.2.3 MEDICIÓN TIEMPO DE ENVIO.....84
3.2.4 INICIAR TEMPORIZADOR TIEMPO DE VUELO.....84
3.2.5 EMISIÓN DE PULSOS DE ULTRASONIDO.....85
3.2.6 MEDICIÓN DEL TIEMPO DE VUELO.....85
3.2.7 INICIAR TEMPORIZADOR TIEMPO DE ENVÍO.....86
3.2.8 ENVÍO DE LA INFORMACIÓN POR EL USB.....86

3.3 DESARROLLO DEL SOFTWARE DEL PC....86
3.3.1 SOFTWARE DE SOPORTE....87
3.3.2VENTANA DE PRESENTACIÓN....90
3.3.3 FORMULARIO PRINCIPAL....90
3.3.4 FORMULARIO POSICIÓN....95
3.3.5 FORMULARIO VELOCIDAD....97
3.3.6 FORMULARIO ACELERACIÓN....99

CAPÍTULO 4: PRUEBAS Y RESULTADOS

4.1 PRUEBAS Y RESULTADOS DEL HARDWARE COMPLETO DE MEDICIÓN DE POSICIÓN.... 103
4.1.1 PRUEBAS Y RESULTADOS REALIZADOS CON EL OBJETO EN REPOSO 103
4.1.2 PRUEBAS Y RESULTADOS REALIZADOS CON EL OBJETO EN MOVIMIENTO 106
4.1.2.1 Pruebas y resultados para el algoritmo de derivación....107
4.1.2.2 Pruebas y resultados realizados para el cálculo de la aceleración en base a la medición de distancia....109
4.1.3 PRUEBAS Y RESULTADOS DE LA INSTALACIÓN Y FUNCIONAMIENTO DEL SOFTWARE 113
4.1.4 ANÁLISIS DE COSTOS....114

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES....115

5.2 RECOMENDACIONES.... 118

BIBLIOGRAFÍA....120

ANEXOS

ANEXO A	DESCRIPTORES
ANEXO B	GLOSARIO DE TÉRMINOS USB
ANEXO C	DATOS TÉCNICOS DEL MICROCONTROLADOR PIC
ANEXO D	DATOS TÉCNICOS DE LOS ELEMENTOS

RESUMEN

En el presente trabajo se diseña y construye un módulo experimental para el análisis de las principales variables que rigen el movimiento rectilíneo y la adquisición de estos datos hacia un PC para su visualización por medio del pórtico USB

El módulo experimental está construido a base de un microcontrolador PIC 16C745, maneja la emisión y recepción de los pulsos de ultrasonido para la medición de la distancia del objeto cuyo movimiento se desea analizar, además este chip tiene como característica principal la incorporación de la interfaz USB que maneja una buena parte del protocolo por medio de hardware.

El programa de visualización de las variables del movimiento como son pocisiòn, velocidad y aceleración se desarrolla en el lenguaje computacional con interfaz gráfica Visual Basic 6.0 de Microsoft, desde el cual se podrá establecer comunicación entre el PC y el dispositivo construido, adquirir los datos necesarios para posteriormente procesarlos y visualizar las variables que rigen el movimiento del cuerpo a analizar.

La comunicación USB se implementa bajo la especificación 2.0 del USB para baja velocidad y todo el handshake necesario para la transmisión en ambas vías de la información de acuerdo al estándar.

Luego de haber concluido este proyecto se tiene un error del 8% en el valor experimental de la aceleración con respecto al esperado. En cambio el valor obtenido experimentalmente de la distancia tiene errores menores al 1%.

PRESENTACIÓN

Hoy en día el computador es una herramienta indispensable en todo ámbito social y más aun en el campo de la educación; resulta de utilidad entonces, disponer de una herramienta que permita por ejemplo realizar el análisis de fenómenos físicos mediante un PC y de un dispositivo diseñado para tal propósito, de forma que mediante pruebas realizadas sobre un prototipo construido para ello, ayude a determinar el comportamiento de tal fenómeno.

Con este objetivo en mente, este proyecto se desarrolla para satisfacer una de tales necesidades, pues se comienza con la implementación de un sistema que permita el análisis computacional del movimiento rectilíneo de un objeto. Esta tarea se completa con el diseño del software adecuado, que realice la adquisición de tales variables hacia un computador con sistemas operativos de 32 bits con el suficiente soporte USB.

Para cumplir con dicho objetivo en el Capítulo 1 se hace un análisis completo sobre el pórtico USB, un estudio del ultrasonido y su aplicación en la medición de posición además del análisis de las ecuaciones físicas de la planta.

En el Capítulo 2 se realiza el diseño del hardware a implementar para realizar las mediciones de posición.

El Capítulo 3 trata el diseño del software que se ejecutará, tanto en el microcontrolador como en la PC, para realizar la adquisición de datos y visualización de los mismos.

En el Capítulo 4 se presentan las pruebas y los resultados realizados sobre el sistema implementado.

El Capítulo 5 presenta las conclusiones y recomendaciones sobre el proyecto.

CAPÍTULO 1

FUNDAMENTO TEÓRICO DEL PUERTO USB

CAPÍTULO 1

FUNDAMENTO TEÓRICO DEL PUERTO USB

1.1 OBJETIVO GENERAL

En el presente trabajo se pretende implementar un módulo experimental que permita obtener los principales parámetros del movimiento rectilíneo, y realizar la adquisición de estos datos hacia una PC, en la cual se diseñará un programa que realice el análisis de los mismos.

Se ha propuesto que las mediciones se realicen por medio de sensores ultrasónicos, de los cuales se piensa aprovechar la técnica de medición del tiempo de vuelo para adquirir las variables de tiempo y posición del objeto en experimentación y, mediante estos datos, el software que residirá en el computador, se encargará de deducir la velocidad y aceleración, así como de representarlos gráficamente.

Lo novedoso de este proyecto es que la adquisición de datos se realizará mediante el pórtico USB del cual se tratará más adelante en este capítulo como fundamento para el desarrollo posterior de la aplicación. A continuación se empieza por estudiar los fundamentos físicos del movimiento rectilíneo cuyo comportamiento es el que debe analizarse en este proyecto.

1.2 FUNDAMENTOS FÍSICOS DEL MOVIMIENTO RECTILÍNEO

La parte de la física que estudia el movimiento de los cuerpos, en los que se incluye el movimiento rectilíneo es la Cinemática. Esta rama de la física no se interesa en las causas que originen dicho movimiento. Las magnitudes que define la cinemática son principalmente tres: la posición, la velocidad y la aceleración.

Por otro lado, la rama de la física que se encarga de analizar las causas que generan el movimiento de los cuerpos se llama Dinámica. Estas causas son el resultado directo de la interacción del cuerpo analizado con otros que lo rodean, y son bien definidas por un concepto matemático denominado fuerza que tiene características vectoriales. Las leyes que definen la dinámica de un cuerpo son las conocidas tres leyes de Newton.

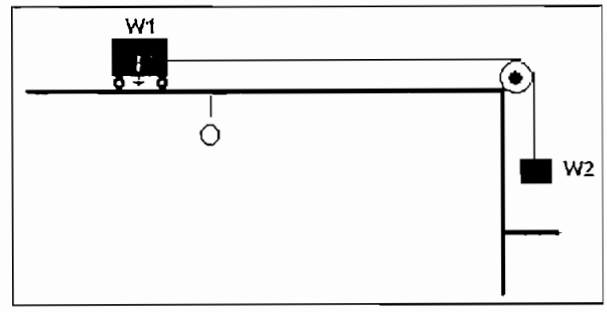


Figura 1.1 Modelo físico del sistema a analizar

Dado que el presente proyecto analiza las tres principales magnitudes del movimiento en un sistema como el que se muestra en la Figura 1.1, es suficiente tomar en cuenta los siguientes cuatro principios físicos que se detallan a continuación.

1.2.1 POSICIÓN

Es el lugar en que se encuentra el móvil en un cierto instante de tiempo ***t***. Suele representarse con el vector de posición:

$$\vec{X}(t)$$

El conocimiento de este vector junto con la variable de tiempo, daría toda la información necesaria para realizar el cálculo de los otros parámetros cinemáticos. La diferencia de un vector de posición final y un vector de posición inicial define el desplazamiento.

1.2.2 VELOCIDAD

Es la relación que se establece entre el desplazamiento realizado por la partícula en movimiento y el intervalo de tiempo en que se efectuó:

$$\vec{v}_m = \frac{\Delta \vec{X}}{\Delta t} \tag{1.1}$$

La ecuación anterior define la velocidad en la que los intervalos de tiempo son apreciablemente mayores que cero como la velocidad media. No obstante aunque ésta es útil, hay que destacar que en su cálculo se deja mucha información sin precisar, por ello se define una magnitud que exprese la velocidad instantánea, es decir, la velocidad en cierto y determinado instante y que pueda calcularse como una velocidad media donde los intervalos sean tan pequeños que pueda decirse exactamente a qué velocidad se desplazaba el móvil en cada instante. Tal definición se logra tomando como velocidad instantánea la siguiente expresión:

$$\vec{v} = \lim_{\Delta t \to 0} \frac{\Delta \vec{X}}{\Delta t} \tag{1.2}$$

La misma ecuación anterior puede ser representada mediante la notación de puntos, que representa la primera derivada de la posición con respecto al tiempo:

$$v = \dot{x}(t)$$

1.2.3 ACELERACIÓN

Es la relación que se establece entre la variación de la velocidad que experimenta un móvil y el tiempo en el que se realizó tal variación:

$$\vec{a} = \frac{\Delta \vec{v}}{\Delta t} \tag{1.3}$$

La ecuación anterior define la aceleración, en la que los intervalos de tiempo son apreciablemente mayores que cero como la aceleración media. De igual manera a lo que sucedía con la velocidad media, es preciso definir una magnitud que exprese la aceleración instantánea. Tal definición se logra tomando como aceleración instantánea la siguiente expresión:

$$\vec{a} = \lim_{\Delta t \to 0} \frac{\Delta \vec{v}}{\Delta t} \tag{1.4}$$

La misma ecuación anterior puede ser representada mediante la notación de puntos, que representa la primera derivada de la velocidad o segunda derivada de la posición con respecto al tiempo:

$$a = \dot{v}(t) = \ddot{x}(t)$$

1.2.4 SEGUNDA LEY DE NEWTON

La segunda ley de Newton, conocida también como ley de la fuerza, en su forma más simple se define mediante la siguiente ecuación:

$$\vec{F} = m * \vec{a} \tag{1.5}$$

De la anterior ecuación se deben aclarar ciertos puntos:

- La fuerza que se hace mención en la ecuación, no se refiere únicamente a una sola, sino al sumatorio de todas las que estén interactuando con el cuerpo, por lo tanto se la denomina fuerza neta.
- Dado que la ecuación tiene un carácter vectorial, el análisis se hace mucho más sencillo si se considera un sistema de coordenadas apropiadamente seleccionado y sobre él se realiza una descomposición de las fuerzas, así la aplicación de la segunda ley de Newton se la realizaría sobre cada dimensión por separado.

1.2.5 DESCRIPCIÓN DE LA PLANTA

En esta sección se presentará un breve análisis de las ecuaciones físicas que intervienen en el modelo a usar en el presente proyecto, al que denominaremos "PLANTA" y que es mostrado en la Figura 1.1. Estas ecuaciones físicas se obtendrán a partir de los principios de cinemática y dinámica brevemente descritos anteriormente.

El análisis comenzará extrayendo un diagrama del cuerpo libre de cada uno de los cuerpos que intervienen en el movimiento analizado. De esta forma se obtendrán los diagramas mostrados en la Figura 1.2.

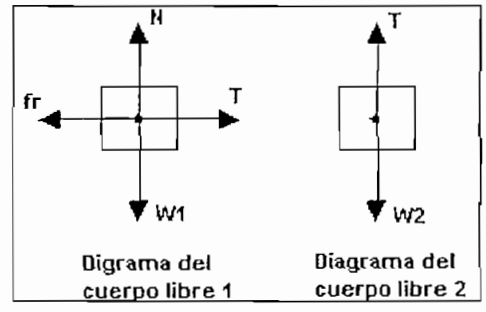


Figura 1.2 Diagramas del cuerpo libre para la planta

Después de una elección adecuada del sistema de coordenadas a usar y de la aplicación de la segunda ley de Newton en cada uno de estos ejes de tal sistema coordenado mencionado, se deducen las siguientes ecuaciones:

$$T - f_r = m_1 * \ddot{x} \quad \textit{Segunda ley de Newton eje horizontal cuerpo 1} \qquad (1.6)$$

$$N - W_1 = 0 \quad \textit{Segunda ley de Newton eje vertical cuerpo 1} \qquad (1.7)$$

$$W_2 - T = m_2 * \ddot{x} \quad \textit{Segunda ley de Newton eje vertical cuerpo 2} \qquad (1.8)$$

$$f_r = \mu * N \quad \textit{Definición de la fuerza de rozamiento o fricción} \qquad (1.9)$$

Antes de deducir la aceleración hay que aclarar primeramente que ésta se ha representado con la notación de dos puntos en las anteriores ecuaciones y que aquella es la misma tanto para el cuerpo 1 como para el cuerpo 2 y, por tal razón, está presente en las ecuaciones de ambos. Además μ representa el coeficiente de rozamiento dinámico.

De las ecuaciones anteriormente expuestas y tomando en consideración la planta física que se muestra en la Figura 1.1, las expresiones que describen el desplazamiento del cuerpo W1 en función del cuerpo W2 son como sigue:

$$Con \quad rozamiento \qquad \ddot{x} = \frac{(m_2 - \mu * m_1) * g}{m_1 + m_2} \qquad Si\ \mu \neq 0 \qquad (1.10)$$

$$Sin \quad rozamiento \qquad \ddot{x} = \frac{m_2 * g}{m_1 + m_2} \qquad Si\ \mu = 0 \qquad (1.11)$$

De las expresiones anteriores, cabe mencionar que la aceleración del cuerpo, depende principalmente de las masas de ambos cuerpos y, en el caso de existir rozamiento, depende de éste.

1.3 ULTRASONIDO

Durante los últimos años la tecnología ultrasónica se ha desarrollado a tal punto que se considera una importante rama de la física. El ultrasonido no es más que vibraciones(ondas) en un medio material similar a las ondas sonoras, pero su frecuencia es demasiada elevada para su percepción por el oído humano, pues ésta tiene un rango que va desde los 20kHz hasta el orden de los MHz.

1.3.1 PROPAGACIÓN DEL ULTRASONIDO

El ultrasonido tiene una forma de emisión cónica lobular. Cuando éste choca con una superficie, varias cosas pueden pasar, incluyendo:

- **Transmisión.**- El sonido atraviesa completamente una superficie.

- **Absorción.**- La superficie absorbe el sonido, como una esponja absorbe el agua.
- **Reflexión.**- El sonido choca contra la superficie y este cambia de dirección.
- **Difusión.**- El sonido golpea la superficie y este se esparce en varias direcciones.

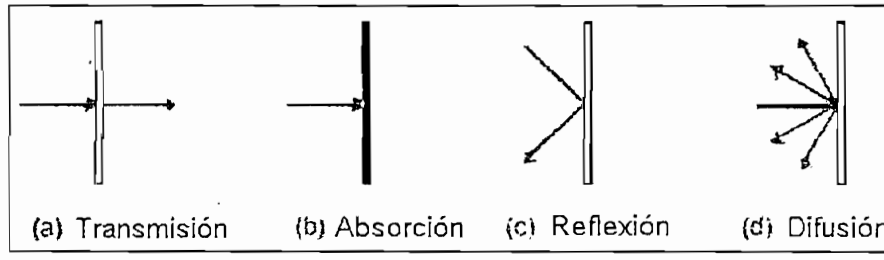


Figura 1.3 Formas de Propagación del Ultrasonido

El ultrasonido no solo se transmite en el aire, sino en cualquier otro tipo de material, sea sólido, liquido o gaseoso, pero no se puede propagar en el vacío. La velocidad con la que se propaga depende del material que sirve como medio de trasporte además de la temperatura; por lo que se estima que la velocidad del sonido tiene un valor de 331m/s a 0°C en el aire seco, aumentando éste en 0.62 m/s por grado de temperatura.

1.3.2 IMPEDANCIA ACÚSTICA

La oposición que presentan las partículas a ser desplazadas por el sonido, es la impedancia acústica de un material. El límite entre dos materiales de diferentes impedancias acústicas es llamado interfaz acústica. Cuando el sonido llega a una interfaz acústica con una incidencia normal, una cantidad de energía es reflejada y otra parte es transmitida. El ultrasonido es atenuado a medida que pasa a través de un material, asumiendo que no existen reflexiones provenientes de discontinuidades. Existen tres causas de atenuación: difracción, difusión y absorción. La cantidad de atenuación, dentro de un material, puede jugar un papel muy importante en la selección de un transductor en una determinada aplicación.

1.3.3 SENSORES DE ULTRASONIDO

Son el medio por el cual la energía eléctrica se convierte en energía mecánica (ondas sonoras) o viceversa. Opera principalmente debido a efectos como el piezoeléctrico o el electrostático.

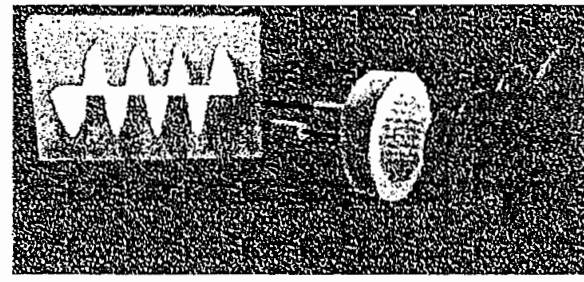


Figura 1.4 Sensor de Ultrasonido

1.3.3.1 Efecto Piezoeléctrico

Este efecto consiste, en que ciertos cristales[1] cuando se deforman, se polarizan eléctricamente y generan voltaje eléctrico entre las superficies opuestas. Esto es reversible en el sentido de que al aplicar un voltaje a través las caras de un cristal, se produce una deformación del mismo. Este efecto microscópico se origina por las propiedades de simetría de algunos cristales

1.3.3.2 Efecto Electrostático

Consiste en aplicar una diferencia de potencial constante a través de dos láminas metálicas paralelas, las mismas que experimentan entre sí una fuerza de atracción. Si se superpone un voltaje alterno de amplitud inferior a esta diferencia de potencial constante, la fuerza de atracción varía sinusoidalmente con la misma frecuencia que la del voltaje alterno aplicado. La sensibilidad puede mejorar insertando entre las láminas un dieléctrico sólido. Este efecto es reversible y casi similar al piezoeléctrico, por lo que se lo utiliza tanto en transmisores como receptores.

[1] Cuarzo. blenda, turmalina, titanio de bario, etc

1.3.4 PARÁMETROS A CONSIDERAR EN UN SISTEMA ULTRASÓNICO

Los principales parámetros que deben ser considerados al implementar un sistema ultrasónico son:

- **Frecuencia central.-** La frecuencia a la cual la respuesta de los transductores al ultrasonido es óptima.
- **Resolución axial.-** Es la capacidad del sistema para distinguir entre dos objetos que están juntos a lo largo de la línea de incidencia del rayo. Esta propiedad esta directamente relacionada con la longitud de onda ultrasónica del rayo emitido. La longitud de onda λ y la frecuencia f de una onda de ultrasonido se relacionan a través de la velocidad de propagación c mediante la siguiente ecuación:

$$\lambda = \frac{c}{f} \tag{1.12}$$

- **Atenuación del haz.-** Es la pérdida de energía de una onda ultrasónica al desplazarse a través de un material. Las causas principales son la difusión y la absorción.
- **Modelo de emisión sónico (beam pattern) .-** Una de las principales características de los sensores de ultrasonido es el modelo de emisión sónico (beem pattern); éste indica el área que abarca la onda sónica expansiva.

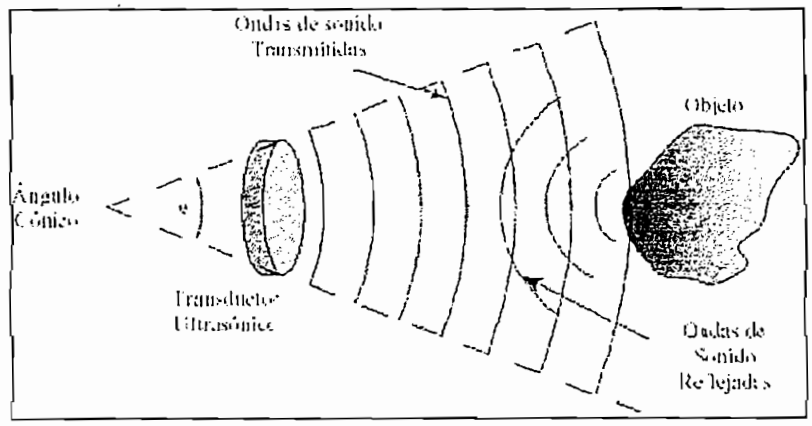


Figura 1.5 Modelo de Emisión

Dicha información resulta de mucha utilidad para determinar ciertas condiciones y características del lugar donde debe ubicarse el objeto para determinar su distancia.

1.3.5 DETERMINACIÓN DE LA DISTANCIA MEDIANTE EL ULTRASONIDO

La finalidad principal de este proyecto radica en la obtención numérica de la distancia entre un objeto móvil y un punto fijo. Esta medición se realizará utilizando como base la emisión y recepción de las ondas sónicas a través de transductores de ultrasonido.

Para esto se utiliza el principio pulso–eco o también conocido como tiempo de vuelo. Con el tiempo transcurrido entre emisión y recepción, considerando la velocidad del sonido constante en el aire (341m/s), se calcula la distancia con los obstáculos mediante la siguiente ecuación:

$$d = v * \frac{T}{2} \tag{1.13}$$

Como se muestra en la expresión anterior, el tiempo que se toma en cuenta en el cálculo de la distancia es la mitad del transcurrido entre emisión y recepción (T/2) pues la onda se refleja en el objeto y regresa para ser detectada, la Figura 1.6 lo grafica.

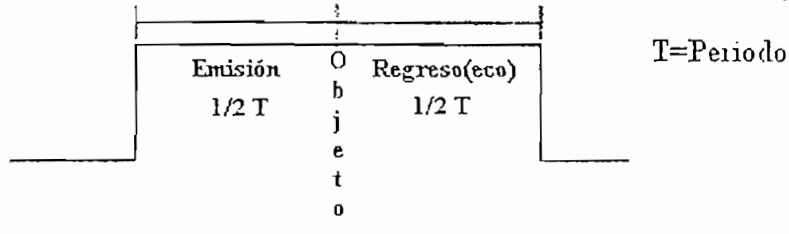


Figura 1.6 Pulso-Eco

En resumen, midiendo el tiempo de vuelo desde un punto fijo hasta el objeto en movimiento será posible determinar su posición con respecto al tiempo.

1.4. ASPECTOS GENERALES DEL PUERTO USB

En la actualidad resulta muy interesante observar como los avances tecnológicos sorprenden por la evolución tan rápida que presentan. Otra característica importante es que cada vez son más fáciles de usar por cualquier persona, gracias a que son desarrollados de tal forma que no se necesita ser un experto para comprender su funcionamiento básico, usarlos o instalarlos, como es el caso del bus serie universal conocido como USB. Pero, qué acerca de cómo controlarlos para emplearlos en aplicaciones propias?

Esto último constituye un verdadero reto para el ingeniero que intenta comprender el interior de estos dispositivos. En este capítulo se pretende determinar las complejidades del funcionamiento del pórtico USB con el objetivo de adecuar su poder en aplicaciones de adquisición de datos y control.

El USB es una interfaz rápida, fiable y flexible para conectar dispositivos a los computadores; es un estándar abierto que ya ha sido adoptado por cientos de fabricantes de periféricos, y ha recibido una gran aceptación entre los fabricantes de computadores personales.

La llegada del USB viene a simplificar la conexión de periféricos a los PCs. Sus principales ventajas residen en la capacidad de proporcionar una verdadera interfaz *Plug&Play a* los periféricos que se conectan al PC, no haciendo falta su configuración para que funcionen correctamente.

El USB se basa en el protocolo denominado paso de testigo. Un controlador USB distribuye un testigo por el bus. El dispositivo cuya dirección coincide con la que porta el testigo responde aceptando o enviando datos al controlador. Este también gestiona la distribución de energía a los periféricos que lo requieran. Un dispositivo USB puede usar cualquiera de cuatro tipos de transferencias existentes y una sus tres velocidades de transferencia de datos, esto de acuerdo a sus requerimientos de funcionalidad.

Las especificaciones del USB siguen cambiando, y los miembros del USB van por la revisión 2.0 de las mismas. La única desventaja que tiene el USB es desde el punto de vista de un diseñador; el protocolo del USB es realmente complejo. Sin embargo, esta desventaja se compensa por la velocidad, flexibilidad y fiabilidad del USB, y rasgos adicionales que ofrece, como la corrección de errores.

1.4.1 BENEFICIOS DEL USB

Una de las características relevantes del pórtico USB es su versatilidad de usarse con muchas clases de periféricos, en lugar de tener diferentes tipos de conectores y hardware de soporte para cada uno de ellos. Una PC típica tiene al menos dos puertos USB, pudiendo expandírselos con la conexión de un hub USB a un puerto existente.

Se puede conectar y desconectar un periférico siempre que se desee, pudiendo estar el PC o el periférico encendido o no, sin posible daño para alguno de los dos, esto debido a que el sistema operativo está continuamente detectando cuando un dispositivo es agregado o removido del bus. La interfaz USB incluye líneas de alimentación y tierra que proveen +5V desde la PC o desde del hub. Un periférico que requiere hasta 500mA puede drenar toda su potencia desde el bus en lugar de tener su propia fuente de poder.

La especificación 1.1 del USB emitida alrededor del año 1998 presentaba dos tipos de velocidad: baja velocidad (low – speed) que trabaja a 1.5 Megabits por segundo y mediana velocidad (full – speed) que trabaja a 12 Megabits por segundo. En el 2001, cuando se emitió la versión 2.0 de la especificación USB, se presentó un nuevo tipo de velocidad denominada alta velocidad (high – speed). que trabaja a 480 Megabits por segundo.

Cada PC que tiene capacidad USB, soporta baja y mediana velocidad. En cambio alta velocidad fue adherida en la versión 2.0 de la especificación USB y requiere hardware capaz de soportar USB 2.0 en su motherboard o en una tarjeta de expansión.

Por lo tanto, un dispositivo que cumpla con la versión USB 1.x puede conectarse a un computador con soporte 2.0 sin problema, en tanto que, un dispositivo 2.0 que maneje únicamente alta velocidad no puede conectarse en un host que tenga soporte solo para la versión USB 1.x, porque en ésta no está definida la alta velocidad.

En el presente trabajo, no se hará uso de alta velocidad (high – speed), por algunas razones que a continuación se describirán:

- Debido a que alta velocidad es propio de la versión 2.0 de la especificación, no todas las PCs tendrán soporte para ésta; por tal motivo, el modulo experimental que se pretende implementar usará mediana o baja velocidad, con lo cual se tendrá la capacidad de conectarse con un computador host que tenga soporte USB 2.0 o soporte USB 1.x, mejorando notablemente la versatilidad de este trabajo.
- La alta velocidad fue introducida para realizar transferencias de grandes bloques de información a altas velocidades, como por ejemplo para aplicaciones de captura de imágenes, de audio en tiempo real, comunicaciones con modems, etc; ahora bien, como en este trabajo no se necesita que se transfieran grandes bloques de información a alta velocidad se descartó ésta posibilidad, es así que para conseguir los objetivos finales del proyecto baja o mediana velocidad serían suficientes.
- Una última razón por la cual no se pretende hacer uso de alta velocidad se debe a que al hacer un análisis del problema planteado se obtuvo que los tiempos requeridos para la adquisición de datos no eran tan críticos como para usar esa alta velocidad.

El nivel de soporte puede variar. En el nivel fundamental, un sistema operativo que soporta USB debe hacer tres cosas:

- Detectar cuando un dispositivo es agregado o removido del sistema.

- Comunicarse con dispositivos agregados recientemente para encontrar como intercambiar datos con ellos.
- Proveer un mecanismo que habilite a los drivers (software) comunicarse con el hardware USB del computador host y las aplicaciones que deseen acceder a los periféricos USB.

En un nivel más alto, el soporte del sistema operativo puede significar también la inclusión de drivers de dispositivo, para habilitar a los programadores de aplicaciones, acceder a dispositivos llamando a funciones que reciban soporte del sistema operativo. Si el sistema operativo no incluye un driver de dispositivo apropiado para un periférico específico, el vendedor del periférico debe proveer uno.

Del lado del periférico, cada hardware del dispositivo USB debe incluir un chip controlador que maneje los detalles de la comunicación USB. Algunos controladores son microcomputadores completos que incluyen un CPU y memoria para almacenar código específico del dispositivo que corre dentro del periférico. Otros manejan únicamente tareas específicas del USB, con un bus de datos que conecta a otro microcontrolador que desempeña funciones no relacionadas con el USB y se comunica con el controlador USB como sea necesario.

1.4.2 DESVENTAJAS DEL USB

El USB fue diseñado como un bus de escritorio (ejemplo: teclados, ratones, scaners, cámaras, etc). De allí que el cable de conexión del periférico puede ser de máximo 5 metros. Se puede incrementar la longitud del enlace USB hasta 30 metros usando cables que enlacen cinco hubs USB y un dispositivo. Para extender el rango más allá de esto, una opción es usar una interfaz USB en el PC, que convierta a RS – 485 o a otra interfaz para unir a más larga distancia el periférico. Los periféricos no pueden hablar directamente entre si. Todas las comunicaciones son hacia o desde el computador host.

Para programar un periférico USB, se necesita conocer del protocolo del USB. Los chips controladores manejan automáticamente mucho de las comunicaciones, pero todavía deben ser programados, y esto requiere el conocimiento para escribir los programas y las herramientas para programarlo. En el lado del PC, el driver del dispositivo aísla a los programadores de aplicaciones del conocimiento de muchos detalles, pero los escritores de drivers de dispositivo necesitan familiarizarse con el protocolo USB y las responsabilidades de los drivers.

1.5 TOPOLOGÍA DEL BUS

Hay cuatro partes principales para la topología USB:

- Host y dispositivos.- Los componentes primarios del sistema USB.
- Topología física.- Como son conectados los elementos de un sistema USB.
- Topología Lógica.- Las reglas y responsabilidades de varios elementos USB y como el USB aparece desde la perspectiva del host y un dispositivo.
- Relación software cliente - dispositivo.- Como el software cliente y sus funciones relacionadas interactúan sobre el dispositivo USB sin importar la perspectiva de otros dispositivos conectados al mismo bus.

1.5.1 TOPOLOGÍA FÍSICA DE BUS

Los dispositivos sobre el USB son conectados físicamente al host por medio de una topología en estrella escalonada, como se ilustra en la Figura 1.7. Los puntos USB agregados son provistos por hubs. Estos puntos son llamados puertos. Un host incluye un hub incrustado llamado el hub raíz, que provee uno o más puntos de agregación. Los dispositivos USB que proveen funcionalidad adicional al host son conocidos como funciones, y todos comparten la misma ruta hacia el computador host; por lo que solamente un dispositivo puede comunicarse con el host a la vez. Para prevenir agregaciones circulares, un ordenamiento escalonado es impuesto sobre la topología estrella del USB, pudiéndose colocar un máximo de cinco hubs en cascada; hasta un total de 127 periféricos y hubs incluyendo el hub raíz, como se muestra en la Figura 1.7

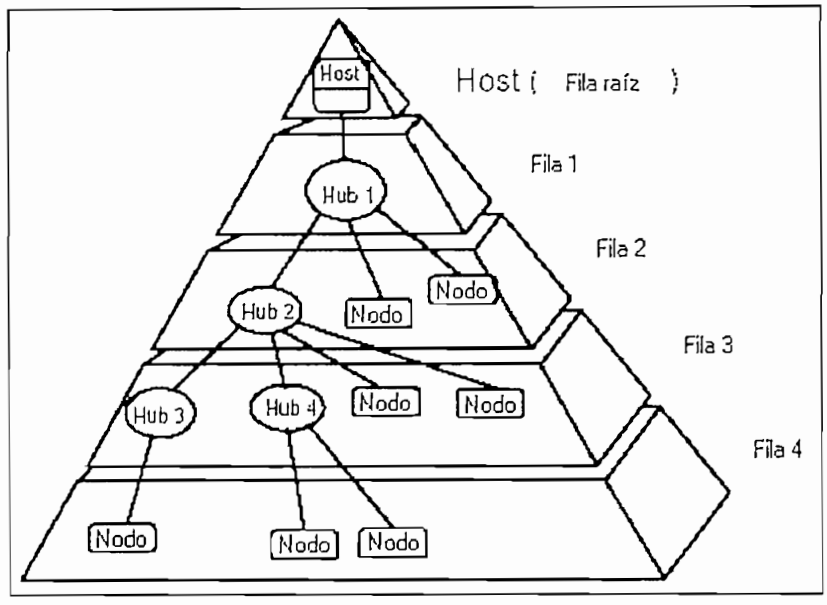


Figura 1.7 Topología de bus

1.5.2 TOPOLOGÍA LÓGICA DEL BUS

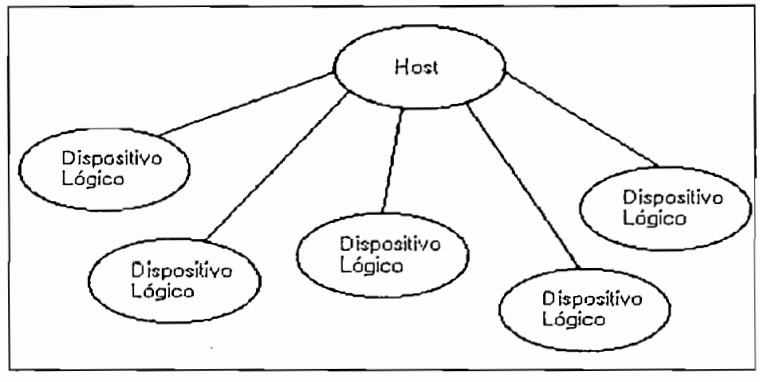


Figura 1.8 Topología lógica

Mientras los dispositivos físicos agregados al USB están en estrella escalonada, el host se comunica con cada dispositivo lógico como si ellos estuvieran conectados directamente al puerto raíz.

Esto crea la vista lógica ilustrada en la Figura 1.8. Los hubs también son dispositivos lógicos, pero no se muestran en la Figura 1.8 para simplificar el cuadro.

1.5.3 RELACIÓN SOFTWARE CLIENTE -A- FUNCIÓN

Aunque la topología física y lógica del USB refleja la naturaleza compartida del bus, el software cliente (CSw) manipula un dispositivo USB con la perspectiva de que aquel trata solamente con su interfaz de interés. Durante la operación, el software cliente debería ser independiente de otros dispositivos que pueden ser conectados al USB. Esto permite al diseñador del dispositivo y del software - cliente enfocarse en los detalles del diseño de la interacción entre hardware y software. La Figura 1.9 ilustra las relaciones del software - cliente y función USB desde la perspectiva del diseñador del dispositivo con respecto a la topología lógica de la Figura 1.8.

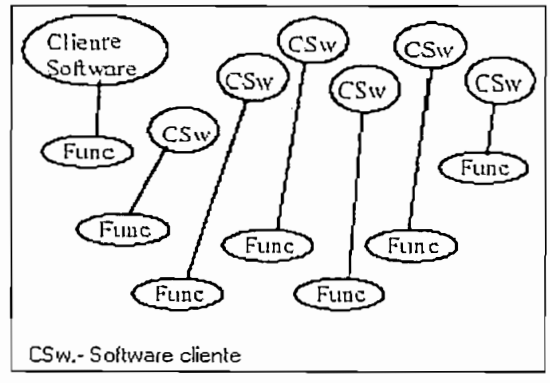


Figura 1.9 Relación software cliente a función

1.6 ESPECIFICACIÓN ELÉCTRICA DEL USB

Una característica muy conveniente del USB es la habilidad que tiene de proveer potencia a los dispositivos que estén conectados al bus. Esta tiene varias ventajas; por ejemplo, elimina la necesidad de una fuente externa y hace al dispositivo pequeño y ligero.

1.6.1 VOLTAJES

Los voltajes nominales entre los puntos de conexión: VBUS y GND en un cable USB es 5 voltios, aunque dependiendo del tipo de dispositivo se permite un rango de variación, como se aprecia en la Tabla 1.1.

Dispositivo	Voltaje mínimo	Voltaje máximo
Alta Potencia	4.75	5.25
Baja Potencia	4.4	5.25

Tabla 1.1 Rango de variación de voltaje disponible en el puerto USB

Debido a pérdidas en los cables y otras más, un dispositivo debería ser capaz de funcionar con fuentes de voltaje de unas pocas décimas de voltios menos que el mínimo disponible en el puerto. El USB ha hecho una clasificación de los dispositivos de acuerdo al origen de su fuente de alimentación.

1.6.2 DISPOSITIVOS ALIMENTADOS POR EL BUS (BUS – POWERED)

Son todos los dispositivos (incluyendo hubs) que drenan potencia desde el bus USB. La máxima potencia que se puede drenar del bus es de 500 mA.

1.6.3 DISPOSITIVOS AUTO – ALIMENTADOS (SELF – POWERED)

Son dispositivos (incluyendo hubs) que no consumen potencia desde el bus USB. Un dispositivo de estas características tiene su propia fuente de potencia.

Este tipo de clasificación no solo incluye a dispositivos que necesitan corrientes superiores a los 500 miliamperios, sino a dispositivos que necesitan operar aún cuando no estén conectados al bus USB. Sin embargo un dispositivo auto – alimentado debe tener una conexión a VBus, para detectar la presencia de la potencia del bus, incluso si éste no la necesita.

Durante la enumeración (proceso en el que el host configura al dispositivo), el host aprende si el dispositivo es auto – alimentado o alimentado desde el bus y la máxima corriente que el dispositivo drenará del bus. Existe un estado del USB denominado "Suspendido" que asegura que un dispositivo no consume potencia del bus cuando el host no tiene necesidad de comunicarse con éste. Un dispositivo entra en este estado cuando no hay actividad en el bus por al menos 10 milisegundos, o cuando el host envía un pedido de estado suspendido a los dispositivos del hub.

1.7 COMPONENTES FÍSICOS DEL SISTEMA USB

Los componentes físicos del Bus Serie Universal consisten de circuitos, conectores y cables entre un host y uno o más dispositivos.

1.7.1 DISPOSITIVOS

Los dispositivos físicos USB proveen funcionalidad adicional al host. En bajo nivel un dispositivo puede referirse a un simple componente de hardware como una memoria. En alto nivel un dispositivo puede referirse a una colección de componentes de hardware que realizan una función particular, por ejemplo un Data/Fax MODEM, hub, etc.

Para asistir al host en identificación y configuración de dispositivos USB, cada dispositivo lleva y reporta información relacionada con la configuración. Otra información es específica con respecto a la funcionalidad provista por el dispositivo. El formato detallado de esta información varía y depende de la clase del dispositivo.

1.7.2 CABLES

El cable USB consiste de 4 hilos o conductores, como muestra la Figura 1.10, blindado para transmisiones a 480 Mbps o 12 Mbps y no blindado para transmisiones a 1.5 Mbps.

De los cuatro conductores que lleva el cable, un par son trenzados y están destinados para la transmisión de datos: las líneas D_+ y D_-, en tanto que los otros dos no son trenzados y están destinados para alimentación V_{BUS} y GND.

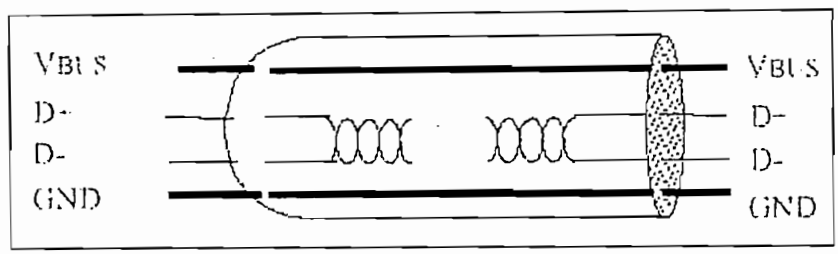


Figura 1.10 Composición del cable USB

El calibre de los conductores destinados a alimentación de los periféricos varía desde 20 a 26 AWG, mientras que los conductores de señal son de 28 AWG.

