INSTITUTO TECNOLÓGICO DE SONORA

TRAZADOR DE GRÁFICAS DE BODE USANDO SÍNTESIS DIGITAL DIRECTA (DDS)

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

PRESENTA:

ARTURO AGANZA TORRES

CD. OBREGÓN, SONORA **AGOSTO 2009**

INSTITUTO TECNOLÓGICO DE SONORA

Hoja de Control para TESIS

Alumno: ARTURO AGANZA TORRES
Matrícula: 63687
Carrera: INGENIERO EN ELECTRÓNICA
Fecha de registro: 13 DE AGOSTO DE 2009

1. TEMA:

Trazador de gráficas de bode usando síntesis digital directa (DDS).

2. APROBACIÓN:

Fecha de aprobación: 13 DE AGOSTO DE 2009

DR. JUAN JOSÉ PADILLA YBARRA
DIRECTOR ACADÉMICO

M.C. RICARDO T. SOLIS GRANADOS
COORDINADOR DE CARRERA

3. LIBERACIÓN:

Ing. Andrés Othón Pizarro Lerma Nombre del asesor	firma	16/Julio/2009 fecha
Dr. Joaquín Cortez González Nombre del revisor	firma	13/Agosto/2009 fecha
M.C. Raymundo Márquez Borbón Nombre del revisor	firma	10/Agosto/2009 fecha
Ing. Darcy Daniela Flores Nieblas Nombre del revisor	Darcy D. Flores N. firma	13/Agosto/2009 fecha

4. ATENTAMENTE:

AL JEFE DEL DEPARTAMENTO DE REGISTRO ESCOLAR:

INFORMO A USTED QUE EL ALUMNO HA CUMPLIDO CON LOS REQUISITOS DE TITULACIÓN

ATENTAMENTE:
COORDINADOR DE LA CARRERA

M.C. RICARDO T. SOLÍS GRANADOS

C-006

A Dios, a mí
a mi madre, a mi hermano,
a mis amigos y a mi familia.

AGRADECIMIENTOS

A Dios, por darme la oportunidad de estar en este mundo, por ser como soy, y no cambiaria por nada las cosas que he vivido, la gente que he conocido y los momentos que he tenido, bueno a lo mejor un detalle o dos cambiaria nada más.

A mi madre Alma Delia Torres Hernández, por darme la vida, por hacerme como soy, por apoyarme toda la vida a pesar de los obstáculos y darme una educación, por aguantarme y ser persistente conmigo para que me titulara, por todo lo que me has dado, eres la mejor, te quiero mucho.

A mi hermano Alejandro Aganza Torres, por apoyarnos en todo lo que pudiste, por estar ahí cuando necesite de un apoyo, aunque a veces no lo noté, gracias.

A mis amigos de toda la vida Gonzalo López Cortes, José Ramón García Mendivil y Joel Alejandro Velazquez Campaña, por la compañía en las buenas y en las malas y por su amistad.

A mis amigos Aida Aracely Flores Nolasco, Juan Eduardo Frias, Jorge Ruiz “Michi”, Armando León, Armando Loaiza “el padre”, Jesús Gutiérrez “el maza”, Oliver, José Ángel Peñuelas, Gerardo Gaxiola, al equipo de futbol soccer “Slam Drunk”, gracias por su amistad, y los momentos que tuvimos en la escuela y fuera de ella.

A mi asesor Andrés Othón Pizarro Lerma, por su ayuda, su apoyo, consejos, por estar atento al seguimiento del proyecto y por su conocimiento, gracias a eso no estaría donde me encuentro ahora o no hubiese concluido con este proyecto.

A los laboratorista Juanito, Antonio, Emmanuel y Chema, por aguantarme cada vez que pedía material, y que a final de cuenta fue la base de una amistad, gracias.

A mi familia, maestros que tuve durante mi carrera, gracias por compartir sus conocimientos, por sus consejos, su apoyo y su paciencia para enseñar tanto en la vida como en los estudios.

AGRADECIMIENTOS

A todas las personas que me topé durante el transcurso de educación profesional, demás estudios y en la vida, y que tuve el placer de conocer y toparme con ellos, aunque no nos llegamos a conocer muy a fondo, me brindaron su amistad y aunque me hubiese gustado conocerlos un poco más a fondo por razones del destino no se pudo.

A todos, gracias totales!!!.

ÍNDICE

Capítulo III. Desarrollo

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS Y ABREVIATURAS

AC	Corriente alterna (Alterning Current)
ADC	Convertidor analógico-digital (Analogic-Digital Converter)
CAN	Controlador de red de área (Controller Area Network)
CS	Selector de chip (Chip Select)
CPU	Unidad central de procesamiento (Central Processing Unit)
dB	Decibles
DAC	Convertidor digital-analógico (Digital-Analogic Converter)
DC	Corriente directa (Direct Current)
DDS	Síntesis digital directa (Direct Digital Synthesis)
DSP	Procesador de señal digital (Digital Signal Processor)
FFT	Transformada rápida de Fourier (Fast Fourier Transform)
C.I.	Circuitor integrado
IDE	Entorno de desarrollo integrado (Integrated Development Environment)
LCD	Pantalla de cristal líquido (Liquid Crystal Display)
MFC	Fundación de clases de Microsoft (Microsoft Foundation Classes)
OP-AMP	Amplificador operacional (Operational Amplifier)
PC	Computadora personal (Personal Computer)
PIC	Controlador de interfaz de periféricos (Peripheral Interface Controller)
pot	Potenciómetro
POO	Programación Orientada a Objetos
PGA	Amplificador de ganancia programable (Programmable Gain Amplifier)
RISC	Computadora con un conjunto reducido de instrucciones (Reduced Instruction Set Computer)
SFR	Registros de función especial (Special Function Register)
USB	Bus universal en serie (Universal Serial Bus)

Resumen

El estudio de la respuesta en frecuencia de los filtros electrónicos es un área muy importante en la electrónica, por lo tanto es importante crear un sistema que se encargue del estudio de la respuesta en frecuencia de los filtros electrónicos de manera rápida y fácil, con la ayuda de nuevos protocolos de comunicación como lo es el USB el cuál es una nueva tecnología conocida como plug and play, el sistema se divide en tres etapas básicas en las cuales se basa el proyecto:

- Generar la señal de entrada al filtro.
- Adecuar la señal generada.
- Capturar la magnitud y fase del filtro electrónico en estudio.

RESUMEN

En el presente trabajo se presentará de manera detallada los pasos que se llevaron a cabo para la realización del Bode Plotter o del sistema que se encarga del estudio de la respuesta en frecuencia de los filtro electrónicos, el presente trabajo se divide en cuatro capítulos.

En el capítulo I se presentan los antecedentes teóricos de este proyecto, el primer sistema propuesto para la resolución del problema. Se exhiben los objetivos que se esperan alcanzar a lo largo del proyecto.

El capítulo II se expone los conceptos básicos y la teórica necesaria para la compresión y el desarrollo de este trabajo. Se desarrollan temas como la generación de la señal por medio del método de la síntesis digital directa conocida como DDS, también temas como el protocolo USB, el protocolo SPI y la programación orientada a objetos.

EL capítulo III contiene la descripción detallada de cómo el Bode Plotter fue desarrollado, iniciando con la descripción general del sistema, pasando por el diseño, el análisis de varias partes del código del Firmware del programa, introduciéndonos cada vez más a detalle de cómo se realizó el mismo diseño del mismo.

El capítulo IV se muestran las pruebas realizadas al sistema, se verifican los objetivos propuestos y las mejoras que se tienen en diseños anteriores, analizando sistemas reales y comparando resultados de dichas pruebas con el diseño anterior.

Y para finalizar el proyecto se presentan las conclusiones obtenidas al terminar el proyecto, así como las recomendaciones y propuestas de mejoras que pudieran realizarse al sistema propuesto en este proyecto.

CAPÍTULO I
Introducción

En la electrónica, una de sus áreas de estudio son los filtros electrónicos debido a su gran uso en la industria, en el procesamiento digital de señales, que abarca desde el área de telecomunicaciones, como por ejemplo, en lo que es el procesamiento de audio, así también como el procesamiento de señales de video; en el área aeroespacial para el procesamiento de imágenes, el análisis sensorial inteligente a distancia por las ondas espaciales, y compresión de datos, también en el área de la medicina para imágenes de diagnóstico, análisis de electrocardiogramas, almacenamiento de imágenes médicas, sin olvidar en el área comercial como los efectos especiales para películas, video llamadas en conferencia, compresión de

imágenes y sonido para presentaciones multimedia, en el área de la telefonía el uso de filtros para la compresión de datos y voz; la reducción de ruido y el filtrado, en el área militar el uso de filtros para radares, sonares, comunicación segura y en artefactos de orientación y otras de las muchas áreas tal como el área científica en el análisis espectral, modelado y simulación, esto en lo que se refiere a filtros digitales. Los filtros analógicos tienen otras áreas en las que se aplican, tal como en el área de potencia, audio, filtros resonantes y en muchas más áreas de la electrónica.

Es por eso la importancia del estudio de los filtros electrónicos, por que tienen un gran auge en la industria, en el comercio, en la medicina, en el manejo de potencia eléctrica, y en muchas áreas ya mencionadas antes y muchas otras más que no fueron mencionadas, esto por que la electrónica es una rama de la física muy amplia y ocuparía bastante tiempo y trabajo mencionar cada una de ellas y el uso que tienen los filtros electrónicos en cada una de ellas.

Por ese motivo que se crea un dispositivo que se encargue de estudiar la respuesta de los filtros electrónicos de manera fácil y rápida, usando componentes de alta tecnología como lo son los microcontroladores, usando también dispositivos de nueva generación que utilizan el método para generar señales sinusoidales, rectangulares y cuadradas entre otras, con una amplitud y fase variables conocida como el método DDS (Síntesis Digital Directa por sus siglas en inglés, Direct Digital Synthesis). Usando protocolos y puertos de comunicación con la computadora más viables, rápidos y de menor costo como lo es la comunicación USB, y también uno de los IDE más nuevos en el entorno de la programación y más fáciles de usar debido a que se puede programar de manera más fácil usando *Windows Forms* que es una programación de manera gráfica. Esto y más se mencionará en este proyecto para lograr el objetivo principal, el análisis de filtros electrónicos.

1.1 Antecedentes

El estudio de los filtros electrónicos siempre ha sido de suma importancia ya que son utilizados en muchos proyectos de electrónica, debido que éstos tienen como función manipular y modificar el espectro de frecuencia de la señal de entrada para obtener en la salida la función que se requiera aplicar a diferentes sistemas electrónicos. Es por eso que este proyecto se dedica al estudio matemático de filtros electrónicos. Estos filtros se diseñan a partir de un modelo matemático, para así lograr el comportamiento que se desea y son lineales debido a su comportamiento que se mencionará más adelante.

Es por el estudio de su respuesta en frecuencia que podemos saber el comportamiento de un filtro mediante gráficas y que mejor que diseñar un instrumento que nos facilite la tarea de conocer la respuesta en frecuencia de un filtro, objetivo de este proyecto, un instrumento que nos otorgue exactamente el comportamiento del filtro a diferentes frecuencias, dándonos a conocer posibles fallas en su diseño. La creación de un instrumento que nos pueda proporcionar por medio de gráficas la magnitud y la fase de su respuesta en frecuencia, es una herramienta que proporciona tanto al maestro como el alumno una reducción del tiempo en el proceso de revisión del filtro en estudio.

Gracias a la ayuda de los microcontroladores es que podemos realizar este análisis, controlando o coordinando cada recurso, componente y proceso involucrado, debido a que el microcontrolador es una herramienta la cual tiene muchos recursos como son la transmisión de datos por SPI™, el módulo USB y puertos de entrada y salida, y una de las ventajas es que se puede reducir la cantidad de circuitos electrónicos utilizados cambiando solamente el Firmware del microcontrolador, así también como a la ayuda de las computadoras, debido a que pueden facilitarle mucho trabajo duro al microcontrolador.

Uno de los puntos a remarcar de este proyecto es la versión actual del mismo, la cuál es la tercera versión que se ha realizado, en la que se pretende mejorar varios puntos los cuales son: utilizar técnicas más avanzadas y exactas en la etapa de la generación de la señal, aumentar el número de frecuencias en las que se puede analizar el filtro, ya que se puede elegir y limitar el rango de frecuencias que se desea analizar, así como los pasos, a comparación de la versión anterior [11, 12] en la que sólo se podían analizar 62 frecuencias desde 1Hz a 100kHz, y claro uno de los puntos más importantes es el uso del puerto USB para la comunicación con la computadora, el cuál hace la comunicación más sencilla, con menos circuitería ya que el microcontrolador utilizado ya contiene un módulo que crea esta comunicación.

1.2 Justificación

El proceso del análisis de filtros lineales requiere de instrumentos que nos otorguen la respuesta en frecuencia de los filtros, los cuales se pueden encontrar en el mercado como los son los analizadores de espectros, pero son de costos muy elevados y de acceso muy limitado, haciendo imposible la adquisición de uno ó del uso de unos de estos equipos para los estudiantes de electrónica de Instituto Tecnológico de Sonora (ITSON). Debido a eso, el principal objetivo de este proyecto es facilitar, principalmente a los estudiantes de electrónica, un útil y valioso instrumento para el análisis del diseño de filtros lineales.

1.3 Planteamiento del problema

Diseñar e implementar un instrumento analizador de filtros lineales, que represente el comportamiento de los filtros electrónicos en magnitud y fase en gráficas logarítmicas conocidas por el nombre de Gráficas de Bode. Este proyecto se divide en tres partes debido a su complejidad: el hardware del dispositivo, un Firmware que controle los circuitos electrónicos del hardware y envíe los datos a la computadora, y la parte de la interfaz con la computadora que reciba los datos provenientes del microcontrolador y los decodifique para graficarlos en la computadora. Y así corregir los problemas de

la versión anterior los cuáles eran: la generación de la señal menos eficiente y más complicada, la comunicación con la computadora mediante el puerto serie, que es un puerto obsoleto y más lento que la comunicación USB y que necesitaba de un IC que convertía la señal del microcontrolador al protocolo RS232 de 15 volts, en fin, hay muchos puntos que se mejoraron y que veremos con más tiempo para detallarlos en otro capítulo.

1.4 Objetivos

Crear un instrumento trazador de Gráficas de Bode con interfaz con la computadora capaz de representar gráficamente la respuesta del filtro electrónico en magnitud y fase.

1.4.1 Objetivos específicos

- Desarrollar el hardware para el trazador reduciendo el número de componentes usados en la versión anterior.
- Un análisis más rápido y exacto.
- Comunicación del microcontrolador a la PC por medio de puerto USB.
- Lograr la interfaz gráfica de la PC para que funcione por medio del puerto USB.
- Usar un nuevo dispositivo para generar la señal de entrada para el filtro analizar, como el circuito integrado AD9833 de Analog Devices que utiliza el método DDS para generar la señal sinusoidal.

1.5 Hipótesis

El instrumento creado en este proyecto nos servirá para el análisis rápido, eficaz, eficiente y exacto del filtro en estudio, creado a partir de una interfaz en la computadora y un Firmware en un microcontrolador. En esta versión se mejorarán aspectos tales como, medir la señal con un mayor rango de atenuación y ganancia,

analizar un número más amplio de frecuencias y poder conectar a más computadoras este dispositivo debido al uso del puerto USB, ya que muchas de las computadoras ya no cuentan con puertos serie o paralelo, sobre todo las computadoras portátiles, y los puertos seriales como los paralelos ya se han vuelto con el tiempo más obsoletos.

1.6 *Limitaciones y delimitaciones*

Algunos de los factores que limitaron este proyecto fueron:

- Dificultad para utilizar circuitos de montaje superficial como lo son los circuitos integrados de Mini Small Outline Package (MSOP) y Shrink Small Outline Packaged (SSOP).
- El tiempo de espera de los materiales y la búsqueda de los mismos fue un factor clave por la cual este proyecto tardó en realizarse, debido a los dispositivos de montaje superficial utilizados y también a que la institución no cuenta con los materiales para la utilización de los mismos.
- El uso de librerías ya hechas por otros fabricantes de software, ya que las licencias son caras.

Las características que se delimitaron para este proyecto fueron:

- Despliegue final de las gráficas de Bode por medio de una computadora.
- El uso de nueva tecnología como el circuito integrado generador de señales utilizando el método conocido como DDS que genera señales sinusoidales muy precisas a comparación con otro tipo de dispositivo usado en el proyecto anterior [11, 12].
- La comunicación entre el microcontrolador y la PC es por medio de conexión USB.
- Análisis de los filtros limitados al estudio de frecuencias entre 1 Hz y 100 KHz.

- El usuario de este dispositivo es el responsable de diseñar un filtro con bajo nivel de OFFSET y ruido para que le permita al dispositivo analizar el filtro de una manera más exacta y precisa.
- Y derivado del punto anterior, si se analiza un sistema ó filtro, éste tiene que ser lineal.

CAPÍTULO II

Marco teórico

En este capítulo se presentan los términos y conceptos básicos usados y desarrollados a lo largo de este proyecto, correspondiente a los métodos y recursos utilizados por el mismo.

La ingeniería electrónica se encarga tanto de diseñar un circuito electrónico como del estudio de ellos a partir de su análisis matemático. Es por eso que para el estudio de un circuito electrónico es más fácil conocer su comportamiento por medio de gráficas sin saber con anterioridad la composición de éste, de ahí la necesidad de obtener esas gráficas. En este proyecto sólo nos enfocaremos al estudio de la respuesta en

frecuencia de los filtros electrónicos lineales através del estudio de su comportamiento en magnitud y fase.

2.1 Respuesta en frecuencia

Por el término de respuesta en frecuencia, se entiende la respuesta en estado de régimen permanente o estado estacionario de un sistema ante una entrada sinusoidal. Uno de los métodos de estudio de respuesta en frecuencia más comunes, está el variar la frecuencia de la señal de entrada en un cierto rango y estudiar su respuesta en magnitud y fase.

Para un sistema lineal e invariante en el tiempo con entrada sinusoidal $x(t) = X\text{sen}\omega t$, como el que se muestra en la figura 2.1, cuya función de transferencia está dada por la relación de salida sobre entrada con condiciones iniciales (C.I.) iguales a cero, la cuál para un análisis más sencillo se traslada del domino del tiempo al dominio en frecuencia mediante la transformada de *Laplace*, por lo tanto la función de transferencia se define como:

$$G(s) = \frac{Y(s)}{X(s)};\ C.I. = 0 \tag{2.1}$$

Se tiene que la función de transferencia *G(s)* puede ser escrita como una relación de dos polinomios en *s*, es decir

$$G(s) = \frac{p(s)}{q(s)} = \frac{k(s+a)}{(s+b_1)(s+b_2)(s+b_3)\cdots(s+b_n)} \tag{2.2}$$

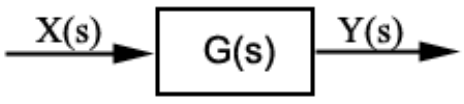


Figura 2.1. Sistema lineal.

2.2 Gráficas de Bode

Se puede representar una función de transferencia sinusoidal por dos diagramas distintos, uno de la amplitud en función de la frecuencia, y el otro del ángulo de fase en función de la frecuencia. Un diagrama de Bode entonces consta de dos trazos, uno es el diagrama del logaritmo del módulo de una función de transferencia sinusoidal; el otro es un diagrama del ángulo de fase. Ambos son representados en función de la frecuencia en escala logarítmica.

La representación normal de la amplitud logarítmica *G(s)* es $20\log|G(s)|$, con base de logaritmos igual a 10. La unidad utilizada para representación de amplitud es el decibel (dB). La ventaja principal de usar diagrama logarítmico, es que se pueden convertir las multiplicaciones de amplitudes en adición. Se hace notar que se puede realizar de manera simple la determinación experimental de una función de transferencia, si se representan los datos de respuesta de frecuencia, en forma de un diagrama logarítmico. La representación logarítmica es útil, porque representa las características de altas y bajas frecuencias de la función de transferencia en un solo diagrama, lo cual sería demasiado complicado lograrlo con una escala lineal. Lo anterior permite expandir el rango de visualización de bajas frecuencias hasta muy altas frecuencias.

2.3 Magnitud en decibel (dB)

La gráfica de proporciones iguales en desplazamientos iguales es llamada *Escala Logarítmica*, o también conocida como *Escala en decibeles*, esta escala está definida por la siguiente ecuación:

$$R(dB) = 10\log_{10} R \tag{2.3}$$

En donde *R*, es la proporción que existe entre dos potencias P_1 y P_2, como se describe a continuación:

$$Ganancia(dB) = 10\log_{10}(\frac{P_2}{P_1}) \tag{2.4}$$

Para un sistema eléctrico, en donde, si se están considerando magnitudes de voltaje de entrada y salida cambiantes sobre un mismo valor de impedancia de entrada y salida iguales se tiene que:

$$Ganancia(dB) = 10\log_{10}(\frac{V_{sal}{}^2 / Z}{V_{ent}{}^2 / Z}) \tag{2.5}$$

$$Ganancia(dB) = 10\log_{10}(\frac{V_{sal}{}^2}{V_{ent}{}^2}) = 10\log_{10}(\frac{V_{sal}}{V_{ent}})^2 \tag{2.6}$$

Por propiedades de logaritmos se tiene que:

$$Ganancia(dB) = 20\log_{10}\left|\frac{V_{sal}}{V_{ent}}\right| \tag{2.7}$$

La cual se conoce como ganancia normalizada por el hecho de considerar iguales a la impedancia de salida y la de entrada.

2.4 Filtros electrónicos

Empezaremos por definir que es un filtro. La palabra filtro es usada muy comúnmente, tal como un filtro de aceite para el motor de un automóvil, como también el filtro de aire y de combustible del mismo. Un filtro de aire se usa para filtrar aire en los aparatos de aire acondicionado. Incluso en las cámaras fotográficas se usan algunos tipos de lentes que filtran la imagen permitiendo el rechazo de algunas longitudes de onda. En general un filtro se puede definir como un dispositivo que modifica de un modo la entrada para obtener una salida deseada. Ya que definimos que son los filtros en forma general, pasemos a los filtros electrónicos.

Un filtro electrónico es aquel que modifica la señal de entrada, que en este caso es una señal eléctrica, para obtener a la salida la señal deseada, que puede ser libre de armónicos, ruido, etc. En este proyecto se analizaran más que nada el comportamiento del filtro en su respuesta en frecuencia, tanto en magnitud como en fase. Existen diversos tipos de filtros como los son: los lineales y no lineales, los

analógicos y digitales, los activos y pasivos, etc. los cuáles se van a definir más adelante, el proyecto del trazador de gráficas de Bode con interfaz en la computadora sólo se enfocará en filtro lineales, que también explicaremos más adelante el por que solamente este tipos de filtros.

2.5 Clasificación de filtros

Los filtros se pueden clasificar en diferentes tipos según sus elementos que lo conforman para realizar el filtrado de una señal, los cuáles se dividen en los siguientes.

2.5.1 Filtros lineales y no lineales

Los filtros lineales o sistemas lineales invariantes en el tiempo (para hablar de forma más general) son aquellos que cumplen con tres requerimientos que son: homogeneidad, invariancia en el tiempo y aditividad.

Si el sistema tiene *homogeneidad*, la multiplicación de la entrada del sistema por una ganancia constante es equivalente a multiplicar la salida por la misma ganancia constante:

$$H\left[ax\right] = aH\left[x\right] \tag{2.8}$$

Donde a es la ganancia constante.

Si el sistema es *aditivo*, la salida producida por la suma de dos señales de entrada es igual a la suma de las salidas producida por cada entrada individualmente:

$$H\left[x_1 + x_2\right] = H\left[x_1\right] + H\left[x_2\right] \tag{2.9}$$

El sistema tiene *invariancia en el tiempo* si no cambia según el tiempo, si una específica entrada causa una específica salida, se puede asegurar que la misma específica entrada va a causar la misma específica salida en otro tiempo. Esto es

bueno, por que si no es así, significaría lidiar con sistemas que cambian su comportamiento o sus características mientras operan.

Los filtros o sistemas no lineales variantes con el tiempo son aquellos que no cumplen con las tres requerimientos anteriores, debido a que este tipo de sistemas no tiene una linealidad, por ejemplo, son afectados por cambios de temperaturas, a diferentes temperaturas tienen una diferente respuesta.

2.5.2 Filtros pasivos y activos

Los filtros pasivos se implementan en general con inductores y capacitores. Se caracterizan por que estos tipos de filtros no necesitan una fuente de alimentación externa, una de las desventajas de estos filtros es que dependiendo de la frecuencia a las que trabajan los dispositivos que utilizan para crearlos pueden ser muy voluminosos. *Los filtros activos* son aquellos que cuentan con una fuente de alimentación externa, debido a esto los componentes que lo conforman son más chicos y facilitan el diseño del mismo, proporcionan una gran amplificación de la señal de entrada (una ganancia), unas de las desventajas es que necesitan una fuente de alimentación externa y su respuesta está limitada por la capacidad de los amplificadores operacionales utilizados por lo que es imposible su aplicación en sistemas de media o alta potencia. A pesar de esto los filtros activos presentan cada vez mayor servicio al campo de la electrónica, especialmente en el área de instrumentación y dentro de las telecomunicaciones.

2.5.3 Filtros digitales y analógicos

Un filtro analógico utiliza circuitos electrónicos que hacen uso de resistencias, capacitores y amplificadores para producir el efecto requerido en el filtrado. Como los filtros usados para aplicaciones con reducción de ruido, trato de señales como las de video, etc.

El filtro digital es un sistema de tiempo discreto que puede realizar funciones de filtrado de una señal. Aprovecha avances de la tecnología digital para emular sistemas analógicos. Debe cumplir los requisitos necesarios para procesar las señales analógicas (teorema de muestreo). Los filtros digitales requieren un procesador digital para realizar cálculos numéricos de los valores muestreados de la señal, el procesador puede ser un ordenador corriente como lo puede ser un PC, o un chip especializado para ese tipo de cálculos como lo son los DSP (Digital Signal Processor o Procesador Digital de Señales por sus siglas en inglés).

2.6 Filtros ideales

2.6.1 Filtros pasa-bajas

Permite el paso de frecuencias bajas, estas son frecuencias menores a la frecuencia de corte denominada ω_c, frecuencias mayores a la frecuencia de corte ω_c se bloquea evitando así el paso de ellas como se muestra en la figura 2.2.

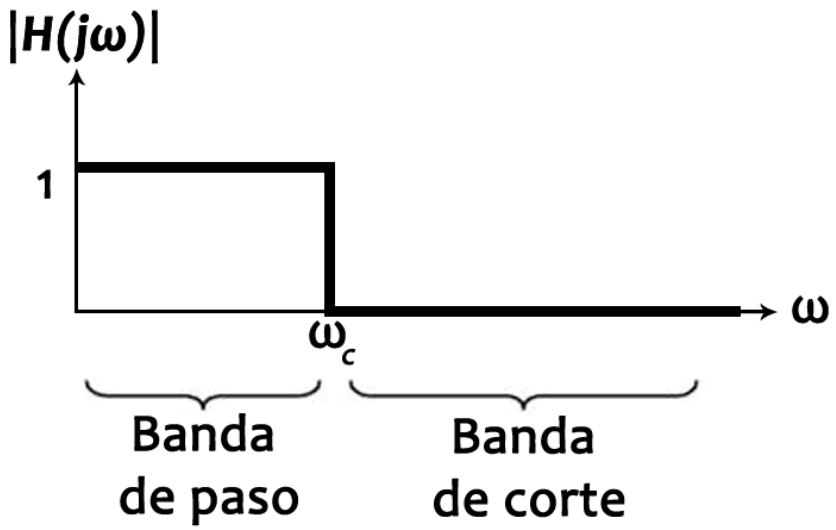


Figura 2.2. Diagrama de amplitud de un filtro pasa-bajas ideal.

2.6.2 Filtro pasa-altas

Permite el paso de las frecuencias mayores que cierta frecuencia ω_c, también denominada frecuencia de corte, y bloquea las menores. La respuesta se muestra en la figura 2.3.

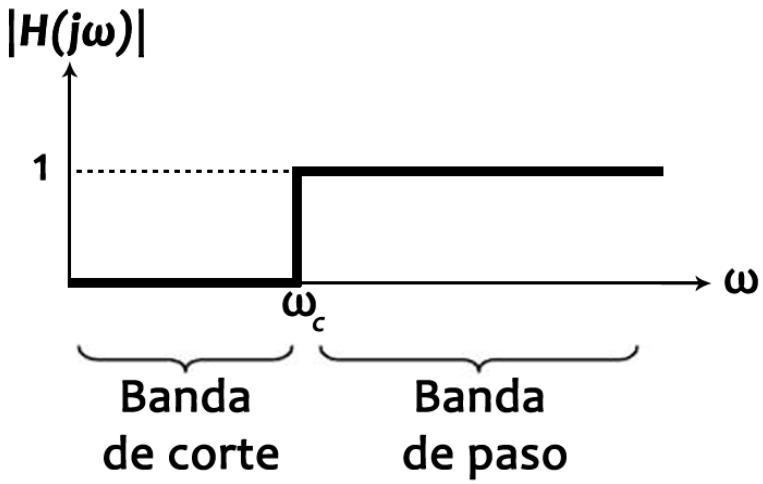


Figura 2.3. Diagrama amplitud de un filtro pasa-altas ideal.

2.6.3 Filtro pasa-banda

Permite el paso de las frecuencias comprendidas entre dos frecuencias ω_1 y ω_2, denominadas frecuencia inferior de corte y frecuencia superior de corte, bloqueando las restantes. Su respuesta se observa en la figura 2.4.

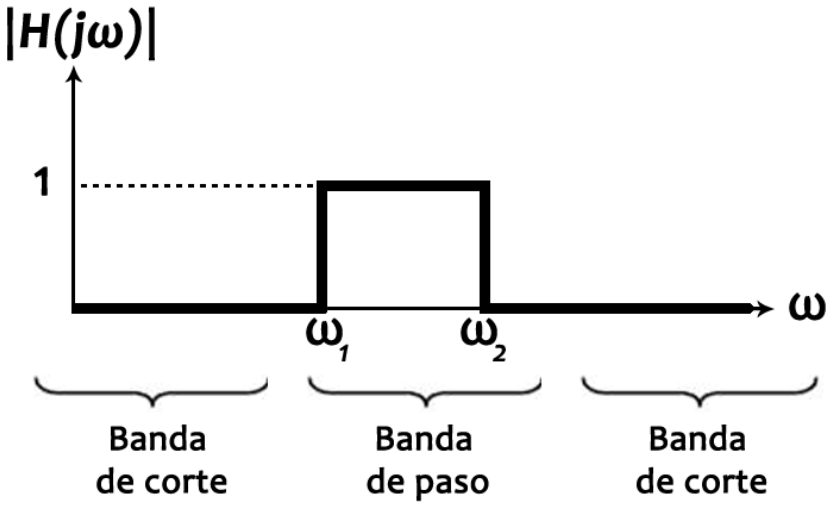


Figura 2.4. Diagrama de amplitud de un filtro pasa-bandas ideal.

2.6.4 Filtro rechaza-banda

Bloquea las frecuencias comprendidas entre las frecuencias de corte ω_1 y ω_2, dejando pasar las restantes. Su respuesta se muestra en la figura 2.5.

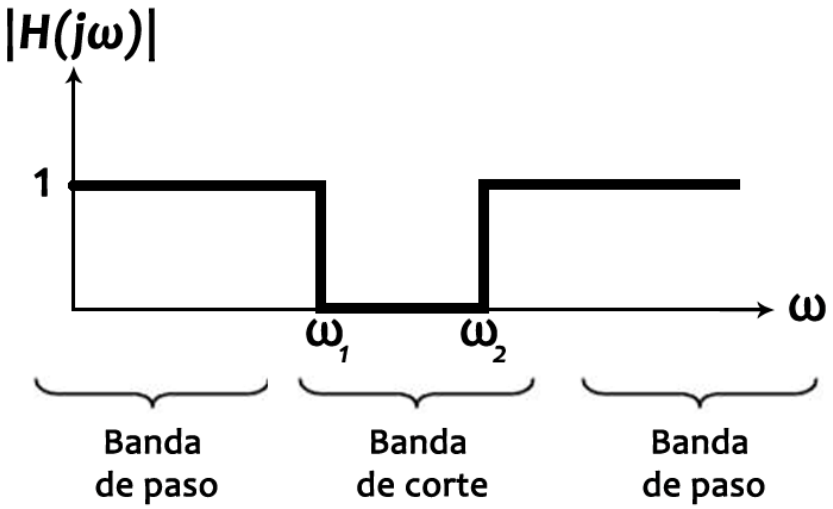


Figura 2.5. Diagrama de amplitud de un filtro rechaza-banda ideal.

2.7 Tipos de filtros según su aproximación matemática

Los filtros se pueden caracterizar por el tipo de respuesta que tienen a la frecuencia debido a su modelo matemático, algunos presentan rizos y otros no, solo se describirán algunos de esos tipos de filtros.

2.7.1 Filtros *Bessel*

Los *filtros Bessel* son diseñados para obtener una fase lineal o un retraso plano en la banda de paso, este filtro sólo tiene polos y debido a su linealidad son utilizados para aplicaciones de audio. Su función de trasferencia para un orden *n* de un filtro pasa-bajas está dada por:

$$H(s) = \frac{b_o}{q_n(s)} \tag{2.10}$$

Donde

$$q_n(s) = \sum_{k=1}^{n} b_k s^k \tag{2.11}$$

$$b_k = \frac{(2n-k)!}{2^{n-k} k!(n-k)!}$$

La respuesta en frecuencia de la magnitud de un filtro pasa-bajas *Bessel* en diferentes orden n se muestra en la figura 2.6.