1.7.3 CONECTORES

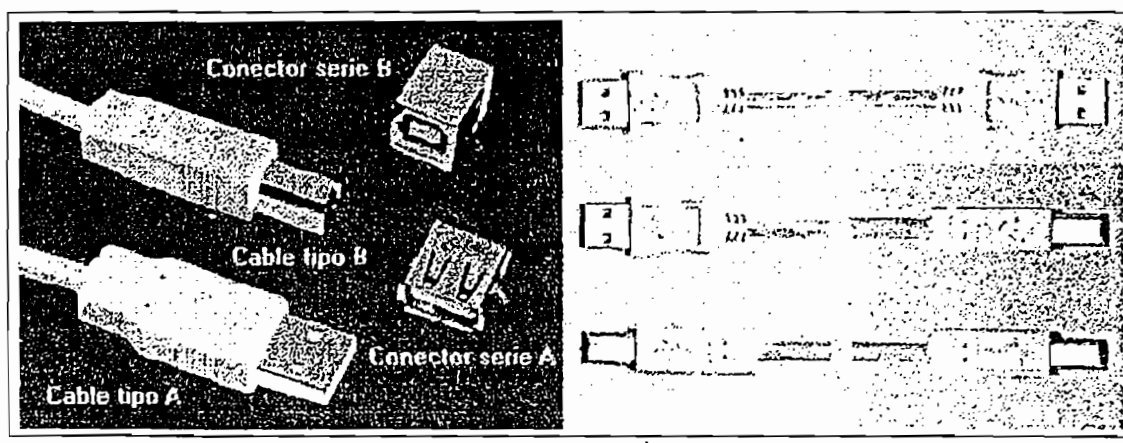


Figura 1.11 Cables y conectores USB

En cuanto a conectores, la especificación los define básicamente como conector (macho) y receptáculo (hembra) de dos tipos: serie A y serie B. El conector serie A es bastante común dentro de los dispositivos listos para ser empleados con el host, y lo más probable es que tengan su propio cable con su conector serie A, como por ejemplo ratones, teclados, etc.

Los conectores de la serie B se emplean en los dispositivos USB que no poseen cable incorporado, para los cuales el conector serie B será una característica, como por ejemplo impresoras, scanners, y módems. Debido a que ambos conectores son físicamente diferentes una inserción equivocada es imposible incluso por la forma de las ranuras. La Figura 1.11 muestra los diferentes tipos de conectores USB, y las respectivas ranuras.

Los pines del conector se identifican a continuación en la Tabla 1.2:

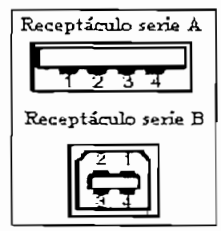
Conector	Pin	Señal	Color
	1	+5V	Rojo
	2	D-	Blanco
	3	D+	Verde
	4	Tierra	Negro

Tabla 1.2 Identificación de pines en los conectores USB

1.8 ANÁLISIS BÁSICO DE LAS TRANSFERENCIAS USB

Se puede dividir las comunicaciones USB en dos categorías, dependiendo de si son usadas en la configuración de un dispositivo o en el traslado de información propia del dispositivo para una aplicación.

En la configuración, el host aprende acerca del dispositivo y lo prepara para intercambiar datos. Muchas comunicaciones toman lugar cuando el host enumera el dispositivo en la alimentación o agregación del mismo.

Las comunicaciones de aplicación ocurren cuando el host intercambia datos para uso de las aplicaciones. Estas son las comunicaciones que desempeñan las funciones para las que el dispositivo fue diseñado.

1.8.1 COMUNICACIONES DE CONFIGURACIÓN

Durante la enumeración el firmware del dispositivo responde a una serie de pedidos estándar desde el host. El dispositivo debe identificar cada pedido retornando la información requerida, y otra acción especificada por el pedido.

En el PC, Windows realiza la enumeración, así no se requiere intervención del usuario. Sin embargo, para completar la enumeración, Windows debe tener dos archivos disponibles: un archivo INF que identifica el nombre y localización del driver del dispositivo y, por supuesto, el driver del dispositivo. Si los archivos están disponibles y el firmware trabaja correctamente, el proceso de enumeración es invisible para el usuario.

Dependiendo del dispositivo y de como será usado, el driver del dispositivo puede ser uno que esté incluido en Windows o uno que deba proveer el vendedor del producto. El archivo INF es un archivo de texto con un formato especifico.

1.8.2 COMUNICACIONES DE APLICACIÓN

Después de que el host ha intercambiado información con el dispositivo en el proceso de enumeración y un driver del dispositivo ha sido asignado y cargado, las comunicaciones de aplicación pueden empezar. En el host, las aplicaciones pueden usar funciones API de Windows para leer y escribir al dispositivo. En el dispositivo, la transferencia de datos requiere típicamente colocar los datos a enviar en el buffer de transmisión del controlador USB del dispositivo; lectura de datos recibidos desde el buffer receptor, y en la conclusión de una transferencia, asegurarse que el dispositivo está listo para la siguiente transferencia.

1.8.3 MANEJO DE DATOS EN EL BUS

Dos líneas de señal del USB llevan datos hacia y desde todos los dispositivos enlazados al bus. Estas forman un solo camino de transmisión que todos los dispositivos deben compartir, los mismos que llevan una sola señal diferencial.

El host está encargado de supervisar que todas las transferencias ocurran tan rápido como sea posible, manejando el tráfico por medio de dividir el tiempo en tramas (frames) o microtramas (microframes) en alta velocidad. El host da a cada transferencia una porción de la trama o de la microtrama como lo muestra la Figura 1.12. Para datos en baja y mediana velocidad las tramas son de un milisegundo.

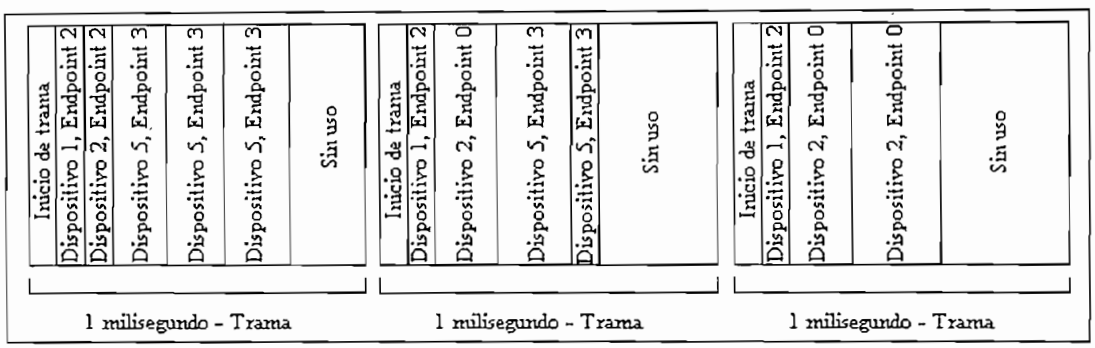


Figura 1.12 Esquema del tráfico USB

Para datos en alta velocidad el host divide cada trama en ocho microtramas de 125 microsegundos. Cada trama o microtrama empieza con un sincronizador de referencia denominado inicio de trama SOF (Start – of – frame).

Cada transferencia consiste de una o más transacciones. Las transferencias de control siempre se componen de múltiples etapas, consistiendo cada una de ellas por una o más transacciones. Otras transferencias usan múltiples transacciones cuando tienen muchos datos que no entran en una sola transacción.

Dependiendo de cómo el host programa las transacciones y la velocidad de respuesta del dispositivo, una transacción puede estar toda ella en una sola trama o microtrama o ellas pueden ser repartidas sobre múltiples (micro) tramas.

Debido a que todas las transferencias comparten la misma ruta de datos, cada transacción debe incluir una dirección de dispositivo. Cada dispositivo tiene asignada una dirección única por el host, y todos los datos viajan hacia o desde el host. Cada transacción empieza cuando el host envía un bloque de información que incluye la dirección del dispositivo receptor y una localización específica llamada "endpoint" la misma que se encuentra dentro del dispositivo. Todo lo que el dispositivo envíe, lo hace en respuesta a la petición recibida desde el host para enviar información de datos o estados.

1.8.3.1 Codificación de la señal

El USB es un bus de comunicación serial con señal diferencial llevada por medio de dos conductores usando el formato de codificación NRZI (Non- Return to Zero Inverted) con bit stuffing[1]. Este formato en lugar de definir 0s y 1s lógicos como voltajes, define cada 0 lógico como un cambio de voltaje del estado previo y cada 1 lógico no produce ningún cambio en el voltaje; la Figura 1.13 muestra un ejemplo.

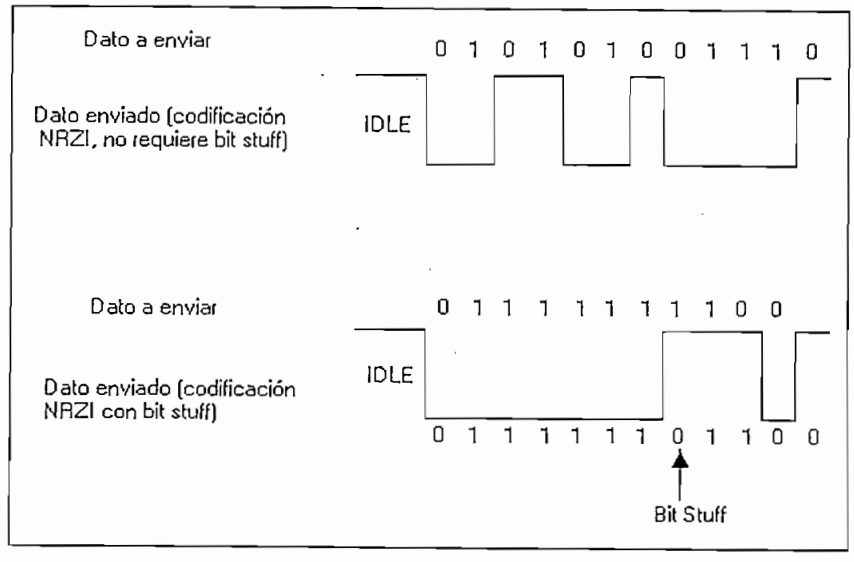


Figura 1.13 Codificación NZRI de datos

[1] **Bit stuffing.**- Consiste en insertar un bit 0 para forzar un cambio de estado cuando hay una secuencia superior a 6 bits 1 y así conseguir que el receptor esté sincronizado al transmisor.

NRZI asegura que el receptor permanezca sincronizado con el transmisor sin la necesidad de enviar una cabecera, una señal de reloj separada, o bits de inicio y parada con cada byte. Los bits son transmitidos desde el menos significativo (LSB).

1.8.4 ELEMENTOS DE UNA TRANSFERENCIA

Cada transferencia está compuesta de transacciones, las mismas que se componen de paquetes; y cada paquete contiene información. Para entender transacciones, paquetes y sus contenidos, también se necesita conocer acerca de los endpoints y los "pipes".

1.8.4.1 Endpoints de dispositivos

El endpoint es un buffer o bloque de memoria que almacena múltiples bytes en el dispositivo. Los datos almacenados en un endpoint pueden ser datos recibidos o datos esperando a ser transmitidos. Todas las transmisiones viajan hacia y desde un endpoint del dispositivo.

El host también tiene buffers para datos recibidos y para datos listos a transmitir, pero a éstos no se les denomina endpoints, sino que sirven como un punto de inicio para comunicarse con los endpoints de los dispositivos.

La especificación define a un endpoint como parte de un dispositivo USB que tiene una dirección única, el mismo que puede ser un receptor o una fuente de información para un flujo de comunicación entre el host y el dispositivo. Esto sugiere que un endpoint lleve datos en una sola dirección, sin embargo un endpoint de control[1] es un caso especial pues el flujo de comunicación es bidireccional. La única dirección requerida para cada endpoint consiste de un número de endpoint y la dirección del flujo de datos (IN u OUT).

[1] **Endpoint de control.**- Endpoint usado en las transferencias de control.

El número de endpoint puede estar en un rango de 0 a 15; en tanto que la dirección del flujo de datos se asigna desde la perspectiva del host: IN se refiere a un flujo de comunicación que va desde el dispositivo hacia el host mientras que, OUT se refiere a un flujo de comunicación que va desde el host hacia el dispositivo.

Cada dispositivo debe tener un endpoint 0 configurado como un endpoint de control que acepte flujos de comunicaciones en ambas direcciones. Los otros tipos de transferencias envían datos en una sola dirección.

Un dispositivo de mediana velocidad puede tener hasta 30 endpoints (del 1 al 15 cada uno soportando direcciones de flujo IN y OUT) aparte del endpoint 0. Un dispositivo de baja velocidad es limitado a dos endpoints adicionales con cualquier combinación de direcciones de flujo.

Cada transacción en el bus incluye un número de endpoint y un código que indica la dirección del flujo de datos y si la transacción está o no inicializando una transferencia de control. Los códigos son IN, OUT y SETUP, como se observa en la Tabla 1.3.

Tipo de transacción	Fuente de datos	Tipo de transferencia que usa este tipo de transacción	Contenido
IN	Dispositivo	Todas	Datos genéricos
OUT	Host	Todas	Datos genéricos
Setup	Host	Control	Un pedido

Tabla 1.3 Tipos de transacción

Como con las direcciones de endpoint, la convención de nombres para transacciones IN y OUT está dada desde la perspectiva del host.

En una transacción Setup los datos también viajan desde el host hacia el periférico, pero una transacción Setup es un caso especial porque inicializa una transferencia de control.

Los dispositivos necesitan identificar las transacciones Setup para saber como interpretar los datos que éstas contengan. Cualquier transferencia puede usar transacciones IN u OUT, pero solamente la trasferencia de control usa las transacciones Setup.

Cada transacción contiene una dirección de dispositivo y una dirección de endpoint. Cuando el dispositivo recibe una transacción OUT o Setup conteniendo la dirección del dispositivo, el hardware almacena los datos recibidos en una localidad apropiada del endpoint. Cuando un dispositivo recibe una transacción IN conteniendo la dirección del dispositivo, si el dispositivo tiene datos listos para su envío, el hardware envía los datos desde el endpoint especificado.

1.8.4.2 Pipes

Antes de que una transferencia ocurra, el host y el dispositivo deben establecer un "pipe". Un "pipe" no es un objeto físico, sino que es una asociación entre el endpoint del dispositivo y el software del controlador del host.

El host establece los "pipes" brevemente después de que el sistema se ha encendido o un dispositivo se ha agregado, pidiendo información de configuración desde el dispositivo. El host puede pedir nuevos "pipes" o remover los "pipes" innecesarios. Cada dispositivo tiene un "pipe" de control predeterminado el mismo que usa el Endpoint 0.

1.8.4.2.1 Pipes de flujo (stream) y de mensaje (message)

Además de la clasificación del "pipe" de acuerdo al tipo de transferencia que este lleva, la especificación también define los "pipes" como de tipo flujo o mensaje (stream o message), de acuerdo a si la información viaja en una o ambas direcciones.

La transferencia de control es la única transferencia que usa "pipes" tipo mensaje (message) bidireccionales; todas las otras usan "pipes" tipo flujo (stream) unidireccionales.

En un "pipe" tipo mensaje, cada transferencia empieza con una transacción setup conteniendo un pedido. En una "pipe" de flujo, los datos no tienen un formato definido por la especificación USB. El dispositivo receptor acepta todo lo que se le envía.

1.8.4.3 Inicializando una transferencia

Cuando el driver del dispositivo quiere comunicarse con su dispositivo, este inicializa una transferencia. La especificación define a una transferencia como el proceso de hacer y llevar un pedido de comunicación.

Típicamente una aplicación en Windows abre la comunicación con un dispositivo, usando un recurso (handle) recuperado por medio de una función API estándar. Luego la aplicación hace uso de este recurso llamando a una función API que requiera la transferencia desde el driver del dispositivo.

Cuando una aplicación requiere transferir, el sistema operativo pasa el pedido desde la aplicación al driver del dispositivo apropiado, el cual entrega el pedido a los drivers de más bajo nivel; es decir, a los drivers del sistema USB[1] y los drivers del controlador del host[2]. El controlador host entonces inicia la transferencia en el bus.

En algunos casos, el driver es configurado para pedir transferencias periódicas, y la aplicación lee los datos recuperados o provee datos para ser enviados en esta transferencia. Otras transferencias, tales como aquellas que se hacen en la enumeración son inicializadas por el sistema operativo al detectar el dispositivo.

---

[1] Drivers que manejan transacciones USB y el proceso de enumeración
[2] Drivers que manejan la comunicación con el hardware.

1.8.4.4 Bloques de una transferencia

En la terminología usada en el USB existen términos que son muy similares y confusos, en tal virtud, en la Tabla 1.4 se presenta un compendio de tales términos que ayudarán a entenderlos mejor, para así evitar confusiones.

Tipo de transferencia	Etapas (1 o más transacciones)	Fases (paquetes)
Control	Setup	Token
		Datos
		Handshake
	Datos (IN u OUT) opcional	Token
		Datos
		Handshake
	Estado (IN u OUT)	Token
		Datos
		Handshake
Bulk	Datos (IN u OUT)	Token
		Datos
		Handshake
Interrupción	Datos (IN u OUT)	Token
		Datos
		Handshake
Isocrónico	Datos (IN u OUT)	Token
		Datos

Tabla 1.4 Etapas y fases en una transferencia

Cada transferencia consiste de una o mas transacciones y a la vez cada transacción consiste de uno, dos, o tres paquetes, como se puede apreciar en la Figura 1.14. Cada una de las cuatro tipos de transferencia consiste de una o mas etapas y cada etapa está hecha de dos o tres fases como lo muestra la Tabla 1.4.

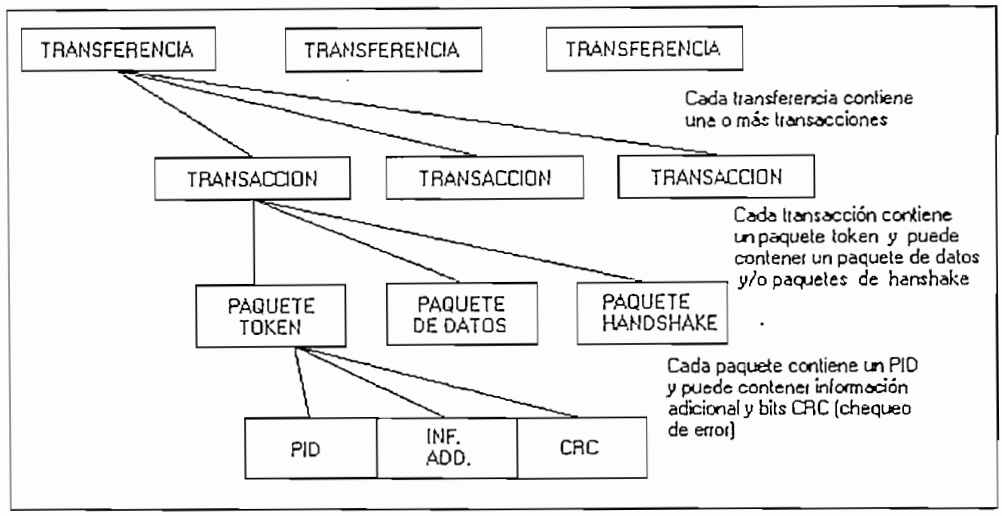


Figura 1.14 Elemento de una transferencia

Los tres tipos de transacciones están definidas por sus propósitos y por la dirección del flujo de datos: Setup es utilizado para enviar transferencias de control requeridas para el dispositivo, IN para recibir datos desde un dispositivo, y OUT para el envío de datos hacia un dispositivo. Una transferencia completa toma lugar sobre múltiples tramas, pero una transacción es una simple comunicación que debe completarse ininterrumpidamente. Otras comunicaciones no pueden partir una transacción.

1.8.4.5 Fases de una transacción

Cada transacción puede tener hasta tres fases que ocurren en secuencia: token, datos y handshake. Cada fase consiste de uno o dos paquetes transmitidos. Cada paquete es un bloque de información con un formato definido. Todos los paquetes empiezan con un ID de paquete (PID) que contiene información de su identificación. Dependiendo de la transacción, el PID puede ser seguido por una dirección de endpoint, datos, información de estado, o un número de trama, junto con bits de chequeo de error.

En la fase Token de una transacción el host envía un pedido de comunicación en un paquete denominado Token. El PID de este paquete indica el tipo de transacción tal como Setup, IN, OUT, o inicio de trama[1].

En la fase de datos el host o el dispositivo pueden transferir cualquier clase de información en el paquete de datos. El PID indica el valor de secuencia de paquete llamado data – toggle cuando existen múltiples paquetes.

En la fase de Handshake el host o el dispositivo envían información de estado o de Handshake en un paquete de Handshake. El PID contiene códigos de estado como ACK, NAK, STALL, NYET.

1.8.5 CÓDIGOS DE HANDSHAKE

Para asegurar que cada transacción ha tenido éxito el USB usa Handshake y chequeo de error para ayudar a manejar el flujo de datos. USB usa Handshake en software, el mismo que es un código que indica el éxito o la falla de una transacción excepto en la transferencia isocrónica que no maneja códigos de Handshake. Muchas de las señales de Handshake se transmiten en el paquete de Handshake, aunque algunas usan el paquete de datos. A continuación se describe con más detalle cada uno de estos errores.

1.8.5.1 Ack

Reconocimiento (acknowledge) que indica que el host o el dispositivo han recibido datos sin errores.

1.8.5.2 Nak

Reconocimiento negativo (negative acknowledge), significa que un dispositivo está ocupado o no tiene datos para retornar.

---

[1] SOF.- Sincronizador de referencia que el host envía cada milisegundo en mediana velocidad, y 125 microsegundos en alta velocidad.

1.8.5.3 Stall.

El Handshake STALL puede tener tres significados: No soporta el pedido de Control, el pedido de control ha fallado, o el endpoint ha fallado.

1.8.5.4 NYet (not yet)

Solamente usado por dispositivos que manejen alta velocidad y en transferencias de tipo bulk o control. Después que el endpoint de un dispositivo ha recibido un paquete de datos, éste puede regresar un Handshake NYET para indicar que los datos fueron aceptados pero que el endpoint no está listo para recibir otro paquete de datos.

1.8.5.5 Sin respuesta

Esto ocurre cuando el host o el dispositivo esperan recibir un Handshake, pero esto no ocurre. Esto usualmente indica que el cálculo en el CRC[1] del receptor ha detectado un error e informa al transmisor que este debería probar otra vez o si múltiples intentos han fallado, debería tomar otra acción.

1.9 TIPOS DE TRANSFERENCIAS USB

La especificación USB define cuatro tipos de transferencias: control, bulk, interrupción, e isocrónica. Los cuatro tipos de transferencias de datos manejan diferentes necesidades, y un dispositivo puede soportar los tipos de transferencias que son más apropiados para su propósito. A continuación, se hará un análisis de cada tipo de transferencia.

[1] **Código de redundancia cíclica.**- algoritmo matemático usado para detección de errores.

1.9.1 TRANSFERENCIAS DE CONTROL

Una transferencia de control tiene dos usos: lleva los requerimientos que son definidos por la especificación USB y usadas por el host para conocer y configurar los dispositivos.

1.9.1.1 Disponibilidad

Todo dispositivo USB debe soportar transferencias de control sobre el "pipe" predeterminado en el Endpoint 0. Además puede tener también "pipes" adicionales configurados para transferencias de control, pero en realidad no se necesita más que uno.

1.9.1.2 Estructura

Las transferencias de control usan una estructura definida con dos o tres etapas: Setup, Datos (opcional) y Estado. La etapa de datos es opcional.

En la etapa Setup, el host empieza una transacción Setup enviando información acerca del pedido. El paquete Token contiene un PID que identifica la transferencia, como una transferencia de control. El paquete de datos contiene información acerca del pedido, información de si la transferencia tiene o no una etapa de datos y, si es así, en que dirección viajaran los datos.

Cuando la etapa de datos está presente, ésta consiste de una o más transacciones IN u OUT, también llamadas transacciones de datos. Dependiendo del pedido, el host o el periférico puede ser la fuente de estas transacciones, pero todos los paquetes de datos en esta o cualquier etapa deben estar en la misma dirección.

La etapa de estado consiste de una transacción IN u OUT, también llamada transacción de estado. En la etapa de estado, el dispositivo reporta el éxito o la falla de la etapa previa.

Cuando no hay etapa de datos, el dispositivo envía un paquete de datos en la etapa de estado. Los paquetes de datos o Handshake enviados por el dispositivo en la etapa de estado contiene un código que indica el éxito o fracaso de las etapas Setup y datos de la transferencia.

1.9.1.3 Tamaño de los datos

El máximo tamaño del paquete en la etapa de datos varía con la velocidad de dispositivo. Para dispositivos de baja velocidad, el máximo es de 8 bytes. Para dispositivos de mediana velocidad, el máximo puede ser de 8, 16, 32 o 64 bytes. Para dispositivos de alta velocidad, el máximo debe ser de 64 bytes. Estos bytes incluye solamente la información transferida en el paquete de datos, excluyendo el PID y los bits CRC.

1.9.1.4 Velocidad

El Host debe hacer su mejor esfuerzo para asegurar que una transferencia de control se realice lo más pronto posible. El controlador del host reserva una porción del ancho de banda del bus para las transferencias de control: 10 por ciento para baja y mediana velocidad y 20 por ciento para alta velocidad. Si el bus tiene ancho de banda disponible, las transferencias de control pueden usar mas ancho de banda que la cantidad reservada para ellas.

En las transferencias de control, además de transmitir datos de configuración también se puede transferir datos de propósito general y, aunque la especificación no lo prohíbe, el reservarlas para atender peticiones USB estándar, ayuda a asegurar que la transferencia se complete rápidamente y mantenga el ancho de banda reservado. Las transferencias de control no son la vía más eficiente para transferir datos. En baja velocidad, el ancho de banda reservado requiere 3 tramas para completar una transferencia de 8 bytes.

En mediana velocidad, el ancho de banda reservado puede llevar una transferencia de 64 bytes por trama (aunque cualquier transferencia puede ser difundida sobre múltiples tramas); y en alta velocidad el ancho de banda puede llevar 6 transferencias de 64 bytes por microtrama, o 512 bytes por trama.

1.9.2 TRANSFERENCIAS BULK

Este tipo de transferencia es útil para transferir datos en los que el tiempo no es una variable crítica. Una transferencia bulk puede enviar una gran cantidad de datos sin obstruir el bus, debido a que estas transferencias esperan hasta que tengan tiempo disponible para transmitir, de este modo si el bus está desocupado las transferencias bulk son las más rápidas.

1.9.2.1 Disponibilidad

Solamente dispositivos que manejen mediana y alta velocidad pueden realizar este tipo de transferencias. Un dispositivo no requiere soportar este tipo de transferencia, aunque una clase específica de dispositivo puede requerirla.

1.9.2.2 Estructura

Una transferencia bulk consiste de una o más transacciones IN u OUT. Transferir datos en ambas direcciones requiere "pipes" separados y transferencias para cada dirección. Una transferencia bulk termina en una de dos formas: Cuando la cantidad de datos pedidos han sido transferidos, o cuando el paquete de datos contiene menos datos que el máximo especificado.

1.9.2.3 Tamaño de los datos

Una transferencia bulk a mediana velocidad puede tener un paquete de tamaño máximo de 8, 16, 32, o 64 bytes. Para alta velocidad el máximo debe ser de 512 bytes.

El host lee el tamaño del paquete máximo para cada "pipe" bulk durante la enumeración del dispositivo. La cantidad de datos en una transferencia puede ser menor igual o mayor que el tamaño máximo del paquete.

1.9.2.4 Velocidad

El controlador del host garantiza que la transferencia bulk eventualmente se completará, pero no reserva algún ancho de banda para las transferencias. Las transferencias de control están garantizadas que tendrán 10% de ancho de banda para baja y mediana velocidad y 20% para alta velocidad. Las transferencias de interrupción e isocrónicas pueden usar el resto. Así si el bus está muy ocupado una transferencia bulk puede ser muy larga. Sin embargo cuando el bus está libre, la transferencia bulk puede usar el mayor ancho de banda que cualquier otro tipo, y como tiene una cabecera muy pequeña, son las más rápidas de todas. En mediana velocidad cuando el bus está libre, 19 transferencias bulk de 64 bytes pueden transferir hasta 1216 bytes de datos por trama, para una frecuencia de datos de 1.216 Megabytes por segundo. Esto deja disponible para otros usos el 18% del ancho de banda del bus. En alta velocidad en un bus desocupado, hasta 13 transferencias bulk de 512 bytes pueden transferir hasta 6656 bytes de datos por microtrama, para una impresionante frecuencia de 53.248 Megabytes por segundo, dejando disponible el 2% del ancho de banda del bus.

1.9.3 TRANSFERENCIA DE INTERRUPCIÓN

Las transferencias de interrupción son útiles cuando los datos tienen que transferirse dentro de una cantidad de tiempo específica. Aplicaciones típicas incluyen teclados, ratones, joysticks, envío de reportes de estado de un hub, y pequeñas unidades de adquisición de datos que entren en una clase específica denominada HID[1]. Dispositivos de baja velocidad los cuales soportan únicamente transferencias de control e interrupción, probablemente usarán transferencias de interrupción para datos genéricos.

[1] **HID** (Dispositivo de interfaz humana).- Clase especial de dispositivos que son de uso genérico como por ejemplo mouse, teclados, scanners de códigos de barras, unidades de adquisición de datos.

Las transferencias de interrupción son populares también, porque Windows incluye drivers que habilitan a las aplicaciones hacer transferencias de interrupción con dispositivos que formen parte de la especificación HID. El nombre transferencia de interrupción sugiere que un dispositivo puede usar un hardware de interrupción que implique una rápida respuesta del PC. Pero la verdad es que las transferencias de interrupción, como las otras transferencias USB, ocurren solamente cuando el host realiza un polling al dispositivo. Las transferencias son como interrupciones, debido a que ellas garantizan que el host pedirá o enviará datos con un mínimo retardo.

1.9.3.1 Disponibilidad

Todas las tres velocidades soportan transferencias de interrupción. Los dispositivos no requieren soportar este tipo de transferencia pero una clase de dispositivo puede requerirla. Por ejemplo un dispositivo de clase HID debe soportar transferencias de interrupción IN para enviar datos al host.

1.9.3.2 Estructura

Una transferencia de interrupción consiste de una o más transacciones IN o una o más transacciones OUT. La estructura de una transferencia de interrupción es idéntica a una transferencia bulk, la única diferencia estriba en la programación. Una transferencia de interrupción es de una sola vía; es decir las transacciones deben ser todas transacciones IN o todas transacciones OUT. Transferir datos en ambas direcciones requiere transferencias y "pipes" separados para cada dirección.

1.9.3.3 Tamaño de los datos

Para dispositivos de baja velocidad, el máximo tamaño del paquete puede estar entre 1 y 8 bytes, para mediana velocidad el máximo tamaño del paquete puede estar en un rango de 1 a 64 bytes, y para alta velocidad el rango es de 1 a 1024 bytes.

1.9.3.4 Velocidad

Una transferencia de interrupción garantiza una latencia (tiempo entre transacciones) máxima. En otras palabras, no hay una frecuencia de transferencia (transacciones por segundo) garantizada. Transferencias de interrupción en alta velocidad pueden ser muy rápidas. Una transferencia en alta velocidad puede pedir hasta tres paquetes de 1024 bytes cada microtrama, con lo cual se trabaja a 24.576 Megabytes por segundo. Un endpoint que requiera más de 1024 bytes por microtrama, es un endpoint de alto ancho de banda, una transferencia de mediana velocidad puede pedir hasta 64 bytes cada trama, o 64 Kilobytes por segundo, y una transferencia a baja velocidad puede pedir hasta 8 bytes cada 10 milisegundos (10 tramas) u 800 bytes por segundo.

El descriptor de endpoint almacenado en el dispositivo almacena la máxima latencia. Para dispositivos de baja velocidad, la máxima latencia puede tener un valor entre 10 y 255 milisegundos, para mediana velocidad esta puede estar entre 1 y 255 milisegundos, y para alta velocidad el rango se extiende desde 125 microsegundos hasta 4 segundos en incrementos de 125 microsegundos.

1.9.4 TRANSFERENCIAS ISOCRÓNICAS

Las transferencias isocrónicas usan transferencias en tiempo real que son útiles cuando los datos deben llegar a una frecuencia constante, o para un tiempo específico, y pueden ser tolerados errores ocasionales. En mediana velocidad, las transferencias isocrónicas pueden transferir más datos por trama que las transferencias de interrupción, pero no hay forma de prever retransmisión de datos con error. Ejemplos de uso de transferencias isocrónicas incluyen codificación de voz y música para reproducirlas en tiempo real, pero datos que eventualmente serán usados a una frecuencia constante no necesariamente requieren de una transferencia isocrónica. A diferencia de las transferencias bulk, una vez que una transferencia isocrónica empieza, el host garantiza que el tiempo estará disponible para enviar datos a una frecuencia constante.

1.9.4.1 Disponibilidad

Solamente dispositivos de mediana y alta velocidad pueden realizar este tipo de transferencias. Dispositivos no requieren soportar transferencias isocrónicas, aunque una clase específica de dispositivo puede requerirlas.

1.9.4.2 Estructura

Isócrono significa que los datos tienen una frecuencia de transferencia fija, transfiriendo un número de bytes definido en cada trama o microtrama. Ninguno de los otros tipos de transferencias garantizan enviar un número específico de bytes en cada trama (con la excepción de la transferencia de interrupción con la menor máxima latencia posible). Una transferencia isocrónica es de un solo sentido; las transacciones en una transferencia deben ser todas transacciones IN o todas transacciones OUT. Transferir datos en ambas direcciones requiere de "pipes" y transferencias separados para cada dirección. Antes de configurar un "pipe" para transferencias isocrónicas, el controlador del host compara el tamaño del buffer requerido con el actual disponible, para determinar si el ancho de banda pedido puede ser otorgado a la transferencia.

1.9.4.3 Tamaño de los datos

Para un endpoint de mediana velocidad, el tamaño máximo del paquete puede tener un rango que va desde 0 a 1023 bytes de datos, los endpoints de alta velocidad pueden tener un tamaño máximo de paquete de hasta 1024 bytes.

1.9.4.4 Velocidad

Una transacción isocrónica en mediana velocidad puede transferir hasta 1023 bytes por trama, o hasta 1.023 Megabytes por segundo. Esto deja 31% del ancho de banda del bus disponible para otros usos. El mínimo ancho de banda requerido por una transferencia de mediana velocidad es de 1 byte por trama, o un kilobyte por segundo.

Una transacción isocrónica en alta velocidad puede transferir hasta 1024 bytes por trama. Un endpoint isocrónico que requiere más de 1024 bytes por microtrama puede requerir dos o tres transacciones por microtrama, para una máxima frecuencia de 24.576 Megabytes por segundo. Un endpoint que requiere múltiples transacciones por microtrama es un endpoint de alto ancho de banda.

En un bus de alta velocidad desocupado se puede llevar dos transferencias isocrónicas en la frecuencia máxima. El precio a pagar para garantizar la entrega a tiempo de grandes bloques de datos es un protocolo sin corrección de errores. Transferencias isocrónicas están diseñadas para uso donde errores pequeños y ocasionales son aceptables; y en realidad, bajo circunstancias normales, una transferencia USB debería experimentar no más que un error ocasional debido a ruido en las líneas.

1.10 PROCESO DE ENUMERACIÓN

Antes que una aplicación pueda comunicarse con un dispositivo, el host necesita aprender acerca del dispositivo y asignar un driver para el dispositivo. La enumeración es el intercambio inicial de información que cumpla con este proceso. El proceso incluye asignar una dirección al dispositivo, leyendo estructuras de datos desde el dispositivo, asignando y cargando un driver de dispositivo, y seleccionando una configuración desde las opciones presentadas en los datos recuperados.

Para aprender de un nuevo dispositivo, el host envía una serie de pedidos al dispositivo o al hub del dispositivo, causando que el hub establezca una ruta de comunicación entre el host y el dispositivo. El host entonces intenta enumerar el dispositivo enviando transferencias de control conteniendo pedidos USB estándar dirigidos al endpoint 0. Para tener éxito en el proceso de enumeración, el dispositivo debe responder a cada petición retornando la información pedida y tomando otras acciones requeridas.

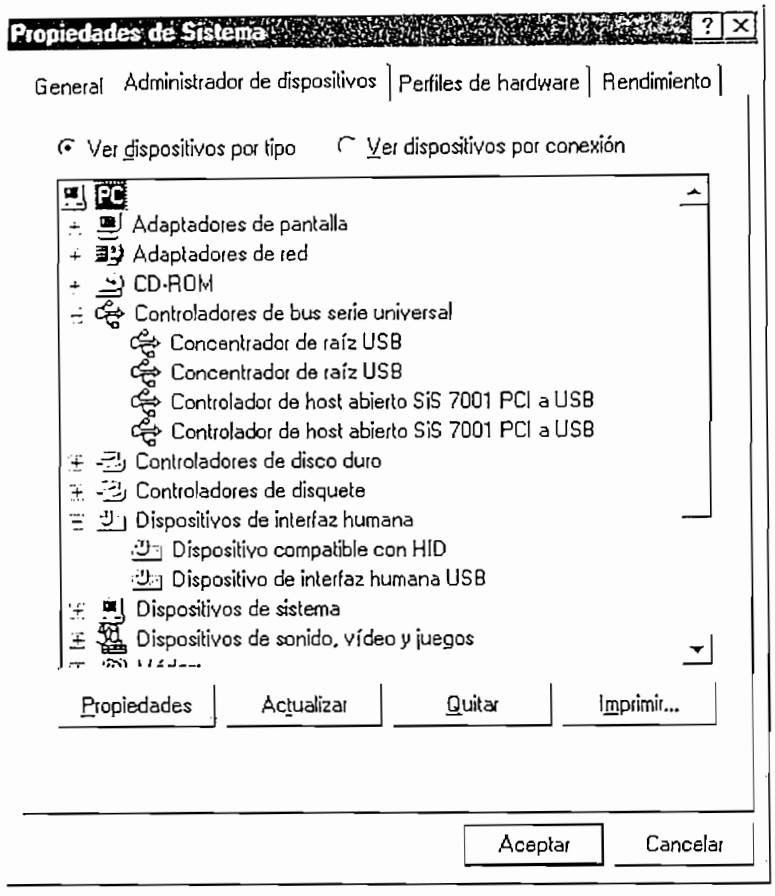


Figura 1.15 Dispositivo enumerado en el Administrador de Dispositivos (Device Manager)

Desde la perspectiva del usuario, la enumeración debería ser invisible y automática, excepto por posibles ventanas que anuncien la detección de un nuevo dispositivo y si el intento de configurar el dispositivo ha tenido éxito o no. Algunas veces al agregar un dispositivo por primera vez, el usuario necesita proveer un disco conteniendo el archivo INF[1] y el driver del dispositivo. Cuando la enumeración esta completa, Windows añade el nuevo dispositivo al Administrador de Dispositivos (Device Manager) de Windows desplegado en el Control Panel del PC.

[1] Archivo de texto que identifica el driver adecuado para uso del dispositivo.

La Figura 1.15 muestra un ejemplo en la que al USB está agregado un dispositivo HID (Dispositivo de interfaz humana), además de mostrarse agregados el hub raíz con los puertos incorporados al PC como otro dispositivo USB. En un periférico típico, el código de programa contiene la información que el host requiere, y una combinación de hardware y firmware decodifica y responde a pedidos para la configuración. Por otro lado, bajo Windows no hay necesidad de escribir código para la enumeración, porque Windows lo maneja automáticamente.

1.10.1 PASOS DE ENUMERACIÓN

Durante el proceso de enumeración, un dispositivo se mueve a través de cuatro de los seis estados del dispositivo definidos por la especificación: estado alimentado, estado predeterminado (default), estado de dirección y estado configurado; los otros estados son agregado y suspendido. El firmware del dispositivo no debería asumir que los pedidos de enumeración y eventos ocurren en un orden particular; por el contrario el dispositivo debería estar listo para detectar y responder a cualquier pedido en cualquier tiempo.