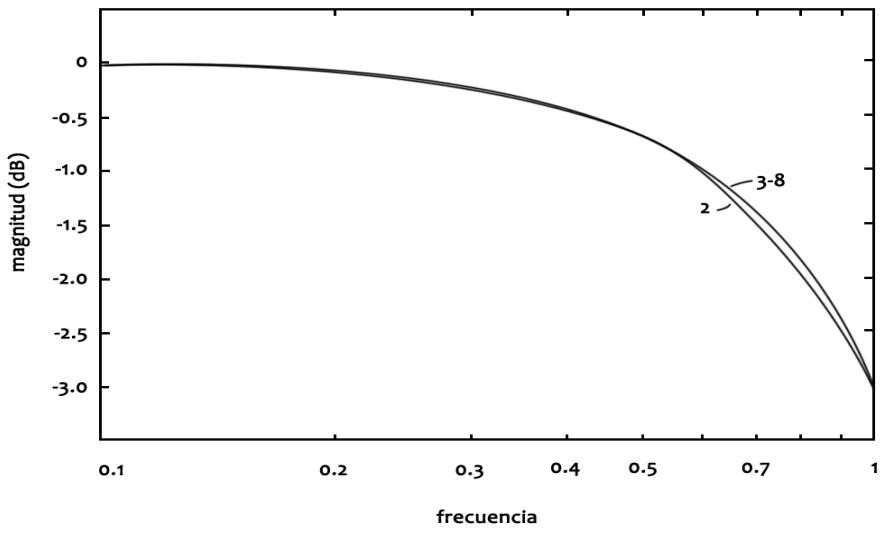


Figura 2.6. Respuesta en frecuencia en magnitud de un filtro *Bessel* pasa-bajas.

2.7.2 Filtros *Chebyshev*

Los filtros *chebyshev* son diseñados para obtener una respuesta de amplitud característica que tiene una relativa brusca transición de la banda de paso a la banda de rechazo. Esta selectividad se logra a expensas de ondas que se introducen en la respuesta. La función de transferencia para un filtro pasa-bajas *chebyshev* de orden n está dada por:

$$H(s) = \frac{H_0}{\prod_{i=1}^{n}(s - s_i)} = \frac{H_0}{(s - s_i)(s - s_2)\cdots(s - s_n)} \tag{2.12}$$

Donde: $H_0 = \prod_{i=1}^{n}(-s_1)$ para orden n impar. (2.13)

$H_0 = 10^{r/20}\prod_{i=1}^{n}(-s_1)$ para orden n par. (2.14)

La gráfica de respuesta típica en magnitud para un filtro *chebyshev* pasa-bajas se muestra en la figura 2.7.

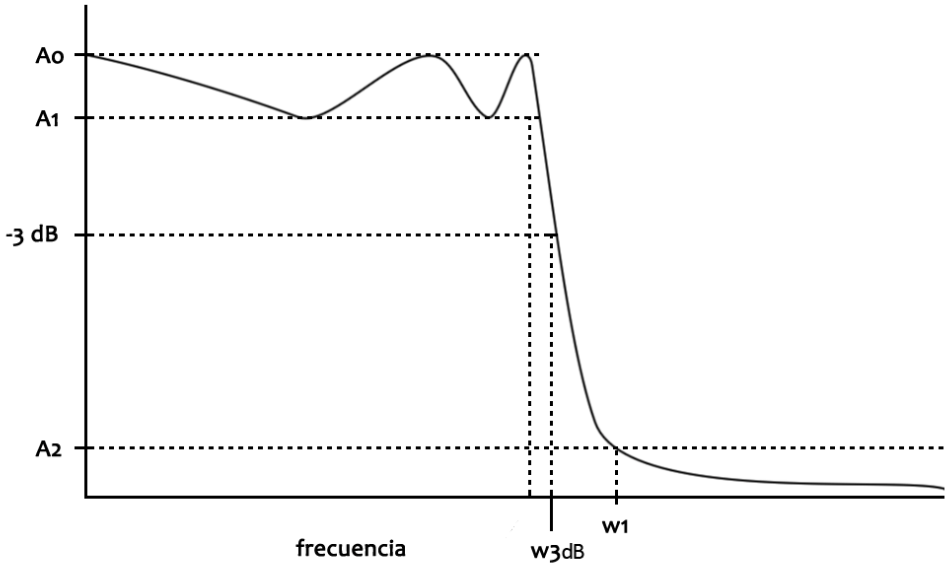


Figura 2.7. Respuesta típica en magnitud de un filtro *chebyshev* pasa-bajas.

2.7.3 Filtros *Butterworth*

Los filtros *Butterworth* son diseñados para obtener una amplitud de respuesta característica tan plana como sea posible a bajas frecuencias y se disminuya cada vez más con mayor frecuencia de una manera monotónica. La expresión general para la función de transferencia del filtro *Butterworth* en orden n está dada por:

$$H(s) = \frac{H_0}{\prod_{i=1}^{n}(s - s_i)} = \frac{H_0}{(s - s_i)(s - s_2)\cdots(s - s_n)} \tag{2.15}$$

Donde: $s_i = e^{jn[(2i+n-1)/2n]} = \cos\left(\pi \frac{2i+n-1}{2n}\right) + j\sin\left(\pi \frac{2i+n-1}{2n}\right)$ (2.16)

La respuesta típica en magnitud para un filtro *Butterworth* pasa-bajas de diferentes orden n está dada por la figura 2.8.

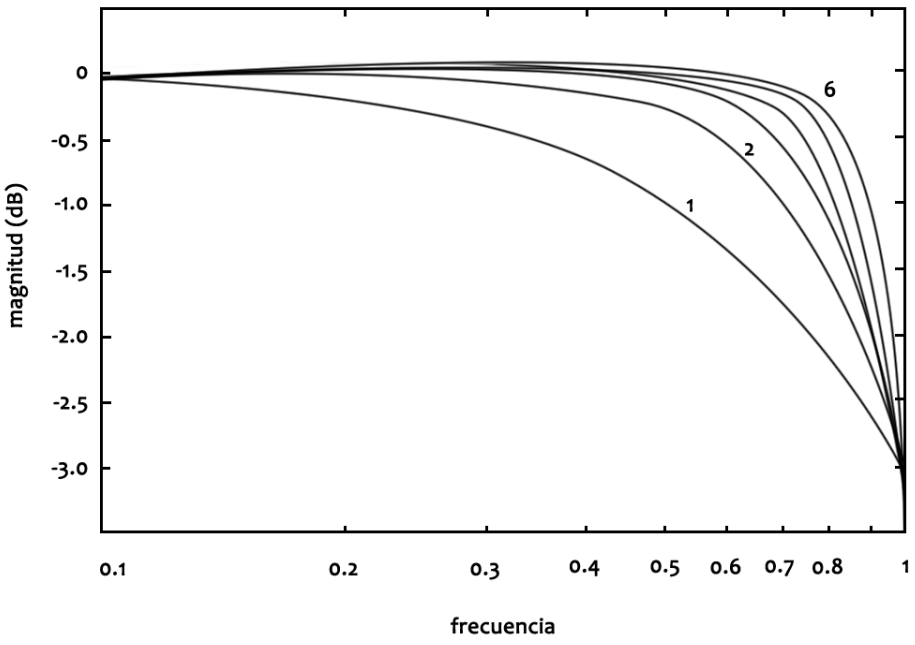


Figura 2.8. Respuesta típica en magnitud de un filtro *Butterworth* pasa-bajas de diferente orden n.

2.8 ¿Por qué usar filtros?

¿Por qué estamos tan interesados en usar filtros?, es una pregunta que se debe de hacer uno al crear un filtro. La respuesta a esa pregunta es cuando queremos detectar una señal deseada, pues es una tarea imposible si aun tenemos señales y ruido que no han sido removidas por medio del filtrado. Es por eso que los filtros electrónicos permiten el paso de algunas señales y detienen a otras. Para ser más precisos, los filtros permiten que algunas frecuencias de las señales de entrada aplicadas a la entrada del filtro pasen a través de él y que en su salida se obtenga la señal con poco o ningún nivel de reducción de la señal.

2.9 ¿Por qué analizar solamente filtros o sistemas lineales?

Como lo vimos anteriormente un sistema lineal es aquel que cumple con las propiedades de aditividad, homogeneidad e invarianza en el tiempo, se estudian solamente este tipo de sistemas debido a que sus cambios de amplitud, en este caso voltaje que atraviesa a los sistemas, es proporcional a la amplitud de sus extremos, y se puede saber su comportamiento más exacto, ya que los sistemas no lineales se comportan de una manera que es muy difícil estudiarlos matemáticamente, por que su comportamiento no es proporcional. Por ejemplo, un filtro no lineal es aquel que en su entrada se tiene una señal con frecuencia determinada y en su salida una frecuencia mayor o menor a la frecuencia de entrada, por lo tanto, ese filtro no es lineal.

2.10 Comunicación SPI™

La comunicación SPI (Serial Peripheral Interface, o interfaz periférica serial por sus siglas en inglés) es un protocolo de comunicación síncrono que posee un dispositivo maestro para iniciar la comunicación a otro dispositivo que funcionaría como el esclavo. Este módulo SPI™ es implementado dentro del microcontrolador por hardware y pudiendo tener comunicación un mismo módulo con varios dispositivos.

Este protocolo cuenta con varias señales. Una de las cuáles es la señal que sincroniza y controla cuando el dato es cambiado y cuando se debería leer, otra señal es la del chip select o slave select (CS o SS) y la última es el dato que se está enviando (SDO) y en el caso de recibir datos la línea de recibir datos (SDI). Existen dos tipos de modos de operación, en modo master (o maestro) y modo slave (o esclavo).

En el modo maestro se controla la señal de sincronización (SCK) y todos los dispositivos son controlados por el maestro, ningún esclavo debe manipular esta señal de sincronización. Y en el modo esclavo pues sucede lo mismo, el esclavo no puede controlar al maestro tiene y que esperar a que lo habiliten y mandar el dato sincronizado por la señal de sincronización (SCK).

El protocolo funciona de la siguiente manera; el maestro selecciona al dispositivo que desea enviar la información con el chip select, envía la señal de sincronización (SCK) y el dato (SDO) y el dispositivo a recibir el dato lo recibe en la entrada (SDI). En caso de que el maestro quiera recibir datos, éste tiene que cambiar al modo de esclavo.

2.11 Universal Serial Bus

2.11.1 Introducción

Conocido como Bus Universal en Serie (USB, por sus siglas en inglés, Universal Serial Bus) es uno de los temas más importantes de este proyecto, ya que es lo que lo diferencía de los proyectos pasados [11, 12] que se realizaron con una comunicación serial con el protocolo RS-232, en este proyecto se utiliza la comunicación USB debido a ciertos aspectos que se hablarán más adelante.

Una de las principales causas por las cuales se eligió este protocolo de comunicación es por que muchas de las computadoras de hoy en día traen puertos USB, debido a que muchos de los protocolos de comunicaciones pasados son muy

grandes y menos eficientes, y es donde se dirigen en un futuro las computadoras para ser más compactas y tener periféricos de comunicación más chicos en tamaño, económicos y rápidos, esto es debido a que las computadoras personales conocidas como laptop es muy poco común que traigan ya incorporados en ellas protocolos de comunicación viejos como lo son la comunicación serial RS-232 y paralelo. Hay mucho que decir para este protocolo del USB pero en este proyecto no entraremos muy a fondo, más que nada se revisará los aspectos más importantes, si se desea más información acerca del protocolo USB cheque la bibliografía y referencias del trabajo.

Las ventajas de utilizar este protocolo es que es muy fácil de usar, rápido, versátil, barato, soporta Windows y otros sistemas operativos.

2.11.2 ¿Qué puede hacer el USB?

USB es una solución muy buena para comunicarte a través de una computadora con un dispositivo afuera de ésta. La interfaz es adecuada para producirse en masa.

Para ser exitoso, una interfaz tiene dos tipos de audiencias: el usuario quien quiere usar el periférico y el desarrollador quien diseña el hardware y escribe el código para poderse comunicar con el dispositivo. USB tiene características para cumplir las dos.

2.11.3 Beneficios del uso del USB

Desde la perspectiva del usuario, los beneficios del USB son la facilidad de usarlo, es rápido y confiable para transferir información, flexible, de bajo costo, y conservación de energía. En la tabla 2.1 se presenta una comparación del USB con otro tipo de interfaces.

Tabla 2.1. Comparación de diferentes interfaces, donde no se especifica un máximo, la tabla muestra un máximo típico.

Interfaz	Formato	Número de dispositivos (máximo)	Distancia (máxima en metros)	Velocidad (máxima bits/seg.)	Uso típico
USB	Asíncrono serial	127	5 (hasta 30 pies con 5 hubs)	1.5M, 12M, 480M	Ratones, teclados, audio, otros y personalizados dispositivos
Ethernet	Serial	1024	488	10G	Comunicaciones generales de Internet
IEEE-1394b	Serial	64	92	3.2G	Video, almacenamiento masivo
IEEE-488	Serial	15	18	8M	Instrumentación
IrDA	Asíncrono Infrarrojo serial	2	2	16M	Impresoras
I^2C	Síncrono Serial	40	5.5	3.4M	Comunicaciones en microcontroladores
Microwire	Síncrono serial	8	3	2M	Comunicaciones en microcontroladores
SPI	Síncrono serial	8	3	2.1M	Comunicaciones en microcontroladores

Configuración automática. Cuando un usuario conecta un periférico USB al computador, el sistema operativo, en este caso Windows, detecta el periférico y carga el controlador apropiado para su uso.

Fácil de conectar. Con el USB no es necesario abrir la computadora para conectar el dispositivo o adherir una tarjeta de expansión para cada periférico. Una

computadora típica ya cuenta con por lo menos cuatro puertos USB. Y se pueden expandir el número de estos puertos agregando hubs o concentradores con puertos adicionales.

Conectores sencillos. El cable conector USB está definido en medidas estándar de la interfaz USB, por lo tanto es imposible conectarlos mal.

Gran conectividad. Se pueden conectar y desconectar un periférico USB cuando se desee, si o no el sistema y el periférico se alimentan, sin dañar la computadora o el dispositivo. El sistema operativo detecta cuando un periférico es adjuntado y se prepara para su uso.

El USB cuenta con muchos beneficios los cuáles no se hablarán a detalle en este proyecto, pero la siguiente lista revisa algunos de ellos.

2.11.4 Velocidad

El periférico USB soporta tres tipos de velocidades: high-speed a 480Mbps, full-speed de 12 Mbps y low-speed a 1.5Mbps. Los controladores host o los controladores del anfitrión en recientes PC soportan los tres tipos de velocidades.

El bus de velocidades describe el tipo de información que viaja a través del bus. Además de los datos, el bus debe tener el status de él mismo, el control, y las señales de error y auto chequeo. Además todos los periféricos USB deben compartir el mismo bus. Por eso que el tipo de información a transferir se puede esperar que sea menor que la velocidad del bus para cada periférico individual. La tasa máxima teórica para una simple transferencia de datos es acerca de 53 Mbps para los periféricos high-speed, 1.2 Mbps a full-speed y 800 Kbps a low-speed.

2.11.5 Beneficio para desarrolladores

Muchas de las ventajas descritas a continuación también hacen las cosas fáciles para los desarrolladores. Por ejemplo, Los cables del USB están definidos para estándares y chequeo automático de errores, lo que quiere decir que los desarrolladores no se tienen que preocupar acerca de las características del cable o proveer el chequeo del error por medio de software.

Las cuatro tipos de transferencias y los tres tipos de velocidades hacen de la interfaz una herramienta factible para muchos tipos de periféricos. Hay muchos diferentes tipos de transferencias que son adecuadas para el intercambio de pequeños o grandes bloques de información, con o sin contratiempos. Para información que no requiere retrasos, el USB puede garantizar el ancho de banda o el máximo tiempo entre transferencias.

A diferencia de otras interfaces, el USB no asigna funciones específicas a líneas de señales o hace hipótesis acerca de cómo la interfaz debe de ser usada. Por ejemplo, las líneas del status y control en el puerto paralelo de la PC fueron definidas con la intención de comunicarse con las líneas de las impresoras. Hay cinco líneas de salida con funciones asignadas tal que indican una condición de ocupado o de sin papel, en el caso de las impresoras. Es por eso que los desarrolladores comenzaron usando el puerto para escaners u otros periféricos que mandan grandes cantidades de información a la PC, teniendo cinco líneas de ese puerto limitadas debido a su funcionamiento. USB no hace tales hipótesis y es adecuado para cualquier tipo de periférico.

Para comunicarse con tipos comunes de periféricos como las impresoras, teclados, y controladores, USB ha definido clases que especifican los requerimientos del dispositivo y los protocolos. Los desarrolladores pueden usar las clases como guía en vez de tener que reinventar todo desde cero.

Sistemas operativos soportados. Windows 98 fue el primer sistema operativo con soporte para USB, y ediciones que le siguen incluyendo Windows 2000, Windows Me, Windows XP, Windows Server 2003, soportan el USB también.
Una reclamación de un sistema operativo que soporta USB puede significar muchas cosas. Al nivel básico, un sistema operativo que soporte USB debe de hacer tres cosas:

- Detectar cuando un dispositivo es conectado y removido del sistema.
- Comunicarse con un nuevo dispositivo adjunto debe encontrar la manera de intercambiar información con él.
- Proporcionar un mecanismo para que permita a los controladores comunicarse con el hardware de la PC y las aplicaciones desean tener acceso a un periférico USB.

En altos niveles, los sistemas operativos de apoyo también puede significar la inclusión de los controladores de las clases que permiten a los programadores de la aplicación para acceder a los dispositivos. Si el sistema operativo no incluye el controlador para el periférico, el vendedor del periférico debe proporcionar el controlador.

2.11.6 Límites de la interfaz USB

Toda interfaz de comunicación tiene sus límites que hacen a ésta impráctica para algunas aplicaciones. Para USB, los límites para preocuparse incluyen la velocidad y la distancia. Carecer de un apoyo para comunicaciones par a par, no capacidad de *broadcast*, la falta de soporte en hardwares viejos, y sistemas operativos viejos.

Velocidad. USB es muy versátil, pero no está diseñado para hacerlo todo. USB *high-speed* hace competencia a la interfaz IEEE-1394a (Firewire) a 400Mbps, pero IEEE-1394b es más rápida, a 800 Mbps y la IEEE-1394 3200 a 3.2Gbps.

Distancia. USB fue diseñado como un bus de expansión-escritorio con la expectativa de que los periféricos estarían relativamente al alcance de la mano. Un segmento del cable de USB puede ser tan largo como 5 metros. Otras interfaces, incluyendo RS-232, RS-485, IEEE-1394b, y Ethernet, permiten cables de una longitud más grande. Se puede incrementar el cable de la conexión del USB tan largo como 30 metros usando cables que conecten a 5 hubs y un dispositivo.

Comunicación par a par. Cada comunicación USB es entre un host, en este caso la PC, y un periférico. USB provee una solución parcial con USB On-The-Go. Un dispositivo On-The-Go puede funcionar como los dos, como un periférico y como un host de limitada capacidad que pueden comunicarse con otros dispositivos. Dos host pueden comunicarse unos con otros a través de un puente PC a PC de red por cable, que contiene dos dispositivos que conecte a cada uno diferentes PC y pasar datos entre PC's.

Broadcasting. USB no permite de ninguna manera mandar un mensaje simultáneamente a múltiples dispositivos. El host debe mandar el mensaje a cada dispositivo individualmente. Si tuvieras la necesidad de usar broadcasting, es conveniente usar otro tipo de interfaz como IEEE-1394 o Ethernet.

Estos son algunos de los límites que tiene la interfaz USB, es conveniente pues analizar primero que interfaz es la mejor para la aplicación que se desea usar, para evitar problemas.

2.11.7 USB versus IEEE-1394

Otra interfaz popular que escoger para los periféricos es la IEEE-1394. APPLE implementó un interfaz de computadora llamada Firewire. Generalmente, IEEE-1394 puede ser más rápida y flexible que USB pero es más cara de implementar. Con USB, un sólo host controla la comunicación con muchos dispositivos. El host maneja más complejidad, entonces los dispositivos electrónicos pueden ser relativamente

más simples y baratos. Dispositivos IEEE-1394 pueden comunicarse con otros directamente, y una simple comunicación puede ser direccionada a varios receptores. El resultado es una interfaz más flexible, pero los dispositivos electrónicos son más complejos y más caros.

2.11.8 USB versus Ethernet

Para algunas aplicaciones, la opción está entre USB y Ethernet. Ethernet tiene varias ventajas incluida la habilidad de usar cables muy largos, habilidad de hacer broadcasting, y soporte para protocolos de Internet en PCs y Ethernet con capacidad de desarrollo de sistemas. Así como IEEE-1394, el hardware requerido para soportar Ethernet es más complejo y caro que el hardware típico para el periférico USB. El USB es también más versátil con cuatro tipos de transferencias y una variedad de clases definidas para diferentes propósitos.

2.11.9 Componentes del bus

Los componentes físicos del USB consisten en circuitos, conectores, y cables entre el host y uno o más dispositivos.

El host es una PC u otra computadora que contiene un controlador host USB y un hub raíz. Estos componentes trabajan en conjunto para habilitar el sistema operativo para comunicarse con el dispositivo en el bus. El controlador del host le da formato a la información para ser transmitida en el bus y traslada la información recibida a un formato que el sistema operativo pueda entender. El controlador del host también realiza otras funciones relacionadas con la administración en la comunicación en el bus. La raíz del hub tiene uno o varios conectores para conectar más dispositivos. La raíz del hub, en combinación con el controlador del host, detecta dispositivos adjuntados y removidos, manda peticiones desde el controlador del host, pasa información entre dispositivos y el controlador del host.

2.11.10 Topología

La topología, o un arreglo de conexiones, del bus es una topología tipo estrella. En el centro de cada estrella se tiene un hub. La figura 2.9 muestra la topología estrella típica del USB. Cada punto en la estrella es un dispositivo que conecta a un puerto en un hub. El número de puntos en cada estrella puede variar, un hub típico puede tener dos, cuatro o siete puertos.

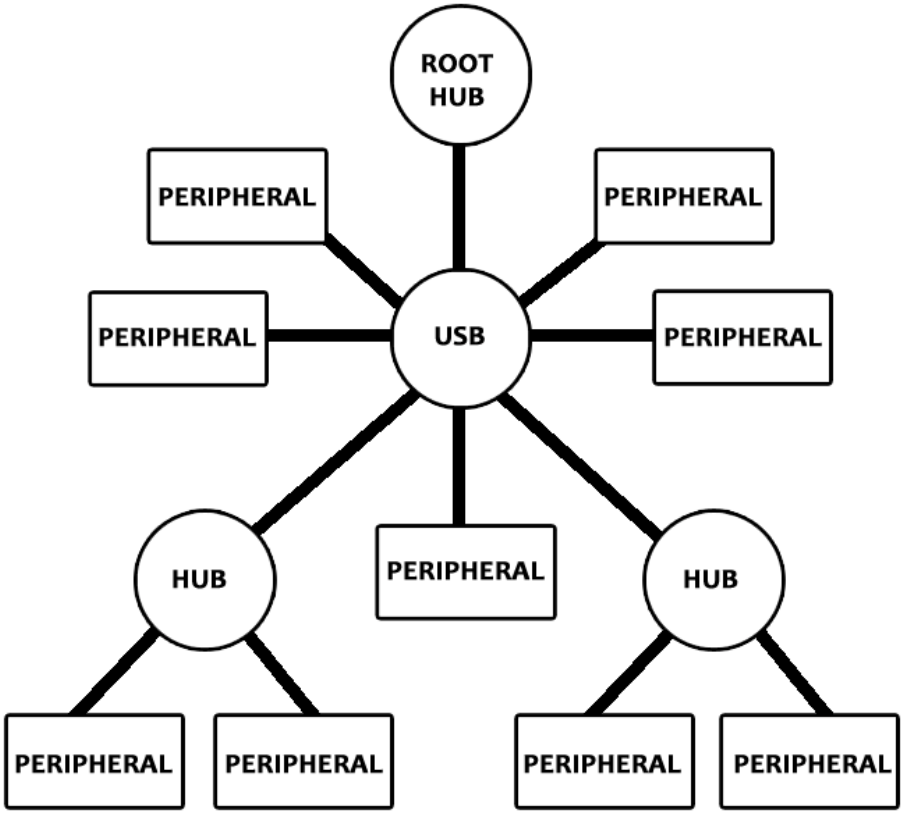


Figura 2.9. Topología estrella usada por el USB, donde cada hub es el centro de la estrella pudiendo conectar a periféricos o hubs adicionales.

2.11.11 Evolución de una interfaz

La razón principal por la que una interfaz no sale tan seguido es que las interfaces existentes tienen la irresistible atracción de que todos los periféricos que los usuarios usan no quieren usar chatarra, o interfaces que no sean óptimas. Usar una interfaz existente también ahorra tiempo de diseño a una nueva interfaz.

2.11.12 Un tipo de transferencia para cada propósito

Existen varios tipos de transferencias que soporta el USB que son: control, *bulk*, interrupción, y síncrono. Cada tipo de transferencia tiene habilidades y límites que hacen la transferencia adecuada para diferentes propósitos. La tabla 2.2 compara la cantidad de información que cada tipo de transferencia puede mover a cada una de las tres diferentes velocidades.

Tabla 2.2. Comparación de la máxima transferencia de datos posible variando el tipo de transferencia y la velocidad de bus.

Tipo de transferencia	Máximo transferencia de información por tipo de *endpoint* (KBps de carga útil por transferencia con datos = máximo tamaño de paquete permitido para la velocidad)		
	Low-speed	Full-speed	High-speed
Control	24	832	15,872
Interrupción	0.8	64	24,576
Bulk	No permitido	1216	53,248
síncrono		1023	24,576

2.11.12.1 Transferencia de control

Este tipo de transferencia tiene dos usos, una es llevar las peticiones que están definidas por las especificaciones del USB y usadas por el host para aprender acerca del dispositivo y las configuraciones del dispositivo. Cada dispositivo debe soportar transferencia de control sobre la tubería (pipe, que es una conexión virtual entre el dispositivo y el host) en el *endpoint* 0. El dispositivo puede soportar varios pipes de transferencias de control, pero en realidad no hay necesidad de tener más de una. Si el dispositivo no regresa un paquete esperado con información de control durante la transferencia de control, el host trata dos veces más. Al no recibir ninguna respuesta al tratar con el dispositivo tres veces, el host notifica al software esta petición y

detiene la comunicación con el *endpoint* del dispositivo hasta que el problema sea corregido.

2.11.12.2 Transferencia *bulk*

Las transferencias *bulk* son muy útiles para transferir información cuando el tiempo no es un factor crítico. Las transferencias tipo *bulk* pueden mandar grandes cantidades de información sin obstruir el bus por que las transferencias tipo *bulk* postergan su transferencia y esperan hasta que el bus esté disponible. El uso de este tipo de transferencia incluye mandar información desde el host hasta una impresora, mandar información desde un scanner al host, leer y escribir sobre un disco. En un bus de modo pasivo, las transferencias *bulk* son las más rápidas. Unas de las ventajas de usar este tipo de transferencias es que las transferencias *bulk* detectan errores. Si el dispositivo no regresa un paquete que le indique al host que la información ha sido recibida correctamente (*handshake*), el host intenta dos veces más. El controlador del host determina si el host se rinde en estar recibiendo paquetes que le indiquen si el bus está libre o si no hay errores en la información.

2.11.12.3 Transferencia de interrupción

Este tipo de transferencias son útiles cuando la información a transferir requiere una específica cantidad de tiempo. Aplicaciones típicas incluyen teclados, controles para juegos, y reportes de estado de hubs. Los usuarios no quieren un notable retraso entre presionar una tecla o mover el *Mouse* y ver el resultado en el monitor. Un hub necesita reportar si un dispositivo es conectado o desconectado de una manera rápida. Las tres tipos de velocidades soportan este tipo de transferencias. Detecta si hay errores, pero el host siempre está intentando sin límite en recibir paquetes que indique el estado del bus o del dispositivo (*handshake NAK*).

2.11.12.4 Transferencia síncrona

Transferencias de este tipo se usan cuando la información debe llegar en un ritmo constante, o por un tiempo específico, y donde errores ocasionales pueden ser tolerados. Ejemplos donde es usada este tipo de transferencia es cuando se transmite audio o video en tiempo real. Pero hay que ver que la información que eventualmente es enviada a un ritmo constante no siempre requiere un tipo de transferencia síncrona. Por ejemplo, un host puede usar la trasferencia *bulk* para mandar un archivo de música a un dispositivo. Después de recibir el archivo, el dispositivo puede empezar a reproducir el archivo de música a un ritmo apropiado. Pero tiene una desventaja, el precio que hay que pagar para garantizar la entrega a tiempo de grandes bloques es que la información no es corregida. Este tipo de transferencia es usada donde pequeños errores ocasionales son aceptables. Por ejemplo, los oyentes pueden tolerar o ni siquiera notar un corto en el audio de la música.

2.11.12.5 Resumen de tipos de transferencias

Bueno, hablamos sólo lo básico acerca de los tipos de transferencias existentes y de los posibles usos, es recomendable pensar un poco en las necesidades de nuestro proyecto, y tomar la decisión correcta acerca del tipo transferencia que necesita, también recomendar que lo que se menciona aquí es de los más básico acerca de los tipos de transferencias y si quieren saber un poco más de esto, hay libros especializados que profundizar más su estudio. Puede consultar la bibliografía y referencias del trabajo.

2.11.13 Enumeración

En este tema hablamos acerca de cómo el host aprende acerca del dispositivo. Antes de que las aplicaciones se puedan comunicar con el dispositivo USB, el host necesita aprender sobre el dispositivo y asignarle un controlador al mismo. El

proceso de enumeración incluye asignar una dirección al dispositivo, leer los descriptores desde el dispositivo, asignar y cargar el controlador del dispositivo y seleccionar una configuración que especifique los requerimientos de energía, los *endpoints*, y otras características. El dispositivo está listo para transferir información usando cualquier *endpoint* en su configuración.

2.11.13.1 El proceso

Una de las tareas de un hub es el detectar cuando un dispositivo es adjuntado o removido del bus. Es por eso que cada hub tiene un *endpoint* de interrupción el cuál su tarea es reportar este tipo de eventos. Cuando el sistema es reiniciado, el host verifica la raíz del hub para ver si algún dispositivo ha sido conectado, incluyendo hubs adicionales y dispositivos conectados a este hub. Después del reinicio del sistema, el host continua revisando periódicamente el bus a ver si un nuevo dispositivo ha sido conectado a éste.

En este proceso de aprender sobre el nuevo dispositivo, el host manda una serie de peticiones al dispositivo a través del hub, causando establecer una comunicación entre el host y el dispositivo. El host intenta enumerar el dispositivo enviando transferencias de control, ya antes mencionadas, estas transferencias contienen peticiones estándar USB y son mandadas al *endpoint* 0. Cuando la enumeración está completa, Windows pone el nuevo dispositivo USB en el administrador de dispositivos del panel de control. Y cuando un usuario remueve un dispositivo USB, Windows remueve éste del administrador de dispositivos.

2.11.13.2 Pasos del proceso de enumeración

Las especificaciones del USB tienen definidos seis tipos de estados para los dispositivos. Durante la enumeración, el dispositivo se mueve a través de estos cuatro estados: Energizado (*powered*), *Default*, Dirección (*Address*), y Configurado (*Configured*). Los otros dos estados son adjunto (*Attached*) y suspendido (*Suspend*).

En cada estado, el dispositivo tiene definido las capacidades y comportamiento del mismo.

Los pasos descritos a continuación son una secuencia típica de eventos que ocurren durante la enumeración bajo el sistema operativo de Windows. Pero el Firmware no debe de asumir que las peticiones de enumeración y los eventos ocurren en un orden particular. Para una función exitosa, un dispositivo debe detectar y responder a cada petición de control u otro evento del bus en cualquier tiempo. Los pasos son los siguientes:

1. **El usuario adjunta o conecta un dispositivo al puerto USB.** O el sistema operativo inicia con un dispositivo ya conectado. El puerto puede estar en la raíz del hub del host u otro hub conectado al host. El hub provee energía al puerto, y el dispositivo está en el estado de energizado.

2. **El hub detecta el dispositivo.** El hub monitorea los voltajes para cada línea de señal del puerto. El hub tiene resistencias de pull-up de 14.25 a 24-8kΩ en cada una de las dos líneas de señal de los puertos (D+ y D-). Un dispositivo tiene resistencias de pull-up de 900 a 1575Ω ya sea en D+ para dispositivos de *full-speed* o en D- para dispositivos *low-speed*. Los dispositivos *high-speed* son capaces de ser conectados a velocidad *full-speed*. Mientras detecta un dispositivo, el hub continua mandando energía pero todavía no empieza con el tráfico de las peticiones del USB.

3. **El host aprende acerca del dispositivo.** El host manda peticiones del estado al hub, del bus y otras que describen para saber si un nuevo dispositivo ha sido recientemente conectado al hub. La información regresada del hub al host le dice si ha sido conectado o no.