1.- El usuario inserta un dispositivo en un puerto USB o el sistema es alimentado con un dispositivo ya conectado a un puerto. El dispositivo está en el estado alimentado.

2.- El hub o hub raiz detecta al dispositivo monitoreando el voltaje en las líneas de señal de cada uno de sus puertos. El hub tiene un resistor pull – down de 15 Kohm en cada una de las dos líneas de señal del puerto (D_+ y D_-), mientras que un dispositivo tiene un resistor pull – up de 1.5 Kohm en la línea D_+ para un dispositivo que maneje mediana velocidad o en la línea D_- para un dispositivo de baja velocidad.

Dispositivos que manejen alta velocidad se agregan a mediana velocidad. Cuando un dispositivo se conecta a un puerto, el pull – up del dispositivo habilita al hub detectar que aquel dispositivo ha sido agregado.

3.- El host aprende del nuevo dispositivo por medio de un evento reportado por el hub al cual el dispositivo está agregado.

4.- El hub detecta si un dispositivo es de baja o mediana velocidad antes de que el hub resetee el puerto USB del dispositivo. El hub detecta la velocidad de un dispositivo determinando cual línea de voltaje está en alto, revisándola cuando el bus está desocupado.

5.- El hub reseta el dispositivo. El controlador del host envía al hub un pedido Setear_Característica_Del_Puerto (Set_Port_Feature) que le indica resetear el puerto. El hub coloca las líneas de datos USB del dispositivo en la condición de reset por al menos 10 milisegundos.

El reset es una condición especial donde ambas líneas de señal (D_+ y D_-) están en voltaje bajo. Normalmente las líneas tienen estados lógicos opuestos. El hub envía el reset únicamente al nuevo dispositivo, en tanto que los otros hubs o dispositivos agregados al bus no lo ven.

6.- El host aprende si un dispositivo de mediana velocidad maneja alta velocidad, usando dos estado especiales de señal. Todo dispositivo de alta velocidad debe ser capaz de responder a pedidos de enumeración en mediana velocidad.

7.- El hub establece una ruta entre el dispositivo y el bus. El host verifica que el dispositivo ha salido del estado de reset enviando un pedido Conseguir_Estado_Del_Puerto (Get_Port_Status). Un bit en los datos retornados indican si el dispositivo está todavía en el estado de reset; y si es necesario, el host repite el pedido hasta que el dispositivo haya salido de este estado.

Cuando un hub remueve el reset, el dispositivo entra al estado predeterminado (default). Los registros USB del dispositivo están en su estado de reset y el dispositivo está listo para responder a transferencias de control sobre el "pipe" predeterminada (default pipe) en el endpoint 0. El dispositivo puede ahora comunicarse con el host, usando la dirección 00h.

8.- El host envía un pedido Conseguir_Descriptor (Get_Descriptor) para aprender el tamaño máximo del paquete de datos que el "pipe" predeterminado (default pipe) puede usar. El host envía este pedido a la dirección 00h, y al endpoint 0. Debido a que el host enumera un solo dispositivo a la vez, únicamente responderá aquel dispositivo con la dirección 00h, aún si varios dispositivos se agregan a la misma vez. En esta parte el host pide al hub resetear nuevamente al dispositivo.

9.- El host asigna una única dirección al dispositivo enviando un pedido Setear_Pedido (Set_Request). El dispositivo lee el pedido y almacena la nueva dirección. El dispositivo está ahora en el estado de dirección. Toda comunicación subsiguiente usará la nueva dirección. La dirección es válida hasta que el dispositivo es removido del bus, es reseteado o el sistema se apaga. En la siguiente enumeración, al dispositivo se le puede asignar una dirección diferente.

10.- El host aprende acerca de la habilidades del dispositivo enviando un pedido Conseguir_Descriptor (Get_Descriptor) a la nueva dirección para leer tal descriptor, esta vez leyéndolo todo.

11.- El host asigna y carga un driver del dispositivo. Una vez que el host ha aprendido tanto como pudo acerca del dispositivo desde sus descriptores, observa por el mejor driver del dispositivo que maneje las comunicaciones con el mismo.

Para seleccionar el driver, Windows prueba emparejar la información almacenada en los archivos INF del sistema con el ID del vendedor, ID del producto y, número de versión recuperada desde el dispositivo. Si no lo hay, Windows mira para emparejarlo con una clase, subclase y valores de protocolo recuperado desde el dispositivo.

Después que el sistema operativo asigna y carga el driver, el driver a menudo requiere que el dispositivo reenvíe los descriptores u otros descriptores específicos de clase.

12.- El driver del dispositivo en el host selecciona una configuración, enviando un pedido Setear_Configuración (Set_Configuration) con el número de configuración deseada. Muchos dispositivos soportan únicamente una configuración. Si un dispositivo usa múltiples configuraciones, el driver puede decidir cual usar basado en toda la información que tenga acerca de cómo el dispositivo se usará o, puede preguntar al usuario que hacer o, únicamente seleccionar la primera configuración.

El dispositivo lee el pedido y setea su configuración para emparejarla. El dispositivo está ahora en el estado configurado y la(s) interfaz(s) del dispositivo está(n) habilitadas. El dispositivo está listo ya para usarlo. Los otros dos estados, agregado y suspendido pueden existir en cualquier instante.

1.10.2 REMOCIÓN DEL DISPOSITIVO

Cuando un usuario remueve un dispositivo del bus, el hub deshabilita el puerto del dispositivo. El host sabe de la remoción después que realiza un polling al hub, y enviando un pedido Conseguir_Estado_Del_Puerto (Get_Port_Status) para aprender que evento ocurrió. Windows remueve el dispositivo del Administrador de Dispositivos (Device Manager) y la dirección del dispositivo queda disponible para cualquier otro dispositivo que sea agregado posteriormente.

1.10.3 TIPOS DE DESCRIPTORES

Los descriptores son estructuras de datos o bloques de información, que habilitan al host a aprender acerca del dispositivo. Todo periférico USB debe responder a pedidos de descriptores estándar USB. Esto quiere decir que el periférico debe hacer dos cosas: almacenar la información en los descriptores y responder a los pedidos para los descriptores en el formato esperado.

1.10.3.1 Tipos

En la Tabla 1.5 se enlista los tipos de descriptores. Cada dispositivo tiene uno y solamente un descriptor de dispositivo que contiene información acerca de aquel como un todo y especifica el número de configuraciones que él soporta. Cada dispositivo tiene uno o más descriptores de configuración que contiene información referente al uso de potencia del mismo y el número de interfaces soportadas por la configuración.

Tipo de descriptor	Descriptor requerido?
Descriptor de dispositivo	Si
Descriptor calificador de dispositivo	Si, para dispositivos que soportan mediana y alta velocidad. No permitido para otros dispositivos.
Descriptor de configuración	Si
Descriptor de otra configuración de velocidad	Si, para dispositivo que soportan mediana y alta velocidad. No permitido para otros dispositivos.
Descriptor de interfaz	Si
Descriptor de endpoint	No, si el dispositivo usa solo endpoint 0
Descriptor de texto	No. Descripción opcional de texto
Descriptor de potencia de la interfaz	No. Soporta manejo de potencia a nivel de interfaz

Tabla 1.5 Tipos de descriptores

Cada descriptor de interfaz tiene cero o más descriptores de endpoint que contienen la información necesaria para comunicarse con un endpoint. Una interfaz sin descriptores de interfaz puede usar aún el endpoint de control para comunicarse.

Al recibir un pedido por un descriptor de configuración, el dispositivo debería regresar el descriptor de configuración y todo descriptor de interfaz, de endpoint y cualquier otro descriptor subordinado perteneciente a la configuración, hasta el número pedido de bytes. No hay pedido específico para recuperar, por ejemplo, únicamente un descriptor de endpoint. Un descriptor de texto (string descriptor) puede almacenar texto tal como el nombre del dispositivo o el vendedor. Los otros descriptores pueden almacenar índices que apunten a estos descriptores de texto, y el host puede leer el descriptor de texto (string descriptor) usando el pedido Conseguir_Descriptor (Get_Descriptor).

Además de los descriptores estándar, un dispositivo puede contener descriptores específicos de clase o vendedor, los cuales ofrecen información más detallada acerca de si mismo. Por ejemplo, un descriptor de interfaz puede especificar que la interfaz pertenece a la clase HID y soporta un descriptor de clase HID. Cada descriptor contiene un valor que identifica el tipo de descriptor. La tabla 1.6 enumera los valores definidos por la especificación USB y HID.

Tipo	Valor (Hexadecimal)	Descriptor
Estándar	01	Dispositivo
	02	Configuración
	03	Texto
	04	Interfaz
	05	Endpoint
	06	Calificador de dispositivo
	07	Otra configuración de velocidad
	08	Potencia de la interfaz
Clase	21	HID
	29	Hub
Específico a la clase HID	22	Reporte
	23	Físico

Tabla 1.6 Constantes para los descriptores

Cada descriptor consiste de una serie de campos. Muchos de los nombres de los campos usan prefijos para indicar algo acerca del formato o contenido de los datos en el campo; por ejemplo: b = byte (8 bits), w = word (16 bits), bm = bit mapeado, bcd = código decimal binario, i= índice, id= identificador. Si se desea conocer más acerca de los tipos de descriptores y el contenido de sus campos revisar el Anexo A.

1.11 COMO SE COMUNICA EL HOST CON EL DISPOSITIVO

Bajo Windows, cualquier comunicación con un dispositivo USB debe pasar a través de un driver de dispositivo que conozca como comunicar los drivers del sistema USB con la aplicación que accede al dispositivo.

1.11.1 CONCEPTOS BÁSICOS SOBRE EL DRIVER DE DISPOSITIVO

Un driver del dispositivo es un componente de software que habilita a las aplicaciones acceder al hardware de un dispositivo, el mismo que debe conocer como comunicarse con los drives del bus de bajo nivel que controlan el hardware. Algunos drivers del dispositivo son drivers de clase que manejan comunicaciones con una variedad de dispositivos que tienen funciones similares. Es así, que los drivers cumplen con la misión de traducir entre el código a nivel de aplicación y el código a nivel de hardware.

El código a nivel de aplicación usa funciones soportadas por el sistema operativo para comunicarse con los drivers del dispositivo. El código específico de hardware maneja el protocolo necesario para acceder a los circuitos del periférico, detección de las señales de estado y cambiar las señales de control al tiempo apropiado. Windows incluye funciones API[1] que habilitan a las aplicaciones escritas por ejemplo en: Visual Basic, C/C++, Delphi, etc, poder comunicarse con los drivers del dispositivo.

---

1 API.-funciones de interfaz para programadores de aplicación, que son parte del subsistema Win32 de Windows.

Por ejemplo tres funciones que un driver de dispositivo puede dar soporte para leer y escribir a un dispositivo USB son: ReadFile, WriteFile y DeviceIoControl; vale aclarar que, la forma en que una aplicación se comunica con un dispositivo USB varía con el driver asignado a aquel dispositivo.

1.11.2 OPCIONES PARA DRIVERS DEL DISPOSITIVO

Existen varias formas de obtener un driver para un dispositivo: algunos pueden usar un driver que esté incluido en Windows para un tipo de dispositivo estándar o se puede usar un driver provisto por un vendedor de un chip u otra fuente o para otros dispositivos puede ser necesario escribir un driver personalizado.

1.11.2.1 Tipos de dispositivos estándar

Muchos periféricos caben en una clase estándar tales como impresoras, modems, teclados, ratones. Windows incluye drivers de clase para muchos tipos de dispositivos estándar.

1.11.2.2 Dispositivos personalizados

Algunos periféricos son dispositivos personalizados hechos para usar con una aplicación específica, tales como: unidades de adquisición de datos, controladores de motores e instrumentos de prueba. Windows no tiene conocimiento de estos dispositivos, así que este no tiene construido drivers para aquellos periféricos. Dispositivos como estos pueden usar drivers personalizados, o pueden ser diseñados para que cumplan con los requerimientos para una clase cuyo driver esté disponible; como por ejemplo, lo que se realizó en el presente trabajo, considerar al proyecto como un pequeño sistema de adquisición de datos que cumpla con ciertas especificaciones para dispositivos HID, y por lo tanto usar el driver HID incorporado en las últimas versiones de Windows, eliminándose así la necesidad de escribir un driver personalizado para este dispositivo.

1.11.3 LENGUAJES DE PROGRAMACIÓN

Los programadores de aplicaciones tiene que escoger cual lenguaje de programación usar, incluyendo en estos Visual Basic, Delphi, y Visual C++. Pero para escribir drivers personalizados para un dispositivo USB se necesita una herramienta que sea capaz de compilar un driver WDM[1], y esto significa usar como requerimiento mínimo Visual C++ de Microsoft.

Más allá de este requerimiento mínimo, existen otras herramientas que pueden ayudar a desarrollar un driver personalizado, tales como el kit de desarrolladores de drivers para Windows (DDK), que se lo puede obtener de forma gratuita desde el website de Microsoft. Hay que aclarar que escribir un driver personalizado de cualquier tipo es un tema mucho más complejo y avanzado que requiere de experiencia en programación en C.

1.11.4 PROCESO DE SELECCIÓN DE UN DRIVER BAJO WINDOWS

Cuando Windows detecta un nuevo dispositivo USB, una de las cosas que debe hacer es decidir cual driver del dispositivo debería usar la aplicación para comunicarse con el dispositivo y si es necesario, cargar el driver seleccionado. Este es el trabajo del Administrador de Dispositivos (Device Manager) de Windows.

El archivo INF le dice a Windows que driver o drivers usar y que información almacenar en el registro de Windows. Cuando Windows enumera un nuevo dispositivo USB, el Administrador de Dispositivos compara los datos en todo los archivos INF del sistema con la información recuperada de los descriptores desde el dispositivo. Si este dispositivo es agregado y no tiene su propio archivo INF el Administrador de Dispositivos de Windows buscará el archivo INF más apropiado.

---

[1] **WDM (Modelo de driver Win32).**- modelo de drivers para las versiones posteriores a Windows 98 SE

1.11.5 CLASES DE DISPOSITIVOS

Muchos periféricos no son totalmente únicos, sino que comparten muchas cualidades con otros dispositivos; así, cuando un grupo de dispositivos o interfaces comparten muchos atributos o cuando proveen o piden servicios similares, toma sentido definir los atributos y servicios en una especificación de clase; la misma que sirve como una guía para quienes desarrollan dispositivos y escriben drivers del dispositivo.

Las clases ofrecen varias ventajas, y una de ellas es el hecho de que Windows y otros sistemas operativos incluyen drivers para clases comunes; y si la clase de un dispositivo es soportada por el sistema operativo no sería necesario proveer o escribir un driver para tal dispositivo, por ejemplo dispositivos de audio, dispositivos de comunicaciones, dispositivos de almacenamiento masivo de datos, impresoras, imagen, etc tienen un driver incluido con Windows o aunque el dispositivo no sea de algún tipo estándar de periférico, diseñar este para que encaje en una clase definida como por ejemplo, muchos controladores de motores y unidades de adquisición de datos que son periféricos especializados cuyo driver no se encuentra en los PCs, se podría hacer que cualquiera de ambas categorías o cualquier dispositivo que transfiera datos a moderada velocidad, pueda ser diseñado para entrar en la clase HID cuyo driver existe.

1.11.6 DISPOSITIVO DE INTERFAZ HUMANA (HID)

Pueden ser periféricos tales como teclados, ratones, joysticks, o cualquier dispositivo que transfiera bloques de información a velocidad moderada, usando transferencias de control o transferencias IN u OUT de interrupción. Un HID no tiene que ser un tipo de periférico estándar, y aun no necesita una interfaz humana. El único requerimiento es que el descriptor almacenado en el dispositivo debe cumplir los requerimientos para los descriptores de clase HID, y el dispositivo debe enviar y recibir datos usando transferencias de interrupción o control como se define en la especificación HID, misma que se la puede encontrar en la página web www.usb.org en la sección desarrolladores > HID tools.

La principal limitación para las comunicaciones con un HID es la disponibilidad de tipos de transferencias, las mismas que como ya se mencionó se reducen al uso de las transferencias de control o transferencias IN u OUT de interrupción a partir de la segunda edición de Windows 98.

El intercambio de datos bajo la clase HID reside en estructuras llamadas reportes, así el host envía y recibe datos enviando y pidiendo reportes en la transferencias ya mencionadas; además, el formato del reporte es flexible, y puede manejar cualquier tipo de datos.

1.11.6.1 Requerimientos de hardware

Una interfaz HID debe cumplir con los requerimientos de la clase HID como se define en la especificación. El documento describe los descriptores requeridos, la frecuencia de transferencia y los tipos de transferencias disponibles así como otros requerimientos específicos a los endpoints y pipes..

1.11.6.1.1 Requerimientos de endpoints

Toda transferencia HID usa el "pipe" de control predeterminado o un "pipe" de interrupción. Un HID debe tener un endpoint de interrupción IN para enviar datos al host, además, de un endpoint de interrupción OUT opcional para recibir datos desde el host.

1.11.6.1.2 Requerimiento del "pipe" de control

El "pipe" de control para un HID lleva los pedidos estándar USB además de seis pedidos específicos de clase definidos en la especificación HID. Dos de los pedidos específicos de la clase HID son el pedido Setear_Reporte (Set_Report) y Conseguir_Reporte (Get_Report) que proveen un camino para que el host y el dispositivo transfieran un bloque de cualquier clase de datos en cualquier dirección.

1.11.6.1.3 Requerimientos de transferencias de interrupción

El "pipe" o "pipes" de interrupción proveen un vía alterna para intercambiar datos, especialmente cuando el receptor debe conseguir los datos rápidamente o periódicamente. Un "pipe" de interrupción IN lleva datos hacia el host, y un "pipe" de interrupción OUT lleva datos hacia el dispositivo.

1.11.6.2 Requerimientos de firmware

El descriptor de dispositivo debe identificar al dispositivo que tiene interfaz HID, y el firmware debe soportar un endpoint de interrupción IN y además un "pipe" de control predeterminado. El firmware también contiene un descriptor de reporte que define el formato para transmitir y recibir datos del dispositivo. Para enviar datos, la especificación requiere que el firmware soporte el pedido Coseguir_Reporte (Get_Report) en una transferencia de control o transferencias de interrupción IN, y para recibir datos el firmware debe soportar el pedido Setear_reporte (Set_Report) en una transferencia de control o puede también soportar transferencias de interrupción OUT.

Todo HID debe usar un formato de reporte definido que indica el tamaño y contenido de los datos en el reporte. Un descriptor de reporte en el firmware del dispositivo describe los reportes, y puede incluir información acerca de como el receptor de los datos debería usarlo. Un valor específico en cada reporte lo define como un reporte de entrada, salida o de característica (feature). El host recibe datos en reportes de entrada, y envía datos en reportes de salida, en cambio, los reportes de característica (feature reports) viajan en cualquier dirección.

1.11.6.3 Identificando un dispositivo como HID

Como cualquier dispositivo HID, un descriptor HID le dice al host lo que necesita saber para comunicarse con el dispositivo. El host aprende acerca de la interfaz HID cuando éste envía un pedido Conseguir_Descriptor (Get_Descriptor) para obtener la configuración que contiene la Interfaz HID.

El descriptor de clase HID especifica el número de descriptores de reporte soportados por la interfaz y durante la enumeración el driver HID recupera los descriptores de clase HID y de reporte.

1.11.6.4 Contenido de los descriptores

Los descriptores de dispositivo y configuración no tienen información específica HID. El descriptor de dispositivo contiene un campo para un código de clase, pero este campo no es donde se define al dispositivo como un HID. En lugar de ello, el descriptor de interfaz es el lugar donde el host aprende que un dispositivo, o más apropiadamente, una interfaz pertenece a la clase HID.

1.11.6.4.1 Descriptor de clase HID

El propósito principal del descriptor de clase HID es identificar descriptores adicionales para uso de las comunicaciones HID. El descriptor de clase tiene siete o más campos, dependiendo del número de descriptores adicionales; si se desea conocer más en detalles el contenido de estos campos remítase al Anexo A correspondiente a los descriptores.

1.11.6.4.2 Descriptor de reporte

Un descriptor de reporte define el formato y uso de los datos que lleva el propósito del dispositivo. Éste necesita ser lo suficientemente flexible para manejar dispositivos con muy diferentes propósitos. El formato no limita el tipo de datos en el reporte, pero el descriptor de reporte debe describir el tamaño y contenido del reporte. Un descriptor de reporte está compuesto de varios items que son requeridos en todo descriptor de reporte, algunos de estos items aplican al descriptor entero, mientras que otros son especificados separadamente para los datos de entrada y salida. Si se desea profundizar en el contenido de cada descriptor de reporte se puede visitar la página web antes mencionada, donde se encuentra la especificación de clase HID así como el contenido y significado de cada item en el descriptor.

1.12 PROPUESTA PARA EL SISTEMA DE ADQUISICIÓN DE DATOS

Hasta el momento se ha presentado un fundamento teórico simplificado acerca del pórtico USB, que ayudará a desarrollar un bloque de adquisición de datos el cual será útil para el módulo que se pretende construir. Por tal razón, a continuación se presenta en la Figura 1.16 un esquema en bloques que describe a breves rasgos y sin mayor detalle los componentes básicos que integran este proyecto.

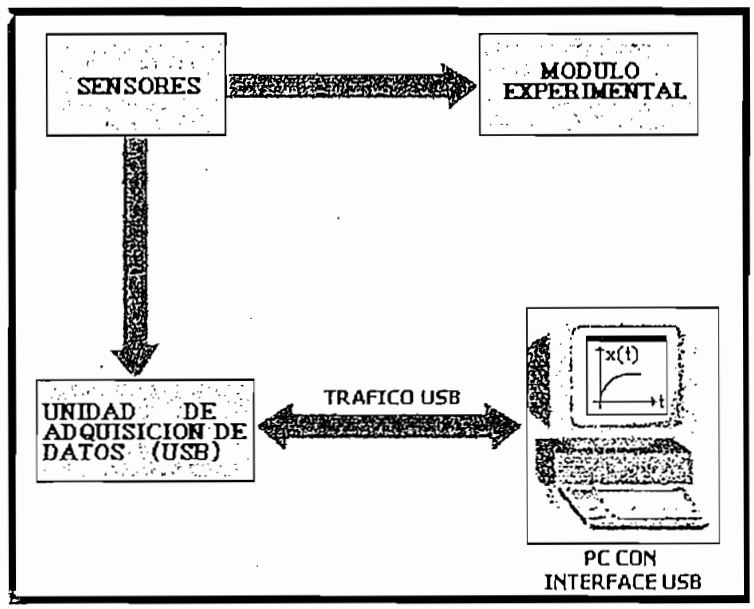


Figura 1.16 Diagrama de bloques

En el esquema anterior se pueden apreciar los siguientes componentes.

1.12.1 MÓDULO EXPERIMENTAL

Componente del proyecto, en el que se encuentra el objeto en movimiento rectilíneo. del cual se adquirirán las variables del fenómeno físico como son: tiempo, posición, etc. Este módulo es provisto por el Departamento de Física de la Escuela Politécnica Nacional y se encuentra en el laboratorio respectivo.

1.12.2 SENSORES

Conjunto de dispositivos que servirán para captar las variables del fenómeno físico antes mencionado. El tipo de sensores a usar para este fin son los de ultrasonido, que luego de realizar un estudio teórico práctico se observó que eran los más adecuados para cumplir con la meta.

1.12.3 UNIDAD DE ADQUISICIÓN DE DATOS

Componente del proyecto que está compuesto por: un circuito acondicionador de señal para la emisión y recepción del ultrasonido, un chip inteligente que tiene el firmware adecuado con la capacidad de procesar los datos recibidos, desde el acondicionador y manejar todo el tráfico con la interfaz USB, que enviará los datos adquiridos a través del bus hacia el computador. Este chip controlador que dispone de la interfaz USB utiliza una de las velocidades dadas en la especificación USB 1.1. Cabe recalcar que el presente proyecto es bastante novedoso por cuanto se hará uso del pórtico USB y aprovechará las varias ventajas que éste ofrece como herramienta para realizar adquisición de datos.

1.12.4 PC CON INTERFAZ USB

Es el computador que disponga de un chip controlador USB, ya sea que aquel esté incorporado en la tarjeta madre (mainboard), o éste sea agregado por medio de una tarjeta PCI con al menos un puerto para realizar el enlace. Es en este componente del proyecto donde residirá el software de aplicación, que adquirirá los datos desde la unidad de adquisición y los procesará para obtener las gráficas de las variables físicas en cuestión.

CAPÍTULO 2

CONSTRUCCIÓN DEL HARDWARE DEL SISTEMA

CAPÍTULO 2

CONSTRUCCIÓN DEL HARDWARE DEL SISTEMA

2.1 INTRODUCCIÓN

Se mencionó ya que el objetivo principal de este proyecto es la implementación de un sistema que permita medir la distancia, velocidad y aceleración de un objeto en movimiento rectilíneo, y mediante el adecuado software de soporte adquirir estos parámetros a un PC por medio del pórtico USB.

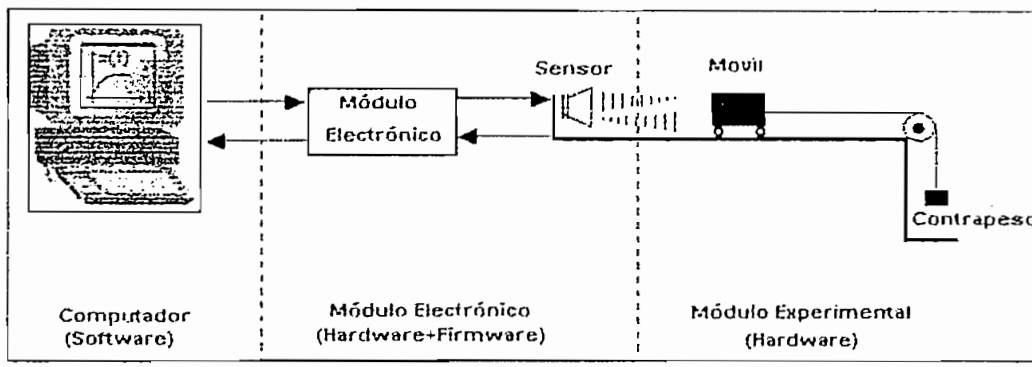


Figura 2.1 Esquema del sistema

La Figura 2.1 muestra un esquema simplificado del sistema que se pretende implementar para conseguir el objetivo planteado, el mismo que consta de tres bloques principales. En este capítulo se analiza únicamente aquello que concierne al hardware, quedando para posterior análisis la parte del software en el PC y el firmware en el microcontrolador PIC 16C745 a utilizar.

2.2 MÓDULO EXPERIMENTAL

El módulo a utilizar en el proyecto se muestra en la Figura 2.2, el mismo que es provisto por el Departamento de Física de la Escuela Politécnica Nacional.

Éste básicamente está conformado por un tubo rectangular de aluminio, con dimensiones aproximadas de 140 cm de longitud, 4 cm de ancho, 5 cm de espesor y está ubicado sobre un plano horizontal; además, al módulo se acopla un dispositivo generador de viento, el cual servirá como un colchón de aire sobre la superficie en la que se moverá el objeto para generar un movimiento rectilíneo libre de fricción. Cabe mencionar que dicho móvil puede variar sus condiciones de masa, y coeficiente de rozamiento; sin embargo, para el proyecto se hace uso de un objeto liviano y con una superficie inferior cuyo coeficiente de fricción es bajo.

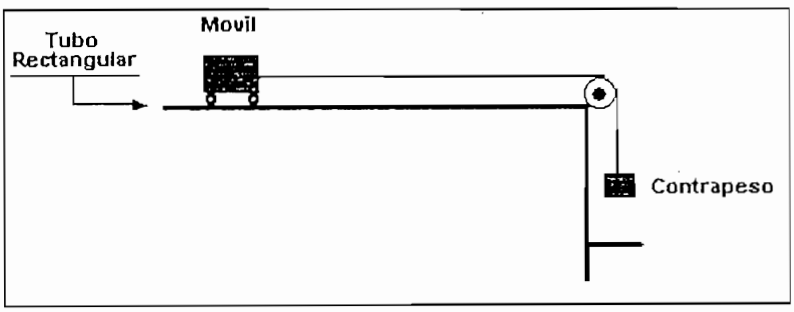


Figura 2.2 Módulo Experimental

2.3 MÓDULO ELECTRÓNICO

El módulo electrónico a construirse debe permitir hallar los valores de distancia del objeto versus tiempo y realizar la adquisición de datos por medio del pórtico USB al PC. Para lograr esto, el módulo electrónico usa sensores de ultrasonido para medir la distancia por medio de la técnica del tiempo de vuelo o pulso eco ya descrita en el Capítulo 1; a partir de estas medidas de tiempo y distancia se deducirán, usando ecuaciones incorporadas en el software del PC, las demás variables de interés como son la velocidad y aceleración. El tiempo será medido por medio de un microcontrolador PIC 16C745 de la Microchip, el mismo que además de los periféricos genéricos que se incluyen en todo microcontrolador ofrece una interface USB para la comunicación con el PC.

La Figura 2.3 muestra el diagrama de bloques del cual se partió para realizar la implementación del módulo electrónico.

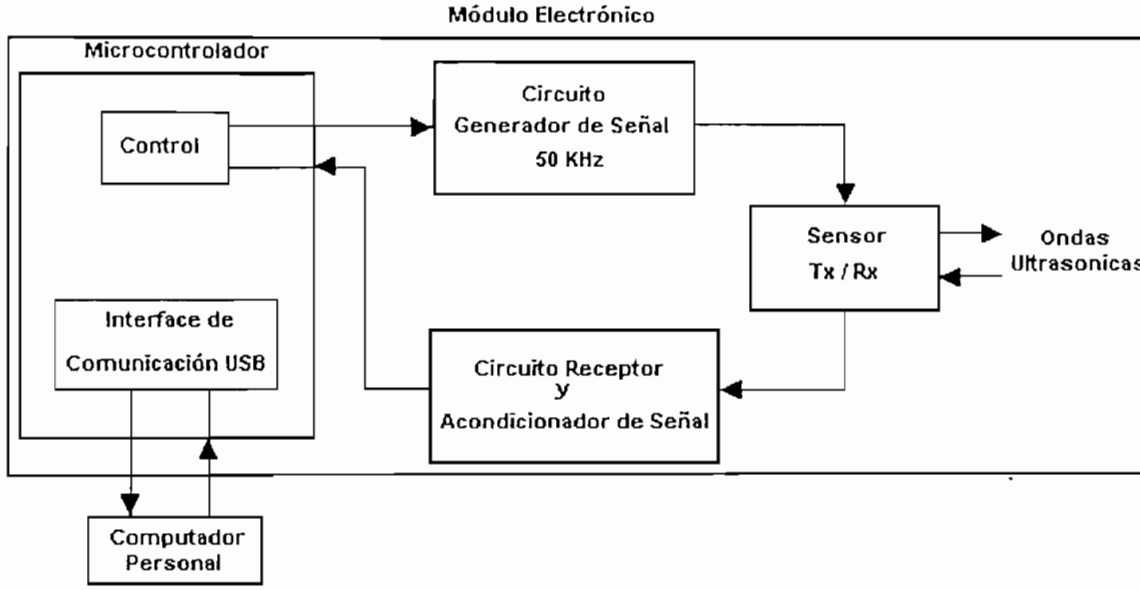


Figura 2.3 Esquema módulo electrónico

- ***El sensor ultrasónico.-*** de tipo electrostático será quien emita y reciba las ondas de ultrasonido.

- ***El circuito generador de señal.-*** genera una señal de voltaje de la frecuencia adecuada para alimentar al sensor ultrasónico.

- ***El circuito receptor y acondicionador de señal.-*** circuito mediante el cual se captan las ondas ultrasónicas reflejadas y se acondicionan para la medición del tiempo de vuelo.

- ***Control e interface de comunicación.-*** circuito que se encarga de tomar las medidas del tiempo de vuelo para el posterior cálculo de la distancia y además permite enviar estos datos al PC mediante la interface USB.

2.3.1 SELECCIÓN DEL SENSOR ULTRASÓNICO

Una de las partes más complicadas en la realización de este proyecto fue la búsqueda del sensor (emisor-receptor) de ultrasonido más adecuado para lograr el objetivo planteado. Después de una búsqueda exhaustiva y dentro de las necesidades requeridas y posibilidades de adquisición se localizaron varios modelos con diferentes características, los cuales fueron sometidos a varios ensayos. Después de realizar tales ensayos con diferentes modelos de sensores a sus respectivas características de frecuencia y voltaje de excitación, se obtuvo que el más apropiado para el proyecto es el de la compañía Polaroid, denominado transductor electrostático Polaroid 6500, mostrado en la Figura 2.4, el mismo que puede ser adquirido en la página web www.senscomp.com

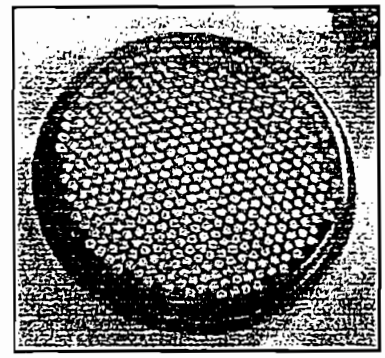


Figura 2.4 Transductor electrostático Polaroid 6500

La razón por la cual se escogió este sensor como el más adecuado, se debe a que éste permite tener mayor repetibilidad en las medidas de distancia, además de mejor exactitud y precisión que con los otros sensores probados. Entre las características de este sensor se pueden mencionar las siguientes: específicamente diseñado para trabajar a frecuencias ultrasónicas en el aire, el ensamblado incluye una cubierta protectora perforada, su frecuencia de operación es de 50 kHz. Este dispositivo trabaja como emisor y receptor simultáneamente, su voltaje de excitación es de 400V pico-pico, y su modelo de emisión sónico (beam patern) se lo representa en la Figura 2.5.

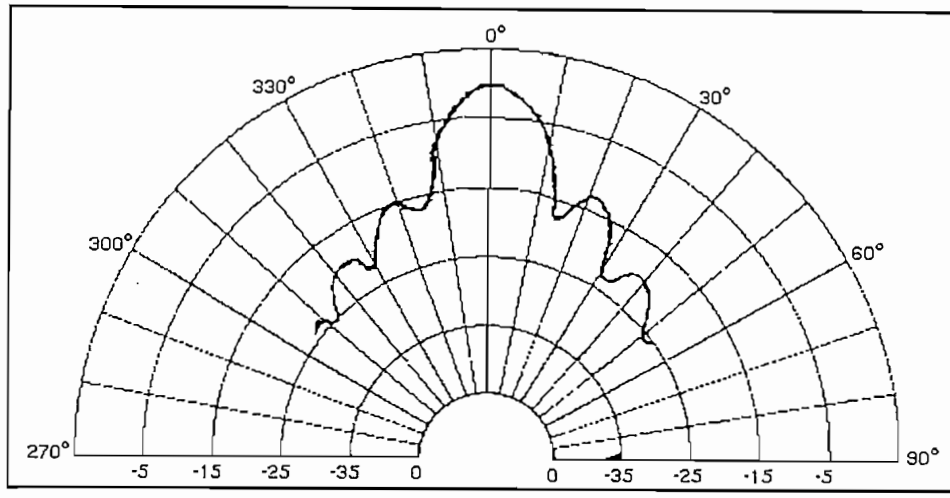


Figura 2.5 Modelo de emisión sónico

Cabe destacar que un modelo de emisión sónico presenta una información valiosa llamada ángulo de emisión, la misma que determina ciertas condiciones y características del lugar donde debe ubicarse el sensor para detectar objetos en línea recta. En este caso para el Polaroid 6500 el modelo de radiación muestra un ángulo de emisión de 60°. Para más detalles acerca del transductor remítase al Anexo D.

2.3.2 DISEÑO Y CONTRUCCIÓN DEL CIRCUITO ACONDICIONADOR DE SEÑAL PARA LA EMISIÓN Y RECEPCIÓN DE LA SEÑAL DE ULTRASONIDO

Antes de comenzar con el diseño de los circuitos acondicionadores para la emisión y recepción del ultrasonido, se debe mencionar que la compañía que fabrica y distribuye los sensores Polaroid 6500 ofrece un modulo extra para acondicionar la señal proveniente de este sensor con un precio aproximado de 100 USD en Estados Unidos. Este módulo acondicionador viene con toda la circuiteria necesaria para la emisión, recepción y acondicionamiento de la señal ultrasónica.

Pese a que este sensor ya incluye este módulo de acondicionamiento, se decidió no usarlo debido a que uno de los objetivos planteados para el presente proyecto fue la elaboración de un sistema de tecnología propia.

2.3.2.1 Diseño del circuito de emisión de ultrasonido

Como ya se ha mencionado anteriormente, el sensor usado para el presente proyecto funciona como transmisor - receptor, con un voltaje de excitación alterno senoidal de aproximadamente 400 Vpp. Con esto se evita el uso de un sensor solo para la emisión y otro dedicado únicamente para la recepción. Ahora, el problema que se debe solucionar es la obtención del voltaje de 400 Vpp necesarios para que el sensor trabaje adecuadamente.

Este problema se resolvió siguiendo las recomendaciones que da el fabricante del sensor, el cual sugiere el uso de un transformador en una configuración de fuente flyback que trabaje a la frecuencia de 50 Khz., el mismo que genera en su secundario un voltaje de alterna elevado a partir de un voltaje de 5 VDC pulsatorio presente en el primario. De esta manera se implementará el emisor de ultrasonido con un transformador de las siguientes características:

- Inductancia en el secundario : 21 ± 2mH a 55 kHz
- Resistencia DC en el secundario : 150 - 200 Ω
- Amplitud del Voltaje de salida : 350 to 450 Volts Pico-Pico
- Voltaje Necesario en el primario : 5 . 0 ± . 1 V D C
- Resistencia DC en el primario : 4 7 ± 1 5 % o h m s

Este transformador se aprecia en la Figura 2.6 puede, ser adquirido al por mayor (compras mayores a 10 unidades) en la casa SENSCOMP cuya página web es www.senscomp.com.

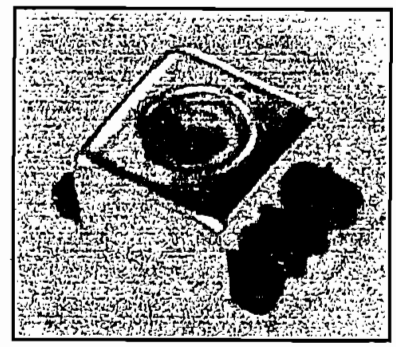


Figura 2.6 Transformador

En la Figura 2.7 se muestra el esquema del circuito implementado para la emisión de ultrasonido, haciendo notar que debido al uso de un único sensor para la emisión y recepción, el secundario del transformador forma parte del circuito receptor de ultrasonido del que se hablará más adelante.

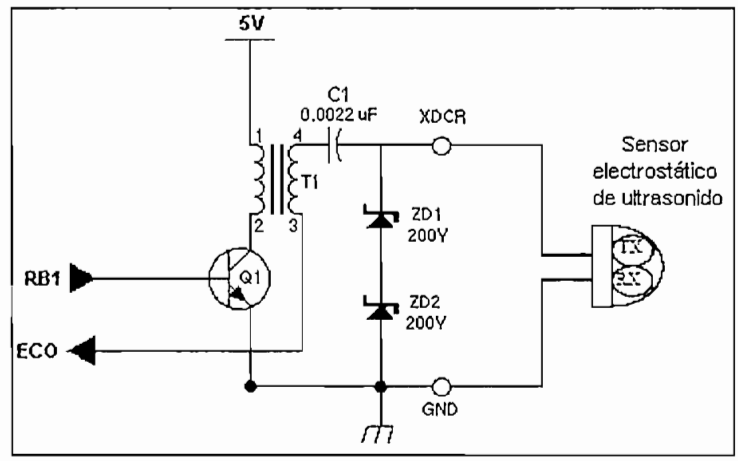


Figura 2.7 Emisor de ultrasonido

Para que el transformador colocado en el circuito emisor genere el voltaje necesario para el sensor, el primario de éste debe conmutar a la frecuencia de 50 Khz. (20 microsegundos de periodo) durante un tiempo equivalente a 10 periodos de la señal (200 microsegundos); esto siguiendo las recomendaciones que los expertos en ultrasonido dan con respecto a la de medición de distancia por medio de la técnica del impulso eco que se utiliza en este trabajo.

Por lo tanto la conmutación se la realizará usando un transistor 2N3904 trabajando en corte y saturación, cuya base está conectada al pin RB1 del microcontrolador PIC16C745 desde donde sale la señal TTL de 50 Khz, generada internamente. Adicionalmente, en la etapa de salida del transformador, por recomendaciones del fabricante del sensor, se colocó un capacitor C1 con las siguientes características:

C1= 0.0022 uF, 400 V

Este capacitor sirve para evitar que el voltaje DC almacenado en el secundario pase hacia el circuito de recepción de ultrasonido, cuando sobre el primario del transformador no se aplica un tren de pulsos. Para asegurar que el voltaje que se genera en el secundario no exceda del requerido por el sensor se coloca un par de zeners de 200V en serie, para así limitar el voltaje de salida a máximo 400V.

2.3.2.2 Diseño del circuito receptor de ultrasonido

El circuito receptor debe captar la señal de rebote (eco) desde el objeto analizado. El eco recibido es una señal alterna senoidal pero de amplitud mucho menor, por lo que es necesario implementar etapas de amplificación previas para tener aquella señal en un nivel aceptable para su análisis; además, ya se ha mencionado que el secundario del transformador utilizado para la emisión de los pulsos ultrasónicos forma parte del circuito receptor, por tal razón antes de amplificar cualquier eco recibido, se debe evitar que la señal de 400 Vpp aproximados que entran al sensor para su funcionamiento, lleguen hacia las etapas de amplificación. Una vez realizado el proceso anteriormente descrito, se procede a la comparación del pulso con un nivel de voltaje de referencia para evitar que señales de ruido indeseables den falsas medidas; y, finalmente se obtiene un pulso de salida positivo el mismo que ingresará al microcontrolador para la cuenta del tiempo por medio de un timer interno.

El diseño e implementación del circuito receptor de ultrasonido parte del diagrama de bloques de la Figura 2.8.

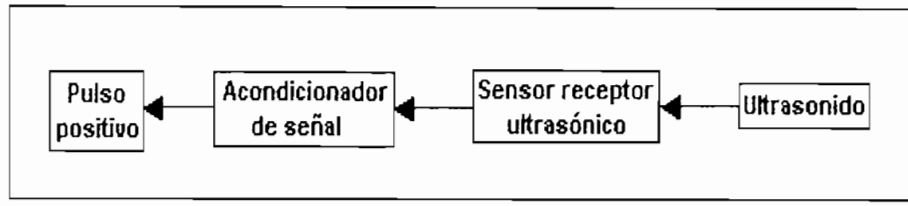


Figura 2.8 Diagrama de bloques del circuito receptor ultrasónico

La Figura 2.9 muestra el bloque de acondicionamiento del diagrama previo dividido en varias etapas, para explicar mejor su funcionamiento e implementación.

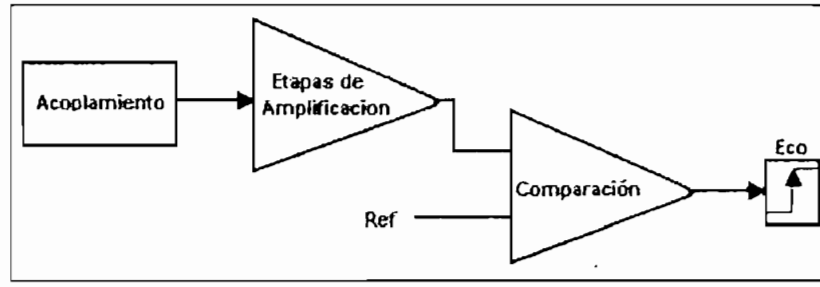


Figura 2.9 Diagrama del bloque del circuito acondicionador receptor de ultrasonido

- ***Etapa de acoplamiento.-*** el objetivo principal de esta etapa es acoplar al sensor, que como se mencionó, forma parte de la emisión y recepción de ultrasonido. Esta etapa es importantísima ya que la recepción se toma del mismo transformador usado para la emisión, el cual genera altos voltajes (aunque con pequeñas corrientes), los mismos que no deben ingresar a las posteriores etapas de acondicionamiento, pues podría alterar su adecuado funcionamiento, o aún peor, dañarlas.

- ***Etapa de amplificación.***- esta etapa tiene por función amplificar la señal proveniente del sensor de ultrasonido.

- ***Etapa de comparación.***- el propósito de esta etapa es generar una señal lógica TTL bien definida, que indique la presencia de un eco válido de ultrasonido. Las señales indeseables de ruido no tendrán esta habilidad y serán rechazadas por el comparador.

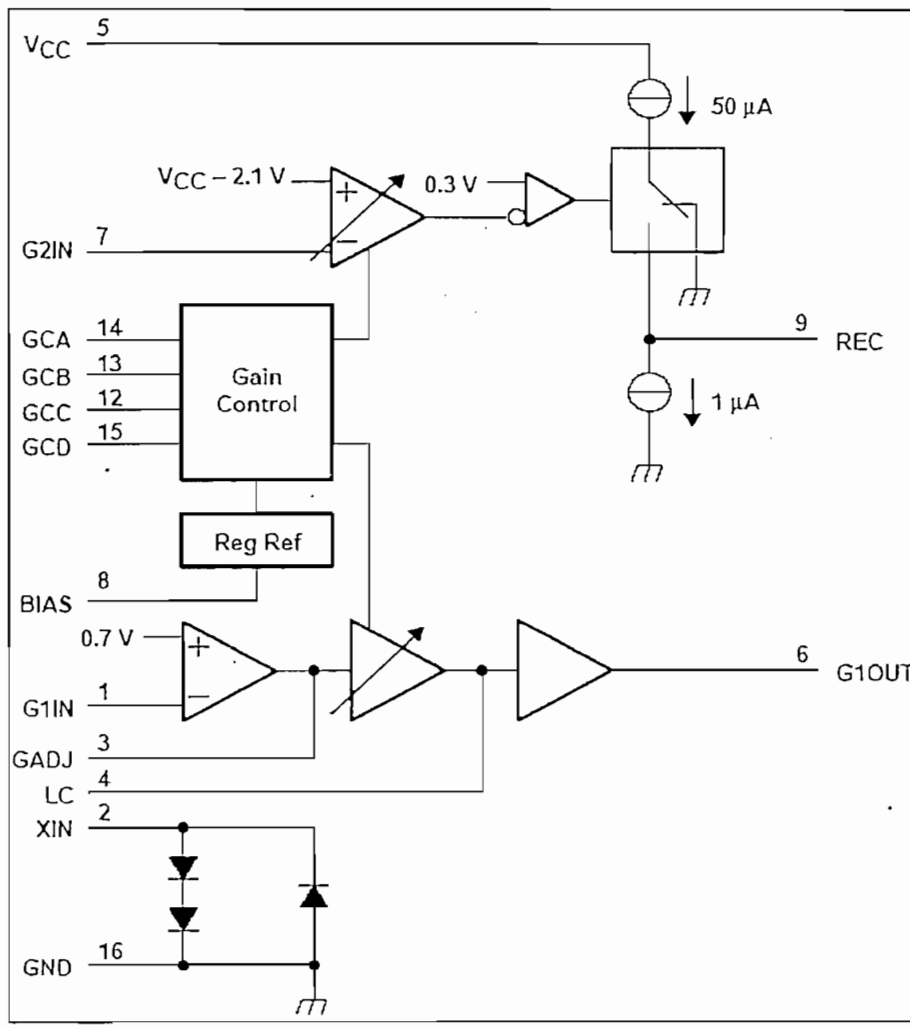


Figura 2.10 Distribución de la constitución interna del TL852

Antes de continuar con la explicación de la implementación de la etapa de acondicionamiento del receptor, se debe mencionar que si bien es cierto se pudo diseñar cada una de estas etapas por separado e implementarlas usando componentes básicos, la compañía Texas Instrument cuya página web es www.ti.com, ofrece el circuito integrado TL852, en el que ya se incluyen las etapas de acoplamiento y amplificación en un único chip para trabajar con los sensores de ultrasonido usados en el presente proyecto.

Por lo tanto, se decidió hacer uso de este chip en el diseño del receptor, pues ofrece varias ventajas entre las cuales se menciona las siguientes: al venir en un único chip implementadas estas dos etapas se evitan los riesgos de introducción de ruidos externos que resultan de acoplar dos etapas distintas en un circuito acondicionador y que podrían causar lecturas erróneas, además de utilización de filtros externos para eliminar tales ruidos. Por otro lado, el circuito de acondicionamiento se hace más pequeño que realizándolo por etapas separadas de tal manera que el trabajo de colocarlo en una tarjeta de circuito impreso es más sencillo y económico.

Por otro lado, si bien es cierto que usar este único chip trae algunas ventajas, al seguir las recomendaciones de implementación del fabricante se presentaron dificultades que obligaron a realizar ciertas modificaciones a los componentes externos usados para la construcción del circuito receptor, las mismas que se discutirán más adelante en este mismo capítulo. A continuación se procederá con la explicación de las etapas de acondicionamiento presentes en el circuito integrado TL852 usado en este proyecto, y cuyo diagrama de constitución interna se muestran en la Figura 2.10.

2.3.2.2.1 Circuito de acoplamiento

La Figura 2.11 indica la parte de acoplamiento presente en el integrado TL852, la misma que básicamente consta de dos diodos colocados en serie, y uno colocado en inverso paralelo.

Según las especificaciones del fabricante esta entrada denominada XIN en el integrado sirve de enganche para el secundario del transformador cuando es usado para manejar el transductor transmisor.

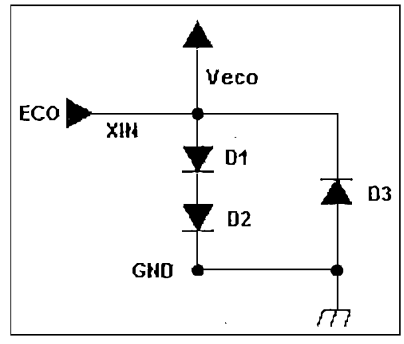


Figura 2.11 Circuito de acoplamiento del sensor ultrasónico

Básicamente lo que hace esta etapa es rectificar en media onda eliminado el semiciclo negativo de las señales presentes en el secundario y luego limitando el voltaje positivo a máximo 1.2V, para de esta manera no dañar el resto de la circuiteria con el voltaje de 400 Vpp que se generan en el transformador en el instante de la transmisión. Se debe recalcar que la señal de eco recibida tiene un valor muy pequeño razón por la cual los diodos en serie no le afectan para nada; finalmente el voltaje ***Veco*** presente, pasa a la etapa de amplificación.

2.3.2.2.2 Circuito de amplificación

La Figura 2.12 muestra la parte de amplificación presente en el integrado TL852, y el diseño de la misma se llevó a cabo siguiendo las recomendaciones tanto del fabricante del chip (Texas Instrument) como del fabricante del sensor (Polaroid). A continuación se describirá la etapa de acondicionamiento total en este circuito integrado dividiéndola en dos partes, que en la Figura 2.12, se representan como primera etapa de amplificación y como segunda etapa de amplificación.

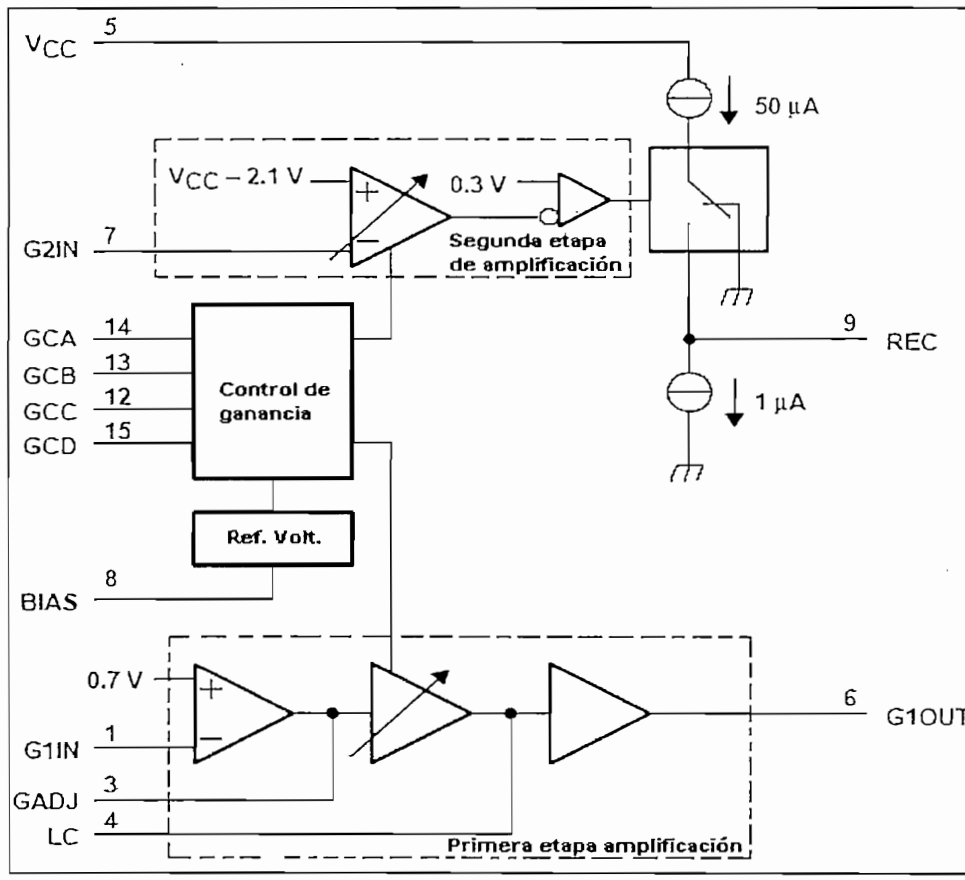


Figura 2.12 Circuito de amplificación del sensor ultrasónico

- ***Primera etapa de amplificación.-*** Como se puede apreciar en la Figura 2.13 está compuesta de tres amplificadores operacionales colocados en cascada, de los cuales, los primeros dos constituyen la parte de amplificación, en tanto que el tercero es únicamente un acoplador de impedancias para unir la señal de salida de esta etapa con la denominada segunda etapa de amplificación. El fabricante del chip TL852 da acceso a la entrada inversora y a la salida del primer amplificador operacional en cuyos pines (1 y 3) se debe colocar la resistencia de realimentación que controlará la ganancia para este operacional, y es en la entrada inversora del mismo donde ingresará la señal de eco recibida por el sensor.

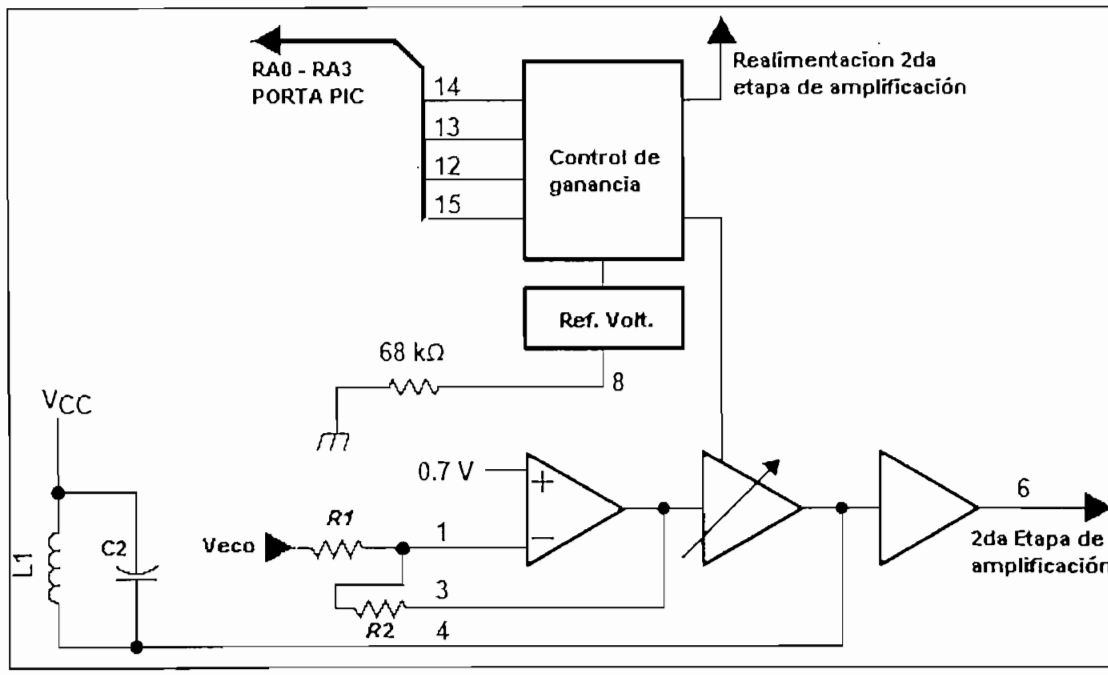


Figura 2.13 Circuito, primera etapa de amplificación

Tomando en cuenta las recomendaciones que da el fabricante, es aconsejable que la salida total de la primera etapa de amplificación sea de máximo 230 mVpp para no tener errores en la medida de distancia, y para conseguir esto se deben tener en cuenta dos parámetros: primero la resistencia a la entrada inversora al primer amplificador operacional puede ser de 1.5 kΩ, y segundo, la resistencia de realimentación debe tener un valor comprendido entre 1.5kΩ y 20 kΩ, el mismo que depende del tipo de aplicación y tipo de sensor ultrasónico a usar. En consecuencia, al hacer las debidas pruebas experimentales se encontró que para esta aplicación un valor de resistencia de 3.3 kΩ era la precisa para tener una buena respuesta a la distancia máxima esperada para el proyecto, por lo tanto, como se aprecia en la Figura 2.13, estas resistencias serán:

$$R1 = 1.5\ k\Omega,\ y\ \ R2 = 3.3\ k\Omega$$

Con estos valores de resistencias se tendrá para el primer amplificador operacional una ganancia de:

$$Av_1 = \frac{R_2}{R_1} = \frac{3.3k\Omega}{1.5k\Omega} = 2.2$$

El segundo amplificador operacional se compone de un bloque de control de ganancia variable, que permite realizar una variación de la misma, cambiando la palabra digital de 4 bits que se coloca en los pines 12 al 15 del chip TL852. En la figura 2.14 se puede apreciar el valor de la ganancia de este operacional en función del valor de la palabra o paso colocado en este chip.

Examinado varias aplicaciones hechas con el sensor ultrasónico usado en este proyecto, existe un chip digital que permite variar la ganancia de este segundo amplificador de acuerdo a la distancia a la cual se encuentra el objeto cuya distancia se desea medir; esto es especialmente útil para cuando el rango de medida deseado es mayor a 1.5 m. Sin embargo, para esta aplicación la ganancia se controla por medio de cuatro salidas del microcontrolador PIC 16C745, al que se le da un paso de ganancia .

Por último, el fabricante del sensor recomienda colocar una red LC en paralelo entre Vcc y el pin 4 del chip. Esto se realiza con el fin de obtener un amplificador con control de ganancia variable para compensar la atenuación de la señal con la distancia y además maximiza el rechazo al ruido. Se debe tener especial cuidado en elegir los valores de capacitancia C1 e inductancia L1 para que trabajen adecuadamente a la frecuencia de trabajo del sensor; es este el caso que para el sensor Polaroid 6500, tales valores son los siguientes:

C2=0.01uF, y L1=1mH

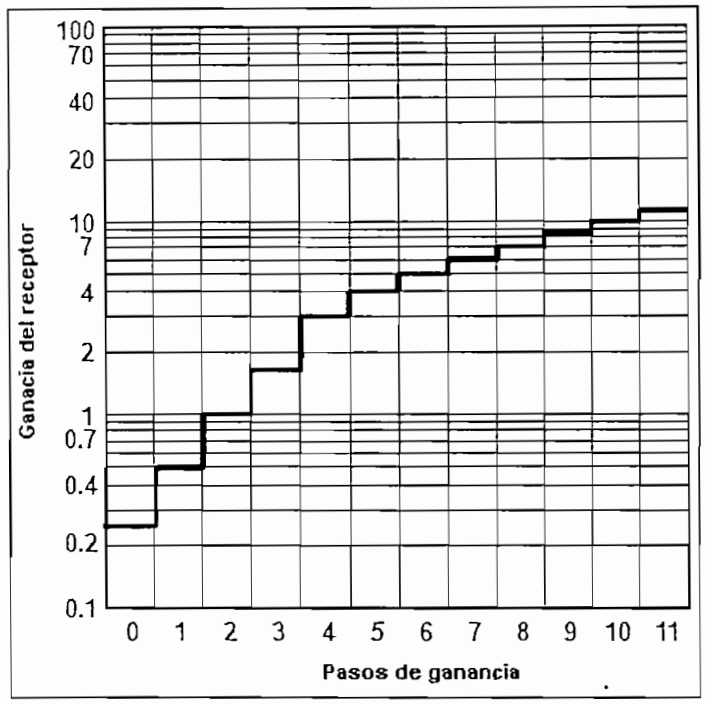


Figura 2.14 Gráfico, ganancia del receptor vs. pasos de ganancia

La salida por el pin 6 del chip TL852 ingresará a la denominada segunda etapa de amplificación.

- ***Segunda etapa de amplificación.-*** esta etapa de ganancia básicamente está constituida por un único amplificador operacional, el cual, como puede apreciarse en la Figura 2.12, también está controlado por medio del bloque de control digital. El seteo de ganancia ya fue considerado en la etapa anterior; de tal manera que únicamente aquí se discutirá brevemente el acoplamiento que se realiza entre esta etapa y la anterior.

El fabricante del circuito integrado TL852 aconseja colocar un capacitor C3 de 0.01 uF para un acoplamiento de AC adecuado entre la primera y la segunda etapa de amplificación. La Figura 2.15 muestra la circuiteria implementada para esta última.

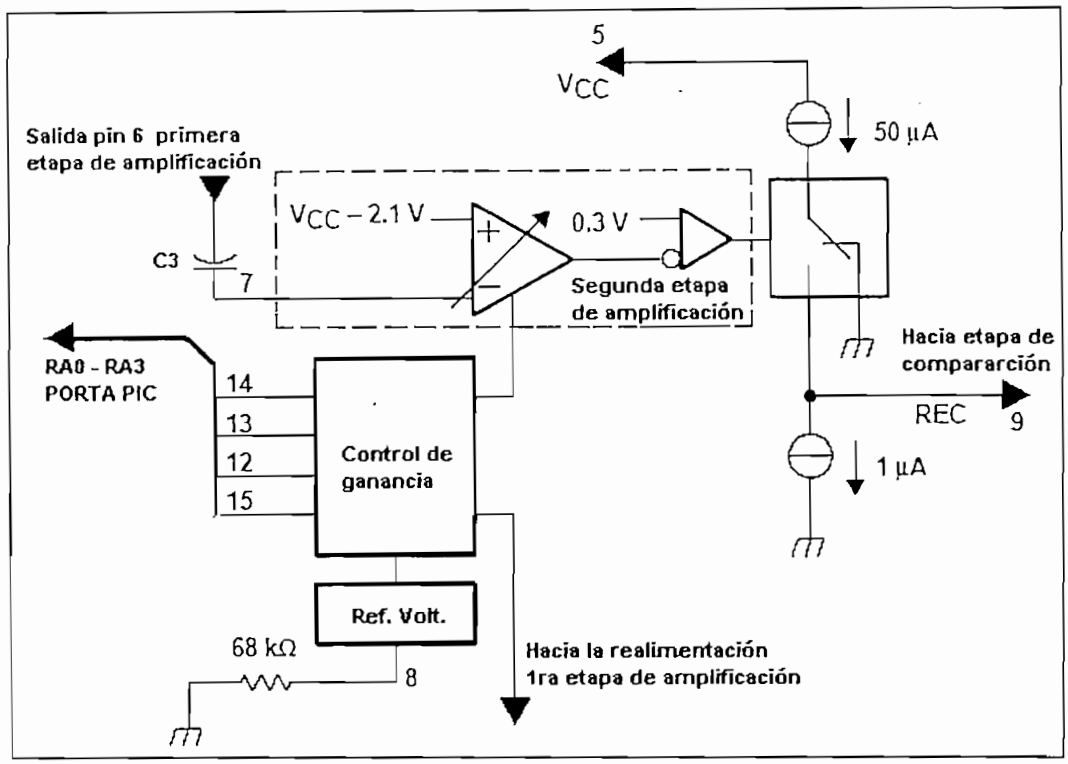


Figura 2.15 Diagrama, segunda etapa de amplificación

La salida del pin 9 irá hacia la etapa final de comparación. Se puede observar de la gráfica anterior que la salida del chip es una fuente de corriente que conmuta entre 1uA a tierra cuando no hay algún eco ultrasónico recibido y 50uA cuando se ha recibido un eco de amplitud aceptable y mayor a 0.3V.

2.3.2.2.3 Circuito de comparación

El circuito de comparación diseñado e implementado se lo muestra en la Figura 2.16. El diseño de esta etapa se inicia eligiendo el operacional a usar para la comparación de la señal de salida total del circuito integrado TL852 (pin 9), con un voltaje de referencia de 0.7 V.

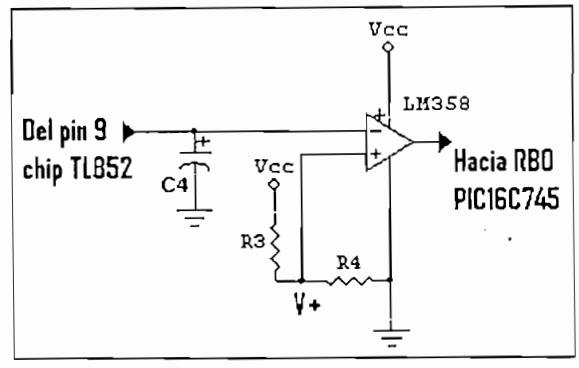


Figura 2.16 Circuito de comparación

Se escogió colocar un amplificador operacional LM358 por las siguientes razones: es un circuito integrado de fácil consecución y de tamaño pequeño adecuado para no complicar las dimensiones de la tarjeta de circuito impreso final.

Además, el proyecto implica el uso de una única fuente de +5V que saldrá del PC por medio del puerto USB utilizado como interface, por ello el LM358 es adecuado, pues puede funcionar con fuente única en un rango de alimentación de +4 a +32 Vdc; y, por último, es un amplificador de bajo ruido lo que lo hace aún más ideal que los amplificadores operacionales de uso genérico como el LM324.

A la entrada inversora (V_+) se conecta una referencia de voltaje de 0.7 Vdc, la cual es útil para evitar falsos disparos que pudieran suscitarse a causa de señales de ruido indeseables.

Como se aprecia en la Figura 2.16 este voltaje se lo obtiene de un sencillo divisor de voltaje, que se conecta directamente a tal entrada sin temor a caídas de voltaje, pues la impedancia de entrada de los amplificadores operacionales es de algunos cientos de kΩ, así las resistencias fueron diseñadas de la siguiente manera:

$$V_+ = \frac{R_4}{R_3 + R_4} * V_{cc}, \text{ de donde asumiendo } R_4 = 1.5\ k\Omega \text{ se obtiene un } R_3 = 10 \quad k\Omega$$

Con estos valores de resistencia se obtiene un V_+ de 0.65 Vdc que también es conveniente, y se limita la corriente que circule por este ramal a un valor seguro que no dañe las entradas del amplificador operacional. Por otro lado, la salida del circuito acondicionador de señal (pin 9 TL852) se acopla a la entrada inversora del LM358 por medio de un capacitor C_3, el mismo que sirve como un integrador de la fuente de corriente constante que sale del chip amplificador – acondicionador; y de acuerdo a recomendaciones técnicas del fabricante este debe tener un valor de :

$$C_4=1000\ pF$$

Así se obtendrá de este pin una señal tipo rampa creciente apropiada para la comparación que se realiza en esta etapa, como lo muestra el diagrama de la Figura 2.17.

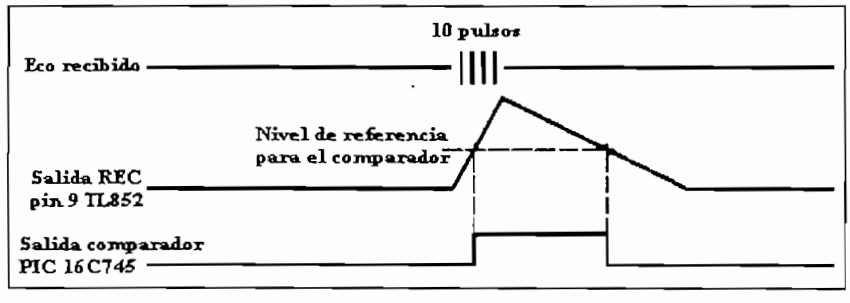


Figura 2.17 Formas de onda para el circuito acondicionador y comparador

Para finalizar con la parte de diseño e implementación del hardware, se mencionará que se escogió el microcontrolador PIC 16C745 por los siguientes motivos:

- Este microcontrolador tiene implementado en su hardware toda la interface necesaria para conectarse con un PC por medio del puerto USB, lo que implica los drives para manejar el tipo de señales que la interface USB usa, así como el formato de codificación utilizado.
- Este microcontrolador permite manejar por firmware parte del protocolo que la especificación USB manifiesta en su documento.

- Permite trabajar en baja velocidad de acuerdo al formato de la especificación USB, lo que para este proyecto es suficiente dadas las restricciones de fabricación del sensor.

Así el hardware total implementado para la emisión, recepción y control de las señales ultrasónicas se indican en la Figura 2.18

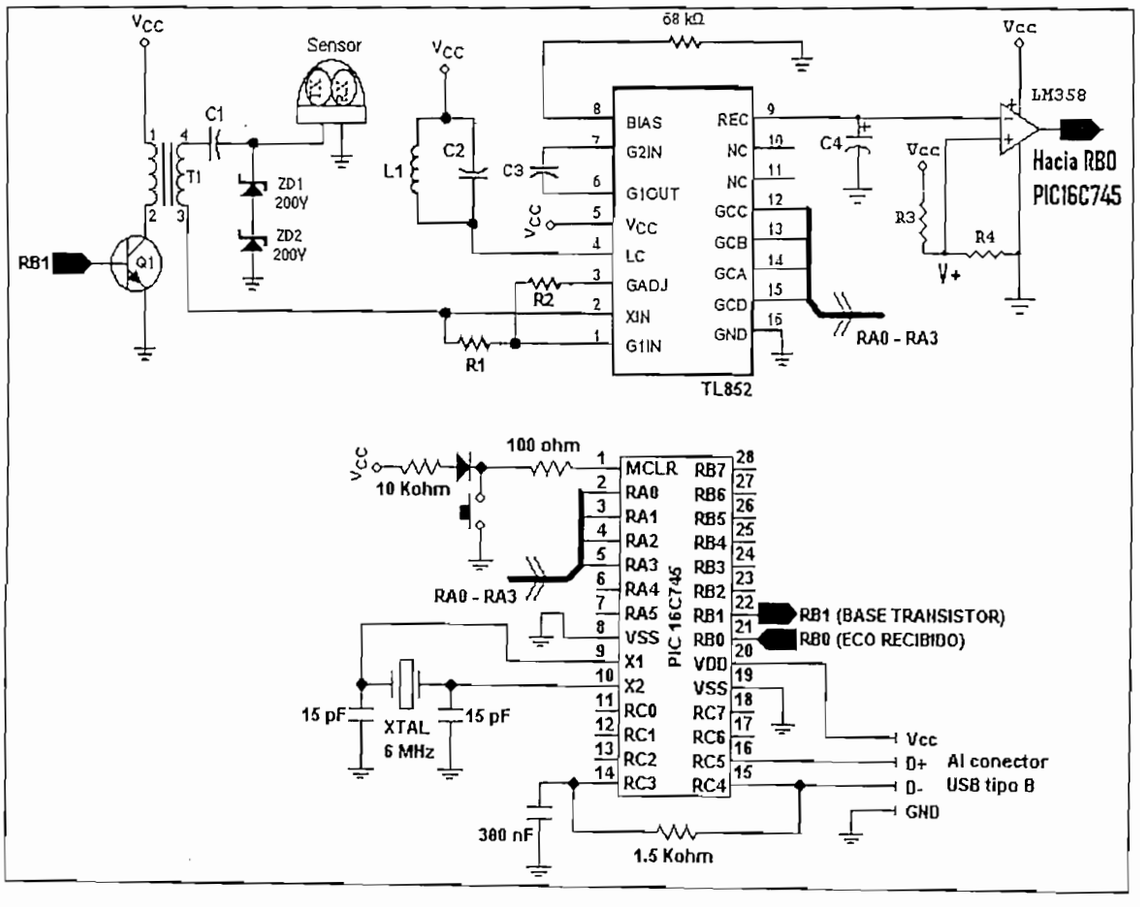


Figura 2.18 Hardware completo implementado

En el próximo capítulo se hablará sobre el desarrollo del software de control y soporte del presente sistema.

CAPÍTULO 3

DISEÑO DEL SOFTWARE DEL SISTEMA

3.1 INTRODUCCIÓN

Este capítulo describirá principalmente el desarrollo del firmware en el microcontrolador y el software en el PC. El programa en el microcontrolador básicamente consta de dos puntos: la emisión - recepción del ultrasonido y el envío de los datos a través del puerto USB hacia el computador. Por otro lado, el software en el PC será quién proporcione la interfaz con el usuario, pida los datos al dispositivo USB y en base a estos, mediante la implementación de algoritmos, se determinará los valores de la posición, velocidad y aceleración del objeto. Una vez que se han conseguido estos valores el software se encargará de presentarlos al usuario, ya sea en forma gráfica o a través de una hoja de cálculo electrónica.

3.2 DESARROLLO DEL PROGRAMA PARA EL MICROCONTROLADOR

El programa en el microcontrolador debe realizar las siguientes tareas, para cumplir con el objetivo planteado: primero el microcontrolador debe establecer un enlace con el computador host por medio del pórtico USB, una vez establecida la comunicación entre los dispositivos se debe configurar adecuadamente el microcontrolador para que emita un tren de pulsos por uno de los pines de este chip (RB1) hacia el circuito de acondicionamiento del emisor que generará la onda ultrasónica. Otra cosa muy importante que debe realizar el microcontrolador es tomar el tiempo que tarda la onda ultrasónica en reflejarse en el móvil y regresar, que es la variable principal para determinar la posición. Otra tarea adicional es tomar el tiempo entre muestreos de posición, pues el objetivo principal es determinar como se encuentra la posición con respecto al tiempo.

Por último, este dispositivo se encargará de transmitir el valor de estas dos variables al PC. El siguiente diagrama de flujo explica con más detalles las funciones que el microcontrolador realizará.

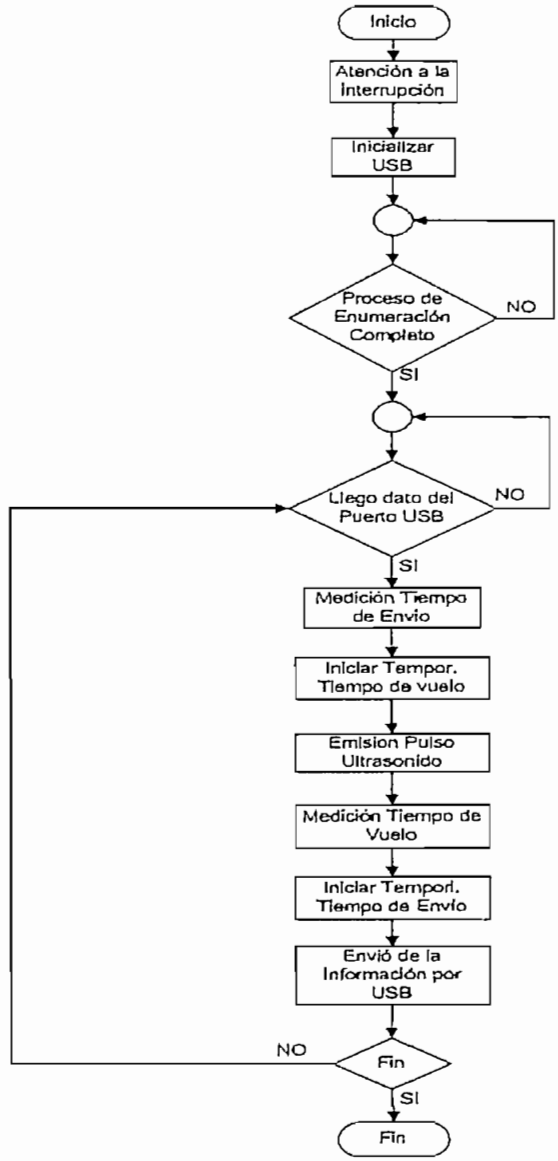


Figura 3.1 Diagrama de flujo del Microcontrolador.

A continuación se explicará la forma estructural de cada bloque que forma el algoritmo.

3.2.1 ATENCIÓN A LA INTERRUPCIÓN

En esta parte del programa se examina la causa que provocó la interrupción y se dirige a una rutina de atención a dicha interrupción.

Las causas que pueden originar una interrupción pueden ser: interrupción originada por algún evento del USB o por desbordamiento del Timer 1. A continuación se detalla esta tarea en lenguaje estructurado.

Atención a la interrupción

- *Salvar registros de: trabajo, estado, bits más significativos del contador de programa y de direccionamiento indirecto*
- *Examinar banderas de interrupción para ver cual fue la causa de la misma y saltar a la rutina correspondiente*
- *Restaurar registros de: trabajo, estado, bits más significativos del contador de programa y de direccionamiento indirecto a sus valores originales*
- *Regreso a la siguiente instrucción desde donde se saltó a la rutina de atención a la interrupción*

Fin Tarea

3.2.1.1 RUTINAS DE INTERRUPCIÓN

A continuación se describe cada una de las rutinas de atención a las posibles causas de interrupción. Todas estas rutinas excepto la del Timer 1 son originadas por algún tipo de evento, debido al trafico USB entre el dispositivo y el host.

RUTINA TOKEN

Rutina que se encarga de examinar que tipo de paquete TOKEN ha llegado a la interfaz USB del microcontrolador. Los tipos de paquetes TOKEN que puedan llegar al microcontrolador son: IN, OUT, Setup. De estos se debe examinar que tipo de información es la que contienen cada unos de ellos para determinar si se ha realizado un tipo de pedido por algún descriptor, necesario en el proceso de enumeración, o se han pedido datos de entrada o salida genéricos.

Rutina de interrupción debido a la recepción de un paquete TOKEN

- *Copiar los datos recibidos desde el buffer del Endpoint a un espacio de memoria en RAM compartida.*
- *Limpiar banderas de interrupción activadas por esta interrupción*
- *Examinar, se ha recibido un paquete TOKEN IN?*

  SI:

  Paquete recibido para el Endpoint 0?

  SI:

  Es un pedido Conseguir Descriptor?

  SI:

  Examinar que tipo de descriptor y enviarlo usando el Endpoint 0

  NO:

  Examinar otro tipo de pedido y ejecutarlo

  Retorno de rutina a seguir examinando otra causa de interrupción

  NO:

  Examinar, se ha recibido un paquete TOKEN OUT?

SI:

Paquete recibido para el Endpoint 0?

SI:

Es un pedido Setear Reporte HID?

SI:

Recibirlo y copiarlo como si fuera del Endpoint 1

Retorno de rutina a seguir examinando otra causa de interrupción

NO:

Resetear Endpoint 0 OUT y enviar un paquete de longitud 0 con el Endpoint 0 IN

NO:

Examinar, se ha recibido un paquete TOKEN SETUP?

SI:

Copiar los datos del buffer del Endpoint 0 a RAM compartida para posterior análisis

Llegó un pedido dirigido con datos para el host?

SI:

Chequear tipo de pedido y enviar los datos correspondientes hacia el PC

Retorno de rutina a seguir examinando otra causa de interrupción

NO:

Tipo de pedido dirigido hacia el dispositivo y por tanto recibir los datos correspondientes desde el PC

Retorno de rutina a seguir examinando otra causa de interrupción

NO:

Retorno de rutina a seguir examinando otra causa de interrupción

Fin rutina

RUTINA DE RESET USB

Esta rutina setea la dirección del dispositivo a la dirección por defecto 0 y habilita el USB, además realiza la transición de estado del dispositivo desde el Estado Alimentado al Estado por Defecto.