4. **El hub detecta de que tipo de velocidad es el dispositivo, low-speed o high-speed.** Sólo antes de que el hub reinicie el dispositivo, el hub determina si el

dispositivo es *low-speed* o *high-speed* examinando los voltajes en las dos líneas de la señal. El hub detecta la velocidad determinando cual de las líneas tiene un voltaje mayor en estado pasivo. El hub manda la información al host en respuesta a la siguiente petición del estado del puerto.

5. **El hub resetea el dispositivo.** Cuando un host aprende sobre un nuevo dispositivo, el controlador del host manda una petición de características del puerto que le pregunta al hub para resetear el puerto. El hub pone la línea de los dispositivos USB en la condición de RESET por al menos 10 milisegundos. El RESET es una condición especial donde D+ y D- están en bajo lógico. (Normalmente, las líneas tienen estados lógicos opuestos). El hub manda el RESET sólo a los nuevos dispositivos. Otros hubs y otros dispositivos no se dan cuenta del RESET.

6. **El host checa si un dispositivo full-speed soporta high-speed.** Se detecta si un dispositivo soporta high-speed usando los estados de dos señales especiales, midiendo el voltaje entre los estados solamente durante el RESET del dispositivo.

7. **El hub establece la ruta de la señal entre el dispositivo y el bus.** El host verifica que el dispositivo ha salido del estado del RESET mandando peticiones acerca del estado del puerto. Un BIT en la información regresada indica si el dispositivo sigue en el estado de RESET. Si es necesario el host repite esto hasta que el dispositivo ha salido del estado de RESET.

8. **El host manda la petición de los descriptores para aprender el máximo tamaño del paquete del pipe por default.** El host manda una petición a la dirección 0 del dispositivo, *endpoint* 0. Eso por que el host sólo enumera un único dispositivo a la vez, y sólo un dispositivo debe responder a la comunicación del direccionamiento en la dirección 0, incluso si muchos dispositivos han sido conectados al mismo tiempo.

9. **El host asigna una dirección.** El controlador del host asigna una única dirección al dispositivo mandando una petición que establece la dirección. El dispositivo completa el estado de status de la petición usando la dirección por *default* y luego implementa otra dirección. El dispositivo está ahora en el estado de dirección. Todas las comunicaciones en este punto de la enumeración usan esta última dirección.

10. **El host aprende acerca de las habilidades del dispositivo.** El host manda una petición para obtener el descriptor a la nueva dirección para leer el descriptor del dispositivo. En esta vez el host recupera todo el descriptor del dispositivo. El descriptor es una estructura de información conteniendo el tamaño máximo del paquete del *endpoint* 0, el número de configuraciones, y otra información básica que el dispositivo soporta. El host usa esta información para seguir con el paso que sigue. El host sigue aprendiendo del dispositivo pidiendo uno o más descriptores dentro del dispositivo. El host de Windows empieza por pedir sólo los nueve bytes de la configuración del descriptor.

11. **El host asigna y carga el controlador del dispositivo.** (excepto por dispositivos compuestos). Después de aprender de los descriptores, el host checa el mejor controlador para el dispositivo que pueda administrar la comunicación con el dispositivo. Windows trata que concuerde la información del archivo INF de la PC con el *Vendor ID*, *Product ID*, y (opcional) el número de la versión recuperada del dispositivo. Si no concuerdan, Windows busca un controlador que concuerde con alguna clase, subclase, y el protocolo de los valores obtenidos del dispositivo. Si el dispositivo ha sido enumerado anteriormente, Windows puede usar la información en el sistema en vez de buscar en el archivo INF. Después de que el sistema operativo asigne y cargue el controlador, el controlador puede pedir el dispositivo para reenviar los descriptores o mandar otro descriptor de una clase específica.

12. **El controlador del dispositivo en el host selecciona una configuración.** Después de aprender del dispositivo desde los descriptores, el controlador del dispositivo realiza una petición con número de configuración deseada. Algunos dispositivos soportan una sola configuración. Si el dispositivo usa varias configuraciones, el controlador puede decidir cual de las configuraciones pedir basada en la información que el controlador tiene acerca de cómo se debe usar el dispositivo, o el controlador puede preguntar al usuario que hacer o sólo elegir la primera configuración.

Los otros estados de los dispositivos son adjunto (*Attached*) y suspendido (*Suspended*).

Estado Attached. Si el hub no provee energía a la línea VBUS del dispositivo, el dispositivo está en el estado de adjunto (*Attached state*). La ausencia de energía puede ocurrir si el hub ha detectado una condición que el dispositivo ha sobrepasado el límite de corriente del bus o si se ha dado la petición del host al hub de remover la energía del puerto. Sin energía en VBUS, el host y el dispositivo no pueden comunicarse, por lo tanto, la situación sería la misma si el dispositivo no estuviera conectado.

Estado Suspended. El dispositivo entra en un estado suspendido después de no detectar actividad sobre el bus, por al menos 3 milisegundos. En el estado suspendido el dispositivo debe limitar el uso de energía del bus.

2.11.14 Descriptores

Los descriptores del USB son estructuras de datos o bloques de información con un formato definido, que habilitan al host para aprender acerca del dispositivo. Cada descriptor contiene información acerca del dispositivo como un todo o solamente de un elemento del dispositivo.

Todos los dispositivos USB deben responder a las peticiones de los descriptores estándares del USB. El dispositivo debe almacenar la información de los descriptores y responder a las peticiones que le hagan acerca de los descriptores. USB usa transferencias de control para pedir los descriptores al dispositivo, mientras la enumeración está en proceso, las peticiones de los descriptores son cada vez menores, o sea, cada vez se piden menos elementos del dispositivo como por ejemplo: primero se pide toda la información del dispositivo, luego cada configuración, luego la configuración de cada interfaz o interfaces, y por último cada interfaz del *endpoint* o *enpoints*. Los descriptores contienen solamente información básica del dispositivo.

Los descriptores forman una de las más importantes características del USB, ya que con los descriptores el sistema operativo puede saber el *Product ID* y el *Vendor ID* para poder asignarle un controlador de una manera más fácil, así como saber que corriente máxima suministrarle al dispositivo, entre otras cosas.

El *Vendor ID* es un identificador que viene en la estructura del descriptor del dispositivo USB, este *Vendor ID* es asignado por *USB-IF* (USB implementers forum, Inc.). USB-IF su misión es soportar la adopción de la tecnología USB. Este *Vendor ID* es obtenido por esta organización por medio de un pago ya que es oficial, como por ejemplo el *Vendor ID* de microchip para sus controladores que soportan el módulo USB es 04D8. El *Product ID* es especificado por el *Vendor*, y puede ser diferente dependiendo del *Vendor* y de la aplicación, éste no tiene ningún costo, solamente el *Vendor ID*, ya que está registrado por *USB-IF*.

2.11.15 Peticiones (Request)

Estas son peticiones que hace el sistema operativo al dispositivo USB, como por ejemplo, la petición que hace el sistema operativo al dispositivo de que le envíen los descriptores, ver el estado del puerto, entre otras. La razón por la cuál no las vamos

a ver en este proyecto, es que son varias y no necesitamos entrar tan a fondo a lo que es el protocolo USB.

2.11.16 Clases

Muchos dispositivos USB tienen mucho en común con otros dispositivos que hacen la misma función. Como por ejemplo un ratón acerca de los movimientos del ratón y de los clicks de éste. Todas las impresoras reciben información y a su vez mandan información de su estado.

Cuando un grupo de dispositivos o interfaces comparten muchos atributos o proveen peticiones similares de servicio, es cuando tiene sentido definir los atributos y servicios en una clase específica. Esta especificación puede servir de guía para desarrolladores que diseñan y programan los dispositivos en una clase y para los programadores que escriben los controladores de los dispositivos para el sistema del host y así comunicarse con el dispositivo.

Una clase se identifica para definir la funcionalidad del dispositivo y cargar un controlador basado en su funcionalidad, se dividen en grupos dependiendo de la función de la que se encarga el dispositivo. La clase está definida dentro de la estructura de los descriptores, en la tabla 2.3 se ven las clases que hay, especificando el código en hexadecimal, su utilización, etc.

Tabla 2.3. Tipos de clases.

Clase	Utilización	Descripción	Ejemplos
00h	Dispositivo	No identificada	(Esta clase de dispositivos no está especificada, los descriptores de interfaz son usados para determinar el controlador.)
01h	Interfaz	Audio	Bocinas, micrófonos, tarjetas de sonido
02h	Ambos	Comunicaciones y Control CDC	Adaptador de Ethernet, MODEM, adaptador de puerto serial
03h	Interfaz	Dispositivo de interfaz humana (HID)	Teclados, mouse, joystick
05h	Interfaz	Interfaz de dispositivo físico (PID)	Force feedback joystick
06h	Interfaz	Imagen	Cámaras digitales (La mayorías de las cámaras funcionan como almacenaje masivo).
07h	Interfaz	Impresoras	Impresoras láser, impresoras de tinta
08h	Interfaz	Almacenamiento masivo	Memorias USB, lector de tarjeta de memoria, reproductor digital de audio controladores externos.
09h	Dispositivo	USB hub	Full speed hub, hi-speed hub
0Ah	Interfaz	CDC-Datos	(Esta clase es usada junto con la clase 02h
0Bh	Interfaz	Smart Card	Lector de USB smart card
0Dh	Interfaz	Content Security	-
0Eh	Interfaz	Video	Webcam
0Fh	Interfaz	Cuidado de la salud personal	-
DCh	Ambos	Dispositivo de diagnostico	Dispositivo de prueba USB
E0h	Interfaz	Controlador inalámbrico	Adaptador Wi-Fi, adaptador Bluetooth
EFh	Ambos	Varios	Dispositivo ActiveSync
FEh	Interfaz	Aplicación especifica	Puente IrDA
FFh	Ambos	Vendedor especifico	(Este clase indica que el vendedor necesita especificar el controlador)

2.11.17 ¿Cómo se comunica el host?

Bajo el sistema operativo Windows, si una aplicación quiere tener acceso a un periférico USB tiene que comunicarse con el controlador del dispositivo que sabe cómo manejar la comunicación con el controlador USB del sistema.

2.11.17.1 Controlador del dispositivo básico

El controlador del dispositivo es un componente de software que habilita la aplicación para accesar al dispositivo hardware. El dispositivo hardware puede ser una impresora, un MODEM, un teclado, una pantalla de video, una unidad de adquisición de información, o cualquier circuito que el CPU puede accesar.

2.11.17.2 Aislando aplicaciones de los detalles

Las aplicaciones son programas que el usuario ejecuta, incluyen, desde reproductores de audio, procesadores de palabras y bases de datos para aplicaciones de propósito especial que acceden al hardware. El controlador del dispositivo aísla a la aplicación de tener que saber todos los detalles acerca de conexiones físicas, señales, y protocolos requeridos para comunicarse con el dispositivo.

Un controlador del dispositivo puede habilitar el código de la aplicación para accesar al periférico cuando la aplicación conoce el nombre del periférico o la función del dispositivo. La aplicación no tiene que conocer la dirección física del puerto donde está conectado el dispositivo o monitorear las señales del mismo. La aplicación ni siquiera tiene que saber si el dispositivo está conectado por medio del USB o cualquier otra interfaz. El controlador del dispositivo crea la interconexión entre la aplicación y el código del hardware. Las aplicaciones se comunican con el controlador usando funciones soportadas por el sistema operativo.

2.11.17.3 Opciones para dispositivos USB

Hay diferentes enfoques para obtener un controlador para un dispositivo USB que se esté desarrollando. Muchos dispositivos pueden usar controladores que estén incluidos en Windows, o sean proveídos por el vendedor del chip o por otra fuente. Otros dispositivos pueden tener controladores a la medida. Cuando un controlador a la medida es necesario, hay herramientas disponibles que simplifican y aceleran la tarea de escribir un controlador. Algunas funcionan de más de una forma, y la opción depende de que sea más fácil, barato, u ofrezca mejores funcionalidades al usuario.

2.11.18 Archivo INF

Un archivo de información para la configuración del dispositivo o archivo INF, es un archivo de texto que contiene información acerca de uno o más dispositivos en una clase de configuración del dispositivo. El archivo le dice a Windows que controlador o controladores usar y contiene información para guardar en el registro de Windows. Windows incluye archivos INF para controladores proveídos por el mismo sistema operativo.

2.11.19 Gestión de la energía

Una característica conveniente del protocolo USB es la habilidad para los dispositivos de alimentarse del bus USB. Muchos dispositivos pueden ser totalmente energizados por el bus, pero tomar energía del bus trae consigo la responsabilidad de respetar el límite de la energía disponible, incluyendo el estado de suspensión (*Suspend State*) cuando es requerido.

Dentro de una computadora típica se tiene una fuente de alimentación con corrientes que la computadora no necesita, los cuáles son para proporcionárselos al puerto USB. Muchos hubs tienen sus propias fuentes de energía también. Muchos

periféricos USB pueden tomar ventaja de estas fuentes de alimentación en vez de proveer su propia fuente de energía.

La habilidad de tomar energía desde el mismo cable que lleva la información para o desde la computadora tiene muchas ventajas. Los usuarios no necesitan electricidad afuera cerca del periférico, y el periférico puede ser pequeño y liviano. Un periférico sin alimentación cuesta menos fabricarlo. Esta es una de las ventajas que tiene USB contra otros protocolos de comunicación como lo son los puertos seriales y los puertos paralelos.

2.11.19.1 Voltajes

El voltaje nominal entre el VBUS y la tierra en un cable USB es de 5V, pero el actual valor puede ser un poco más o un poco menos de éste. El dispositivo que usa la energía del bus debe ser capaz de manejar estas variaciones. Los voltajes mínimos y máximos permitidos por el bus USB o hubs se muestran en la tabla 2.4.

Tabla 2.4. Voltajes mínimos y máximos permitidos.

Tipo de hub	Voltaje mínimo	Voltaje máximo
Energía alta	4.75V	5.25V
Energía baja	4.4V	5.25V

2.11.19.2 ¿Qué periféricos pueden usar energía del bus?

No todos los periféricos pueden tomar ventaja del bus. Los avances en la tecnología han reducido el requerimiento de la energía de muchos dispositivos electrónicos. Un dispositivo que requiera arriba de 100 miliamperes puede ser energizado desde cualquier host o hub USB. Un dispositivo que requiera más de 500 miliamperes debe de utilizar una fuente de energía propia, como una fuente de alimentación, que le pueda proporcionar la energía que el dispositivo requiera. **UN DISPOSITIVO NO**

DEBE DE TOMAR MÁS DE 100 MILIAMPERES HASTA QUE EL HOST HA CONFIGURADO EL DISPOSITIVO. Y todos los dispositivos deben limitar su consumo cuando están en estado suspendido.

2.11.20 Resumen del USB

Hay mucho que mencionar del protocolo del USB, en este proyecto sólo se mencionaron temas básicos como: los tipos de conexión USB que existen, la velocidad de cada uno de ellos, los tipos de transferencias, los controladores, las clases, las ventajas y desventajas contra otros protocolos, la energía que puede proporcionar, entre otras cosas. Pero todo lo que se mencionó aquí fue muy técnico, quedaron muchas cosas pendientes como: registros que usa, qué depende de cada desarrollador del dispositivo, las señales, los tiempos de la señal, etc. Pero no se habló de esto porque el tema del USB es muy extenso y nos tomaría mucho tiempo, trabajo y páginas para hablar de ello, por eso es que existen libros especializados para esto y documentos oficiales publicados por los desarrolladores de éste protocolo, y aparte el objetivo del proyecto no es hablar sobre el protocolo USB sino sobre su aplicación al estudio de filtros, es por eso que no se toca este tema muy a fondo, por eso es que recomiendo que dependiendo de lo que quieran realizar con este protocolo lean sobre el mismo, hay ciertos aspectos que los desarrolladores deben de tomar en cuenta, pero no es necesario que se aprenda a la perfección el protocolo, aunque nunca está demás aprender de él, si quieren saber más información de este protocolo llamado USB, les recomiendo los libros y las páginas oficiales sobre este tema los cuáles están en la bibliografía de este proyecto [1].

2.12 Microcontroladores PIC

La familia de microcontroladores PIC son manufacturados por Microchip Technology Inc. Actualmente éstos son uno de los más populares microcontroladores en el mercado, usados en muchas aplicaciones en la industria y en el comercio. Microchip vende cerca de 120 millones de dispositivos al año.

La arquitectura de los microcontroladores PIC está basada en una arquitectura Harvard RISC modificada (Computadora con un conjunto de instrucciones reducidas por sus siglas en inglés, Reduced Instruction Set Computer) con una arquitectura de doble bus, que provee un rápido y flexible diseño con una ruta de emigración fácil de 6 pines a 80 pines, y de 384 bytes a 128 Kbytes de memoria de programa.

Los microcontroladores PIC están disponibles con muchas diferentes especificaciones dependiendo de lo que se desee desarrollar, tales como:

- Tipo de memoria.
  - Memoria Flash.
  - OTP (Programable una vez por sus siglas en inglés, One Time Programmable).
  - Memoria ROM.
  - Sin memoria ROM.
- Cantidad de pines de entrada/salida.
  - 4-18 pines.
  - 20-28 pines.
  - 32-44 pines.
  - 45 y más pines.
- Tamaño de la memoria.
  - 0.5-1 K.
  - 2-4 K.
  - 8-16 K.
  - 24-32 K.
  - 48-64 K.
  - 96-128 K.
- Características especiales.
  - Módulo CAN.
  - Módulo USB.
  - LCD.

- Control de motores.
- Para frecuencias de radio.

Al igual que el tema relacionado con el USB, el tema de los microcontroladores es también muy extenso, cada compañía desarrolladora de microcontroladores crea sus microcontroladores con diferentes características, registros, etc., nosotros nos enfocamos a los microcontroladores creados por Microchip Technology Inc. debido a que son los microcontroladores más usados en la industrias, y son los microcontroladores que nos enseñan a usar en nuestra formación académica.

2.12.1 Microcontrolador PIC 18F2553

Nuestro proyecto se enfoca a los microcontroladores de la gama alta, el microcontrolador 18F2553 de la familia de los 18Fxxxx debido a que cuenta con varios recursos para la realización de este proyecto, uno de los más importantes a resaltar, o el más importante, es que cuenta con un módulo USB que nos proporciona la comunicación con la computadora por medio de éste, también cuenta con recursos aptos para este proyecto, como: temporizadores, módulo SPI, y un módulo ADC de 12bits. La figura 2.10 muestra la arquitectura interna del microcontrolador con los diferentes módulos con los que cuenta, así como sus características más sobresalientes en la siguiente lista:

- Velocidad máxima de operación de hasta 48 MHz (para el módulo USB).
- Memoria de programa de 32 Kbytes.
- Memoria de datos de 2048 bytes.
- Un módulo conversión analógico-digital de 10 canales con una resolución de 12 bits.
- 2 módulos de captura/comparar/PWM.
- 4 temporizadores.
- 19 fuentes de interrupción.
- Módulo de comunicación MSSP y USART.

- Módulo USB (Bus Universal en Serie, por sus siglas en inglés, Universal Serial Bus).
- Entre otros.

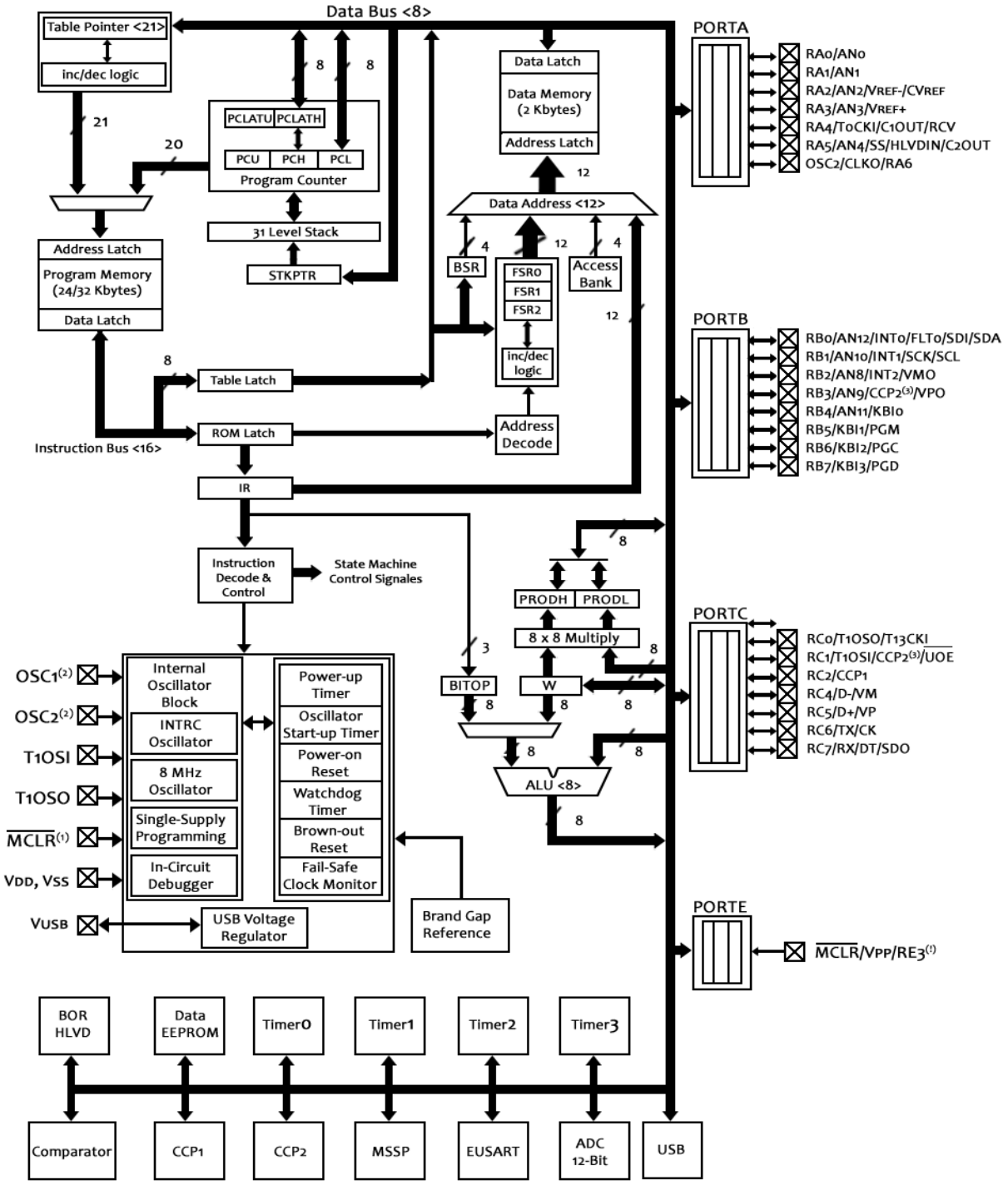


Figura 2.10. Arquitectura interna del microcontrolador PIC 18F25553.

2.13 Programación de alto nivel para microcontroladores PIC

Para que nuestro microcontrolador PIC haga lo que queremos hay dos maneras de lograrlo, la primera es programándolo con un lenguaje de bajo nivel, el ensamblador, éste proporciona poca o ninguna abstracción al PIC, y consecuentemente es fácil trasladarlo a un lenguaje máquina, el nombre de programación de bajo nivel no significa o no implica que sea inferior a la programación de alto nivel, se refiere a la reducida abstracción entre el lenguaje y el hardware. La programación de bajo nivel tiene sus ventajas como: mayor adaptación al PIC, posibilidad de tener mayor velocidad con un mínimo uso de memoria. Y desventajas como: mayor dificultad en la programación y comprensión del programa, el programador debe conocer más de un centenar de instrucciones (en este PIC son 75 instrucciones), es necesario conocer a detalle la estructura del PIC. También la programación a bajo nivel tiene la característica que se trabaja a nivel de instrucciones, es decir, su programación es al más fino detalle.

La segunda manera de programar al PIC es con programación de alto nivel. Ésta se caracteriza por expresar algoritmos de una manera adecuada a la capacidad cognitiva del humano, en lugar de la capacidad ejecutora de la máquina o del PIC. Otra limitación de este lenguaje es que requiere de ciertos conocimientos de programación para realizar secuencias de instrucciones lógicas. Este tipo de lenguaje alto se creó para que el usuario común pudiese solucionar el problema del procesamiento de datos de una manera más fácil y rápida.

Algunas desventajas de este tipo de lenguaje son: que reduce la velocidad al ceder el trabajo de bajo nivel a la máquina o al PIC. Pero las ventajas de éste son: que genera un código más sencillo y comprensible, y el código es válido para diversas máquinas o PIC, es por eso que se eligió un código de alto nivel para la programación del PIC de este proyecto, debido a que es más fácil y más comprensible y que contamos con librerías para hacer el trabajo más fácil aún, como el del módulo USB por ejemplo, y también la de comunicación SPI™.

2.14 Microchip MPLAB IDE y C18

MPLAB es un entorno de desarrollo integrado (IDE, por sus siglas en inglés, Integrated Development Environment) gratuito, que es un conjunto de herramientas integradas para el desarrollo de aplicaciones integradas que emplean los microcontroladores PIC y dsPIC. MPLAB IDE corre en aplicaciones de 32 bits como Windows, es fácil su uso e incluye un software gratis para aplicaciones rápidas de desarrollo y un debugging súper cargado. MPLAB IDE también sirve como única y unificada interfaz gráfica para el usuario para el uso adicional de elementos de hardware y software de Microchip u otras compañías. Pero C18 es un compilador de la misma compañía Microchip Technology Inc. que se complementa con el MPLAB IDE. El compilador C18 debido a que fue desarrollado por Microchip, la misma compañía que crea los microcontroladores PIC, genera un código máquina altamente optimizado para la serie PIC18, con alto desempeño para la los microcontroladores de la serie PIC24. Este compilador C18 es el usado en este proyecto debido a estas características y es el que genera el código máquina a partir de código C ANSI.

2.15 Programación orientada a objetos (POO)

Es un paradigma de la programación que usa objetos y la interacción de ellos para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento. Actualmente muchos lenguajes de programación soportan la orientación a objetos.

2.15.1 Introducción

Los Objetos son entidades que combinan estado, comportamiento e identidad.

- El *estado* está compuesto de datos, que serán uno o varios atributos del objeto a los que se habrán asignado unos valores concretos (datos).

- El *comportamiento* está definido por los procedimientos o *métodos* con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.
- La *identidad* es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan ni deben separarse el estado y el comportamiento.

2.15.2 Conceptos fundamentales

La programación orientada a objetos es una nueva forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- **Clase**: definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.
- **Herencia:** (por ejemplo, herencia de la clase D a la clase C) Es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y

operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de OOP.

- **Objeto**: entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos). Se corresponde con los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa). Es una instancia a una clase.
- **Método**: Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
- **Evento**: un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.
- **Mensaje**: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- **Propiedad o atributo**: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
- **Estado interno**: es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para

indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

- **Componentes de un objeto**: atributos, identidad, relaciones y métodos.
- **Representación de un objeto**: un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

2.15.3 Características de la POO

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje le definen como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes:

- **Abstracción**: Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar *cómo* se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.
- **Encapsulamiento**: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
- **Principio de ocultación**: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una *interfaz* a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas,

eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- **Polimorfismo**: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama *asignación tardía* o *asignación dinámica*. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.
- **Herencia**: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en *clases* y estas en *árboles* o *enrejados* que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay *herencia múltiple*.

2.15.4 Resumen

El tema de la POO es muy extenso, en este capítulo sólo se ven los significados técnicos de éste, así como sus características y fundamentos explicados de una manera muy simple y básica. Básicamente es un paradigma que utiliza objetos como elementos fundamentales en la construcción de la solución. Surge en los años 70. Un objeto es una abstracción de algún hecho o cosa del mundo real que tiene

atributos que representan sus características o propiedades y métodos que representan su comportamiento o acciones que realizan. Todas las propiedades y métodos comunes a los objetos se encapsulan o se agrupan en clases.

2.16 Microsoft Visual Studio 2005 y Visual C++

Microsoft Visual Studio IDE es un entorno de desarrollo integrado para sistemas Windows. Soporta lenguajes tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Este IDE permite a los desarrolladores crear aplicaciones, sitios y aplicaciones Web. Nosotros optamos por el Visual C++ debido a que es uno de los tipos de lenguajes más usados en la actualidad, y desde mi punto de vista es un lenguaje más estructurado. Usando el entorno de desarrollo integrado de Microsoft Visual Studio, que nos proporciona un IDE de Visual C++ para lenguajes de programación en C, C++ y C++/CLI, podemos crear el programa principal para nuestro proyecto, Bode Plotter V3.0. Con este IDE podemos programar nuestro programa, valga la redundancia, usando WinForms con C++ que nos proporciona una manera más fácil y rápida de programar, debido a que su programación es más rápida ya que es mucho más gráfica que MFC u otras.

2.17 Síntesis digital directa (DDS)

2.17.1 Introducción

La síntesis digital directa (DDS, por sus siglas en inglés, Direct Digital Synthesis) es un método o técnica electrónica por la que se genera una señal sinusoidal con frecuencia, fase y amplitud muy precisas rápidamente manipuladas digitalmente. La amplitud de la señal sinusoidal es de un valor constante, debido a la naturaleza del DDS que emplea un DAC, así que se necesita la utilización de circuitos externos para variar esta amplitud. Otro atributo que tiene el método DDS es que incluye la habilidad de cambiar la resolución de la frecuencia y de la fase de manera

extremadamente fina, y saltar de frecuencia en frecuencia rápidamente. Esta combinación de características ha hecho muy popular a esta tecnología en el área militar y en los sistemas de comunicaciones. Un hecho es que la tecnología DDS fue aplicada primeramente y exclusivamente para aplicaciones militares y en la alta gama debido a su elevado costo, y el hecho que era tecnología muy avanzada para el uso en asuntos bélicos, difícil de implementar, y requería un DAC discreto de muy alta velocidad. Gracias a la mejora de la tecnología de circuitos integrados, ahora presenta una alternativa viable a la tecnología PLL para la generación rápida de salida analógica en aplicaciones de sintetización en el área de electrónica para el consumidor.

Cabe mencionar que es fácil incluir capacidades diferentes de modulación en DDS utilizando métodos de procesamiento de señales, debido a que la señal es tratada en forma digital. La programación del DDS, anchos de banda de canal de adaptación, formatos de modulación, saltos de frecuencia y las velocidades de los datos son fácilmente logrados. Los circuitos digitales usados para implementar funciones de procesamiento de señal no sufren efectos térmicos, envejecimiento y variaciones de los componentes asociados con sus equivalentes análogos.

Hay mucho de que hablar del DDS como el hecho de que gracias al avance en la manufactura de circuitos integrados reduce el tamaño del DDS y la potencia que consume éste, además del hecho que son usados en métodos de modulación y demodulación debido a su exactitud en la generación de señales. Pero en este capítulo, así como en el proyecto, sólo se va a ver el método DDS de manera básica.

2.17.2 Funcionamiento básico

En este apartado se explicará como funciona el método DDS, de una manera básica, considerando que hay formas más avanzadas de generar señales a través de este método, esas maneras más avanzadas tienen diferentes ventajas, la exactitud de la

frecuencia generada es más exacta, así como, el ruido, y los armónicos de menor potencia.

En un periodo de una señal sinusoidal, la fase es directamente proporcional al tiempo, como se muestra en la figura 2.11.

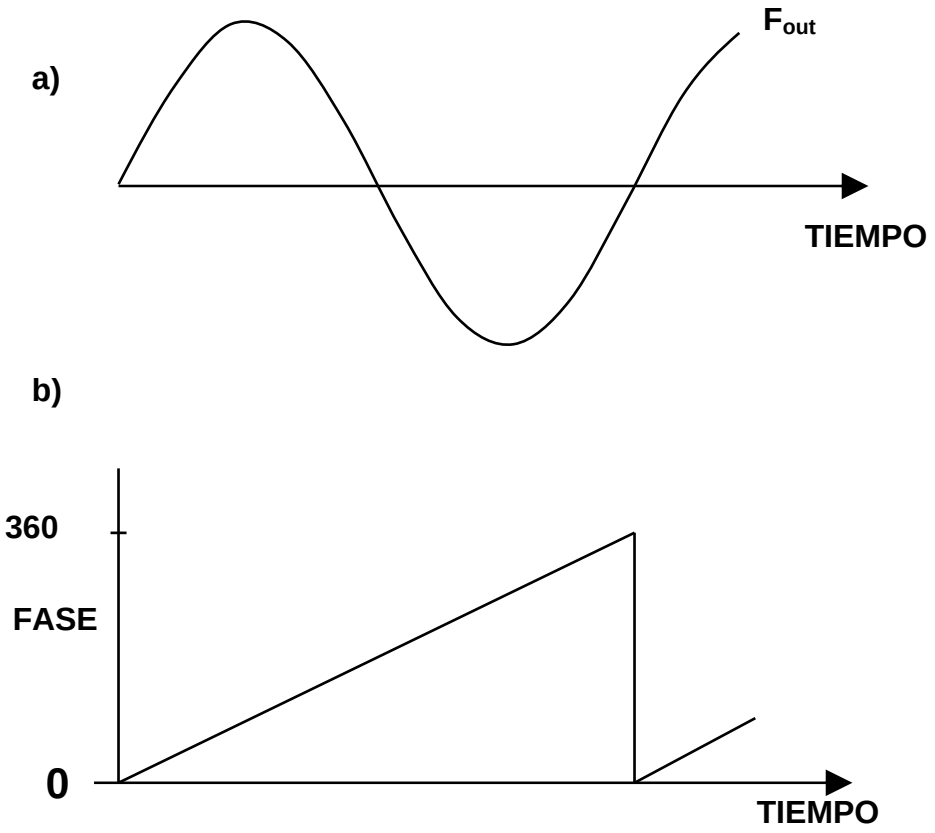


Figura 2.11. a) Señal sinusoidal. b) Fase proporcional.

Lo mismo ocurre en un sistema discreto, le toma 2^n periodos de reloj para dar una salida de un periodo. En la figura 2.12, n=5, y es por eso que le toma 32 ciclos de reloj para obtener un ciclo completo en forma discreta.

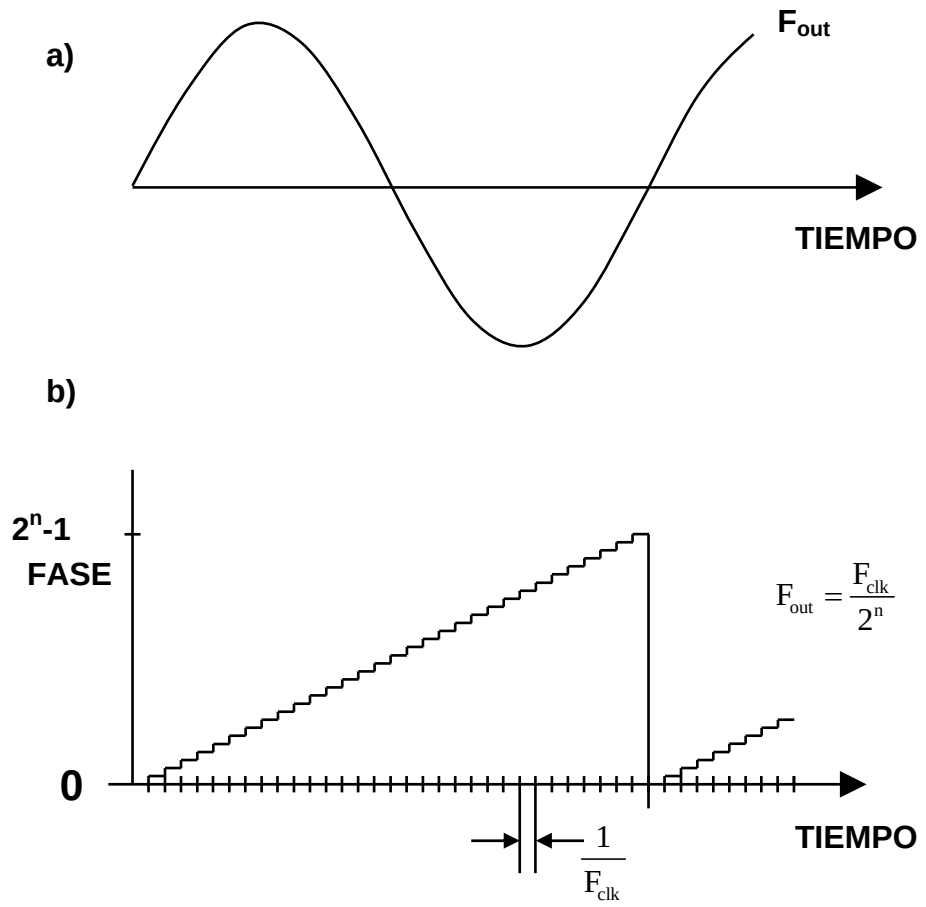


Figura 2.12. a) Señal sinusoidal. b) Fase de la señal sinusoidal en tiempo discreto.

Esto se genera por medio de un sencillo acumulador. Después de contar 2^n (32 en este caso), el contador se desborda y empieza desde cero otra vez. Este paso está representado en la figura 2.13.

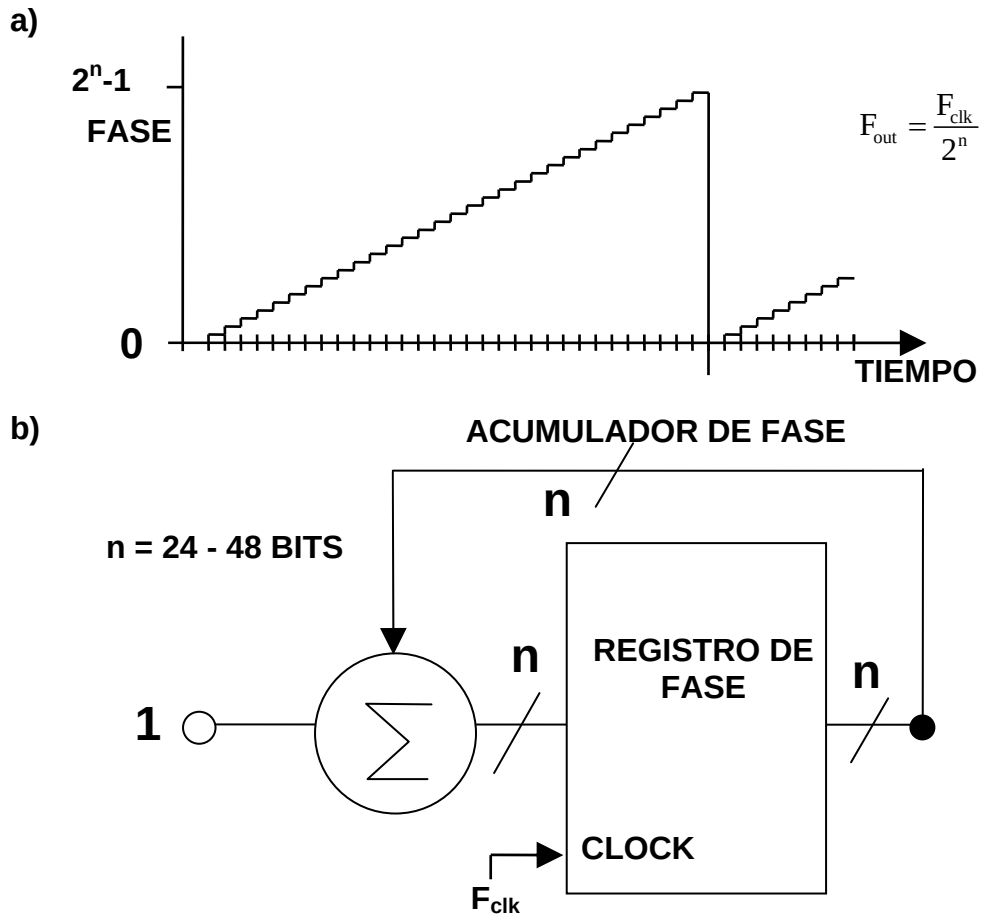


Figura 2.13. a) Salida de la fase de la señal sinusoidal en forma discreta del acumulador. b) Acumulador de fase.

Se puede obtener una mayor frecuencia usando un incremento de fase más largo (4 en este caso). La ecuación se ve poco creíble como la M^{th} de la FFT, la cuál es otra forma de ver esto. Para una "n" muy grande, el tamaño de la FFT es muy pequeño, permitiendo así una resolución muy fina de la frecuencia. Hay que tomar en cuenta que cuando se cambia "M", la frecuencia cambia, no la fase de ésta. En la figura 2.14 se muestra el ejemplo para cambiar la frecuencia.

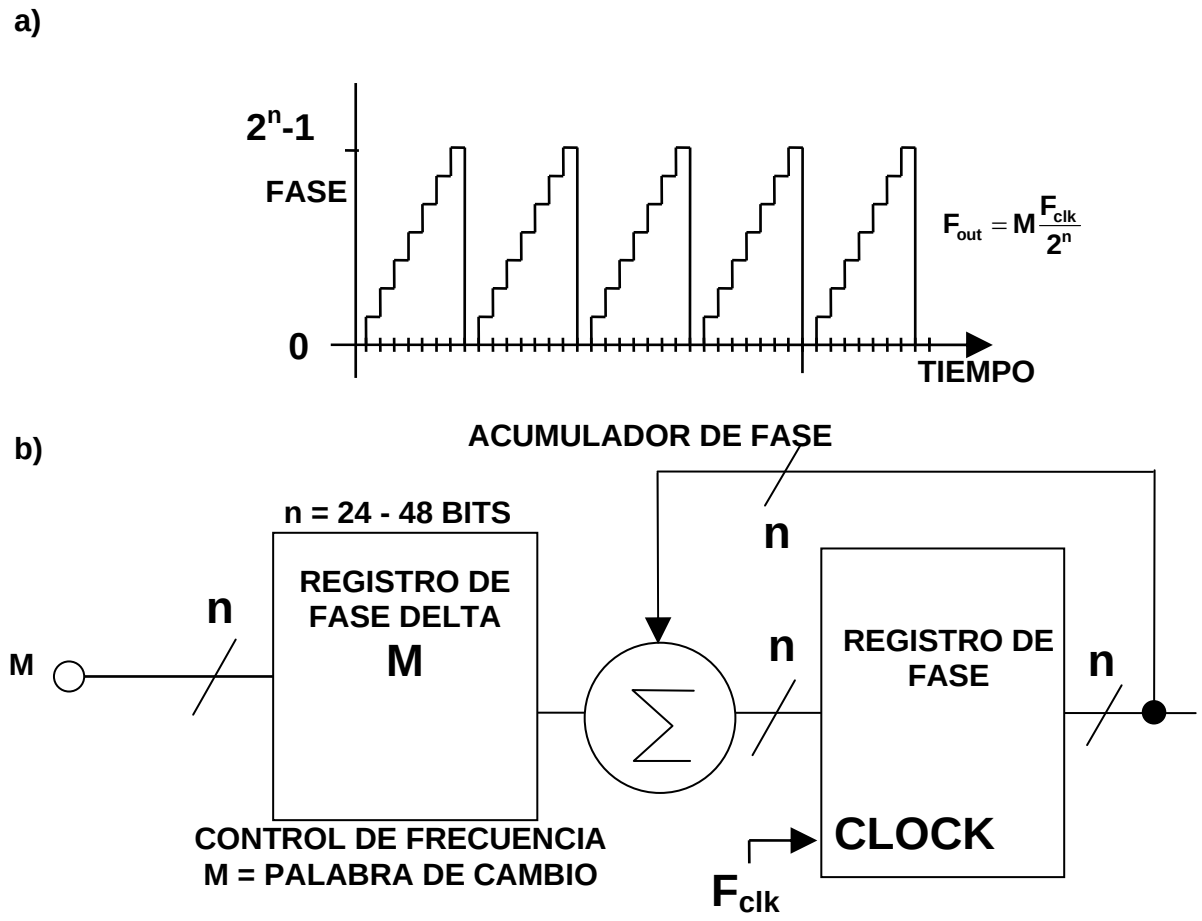


Figura 2.14. a) Salida del acumulador a una frecuencia mayor. b) Diagrama de bloques para una salida de una frecuencia mayor.

Pero lo que nosotros deseamos tener a la salida es una señal en una amplitud, no en fase. El convertidor de fase-a-amplitud se refiere a menudo a la búsqueda en una tabla sinusoidal, pero requiere poca capacidad de procesamiento para ejecutar un algoritmo comparado a una memoria ROM. Aun así, toma demasiada capacidad de procesamiento el ejecutar esto usando “n” bits (donde 24<n<48). Por lo tanto la fase es truncada a “p” bits primero. Para convertir la amplitud de los números a una amplitud de voltaje, se coloca un DAC (Convertidor digital-analógico por sus siglas en inglés, Digital-Analog Converter) en la salida. La figura 2.15 muestra esta etapa del DDS.

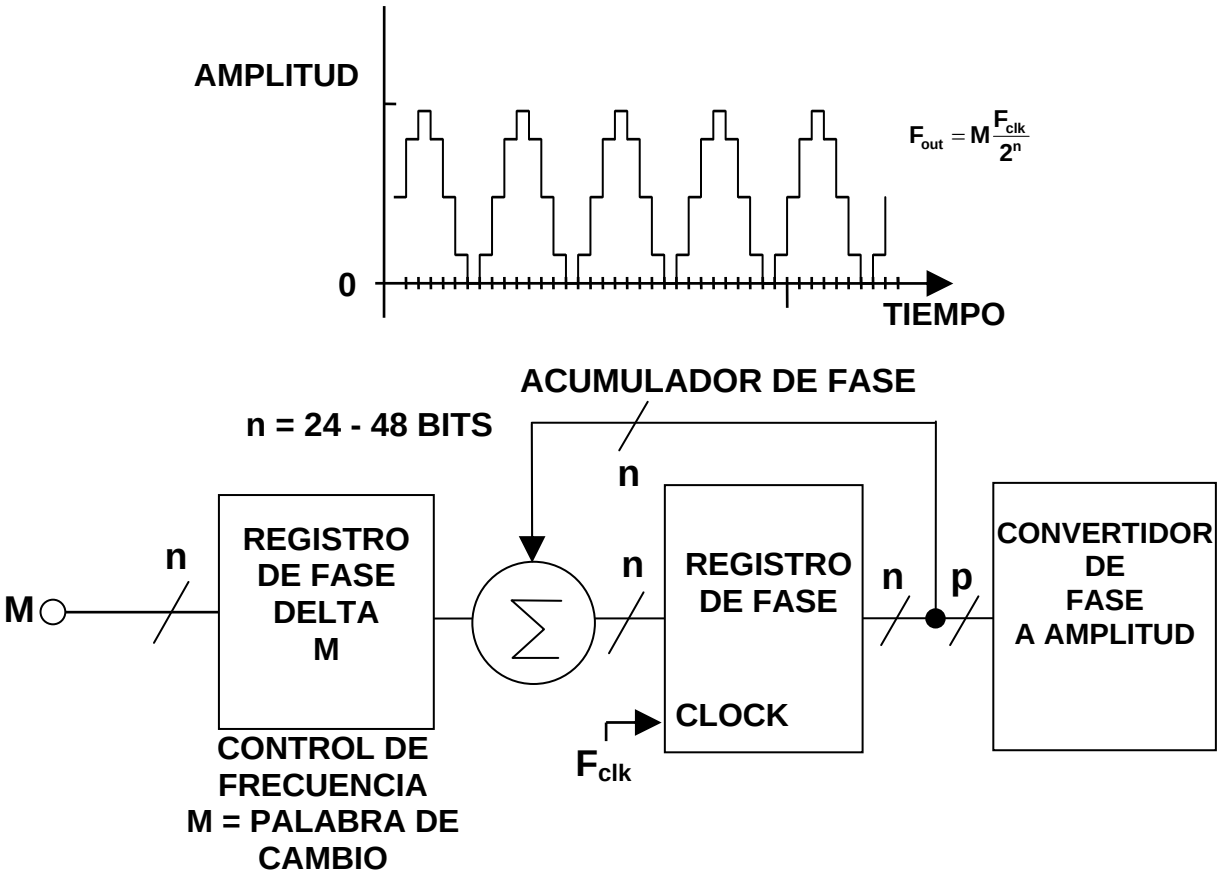


Figura 2.15. a) Fase de la señal sinusoidal convertida en amplitud. b) Diagrama a bloques del DDS para convertir la fase a amplitud.

A continuación se da un pequeño resumen del flujo de la señal a través del la arquitectura del DDS. Pero antes de empezar, mostraremos ya el diagrama total de lo que se ha descrito en este momento, en forma de bloques, hay que recordar que ésta es la arquitectura básica que se muestra en la figura 2.16.

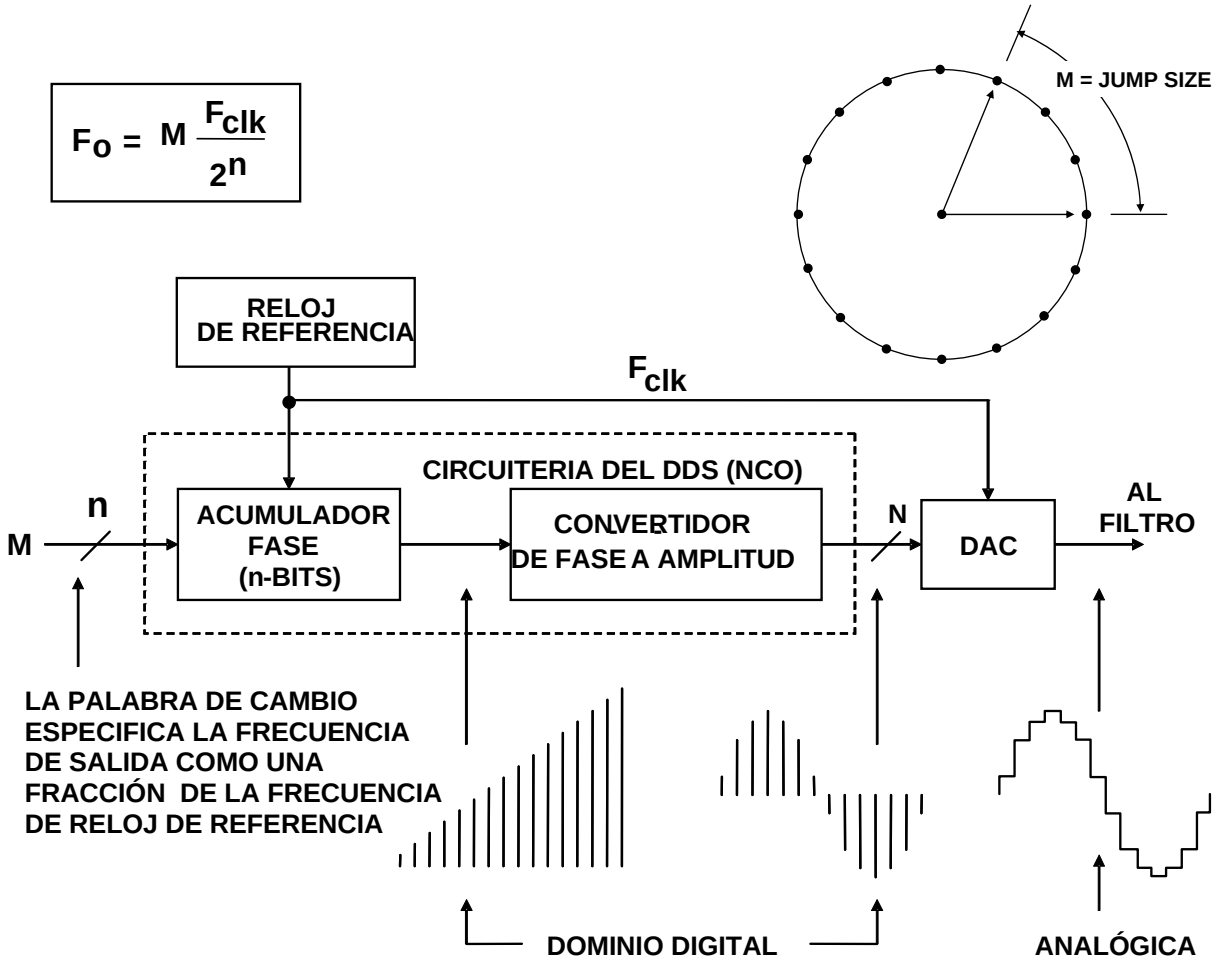


Figura 2.16. Arquitectura básica del DDS.

- El acumulador de fase es actualmente un módulo contador “M” que incrementa su número almacenándolo cada vez que recibe un pulso de reloj. La magnitud del incremento está determinado por el número binario de entrada o la palabra “M” contenida en el registro de fase delta (delta phase register) que se resume con el desbordamiento del contador.
- La información digital de la fase del acumulador de fase es convertida a su correspondiente amplitud digital por el convertidor fase-a-amplitud.
- Finalmente, el DAC convierte la amplitud digital en su correspondiente señal analógica.

Algunos de los DDS del mercado contienen todas estas características en un sólo circuito integrado así como un filtro pasa bajas de reconstrucción.

2.17.3 Ventajas y desventajas del DDS

Algunas de las ventajas que poseen los DDS son:

- La frecuencia de la señal sinusoidal se sintoniza digitalmente, típicamente con resoluciones de menos de un 1Hz.
- La fase de la onda sinusoidal es digitalmente ajustable, con sólo un ligero aumento en la complejidad del circuito.
- Debido a que el DDS es digital, la frecuencia y la fase son determinados numéricamente, NO HAY ERRORES de la señal por el cambio de temperatura o por el envejecimiento de los componentes.
- El salto de frecuencia a frecuencia, o de fase es extremadamente rápido.
- La interfaz del control digital de la arquitectura del DDS facilita el medio donde este sistema puede ser controlado remotamente y optimizado con una alta resolución bajo el control del procesador.
- Control preciso de la frecuencia de salida sin afectar la fase.
- Rápidos cambios arbitrarios tanto como en frecuencia, como en fase.
- Modulación precisa.

Algunas desventajas que poseen los DDS son:

- La frecuencia de salida debe de ser menor a o igual a ½ de la fuente del reloj de la frecuencia.
- La amplitud de la señal sinusoidal es fija, esto debido a la naturaleza del DAC interno del DDS. Esta amplitud puede ser modificada con circuitos externos adicionales.
- Debido a que la señal sinusoidal es generada digitalmente por técnicas de muestreo, el usuario debe aceptar una pequeña cantidad de distorsión, esto

es que la señal no es 100% pura. A no ser que se cuente con un filtro pasa bajas de reconstrucción.

2.17.4 Resumen de la técnica DDS

Hay mucho de que hablar sobre este tema, pero en este proyecto no nos basamos en una investigación muy profunda acerca del mismo, es por eso que hay temas del DDS que no se hablan aquí como lo son: el truncamiento, el *spur* y los errores que tiene. Pero lo que podemos hacer es resumir en pocas palabras lo que es el DDS: el DDS es una técnica que es usada para la generación de señales de varios tipos, no solamente sinusoidales, también triangulares y rectangulares, todas estas señales pueden ser generadas con una increíble precisión. Comparada con otras técnicas para generar señales, el DDS tiene muchas ventajas. Aunque en este capítulo sólo se vio la arquitectura más sencilla del DDS, existen varias, aún más complejas que ésta, la cuál usamos simplemente porque es la más sencilla para explicar el funcionamiento básico del DDS y que tengan un conocimiento básico acerca de esta técnica. Para conocer más sobre el funcionamiento pueden consultar la bibliografía y referencias del trabajo [7, 8, 9, 10] o pueden buscar información en el buscador de una de las compañías que manufacturan este tipo de tecnología como es *Analog Devices.*

CAPÍTULO III

Desarrollo

En este capítulo se describe el proceso que se realizo para desarrollar el proyecto trazador de gráficas de bode con interfaz con la computadora, los componentes que se utilizan y cuál es su función en el proyecto, acerca de las librerías que se utilizan para la comunicación del puerto USB (las usadas para escribir en el módulo SPI no se van a mencionar debido a que la gran parte de las librerías son muy fáciles de entender ya que son funciones muy sencillas, y la mayoría de las librerías usadas para el USB contienen el mismo tipo de información y no tendría caso repetirlas), ya si se desea conocer más sobre éstas, pues es necesario también conocer por lo menos un poco del protocolo que se usa para la comunicación por medio de este

puerto. Se explicará de una forma sencilla y general tratando de cubrir con todas las perspectivas, también se establecerán las diferencias de este proyecto con los anteriores [11, 12] y las mejoras, algunas de las cuales ya se han mencionado, así como las ventajas y desventajas al entorno de este proyecto. Todo lo que se incluye en este capítulo del desarrollo del proyecto es con la finalidad de hacer esta versión del proyecto más rápida, más exacta, el hardware utilizado más compacto y moderno a través del uso de técnicas más avanzadas y que el proyecto sea más amplio en el estudio de los filtros electrónicos.

3.1 Bode Plotter

En este capítulo vamos a hablar de cómo funcionan los diferentes elementos de este proyecto, como lo es el algoritmo del microcontrolador y demás elementos del mismo.

Este proyecto trata de analizar filtros electrónicos lineales para ver su ganancia en magnitud y fase por medio de las gráficas de Bode, para esto tenemos que generar una señal sinusoidal que será introducida al filtro electrónico y a su salida de este será medida por nuestro dispositivo y posteriormente mandar la información de las mediciones por medio del puerto USB a la computadora y por medio de un software mostrar los resultados, todo esto será descrito en este capítulo, de manera detallada y entendible.

3.2 Diagrama a bloques del Bode Plotter

En la figura 3.1 se puede observar el diagrama de bloques cómo está compuesto el hardware del Bode Plotter. De manera que se pueda apreciar más fácilmente y se mejore la comprensión del mismo con más detalle.

Así, mostrando la figura 3.1 se pueden describir las diferentes etapas del hardware del proyecto, para conocer la función de cada una de ellas. Hay que recordar que

este proyecto ya se ha realizado con anterioridad pero con otras técnicas de generación de la señal de entrada al filtro, técnicas menos eficientes que las utilizadas en esta versión, con otra manera de medir la fase de los filtros electrónicos y con otras diferencias que se mencionarán más adelante.

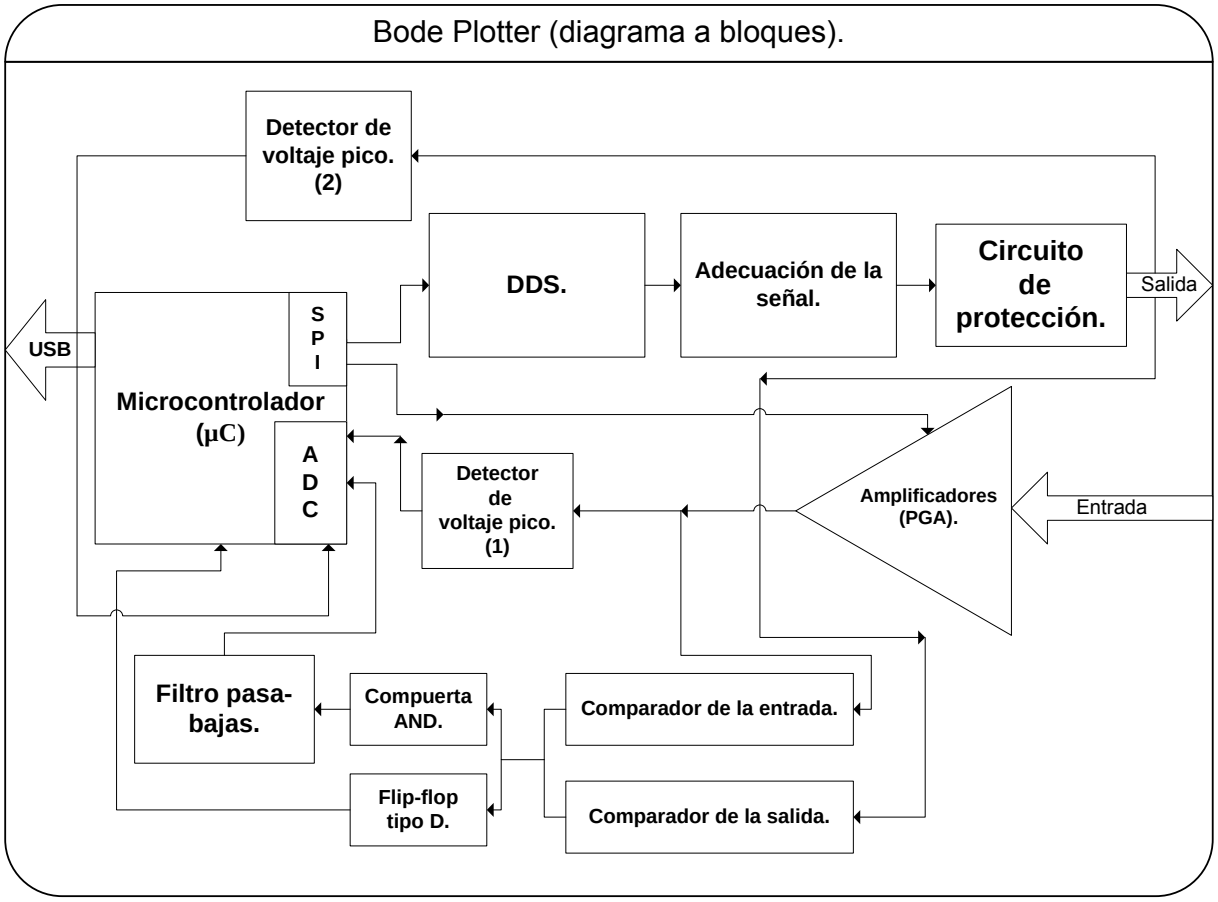


Figura 3.1. Diagrama a bloques del Bode Plotter.

3.3 Generación de la señal

Para analizar un filtro electrónico es necesario introducirle una señal sinusoidal debido al concepto de la respuesta en frecuencia. Estas señales que se introducen al filtro son de valores y magnitudes conocidas, estas frecuencias se inyectan al filtro de cuatro maneras distintas, y no me refiero a que varíen el valor de la magnitud de la señal de entrada al filtro, si no del modo del análisis que se puede realizar variando el barrido de frecuencias, de los cuatro tipos de análisis, tres son de frecuencias fijas

o frecuencias ya definidas, o sea, se le mandan al filtro en un barrido de frecuencias, 62 frecuencias para ser exactos que varían desde 1Hz hasta 100kHz, y un último análisis de manera configurable, tal que el usuario puede decirle al hardware de que frecuencia empezar a analizar el filtro, a que frecuencia terminar el barrido y con que pasos de frecuencia debe de hacerlo, claro que hay un rango máximo y mínimo que puede aceptar como ya lo mencionamos antes en el primer capítulo, pero en este caso se propuso una frecuencia máxima diferente, que va desde 1Hz hasta 1MHz, este cambio para comprobar si el DDS como los demás circuitos son capaces de soportar frecuencias más altas. Este último análisis con una resolución máxima y mínima configurada de tal manera que no genere las ventanas auxiliares de error que se mencionarán más adelante.

3.3.1 Síntesis digital directa

Conocida por el método DDS que se muestra en la figura 3.1, esta parte se encarga de generar la frecuencia de entrada al filtro, es muy diferente a la manera con la que se generan las señales al proyecto pasado [11, 12], en el proyecto pasado se utiliza el circuito integrado XR2206 y el microcontrolador, y como para señales de 1Hz a 1.8kHz el XR2206 genera mucho ruido, para evitar eso se generan las frecuencias de 1Hz a 1.8kHz por medio del microcontrolador, generando pasos de frecuencia y mandándolos a un DAC y después pasando la señal proveniente del DAC a un filtro pasa bajas, y para realizar eso se necesitan más circuitos ya que se deben de separar la señal generada por medio del PIC de la generada por el XR2206, después de generar la señal con el PIC, el XR2206 genera las demás frecuencias variando una resistencia que usa el integrado, para variar esa resistencia se emplea un IC, el cuál es un potenciómetro digital, que al mandarle un dato por el módulo SPI del PIC cambia su valor en la resistencia y cambia la frecuencia, y por eso esta manera de generar la señal en el proyecto pasado necesita más circuitos, es más complicado y demás.

En esta versión utilizamos un circuito integrado de muy alta tecnología que genera señales de muy amplio rango de frecuencias, que van desde 0.1Hz hasta 12.5MHz, que es mucha la diferencia de los circuitos usados en el proyecto pasado [11, 12] que sólo podían generar hasta 1MHz y con mucho menos eficiencia que este IC, menos resolución y exactitud. Es por eso que usamos el DDS AD9833 de Analog Devices. Este C.I. tiene la capacidad de generar dos señales de frecuencias y fases diferentes una de la otra con sólo cambiar un registro de control, este integrado tiene dos registros para la frecuencia de 28 bits y 2 registros para la fase de 12 bits, o sea, uno para cada frecuencia que se puede generar, y gracias al tamaño del registro tiene una resolución de hasta .004Hz para una frecuencia de reloj de 1MHz, en el caso del este proyecto se tiene una frecuencia de reloj de 25MHz lo que hace que el DDS AD9833 tenga una resolución de 0.1Hz.

3.3.2 Adecuación de la señal

Debido a que la naturaleza del DAC del DDS AD9833 nos entrega un voltaje puramente positivo, debemos adecuar la señal para poderla introducir al filtro electrónico en estudio, ya que esta señal debe de ser bipolar. Para esto se utiliza un OP-AMP en configuración de amplificador inversor debido a que con esta configuración se le puede dar una ganancia menor a 1 a la señal proveniente del DDS y así generar la señal de entrada al filtro como se desea, con ayuda de potenciómetros manuales se realiza esta tarea proporcionándole un OFFSET a la señal y una ganancia, claro, que esta ganancia y este OFFSET el usuario lo podrá calibrar manualmente por medio del programa en la computadora y por medio de un osciloscopio, aunque no es recomendable que se haga muy frecuentemente, debido al desgaste del los componentes. Hay que considerar que cuando se modifica la ganancia y el OFFSET es necesario que el programa tenga conocimiento de cual es el voltaje de entrada, es por eso que el Firmware del microcontrolador lee el voltaje de entrada cada vez que va a analizar y se manda en el primer bloque. En la figura 3.2 se muestra el circuito de adecuación de la señal, donde, el pot 1 es el que

proporciona la ganancia con la resistencia de 10kΩ y el pot 2 proporciona el OFFSET a la señal.