Rutina de interrupción debido a un Reset USB

- *Limpiar el registro de estado del USB o forzar a que este registro sea cargado en un registro de respaldo para procesar otro paquete TOKEN entrante*
- *Configurar un Endpoint 0 OUT y asignarle un buffer para sus datos*
- *Configurar un Endpoint 0 IN y asignarle un buffer para sus datos*
- *Configurar al dispositivo con la dirección por defecto 0 y limpiar todas las banderas de interrupción de los eventos USB*
- *Configurar el Endpoint 0 como Endpoint de control*
- *Habilitar todas las interrupciones de eventos USB excepto la de reanudar actividad (ACTIVITY) y la de suspender el periférico (Sleep)*
- *Habilitar todas las interrupciones debidas a errores en la transmisión USB*

- *Configurar el registro de estado del USB al Estado por Defecto*
- *Retorno de rutina a seguir examinando otra causa de interrupción*

Fin rutina

RUTINA STALL

Esta rutina solo le dice a la interfaz serial USB del microcontrolador que se ha enviado un Handshake STALL. Realmente no se requiere alguna acción sino solo limpiar la bandera de aviso de esta interrupción.

Rutina de interrupción debido a un código de Handshake STALL

- *Limpiar banderas de interrupción activadas por esta interrupción*
- *Retorno de rutina a seguir examinando otra causa de interrupción*

Fin rutina

RUTINA DE ERROR USB

Cuando la interfaz serial del USB ha detectado un error, esta rutina lo que hace es incrementar un registro auxiliar que sirve de contador del numero de errores que se han presentado durante las transacciones USB.

Rutina de interrupción debido a un código de Error USB

- *Limpiar banderas de interrupción activadas por esta interrupción*

- *Examinar tipo de error generado e incrementar el registro auxiliar correspondiente usado como contador de errores*
- *Retorno de rutina a seguir examinando otra causa de interrupción*

Fin rutina

RUTINA DEL TIMER 1

Esta parte del programa setea los contadores del Timer 1 a su máximo valor, con el fin de dar un indicativo de desborde de los mismos y que por alguna razón se ha perdido el pulso de eco.

Rutina de interrupción debido al desbordamiento del Timer 1

- *Poner los registros auxiliares de almacenamiento del valor del Timer 1 al máximo valor como indicación de desbordamiento*
- *Poner en cero los registros contadores del Timer 1 y limpiar la bandera de interrupción del mismo*
- *Retorno de rutina a seguir examinando otra causa de interrupción*

Fin rutina

3.2.2 INICIALIZAR USB

El punto de inicio del programa en el que se configura al microcontrolador como un dispositivo USB agregado al bus, y se configuran adecuadamente las interrupciones a utilizar durante el proceso de enumeración del dispositivo USB. A continuación se lista esta rutina.

Inicializar USB

- *Configurar el registro de estado del USB al Estado Alimentado*
- *Habilitar únicamente interrupción por Reset del bus USB y limpiar todos las banderas de interrupción*
- *Programar el registro de configuración del USB para habilitar el pin 14 del microcontrolador (V_{USB}) con el voltaje nominal de 3.3V a ser manejado por el USB*
- *Habilitar interrupción general de eventos del puerto USB y las interrupciones globales*

Fin Tarea

3.2.3 MEDICIÓN DE TIEMPO DE ENVIO

Este bloque de código toma el tiempo que tarda el envío de los datos hacia el PC hasta que se de una nueva orden para la emisión de otros pulsos ultrasónicos; es decir, se mide el tiempo desde que se detecta un pulso de eco y la emisión de otra onda ultrasónica que pude ser tan grande como 10ms. Por tal razón, es necesario el uso de un timer de 16 bits; es muy importante decir que este tiempo no es constante debido a que la duración de los tiempos de vuelo varían con la posición del objeto. El gráfico de la Figura 3.2 representa la medida del tiempo de envío.

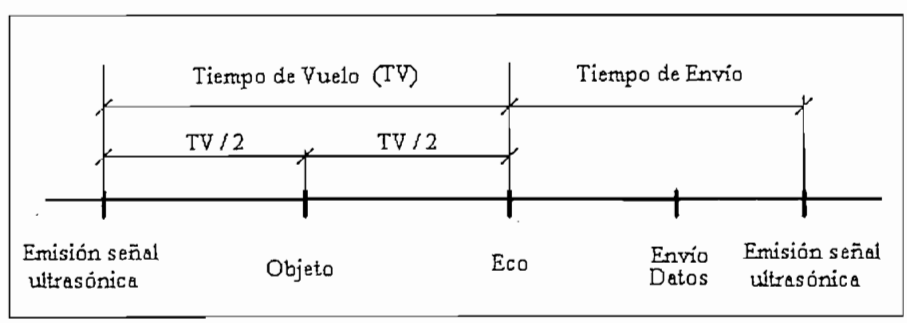


Figura 3.2 Representación gráfica del Tiempo de Envío

Medición tiempo de envío

- *Detener Timer1*
- *Almacenar los valores del tiempo de envío (Timer 1)*

Fin tarea

3.2.4 INICIAR TEMPORIZADOR TIEMPO DE VUELO

En esta tarea se inicializarán los registros del Timer1 para tomar los valores del tiempo de vuelo del ultrasonido.

Iniciar temporizador tiempo de vuelo

- *Limpiar los registros contadores TH1,TL1 del Timer 1*
- *Arrancar el Timer1*

Fin tarea

3.2.5 EMISIÓN DE PULSOS DE ULTRASONIDO

En esta parte del programa se generan 20 pulsos de 50 Khz por el pin RB1 del microcontrolador. Este tren de pulsos, así como la frecuencia de 50 Khz, se generan en base a lazos internos en el microcontrolador, tomando en cuenta que cada ciclo de máquina del mismo es de 166ns.

Emisión de pulsos de ultrasonido

- *Cargar registros con valores tal que generen retardos de 20µs mediante lazos*
- *Complementar RB1 cada 20µs durante 400µs*

Fin tarea

3.2.6 MEDICIÓN DEL TIEMPO DE VUELO

Se espera un tiempo de 400µs luego de la emisión del ultrasonido para detectar un pulso en el pin RB0 del microcontrolador, éste representa el eco de la señal ultrasónica e indica que se debe parar el timer que cuenta el tiempo de vuelo. El tiempo de espera es útil para evitar la recepción de falsos ecos.

Medición del tiempo de vuelo

- *Generar retardo de 400µs para evitar recepción de falsos ecos*
- *Esperar recibir el eco en el pin RB0*
- *Eco detectado?*

  SI:

  Detener Timer1

  Almacenar los valores del tiempo de vuelo presentes en el Timer1

  NO:

  Seguir esperando llegada del eco

Fin tarea

3.2.7 INICIAR TEMPORIZADOR TIEMPO DE ENVÍO

En esta tarea se inicializarán los registros del Timer1 para tomar los valores del tiempo de envío; cabe resaltar que se utilizará el Timer1 que también es usado en la medición del tiempo de vuelo. Esto no conlleva problemas pues estas tareas, como se indica en el diagrama de flujo, son secuenciales; además, es imprescindible la utilización de un timer de 16 bits y el Timer1 es el único que brinda esa opción.

Iniciar temporizador tiempo de vuelo

- *Borrar los registros contadores TH1,TL1 del Timer 1*
- *Arrancar Timer1.*

Fin tarea

3.2.8 ENVÍO DE LA INFORMACIÓN POR EL USB

Esta parte del programa se encarga de coger los datos a ser enviados al PC y colocarlos en el buffer asignado para el Endpoint 1, cargar el número de bytes a enviar y esperar hasta que se envíen estos datos correctamente.

Envío de los datos por el USB

- *Cargar los datos a enviar en el buffer asignado al Endpoint correspondiente*
- *Cargar el número de bytes a enviar en el registro correspondiente*
- *Esperar hasta que se envíen los datos por el USB correctamente*

Fin tarea

3.3 DESARROLLO DEL SOFTWARE DEL PC

La segunda parte de este capítulo tiene que ver con el software que residirá en un PC y que no solo servirá como interfaz para el usuario, sino que se encargará de negociar el intercambio de los datos con el dispositivo USB. Se empezará examinando el software de desarrollo más adecuado que se escogió como herramienta de programación, luego de lo cual se explica con mayor detalle y en forma estructurada como está compuesto.

3.3.1 SOFTWARE DE SOPORTE

La tarea inicial, al desarrollar el software en el PC, fue la de decidir que lenguaje de programación utilizar para el desarrollo de esta aplicación. Si bien es cierto que se cuenta con una gran variedad de lenguajes de programación de alto nivel y basados en Windows para realizar esta tarea tales como: Visual C++, Visual J++, Power Builder, Delphi, Visual Basic, LabView (National Instruments) entre otros, se eligió usar Visual Basic 6.0 de Microsoft por las siguientes dos razones principales:

- Uno de los objetivos sobre los cuales se basa el presente proyecto es la construcción de módulos que sean de fácil consecución en cuanto a costos y, de los lenguajes de programación mencionados, la adquisición de una licencia para tales, era una cuestión costosa que salía del alcance de este objetivo.

- Dado que el tiempo de desarrollo es una variable importante a considerar en el presente proyecto y dada la complejidad de lo que se deseaba conseguir, se decidió usar una herramienta de programación cuya sintaxis fuera lo más familiar posible, y así evitar el gasto de tiempo en el aprendizaje de una nueva herramienta de programación.

Visual Basic para comunicarse con los dispositivos USB debe hacer uso de llamados a funciones API; estas funciones son parte del subsistema win32 de Windows que permiten comunicar las aplicaciones con los drivers del sistema operativo. Tres de las funciones API que sirven para el intercambio de datos con los dispositivos son: ReadFile, WriteFile, y DeviceIoControl.

No todo driver de dispositivo puede soportar estas funciones, en tanto que otros, como por ejemplo los que se utilizan en este proyecto, si lo hacen. Cada llamada puede incluir un pedido o requerir información tal como: escribir datos o leer una cantidad de datos. Pese a que se puede hacer uso de las funciones API para abrir comunicaciones con algún periférico, existe una tecnología de programación denominada, programación con componentes ActiveX, con la cual se pueden crear bloques de programa con su propia funcionalidad e interfaces de comunicación con otras aplicaciones denominados controles ActiveX.

Estos controles, si bien es cierto tienen su propia funcionalidad, necesitan estar incorporados a alguna aplicación contenedora de este tipo de controles para que puedan funcionar y se puedan hacer uso de sus propiedades, y una de estas aplicaciones contenedoras es Visual Basic.

Por tal razón, para realizar toda comunicación con el pórtico USB, en este proyecto se hace uso de un control ActiveX, que permita realizar estas operaciones, el mismo que fue descargado desde la página web www.microchip.com quien es el fabricante. El programa que fue desarrollado en el presente trabajo consta de una interfaz de usuario que incluye cinco ventanas o formularios tipo Windows que son:

1. El primer formulario es la ventana de presentación.
2. El segundo es la ventana principal.
3. El tercero es la ventana en donde se presentará la gráfica de la posición.
4. El cuarto es la ventana en donde se presentará la gráfica de la velocidad.
5. El quinto es la ventana en donde se presentará la gráfica de la aceleración.

La Figura 3.3 muestra un diagrama de flujo general de la aplicación y a continuación se explica como está compuesto cada formulario del programa.

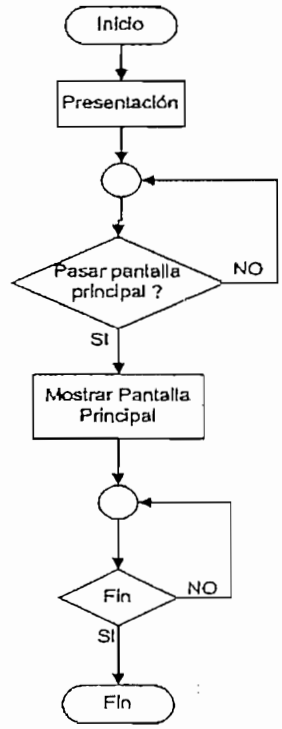


Figura 3.3 Diagrama de flujo general

Inicio

Presentación

- Cargar el formulario Presentación, que muestra información relacionada con el Proyecto de Titulación (Ver Figura 3.4)

Pantalla Principal

- Cargar el formulario Principal que muestra en pantalla un conjunto de controles que permiten elegir a que formulario dirigirse, iniciar la adquisición de datos así como un indicativo de si hubo una conexión exitosa con el dispositivo USB o no (Ver Figura 3.5).

3.3.2 VENTANA DE PRESENTACIÓN

Es la ventana de inicio del programa en el que se muestran datos generales acerca del proyecto. La Figura 3.4 muestra esta ventana.

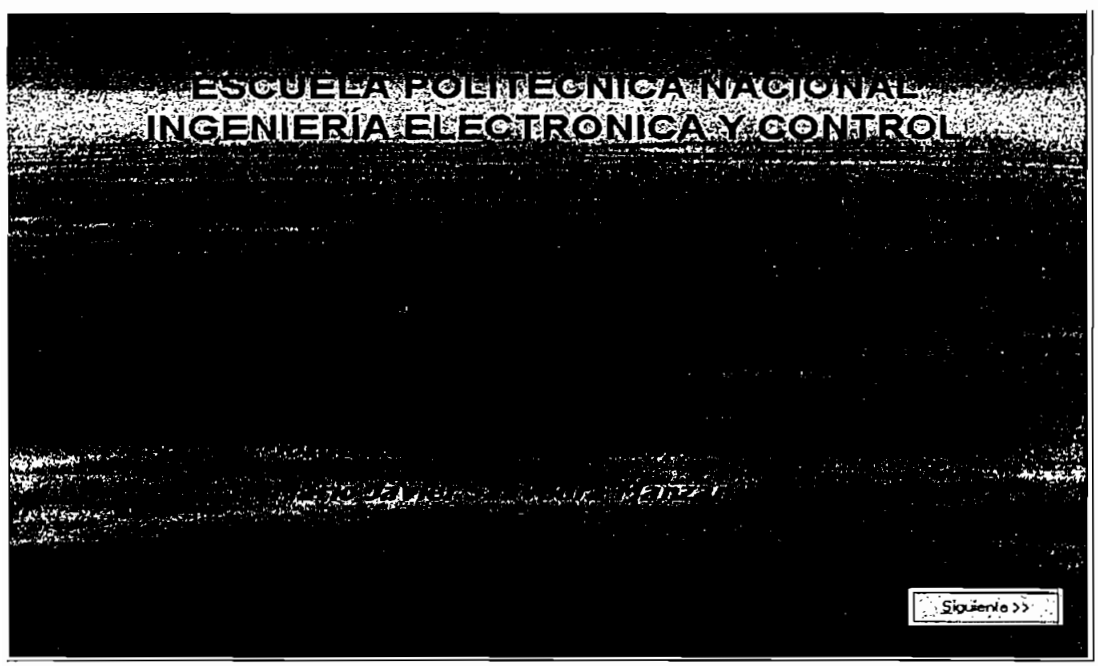


Figura 3.4 Ventana de Presentación

3.3.3 FORMULARIO PRINCIPAL

En el formulario principal se muestran un conjunto de controles que permiten elegir la gráfica de la variable (posición, velocidad, aceleración) que se desea visualizar en otro formulario dedicado para ello, o si se desea comenzar la adquisición de datos a través del pórtico USB. Además se determina la posición a la que se encuentra el objeto utilizando la fórmula ya mencionada en el Capítulo1:

$$d = \frac{T_{vuelo}}{2} * Velocidad\ \ del\ sonido \tag{3.1}$$

Luego, para el cálculo tanto de la velocidad como de la aceleración se realiza la primera y segunda derivada de la posición, y para conseguir esto se hace uso de un algoritmo muy sencillo, el cual realiza la diferenciación en tiempo discreto de una señal muestreada x por medio de la siguiente ecuación:

$$y(i) = \frac{dx_{(i)}}{dt} = \frac{x_{(i+1)} - x_{(i-1)}}{2dt} \quad para\ i = 1,2,3.....n-1 \tag{3.2}$$

Donde :

x .- Señal muestreada a derivar; y .- Derivada de la señal de entrada

dt .- Intervalo de muestreo; n .- Número de muestras

La dificultad que presenta este método se debe a que el intervalo de muestreo debe ser constante y dado que las mediciones de distancia hechas por la técnica a usar en este proyecto, no permiten cumplir esta condición, se observa la necesidad de implementar una solución práctica y coherente a esta dificultad. Después de un análisis, se concluye que la solución más conveniente a este problema, es encontrar la función que pase por el conjunto de puntos obtenidos experimentalmente y que representan la posición del objeto y, una vez obtenida esta función se pueden extraer muestras de la misma a intervalos constantes para calcular por medio del algoritmo de derivación, ya explicado, la correspondiente velocidad y aceleración.

Para encontrar esta función matemática que represente en su totalidad a la función experimental con el mínimo error posible, se usa la técnica de encontrar una función por el método de los mínimos cuadrados, mediante el siguiente conjunto de ecuaciones:

$$\left(\sum_{k=1}^{n} x_k^2\right)a+\left(\sum_{k=1}^{n} x_k\right)b+nc=\sum_{k=1}^{n} y_k$$

$$\left(\sum_{k=1}^{n} x_k^3\right)a+\left(\sum_{k=1}^{n} x_k^2\right)b+\left(\sum_{k=1}^{n} x_k\right)c=\sum_{k=1}^{n} x_k y_k$$

$$\left(\sum_{k=1}^{n} x_k^4\right)a+\left(\sum_{k=1}^{n} x_k^3\right)b+\left(\sum_{k=1}^{n} x_k^2\right)c=\sum_{k=1}^{n} x_k^2 y_k$$

En este conjunto de ecuaciones se deben considerar los siguientes parámetros:

- x_k, y_k son puntos obtenidos experimentalmente a intervalos no constantes.
- a, b, c son los coeficientes de los términos cuadrático, lineal e independiente respectivamente de la función posición experimental
- n es el número posiciones obtenidas del objeto en movimiento

La Figura 3.5 muestra la pantalla del Formulario Principal y la Figura 3.6 un diagrama de flujo estructurado que muestra las tareas que éste realiza.

Figura 3.5 Pantalla del Formulario Principal

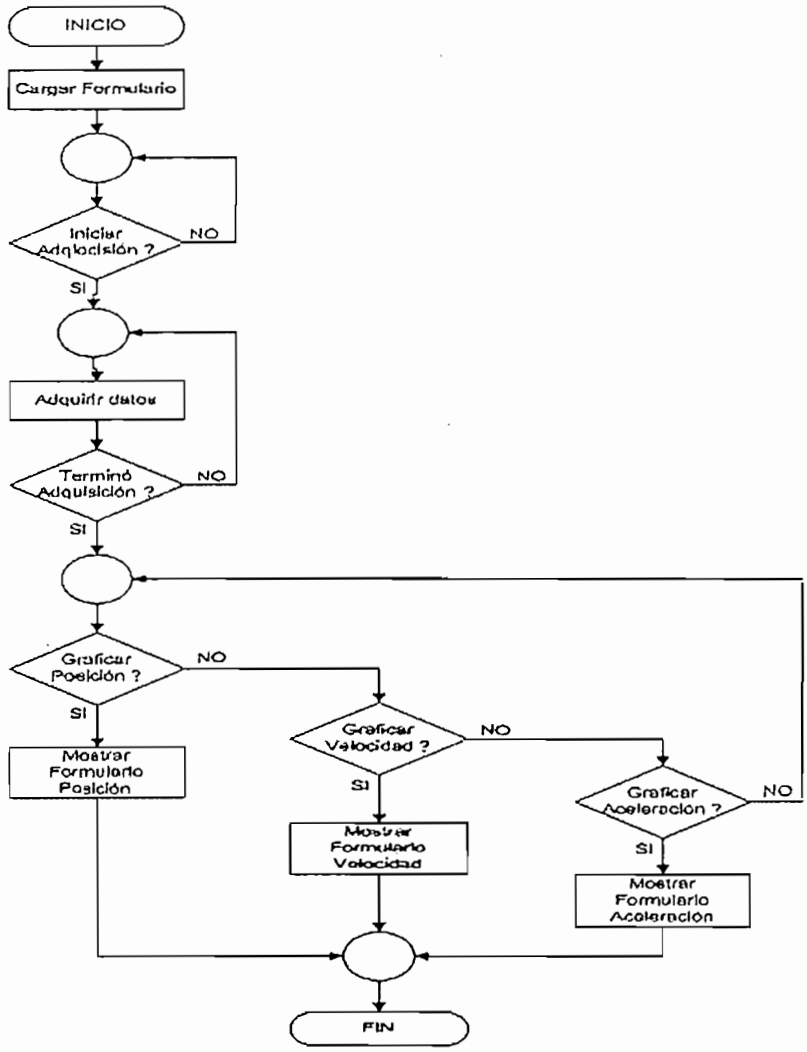


Figura 3.6 Diagrama de flujo del Formulario Principal

A continuación se describe de forma estructurada cada una de las funciones que son parte de este formulario.

Cargar Formulario

- *Establecer los valores de las variables usadas en el proyecto a sus valores predeterminados*
- *Establecer conexión con el dispositivo USB*

- *Fue exitosa la conexión con el dispositivo USB?*

  SI:

  Habilitar opción para iniciar adquisición de datos de tiempo a través del puerto

  Deshabilitar opciones para mostrar gráficas de las variables analizadas

  NO:

  Deshabilitar opción para iniciar adquisición de datos a través del puerto

  Deshabilitar opciones para mostrar gráficas de las variables analizadas

  Mostrar Mensaje "No se pudo establecer comunicación con el dispositivo"

Fin Tarea

Adquirir datos

- *Iniciar comunicación y adquisición de datos de tiempo con el dispositivo por medio del pórtico USB*
- *Finalizó la adquisición de datos?*

  SI:

  Cerrar comunicaciones con el dispositivo USB

  Realizar el cálculo de la velocidad derivando la posición y guardarla en otro arreglo

  Realizar el cálculo de la aceleración derivando la velocidad y guardarla en otro arreglo

Habilitar opciones para mostrar gráficas de las variables analizadas

NO:

Continuar pidiendo datos de tiempo al dispositivo y guardarlos en una arreglo
Calcular la posición del objeto con los datos de tiempo guardados y almacenarlos en otro arreglo

Fin Tarea

Mostrar Formulario Posición

- *Mostrar el formulario donde se graficará la posición*

Fin tarea

Mostrar Formulario Velocidad

- *Mostrar el formulario donde se mostrará la velocidad*

Fin tarea .

Mostrar Formulario Aceleración

- *Mostrar el formulario donde se mostrará la aceleración*

Fin tarea

3.3.4 FORMULARIO POSICIÓN

En este formulario se graficará la posición en función del tiempo; además, el código residente en éste, al igual que en los otros dos que están destinados para las gráficas de velocidad y aceleración, permitirá observar en controles separados el valor de cualquier punto del gráfico.

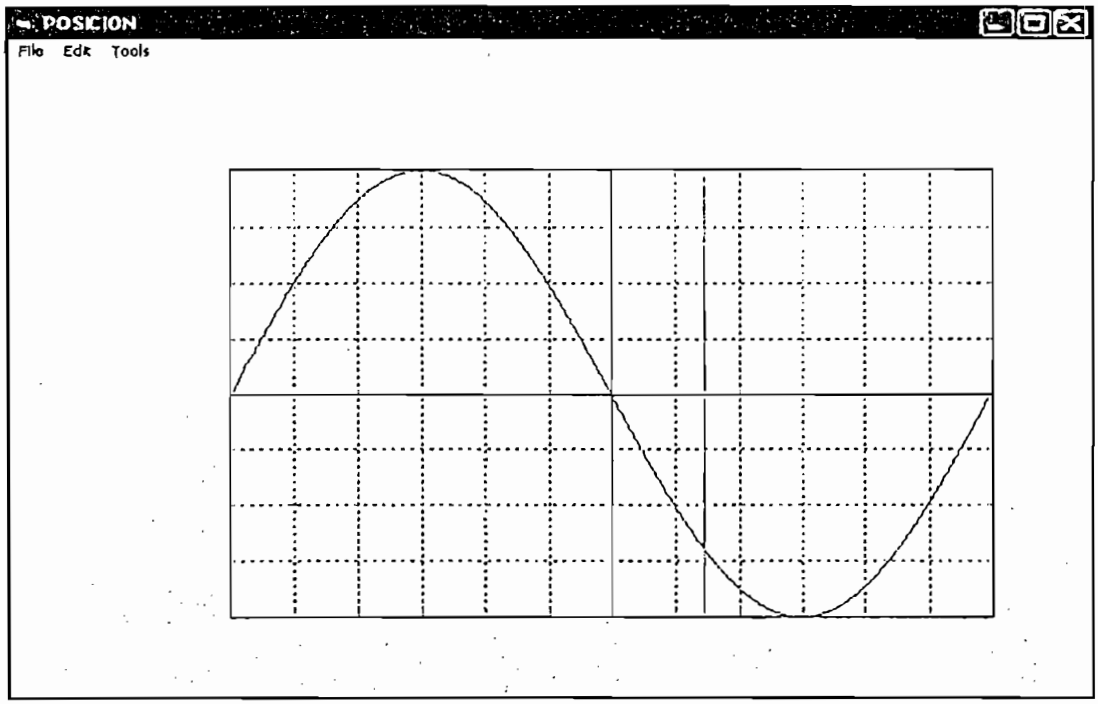


Figura 3.7 Formulario Posición

La Figura 3.7 muestra la pantalla de este formulario y en la Figura 3.8 se indica un diagrama de flujo estructurado con las funciones para este formulario.

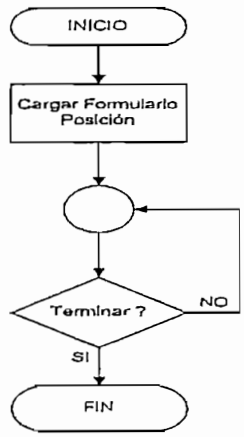


Figura 3.8 Diagrama de flujo para el Formulario Posición

A continuación se describe de forma estructurada las funciones que son parte de este formulario.

Cargar Formulario

- *Dibujar el marco y la gráfica de la posición en función del tiempo*
- *Se cambió el tamaño del formulario?*

  SI:

  Redibujar el marco y la gráfica de la posición en función del tiempo con sus nuevos tamaños
- *Se presionó botón derecho del mouse?*

  SI:

  Mostrar un menú contextual con las opciones de establecer grillas o ejes en el gráfico y ejecutar la acción si que se elija
- *Se presionó botón izquierdo del mouse?*

  SI:

  Mostrar en pantalla el valor del punto actual de la gráfica que intercepta al eje deslizable

Fin Tarea

3.3.5 FORMULARIO VELOCIDAD

Sobre este formulario se graficará la velocidad en función del tiempo; adicionalmente el código residente en este formulario permitirá observar en controles separados el valor de cualquier punto del gráfico.

La Figura 3.9 muestra la pantalla de este formulario y en la Figura 3.10 se indica un diagrama de flujo estructurado con las funciones para este formulario.

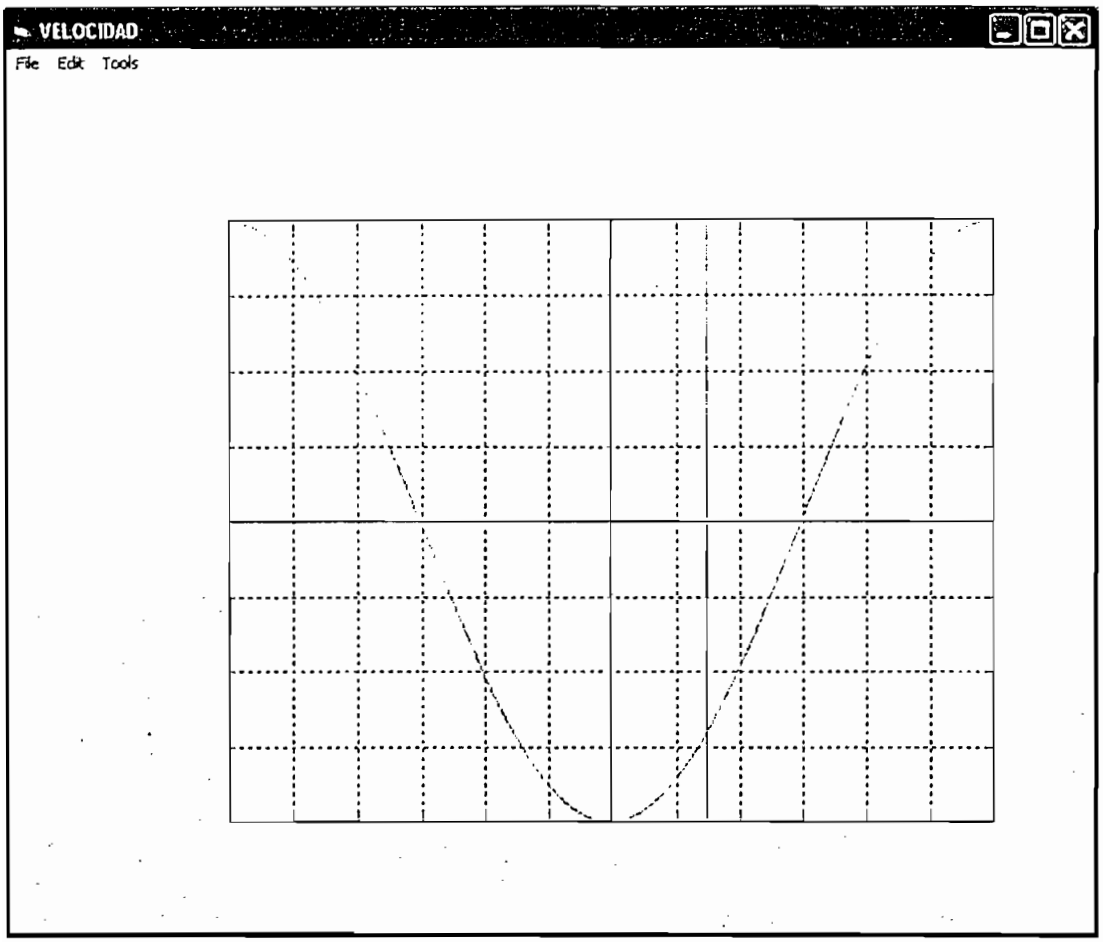


Figura 3.9 Formulario Velocidad

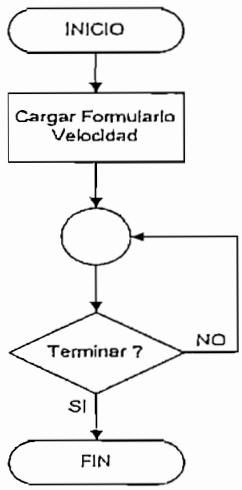


Figura 3.10 Diagrama de flujo para el Formulario Velocidad

A continuación se describe de forma estructurada las funciones que son parte de este formulario.

Cargar Formulario

- *Dibujar el marco y la gráfica de la velocidad en función del tiempo*
- *Se cambió el tamaño del formulario?*

  SI:

  Redibujar el marco y la gráfica de la velocidad en función del tiempo con sus nuevos tamaños
- *Se presionó botón derecho del mouse?*

  SI:

  Mostrar un menú contextual con las opciones de establecer grillas o ejes en el gráfico y ejecutar la acción si que se elija
- *Se presionó botón izquierdo del mouse ?*

  SI:

  Mostrar en pantalla el valor del punto actual de la gráfica que intercepta al eje deslizable

Fin Tarea

3.3.6 FORMULARIO ACELERACIÓN

Sobre este formulario se graficará la aceleración en función del tiempo; adicionalmente el código residente en este formulario permitirá observar en controles separados el valor de cualquier punto del gráfico.

La Figura 3.11 muestra la pantalla de este formulario y en la Figura 3.12 se indica un diagrama de flujo estructurado con las funciones para este formulario.

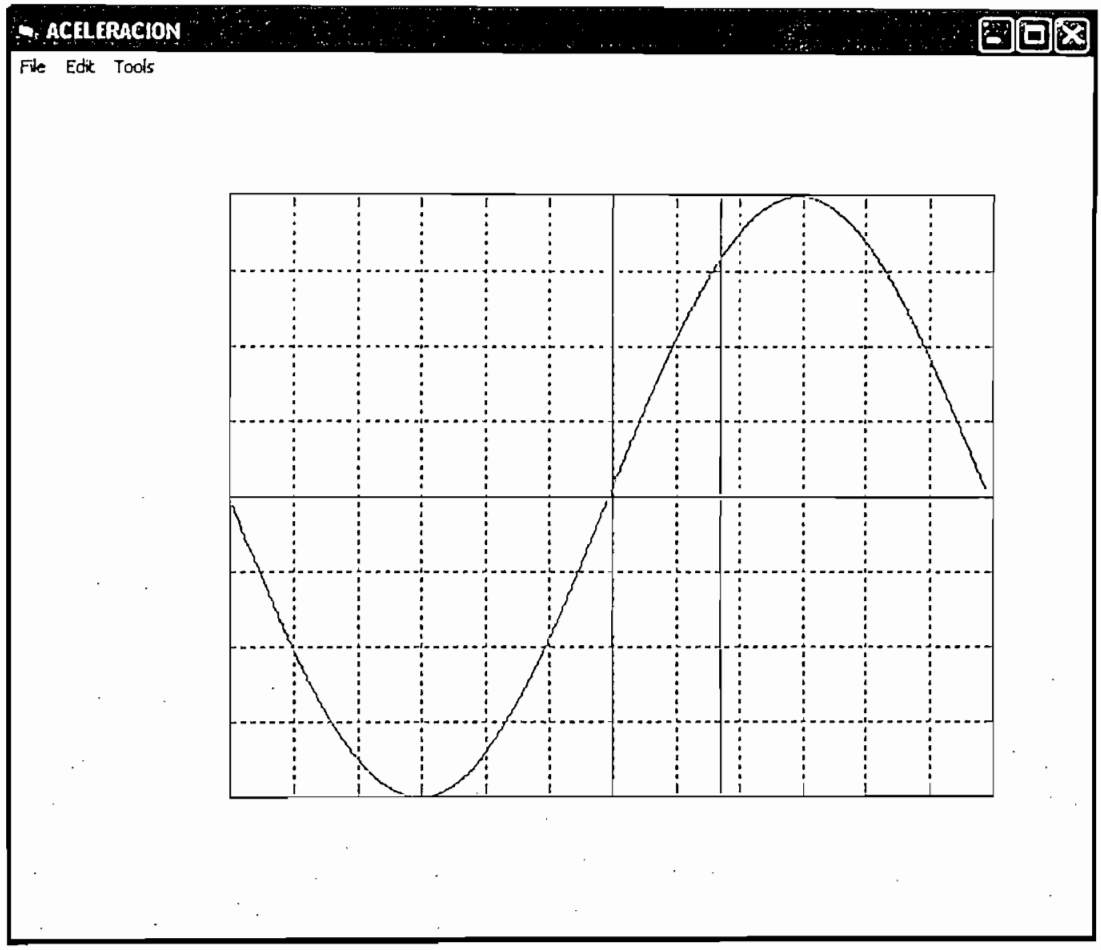


Figura 3.11 Formulario Aceleración

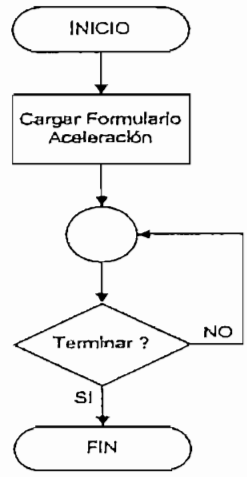


Figura 3.12 Diagrama de flujo para el Formulario Aceleración

A continuación se describe de forma estructurada las funciones que son parte de este formulario.

Cargar Formulario

- *Dibujar el marco y la gráfica de la aceleración en función del tiempo*
- *Se cambió el tamaño del formulario?*

  SI:

  Redibujar el marco y la gráfica de la aceleración en función del tiempo con sus nuevos tamaños
- *Se presionó botón derecho del mouse?*

  SI:

  Mostrar un menú contextual con las opciones de establecer grillas o ejes en el gráfico y ejecutar la acción si que se elija
- *Se presionó botón izquierdo del mouse?*

  SI:

  Mostrar en pantalla el valor del punto actual de la gráfica que intercepta al eje deslizable

Fin Tarea

En el próximo capítulo se hablará sobre las pruebas y resultados obtenidos con la implementación del hardware y software diseñados y descritos en los Capítulos 2 y 3 del presente proyecto.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

CAPÍTULO 4

PRUEBAS Y RESULTADOS

En este capítulo se presentan las pruebas que se diseñaron para comprobar el funcionamiento del sistema desarrollado, se presentan también los resultados obtenidos durante las pruebas.

4.1 PRUEBAS Y RESULTADOS DEL HARDWARE COMPLETO DE MEDICIÓN DE POSICIÓN

Las pruebas realizadas al hardware total implementado se dividieron en dos secciones. La primera de ellas muestra las medidas tomadas sobre un objeto en reposo pero a distintas posiciones; mientras que la subsiguiente sección mostrará las medidas de posición pero en movimiento rectilíneo.

4.1.1 PRUEBAS Y RESULTADOS REALIZADOS CON EL OBJETO EN REPOSO

Con esta prueba se pretende mostrar tanto la repetibilidad como la exactitud alcanzada por el sistema. Si el objeto se coloca en un mismo lugar, las mediciones de su posición deberían ser iguales; esto es lo que se buscó determinar con esta prueba: qué tan bueno es el sistema para repetir un mismo valor. Adicionalmente, cotejando lo que mide el sistema, con la verdadera distancia a la que se encuentra el objeto , se determinará la exactitud.

Para realizar esta prueba se tuvo que ubicar al sensor ultrasónico en la posición más adecuada. Esto quiere decir que al dispositivo se lo tuvo que mover levemente ya sea: hacia la izquierda o derecha o, hacia arriba o abajo hasta conseguir que las medidas de distancia con el objeto en reposo sean lo más constantes posibles, comprobando con esto su repetibilidad.

A continuación se muestran los resultados obtenidos con el objeto en reposo para varios valores de posición medidas durante 1 minuto.