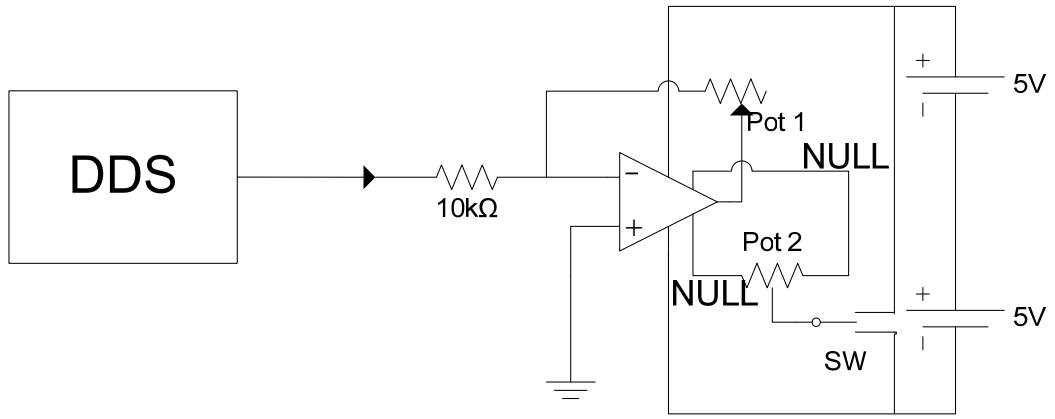


Figura 3.2. Circuito de adecuación de la señal.

3.4 Medición de las señales

¿Cómo saber el comportamiento de los filtros electrónicos, cómo saber la ganancia en magnitud de éste y el comportamiento de la fase?, es por eso la importancia de esta etapa del proyecto. Los parámetros más importantes que se capturan, como se acaba de mencionar, son la magnitud de salida del filtro y su fase las cuáles estudiaremos con un poco más detalle más adelante.

3.4.1 Captura de la magnitud

Para medir la magnitud del filtro se utiliza un detector de pico que se muestra en la figura 3.1, como el detector de voltaje pico (1) que se introduce a un canal analógico del microcontrolador que contiene un ADC de 12 bits, éste le da una resolución de 1.22mV, utilizando como referencia 5 volts. En el proyecto pasado [11, 12] se tenía otro microcontrolador con un ADC de 10 bits, el cual reduce la resolución y la ganancia estaba dada y sigue estando dada por:

$$G(dB) = 20\log_{10}\frac{Vsalida}{Ventrada} \quad (3.1)$$

Y esto se mencionó por que antes se generaba una señal sinusoidal con un voltaje pico fijo a conectar a la entrada del filtro a probar de 500mV, y debido a la naturaleza del ADC del microcontrolador se obtenía que el rango de la ganancia en decibeles máximo y mínimo que se podía capturar es desde 20dB hasta -40dB. Pero ahora es un poco diferente ya que el voltaje de entrada es variable según el gusto del usuario y eso amplía o disminuye el rango de magnitud que se puede medir. Es por eso que se mide el voltaje pico de entrada al filtro cada vez que se va a analizar, evitando así complicaciones en el programa principal.

3.4.1.1 Amplificadores PGA

En el diagrama a bloques, ¿por qué se utilizan los amplificadores PGA (amplificador de ganancia programable por sus siglas en inglés, Programmable Gain Amplifier)? se preguntarán, estos amplificadores que se colocan (en este caso son 4 amplificadores en serie) es para una función importante y tienen como finalidad aumentar más el rango de decibeles negativos de la gráfica logarítmica, si cuando la señal proveniente del filtro es muy pequeña, ésta se amplifica para así darle más amplitud a la señal y poderla medir en un mayor rango, esto tiene sus ventajas: hace la gráfica más suave y aumenta la resolución, lo que proporciona un mejor análisis en la gráfica, pero claro que también tiene sus desventajas como: si el filtro en estudio tiene un OFFSET éste también será amplificado y la medición del voltaje pico y ganancia sería errónea. A estos amplificadores se les configura la ganancia que va desde 1, 2, 5, 10, 20, 50 y 100 veces, por medio del módulo SPI del microcontrolador. Un detalle muy importante de los amplificadores es que al aumentar la ganancia de éstos la impedancia de entrada disminuye, haciendo que el voltaje de entrada disminuya a su vez, más que nada cuando son filtros cuya impedancia de salida es muy alta, como es el caso de los filtros pasivos, ocasionando que al medir la magnitud y variar la ganancia la gráfica tenga saltos en la magnitud, es por eso que se implementó un

buffer diseñado con un amplificador operacional en la entrada, evitando el error en la medición de la magnitud.

3.4.1.2 Consideraciones para la medición de la magnitud

Algunas consideraciones que se tienen en esta etapa, que es la medición de la magnitud de los filtros electrónicos, son:

- Al leer el canal analógico de la señal que proviene del detector de voltaje pico para la medición de la magnitud se mide ésta 16 veces y se promedian los valores para poder contrarrestar el ruido que se pueda generar en el detector de voltaje pico.
- La manera como se ajusta la ganancia en los amplificadores es considerando una tabla, la cual se hizo de tal manera que al amplificar o al ajustar la ganancia nunca se sobrepase o sature la señal, por lo que se propone un rango, el cual es de .5 menor al voltaje máximo que pueden proporcionar los amplificadores, y en este caso es de 5 volts a -5 volts, lo que nos da un máximo de 4.5 volts pico en la señal de salida para todo el rango de ganancias.

3.4.2 Captura de la fase

Para medir la fase del filtro electrónico en estudio, se utilizan dos comparadores de cruce por cero, éstos generan una señal cuadrada a partir de la señal que se les introduce, así generan una señal con valor promedio de 0 Volts a 5 volts. El trabajo de uno de ellos es comparar la señal de salida, o de entrada al filtro, y la del otro es comparar la señal de entrada al dispositivo, o de salida del filtro. Una vez comparadas las dos señales con respecto a tierra, estas dos señales se introducen a una compuerta AND, la cuál su tarea es variar el ciclo útil de la señal de acuerdo con el desfasamiento entre la entrada y la salida y así ésta introducirla a un filtro pasabajas RC de 4to orden con una frecuencia de corte de 1Hz, esto para obtener

solamente la componente de DC de la señal proveniente de la compuerta AND y poderla medir directamente en un canal del ADC del microcontrolador, así pues cuando el ciclo útil de la señal varíe, la componente de DC de salida del filtro pasabajas cambia proporcionalmente al ciclo útil y éste a la fase, por lo que tenemos la medición de la fase (al menos de 0º a 180º). Ahora, sabemos que los filtros se desfasan positivamente como negativamente, la pregunta sería, ¿Cómo saber si el desfase es positivo o negativo?, la respuesta a esa pregunta está en utilizar un flip-flop de tipo D el circuito integrado 74LS74A, el cual tiene dos señales de entrada una la del reloj y otra la entrada D, en este caso el flip-flop deja pasar el nivel lógico que tiene la entrada D cada flanco positivo que entra en el reloj, así si la señal tiene una fase positiva la entrada D tiene un nivel lógico positivo y al pasar el flanco positivo en el reloj la salida Q de éste nos proporcionaría un nivel lógico 1 y en caso que la fase sea negativa, sucedería lo contrario. Y de esta manera evitamos el uso del módulo de captura del microcontrolador y la señal se mide en una forma más rápida y más exacta, en la figura 3.3 se muestra el circuito que se encarga de detectar la fase. Pero esto lo comprobaremos en el capítulo siguiente, donde se discutirán los resultados y las mejoras a la fase, a la medición de la magnitud y a la generación de las señales de entrada al filtro.

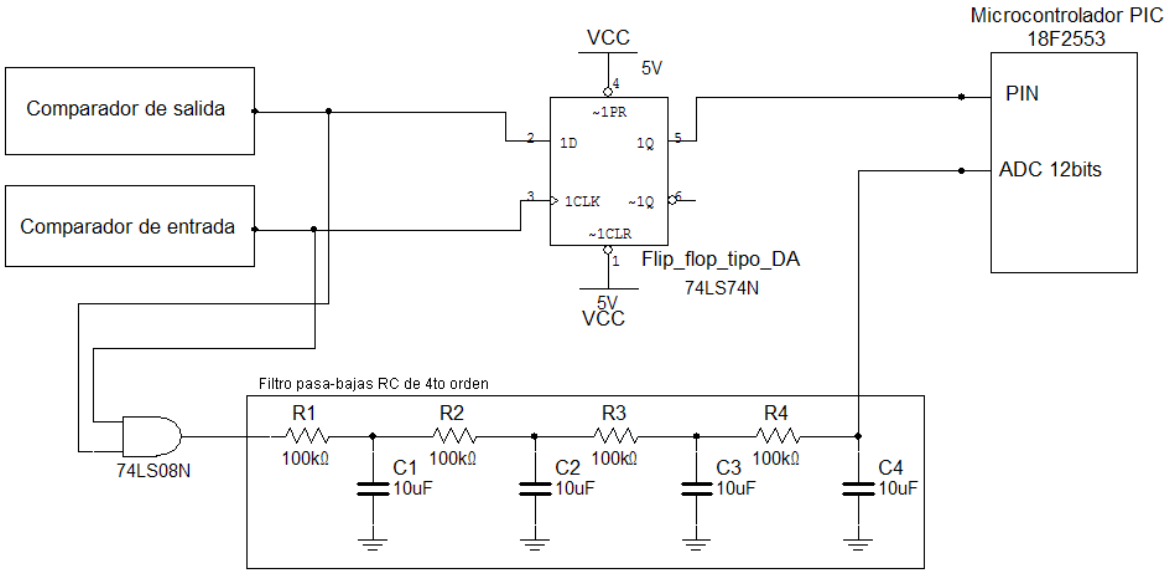


Figura 3.3. Circuito del detector de fase.

3.4.2.1 Consideraciones de la medición de la fase

Éstas son algunas consideraciones y detalles que se deben tener en cuenta al analizar los filtros electrónicos:

- Como ya lo hemos remarcado muchas veces antes, el diseñador del filtro debe de tener en cuenta en crear un filtro con el menor OFFSET posible, ya que si éste cuenta con un OFFSET, al comparar las dos señales con respecto a tierra, éstas no tendrán el mismo ciclo útil el cuál debería de ser del 50% cada una y esto provocaría que el análisis tenga un error de medición.
- Cuando la señal proveniente del filtro es demasiado pequeña, el comparador ya no puede comparar esta señal, es por esa razón que la medición en donde el filtro tiene una magnitud muy pequeña de salida el análisis es erróneo, y si el mismo filtro genera un OFFSET, éste provocaría también una medición errónea, por lo que no tener OFFSET puede significar la diferencia entre si la medición de la fase da un valor positivo o negativo.

Estos detalles o consideraciones que se plantean aquí son los mismos por los cuáles se generaban problemas en la medición de la fase en la versión pasada del proyecto [11, 12], la diferencia está en que no se mencionaban. Estos mismos problemas pueden afectar también la medición en la magnitud pero no de una manera tan crítica como el de la medición de la fase.

3.5 Circuito de protección para sobrecarga

Todas las etapas del proyecto son importantes para el análisis de los filtros electrónicos, pero para proteger la integridad del dispositivo que se está empleando es necesaria esta etapa. Se entiende por sobrecarga cuando la impedancia de entrada del sistema a prueba, en este caso los filtros electrónicos, disminuye provocando que aumente la corriente de salida del circuito del dispositivo que se está utilizando. Esto tiene varias desventajas, puede provocar la atenuación de la

magnitud de la señal de salida, la cuál es la señal de entrada al filtro o al sistema en estudio, lo que provocaría un dato erróneo en las mediciones de su magnitud. Y al aumentar la corriente de salida del sistema en estudio y a su vez en la entrada el dispositivo Bode Plotter causaría que los elementos en los cuáles se compone nuestro dispositivo se dañen. Por eso la importancia de este circuito para la protección de nuestro sistema.

3.5.1 Diseño del circuito de sobrecarga

En el diseño del circuito de protección se iba a utilizar el diseño del circuito pasado [11, 12], pero se observó que este circuito estaba mal diseñado, la idea en sí está bien, pero el diseño no funcionaba como se tenía en mente debido a las malas configuraciones que se implementaron más que nada en los amplificadores operacionales. Básicamente el principio es el mismo, pero en este diseño se propone una resistencia de 1.5Ω para medir la corriente de salida de nuestro sistema que su función es que al reducirse la impedancia de entrada del filtro en estudio se reduce la tensión de la salida de nuestro dispositivo provocando que la corriente aumente, nuestra tarea es evitar que la corriente exceda 10mA, por eso al caer una corriente de 10mA en la resistencia se produce un voltaje de 15mV, esa señal es muy pequeña para que la podamos detectar en un comparador así que esta señal es amplificada 330 veces gracias al amplificador de instrumentación, pero como esta señal es bipolar y debido a la configuración del detector de voltaje pico, solamente detecta el voltaje pico positivo, así que necesitamos que sólo pasen voltajes picos positivos, es por eso que en la etapa siguiente de nuestro circuito de protección se coloca un puente de diodos para obtener el valor absoluto de los voltajes picos, debido a la naturaleza de este puente que no tiene un punto de tierra es necesario colocar otro amplificador de instrumentación con ganancia de 1, ya que la señal es puramente positiva, pasa al detector de voltaje pico, la señal proveniente del detector de voltaje pico entra a un comparador, comparando la señal con 5V y activa el relé con la ayuda de un opto acoplador cuando se exceden estos 5V, así, evitando que la

señal pase al filtro evitamos que aumente la corriente y se dañe nuestro sistema. En la figura 3.4 se muestra el diagrama del circuito de protección.

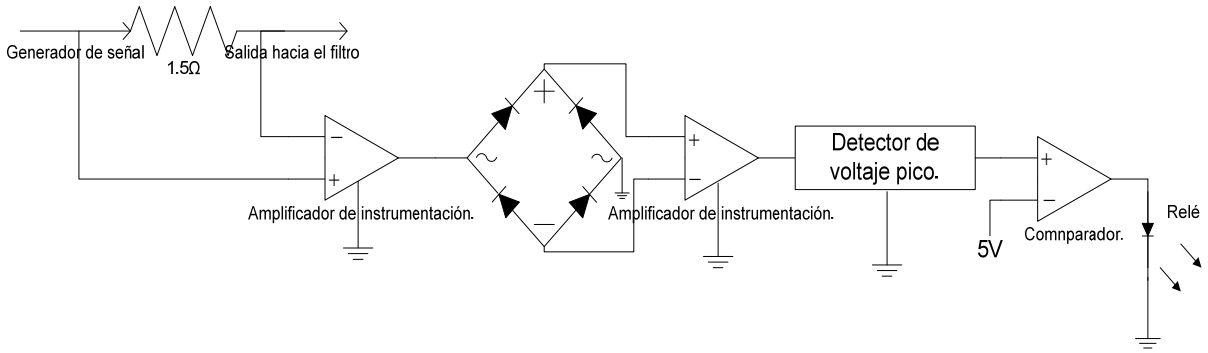


Figura 3.4. Circuito de protección para sobrecarga.

3.6 Conexión USB

La comunicación con la PC como lo hemos dicho antes se hace por medio del puerto USB que en versiones anteriores [11, 12] se hacía por medio de un puerto serial RS-232 el cuál es más lento que el USB. El protocolo USB utiliza solamente 4 líneas, las cuales se observan en la tabla 3.1, dos de ellas son la alimentación que provee el puerto de 5V y GND, y las otras dos son los datos D+ y D-. Debido a que el protocolo USB maneja niveles lógicos de 5V a 0V no es necesario convertir la señal, ya que el microcontrolador utiliza esos niveles de voltaje, es en caso contrario al protocolo RS-232 que se necesitaba la ayuda de un integrado que elevara ese voltaje proveniente del microcontrolador a 15V para luego ser enviado a la computadora.

Tabla 3.1. Asignación de las terminales del conector USB.

Número	Nombre de la señal	Asignación típica del alambrado	Descripción
1	VBUS	Rojo	+5V
2	D-	Blanco	Data -
3	D+	Verde	Data +
4	GND	Negro	Tierra

En la figura 3.5 se muestra el microcontrolador utilizado y se pueden observar los pines D- y D+ para la conexión USB, y como el microcontrolador trabaja con los niveles lógicos del USB, no es necesario otro circuito para que se pueda realizar la comunicación USB. El D- está especificado en el pin número 15 y el D+ en el pin 16 de nuestro microcontrolador, en nuestro proyecto la alimentación del USB de 5V es utilizada para alimentar el microcontrolador y el DDS AD9833 y se espera que se pueda alimentar todo el proyecto por medio del puerto USB, ya que además de proporcionar 5V es capaz de proporcionar un máximo de 500mA de corriente. Nuestro proyecto usa un conector tipo B para el dispositivo y un tipo A para la conexión a la computadora.

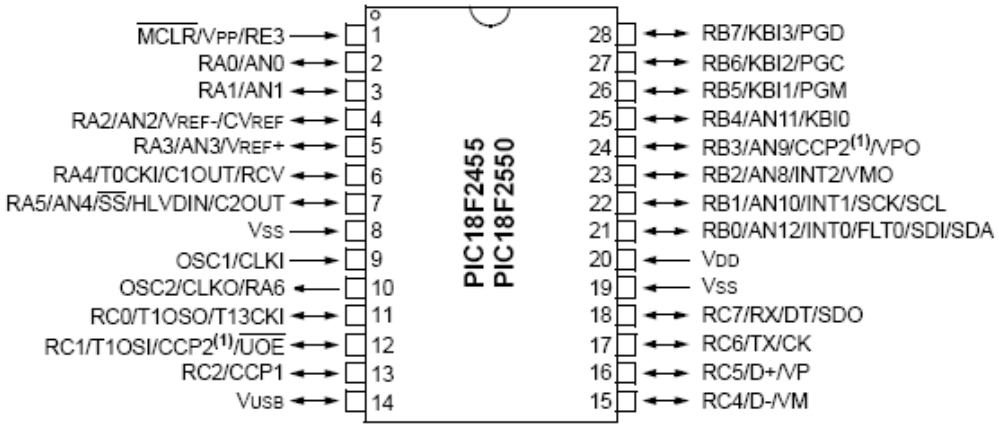


Figura 3.5. Diagrama de pines del microcontrolador PIC 18F2553.

3.7 Algoritmo del Firmware para el microcontrolador

En esta sección del capítulo se muestra el algoritmo del Firmware del microcontrolador, hay varias consideraciones que hay que aclarar, que no se ven en el diagrama debido a que son pequeños detalles que nos quitarían espacio y que no son tan importantes como para ponerlo en el diagrama. Esos pequeños detalles no tan importantes se explicarán en forma escrita más adelante en esta misma sección del capítulo III. En la Figura 3.6 se muestra el algoritmo del Firmware del microcontrolador que controla el Bode Plotter.

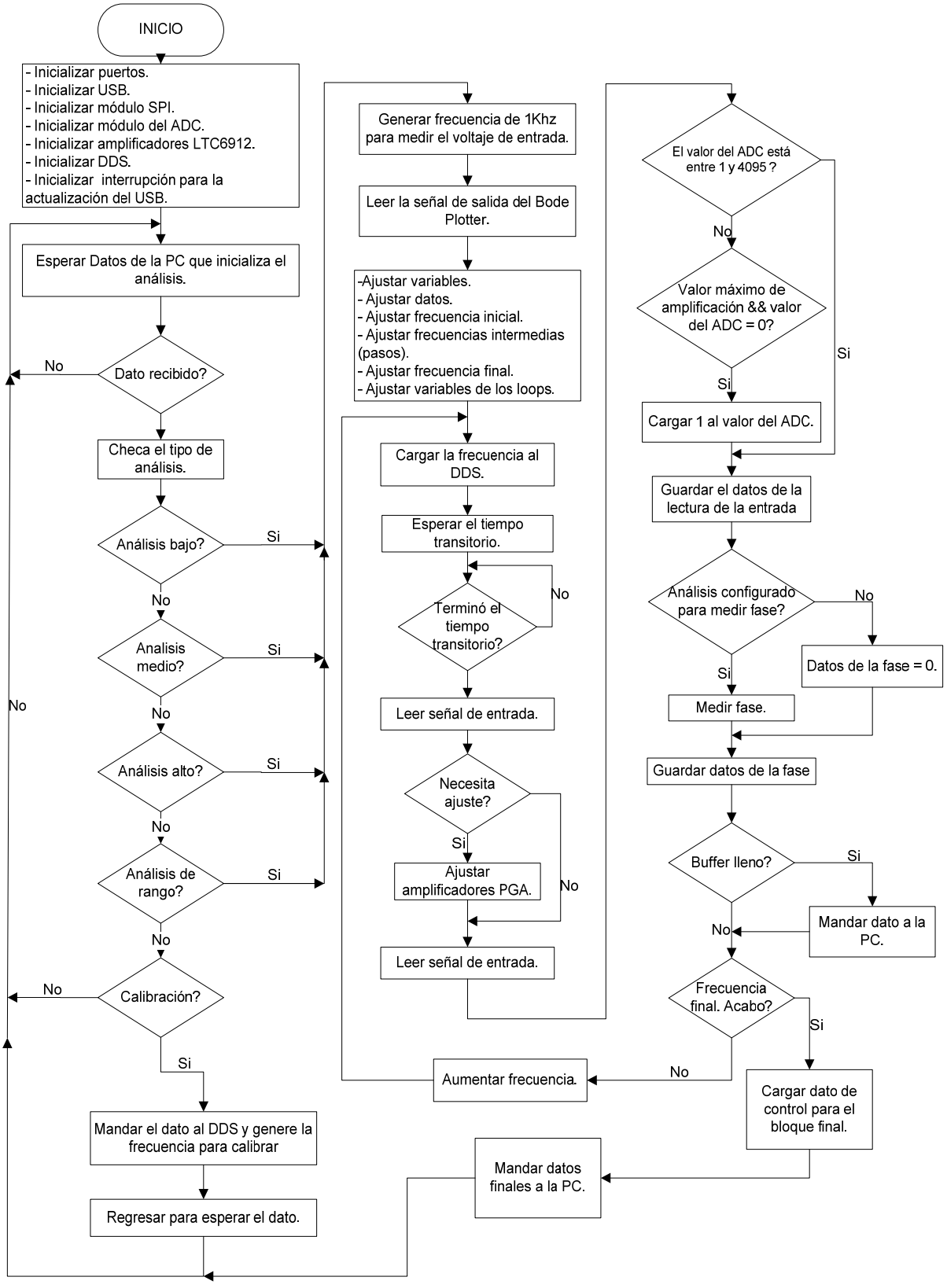


Figura 3.6. Algoritmo del Firmware del Bode Plotter.

3.7.1 Detalles y consideraciones del algoritmo del Firmware microcontrolador

Hay que considerar que cada vez que se habla de señal de entrada nos referimos a la señal proveniente del filtro en análisis o a la señal de entrada al sistema, y cuando nos hablamos de la señal de salida nos referimos a la señal de salida de nuestro sistema o entrada al filtro en análisis, o en pocas palabras, siempre nos referimos a las señales del dispositivo hacia el filtro.

Hay detalles que se tienen que indicar de manera escrita, ya que si las hubiésemos puesto en forma de diagrama de flujo nos ocuparía demasiado espacio y creo que se entienden mejor de manera escrita. Estos son los detalles:

- Siempre que leemos señales, como por ejemplo, las señales de entrada y la de salida, éstas se leen en el ADC del microcontrolador, debido a que son señales sinusoidales de voltaje variable, éstas tienen que pasar por un detector de voltaje pico para poderlas leer de una manera más fácil con el ADC.
- Cada vez que se le indica al detector de voltaje pico que detecte el voltaje pico se tiene que dejar pasar 500μs para que pueda detectar el voltaje pico máximo de manera correcta.
- Considerando que hay frecuencias más lentas que otras, es por eso que para que el voltaje pico llegue al detector de voltaje pico, es necesario también proporcionarle más tiempo para que este voltaje pico de la señal llegue al detector y no tome valores intermedios de la señal, es por eso que se implementó una función que se describirá más adelante que proporciona más tiempo dependiendo en que frecuencia se encuentre. Si este tiempo no se hubiese implementado es probable que el detector de voltaje pico, independientemente de los 500μs que se le da para estabilizarse, tome valores intermedios de la señal. Gracias a las pruebas que se hicieron para verificar errores es que pudimos solucionar este problema.

- Cuando el ADC lee un valor 0 y los amplificadores están amplificando al máximo se le carga un valor 1 como default para evitarnos problemas al hacer el cálculo de ganancia en dB.
- Cada vez que se va a analizar un filtro, se genera primero una frecuencia de 1kHz con el DDS para medir el voltaje de salida, esto es por la naturaleza de la ecuación de ganancia en forma logarítmica que depende de la señal de entrada del filtro o de salida de nuestro dispositivo.
- Desde el programa principal de la computadora se puede indicar al microcontrolador si se desea medir la fase, para no perder tiempo extra en medir de la fase además de la magnitud. En caso que se desee medir la fase, se tiene que dejar pasar 40 segundos para que se estabilice el filtro pasa-bajas RC de 4to orden y por cada frecuencia subsiguiente se le da un segundo para que se estabilice. En caso contrario que no se quiera medir la fase, el análisis se vuelve más rápido midiendo únicamente la magnitud.
- Se considera un tiempo de descarga del detector de voltaje pico para que el voltaje de su salida llegue a cero y no afecte en la medición de la próxima vez que se le indique que lea el voltaje pico

3.7.2 Resumen del algoritmo del microcontrolador

El algoritmo en sí se ha venido modificando para hacer el análisis de filtros mucho más confiable y con mediciones mucho más exactas y menos errores, y haciendo el algoritmo más eficiente y fácil de implementar.

3.8 Algoritmo de las interrupciones del microcontrolador

El algoritmo de las interrupciones es muy sencillo, básicamente lo que se interrumpen son puros temporizadores que son utilizados para los tiempos de estabilización del detector de voltaje pico, del tiempo de transición, de los tiempos para la fase y uno de las más importantes interrupciones es la de la actualización del puerto USB. En la figura 3.7 se observa el diagrama de las interrupciones, hay que

recordar que este microcontrolador tiene dos vectores de interrupciones, los cuáles son el alto (high) y el bajo (low), esto significa que tiene dos prioridades de interrupción que son:

- Si se activa la interrupción alta, el microcontrolador se encargará de atenderla, en caso de que al estar atendiendo la interrupción alta se activa alguna interrupción configurada como baja, el microcontrolador se encargará de terminar de atender la interrupción alta primero y ya que termine de atender la interrupción alta irá a atender la interrupción baja.
- Si se activa una interrupción baja, el microcontrolador se encargará de atenderla, en caso de que se interrumpa la alta al momento de estar atendiendo la baja, dejara de atender la baja para ir a atender la alta, ya después de atender la alta regresará a la baja.

Recordando que es muy importante tener en prioridad alta la rutina de la actualización del puerto USB debido a que si no se actualiza cada 100µs la computadora nos mandará a un estado pasivo durmiendo el puerto y proporcionando menos corriente, esto indica que tanto el puerto USB de la computadora, como el microcontrolador entran a un estado pasivo consumiendo menos corriente, es un estado donde el microcontrolador no hace nada más que esperar a despertarse y esto se hace por medio de software.

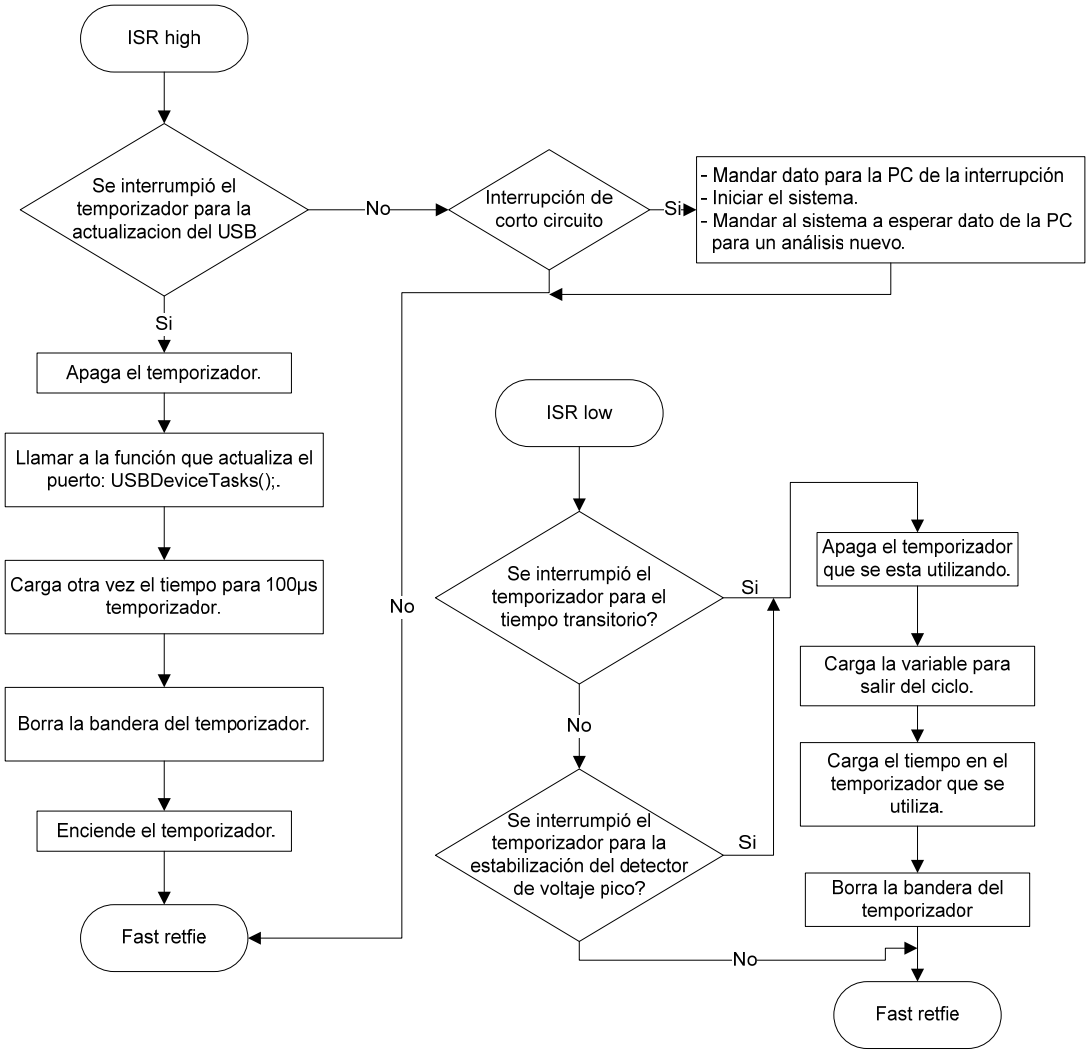


Figura 3.7. Algoritmo de las interrupciones del Firmware Bode Plotter.

Hay que tomar en cuenta que el temporizador que se usa para proporcionar tiempo para el tiempo transitorio de los filtros, se utiliza también para darle tiempo a que se estabilice el filtro pasa-bajas RC de 4to orden en caso que se desee la fase, es por eso que se le carga el valor al temporizador dependiendo de lo que se desee hacer con él.

3.9 Descripción de las funciones del código para el lenguaje C Bode Plotter

En esta sección del capítulo hablaremos de las diferentes funciones que fueron creadas, algunas lo fueron para probar el funcionamiento de diferentes componentes

y permanecieron debido a que han seguido siendo utilizadas para diferentes partes del mismo Firmware. Recordando que no hablaremos de todas las funciones ya que son demasiadas contando las que utilizamos nosotros, las que creamos y las que se utilizan para la inicialización del puerto USB, así que hablaremos de las más relevantes y las que se ven en el programa principal. Aclarando que sólo revisaremos el código de esas funciones creadas por nosotros ya que nos tomaría mucho trabajo innecesario colocar aquellas que no son tan importantes.

3.9.1 Función `void InitializeSystem(void)`

Esta función centraliza la rutina de inicialización. Todas las rutinas requeridas para la inicialización del USB son llamadas por esta función, también la inicialización de la aplicación debe de ser llamada desde aquí, como la configuración de los puertos, temporizadores, módulos, interrupciones, inicialización de variables y demás, esta función sólo se llama una vez al principio de la aplicación.

3.9.2 Función `void USBDeviceTasks(void)`

Como lo hemos mencionado antes esta función es la encargada de la actualización del puerto USB, evitando así con esta función que la computadora y el puerto USB nos manden a un estado pasivo.

3.9.3 Función `void YourHighPriorityISRCode(void)`

Función que se encarga de controlar la rutina de interrupción alta. Es donde se coloca el código de las interrupciones que se deseen utilizar y el trabajo que hacen cada una de las interrupciones configuradas.

3.9.4 Función `void YourLowPriorityISRCode(void)`

Función que se encarga de controlar la rutina de interrupción baja. Es donde se coloca el código de las interrupciones que se deseen utilizar y el trabajo que hacen cada una de las interrupciones configuradas.

3.9.5 Función `void ProcessIO(void)`

Esta función es donde toma lugar para las rutinas de la aplicación, en esta función se usan una combinación de las tareas de la misma aplicación tanto como tareas de la comunicación USB.

3.9.6 Función `void BlinkUSBStatus(void)`

Esta función se encarga de prender o apagar los LED's del estado del USB, dependiendo en que estado se encuentre el dispositivo, los estados a los que nos referimos son los estados que hicimos hincapié en el capítulo II en la sección 2.11.13.2.

3.9.7 Función `void USBDeviceInit(void)`

Función encargada de inicializar al módulo USB configurando las SFR's (registros de funciones especiales, por sus siglas en inglés) y las variables del Firmware para conocer el estado del puerto USB.