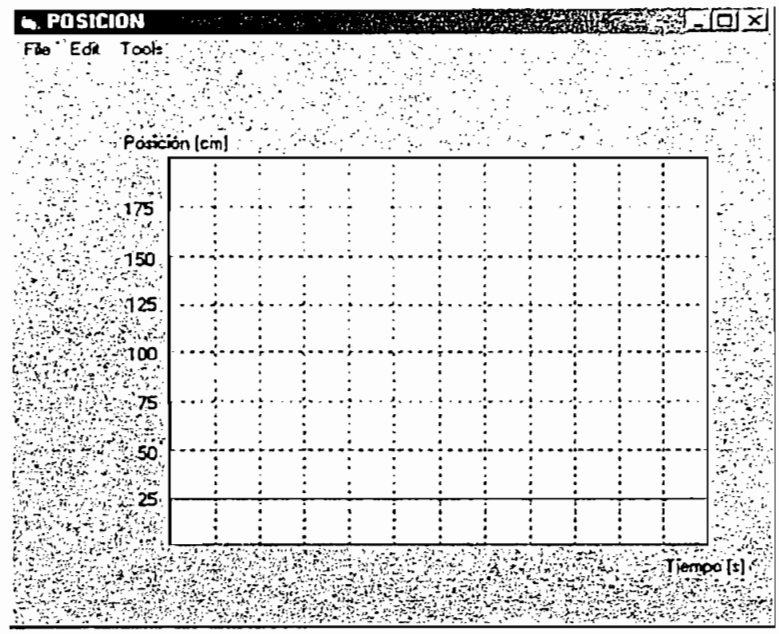


Figura 4.1 Medición a 25 cm

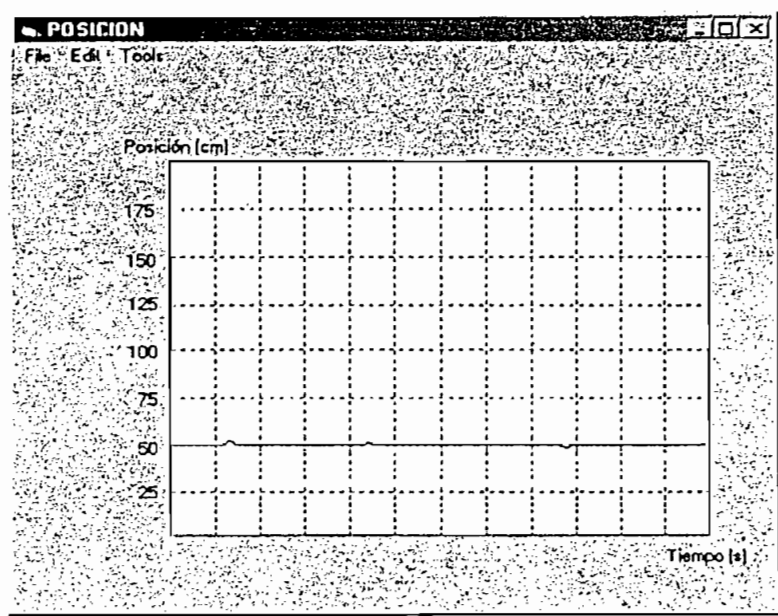


Figura 4.2 Medición a 50 cm

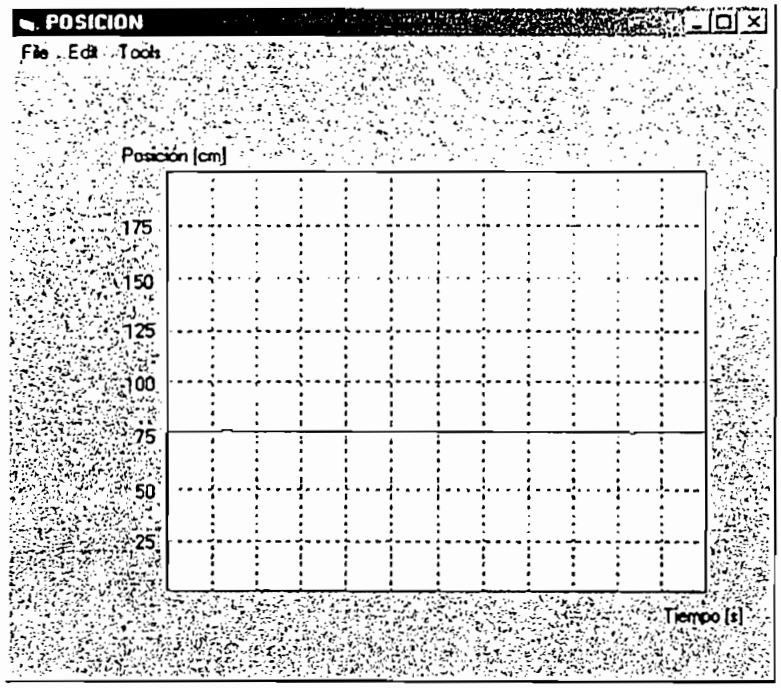


Figura 4.3 Medición a 75 cm

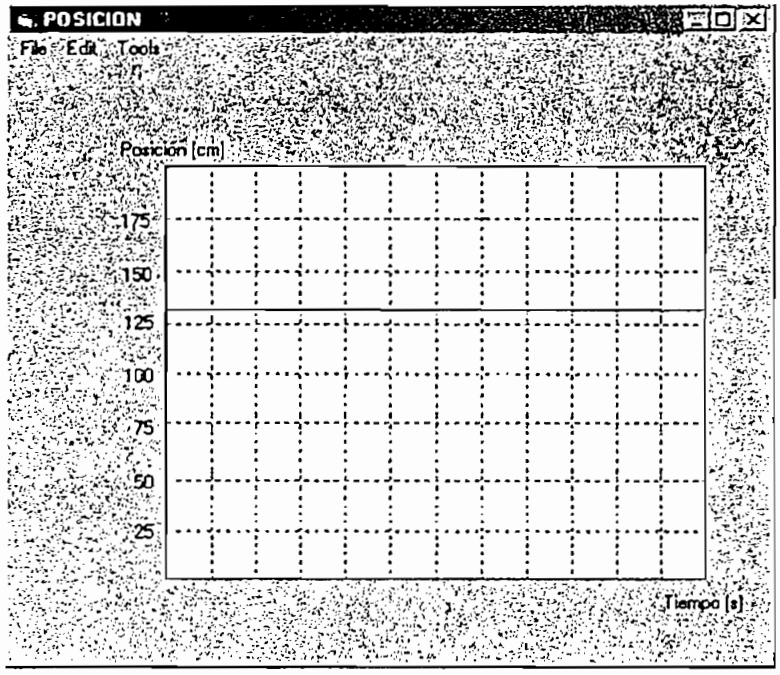


Figura 4.4 Medición a 130 cm

De los gráficos anteriores se puede obtener la Tabla 4.1 en la que se visualizan y resume el valor mínimo, valor máximo y valor medio.

Medición	*Distancia (cm)*	*Valor mínimo*	*Valor máximo*	*Valor medio*	*Error* (%)
1	25	24.98	25.01	24 .99	0.04
2	50	49.97	50.04	49.99	0.02
3	75	74.99	75.01	75	0
4	130	130	130.01	130	0

Tabla 4.1 Resumen de las mediciones para determinar la repetibilidad

De los resultados obtenidos y mostrados en las gráficas y tabla anteriores se puede deducir que la medición de distancia alcanza una exactitud de 99% y repetibilidad de 99%, mucho mejor que los sistemas de medición de distancias por ultrasonido desarrollados en trabajos anteriores.

Luego de comprobado que el sistema es suficientemente confiable y exacto, a continuación se procedió con las pruebas verdaderas.

4.1.2 PRUEBAS Y RESULTADOS REALIZADOS CON EL OBJETO EN MOVIMIENTO

El objetivo principal de este trabajo es obtener las curvas de posición, velocidad y aceleración versus tiempo del movimiento rectilíneo. Para conseguir tal objetivo, como se ha explicado ya en capítulos anteriores, se realizan las mediciones continuas de la posición del objeto en movimiento por medio de la técnica del pulso – eco, ya analizada, obteniéndose de esta manera una secuencia de valores discretos que representa la función de posición del objeto. Luego, a partir de estos datos, se procederá a calcular la primera y segunda derivada para obtener la velocidad y aceleración, respectivamente.

Dado que la solución planteada al problema requiere la derivación de una secuencia discreta en el tiempo, fue necesario implementar un algoritmo matemático que permita realizar esta labor. Este algoritmo fue descrito en el Capítulo 3 sección 3.3.3 y antes de realizar el cálculo de la derivada para las muestras de posición del objeto, se necesitaba comprobar que el algoritmo de derivación realmente funciona y, por tal razón a continuación se presentan las pruebas y resultados realizados sobre tal algoritmo, para comprobar su funcionamiento.

4.1.2.1 Pruebas y resultados para el algoritmo de derivación

En la ecuación 3.2 del Capítulo 3 se muestra la fórmula matemática a usar para implementar el algoritmo de derivación. Para la prueba de funcionamiento de este algoritmo, se realizó un programa en Microsoft Visual Basic 6.0, el mismo que genera un total de 628 valores para una función dada a intervalos constantes de 0.01; así, para esta prueba se utilizó la función sen(x) que se muestra en la Figura 4.5 y, a continuación en la Figura 4.6 y Figura 4.7 se muestran los resultados para el algoritmo de derivación implementado sobre tal función.

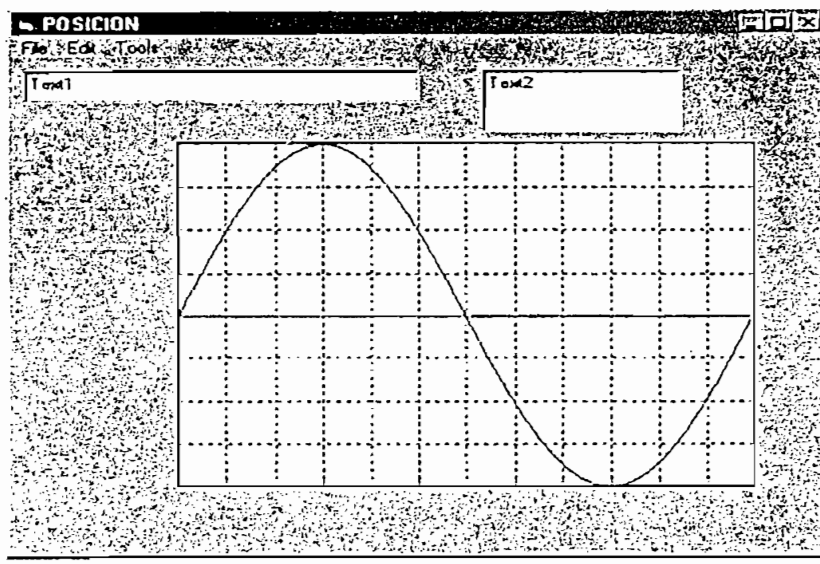


Figura 4.5 Gráfica de la función sen (x) a derivar

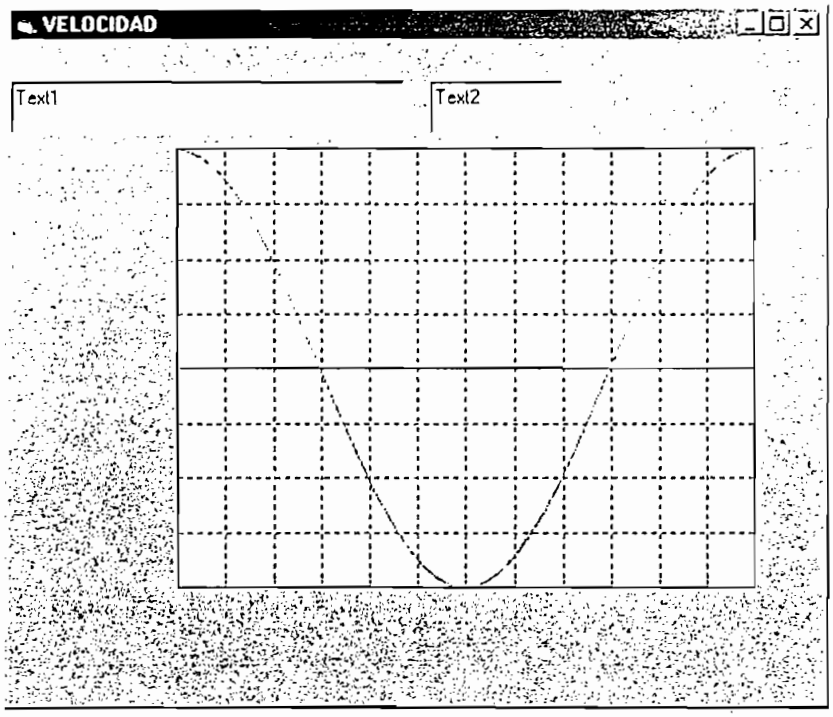


Figura 4.6 Gráfica de la primera derivada de la función sen (x)

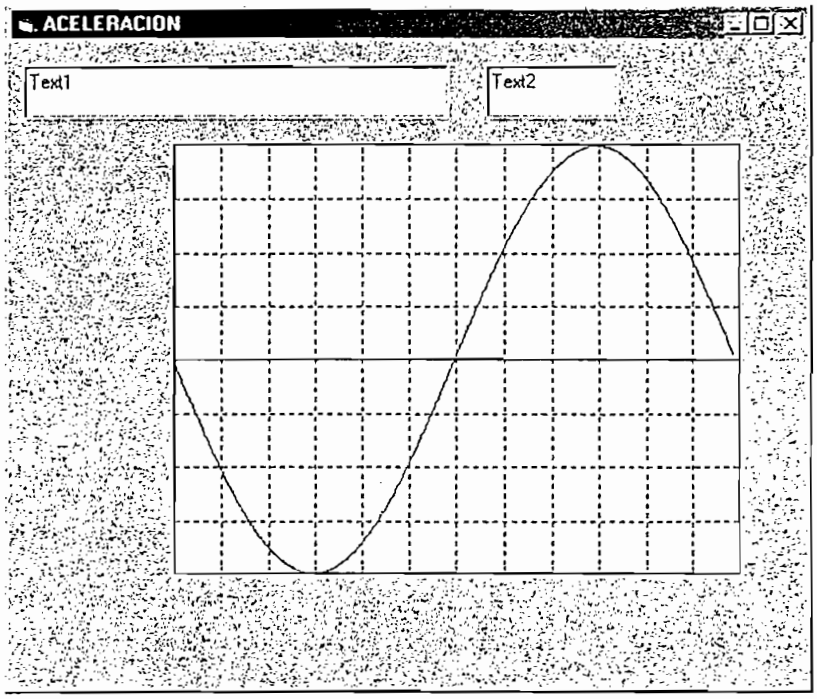


Figura 4.7 Gráfica de la segunda derivada de la función sen (x)

Las anteriores figuras muestran los resultados esperados, concluyéndose de esa manera, que el algoritmo de derivación implementado, realmente funciona y es eficaz.

A continuación se procedió a realizar las pruebas para la medición de la posición del objeto en movimiento rectilíneo y el respectivo cálculo de sus derivadas, comparándolas con los resultados que teóricamente deben esperarse.

4.1.2.2 Pruebas y resultados realizados para el cálculo de la aceleración en base a la medición de distancia

Para estas pruebas se tomaron en cuenta las siguientes consideraciones:

- Se encontrará un valor experimental de aceleración que se contrastará con un valor teórico calculado por la ecuación 1.11 formulada en el Capítulo 1.

- Dado que la ecuación 1.11 implica un fuerza de rozamiento nula, las pruebas se las realiza sobre un módulo en el que prácticamente se elimina la fricción por medio de un colchón de aire.

- Las pruebas se las realiza en un sistema como el que se muestra en la Figura 1.1 y en el que se observa que el objeto en movimiento con masa m_1, sin variación para todas las pruebas, es sometido a una fuerza constante durante toda la trayectoria.

- Esta fuerza constante es originada por un cuerpo de masa m_2 que hala el objeto por medio de una cuerda. Las pruebas se las ejecutó para tres valores diferentes de esta masa.

Prueba 1: m_1=88.78 g y m_2=5.18 g

Con estos datos medidos por medio de una balanza electrónica se obtiene el siguiente valor teórico de aceleración por medio de la ecuación 1.11:

$$\ddot{x} = \frac{m_2 * g}{m_1 + m_2} = a = \frac{5.18g * 9.8 \frac{m}{s^2}}{5.18g + 88.78g} = 0.52 m/s^2$$

Una vez obtenido el valor teórico de aceleración se obtiene la función teórica de velocidad integrando la función aceleración, que para este caso es un valor numérico constante y, tomando en cuenta que el objeto parte del reposo (velocidad inicial cero) este cálculo se realiza de la siguiente manera:

$$v = \int a * dt = at + K \qquad Como \quad v_o = 0 \quad se \quad tiene \quad que \quad K = v_o = 0$$
$$\Rightarrow v = at + v_o = at = 0.52t$$

Con la función teórica de velocidad se puede realizar un integración para obtener la función de posición del objeto, asumiendo que este objeto puede partir de una posición inicial cualquiera de la siguiente manera:

$$d = \int v * dt = \int at * dt = {at^2}/{2} + K \qquad Como \quad puede \quad ser \quad d_o \neq 0 \Rightarrow K = d_o$$
$$\Rightarrow d = {at^2}/{2} + d_o = 0.26t^2 + d_o$$

Ahora, para la obtención del valor experimental de las funciones de posición, velocidad y aceleración se parte de las medidas realizadas de distancia del objeto en movimiento por medio del hardware diseñado e implementado.

A continuación se muestran las gráficas teóricas con color azul y las experimentales con violeta:

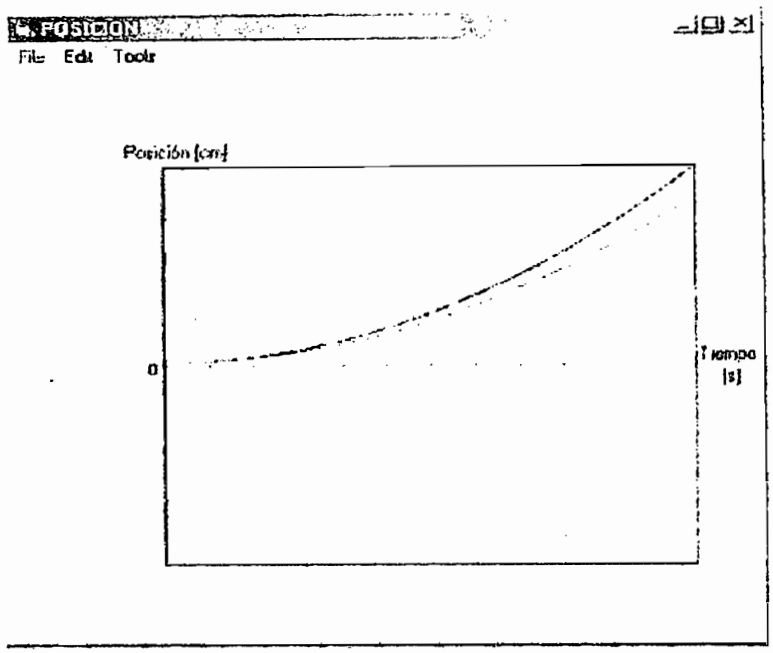


Figura 4.8 Gráfica de las curvas teórica y experimental de posición

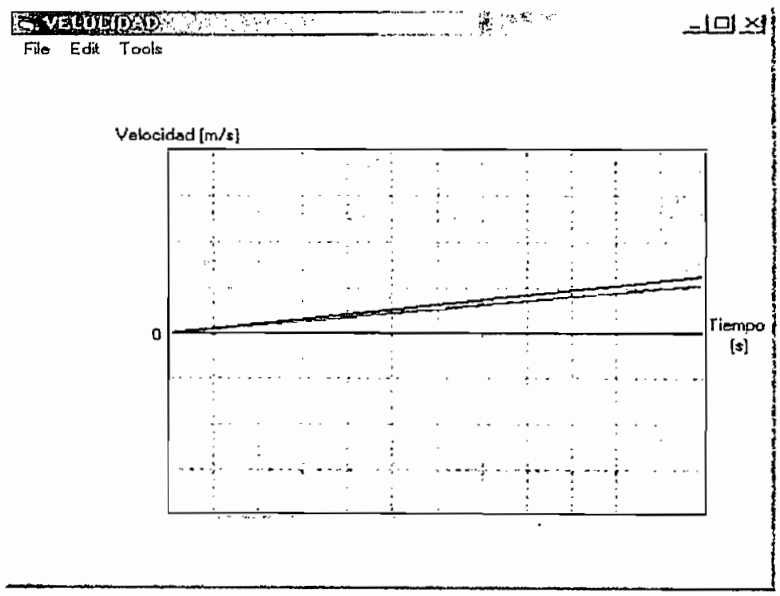


Figura 4.9 Gráfica de las curvas teórica y experimental de velocidad

De la tabla anterior se puede observar que el método usado en el presente proyecto para el cálculo de los principales parámetros que rigen el movimiento rectilíneo de un objeto tiene un error promedio del 8.32%. Para concluir con este Capítulo se menciona las pruebas de funcionamiento del programa hecho en Visual Basic.

4.1.3 PRUEBAS Y RESULTADOS DE LA INSTALACIÓN Y FUNCIONAMIENTO DEL SOFTWARE

Antes de realizar la instalación del software se debe tener en cuenta la siguiente recomendación:

- Se requiere que el sistema operativo sobre el cual se ejecute el programa no sea una versión anterior a Windows 98 SE, por cuanto aquellas no tienen incorporado el suficiente soporte USB y, más aun, para la clase de dispositivo USB sobre el cual se diseñó este proyecto
- Se requiere también una máquina que posea al menos un puerto USB
- El programa puede correr sobre un PC Pentium II o superior
- El software requiere un espacio libre en el disco duro del PC de al menos 20MB y 32 MB de RAM
- Es indispensable que la máquina tenga una buena conexión a tierra o de lo contrario el dispositivo USB no funcionará correctamente. Este problema se presentó en el laboratorio de Física, lugar donde el módulo permanecerá. En este lugar, la estación ni siquiera arrancó por malas conexiones eléctricas

Además de estos requerimientos téngase en cuenta los siguiente:

- El programa desarrollado debe ser instalado por separado en dos partes.
- La primera parte requiere la instalación del Control ActiveX HIDCOMM de libre uso y sin cargo económico.
- La segunda parte requiere la instalación propia del software para el medidor ultrasónico.

4.1.4 ANÁLISIS DE COSTOS

Uno de los objetivos de este proyecto era obtener un producto de bajo costo, fácil consecución y sobre todo, de tecnología propia. Pensando en esto a continuación se presenta la Tabla 4.2 con el resumen de costos implicados en este proyecto.

EQUIPO	COSTO MÁXIMO (USD)
Microcontrolador PIC16C745	11.00
Sensor ultrasónico POLAROID 6500	19.50
Circuito Integrado TL852	5.00
Transformador flyback	5.00
Conector USB tipo B	0.70
Cable USB	2.00
Circuito Impreso y partes mecánicas	30.00
Circuito Integrado LM358	0.80
Diodos resistencias y capacitores	5.00
Zócalos	2.00
Gastos de importación	50.00
TOTAL	**131.00**

Tabla 4.2 Resumen de costos del equipo

El proyecto se realizó en 10 meses, por lo que, considerando un salario de 400 dólares por mes por cada ingeniero, el costo de ingeniería asciende a 4000 dólares; teniéndose un valor total del proyecto de 4131 dólares. Dado que la idea primordial sería la venta de estos equipos a instituciones educativas de nivel medio para la enseñanza de la física, sería necesario la fabricación en serie de tales dispositivos por lo que los costos de cada módulo serían mucho menores a los expuestos anteriormente .

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

La elaboración e investigación del proyecto "Diseño y construcción de un módulo didáctico para el análisis del movimiento rectilíneo por medio del pórtico USB" y sus pruebas de funcionamiento generan conclusiones importantes que se analizan en el presente capítulo.

5.1 CONCLUSIONES

El principal problema que se tuvo con los sensores ultrasónicos fue la repetibilidad y exactitud que estos ofrecían al medir una distancia fija; la gran mayoría de sensores de fácil consecución que se obtuvieron no cumplieron con este objetivo. Principalmente se buscó una buena repetibilidad del sensor debido a que, al no existir ésta, se presentaban discontinuidades en las formas de las curvas de la posición, la misma que al derivarla generaban muchos pulsos en la consecución de la velocidad y peor aún en la obtención de la aceleración.

El sensor Ultrasónico Polaroid 6500 fue la mejor opción, debido a que permite medir distancias con precisión de hasta un milímetro, con buena exactitud y repetibilidad debido al alto voltaje de excitación y frecuencia de trabajo comparado con otros similares; además, sus fabricantes recomiendan el uso del circuito integrado TL852 que resulta de gran utilidad para el acondicionamiento de cualquier sensor de ultrasonido de 50 KHz, el mismo que evita la contaminación de la señal recibida con ruido. Este chip permite controlar la ganancia de la misma en forma cuadrática, ya que la señal ultrasónica emitida es atenuada con el inverso del cuadrado de la distancia que ésta recorre.

Las pruebas realizadas sobre los sensores ultrasónicos de esta marca y de otras permiten llegar a la conclusión que la medida de distancia por el método del pulso – eco es bastante dependiente de la forma en que se encuentre colocado el sensor y los objetos existentes a su alrededor. Debido a esto es necesario e indispensable, como paso previo, a la toma de las medidas de posición, moverlo hasta obtener el valor más exacto repetible posible.

El protocolo USB aunque es un protocolo de gran complejidad ofrece grandes prestaciones que pueden ser utilizadas en la implementación de sistemas de control y adquisición de datos. Además el bus serie universal USB es ahora prácticamente un estándar en todo computador tanto personal como portátil, razón por la cual toda aplicación que originalmente trabajaba con los clásicos puertos seriales y paralelo están siendo reemplazadas por el USB y, pese a sus limitaciones de distancia de transmisión de información, ésta puede ser superada usando drives que cambien de la interfaz USB a otro tipo de interfaz, que permita transmisión a mayores distancias y viceversa.

El manejo de las funciones API de Windows permite a los desarrolladores de aplicaciones USB diseñarlas e implementarlas bajo diferentes lenguajes de programación como pueden ser Visual Basic, Visual C++, Power Builder, LabView, etc. Fue importante el uso de tales funciones para este proyecto, pues son una alternativa que permite establecer comunicación con el hardware de un PC desde cualquiera de esto lenguajes de programación. En el presente proyecto fue de utilidad el uso del lenguaje de programación Visual Basic de Microsoft, pues ofrece un ambiente amigable y sencillo de programación, aparte el código que se puede desarrollar sobre esta plataforma es bastante fácil de usar, lo que conlleva a un tiempo de implementación mucho menor que con otras plataformas.

Es una buena práctica diseñar un sistema de adquisición de datos como un dispositivo USB que pertenezca a la clase genérica HID. Con esto se evita la construcción de un driver para tal dispositivo, pues ésta es una tarea con alto grado de dificultad para realizarla, debido a que es necesario tener una buena experiencia en programación y conocimiento de manejo de drivers.

Un dispositivo HID puede tan solo utilizar dos tipos de transferencias, las de control y las interrupción, además de estar limitado por la alta velocidad, éste es un punto que se debe tomar muy en cuenta para los diseñadores de dispositivos, pues se puede ganar tiempo evitando la construcción de un driver, pero se pierde en velocidad.

En promedio se tiene un porcentaje de error del 8.32% del valor medido de la aceleración con respecto al valor teórico. Este porcentaje de error se puede deber a la siguiente razón:

- El método usado para la determinación únicamente de la posición del objeto es la técnica pulso – eco ultrasónico la cual, como ya se ha explicado, requiere la toma de medidas a intervalos de tiempo desiguales. Este proyecto por las condiciones del mismo no puede tomar medidas de posición a intervalos constantes inevitablemente por dos factores: dependen del tiempo de vuelo del pulso de ultrasonido emitido que variará conforme varía la distancia del objeto y, depende del intervalo de tiempo transcurrido entre cada pedido de transmisión USB de parte del host. Esto origina que el algoritmo de derivación produzca un porcentaje de error en sus cálculos, ya que el mismo necesita como uno de sus parámetros un intervalo entre muestras constante.

5.2 RECOMENDACIONES

Se podría sugerir para uso de este proyecto sensores ultrasónicos de mas alta frecuencia debido a que estos presentarían un longitud de onda pequeña, pero su alta frecuencia implica mayor atenuación de su onda. Esto se pudo deducir de las pruebas realizadas; además, otra limitación importante en cuanto a la adquisición de los sensores, es su precio y su empaquetado que no resiste ambientes hostiles.

Para una perfecta instalación de la aplicación en una PC se recomienda, primeramente instalar el control activex HIDCOM seguido de la aplicación, adicionalmente se recomienda tener una conexión a tierra en el conector en el que la PC será enchufada, de no ser así el dispositivo no funcionará correctamente, esto debido a que la alimentación que recibe el dispositivo, es tomada de las líneas del USB y ésta viene de la fuente del computador.

Se recomienda quizá usar este proyecto como una base para realizar mejoras al sistema de adquisición de datos. Por ejemplo, este proyecto pudiera ser usado para determinar una aproximación al coeficiente de rozamiento dinámico del módulo sobre el que se desplaza el objeto una vez comprobado que los datos de aceleración obtenidos experimentalmente sean correctos.

Se recomienda seguir en la investigación de métodos para determinar las medidas de distancias por medio de ultrasonido de una manera más precisa y milimétrica por medio de técnicas de modulación. Por ejemplo, durante el proceso de investigación para la determinación de un método factible para la medición de distancias, por medio de ultrasonido, se encontró un método que implica el envío de pulsos ultrasónicos modulados siguiendo uno de dos patrones denominados: secuencias Golay o secuencias Barker.

Otra perspectiva desde la cual se pudiera lograr el objetivo principal de este proyecto, es el uso del principio físico denominado efecto Doppler. Mediante este principio se pueden obtener medidas directas de velocidad y, se recomendaría como otro tipo de solución al problema planteado, una investigación de la factibilidad del uso de este principio para conseguir mejorar el actual sistema y, talvez se podrían juntar ambos métodos para obtener todavía una mejor solución.

Este trabajo puede ser de mucha utilidad para futuros trabajos de investigación con respecto al pórtico USB, se deja la teoría necesaria como para desarrollar otro tipo de aplicaciones. Por ejemplo algo que en la actualidad está siendo bastante investigado es el diseño de periféricos USB, que puedan trabajar bajo un sistema operativo diferente al de Windows como por ejemplo Linux.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

- **AXELSON, Jan.** USB Complete, Segunda Edición, Lakeview Research. USA.2002

- **MICROCHIP TECHNOLOGY INC.** Data Sheet PIC16C745/765, 8-Bit CMOS Microcontrollers whit USB. USA. 2000.

- **COMPAQ, INTEL, MICROSOFT, NEC.** Universal Serial Bus Specification, Revision 1.1.1998

- **USB IMPLEMENTERS' FORUM.** Device Class Definition for Human Interface Devices (HID). Version 1.11. 1996-2001

- **PERRY,Gray.** Aprendiendo Visual Basic 6.0. Primera Edición. Editorial Prentice Hall. México. 1999

- **ANGULO USATEGUI, José; ANGULO MARTINEZ, Ignacio.** Microcontroladores PIC, Diseño Práctico de Aplicaciones. Segunda Edición, Editorial McGraw Hill. España. 1999

- **COUGHLIN, Robert; DRISCOLL, Frederick.** Amplificadores Operacionales y Circuitos Integrados. Quinta Edición. Editorial Prentice Hall. México. 1999

- **TORO, José.** Cálculo Vectorial. Primera Edición. Escuela Politécnica Nacional. Quito. 1999

- **SILVA, Mauricio.** Diseño y construcción de un módulo didáctico y software para la simulación en un PC de un medidor de nivel de líquidos en tanques abiertos por medio del ultrasonido. Escuela Politécnica Nacional. Quito. Octubre 1999

- **FLORES, José.** Medidor ultrasónico de nivel de líquido en movimiento en un canal. Escuela Politécnica Nacional. Quito.Julio 1990

REFERENCIAS ADICIONALES

- http://www.usb.org
- http://www.microchip.com
- http://www.lvr.com
- www.microsoft.com
- http://www.acroname.com
- http://www.senscomp.com
- http://www.jameco.com
- http://www.digikey.com

ANEXOS

ANEXO A
DESCRIPTORES USB

DESCRIPTORES USB

Descriptor de dispositivo (Device descriptor)

Offset (Decimal)	Campo	Tamaño (bytes)	Descripción
0	bLongitud	1	Tamaño del descriptor en bytes
1	bTipoDescriptor	1	La constante DISPOSITIVO (01h)
2	bcdUSB	2	Versión de la especificación USB en BCD
4	bClaseDispositivo	1	Código de clase
5	bSubclaseDispositivo	1	Código de subclase
6	bProtocoloDispositivo	1	Código de protocolo
7	bMaxTamañoPaquete (0)	1	Máx. tamaño del paquete para el endpoint 0
8	idVendedor	2	ID del vendedor
10	idProducto	2	ID del Producto
12	bcdDispositivo	2	Versión del dispositivo en BCD
14	iFabricante	1	Indice a un descriptor de texto para el fabricante
15	iProducto	1	Indice a un descriptor de texto para el producto
16	iNúmeroSerial	1	Indice a un descriptor de texto conteniendo el número serial
17	bNumConfiguraciones	1	Número de posibles configuraciones

Tabla A.1Descriptor de dispositivo

Tiene información básica acerca del dispositivo. Es el primer descriptor que el host lee al agregarse un dispositivo e incluye información que el host necesita, así se puede recuperar información adicional desde el dispositivo.

El descriptor tiene 14 campos, que en la Tabla A.1se enlistan en el orden en que ellos están presentes en el descriptor. El descriptor incluye información acerca del descriptor mismo, el dispositivo, sus configuraciones y sus clases.

Descriptor calificador de dispositivo (Device_qualifier descriptor)

Dispositivos que manejan median y alta velocidad deben tener un descriptor de este tipo. Si el dispositivo cambia de velocidad, algunos campos en el descriptor de dispositivo pueden cambiar.

Offset (Decimal)	Campo	Tamaño (bytes)	Descripción
0	bLongitud	1	Tamaño del descriptor en bytes
1	bTipoDescriptor	1	La constante DEVICE_QUALIFIER (06h)
2	bcdUSB	1	Versión de la especificación USB en BCD
4	bClaseDispositivo	1	Código de clase
5	bSubclaseDispositivo	1	Código de subclase
6	bProtocoloDispositivo	1	Código de protocolo
7	bMaxTamañoPaquete (0)	1	Máx. tamaño del paquete para el endpoint 0
8	bNumConfiguraciones	1	Número de posibles configuraciones
9	Reservado	1	Para uso futuro

Tabla A.2 Descriptor calificador de dispositivo

El descriptor calificador de dispositivo almacena los valores para uso de estos campos en la velocidad que no esté actualmente en uso. El contenido de los campos en los descriptores de dispositivo y calificador de dispositivo se intercambian dependiendo de la velocidad que este seleccionada. El descriptor se compone de 9 campos, como se describe en la Tabla A.2 en el orden en el que ocurren.

Este descriptor incluye información acerca del descriptor mismo, el dispositivo, sus configuraciones y sus clases. Los campos son los mismos como los del descriptor de dispositivo. La única diferencia es que ellos describen el dispositivo en la velocidad que no esté actualmente en uso.

El ID del vendedor y producto, número de versión del dispositivo y los textos describiendo al fabricante, producto y número de serie no cambian cuando la velocidad cambia así este descriptor no los incluye.

Descriptor de configuración (Configuration descriptor)

Offset (Decimal)	Campo	Tamaño (bytes)	Descripción
0	bLongitud	1	Tamaño del descriptor en bytes
1	bTipoDescriptor	1	La constante CONFIGURACIÓN (02h)
2	wLongitudTotal	1	Tamaño de todos los datos regresados por esta configuración en bytes
4	bNúmeroInterfaces	1	Número de interfaces que la configuración soporta
5	bValorConfiguración	1	identificador para los pedidos Setear_Configuración y Conseguir_Configuración
6	iConfiguración	1	Indice a un descriptor de texto para la configuración
7	bmAtributos	1	Settings para auto/bus alimentación y despertar remoto
8	MaxPotencia	1	Requerido para alimentación desde el bus, expresados como (max. mA/2)

Tabla A.3 Descriptor de configuración

Después de recuperar el descriptor de dispositivo, el host puede recuperar los descriptores de configuración, interfaz y endpoint. Cada dispositivo tiene al menos un descriptor de configuración que describa las características y habilidades del dispositivo.

A menudo una simple configuración es suficiente, pero un dispositivo con múltiples usos y modos puede soportar múltiples configuraciones; aunque solamente una configuración está activa a la vez. Cada configuración requiere un descriptor. El descriptor de configuración contiene información acerca del uso de potencia del dispositivo y el número de interfaces soportadas. Cada descriptor de configuración tiene descriptores subordinados, incluyendo uno o más descriptores de interfaz y descriptores de endpoint opcionales. El host selecciona una configuración con el pedido Setear_Configuración (Set_Configuration), y lee la configuración actual usando el pedido Conseguir_Configuración (Get_Configuration). El descriptor tiene nueve bytes en ocho campos. La Tabla A.3 enumera los campos en el orden en el que aparecen en el descriptor. Los campos contienen información acerca del descriptor mismo, la configuración, y el uso de potencia cuando se alimenta desde el bus en esa configuración. Para muchas configuraciones algunos campos no aplican.

Descriptor de interfaz (Interface descriptor)

Aunque el término interfaz puede describir al USB como un todo, al referirnos al dispositivo y sus descriptores, interfaz significa un conjunto de endpoints usados por una función o característica del dispositivo. El descriptor de interfaz de una configuración contiene información acerca de los endpoints que la interfaz soporta.

Cada configuración debe soportar una interfaz, y para muchos dispositivo, una es suficiente. Pero una interfaz puede tener múltiples interfaces que son activas a la misma vez, además de múltiples interfaces mutuamente excluyentes. Cada interfaz tiene su propio descriptor de interfaz y un descriptor de endpoint subordinado para cada endpoint soportado por la interfaz. Un dispositivo con una configuración que tiene múltiples interfaces que son activas a la misma vez es un dispositivo compuesto. El host carga un driver para cada interfaz. El descriptor de interfaz tiene 9 bytes en 9 campos, que se muestran en el orden en el que ocurren en el descriptor en la Tabla A.4. Muchos dispositivos no necesitan todos los campos, tales como los que habilitan protocolos alternos.

Offset (Decimal)	Campo	Tamaño (bytes)	Descripción
0	Blongitud	1	Tamaño del descriptor en bytes
1	bTipoDescriptor	1	La constante INTERFACE (04h)
2	bNúmeroInterface	1	Número identificando esta interfaz
3	bSettingAlterno	1	Valor usado para seleccionar un setting alterno
4	bNumEndpoints	1	Número de endpoints soportados, excepto endpoint 0
5	bClaseInterface	1	Código de clase
6	bSubclaseInterface	1	Código de subclase
7	bProtocoloInterface	1	Código de protocolo
8	Interface	1	Índice a un descriptor de texto para la interfaz

Tabla A.4 Descriptor de interfaz

Descriptor de Endpoint (Endpoint descriptor)

Cada endpoint especificado en un descriptor de interfaz debe tener un descriptor de endpoint.

Offset (Decimal)	Campo	Tamaño (bytes)	Descripción
0	BLongitud	1	Tamaño del descriptor en bytes
1	bTipoDescriptor	1	La constante ENDPOINT (05h)
2	bDirecciónEndpoint	1	Número y dirección de endpoint
3	BmAtributos	1	Tipo de transferencia soportado
4	wMaxTamañoPaquete	1	Máximo tamaño soportado del paquete
5	Bintervalo	1	Max. latencia / intervalo de polling / frecuencia NAK

Tabla A.5 Descriptor de endpoint

Los endpoints isocrónicos de mediana velocidad tienen muy poco de nuevo. El descriptor de endpoint puede especificar sincronización (campo bmAtributos), y el intervalo puede ser mayor que 1 milisegundo (campo bIntervalo).

En los descriptores 1.x, estos bits son predeterminados a 0 (no sincronización) y 1 (un milisegundo).

Descriptor	Campo	Cambio
Dispositivo	bcdDispositivo	Cambiar a 0200h
Endpoint	bmAtributos	Solo isocrónico : bits 3..2 son un tipo de sincronización, bits 5..4 son tipo usage
	bInterval	Solo isocrónico : intervalo es $2^{bIntervalo-1}$ milisegundos en lugar milisegundos
	wMaxTamañPaquet	Solo isocrónico : debe ser 0 en la configuración predeterminada

Tabla A.7 Cambios para que los descriptores 1.x cumplan con 2.0

ANEXO B

GLOSARIO DE TÉRMINOS USB

GLOSARIO DE TÉRMINOS USB

En este glosario se definen los términos más usados en el universo del USB, de tal manera que sean entendibles para la mayoría de los lectores; recalcando que muchos de los términos utilizados en la especificación no tienen una traducción directa y coherente al español por lo que en el presente anexo se han usado los términos originales en inglés. Aunque no se definen completamente todos, se presentan los que son de uso más común en este proyecto. Si alguien desea conocer más a fondo la terminología dada por la especificación USB, puede remitirse a la misma en la página web del Foro de Implementadores USB www.usb.org en la sección de desarrolladores.

ACK.- Paquete de Handshake que indica un reconocimiento positivo del dato enviado o recibido.

Ancho de Banda.- La cantidad de datos transmitidos por unidad de tiempo, típicamente en bytes por segundo(B/s) o bits por segundo(b/s).

Controlador Host.- La interfaz USB del host.

CRC.- Son bits para chequeo de error en una transmisión de datos que se calculan mediante un algoritmo matemático llamado Chequeo de redundancia cíclica.

Dispositivo.- Una entidad lógica o física que realiza una función.

Dispositivo de baja potencia (low – power).- La especificación define un dispositivo de baja potencia (low – power) como un dispositivo que drena hasta 100 miliamperios del bus

Dispositivo de alta potencia (high – power).- La especificación define un dispositivo de alta potencia (high – power) como un dispositivo que puede drenar hasta 500 miliamperios del bus.