3.9.8 Función `void SEND_DDS(unsigned long FRECUENCIA)`

Esta función fue creada por nosotros, se encarga de enviar los datos al registro de frecuencia al DDS para controlar la frecuencia de salida del AD9833. Esta función recibe una variable de 32 bits y la divide en 4 bytes que manda de 2 bytes en 2 bytes al DDS, acomodando el dato e incluyéndole los bits de configuración para que el

DDS pueda saber a que registro se le está escribiendo, debido a que nosotros únicamente usamos un registro, sólo le agregamos los bits de configuración para el DDS. Esta función utiliza una función también creada por nosotros que se encarga de convertir una variable de 32bits y convertirla en 8bits especificándole que 8 bits debe tener.

3.9.9 Función `unsigned char MAKE8(unsigned long var, unsigned char offset)`

Esta función se encarga de convertir una variable 32bits en una variable de 8bits especificándole que 8bits de la variable de 32bits desea tener considerando que 32bits se divide en 4 para tener 4 bytes, lo que quiere decir, que esta función regresa 1byte de los 4 bytes en los que se puede dividir exactamente la variable de 32bits, que quiere decir esto, que la función sólo regresa 1byte que puede ser el primer paquete de 8bits, el segundo paquete de 8 bits, el tercero u el cuarto, lo quiere decir que no puede regresar bits intermedios, por ejemplo el 2do bit hasta el bit 9 que serían también 1byte.

3.9.10 Función `void SEND(void)`

Esta función también envía los datos para modificar la frecuencia del DDS al registro del mismo, sabemos que sólo usamos un registro de frecuencia de los dos para generar la salida del DDS, así que sólo lo mandamos a uno, igual que la función anterior que mandaba datos al registro de frecuencia del DDS, la diferencia de esta función a la `SEND_DDS()`, es que esta función no acepta ninguna variable, si no que manda los datos que provienen ya de la computadora, estos ya contienen los bits de configuración y están acomodados, los toma de el buffer de entrada del USB que manda la computadora.

3.9.11 Función `void INICIA_DDS(void)`

Función encargada de inicializar el DDS AD9833, configura el tipo de onda que quieres en la salida, y resetea el DDS.

3.9.12 Función `void INICIA_AMPS(void)`

Esta función se encarga de inicializar los amplificadores PGA, inicializándolos a una ganancia de 1, para que se puedan utilizar, evitando que los amplificadores se inicialicen en un estado que no corresponde.

3.9.13 Función `void WRITE_AMP(unsigned char STEP)`

Función encargada de activar los amplificadores PGA dependiendo, en caso del que el ADC del microcontrolador lea como valor 0, esta función se utilizó antes en el viejo algoritmo de amplificación pero después de implementar el `void CONTROL_LTC(void)` se sigue utilizando para en caso de que la lectura del ADC al iniciar el análisis sea cero, esta función se activa hasta que lea algo. Lo malo de esta función que la manera en que amplificaba la señal era poco efectiva por que sólo tenía 4 niveles de amplificación de 100 en 100, y se iniciaban en cada frecuencia que se cargaba al DDS.

3.9.14 Función `void CONTROL_LTC(void)`

Esta función se encarga de ajustar la señal, en la figura 3.4 esta función entra en la parte de si la señal necesita ajuste, esta función trabaja de tal manera que amplifica la señal proveniente del filtro, leyendo el ADC en el momento y verificando este valor, y dependiendo de ese valor amplificar la señal de modo que no pase de los 4V pico. Esta función modifica a su vez una variable que es mandada a la computadora y le indica en que nivel de amplificación está para poder convertir la lectura del ADC a su valor original.

3.9.15 Función `void TIMERS(void)`

Esta función se encarga de inicializar los temporizadores, configurando sus registros para la frecuencia con la que trabajará cada temporizador, si trabajarán en modo de 8bits o 16bits, en caso que lo tenga y carga los valores para la configuración del temporizador usado para la función de actualización del puerto USB.

3.9.16 Función `void IINTERRUPCION(void)`

Función que se encarga de inicializar las interrupciones de los temporizadores para las diferentes tareas como: el temporizador encargado de actualizar el puerto USB cada 100μs, para el ciclo finito de la variable para el tiempo transitorio que se debe proporcionar a los filtros en estudio, y para el tiempo de estabilización del detector del voltaje pico.

3.9.17 Función `void TIEMPO_A(void)`

Función encargada de mandar al temporizador los tiempos para que dependiendo de la frecuencia que se está analizando se cargue un valor al temporizador y se pueda estabilizar el detector de voltaje pico y así pueda tomar el valor máximo y no tomar valores intermedios de la señal. Esta función se usa para cuando el análisis es el definido y no el configurado, para el configurado se usa otra función.

3.9.18 Función `void TIEMPO_R(void)`

Función que se encarga de hacer lo mismo que la función anterior `void TIEMPO_A(void)` sólo que esta función es para el análisis configurado, ya que sus ciclos dependen de otras variables de las cuáles no depende el análisis ya definido.

3.9.19 Función `void LEER_ADC(void)`

Esta función se encarga de leer el ADC, no importando para que canal esté configurado, y hace un promediado para disminuir el ruido en la lectura y tener una medición más exacta. Esta hace 16 lecturas del ADC cargándola a una variable, claro donde esa variable sea tan grande para poder leerlas 16 veces sin que se desborde, y dividiendo esa variable por el mismo número de lecturas que se hacen en este caso 16, por medio de un corrimiento de 4 hacia la derecha.

3.9.20 Función `USBGenRead(BYTE ep, BYTE* data, WORD len)`

Esta función se encarga de recibir la información proveniente en la computadora de un específico endpoint en un buffer, recibe de 8, 16, 32 y 64 bytes, dependiendo de lo que se utilice o lo que se desea utilizar, o dependiendo de las necesidades de la aplicación.

3.9.21 Función `USBGenWrite(BYTE ep, BYTE* data, WORD len)`

Esta función se encarga de mandar los datos a la computadora dependiendo del endpoint desde donde se desea mandar, de igual manera esta función puede enviar desde 8, 16, 32 y 64 bytes, dependiendo de las necesidades de la aplicación o de lo que se desea hacer con la información.

3.9.22 Resumen

Hay que considerar como ya se ha mencionado con anterioridad que cada lectura que se hace al ADC del microcontrolador, esta señal pasa por un detector de voltaje pico, haciendo más fácil la medición al microcontrolador. Muchas de estas funciones se crearon para hacer primero funcionar los componentes por separado antes de emplearlas en nuestro proyecto como tal, muchas de éstas han sido modificadas para mejorar su funcionamiento.

3.10 Desarrollo del software

Una de las partes fundamentales del proyecto es el software que va a recibir los datos en la computadora y los va a mostrar en las gráficas de Bode. Este software es el que controla todo el sistema, tanto el recibir datos, acomodarlos, y graficar, como indicarle al microcontrolador cuando empezar a trabajar; además interactúa con el usuario final proporcionándole un ambiente amigable y fácil de entender para emprender la tarea de analizar filtros electrónicos.

El lenguaje que se utilizó fue C++ debido a varias características que hacen al lenguaje fiable, como los siguientes puntos:

- Tiene el poder y la extensibilidad para escribir programas de larga escala. Muchos de esos programas los usas en tu computador personal todos los días.
- Ha sido certificado 99.9% como un estándar puro. Esto lo hace un lenguaje portable, debido a que hay un compilador de C++ para cada gran sistema operativo, y me refiero a gran sistema operativo como Windows, y todos estos sistemas operativos soportan el mismo lenguaje C++.
- Existe una herramienta para la programación visual de este lenguaje.
- En este lenguaje existen mucha información disponible para aprender acerca del lenguaje C++.
- Se tienen ejemplos de el uso de funciones y librerías para la comunicación por medio del puerto USB proporcionados por Microchip.
- Es uno de los lenguajes más utilizados.

Pero sobre todos esos puntos y más, es un lenguaje de programación el cuál es fácil de entender para el programador, en este caso yo, y en el cuál me siento muy cómodo al programar, y eso es lo más importante en un lenguaje de programación el hecho de que el programador se sienta cómodo y se familiarice con el mismo.

3.10.1 Herramienta de desarrollo visual

Una vez seleccionado el lenguaje de programación en el cuál se va a programar nuestro proyecto, es momento de elegir la herramienta en la que se va a programar, una herramienta de las más avanzadas y que se eligió para esta tarea fue la de Visual Studio 2005, ya que nos permite desarrollar de manera fácil y rápida nuestra aplicación y puede ejecutarse bajo el sistema operativo de Windows. No entraremos en mucho detalle por que ya que se hace referencia en el capitulo II en la sección 2.16.

3.10.2 Estructura del programa

Al igual que la versión anterior [11, 12] de nuestro proyecto, con la diferencia que la versión anterior se programó en DELPHI, Visual C++ es un lenguaje que trabaja por medio de eventos lo que significa que Windows ejecuta más que la inicialización del programa y empieza a trabajar cada vez que sucede dicho evento, como puede ser el clic en un botón del programa, pero también maneja funciones que siempre se están ejecutando, como la función del programa principal.

Básicamente para crear un nuevo proyecto en Visual C++ se ejecutan los mismos pasos que la versión anterior [11, 12], primero se abre el programa en el cuál se va a trabajar, en este caso Visual Studio 2005, y se crea un nuevo proyecto, después de este paso nos aparece el componente principal llamado Form1.h, el cuál el programa trabaja mostrándote este en dos formas, una forma es mostrándote el Form1 en forma de código y otra en forma de diseño de modo de que en forma de diseño tu puedes agregar componentes solamente arrastrándolos hacia la Form1 y modificar sus propiedades seleccionándolos y cambiando sus propiedades, en forma de código que te muestra todo el código que necesita la máquina para generar la Form1, pudiendo el programador agregar código para que la aplicación ejecute lo que el programador desee. Es imposible poderles decir aquí como trabaja esta herramienta avanzada de programación, ya que el objetivo del proyecto es el análisis de filtros

electrónicos y su respuesta en frecuencia por medio de gráficas de Bode, así que les recomiendo investigar por su cuenta, igual se puede abordar el tema de la programación pero es mejor y se aprende más si uno empieza programando haciendo ejemplos sencillos, claro está si uno como persona está interesado, se quiere dedicar a la programación, le gusta el programar o simplemente quiere aprender más sobre este ámbito sin importar el lenguaje.

Básicamente el programa principal es encargado de empezar el análisis, indicarle al microcontrolador o al dispositivo que tipo de análisis está solicitando el usuario, detener el análisis, comunicarse por el puerto USB con el dispositivo, enviar y recibir información al dispositivo y por supuesto graficar la respuesta en frecuencia de los filtros, entre otros, es por eso que es de suma importancia el programa principal.

3.11 Interfaz del usuario

Es tarea del programador crear un programa que sea amigable, fácil de usar, de manera que el usuario pueda familiarizarse con el programa de una forma muy rápida, y que sea a prueba de fallas o errores del usuario. La interfaz del programa principal, que se describirá con más detalle más adelante, es del tamaño de la pantalla del ordenador, en este programa se muestran controles y diversos mecanismos de configuración para el análisis de filtros electrónicos, pero cuenta también con otras ventanas auxiliares que son de apoyo para la aplicación.

3.11.1 Programa principal

En la figura 3.8 se muestra la pantalla principal del programa de Bode Plotter, es la pantalla que se muestra al ejecutar el programa, se procuró diseñar una interfaz sencilla sin dejar de ser práctica ni funcional para que el usuario final pueda observar sus funciones fácilmente.

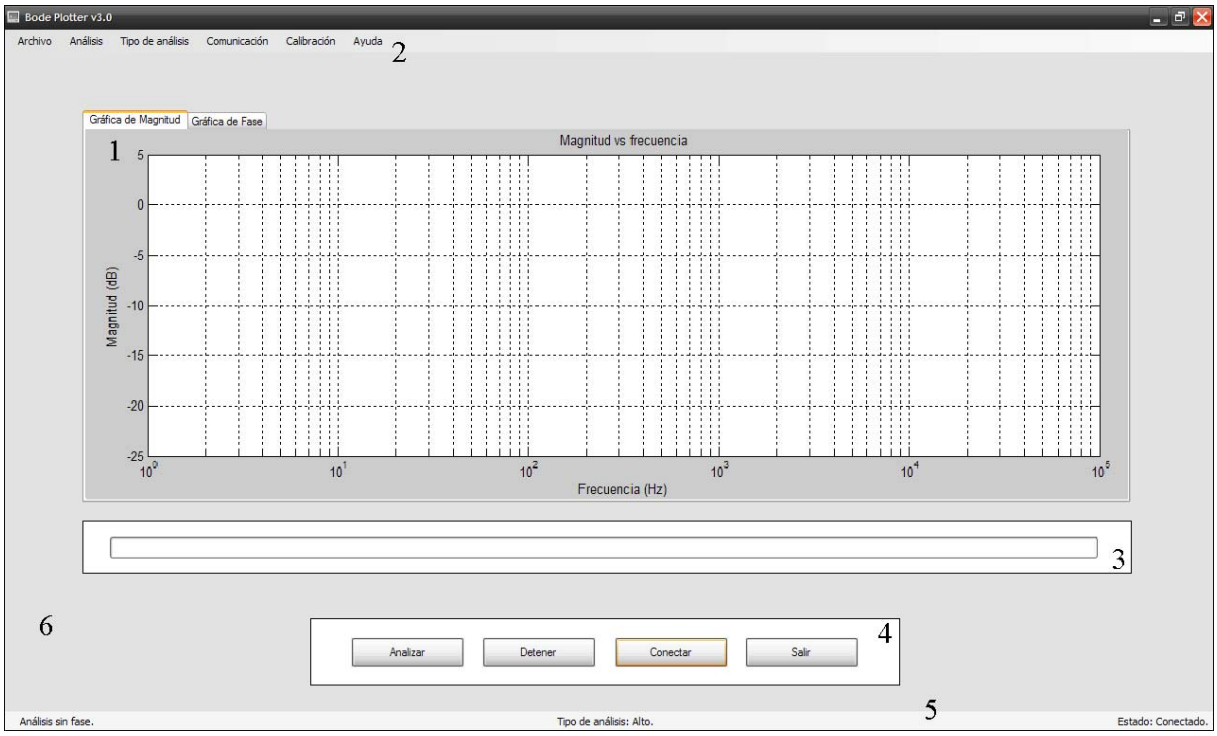


Figura 3.8. Pantalla principal del programa Bode Plotter.

Sección 1. En esta sección se muestran las gráficas logarítmicas de magnitud y fase, las cuáles son los componentes principales del programa, debido a que el despliegue de las gráficas nos proporciona la información de la respuesta en frecuencia en magnitud y fase de los filtros, el cuál es el objetivo de nuestro proyecto. Para graficar en nuestro proyecto se utilizó la ayuda de MATLAB, en la figura 3.8 se muestra la gráfica del MATLAB, debido a que en la elaboración del proyecto no se contaba con la gráfica, y que se tendrá que hacer o conseguir en el futuro. Y por que no usar la gráfica del proyecto pasado [11, 12], bueno es por que la gráfica del proyecto pasado está programada en el lenguaje DELPHI. Pero para nuestra fortuna empleamos en MATLAB creando un archivo .dat con las mediciones para cargar éste en MATLAB y posteriormente graficarlo.

Sección 2. Éste es un menú principal del programa donde contiene todos los controles para que el usuario pueda controlar el programa, en este menú se encuentran controles para la comunicación USB, para que el dispositivo se pueda conectar con el programa, para especificarle el tipo de análisis que se desea realizar

que va desde bajo, medio alto y el de rango configurable, también se encuentra el control para analizar y detener, y la opción de indicarle si el análisis se desea con fase o sin fase, entre otros.

Sección 3. Este elemento es una barra de progreso, que le indica al usuario si ha empezado el análisis y en que porcentaje del proceso se encuentra. Claro que los pasos con los que varía esta barra de progreso depende del tipo de análisis que se lleve a cabo.

Sección 4. A éste se le conoce como panel de control, contiene los botones principales para el control total del análisis del filtro, estos controles también son accesibles por medio del menú de la sección 2, los botones y su función son:

- **Analizar.** Una vez configurado el tipo de análisis, recordando que al ejecutarse el programa el análisis por default es el alto, este botón inicia el análisis del filtro en estudio.
- **Detener.** La función de este control detiene el análisis del filtro en estudio.
- **Conectar.** Este botón conecta el programa principal al dispositivo que usa el puerto USB.
- **Salir.** Este control sale de la aplicación, en este caso el Bode Plotter.

Este es un punto muy importante en el programa, el cuál es que mientras el programa no se conecte con el dispositivo muchos de los controles que se usan para el análisis del filtro, para calibrar entre otras cosas, permanecen deshabilitados, o sea no se pueden utilizar, esto es para que el programa no pueda ocasionar fallos en la computadora o en el dispositivo.

Sección 5. Ésta sección representa un control en la barra de estado de Windows contiene información del estado de nuestra aplicación, en este caso nos muestra el estado de la conexión entre la aplicación y el dispositivo, nos indica si en el análisis se va a analizar la fase y nos indica que tipo de análisis es el que se está eligiendo.

En la figura 3.9 se muestra una imagen de cómo cambia esta barra de estado dependiendo del análisis que se desea aplicar al filtro.

Sección 6. Esta sección hace hincapié a la pantalla principal, que nos sirve de contenedor de todos los controles y componentes del programa

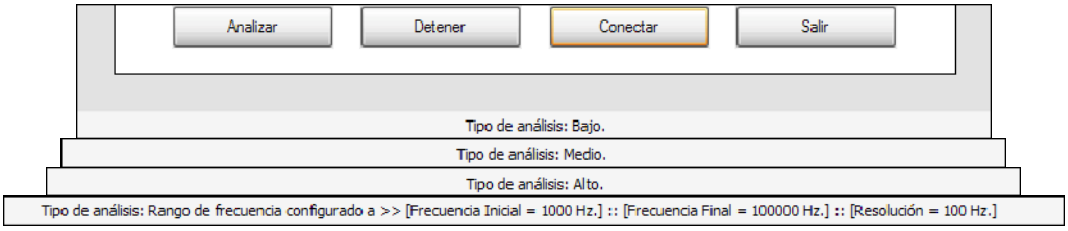


Figura 3.9. Cambio en la barra de estado según el análisis seleccionado.

3.11.2 Pantallas auxiliares

El programa cuenta con pantallas auxiliares que tienen como trabajo el complementar las funciones de nuestro programa reforzando la interacción entre el usuario, la computadora y el dispositivo.

Figura 3.10. Pantalla auxiliar que muestra información del programa Bode Plotter.

La pantalla auxiliar de la figura 3.10 muestra información acerca de el creador del programa principal, el nombre, la versión y la especificación de lo que se trata el programa.

La figura 3.11 muestra el menú de donde se genera la pantalla auxiliar para darle un rango configurado de frecuencia al análisis del filtro. La figura 3.12 muestra la pantalla auxiliar donde se le indica al dispositivo la frecuencia inicial, la final y la resolución o pasos del análisis del filtro.

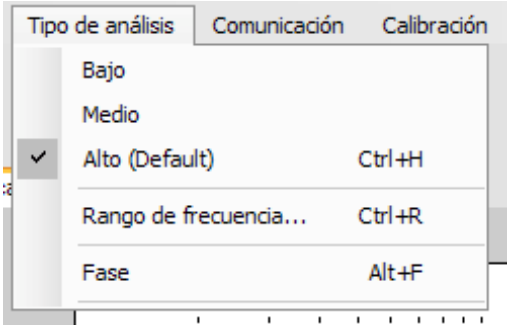


Figura 3.11. Menú que genera la pantalla auxiliar para elegir el rango de frecuencia.

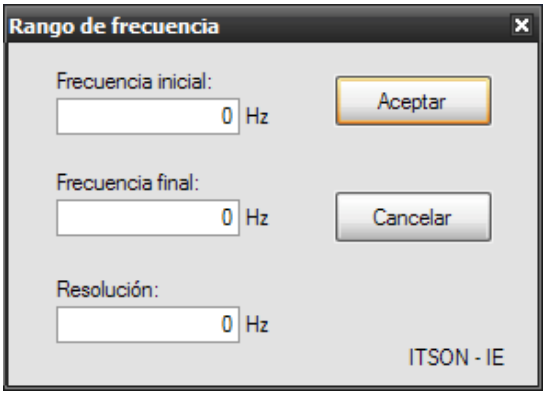


Figura 3.12. Pantalla auxiliar Rango de frecuencia.

Como hemos mencionado antes, la pantalla auxiliar Rango de frecuencia configura la frecuencia inicial, final y la resolución del análisis del filtro, al darle clic en aceptar el programa carga en unas variables la información que se le introduce a esta pantalla auxiliar, considerando varios factores y para evitar errores en la introducción de datos y a su vez para evitar errores en el análisis del filtro, la pantalla auxiliar Rango de

frecuencia sólo acepta un tipo de información y muestra pantallas auxiliares a ésta dependiendo de los datos introducidos, son 9 los errores que generan a su vez 9 diferentes pantallas auxiliares dependiendo del error, los cuáles son:

- Solamente acepta números enteros positivos, en caso de introducirles letras, símbolos, números negativos y números con decimales genera una pantalla auxiliar indicándole al usuario el error.
- En caso de que todos los valores introducidos sea 0, el programa genera una pantalla auxiliar indicándole que las frecuencias no han sido configuradas.
- En caso de que el valor introducido a la frecuencia inicial sea 0, el programa generará una pantalla auxiliar indicando al usuario que la frecuencia inicial no puede ser 0 y que su valor mínimo es 1.
- La frecuencia final no puede ser menor o igual a la frecuencia inicial, como se sabe en el estudio de los filtros la frecuencia final no puede ser mayor que la inicial, por eso el programa genera una pantalla auxiliar para indicarle al usuario del error.
- La resolución no puede ser mayor o igual a la diferencia entre la frecuencia final y la frecuencia inicial, esto genera un error pues debido a que es imposible ese tipo de análisis y genera a su vez una pantalla auxiliar indicando este error al usuario.
- La resolución no puede ser 0, generando una pantalla auxiliar indicando el error.
- La frecuencia final no puede ser 0, esto genera una pantalla auxiliar indicando el error.
- La frecuencia final no puede ser mayor a la frecuencia máxima de análisis que es de 100kHz.
- Y por último el programa genera una pantalla auxiliar si el número de frecuencias a analizar es menor de 10, o sea, menor de 10 pasos.

En caso de que el usuario haga clic en cancelar de la pantalla auxiliar Rango de frecuencia, se le indicará al mismo usuario que no hay frecuencia seleccionada y el

análisis será en la última configuración seleccionada como se muestra en la figura 3.13.

Figura 3.13. Pantalla auxiliar indicando que el análisis se realizará con la última configuración seleccionada.

Cuando el usuario desea calibrar el dispositivo, se va al menú calibración y éste genera una pantalla auxiliar para elegir si se desea calibrar, en caso que si, el dispositivo genera una señal para poder calibrarlo manualmente, la pantalla auxiliar se aprecia en la figura 3.14. y en la figura 3.15 se muestra la pantalla que indica terminado el análisis del filtro.

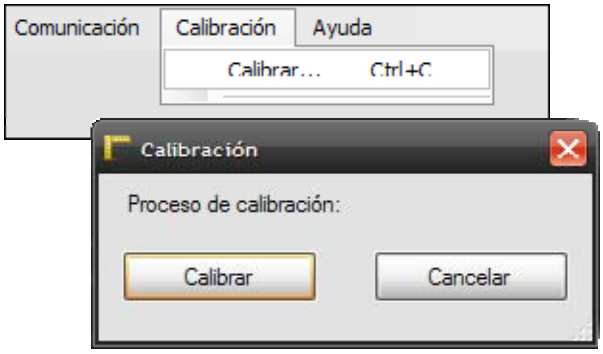


Figura 3.14. Pantalla auxiliar para la calibración del hardware.

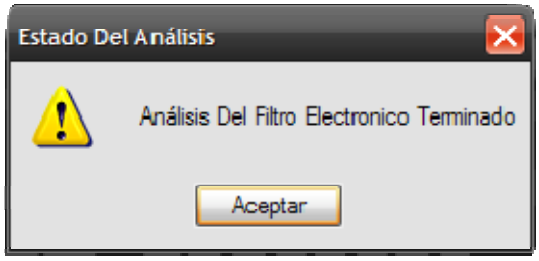


Figura 3.15. Pantalla auxiliar que indica el final del análisis del filtro electrónico.

CAPÍTULO IV
Pruebas y resultados

Como todo trabajo de investigación es necesario realizarle pruebas y verificar los resultados, es por eso que se dedica un capítulo completo para hacer diferentes clases de pruebas y comparar resultados, en este caso, con la versión anterior [11, 12]. Se describirán de todas las pruebas que se le hicieron al proyecto como la comunicación USB, la generación de la señal del DDS, y el análisis de diferentes tipos de filtros. Dichas pruebas se realizaron con material de apoyo del laboratorio de electrónica del ITSON tratando de cubrir todas las expectativas y los puntos de más importancia a las pruebas de nuestro dispositivo con el equipo que tenemos a nuestro alcance.

4.1 Configuración del dispositivo USB

Como todo dispositivo que se conecta a la computadora necesita de una configuración para poderse comunicar con él, como lo hemos mencionado antes el USB requiere de un controlador para que pueda realizar esa tarea, el cual se lo proporcionamos gracias a Microchip. El controlador proporciona la interfaz para que el sistema operativo se pueda comunicar con el dispositivo y hacer lo que queramos con él. Una vez que el dispositivo es conectado, la computadora hace todo el trabajo, como ya hemos mencionado, la computadora se encarga de detectar una caida de tensión, una vez detectándola le hace peticiones al dispositivo, peticiones mencionadas en el capítulo II, y el sistema operativo nos muestra el siguiente mensaje en la barra de tareas, ver la figura 4.1.

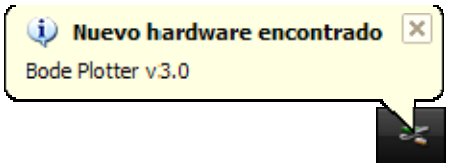


Figura 4.1. Mensaje del sistema operativo cuando detecta el dispositivo.

El sistema operativo muestra el nombre de nuestro proyecto debido a que lee los descriptores dentro de él, estos descriptores son obtenidos por medio de peticiones.

Una vez que el sistema operativo detecta el dispositivo, se encarga de asignarle un controlador de su base de datos, si de todos los controladores el sistema operativo no puede encontrar un controlador que funcione con el dispositivo nos mostrará un cuadro pidiéndonos que le proporcionemos el controlador del dispositivo, debido a que es un dispositivo creado por nosotros, debemos proporcionarle este controlador con los datos necesarios para que coincida con los datos leídos en el dispositivo y pueda realizar la comunicación con él, esto se hace através de un archivo .INF que es el que contiene toda la información del controlador para el sistema operativo, este archivo .INF es mostrada en el apéndice A. En la figura 4.2 se muestra la ventana de petición para el controlador del sistema operativo.

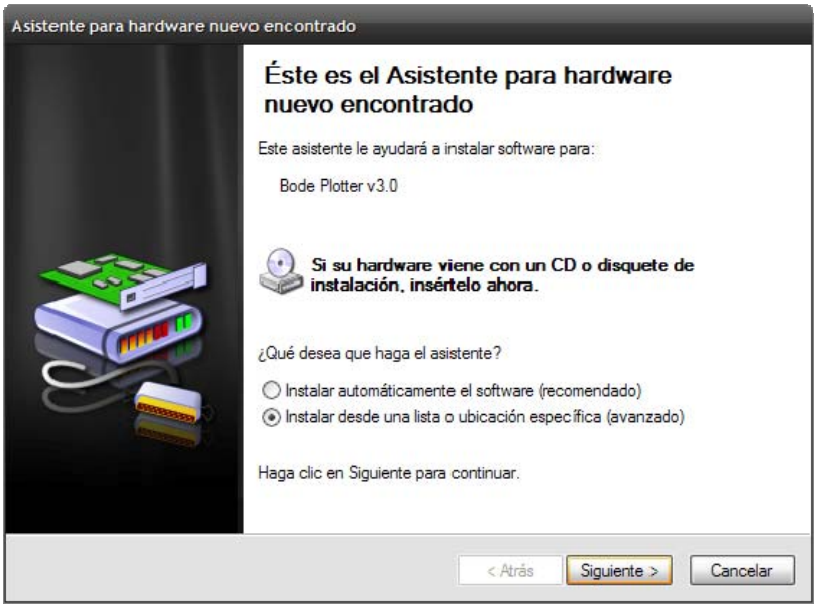


Figura 4.2. Ventana de petición del controlador del sistema operativo.

Debido a que nosotros le proporcionaremos el controlador tenemos que seleccionar la opción de: instalar desde una lista o ubicación específica (avanzado), al darle clic en siguiente el sistema operativo muestra otra pantalla para buscar el controlador ubicando la dirección donde tengamos el controlador en el disco duro de nuestra computadora, esta pantalla de búsqueda se muestra en la figura 4.3.

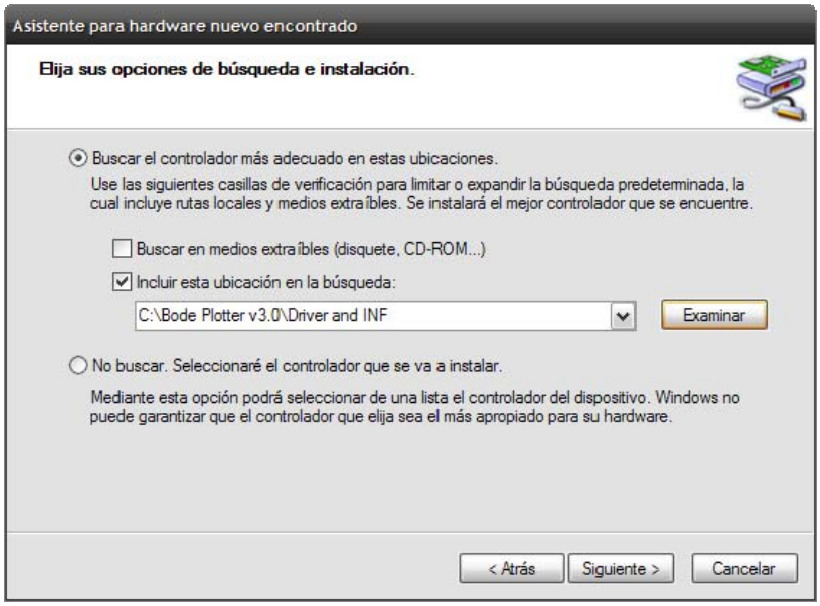


Figura 4.3. Pantalla de búsqueda del controlador para el dispositivo Bode Plotter.

Una vez que se le indica al sistema operativo donde se encuentra nuestro controlador el siguiente paso del sistema operativo es instalar el controlador mostrándonos en una pantalla el porcentaje de este proceso, mostrado en la figura 4.4.

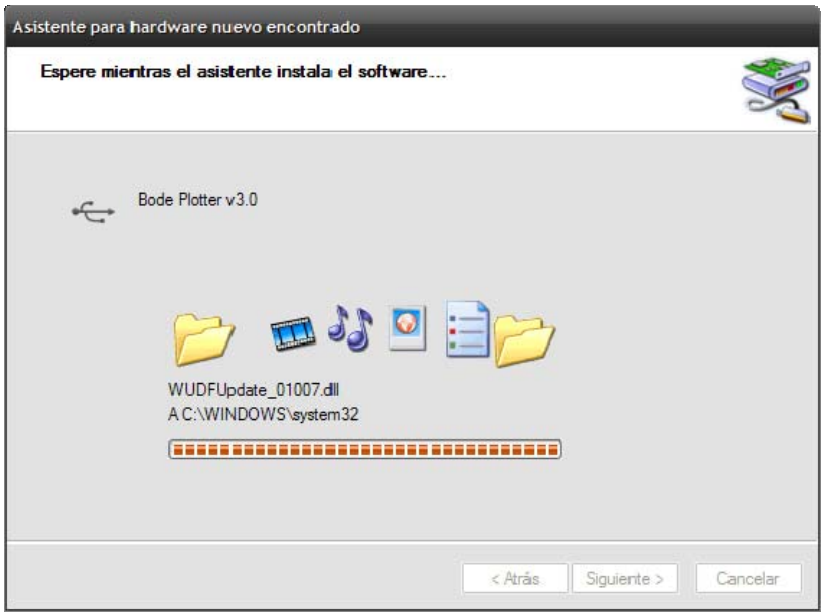


Figura 4.4. Pantalla del sistema operativo mostrando el porcentaje del proceso de instalación.

Una vez completado el proceso de instalación, el sistema operativo muestra una pantalla de que el proceso ha sido terminado, y nos proporciona la información si el dispositivo ha sido instalado correctamente o no. En la figura 4.5 se muestra la etapa de finalización de la instalación.

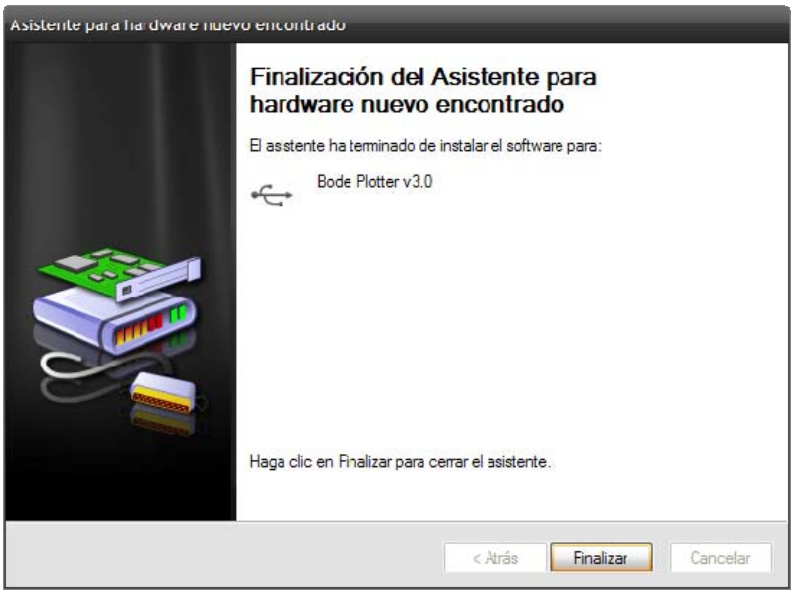


Figura 4.5. Pantalla del sistema operativo de la finalización de la instalación del dispositivo.

Para poder ver si el dispositivo está instalado correctamente, conectado o en que estado se encuentra trabajando, se pueden observar utilizando el administrador de dispositivos que se encuentra siguiendo los siguientes pasos: se le da clic derecho al ícono de Mi PC y se mostrará un menú, nos vamos a la opción de propiedades y aparecerá una pantalla con el nombre de: Propiedades del sistema, en la ficha de Hardware le buscamos el botón de Administrador de dispositivo, dándole clic nos muestra la pantalla de la figura 4.6 y en esta pantalla podemos ver nuestro dispositivo en caso de que esté conectado y dándole clic izquierdo nos proporciona toda la información que el sistema operativo tiene sobre el dispositivo conectado. Hay que recordar que todo el proceso de la instalación de nuestro dispositivo es sobre la plataforma de Windows XP, en otros sistemas operativos este proceso puede variar, así como en el nombre de cada pantalla. En la figura 4.6 se muestra ya nuestro dispositivo conectado indicado en el cuadro de la misma pantalla.

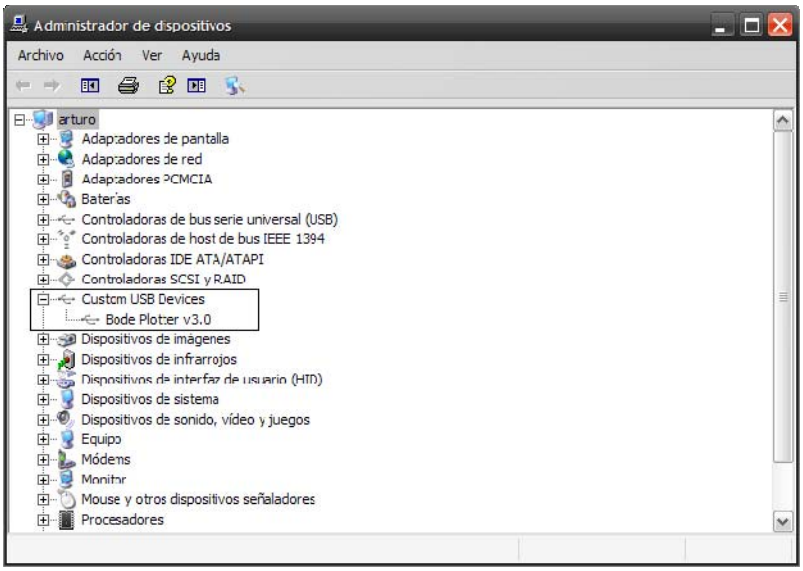


Figura 4.6. Administrador de dispositivos de Windows XP.

4.2 Extracción segura del dispositivo Bode Plotter

Ya que nuestro dispositivo haya pasado por todos esos pasos lo podemos utilizar mandando datos a él, en nuestro caso, lo podemos utilizar para que se conecte con el software y poder analizar filtros electrónicos. Una vez que se deje de utilizar el dispositivo y podamos guardarlo, se puede desconectar de él de manera segura evitando así errores en el dispositivo, aunque es poco probable estos errores debido al protocolo USB, pero para evitar que probablemente ocurra un error es mejor desconectar el dispositivo de una manera segura. Primeramente en el ícono que muestra los dispositivos conectados a través del puerto USB en la barra de tareas de Windows se le da clic derecho y nos muestra un pequeño menú de todos los dispositivos USB conectado a nuestro computador, en este caso sólo tenemos nuestro dispositivo conectado al computador, la figura 4.7 muestra el ícono y el menú para la extracción segura de nuestro dispositivo.

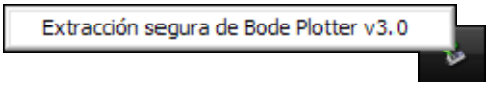


Figura 4.7. Extracción segura del dispositivo Bode Plotter.

Haciendo clic derecho sobre la opción de extracción segura de Bode Plotter v3.0 del este menú, entonces Windows asegura la extracción segura verificando que no se está usando el dispositivo, o que no se está compartiendo ningún tipo de información con el dispositivo, la figura 4.8 muestra el mensaje que genera Windows indicando que el dispositivo puede ser desconectado del puerto USB de manera segura. En caso contrario de que una aplicación u otra cosa esté utilizando el dispositivo Windows proporcionará una pantalla auxiliar indicando que el dispositivo está siendo utilizado.

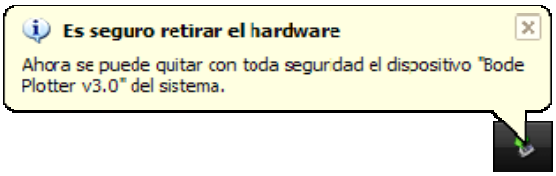


Figura 4.8. Mensaje de Windows para poder retirar el hardware con seguridad.

4.3 Pruebas de los amplificadores PGA

En esta sección hacemos una prueba muy sencilla al amplificador de ganancia programable indicando por medio del módulo SPI al amplificador que nos proporcione una ganancia de 10 a una señal de 1Vp-p. En la figura 4.9 se muestra la respuesta del amplificador PGA.

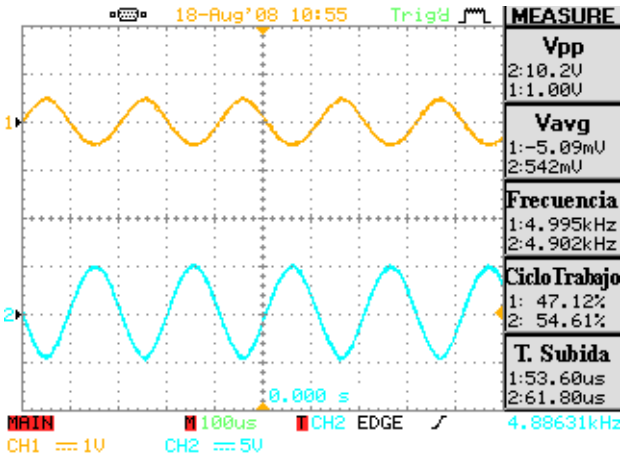


Figura 4.9. Respuesta del amplificador PGA con una ganancia de 10.

En la figura 4.9 la señal de entrada al amplificador es la mostrada en el canal 1 y la salida es el canal 2 de la figura, mostrando que el amplificador PGA trabaja exactamente como lo establecen las hojas de especificaciones del amplificador PGA, hay que tomar en cuenta que la señal de salida nunca sobre pasa la alimentación de los amplificadores, es por eso que la señal de salida no se satura.

4.4 Pruebas al detector de voltaje pico

Esta prueba es muy sencilla, proporcionamos a la entrada del detector un voltaje haciendo que éste detecte el valor pico de una señal. Se hacen pruebas al detector de voltaje pico, para saber cuando considerar el tiempo transitorio de los filtros, si primero se considera el tiempo transitorio después detectar el voltaje pico o al revés, se decidió que es mejor que ejecutar el tiempo transitorio primero y después el detector de voltaje pico, debido a que el detector de voltaje pico al detectar mantiene en su salida el voltaje pico que se le introduce a él sin importar que no esté presente en el todo el tiempo, o sea, que si se presenta una fluctuación en el detector de voltaje pico, detectaría esta fluctuación y se mantendría en este voltaje y nos proporcionaría una medición errónea en el ADC del microcontrolador. Hay una detalle que hay que tomar en cuenta respecto a la alimentación del detector de voltaje pico, y es que la alimentación tiene que ser mayor a la medición máxima que se quiera medir sino no alcanzaría este voltaje y también la medición sería errónea.

4.5 Pruebas al DDS AD9833

En estas pruebas se generaron las mismas señales que se generaron en la versión pasada [11, 12] y se comparan la diferencia de la DDS generada y el XR2206 de la versión pasada con la DDS del AD9833 que es un IC especializado en la DDS. Para las frecuencias generadas para el análisis del rango de frecuencias configurables se realizaron otro tipo de pruebas.

4.5.1 Medición en la frecuencia de barrido

En la tabla 4.1 se muestra la tabla de frecuencias teóricas y reales generadas para el barrido de la versión pasada [11, 12] a este proyecto, estas frecuencias se midieron con la ayuda de un multímetro STEREN MUL-600 debido a que contaba con la facilidad de capturar las frecuencias de salida y al conectarse al computador con el propósito de almacenarlas en tablas de Excel.

Tabla 4.1. Frecuencias teóricas y reales generadas para el barrido de la versión pasada

Frecuencia	Valor teórico	Valor real	Unidad	Error en %
0	1	1.005	Hz	0.50
1	2	2.009	Hz	0.45
2	3	3.009	Hz	0.30
3	4	4.02	Hz	0.50
4	5	5.004	Hz	0.08
5	6	5.98	Hz	0.33
6	7	6.99	Hz	0.14
7	8	8	Hz	0.00
8	9	8.99	Hz	0.11
9	10	10	Hz	0.00
10	12	12	Hz	0.00
11	14	14.01	Hz	0.07
12	16	16.04	Hz	0.25
13	18	18.4	Hz	2.22
14	20	20.02	Hz	0.10
15	30	30.01	Hz	0.03
16	40	40	Hz	0.00
17	50	49.98	Hz	0.04
18	60	60	Hz	0.00
19	70	69.9	Hz	0.14
20	80	79.9	Hz	0.12
21	90	89.8	Hz	0.22
22	100	100	Hz	0.00
23	120	119.9	Hz	0.08
24	140	139.7	Hz	0.21
25	160	159.7	Hz	0.19
26	180	179.4	Hz	0.33
27	200	198.7	Hz	0.65
28	300	296.8	Hz	1.07

29	400	396.3	Hz	0.92
30	500	490.9	Hz	1.82
31	.6	.595	kHz	0.83
32	.7	.69	kHz	1.43
33	.8	.804	kHz	0.50
34	.9	.896	kHz	0.44
35	1	.987	kHz	1.30
36	1.2	1.199	kHz	0.08
37	1.4	1.367	kHz	2.36
38	1.6	1.59	kHz	0.63
39	1.8	1.811	kHz	0.61
40	2	1.992	kHz	0.40
41	3	3.025	kHz	0.83
42	4	4.01	kHz	0.25
43	5	4.998	kHz	0.04
44	6	6.02	kHz	0.33
45	7	6.97	kHz	0.43
46	8	8.04	kHz	0.50
47	9	9	kHz	0.00
48	10	10.02	kHz	0.20
49	12	12.11	kHz	0.92
50	14	14.07	kHz	0.50
51	16	16.21	kHz	1.31
52	18	17.88	kHz	0.67
53	20	19.92	kHz	0.40
54	30	30.26	kHz	0.87
55	40	39.41	kHz	1.48
56	50	50.95	kHz	1.90
57	60	56.4	kHz	6.00
58	70	63.3	kHz	9.57
59	80	72.2	kHz	9.75
60	90	83.5	kHz	7.22
61	100	98.4	kHz	1.60

Como notarán en la tabla 4.1 el error máximo es de 9.75% y la frecuencia no es exacta debido a varios factores, uno es que el potenciómetro digital que se utiliza para el control de las frecuencias en el XR2206 tiene solamente 256 pasos lo que hace que sea muy difícil que las frecuencias sean exactas, y la otra es que las frecuencias generadas de 1 a 1800Hz son generadas a través del microcontrolador por lo que también es muy difícil que sean generadas exactamente. Es por eso que se utiliza el circuito de avanzada generación de señales, el DDS AD9833, la tabla 4.2

muestra las frecuencias teóricas y reales que se generan para el barrido, estas frecuencias fueron medidas con un frecuencímetro de la compañía *BK PRECISION* y generando de la misma forma que se generan en el Firmware de nuestro proyecto pero una frecuencia a la vez y midiendo esta frecuencia con el tiempo más grande de estabilización para medir frecuencias del frecuencímetro utilizado, que en este caso es de 1 segundo, incluso se midieron la señales con un osciloscopio digital Tektronix modelo TDS 2024 que tiene la característica de medir la frecuencia de la señal entre otras, esto para hacer la comparación de las mediciones con diferentes aparatos de medición, y los resultados fueron los mismos.

Tabla 4.2. Frecuencias teóricas y reales del barrido de frecuencias generadas por el DDS AD9833 de Analog Devices.

Frecuencia	Valor teórico	Valor real	Unidad	Error en %
0	1	1	Hz	0.00
1	2	2	Hz	0.00
2	3	3	Hz	0.00
3	4	4	Hz	0.00
4	5	5	Hz	0.00
5	6	6	Hz	0.00
6	7	7	Hz	0.00
7	8	8	Hz	0.00
8	9	9	Hz	0.00
9	10	10	Hz	0.00
10	12	12	Hz	0.00
11	14	14	Hz	0.00
12	16	16	Hz	0.00
13	18	18	Hz	0.00
14	20	20	Hz	0.00
15	30	30	Hz	0.00
16	40	40	Hz	0.00
17	50	50	Hz	0.00
18	60	60	Hz	0.00
19	70	70	Hz	0.00
20	80	80	Hz	0.00
21	90	90	Hz	0.00
22	100	100	Hz	0.00
23	120	120	Hz	0.00
24	140	140	Hz	0.00
25	160	160	Hz	0.00
26	180	180	Hz	0.00

27	200	200	Hz	0.00
28	300	300	Hz	0.00
29	400	400	Hz	0.00
30	500	500	Hz	0.00
31	.6	.6	kHz	0.00
32	.7	.7	kHz	0.00
33	.8	.8	kHz	0.00
34	.9	.9	kHz	0.00
35	1	1	kHz	0.00
36	1.2	1.2	kHz	0.00
37	1.4	1.4	kHz	0.00
38	1.6	1.6	kHz	0.00
39	1.8	1.8	kHz	0.00
40	2	2	kHz	0.00
41	3	3	kHz	0.00
42	4	4	kHz	0.00
43	5	5	kHz	0.00
44	6	6	kHz	0.00
45	7	7	kHz	0.00
46	8	8	kHz	0.00
47	9	9	kHz	0.00
48	10	10	kHz	0.00
49	12	12	kHz	0.00
50	14	14	kHz	0.00
51	16	16	kHz	0.00
52	18	18	kHz	0.00
53	20	20	kHz	0.00
54	30	30	kHz	0.00
55	40	40	kHz	0.00
56	50	50	kHz	0.00
57	60	60	kHz	0.00
58	70	70	kHz	0.00
59	80	80	kHz	0.00
60	90	90	kHz	0.00
61	100	100	kHz	0.00

Como se pueden dar cuenta la generación del barrido de la señal es muy exacta a diferencia de la versión anterior [11, 12], si tiene un error pero es mínimo y está muy por debajo de un Hertz. En la página oficial del DDS AD9833 de Analog Devices se encuentra una herramienta muy importante de diseño para este DDS, el cuál es un asistente de configuración y simula la frecuencia real de salida que debe de tener el AD9833 para cualquier especificación que se le proporcione, en nuestro caso se

cuenta con una frecuencia de reloj de 25MHz, para diferentes frecuencias de salida teóricamente se encuentra diferentes frecuencias de salidas reales de este DDS, La tabla 4.3 muestra diferentes frecuencias de salida que se simulan con ayuda de esta herramienta que nos proporciona el desarrollador del DDS, con su valor real y su valor teórico, esta información nos puede ayudar para que tengamos una idea de cuán grande es el error que genera el DDS AD9833 de Analog Devices.

Tabla 4.3. Valores de frecuencias generadas por la herramienta de simulación de Analog Devices.

Valor teórico	Valor real	Unidad	Error en %
25.0	24.999957531605	kHz	$169.88x10^{-4}$
50.0	50.00000819563866	kHz	$16.38x10^{-6}$
100	100.00001639127731	kHz	$1.63x10^{-5}$
1.00	0.9999999776482582	MHz	$2.24x10^{-6}$
10.0	9.999999962747097	MHz	$3.80x10^{-7}$

Como se puede observar el error que proporciona el DDS AD9833 es demasiado pequeño para que se pueda detectar con alguno de los aparatos disponibles en el laboratorio de electrónica.

4.5.2 Pruebas de calidad de señal

Además de verificar la exactitud de la señal de entrada al filtro hay que verificar otro valor el cuál tiene que ver con que tan pura es la señal, que tanta distorsión armónica posee la señal de entrada. En este caso se realizaron varias pruebas a varias frecuencias midiendo así la señal y verificando sus armónicos según la teoría para medir los armónicos y comparando con la versión anterior [11, 12] para ver que tan pura es la señal. Esta señal se midió con dos aparatos, un osciloscopio de la marca Tektronix modelo TDS 2024 capaz de realizar la tarea de la FFT y proporcionarnos los armónicos, de la misma manera que se hizo en la versión anterior, y la segunda utilizando un analizador de espectros que es un instrumento más especializado en la medición de señales y sus armónicos. Debido a pequeños detalles del analizador de espectros que se fueron tomando en cuenta en el transcurso de las mediciones

como: la impedancia de entrada del analizador de espectro es muy pequeña, de 75Ω, esto repercute en la señal de entrada y proporciona una medición errónea, otra razón es que el rango de medición que tiene este aparato es de frecuencias de 1MHz hasta 1.8GHz y frecuencias fuera de este rango puede causar que su medición sea errónea y debido a que nuestro proyecto la frecuencia máxima que se trabaja es de apenas 100kHz, por razones ya mencionadas con anterioridad, puede provocar errores en la medición, es por eso que se optó por el uso del osciloscopio Tektronix, incluso pondremos dos ejemplos de las mediciones que se tienen del analizador de espectros con respecto al osciloscopio para poder notar la diferencia entre los dos tipos de mediciones, estas comparaciones se realizan en las frecuencias de 4kHz y 600kHz, en la cuál el analizador de espectro en la frecuencia de 4kHz apenas se alcanza a ver.

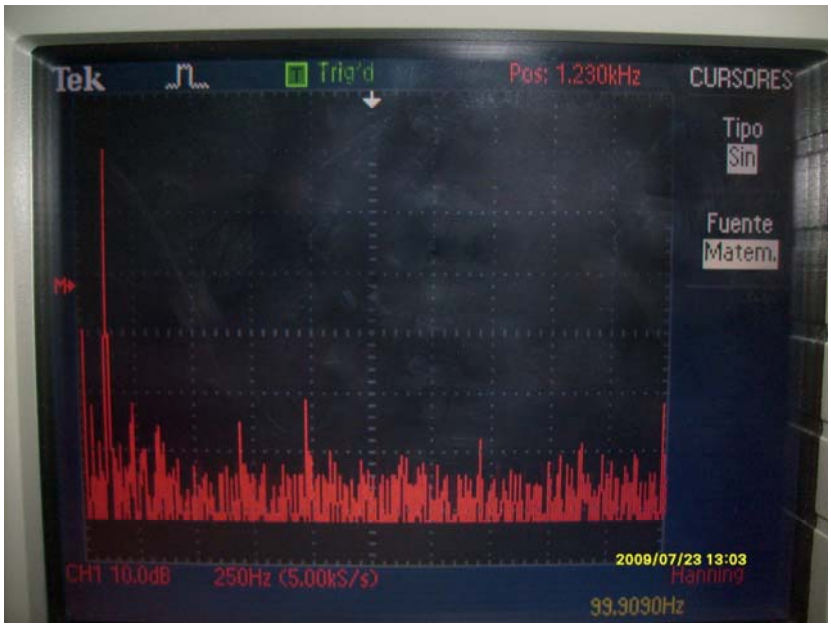


Figura 4.10. Espectro de frecuencias de la señal sinusoidal de 100Hz generada por el DDS AD9833.

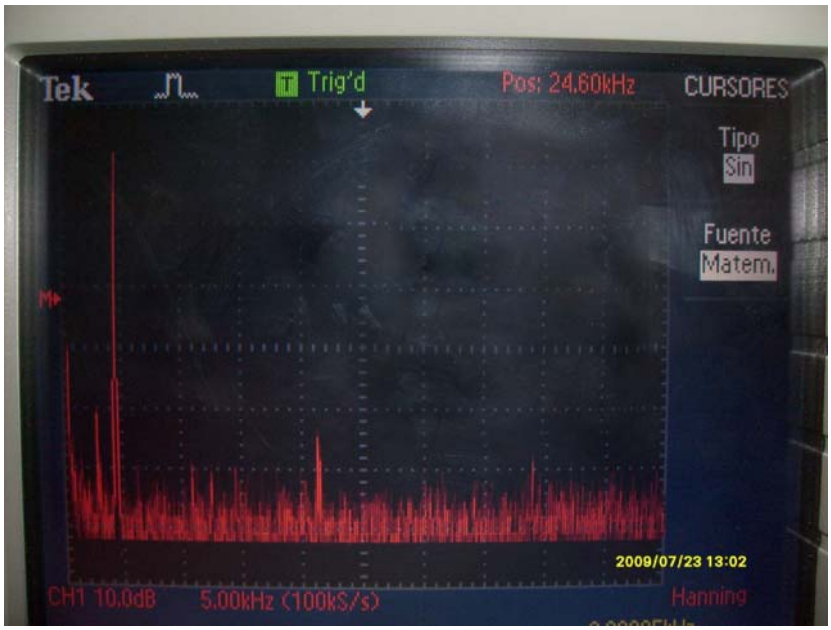


Figura 4.11. Espectro de frecuencias de la señal sinusoidal de 4kHz generada por DDS AD9833.

Tabla 4.4. Valores en decibles de los armónicos en diferentes frecuencias de la señal medidos con el osciloscopio Tektronix modelo TDS 2024.

Frecuencia / **Armónico**	**100Hz**	**4kHz**	**6kHz**	**100kHz**	**600kHz**	**1MHz**
Fundamental	-11dB	-11dB	-11dB	-11dB	-11dB	-13dB
Segundo	-65dB	-70dB	-67dB	-60.6dB	-64dB	-57dB
Tercero	-70dB	-70dB	-61dB	-65.8dB	-65dB	-67dB
Cuarto	-71.1dB	-70dB	-69dB	-67.8dB	-69dB	-70dB
Quinto	-72dB	-70dB	-68.5dB	-67.8dB	-67dB	-59.8dB
Sexto	-73dB	-70dB	-68.6dB	-68.8dB	-71dB	-70dB

Tabla 4.5. Valores lineales de los armónicos para diferentes frecuencias de la señal medidos con el osciloscopio Tektronix modelo TDS 2024.

Frecuencia / **Armónico**	**100Hz Volts**	**4kHz Volts**	**6kHz Volts**	**100kHz Volts**	**600kHz Volts**	**1MHz Volts**
Fundamental	0.2818	0.2818	0.2818	0.2818	0.2818	0.2238
Segundo	5.62×10^{-4}	5.62×10^{-4}	4.46×10^{-4}	9.33×10^{-4}	6.30×10^{-4}	1.41×10^{-3}
Tercero	3.16×10^{-4}	5.62×10^{-4}	8.91×10^{-4}	5.12×10^{-4}	5.62×10^{-4}	4.46×10^{-4}
Cuarto	2.78×10^{-4}	5.62×10^{-4}	3.54×10^{-4}	4.07×10^{-4}	3.54×10^{-4}	3.16×10^{-4}
Quinto	2.51×10^{-4}	5.62×10^{-4}	3.75×10^{-4}	4.07×10^{-4}	4.46×10^{-4}	10.23×10^{-4}
Sexto	2.23×10^{-4}	5.62×10^{-4}	3.71×10^{-4}	3.63×10^{-4}	2.81×10^{-4}	3.16×10^{-6}
THD	0.0028	0.0048	0.0042	0.0044	0.0038	.0082

La siguiente fórmula nos proporciona el voltaje en decibeles de un valor, en este caso, del voltaje.

$$GdB = 20\log(G_{lineal}) \tag{4.1}$$

Debido a que el osciloscopio nos proporciona la magnitud de los armónicos en dB, en la tabla 4.4 se observan los valores en decibles de los armónicos de la señal, es necesario convertirlos a lineales para poder sacar la distorsión armónica total (THD), y despejando la fórmula anterior obtenemos que:

$$G_{lineal} = 10^{\frac{GdB}{20}} \tag{4.2}$$

Y así obtenemos los valores lineales de la señal, vistos ya convertidos de la tabla 4.4 en la tabla 4.5, y conociendo los valores lineales podemos sacar la THD con la fórmula siguiente:

$$THD = \frac{\sqrt{V_2 + V_3 + V_4 + V_5 + V_6}}{V_1} \tag{4.3}$$

Donde V_1 es el valor lineal del primer armónico o fundamental de la señal, y V_2 hasta V_6 son los valores lineales del segundo armónico hasta el sexto armónico, el valor lineal en este caso es en voltaje. En la figura 4.10 y 4.11 se muestran los espectros de frecuencias de las señales generadas por el DDS a 100Hz y 4kHz respectivamente.

Primero vamos a comparar los resultados de estas mediciones hechas por el osciloscopio entre los resultados medidos por el analizador de espectro. Midiendo la señal en 4kHz y de 600kHz con el analizador de espectro tenemos una THD de 0.045 en la frecuencia de 4kHz y una THD de 0.001496 en la frecuencia de 600kHz lo que quiere decir la medición es totalmente diferente a la medida en el osciloscopio y el analizador de espectro es más exacta en frecuencias mayores o cercanas a su

rango, en la medición en 4kHz es totalmente errónea y esto es por que el analizador de espectros apenas detecta la señal de 4kHz, mientras que haciendo una comparación con la hoja de datos del DDS AD9833 en la frecuencia de 6kHz la THD es de 0.00066 que haciendo una relación es mucho más pequeña a la medida de la THD en la frecuencia de 4kHz con el osciloscopio y mucho más a la medida en el analizador de espectro, en cuanto a la frecuencia de 600kHz la THD por el fabricante es de 0.0011, la medida por el nosotros con el osciloscopio nos proporciona una THD de 0.0038 y la medida con el analizador de espectro nos da una THD de 0.001496, que se acerca aun más la del analizador de espectro, esto es porque el analizador de espectro si detecta bien señales de frecuencias muy altas y a pesar de que 600kHz no está en su rango, el valor de THD calculado se acerca bastante al teórico, hay que tomar en cuenta muchos factores, estos son algunos factores que considero:

- El fabricante no proporciona la THD como tal, sino que se obtiene de las gráficas espectrales que proporciona el fabricante, por lo que el cálculo tiene un error humano debido a que la visión del mismo no es muy exacta y depende mucho del tanteo del mismo.
- El fabricante debido a que dedica mucho tiempo y utiliza componentes tanto de medición como de fabricación de muy alta tecnología, mucho más avanzada que con la que contamos en el laboratorio de electrónica de nuestra institución, es por eso que el análisis de su producto es mucho más exacto.
- Debido a que el osciloscopio requiere de análisis matemáticos muy avanzados para proporcionar la FFT a frecuencias altas es probable que el osciloscopio no proporcione un análisis correcto del mismo, ya que el analizador de espectros es una herramienta especializada para el análisis espectral de señal, el único inconveniente es que tiene un rango de trabajo el cual fuera de la medición puede ser errónea, como lo vemos en frecuencias muy pequeñas como la de 4kHz, de hecho la frecuencia de 100Hz el analizador de espectro no la detecta.
- El fabricante analiza sus componentes con un área de trabajo de primera calidad, haciendo las conexiones apropiadas para que el AD9833 trabaje al

máximo, como lo es el alambrado de la conexión proporcionando un alambrado corto a tierra, entre otros, disminuyendo el efecto capacitivo en el alambrado y que puede afectar la generación de señal del DDS AD9833.

También se puede observar que en cuanto más grande es la frecuencia del DDS su distorsión armónica total va aumentado, esto debido a la naturaleza del mismo DDS AD9833, por ejemplo, la no linealidad del DAC del mismo, la eficiencia del filtro pasa-bajas con el que cuenta el DDS, entre otros.

En cuanto a la comparación con la versión anterior [11, 12] se obtuvieron los siguientes resultados: para una frecuencia de 100Hz se obtuvo una THD de 0.0091 generada por la técnica DDS, que implementaron con la ayuda de un microcontrolador y un DAC mientras que para la frecuencia de 4kHz se obtuvo una THD de 0.008 utilizando un generador de señales que se controla por medio de un potenciómetro digital. En la tabla 4.6 se muestra los resultados de las dos versiones, de la actual y la anterior.

Tabla 4.6. Comparación de la THD entre la versión anterior y la actual.

	Frecuencia	**THD**
Versión actual	100Hz	0.0028
	4kHz	0.0048
Versión anterior	100Hz	0.0091
	4kHz	0.008

Como se puede observar en la tabla 4.6 eso sin considerar los factores que ya mencionamos antes, si no considerando los valores medidos por nosotros, la calidad de la señal es mucho mayor en la generación de la señal a través del DDS del AD9833 que con la generación de la señal en la versión anterior [11, 12].

4.5.3 Resumen de las pruebas del DDS AD9833

En resumen hay muchas razones por la cuál se puede tener un análisis erróneo, o mediciones erróneas en cuanto a la calidad de la señal, y no sólo considerada por

ejemplo en el análisis de filtros electrónicos, por la calidad del equipo que se emplea no sólo en la medición, si no en la creación y diseño del componente que hace la tarea de la medición, ya sea de una señal eléctrica o cualquier otro tipo de información, y es imposible que todos los análisis inclusive si se analizan con el mismo equipo y en la misma área de trabajo se tengan los mismos resultados.

4.6 Pruebas al sistema con filtros electrónicos reales

Para verificar que el sistema funcione correctamente o se acerque a la respuesta matemática de un filtro, se realizaron varias pruebas con diferentes tipos de filtros, y comparando los datos de su respuesta en frecuencia con simulaciones hechas con la ayuda del programa Multisim 9, no se decidió hacer su comparación de manera práctica con la ayuda del osciloscopio por que este error sería más humano y más las diferencias en los valores de los componentes del filtro, si los tuviese, sería más grande el error y que más exactitud que la lectura hecha de un ADC por medio de un microcontrolador.

4.6.1 Filtro pasa-bajas RC de primer orden

Los filtros pasivos son filtros electrónicos formados únicamente por elementos resistivos, capacitivos e inductivos, elementos que no requieren de una alimentación externa para que funcionen, este tipo de filtros son de configuraciones muy sencillas y muy fácil de implementar ya que no ocupan mucho espacio, si es un filtro capacitivo y resistivo, como lo es en este caso, ya que las bobinas la mayoría de los casos suelen tener volúmenes muy grandes a frecuencias muy bajas. La figura 4.12 muestra la configuración del filtro pasa-bajas RC de primer orden con una frecuencia de corte cercana a 300Hz, digo cercana, por que los valores de resistencias y de condensadores que fueron medidas no son las exactas para la frecuencia de corte e 300Hz. Su respuesta en frecuencia de su simulación se muestra en la figura 4.13, en la figura 4.14 se muestra la gráfica de su respuesta en frecuencia del nuestro sistema y en la tabla 4.7 su comparación en resultados.

.

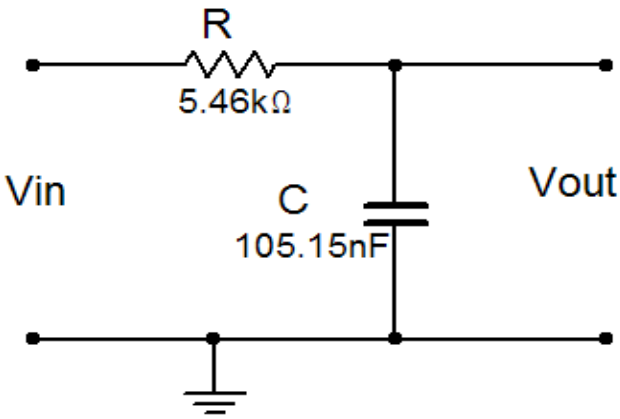


Figura 4.12. Filtro pasivo pasa-bajas RC de primer orden.