Driver USB (USBD).- El software residente en el host responsable de proveer servicios comunes a clientes que están manipulando una o más funciones en uno o más controladores de un host.

Downstream.- La dirección de flujo de datos desde el host hacia afuera. Un puerto downstream es el puerto sobre el hub en el que se conectarán el resto de dispositivos. Puertos downstream reciben tráfico de datos upstream.

Endpoint.- El endpoint es un buffer de almacenamiento físico que se encuentra en el periférico USB y sirve para guardar múltiples bytes; el host no tiene endpoints.

Final de paquete EOP (End of packet).- Señal que retorna el bus al estado desocupado (idle) en preparación para el siguiente paquete.

Etapa.- Una parte de la secuencia que compone una transferencia de control; etapas incluyen la etapa Setup, la etapa de Datos, y la etapa de Estado.

Fase.- Un tipo de paquete que puede ser un paquete token o un paquete de datos o un paquete handshake.

Función.- Un dispositivo USB que provee una capacidad al host, tal como un micrófono digital o parlante.

Host.- El computador que controla la interfaz USB y donde está colocado el controlador host USB. Esto incluye la plataforma del hardware del host (CPU, bus, etc.) y el sistema operativo en uso.

Hub.- Un dispositivo USB que provee puertos de conexión adicionales al USB.

Hub raíz.- Un hub USB directamente enlazado al controlador del Host.

NAK.- Paquete Handsahke indicando un reconocimiento negativo.

Paquete.- Una cantidad de datos organizados en un grupo para transmisión. Los paquetes contienen por lo general tres elementos: información de control, el dato a ser transferido y bits para detección y corrección de errores.

Paquete Handshake.- Un paquete que reconoce o rechaza una condición específica.

Paquete ID (PID).- Un campo en un paquete USB que indica el tipo de paquete, el formato de los datos y el tipo de detección de error aplicado al paquete.

Paquete token.- Un tipo de paquete que identifica cual transacción se esta realizando en el bus.

Pipe.- No es un objeto físico; sino que es una asociación entre el endpoint de un dispositivo y el software del controlador del host.

Polling.- Es el método usado por el host para preguntar a múltiples dispositivos uno a la vez, si tiene datos a transferir.

Software Cliente.- Software residente en el host que interactúa con el Software del Sistema USB que organiza la transferencia de datos entre una función y el host. El cliente es a menudo el proveedor y consumidor de datos para la transferencia.

Software de dispositivo.- Software que es responsable del uso de un dispositivo USB; a este software se le conoce también como firmware del dispositivo. Este software puede o no ser responsable de la configuración del dispositivo para su uso.

Inicio de trama SOF (Start of frame).- La primera transacción en cada trama. Un SOF permite al endpoint identificar el inicio de la trama y sincronizar el reloj interno del endpoint con el host.

SOP (Start of packet) .- Inicio de paquete.

Trama.- El tiempo desde el inicio de un token SOF al inicio del token SOF subsiguiente; consiste de una serie de transacciones.

Transacción.- La entrega de servicio para un endpoint, consiste de un paquete token, un paquete opcional de datos y un paquete opcional de handshake . Estos paquetes específicos son requeridos o permitidos en base al tipo de transacción .

Transferencia.- Una o mas transacciones de bus para mover información entre un software cliente y su función.

Upstream.- Dirección del flujo de datos hacia el host. Un puerto upstream es el puerto eléctricamente conectado con el host y que genera tráfico upstream desde el hub.

ANEXO C

DATOS TÉCNICOS DEL MICROCONTROLADOR PIC 16C745

PIC16C745/765

8-Bit CMOS Microcontrollers with USB

Devices included in this data sheet:

- PIC16C745
- PIC16C765

Microcontroller Core Features:

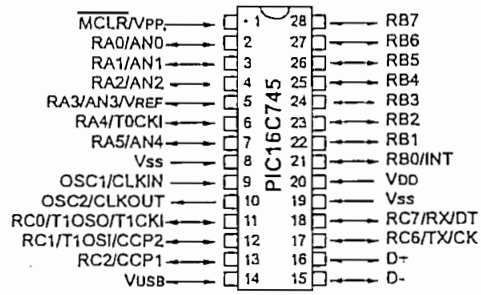
- High-performance RISC CPU
- Only 35 single word instructions

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

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

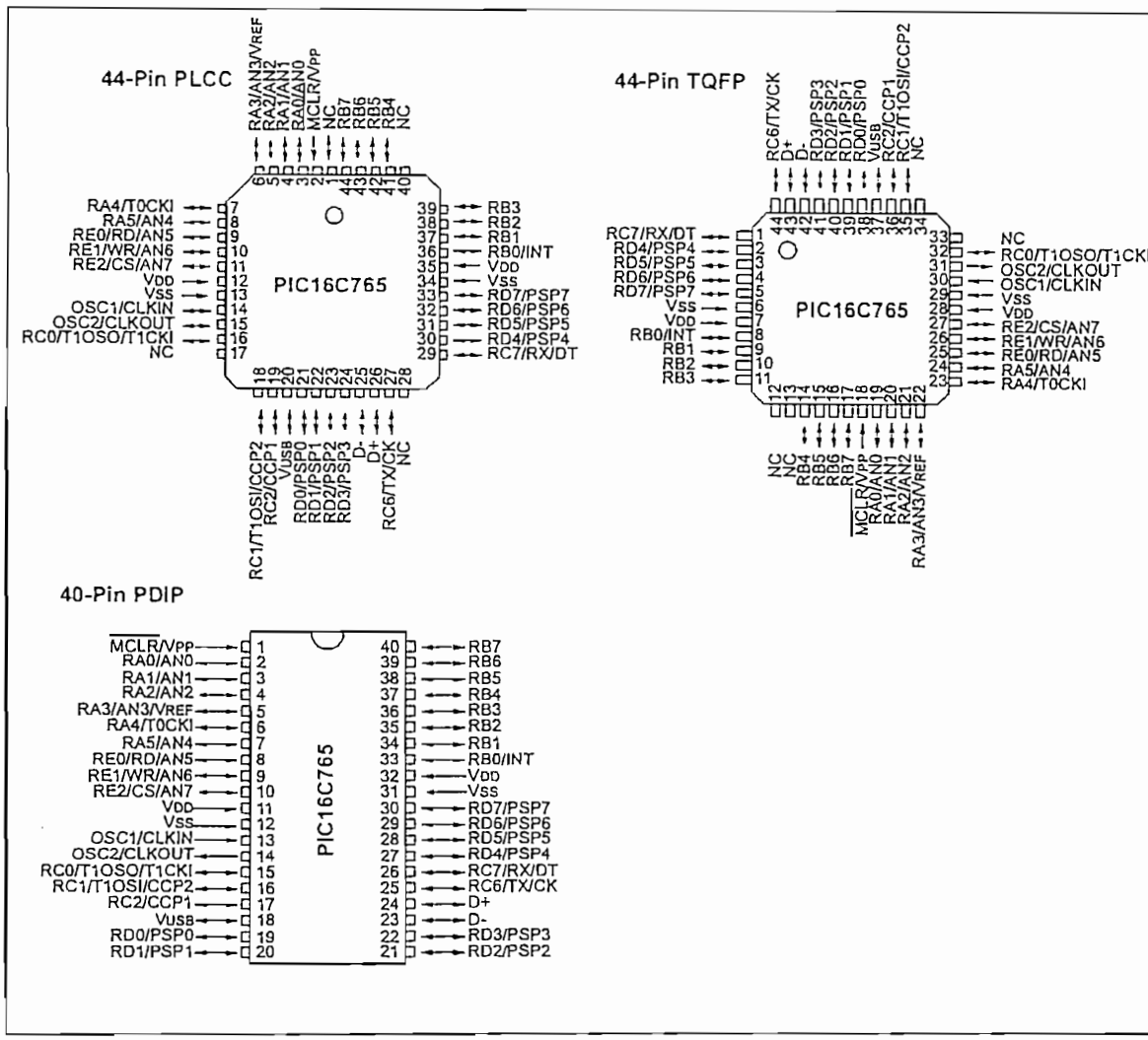
Pin Diagrams

28-Pin DIP, SOIC



Peripheral Features:

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



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

10.0 UNIVERSAL SERIAL BUS

10.1 Overview

This section introduces a minimum amount of information on USB. If you already have basic knowledge of USB, you can safely skip this section. If terms like Enumeration, Endpoint, IN/OUT Transactions, Transfers and Low Speed/Full Speed are foreign to you, read on.

USB was developed to address the increased connectivity needs of PC's in the PC 2000 specification. There was a base requirement to increase the bandwidth and number of devices, which could be attached. Also desired were the ability for hot swapping, user friendly operation, robust communications and low cost. The primary promoters of USB are Intel, Compaq, Microsoft and NEC.

USB is implemented as a Tiered Star topology, with the host at the top, hubs in the middle, spreading out to the individual devices at the end. USB is limited to 127 devices on the bus, and the tree cannot be more than 6 levels deep.

USB is a host centric architecture. The host is always the master. Devices are not allowed to "speak" unless "spoken to" by the host.

Transfers take place at one of two speeds. Full Speed is 12 Mb/s and Low Speed is 1.5 Mb/s. Full Speed covers the middle ground of data intensive audio and compressed video applications, while low speed supports less data intensive applications.

10.1.1 TRANSFER PROTOCOLS

Four transfer protocols are defined, each with attributes. For full speed, choose from the following:

- Isochronous Transfers, meaning equal time, guarantee a fixed amount of data at a fixed rate. This mode trades off guaranteed data accuracy for guaranteed timeliness. Data validity is not checked because there isn't time to re-send bad packets anyway and the consequences of bad data are not catastrophic.
- Bulk Transfers are the converse of Isochonous. Data accuracy is guaranteed, but timeliness is not.

For full speed or low speed, choose from the following:

- Interrupt Transfers are designed to communicate with devices which have a moderate data rate requirement. Human Interface Devices like keyboards are but one example. For Interrupt Transfers, the key is the desire to transfer data at regular intervals. USB periodically polls these devices at a fixed rate to see if there is data to transfer.
- Control Transfers are used for configuration purposes.

10.1.2 FRAMES

Information communicated on the bus is grouped in a format called Frames. Each Frame is 1 ms in duration and is composed of multiple transfers. Each transfer type can be repeated more than once within a frame.

10.1.3 POWER

Power has always been a concern with any device. With USB, 5 volt power is now available directly from the bus. Devices may be self-powered or bus-powered. Self-powered devices will draw power from a wall adapter or power brick. On the other hand, bus-powered devices will draw power directly from the USB bus itself. There are limits to how much power can be drawn from the USB bus. Power is expressed in terms of "unit loads" (≤100 mA). All devices, including Hubs, are guaranteed at least 1 unit load (low power), but must negotiate with the host for up to 5 unit loads (high power). If the host determines that the bus as currently configured cannot support a device's request for more unit loads, the device will be denied the extra unit loads and must remain in a low power configuration.

10.1.4 END POINTS

At the lowest level, each device controls one or more endpoints. An endpoint can be thought of as a virtual port. Endpoints are used to communicate with a device's functions. Each endpoint is a source or sink of data. Endpoints have both an In and Out direction associated with it. Each device must implement endpoint 0 to support Control Transfers for configuration. There are a maximum of 15 endpoints available for use by each full speed device and 6 endpoints for each slow speed device. Remember that the bus is host centric, so In/Out is with respect to the host and not the device.

10.1.5 ENUMERATION

Prior to communicating on the bus, the host must see that a new device has been connected and then go through an "enumeration process". This process allows the host to ask the device to introduce itself, and negotiate performance parameters, such as power consumption, transfer protocol and polling rate. The enumeration process is initiated by the host when it detects that a new device has attached itself to the bus. This takes place completely in the background from the application process.

10.1.6 DESCRIPTORS

The USB specification requires a number of different descriptors to provide information necessary to identify a device, specify its endpoints, and each endpoint's function. The five general categories of descriptors are Device, Configuration, Interface, End Point and String.

The Device descriptor provides general information such as manufacturer, product number, serial number, USB device class the product falls under, and the number of different configurations supported. There can only be one Device descriptor for any given application.

The Configuration descriptor provides information on the power requirements of the device and how many different interfaces are supported when in this configuration. There may be more than one configuration for each device, (i.e., a high power device may also support a low power configuration).

The Interface descriptor details the number of endpoints used in this interface, as well as the class driver to use should the device support functions in more than just one device class. There can only be one Interface descriptor for each configuration.

The Endpoint descriptor details the actual registers for a given function. Information is stored about the transfer types supported, direction (In/Out), bandwidth requirements and polling interval. There may be more than one endpoint in a device, and endpoints may be shared between different interfaces.

Many of the four descriptors listed above will reference or index different String descriptors. String descriptors are used to provide vendor specific or application specific information. They may be optional and are encoded in "Unicode" format.

10.1.7 DEVICE CLASSES/CLASS DRIVERS

Operating systems provide drivers which group functions together by common device types called classes. Examples of device classes include, but are not limited to, storage, audio, communications and HID (Human Interface). Class drivers for a given application are referenced in both the Device descriptor and Interface descriptor. Most applications can find a Class Driver which supports the majority of their function/command needs. Vendors who have a requirement for specific commands which are not supported by any of the standard class drivers may provide a vendor specific ".inf" file or driver for extra support.

10.1.8 SUMMARY

While a complete USB overview is beyond the scope of this document, a few key concepts must be noted. Low speed communication is designed for devices, which in the past, used an interrupt to communicate with the host. In the USB scheme, devices do not directly interrupt the processor when they have data. Instead the host periodically polls each device to see if they have any data. This polling rate is negotiated between the device and host, giving the system a guaranteed latency.

For more details on USB, see the USB V1.1 spec, available from the USB website at www.usb.org.

10.2 Application Isolation

Microchip provides a comprehensive support library of standard chapter 9 USB commands. These libraries provide a software layer to insulate the application software from having to handle the complexities of the USB protocol. A simple Put/Get interface is implemented to allow most of the USB processing to take place in the background within the USB interrupt service routine. Applications are encouraged to use the provided libraries during both enumeration and configured operation.

10.3 Introduction

The PIC16C745/765 USB peripheral module supports Low Speed control and interrupt (IN and OUT) transfers only. The implementation supports 3 endpoint numbers (0, 1, 2) for a total of 6 endpoints.

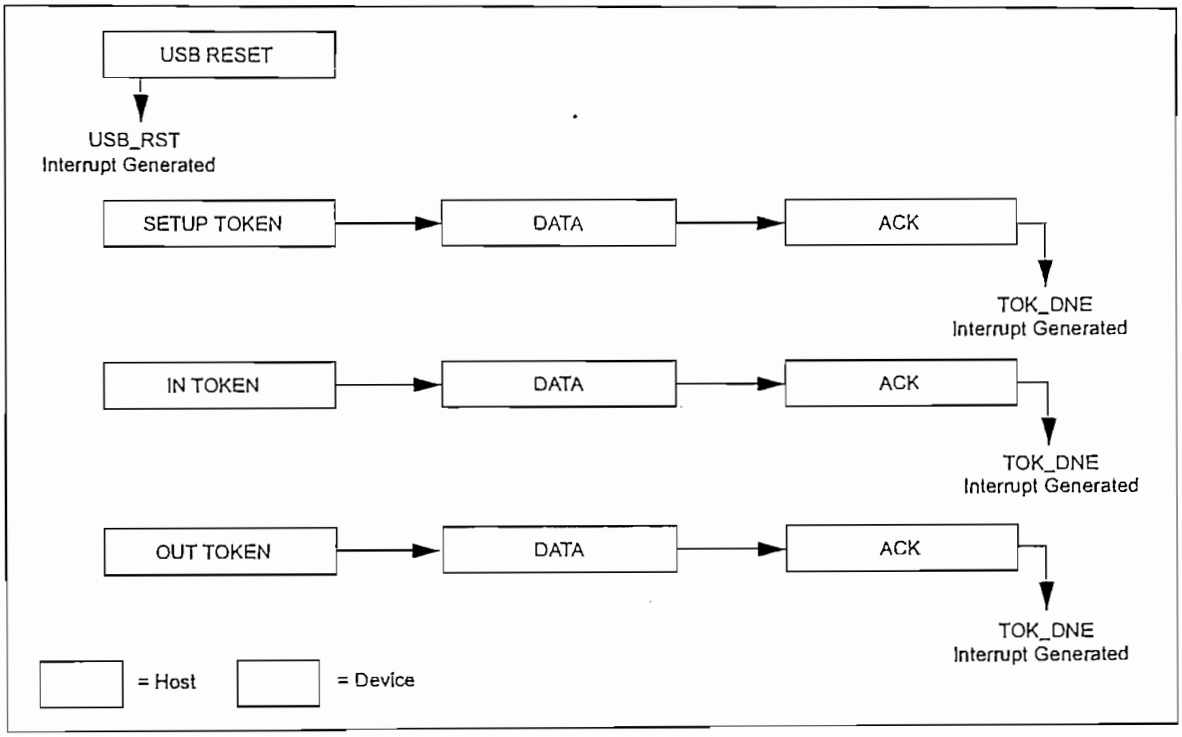
The following terms are used in the description of the USB module:

- MCU - The core processor and corresponding firmware
- SIE - Serial Interface Engine: That part of the USB that performs functions such as CRC generation and clocking of the D+ and D- signals.
- USB - The USB module including SIE and registers
- Bit Stuffing - forces insertion of a transition on D+ and D- to maintain clock synchronization
- BD - Buffer Descriptor
- BDT - Buffer Descriptor Table
- EP - Endpoint (combination of endpoint number and direction)
- IN - Packet transfer into the host
- OUT - Packet transfer out of the host

10.4 USB Transaction

When the USB transmits or receives data the SIE will first check that the corresponding endpoint and direction Buffer Description UOWN bit equals 1. The USB will move the data to or from the corresponding buffer. When the TOKEN is complete, the USB will update the BD status and change the UOWN bit to 0. The USTAT register is updated and the TOK_DNE interrupt is set. When the MCU processes the TOK_DNE interrupt it reads the USTAT register, which gives the MCU the information it needs to process the endpoint. At this point the MCU will process the data and set the corresponding UOWN bit. Figure 10-1 shows a time line of how a typical USB token would be processed.

FIGURE 10-1: USB TOKENS



10.5 USB Register Map

The USB Control Registers, Buffer Descriptors and Buffers are located in Bank 3.

10.5.1 CONTROL AND STATUS REGISTERS

The USB module is controlled by 7 registers, plus those that control each endpoint and endpoint/direction buffer.

10.5.1.1 USB Interrupt Register (UIR)

The USB Interrupt Status Register (UIR) contains flag bits for each of the interrupt sources within the USB. Each of these bits are qualified with their respective interrupt enable bits (see the Interrupt Enable Register UIE). All bits of the register are logically OR'ed together to form a single interrupt source for the microprocessor interrupt found in PIR1 (USBIF). Once an interrupt bit has been set, it must be cleared by writing a 0.

REGISTER 10-1: USB INTERRUPT FLAGS REGISTER (UIR: 190h)

U-0	U-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
—	—	STALL	UIDLE	TOK_DNE	ACTIVITY	UERR	USB_RST
bit7							bit0

R = Readable bit
C = Clearable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7-6: **Unimplemented:** Read as '0'.

bit 5: **STALL:** A STALL handshake was sent by the SIE.

bit 4: **UIDLE:** This bit is set if the USB has detected a constant idle on the USB bus signals for 3 ms. The idle timer is reset by activity on the USB bus. Once a IDLE condition has been detected, the user may wish to place the USB module in SUSPEND by setting the SUSPEND bit in the UCTRL register.

bit 3: **TOK_DNE:** This bit is set when the current token being processed is complete. The microprocessor should immediately read the USTAT register to determine the Endpoint number and direction used for this token. Clearing this bit causes the USTAT register to be cleared or the USTAT holding register to be loaded into the STAT register if another token has been processed.

bit 2: **ACTIVITY:** Activity on the D+/D- lines will cause the SIE to set this bit. Typically this bit is unmasked following detection of SLEEP. Users must enable the activity interrupt in the USB Interrupt Register (UIE: 191h) prior to entering suspend.

bit 1: **UERR:** This bit is set when any of the error conditions within the ERR_STAT register has occurred. The MCU must then read the ERR_STAT register to determine the source of the error.

bit 0: **USB_RST:** This bit is set when the USB has decoded a valid USB reset. This will inform the MCU to write 00h into the address register and enable endpoint 0. USB_RST is set once a USB reset has been detected for 2.5 microseconds. It will not be asserted again until the USB reset condition has been removed, and then reasserted.

Note 1: Bits can only be modified when UCTRL.SUSPND = 0.

10.5.1.2 USB Interrupt Enable Register (UIE)

The USB Interrupt Enable Register (UIE) contains enable bits for each of the interrupt sources within the USB. Setting any of these bits will enable the respective interrupt source in the UIR register. The values in the UIE register only affect the propagation of an interrupt condition to the PIE1 register. Interrupt conditions can still be polled and serviced.

REGISTER 10-2: USB INTERRUPT ENABLE REGISTER (UIE: 191h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	STALL	UIDLE	TOK_DNE	ACTIVITY	UERR	USB_RST
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7-6: **Unimplemented:** Read as '0'.

bit 5: **STALL:** Set to enable STALL interrupts.
1 = STALL interrupt enabled
0 = STALL interrupt disabled

bit 4: **UIDLE:** Set to enable IDLE interrupts.
1 = IDLE interrupt enabled
0 = IDLE interrupt disabled

bit 3: **TOK_DNE:** Set to enable TOK_DNE interrupts.
1 = TOK_DNE interrupt enabled
0 = TOK_DNE interrupt disabled

bit 2[(1)]: **ACTIVITY:** Set to enable ACTIVITY interrupts.
1 = ACTIVITY interrupt enabled
0 = ACTIVITY interrupt disabled

bit 1: **UERR:** Set to enable ERROR interrupts.
1 = ERROR interrupt enabled
0 = ERROR interrupt disabled

bit 0: **USB_RST:** Set to enable USB_RST interrupts.
1 = USB_RST interrupt enabled
0 = USB_RST interrupt disabled

Note 1: This interrupt is the only interrupt active during UCTRL suspend = 1.

10.5.1.3 USB Error Interrupt Status Register (UEIR)

The USB Error Interrupt Status Register (UEIR) contains bits for each of the error sources within the USB. Each of these bits are enabled by their respective error enable bits (UEIE). The result is OR'ed together and sent to the ERROR bit of the UIR register. Once an interrupt bit has been set it must be cleared by writing a zero to the respective interrupt bit. Each bit is set as soon as the error condition is detected. Thus, the interrupt will typically not correspond with the end of a token being processed.

REGISTER 10-3: USB ERROR INTERRUPT FLAGS STATUS REGISTER (UEIR: 192h)

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
BTS_ERR	OWN_ERR	WRT_ERR	BTO_ERR	DFN8	CRC16	CRC5	PID_ERR
bit7							bit0

R = Readable bit
C = Clearable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7: **BTS_ERR:** A bit stuff error has been detected.

bit 6: **OWN_ERR:** This bit is set if the USB is processing a token and the OWN bit within the BDT is equal to 0 (signifying that the microprocessor owns the BDT and the SIE does not have access to the BDT). If processing an IN TOKEN this would cause a transmit data underflow condition. Processing an OUT or SETUP TOKEN would cause a receive data overflow condition.

bit 5: **WRT_ERR:** Write Error. A write by the MCU to the USB Buffer Descriptor Table or Buffer area was unsuccessful. This error occurs when the MCU attempts to write to the same location that is currently being written to by the SIE.

bit 4: **BTO_ERR:** This bit is set if a bus turnaround time-out error has occurred. This USB uses a bus turnaround timer to keep track of the amount of time elapsed between the token and data phases of a SETUP or OUT TOKEN or the data and handshake phases of a IN TOKEN. If more than 17-bit times are counted from the previous EOP before a transition from IDLE, a bus turnaround time-out error will occur.

bit 3: **DFN8:** The data field received was not 8 bits. The USB Specification 1.1 specifies that data field must be an integral number of bytes. If the data field was not an integral number of bytes this bit will be set.

bit 2: **CRC16:** The CRC16 failed.

bit 1: **CRC5:** This interrupt will detect CRC5 error in the token packets generated by the host. If set the token packet was rejected due to a CRC5 error.

bit 0: **PID_ERR:** The PID check field failed.

Note 1: Bits can only be modified when UCTRL.SUSPND = 0.

10.5.1.4 Error Interrupt Enable Register (UEIE)

The USB Error Interrupt Enable Register (UEIE) contains enable bits for each of the error interrupt sources within the USB. Setting any of these bits will enable the respective error interrupt source in the UEIR register.

REGISTER 10-4: USB ERROR INTERRUPT ENABLE REGISTER (UEIE: 193h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BTS_ERR	OWN_ERR	WRT_ERR	BTO_ERR	DFN8	CRC16	CRC5	PID_ERR
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7: **BTS_ERR:** Set this bit to enable BTS_ERR interrupts.
1 = BTS_ERR interrupt enabled
0 = BTS_ERR interrupt disabled

bit 6: **OWN_ERR:** Set this bit to enable OWN_ERR interrupts.
1 = OWN_ERR interrupt enabled
0 = OWN_ERR interrupt disabled

bit 5: **WRT_ERR:** Set this bit to enable WRT_ERR interrupts.
1 = WRT_ERR interrupt enabled
0 = WRT_ERR interrupt disabled

bit 4: **BTO_ERR:** Set this bit to enable BTO_ERR interrupts.
1 = BTO_ERR interrupt enabled
0 = BTO_ERR interrupt disabled

bit 3: **DFN8:** Set this bit to enable DFN8 interrupts.
1 = DFN8 interrupt enabled
0 = DFN8 interrupt disabled

bit 2: **CRC16:** Set this bit to enable CRC16 interrupts.
1 = CRC16 interrupt enabled
0 = CRC16 interrupt disabled

bit 1: **CRC5:** Set this bit to enable CRC5 interrupts.
1 = CRC5 interrupt enabled
0 = CRC5 interrupt disabled

bit 0: **PID_ERR:** Set this bit to enable PID_ERR interrupts.
1 = PID_ERR interrupt enabled
0 = PID_ERR interrupt disabled

10.5.1.5 Status Register (USTAT)

The USB Status Register reports the transaction status within the USB. When the MCU recognizes a TOK_DNE interrupt, this register should be read to determine the status of the previous endpoint communication. The data in the status register is valid when the TOK_DNE interrupt bit is asserted.

The USTAT register is actually a read window into a status FIFO maintained by the USB. When the USB uses a BD, it updates the status register. If another USB transaction is performed before the TOK_DNE interrupt is serviced the USB will store the status of the next transaction in the STAT FIFO. Thus, the STAT register is actually a four byte FIFO which allows the MCU to process one transaction while the SIE is processing the next. Clearing the TOK_DNE bit in the INT_STAT register causes the SIE to update the STAT register with the contents of the next STAT value. If the data in the STAT holding register is valid, the SIE will immediately reassert the TOK_DNE interrupt.

REGISTER 10-5: USB STATUS REGISTER (USTAT: 194h)

U-0	U-0	U-0	R-X	R-X	R-X	U-0	U-0
—	—	—	ENDP1	ENDP0	IN	—	—
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7-5: **Unimplemented:** Read as '0'.

bit 4-3: **ENDP<1:0>:** These bits encode the endpoint address that received or transmitted the previous token. This allows the microprocessor to determine which BDT entry was updated by the last USB transaction.

bit 2: **IN:** This bit indicates the direction of the last BD that was updated.
1 = The last transaction was an IN TOKEN
0 = The last transaction was an OUT or SETUP TOKEN

bit 1-0: **Unimplemented:** Read as '0'.

10.5.1.6 USB Control Register (UCTRL)

The control register provides various control and configuration information for the USB.

REGISTER 10-6: USB CONTROL REGISTER (UCTRL: 195h)

U-0	U-0	R-X	R/C-0	R/W-0	R/W-0	R/W-0	U-0
—	—	SE0	PKT_DIS	DEV_ATT	RESUME	SUSPND	—
bit7							bit0

R = Readable bit
W = Writable bit
C = Clearable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7-6: **Unimplemented:** Read as '0'.

bit 5: **SE0:** Live Single Ended Zero. This status bit indicates that the D+ and D- lines are both pulled to low.
1 = single ended zero being received
0 = single ended zero not being received

bit 4 **PKT_DIS:** The PKT_DIS bit informs the MCU that the SIE has disabled packet transmission and reception. Clearing this bit allows the SIE to continue token processing. This bit is set by the SIE when a Setup Token is received allowing software to dequeue any pending packet transactions in the BDT before resuming token processing. The PKT_DIS bit is set under certain conditions such as back to back SETUP tokens. This bit is not set on every SETUP token and can be modified only when UCTRL.SUSPND = 0.

bit 3: **DEV_ATT:** Device Attach. Enables the 3.3V output.
1 = When DEV_ATT is set, the VUSB pin will be driven with 3.3V (nominal).
0 = The VUSB pins (D+ and D-) will be in a high impedance state.

bit 2: **RESUME:** Setting this bit will allow the USB to execute resume signaling. This will allow the USB to perform remote wake-up. Software must set RESUME to 1 for 10 mS then clear it to 0 to enable remote wake-up. For more information on RESUME signaling, see Section 7.1.7.5, 11.9 and 11.4.4 in the USB 1.1 specification.
1 = perform Resume signaling
0 = normal operation

bit 1: **SUSPND:** Suspends USB operation and clocks and places the module in low power mode. This bit will generally be set in response to a UIDLE interrupt. It will generally be reset after an ACTIVITY interrupt. VUSB regulation will be different between suspend and non-suspend modes. The VUSB pin will still be driven, however the transceiver outputs are disabled.
1 = USB module in power conserve mode
0 = USB module normal operation

bit 0: **Unimplemented:** Read as '0'.

10.5.1.7 USB Address Register (UADDR)

The Address Register (UADDR) contains the unique USB address that the USB will decode. The register is reset to 00h after the reset input has gone active or the USB has decoded a USB reset signaling. That will initialize the address register to decode address 00h as required by the USB specification. The USB address must be written by the MCU during the USB SETUP phase.

REGISTER 10-7: USB ADDRESS REGISTER (UADDR: 196h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADDR6	ADDR5	ADDR4	ADDR3	ADDR2	ADDR1	ADDR0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7: **Unimplemented:** Read as '0'.

bit 6-0: **ADDR<6:0>:** This 7-bit value defines the USB address that the USB will decode.

10.5.1.8 USB Software Status Register

This register is used by the USB firmware libraries for USB status.

REGISTER 10-8: RESERVED SOFTWARE LIBRARY REGISTER (USWSTAT: 197H):.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
Function IDs						Configuration Status	

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

10.5.1.9 Endpoint Registers

Each endpoint is controlled by an Endpoint Control Register. The PIC16C745/765 supports Buffer Descriptors (BD) for the following endpoints:

- EP0 Out
- EP0 In
- EP1 Out
- EP1 In
- EP2 Out
- EP2 In

The user will be required to disable unused Endpoints and directions using the Endpoint Control Registers.

10.5.1.10 USB Endpoint Control Register (EPCn)

The Endpoint Control Registers contains the endpoint control bits for each of the 6 endpoints available on USB for a decoded address. These four bits define the control necessary for any one endpoint. Endpoint 0 (ENDP0) is associated with control pipe 0 which is required by USB for all functions (IN, OUT, and SETUP). Therefore, after a USB_RST interrupt has been received the microprocessor should set ENDPT0 to contain 06h.

REGISTER 10-9: USB ENDPOINT CONTROL REGISTER (UEPn: 198H-19Ah)

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	EP_CTL_DIS	EP_OUT_EN	EP_IN_EN	EP_STALL
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7-4: **Unimplemented:** Read as '0'.

bit 3-1: **EP_CTL_DIS, EP_OUT_EN, EP_IN_EN:** These three bits define if an endpoint is enabled and the direction of the endpoint. The endpoint enable/direction control is defined as follows:

EP_CTL_DIS	EP_OUT_EN	EP_IN_EN	Endpoint Enable/Direction Control
X	0	0	Disable Endpoint
X	0	1	Enable Endpoint for IN tokens only
X	1	0	Enable Endpoint for OUT tokens only
1	1	1	Enable Endpoint for IN and OUT tokens
0	1	1	Enable Endpoint for IN, OUT, and SETUP tokens

bit 0: **EP_STALL:** When this bit is set it indicates that the endpoint is stalled. This bit has priority over all other control bits in the Endpoint Enable register, but is only valid if EP_IN_EN=1 or EP_OUT_EN=1. Any access to this endpoint will cause the USB to return a STALL handshake. The EP_STALL bit can be set or cleared by the SIE. Refer to the USB 1.1 Specification, Sections 4.4.4 and 8.5.2 for more details on the STALL protocol.

10.6 Buffer Descriptor Table (BDT)

To efficiently manage USB endpoint communications the USB implements a Buffer Descriptor Table (BDT) in register space. Every endpoint requires a 4 byte Buffer Descriptor (BD) entry. Because the buffers are shared between the MCU and the USB, a simple semaphore mechanism is used to distinguish which is allowed to update the BD and buffers in system memory. The UOWN bit is cleared when the BD entry is "owned" by the MCU. When the UOWN bit is set to 1, the BD entry and the buffer in system memory is owned by the USB. The MCU should not modify the BD or its corresponding data buffer.

The Buffer Descriptors provide endpoint buffer control information for the USB and MCU. The Buffer Descriptors have different meaning based on the value of the UOWN bit.

The USB Controller uses the data stored in the BDs when UOWN = 1 to determine:

- Data0 or Data1 PID
- Data toggle synchronization enable
- Number of bytes to be transmitted or received
- Starting location of the buffer

The MCU uses the data stored in the BDs when UOWN = 0 to determine:

- Data0 or Data1 PID
- The received TOKEN PID
- Number of bytes transmitted or received

Each endpoint has a 4 byte Buffer Descriptor and points to a data buffer in the USB dual port register space. Control of the BD and buffer would typically be handled in the following fashion:

- The MCU verifies UOWN = 0, sets the BDndAL to point to the start of a buffer, if necessary fills the buffer, then sets the BDndST byte to the desired value with UOWN = 1.
- When the host commands an in or out transaction, the Serial Interface Engine (SIE) performs the following:
  - Get the buffer address
  - Read or write the buffer
  - Update the USTAT register
  - Update the buffer descriptors with the packet ID (PID) value
  - Set the data 0/1 bit
  - Update the byte count
  - Clear the UOWN bit
- The MCU is interrupted and reads the USTAT, translates that value to a BD, where the UOWN, PID, Data 0/1, and byte count values are checked.

REGISTER 10-10: BUFFER DESCRIPTOR STATUS REGISTER. BITS WRITTEN BY THE MCU (BDndST: 1A0h, 1A4h, 1A8h, 1ACh, 1B0h, 1B4h)

W-X	W-X	U-X	U-X	W-X	W-X	U-X	U-X
UOWN	DATA0/1	—	—	DTS	BSTALL	—	—
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7: **UOWN:** USB Own. This UOWN bit determines who currently owns the buffer. The SIE writes a 0 to this bit when it has completed a token. This byte of the BD should always be the last byte the MCU updates when it initializes a BD. Once the BD has been assigned to the USB, the MCU should not change it in any way.
1 = USB has exclusive access to the BD. The MCU should not modify the BD or buffer.
0 = The MCU has exclusive access to the BD. The USB ignores all other fields in the BD.

bit 6: **DATA0/1:** This bit defines the type of data toggle packet that was transmitted or received.
1 = Data 1 packet
0 = Data 0 packet

bit 5-4: **Reserved:** Read as 'X'.

bit 3: **DTS:** Setting this bit will enable the USB to perform Data Toggle Synchronization. If a packet arrives with an incorrect DTS, it will be ignored and the buffer will remain unchanged.
1 = Data Toggle Synchronization is performed
0 = No Data Toggle Synchronization is performed

bit 2: **BSTALL:** Buffer Stall. Setting this bit will cause the USB to issue a STALL handshake if a token is received by the SIE that would use the BD in this location. The BD is not consumed by the SIE (the own bit remains and the rest of the BD are unchanged) when a BSTALL bit is set.

bit 1-0: **Reserved:** Read as 'X'.

Note: Recommend that users not use BSF, BCF due to the dual functionality of this register.

REGISTER 10-11: BUFFER DESCRIPTOR STATUS. BITS READ BY THE MCU. (BDndST: 1A0h, 1A4h, 1A8h, 1ACh, 1B0h, 1B4h)

R/W-0	R/W-X	R/W-X	R/W-X	R/W-X	R/W-X	U-X	U-X
UOWN	DATA0/1	PID3	PID2	PID1	PID0	—	—
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7: **UOWN:** USB Own. This UOWN bit determines who currently owns the buffer. The SIE writes a 0 to this bit when it has completed a token. This byte of the BD should always be the last byte the MCU updates when it initializes a BD. Once the BD has been assigned to the USB, the MCU should not change it in any way.
1 = USB has exclusive access to the BD. The MCU should not modify the BD or buffer.
0 = The MCU has exclusive access to the BD. The USB ignores all other fields in the BD.

bit 6: **DATA0/1:** This bit defines the type of data toggle packet that was transmitted or received.
1 = Data 1 packet
0 = Data 0 packet

bit 5-2: **PID<3:0>:** Packet Identifier. The received token PID value

bit 1-0: **Reserved:** Read as 'X'.

Note: Recommend that users not use BSF, BCF due to the dual functionality of this register.

REGISTER 10-12: BUFFER DESCRIPTOR BYTE COUNT (BDndBC: 1A1h, 1A5h, 1A9h, 1ADh, 1B1h, 1B5h))

U-X	U-X	U-X	U-X	R/W-X	R/W-X	R/W-X	R/W-X
—	—	—	—	BC3	BC2	BC1	BC0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7-4: **Reserved:** Read as 'X'.

bit 3-0: **BC<3:0>:** The Byte Count bits represent the number of bytes that will be transmitted for an IN TOKEN or received during an OUT TOKEN. Valid byte counts are 0 - 8. The SIE will change this field upon the completion of an OUT or SETUP token with the actual byte count of the data received.

REGISTER 10-13: BUFFER DESCRIPTOR ADDRESS LOW (BDndAL: 1A2h, 1A6h, 1AAh, 1AEh, 1B2h, 1B6h)

R/W-X	R/W-X	R/W-X	R/W-X	R/W-X	R/W-X	R/W-X	R/W-X
BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset
X = Don't care

bit 7-0: **BA<7:0>:** Buffer Address. The base address of the buffer controlled by this endpoint. The upper order bit address (BA8) of the 9-bit address is assumed to be 1h. This value must point to a location within the dual port memory space (1B8h - 1DFh). The upper order bits of the address are assumed to point to Bank 3.

Note 1: This register should always contain a value between B8h-DFh.

10.6.1 ENDPOINT BUFFERS

Endpoint buffers are located in the Dual Port RAM area. The starting location of an endpoint buffer is determined by the Buffer Descriptor.

10.7 TRANSCEIVER

An on-chip integrated transceiver is included to drive the D+/D- physical layer of the USB.

10.7.1 REGULATOR

A 3.3V regulator provides the D+/D- drives with power, as well as an external pin. This pin is intended to be used to power a 1.5KΩ ±5% pull-up resistor on the D- line to signal a low speed device, as specified by the USB standard. A ±20% 10nF capacitor is required on VUSB for regulator stability.

10.7.1.1 VUSB Output

The VUSB provides a 3.3V nominal output. This drive current is sufficient for a pull-up only.

10.8 USB Software Libraries

Microchip Technology provides a comprehensive set of Chapter 9 Standard requests functions to aid developers in implementing their designs. See Microchip Technology's website for the latest version of the software libraries.

TABLE 10-1: USB PORT FUNCTIONS

Name	Function	Input Type	Output Type	Description
VUSB	VUSB	—	Power	3.3V for pull up resistor
D-	D-	USB	USB	USB Differential Bus
D+	D+	USB	USB	USB Differential Bus

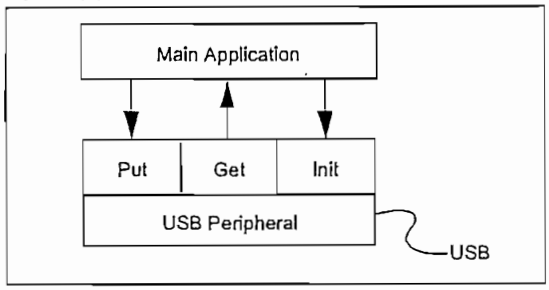
Legend: OD = open drain, ST = Schmitt Trigger

10.9 USB Firmware Users Guide

10.9.1 INTRODUCING THE USB SOFTWARE INTERFACE

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

FIGURE 10-2: USB SOFTWARE INTERFACE



10.9.2 INTEGRATING USB INTO YOUR APPLICATION

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

Communicating on USB is similar to communicating via a hardware USART. The main difference is that a USART typically works on a single byte at a time, where USB operates on a buffer of up to 8 bytes at a time.

There is one function defined to start the enumeration process and two additional functions are defined for moving buffers between the main application and the USB peripheral. InitUSB initializes the USB peripheral allowing the host to enumerate the device. Then for normal data communications, function PutUSB sends data to the host and function GetUSB receives data from the host.

There's a lot that happens behind the scenes to make the communication work, but these calls are all an application needs to communicate on the bus. The rest is handled on an interrupt basis.

InitUSB initializes the Buffer Descriptor table, and enables the USB interrupt so enumeration can begin. The actual enumeration process occurs automatically, driven by the host and interrupt service routine. The macro ConfiguredUSB waits until the device is in the CONFIGURED mode and ready to go. The time required to enumerate is completely dependent on the host and bus loading.

10.9.3 INTERRUPT STRUCTURE CONCERNS

10.9.3.1 Processor Resources

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

These are the considerations you'll need to take into account if you write your own Interrupt Service Routine: Save W, Status, FSR and PCLATH which are the file registers that may be corrupted by servicing the USB interrupt.

We provide a skeleton ISR which will do this for you, and includes tests for each of the possible ISR bits. This provides a good place to start from if you haven't already written your own. See file USB_INT.ASM.

10.9.3.2 Stack Levels

The hardware stack on the device is only 8 levels deep. So the worst case call between the application and ISR can only be 8 levels. The enumeration process requires 6 levels, so it's best if the main application holds off on any processing until enumeration is complete. ConfiguredUSB is a macro that waits until the enumeration process is complete for exactly this purpose.

10.9.3.3 ROM

The code required to support the USB interrupt, including the chapter 9 interface calls, but not including the descriptor tables is about 1kW. The descriptor and string descriptor tables can each take up to an additional 256W. The location of these parts is not restricted, and the linker script may be edited to control the placement of each part. See the Strings and Descriptors sections in the linker script

10.9.3.4 RAM

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

10.9.3.5 Common RAM usage

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

These are particularly useful when responding to interrupts. When an interrupt occurs, the ISR doesn't immediately know which bank is addressed. With devices that don't support common RAM, the W regis-

ter must be provided for in each bank. The 16C745/765 can save the appropriate registers in Common RAM and not have to waste a byte in each bank for W register.

10.9.3.6 Buffer allocation

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

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

The default configuration allocates individual buffers to EP0 OUT, EP0 In, EP1 Out, and EP1 In. The last buffer is shared between EP2 In and EP2 Out. Again, the spec says low speed devices can only use 2 endpoints beyond EP0. This configuration supports most of the possible combinations of endpoints (EP1 OUT and EP1 IN, EP1OUT and EP2IN, EP1 OUT and EP2 OUT, EP1 IN and EP2 OUT, EP1 IN and EP2 IN). The only combination that is not supported by this configuration is Endpoint 2 IN and Endpoint 2 OUT. If your application needs both EP2 IN and EP2 OUT, the function USBReset will need to be edited to give each of these dedicated buffers at the expense of EP1.

10.9.3.7 Page Selects/Bank Selects

Microchip's firmware libraries are implemented with careful use of the page and bank bits. The bank selects ensure the correct file registers are referenced during the calls. Similarly PCLATH is modified by the use of PAGESEL commands. These are needed anytime a CALL or GOTO reference is made outside of the current code segment. ServiceUSBInt modify W, STATUS, FSR and PCLATH, therefore these will need to be saved upon entry to the ISR and restored upon leaving the ISR.

10.9.4 FUNCTION CALL REFERENCE

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

InitUSB should be called by the main program immediately upon power-up. It sets up the Buffer Descriptor Table, transitions the part to the Powered state, and prepares the device for enumeration. At this point the USB Reset is the only USB interrupt allowed, preventing the part from responding to anything on the bus until it's been reset. The USB Reset interrupt transitions the part to the default state where it responds to commands on address zero. When it receives a SET ADDRESS command, the device transitions to the addressed state and now responds to commands on the new address.

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

GetUSB (Buffer Pointer, Endpoint) returns data sent from the host. If there is a buffer ready (i.e., data has been received from the host) it is copied to the destination pointed to by FSR/IRP (A buffer pointer in FSF/IRP and the endpoint number in W must be provided.). If no data is available, it returns a failure code. Thus, the functions of polling for buffer ready and copying the data are combined into the one function.

ServiceUSBInt handles all interrupts generated by the USB peripheral. First it copies the active buffer to common RAM which provides a quick turn around on the buffer in dual port RAM and also to avoids having to switch banks during processing of the buffer.

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

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

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

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

ConfiguredUSB (Macro) Continuously polls the enumeration status bits and waits until the device has been configured by the host.

10.9.5 BEHIND THE SCENES

The ISR calls ServiceUSBInt, which then further has to mask the USB Interrupt register with the USB Interrupt Enable bits, then see what caused the interrupt. InitUSB only enables the Reset interrupt (USB_RST). This prevents the device from responding to anything on the bus until it's been reset by the host. When the reset is received, the Buffer Descriptors are initialized, most of the rest of the interrupts are unmasked and the device transitions from the POWERED to DEFAULT state. Now it can respond to commands on address zero. From there the rest of the enumeration process takes place, including assigning an address to the device through the SET_ADDRESS command and selecting a configuration through the SET_CONFIGURATION command. Once the device is configured, the application can communicate with the host using the GetUSB and PutUSB calls.

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

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

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

10.9.6 EXAMPLES

This example shows how the USB functions are used. This example first initializes the USB peripheral which allows the host to enumerate the device. The enumeration process occurs in the background, via an interrupt service routine. This function waits until enumeration is complete, and then polls EP1 OUT to see if there is any data available. When a buffer is available, it is copied to the IN buffer. Presumably your application would do something more interesting with the data than this example.

```
; *****************************************************************
; Demo program that initializes the USB peripheral, allows the Host
;       to Enumerate, then copies buffers from EP1OUT to EP1IN.
; *****************************************************************
main
        call    InitUSB             ; Set up everything so we can enumerate
        ConfiguredUSB               ; wait here until we have enumerated.

idleloop
        call    CheckSleep          ; Ok, here's a good point to put part to sleep if no activity on the bus.

CheckEP1                            ; Check Endpoint 1 for an OUT transaction
        bcf     STATUS,IRP          ; point to lower banks
        movlw   buffer
        movwf   FSR                 ; point FSR to our buffer
        movlw   1                   ; check end point 1
        call    GetUSB              ; If data is ready, it will be copied.
        btfss   STATUS,C            ; was there any data for us?
        goto    idleloop            ; Nope, check again.

PutBuffer
        bcf     STATUS,IRP          ; point to lower banks
        movwf   bufferlen           ; save buffer length
        movlw   buffer
        movwf   FSR                 ; point FSR to our buffer
        swapf   bufferlen,w         ; upper nybble of W is buffer length
        iorlw   1                   ; lower nybble of W is EndPoint number
        call    PutUSB
        btfss   STATUS,C            ; was it successful?
        goto    PutBuffer           ; No: try again until successful
        goto    idleloop            ; Yes: restart loop

        end
```

10.9.7 ASSEMBLING THE CODE

The code is designed to be used with the linker. There is no provision for include-able files. The code comes packaged as several different files:

- USB_CH9.ASM - handles all the Chapter 9 command processing.
- USB_INTF.ASM - Provides the interface functions PutUSB, GetUSB
- USBMACRO.INC - Macros used by
- USB_DEFS.INC - #Defines used throughout the code.
- USB_INT.ASM - Sample interrupt service routine.
- 16C765.LKR - Linker script (provided with MPLAB)

10.9.7.1 Assembly Options:

There are two #defines at the top of the code that control assembly options.

10.9.7.2 #define ERRORCOUNTERS

This define includes code to count the number of errors that occur, by type of error. This requires extra code and RAM locations to implement the counters.

10.9.7.3 #define FUNCTIONIDS

This is useful for debug. It encodes the upper 6 bits of USWSTAT (0x197) to indicate which function is executing.

ANEXO D

DATOS TÉCNICOS DE LOS ELEMENTOS

SensComp, Inc.
36704 Commerce Rd.
Livonia, MI 48150
Telephone: (734) 953-4783
Fax: (734) 953-4518
www.senscomp.com

600 Series Instrument Transducer

SensComp's Series 600 Instrument Grade electrostatic transducer is specifically intended for operation in air at ultrasonic frequencies. The assembly comes complete with a perforated protective cover.

Features

50 kHz Electrostatic Transducer
Beam Angle of 15° at -6 dB
Ranges from 6" to 35'
Excellent Receive Sensitivity
Perforated Protective Cover.
Specifically Intended for Operation in Air at Ultrasonic Frequencies

Part No.

PID# 604142 – Series 600 Instrument Grade Transducer

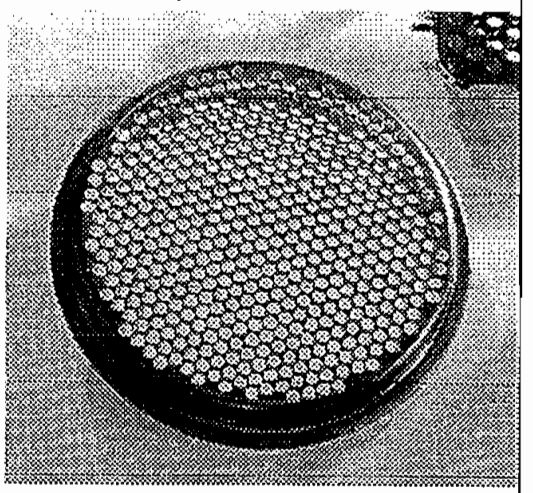
Benefits

Able to Range from 6" to 35'
Excellent Receive Sensitivity

Applications

Level Measurement, Proximity Detection, Presence Detection, Robotics, Educational Products

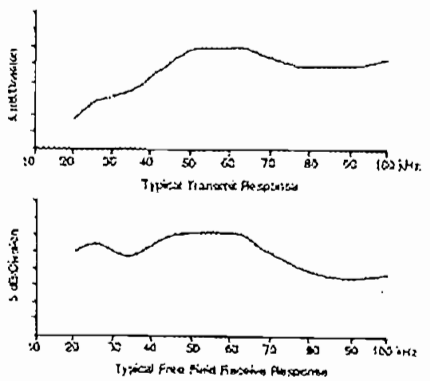
Specifications

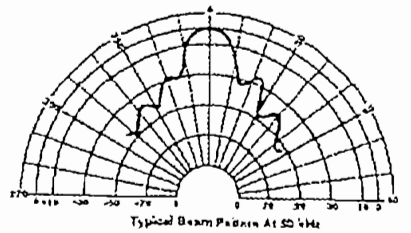


Description

The Series 600 ultra sensitive transducers feature ranging capability from 2.5 cm to 15.2 m when used with SensComp drive electronics. They are ideally suited for demanding applications where the most sensitivity possible is the highest priority. These ultrasonic transducers are among the best available when detecting soft targets. They have a broad band frequency response.

Transmit/Receive Response & Beam Pattern





Note: dB normalized to on-axis response.
Note: Curves are representative only. Individual responses may differ.

For more information, visit our website: www.senscomp.com

7/22/03

Series 600 Instrument Grade Specifications

Usable Frequency Range
- Transmitting ..... See Graph
- Receiving ..... See Graph

Beam Pattern ..... See Graph
- Typical: 15° at -6dB

Transmitting Sensitivity ..... 110 dB min
- at 50.0 kHz; 0dB re 20 µPa at 1 meter
- (300 VAC_{PP}; 150 VDC bias)

Receiving Sensitivity ..... -42 dB min
- at 50.0 kHz; 0dB = 1 volt/µ Pa
- (150 VDC bias)

Distance Range ..... 0.15 to 10.7 M
- (0.5 to 35 feet)

Resolution (± 1% over entire range) ..... ± 3mm to 3m
- (± 0.12 to 10 ft)

Weight ..... 8.2 gm (0.29 oz)

Suggested DC Bias Voltage 200V

Suggested AC Driving Voltage ..... 200V peak

Combined Voltage ..... 400V max

Capacitance at 1 kHz (typical) ..... 400–500 pf
- (at 150 VDC bias)

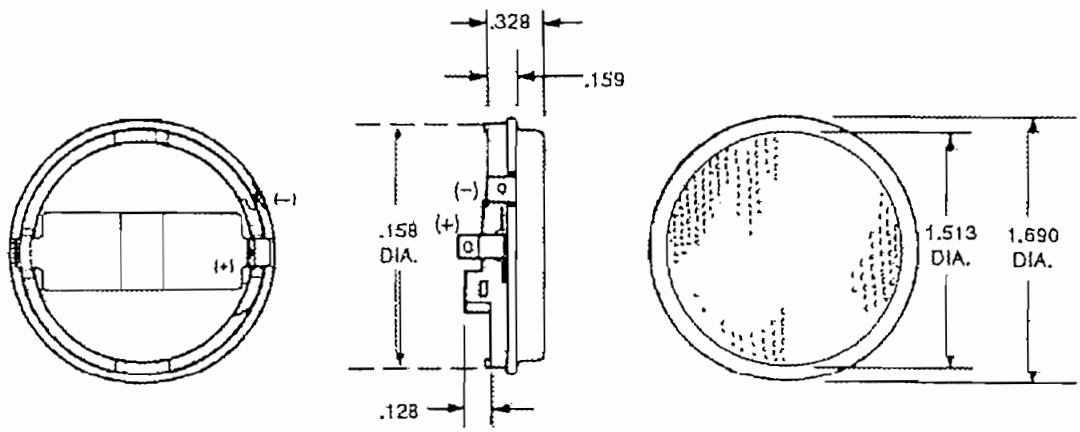
Operating Temperature ..... -30 to +70° C
- (-20 to 160° F)

Relative Humidity (non condensing) .... 5% - 95%

Dimension
- Thickness ..... 0.46 inch
- Diameter ..... 1.69 inch

Standard Finish
- Foil ..... Gold
- Housing ..... Flat Black Cold Rolled Steel

Specifications subject to change without notice



SensComp, Inc.
36704 Commerce Rd.
Livonia, MI 48150
Telephone: (734) 953-4783
Fax: (734) 953-4518
www.senscomp.com

Adjustable Ranging Transformer

SensComp Electronic Component – Ranging Transformer, Adjustable Core

Features

Adjustable Core for Precise Frequency Matching

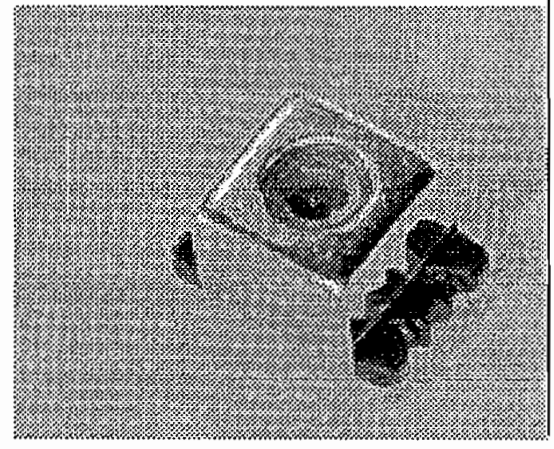
Part No.

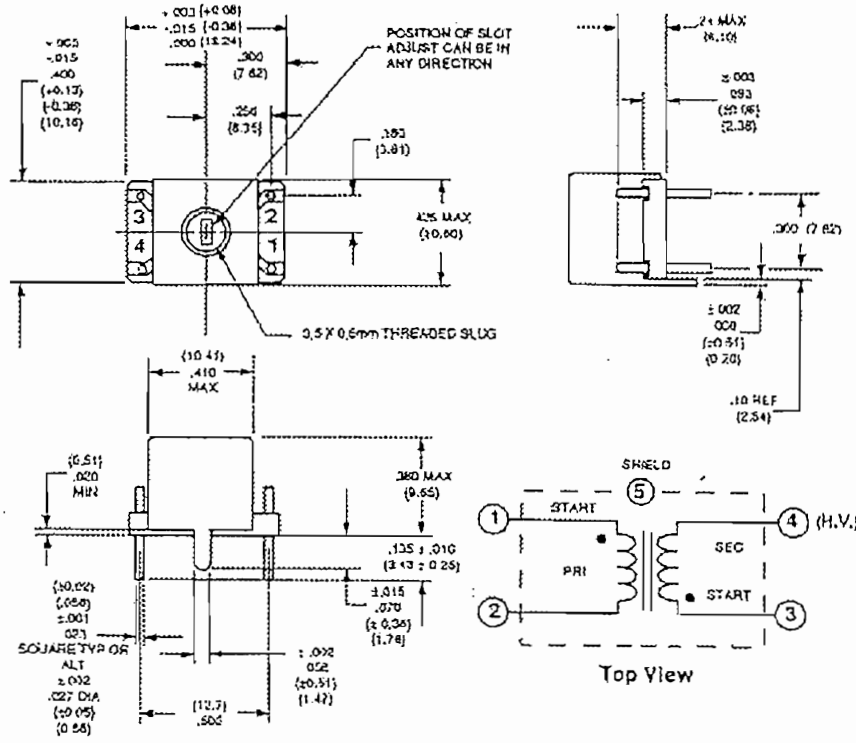
PID# 614096 – Ranging Transformer

Specifications

Primary Supply Voltage ..... 5.0 ± 0.1 VDC
Primary DC Resistance (ohm) ..... 0.47 ± 15%
Primary Winding ..... Outermost
Secondary Inductance (at 55 kHz) ..... 21 ± 2 mH
Secondary DC Resistance (ohm) ..... 150 - 200
Output Voltage Amplitude ..... 350 – 450 V_{PP}
Duty Cycle ..... < 2%
Dimensions: dimensions in parentheses are in mm

Specifications



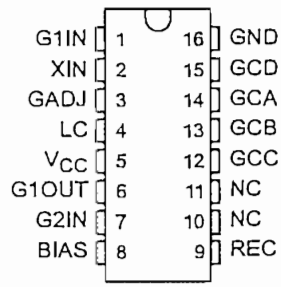


For more information, visit our website: www.senscomp.com

SLSS003 – SEPTEMBER 1983 – REVISED MARCH 1988

- Designed for Use With the TL851 in Sonar Ranging Modules Like the SN28827
- Digitally Controlled Variable-Gain Variable-Bandwidth Amplifier
- Operational Frequency Range of 20 kHz to 90 kHz
- TTL-Compatible
- Operates From Power Sources of 4.5 V to 6.8 V
- Interfaces to Electrostatic or Piezoelectric Transducers
- Overall Gain Adjustable With One External Resistor

N PACKAGE
(TOP VIEW)



NC – No internal connection

description

The TL852 is an economical sonar ranging receiver integrated circuit for use with the TL851 control integrated circuit. A minimum of external components is required for operation, and this amplifier easily interfaces to Polaroid's 50-kHz electrostatic transducer. An external 68-kΩ ±5% resistor from BIAS to GND provides the internal biasing reference. Amplifier gain can be set with a resistor from G1IN to GADJ. Required amplifier gain will vary for different applications. Using the detect-level measurement circuit of Figure 1, a nominal peak-to-peak value of 230 mV input during gain step 2 is recommended for most applications. For reliable operation, a level no lower than 50 mV should be used. The recommended detect level of 230 mV can be obtained for most amplifiers with an R1 value between 5 kΩ and 20 kΩ

Digital control of amplifier gain is provided with gain control inputs GCA, GCB, GCC, and GCD. These inputs must be driven synchronously (all inputs stable within 0.1 μs) to avoid false receive output signals due to invalid logic counts. This can be done easily with the TL851 control integrated circuit. A plot showing relative gain for the various gain steps versus time can be seen in Figure 2. To dampen ringing of the 50-kHz electrostatic transducer, a 5-kΩ resistor from G1IN to XIN is recommended.

An external parallel combination of inductance and capacitance between LC and V_{CC} provides an amplifier with an externally controlled gain and Q. This not only allows control of gain to compensate for attenuation of signal with distance, but also maximizes noise and sidelobe rejection. Care must be taken to accurately tune the L-C combination at operating frequency or gain and Q will be greatly reduced at higher gain steps.

AC coupling between stages of the amplifier is accomplished with a 0.01-mF capacitor for proper biasing.

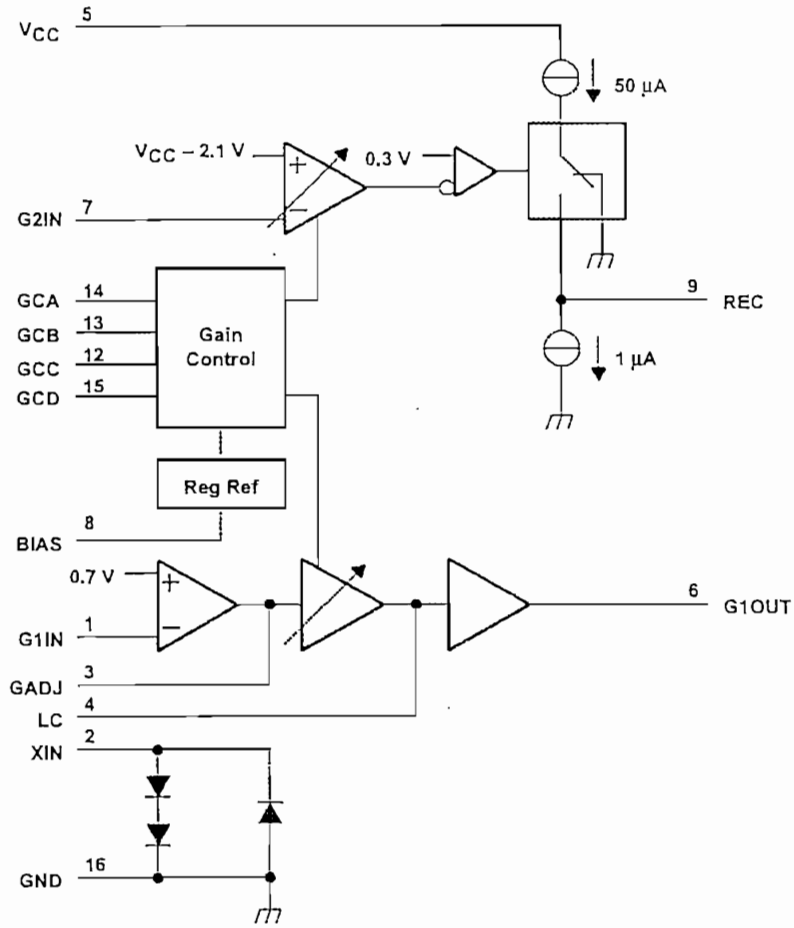
The receive output is normally held at a low level by an internal 1-μA current source. When an input of sufficient amplitude is received, the output is driven alternately by the 1-μA discharge current and a 50-μA charging current. A 1000-pF capacitor is required from REC to GND to integrate the received signal so that one or two noise pulses will not be recognized.

XIN provides clamping for the transformer secondary when used for transducer transmit drive as shown in Figure 4 of the SN28827 data sheet.

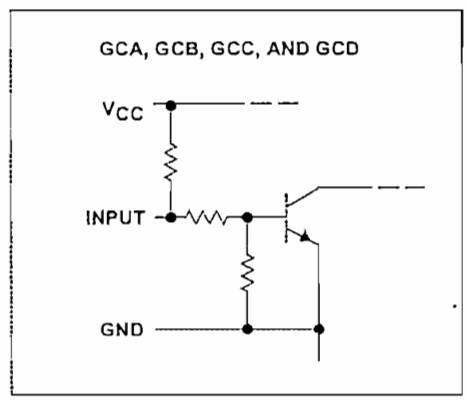
The TL852 is characterized for operation from 0°C to 40°C.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251–1443

functional block diagram



schematic of gain control inputs



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Voltage at any pin with respect to GND ..... – 0.5 V to 7 V
Voltage at any pin with respect to V_{CC} ..... – 7 V to 0.5 V
XIN input current (50% duty cycle) ..... ±60 mA
Continuous power dissipation at (or below) 25°C free-air temperature (see Note 1) ..... 1150 mW
Operating free-air temperature range ..... – 40°C to 85°C
Storage temperature range ..... – 65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C

† Stresses beyond those listed under absolute maximum ratings may cause permanent damage to the device. This is a stress rating only, and functional operation of the device at these or any other conditions beyond those indicated in the recommended operating conditions section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: For operation above 25°C, derate linearly at the rate of 9.2 mW/°C.

recommended operating conditions

		MIN	MAX	UNIT
Supply voltage, V_{CC}		4.5	6.8	V
High-level input voltage, V_{IH}	GCA, GCB, GCC, GCD	2.1		V
Low-level input voltage, V_{IL}			0.6	
Bias resistor between BIAS and GND		64	72	kΩ
Operating free-air temperature, T_A		0	40	°C

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

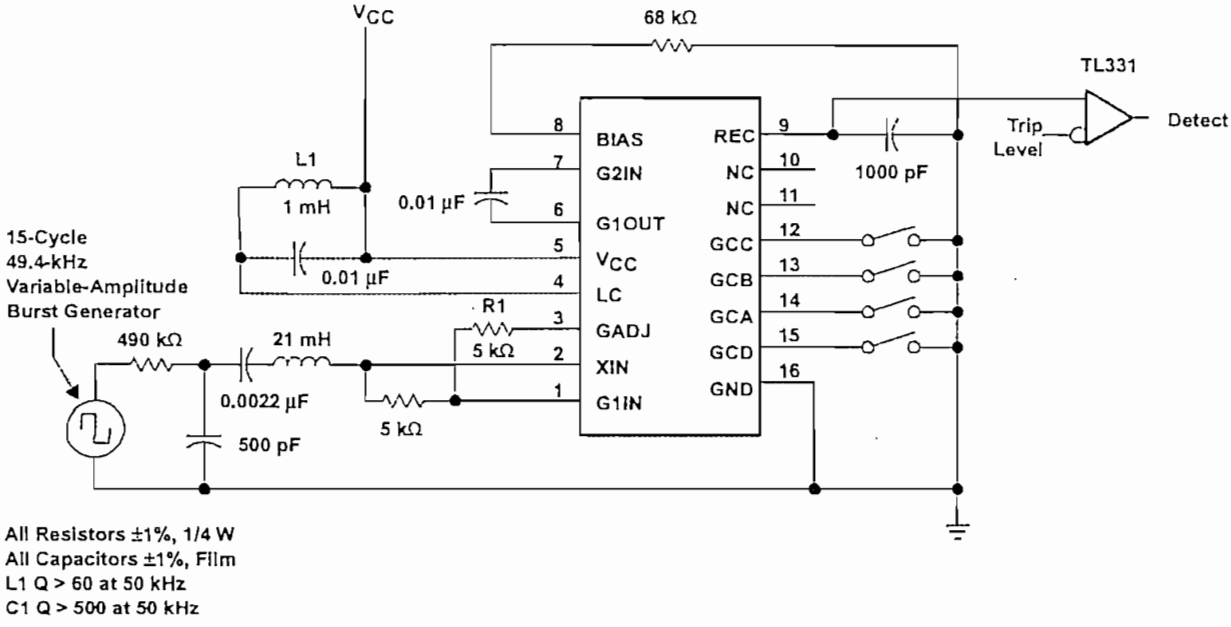
PARAMETER	TEST CONDITIONS		MIN	TYP‡	MAX	UNIT
Input clamp voltage at XIN	I_I = 40 mA				2.5	V
	I_I = – 40 mA				– 1.5	
Open-circuit input voltage at GCA, GCB, GCC, GCD	V_{CC} = 5 V,	I_I = 0		2.5		V
High-level input current, I_{IH}, into GCA, GCB, GCC, GCD	V_{CC} = 5 V,	V_{IH} = 2 V		– 0.5		mA
Low-level input current, I_{IL}, into GCA, GCB, GCC, GCD	V_{CC} = 5 V,	V_{IL} = 0			– 3	mA
Raceive output current	I_{G2IN} = – 100 µA,	V_O = 0.3 V		1		µA
	I_{G2IN} = 100 µA,	V_O = 0.1 V		– 50		
Supply current, I_{CC}					45	mA

‡ Typical values are at V_{CC} = 5 V and T_A = 25°C.

APPLICATION INFORMATION

detect level versus gain step

Detect level is measured by applying a 15-cycle burst of 49.4 kHz square wave just after the beginning of the gain step to be tested. The least burst amplitude that makes REC reach the trip level is defined to be the detect level. System gain is then inversely proportional to detect level. See the test circuit in Figure 1.



All Resistors ±1%, 1/4 W
All Capacitors ±1%, Film
L1 Q > 60 at 50 kHz
C1 Q > 500 at 50 kHz

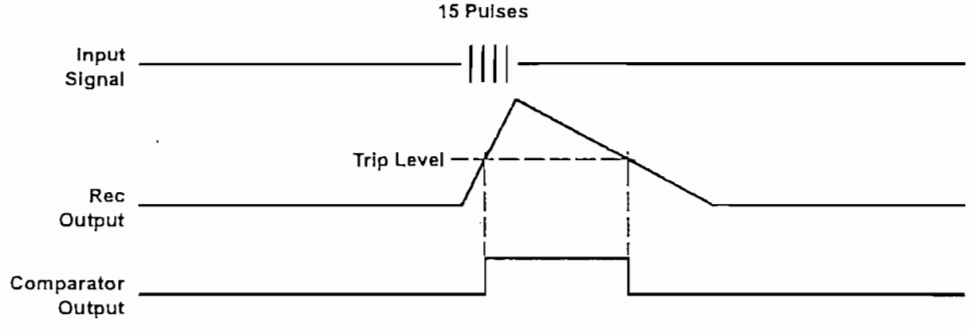


Figure 1. Detect-Level Measurement Circuit and Waveforms

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251–1443

APPLICATION INFORMATION

GAIN STEP TABLE

GCD	GCC	GCB	GCA	STEP NUMBER
L	L	L	L	0
L	L	L	H	1
L	L	H	L	2
L	L	H	H	3
L	H	L	L	4
L	H	L	H	5
L	H	H	L	6
L	H	H	H	7
H	L	L	L	8
H	L	L	H	9
H	L	H	L	10
H	L	H	H	11

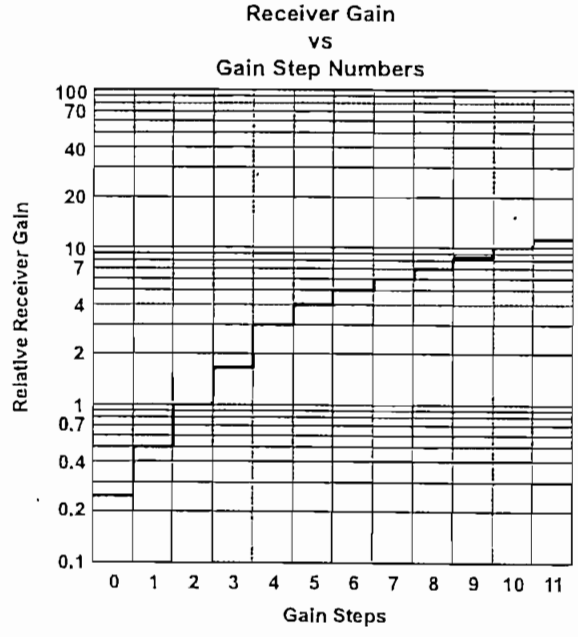


Figure 2

National Semiconductor

November 2000

Jameco Part Number 23966

LM158/LM258/LM358/LM2904
Low Power Dual Operational Amplifiers

General Description

The LM158 series consists of two independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

Application areas include transducer amplifiers, dc gain blocks and all the conventional op amp circuits which now can be more easily implemented in single power supply systems. For example, the LM158 series can be directly operated off of the standard +5V power supply voltage which is used in digital systems and will easily provide the required interface electronics without requiring the additional ±15V power supplies.

The LM358 is also available in a chip sized package (8-Bump micro SMD) using National's micro SMD package technology.

Unique Characteristics

- In the linear mode the input common-mode voltage range includes ground and the output voltage can also swing to ground, even though operated from only a single power supply voltage.
- The unity gain cross frequency is temperature compensated.
- The input bias current is also temperature compensated.

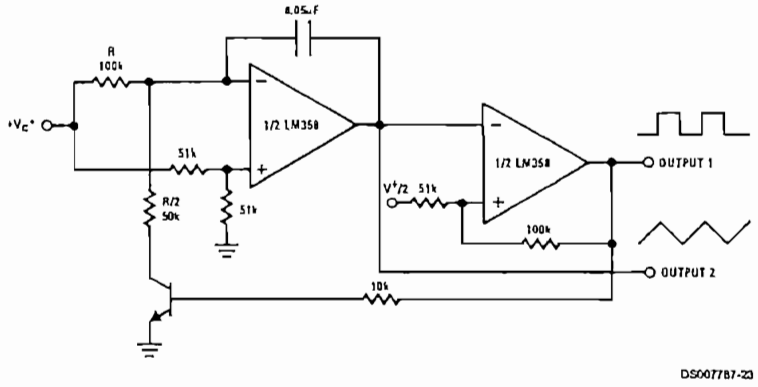
Advantages

- Two internally compensated op amps
- Eliminates need for dual supplies
- Allows direct sensing near GND and V_{OUT} also goes to GND
- Compatible with all forms of logic
- Power drain suitable for battery operation
- Pin-out same as LM1558/LM1458 dual op amp

Features

- Available in 8-Bump micro SMD chip sized package, (See AN-1112)
- Internally frequency compensated for unity gain
- Large dc voltage gain: 100 dB
- Wide bandwidth (unity gain): 1 MHz (temperature compensated)
- Wide power supply range:
  — Single supply: 3V to 32V
  — or dual supplies: ±1.5V to ±16V
- Very low supply current drain (500 µA)—essentially independent of supply voltage
- Low input offset voltage: 2 mV
- Input common-mode voltage range includes ground
- Differential input voltage range equal to the power supply voltage
- Large output voltage swing: 0V to V+ − 1.5V

Voltage Controlled Oscillator (VCO)



DS007787-23

Absolute Maximum Ratings (Note 9)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

	LM158/LM258/LM358 LM158A/LM258A/LM358A	LM2904
Supply Voltage, V^+	32V	26V
Differential Input Voltage	32V	26V
Input Voltage	−0.3V to +32V	−0.3V to +26V
Power Dissipation (Note 1)		
Molded DIP	830 mW	830 mW
Metal Can	550 mW	
Small Outline Package (M)	530 mW	530 mW
micro SMD	435mW	
Output Short-Circuit to GND		
(One Amplifier) (Note 2)		
$V^+ \leq 15V$ and $T_A = 25°C$	Continuous	Continuous
Input Current ($V_{IN} < -0.3V$) (Note 3)	50 mA	50 mA
Operating Temperature Range		
LM358	0°C to +70°C	−40°C to +85°C
LM258	−25°C to +85°C	
LM158	−55°C to +125°C	
Storage Temperature Range	−65°C to +150°C	−65°C to +150°C
Lead Temperature, DIP		
(Soldering, 10 seconds)	260°C	260°C
Lead Temperature, Metal Can		
(Soldering, 10 seconds)	300°C	300°C
Soldering Information		
Dual-In-Line Package		
Soldering (10 seconds)	260°C	260°C
Small Outline Package		
Vapor Phase (60 seconds)	215°C	215°C
Infrared (15 seconds)	220°C	220°C
See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" for other methods of soldering surface mount devices.		
ESD Tolerance (Note 10)	250V	250V

Electrical Characteristics

$V^+ = +5.0V$, unless otherwise stated

Parameter	Conditions	LM158A			LM358A			LM158/LM258			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 5), $T_A = 25°C$		1	2		2	3		2	5	mV
Input Bias Current	$I_{IN(+)}$ or $I_{IN(-)}$, $T_A = 25°C$, $V_{CM} = 0V$, (Note 6)		20	50		45	100		45	150	nA
Input Offset Current	$I_{IN(+)} - I_{IN(-)}$, $V_{CM} = 0V$, $T_A = 25°C$		2	10		5	30		3	30	nA
Input Common-Mode Voltage Range	$V^+ = 30V$, (Note 7) (LM2904, $V^+ = 26V$), $T_A = 25°C$	0		$V^+-1.5$	0		$V^+-1.5$	0		$V^+-1.5$	V
Supply Current	Over Full Temperature Range $R_L = \infty$ on All Op Amps										
	$V^+ = 30V$ (LM2904 $V^+ = 26V$)		1	2		1	2		1	2	mA
	$V^+ = 5V$		0.5	1.2		0.5	1.2		0.5	1.2	mA

Electrical Characteristics

V^+ = +5.0V, unless otherwise stated

Parameter	Conditions	LM358			LM2904			Units
		Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 5) , T_A = 25˚C		2	7		2	7	mV
Input Bias Current	$I_{IN(+)}$ or $I_{IN(-)}$, T_A = 25˚C, V_{CM} = 0V, (Note 6)		45	250		45	250	nA
Input Offset Current	$I_{IN(+)} - I_{IN(-)}$, V_{CM} = 0V, T_A = 25˚C		5	50		5	50	nA
Input Common-Mode Voltage Range	V^+ = 30V, (Note 7) (LM2904, V^+ = 26V), T_A = 25˚C	0		$V^+-1.5$	0		$V^+-1.5$	V
Supply Current	Over Full Temperature Range $R_L = \infty$ on All Op Amps							
	V^+ = 30V (LM2904 V^+ = 26V)		1	2		1	2	mA
	V^+ = 5V		0.5	1.2		0.5	1.2	mA

Electrical Characteristics

V^+ = +5.0V, (Note 4), unless otherwise stated

Parameter		Conditions	LM158A			LM358A			LM158/LM258			Units
			Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Large Signal Voltage Gain		V^+ = 15V, T_A = 25˚C, $R_L \geq 2$ kΩ, (For V_O = 1V to 11V)	50	100		25	100		50	100		V/mV
Common-Mode Rejection Ratio		T_A = 25˚C, V_{CM} = 0V to $V^+-1.5$V	70	85		65	85		70	85		dB
Power Supply Rejection Ratio		V^+ = 5V to 30V (LM2904, V^+ = 5V to 26V), T_A = 25˚C	65	100		65	100		65	100		dB
Amplifier-to-Amplifier Coupling		f = 1 kHz to 20 kHz, T_A = 25˚C (Input Referred), (Note 8)		−120			−120			−120		dB
Output Current	Source	V_{IN}^+ = 1V, V_{IN}^- = 0V, V^+ = 15V, V_O = 2V, T_A = 25˚C	20	40		20	40		20	40		mA
	Sink	V_{IN}^- = 1V, V_{IN}^+ = 0V, V^+ = 15V, T_A = 25˚C, V_O = 2V	10	20		10	20		10	20		mA
		V_{IN}^- = 1V, V_{IN}^+ = 0V, T_A = 25˚C, V_O = 200 mV, V^+ = 15V	12	50		12	50		12	50		µA
Short Circuit to Ground		T_A = 25˚C, (Note 2), V^+ = 15V		40	60		40	60		40	60	mA
Input Offset Voltage		(Note 5)			4			5			7	mV
Input Offset Voltage Drift		R_S = 0Ω		7	15		7	20		7		µV/˚C
Input Offset Current		$I_{IN(+)} - I_{IN(-)}$			30			75			100	nA
Input Offset Current Drift		R_S = 0Ω		10	200		10	300		10		pA/˚C
Input Bias Current		$I_{IN(+)}$ or $I_{IN(-)}$		40	100		40	200		40	300	nA
Input Common-Mode Voltage Range		V^+ = 30 V, (Note 7) (LM2904, V^+ = 26V)	0		V^+-2	0		V^+-2	0		V^+-2	V