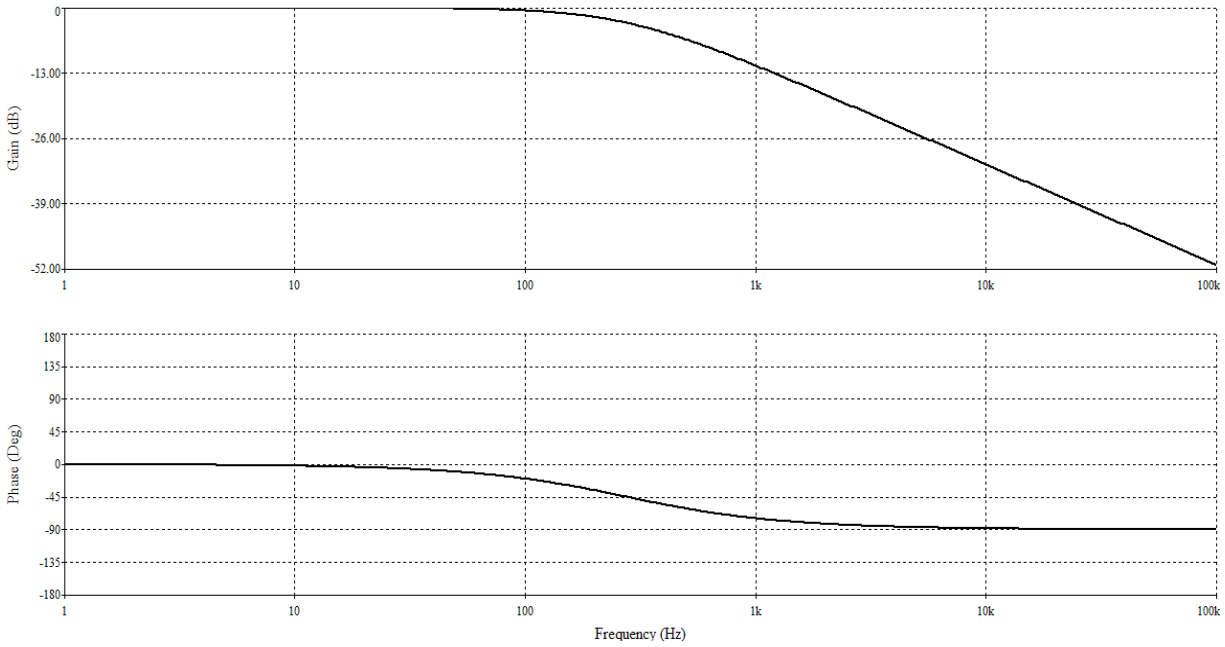


Figura 4.13. Respuesta en frecuencia de la simulación del filtro pasivo pasa-bajas RC de primer orden.

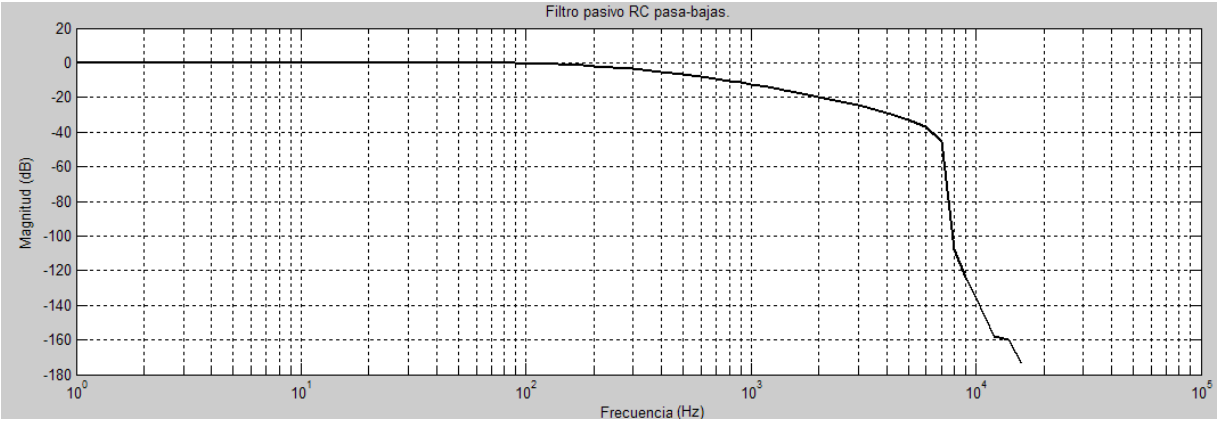


Figura 4.14. Respuesta en frecuencia en magnitud del filtro pasivo RC pasa-bajas generada por el Bode Plotter.

Tabla 4.7. Magnitud de ganancia del filtro pasivo RC pasa-bajas.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la banda de paso			
10	0.056	0.115	105.35
100	-0.531	-0.4532	14.65
200	-1.821	-1.804	0.93
Frecuencias en la región de corte			
250	-2.586	-2.713	4.91
300	-3.3691	-3.457	2.60
400	-4.90	-5.075	3.57
Frecuencias en la pendiente de atenuación			
600	-7.55	-7.965	5.49
1500	-14.81	-16.34	10.33
6000	-27.72	-37.13	33.94
20000	-37.16	-180	X
30000	-40.09	-180	X

Como se puede observar en la figura 4.13 y 4.14 el sistema va fallando entre la frecuencias de 6kHz a 7kHz de la misma manera pasaba en la versión anterior [11, 12] es por eso que en la tabla 4.7 hay que tomar en cuenta que la X en la tabla son valores que no se toman en cuenta para sacar el valor promedio debido a que esos valores pudieron haberse generados por diferentes factores, como lo es el ruido, que el componente no funcione correctamente, etc. pero al contrario de esta versión no contaba con los amplificadores PGA es por eso que no se tenía esa pendiente tan

grande, esto puede ser debido a que el capacitor se comporta de una manera diferente a la esperada en frecuencias relativamente altas para un condensador tan pequeño, o puede ser por otros factores, como, que el condensador esté dañado o desgastado, etc.

4.6.2 Filtro pasa-bajas *Butterworth* de primer orden

Este filtro corresponde a los filtros activos, la figura 4.15 muestra el circuito del filtro *Butterworth*, este filtro fue diseñado para una frecuencia de corte de 1kHz. Para este filtro se utilizó un amplificador operacional LM741 de los que se utilizan más comúnmente en el laboratorio del Instituto Tecnológico de Sonora, es uno de los amplificadores que tienen un OFFSET considerablemente elevado en comparación a otros amplificadores operacionales. La gráfica de la simulación y la gráfica de la respuesta en frecuencia obtenida por nuestro sistema se muestran en las figuras 4.16 y 4.17, respectivamente. Y en la tabla 4.8 se muestran las mediciones de la magnitud del filtro pasa-bajas *Buttherworth* de primer orden.

Tabla 4.8. Magnitud de ganancia del filtro activo pasa-bajas *Butterworth* de primer orden.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la banda de paso			
50	-0.009	0	0.9
500	-0.856	-0.8326	2.73
800	-1.926	-1.947	1.09
Frecuencias en la región de corte			
900	-2.321	-2.284	1.59
1000	-2.7243	-2.737	.466
1100	-3.129	-3.171	1.34
Frecuencias en la pendiente de atenuación			
1500	-4.717	-4.737	0.42
3000	-9.47	-9.464	0.063
8000	-17.546	-17.32	1.28
30000	-28.95	-28.26	2.38
60000	-34.97	-34.04	2.65
100000	-39.42	-38.28	2.89

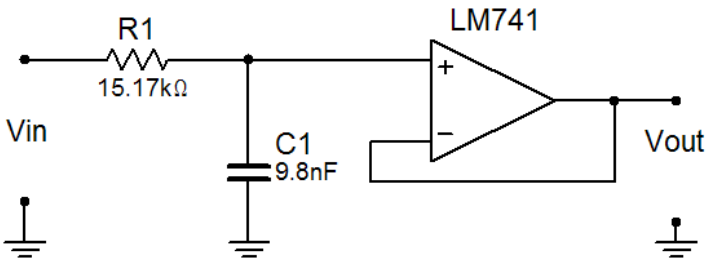


Figura 4.15. Filtro activo pasa-bajas *Butterworth* de primer orden.

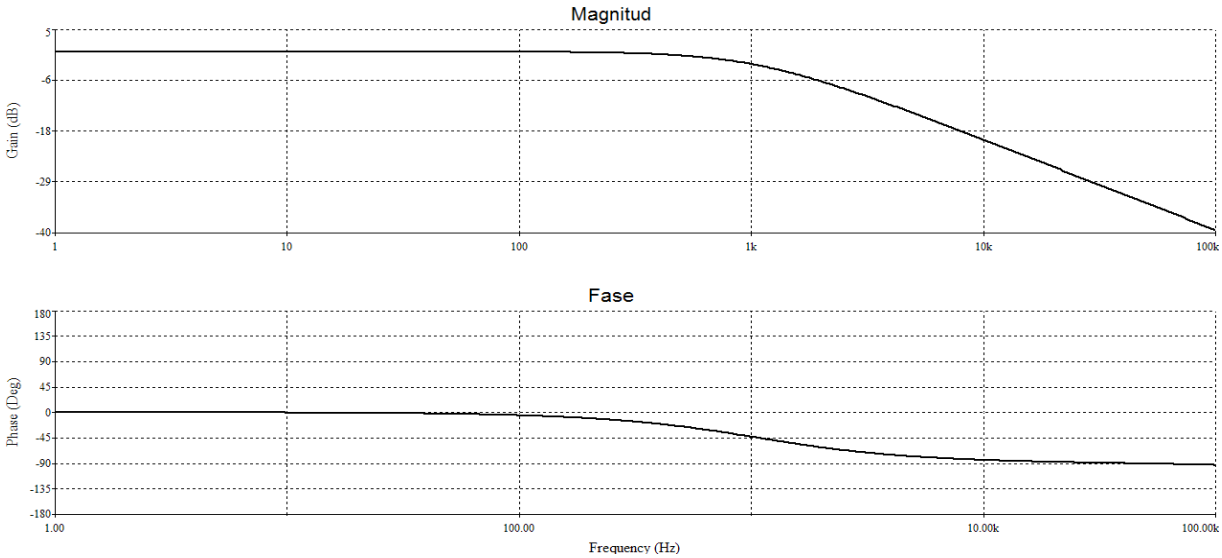


Figura 4.16. Simulación de la respuesta en frecuencia del filtro activo pasa-bajas *Butterworth*.

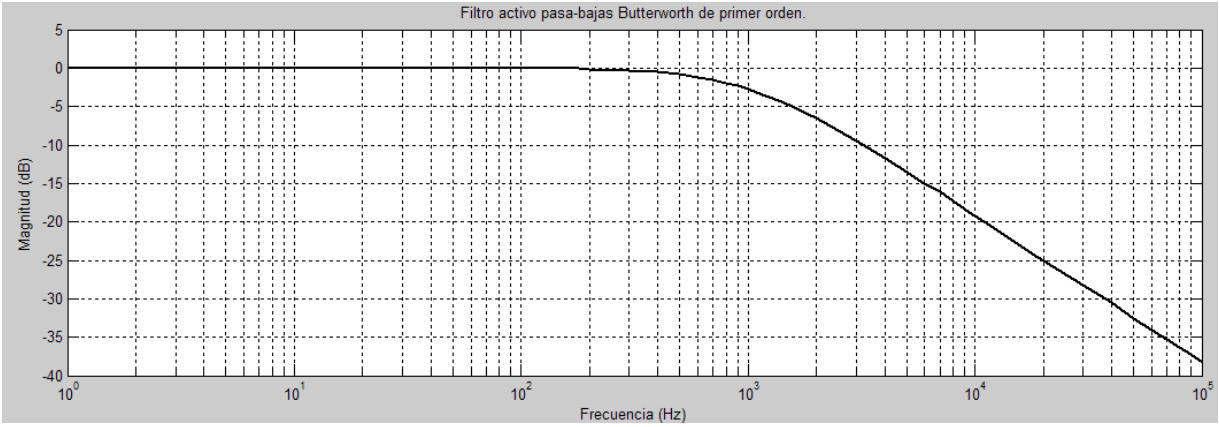


Figura 4.17. Respuesta en frecuencia de la magnitud del filtro activo pasa-bajas *Butterworth* generada por el Bode Plotter.

4.6.3 Filtro pasa-bajas *Butterworth* de cuarto orden

Los filtros Butterworth se caracterizan por tener una respuesta muy plana en el origen y eliminan mucho el ruido presente en los filtros pasivos, y elimina mucho más el ruido si se utilizan los amplificadores operacionales JFET y que tienen menos OFFSET que los LM741, este filtro fue diseñado para una frecuencia de corte de 4000Hz, en la figura 4.18 se muestra el diseño del filtro electrónico y en la figura 4.19 y 4.20 se muestras sus gráficas de su respuesta en frecuencia, la de simulación y la del Bode Plotter, respectivamente. En la tabla 4.9 se muestra las mediciones de la magnitud del filtro a diferentes frecuencias, tanto en la simulación como en el trazador.

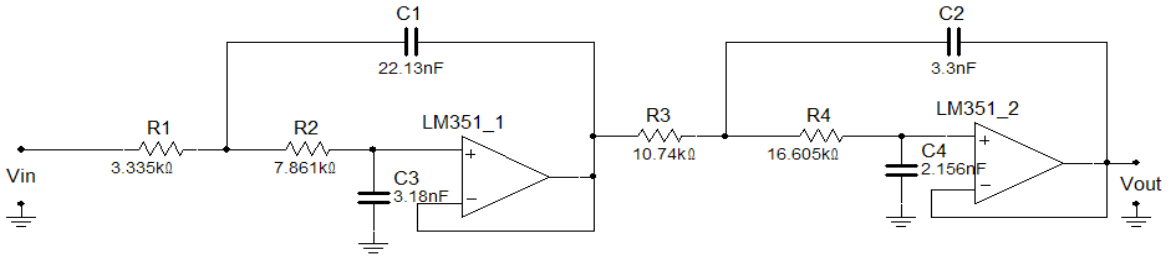


Figura 4.18. Filtro pasa-bajas *Butterworth* de cuarto orden.

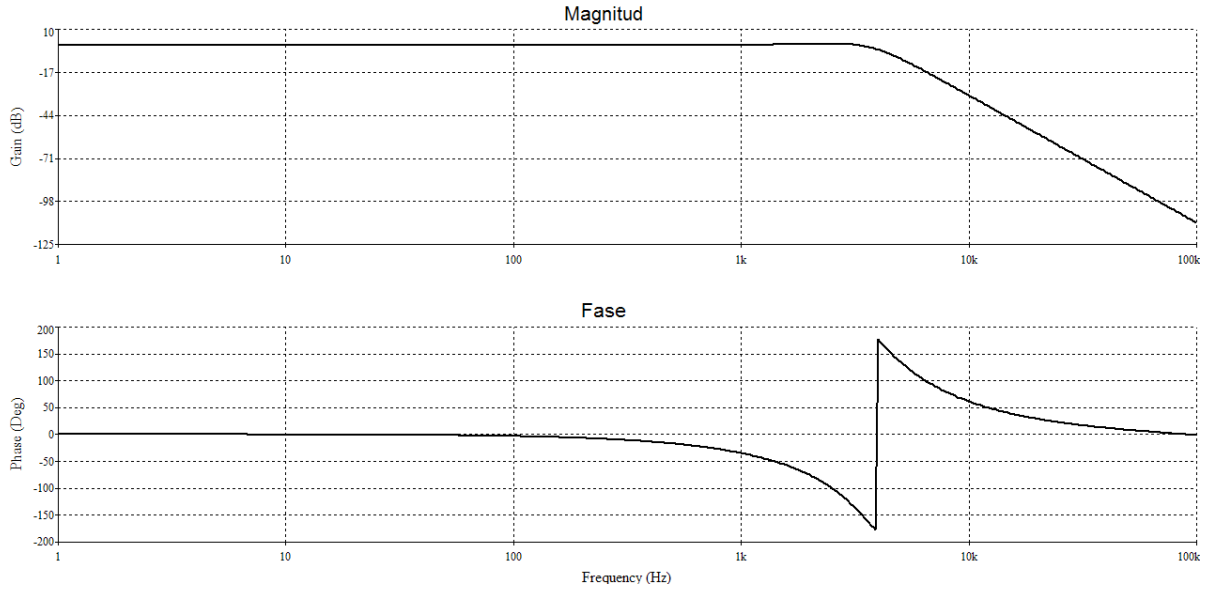


Figura 4.19. Simulación de su respuesta en frecuencia del filtro pasa-bajas *Butterworth* de cuarto orden.

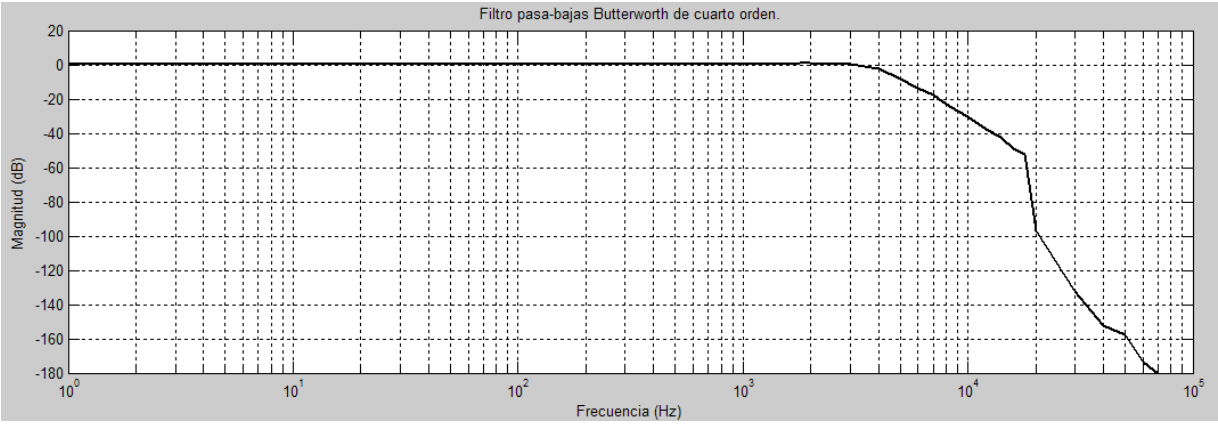


Figura 4.20. Respuesta en frecuencia de la magnitud del filtro pasa-bajas *Butterworth* de cuarto orden generada por el Bode Plotter.

Tabla 4.9. Magnitud de ganancia del filtro activo pasa-bajas *Butterworth* de cuarto orden.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencia en la banda de paso			
100	0.0	0.08	8
2000	0.799	0.795	0.500
3600	-0.926	-1.462	57.88
Frecuencias en la región de corte			
3900	-2.197	-2.332	6.14
4000	-2.681	-2.609	2.68
4200	-3.718	-3.792	1.99
Frecuencias en la banda de atenuación			
4500	-5.392	-5.463	1.61
5000	-8.31	-8.019	3.50
8000	-23.608	-22.68	3.93
13000	-40.38	-39.88	1.23
15000	-45.356	-45.68	0.71
16000	-47.6	-49.12	3.19
17000	-49.708	-50.92	2.43

4.6.4 Filtro pasa-bajas *Chebyshev* de segundo orden

Este filtro se implementó debido a que se su comportamiento en la respuesta en frecuencia del mismo genera un rizo antes de la frecuencia de corte, este rizo tiene un valor de 3dB y el mismo filtro tiene una frecuencia de corte de 1kHz, en la figura

4.21 se muestra el circuito del filtro el cuál utiliza amplificadores JFET que son los LM351, esto para observar mejor respuesta en el análisis. La figura 4.22 y 4.23 muestra su respuesta en frecuencia del su simulación y de su análisis respectivamente.

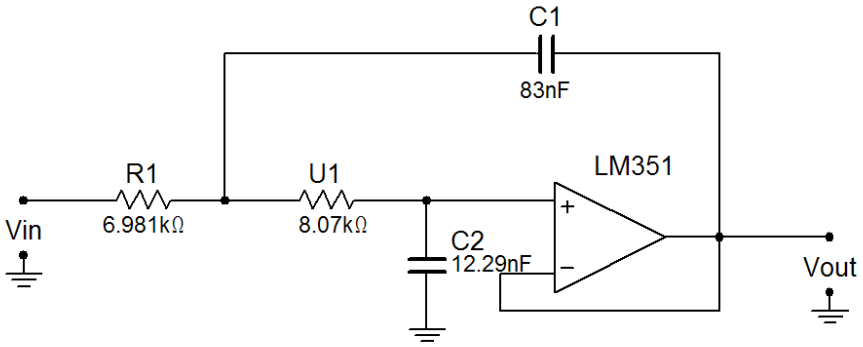


Figura 4.21. Filtro pasa-bajas *Chebyshev* de segundo orden.

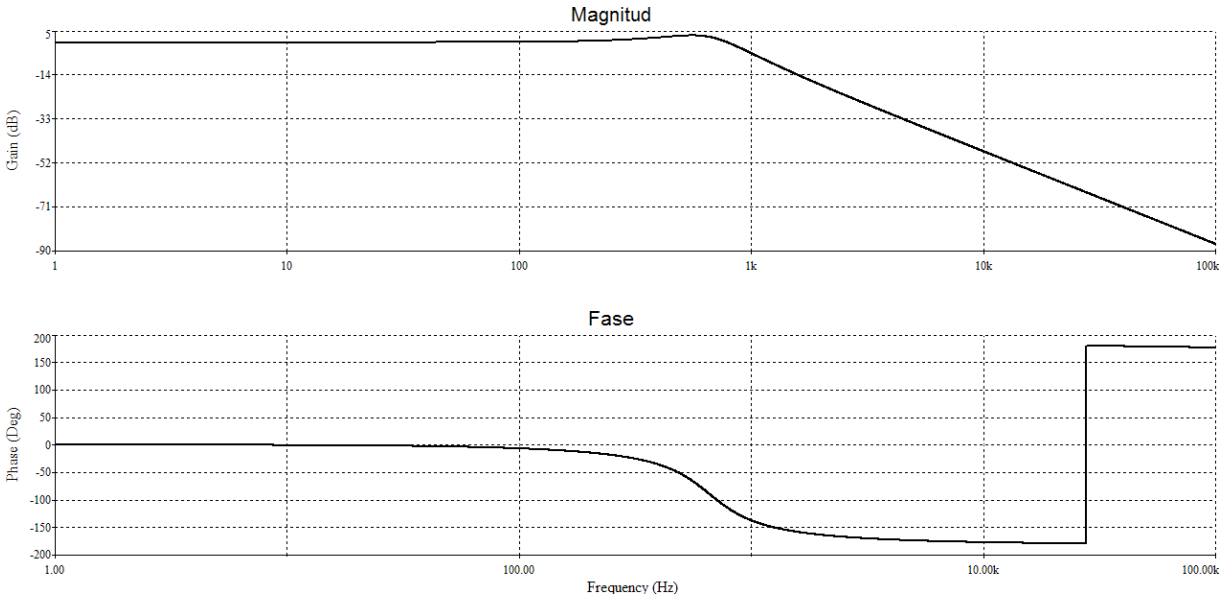


Figura 4.22. Simulación de la respuesta en frecuencia del filtro *Chebyshev* de segundo orden.

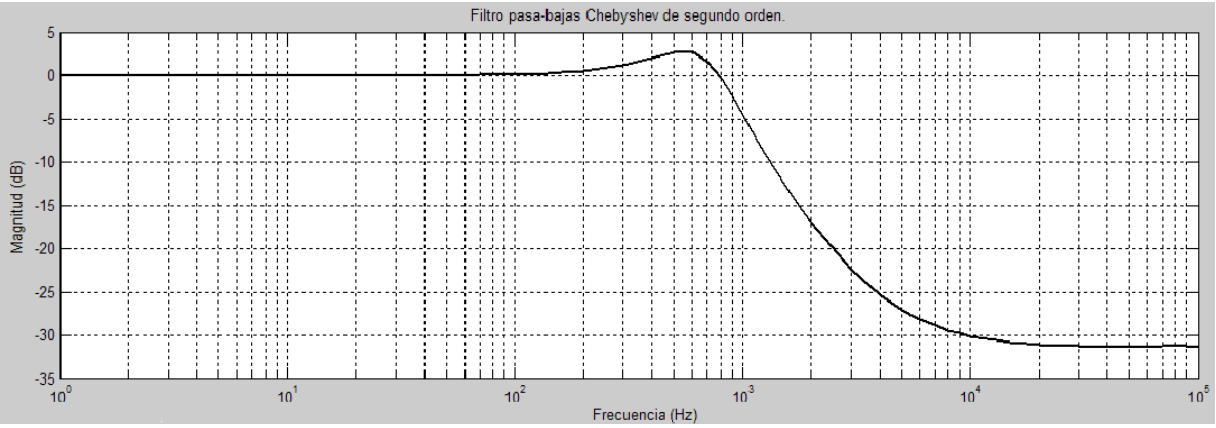


Figura 4.23. Respuesta en frecuencia de la magnitud del filtro *Chebyshev* de segundo orden generada por el Bode Plotter.

Tabla 4.10. Magnitud de ganancia del filtro pasa-bajas *Chebyshev* de segundo orden.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la banda de paso			
5	0	0	0
90	0.142	0.117	17.6
140	0.271	0.298	9.06
190	0.498	0.482	3.21
200	0.552	0.531	3.80
300	1.223	1.217	0.49
Frecuencias en la región de corte			
500	2.802	2.72	2.92
600	2.843	2.735	3.79
800	.294	.295	0.34
1000	-4.722	-4.532	4.02
Frecuencias en la banda de atenuación			
1300	-10.45	-9.628	7.86
1800	-16.51	-15.27	7.51
3000	-25.91	-22.36	13.7
4000	-31.02	-25.28	18.56
5000	-34.97	-27.04	22.67

4.6.5 Filtro pasa-altas *Butterworth* de cuarto orden

Este filtro se diseñó con una frecuencia de corte de 1kHz, el objetivo de este filtro es para comprobar el funcionamiento del algoritmo cuando la pendiente de atenuación se encuentra en frecuencias bajas, así verificando si el algoritmo de los amplificadores PGA funciona correctamente. En la figura 4.24 se muestra el circuito del filtro pasa-altas *Butterworth* de cuarto orden. En la figura 4.25 se muestra la respuesta en frecuencia de su simulación y en la figura 4.26 se muestra la respuesta en frecuencia en su magnitud del filtro pasa-altas *Butterworth* generada por nuestro trazador, así como su magnitud en diferentes frecuencias mostradas en la tabla 4.11.

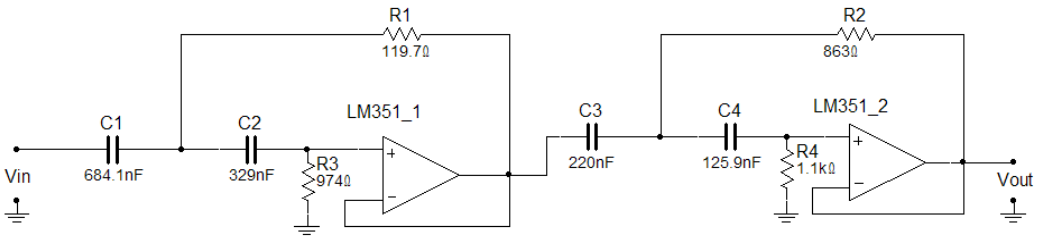


Figura 4.24. Filtro pasa-altas *Butterworth* de cuarto orden.

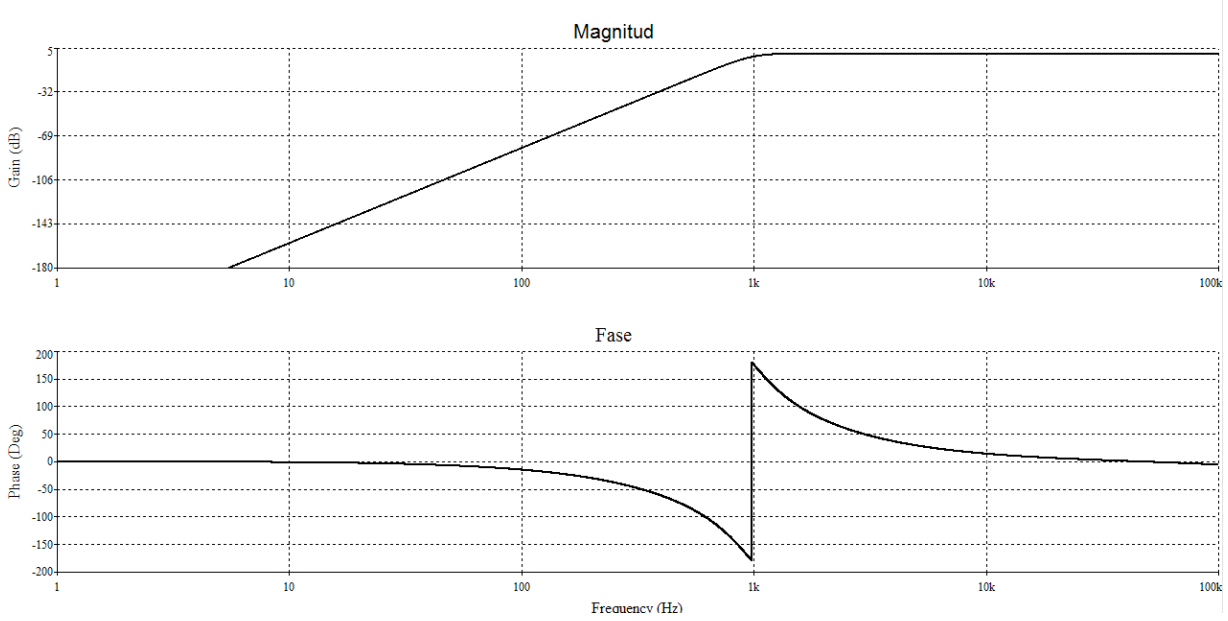


Figura 4.25. Simulación de la respuesta en frecuencia del filtro pasa-altas *Butterworth* de cuarto orden.

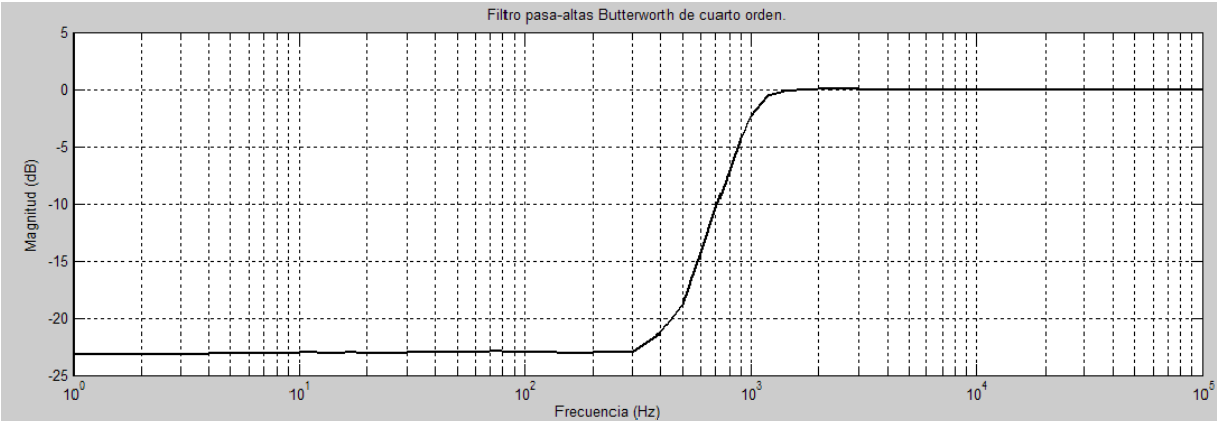


Figura 4.26. Respuesta en frecuencia de la magnitud del filtro pasa-altas *Butterworth* de cuarto orden generada por el Bode Plotter.

Tabla 4.11. Magnitud de ganancia del filtro pasa-altas *Butterworth* de cuarto orden.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la pendiente de atenuación			
300	-41.17	-22.99	44.15
450	-27.07	-19.94	26.33
500	-23.40	-18.78	19.74
800	-7.711	-7.199	6.63
Frecuencias en la región de corte			
900	-4.56	-4.24	7.01
1000	-2.482	-2.363	4.79
1200	-.619	-.589	4.84
Frecuencias en la banda de paso			
2000	.029	.057	96.55
5000	-0.006	-0.02	
15000	-0.015	-0.0225	50
50000	-0.025	-0.044	76

4.6.6 Filtro rechaza-banda Twen-Tee

Diseñado a partir de un BJT LM741 con una frecuencia de corte de 1kHz, este filtro se utilizó para comprobar el funcionamiento del Bode Plotter por que es una manera fácil de crear una respuesta en frecuencia en magnitud con un comportamiento de rechaza-banda con un solo amplificador operacional y pocos componentes pasivos. En el figura 4.27 se muestra el circuito del filtro rechaza-banda Twen-Tee, así como

en la figura 4.28 y 4.29 su respuesta en frecuencia simulada y generada por el Bode Plotter, y claro en la tabla 4.12 su magnitud en diferentes frecuencias.

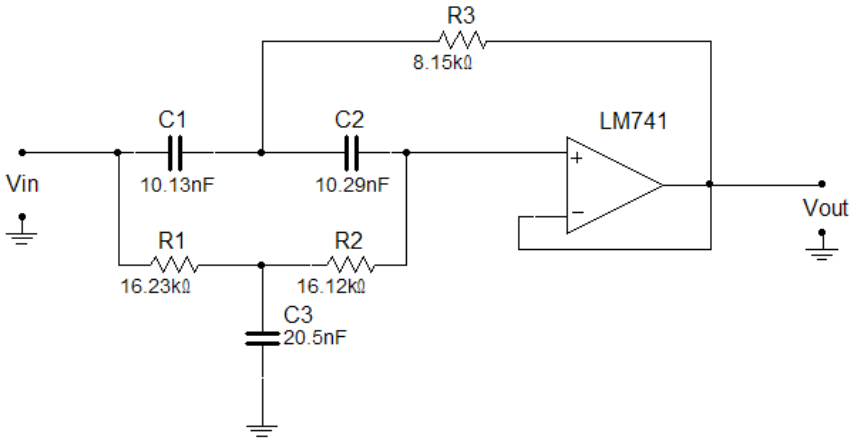


Figura 4.27. Filtro rechaza-banda Twen-Tee con frecuencia de corte de 1kHz.

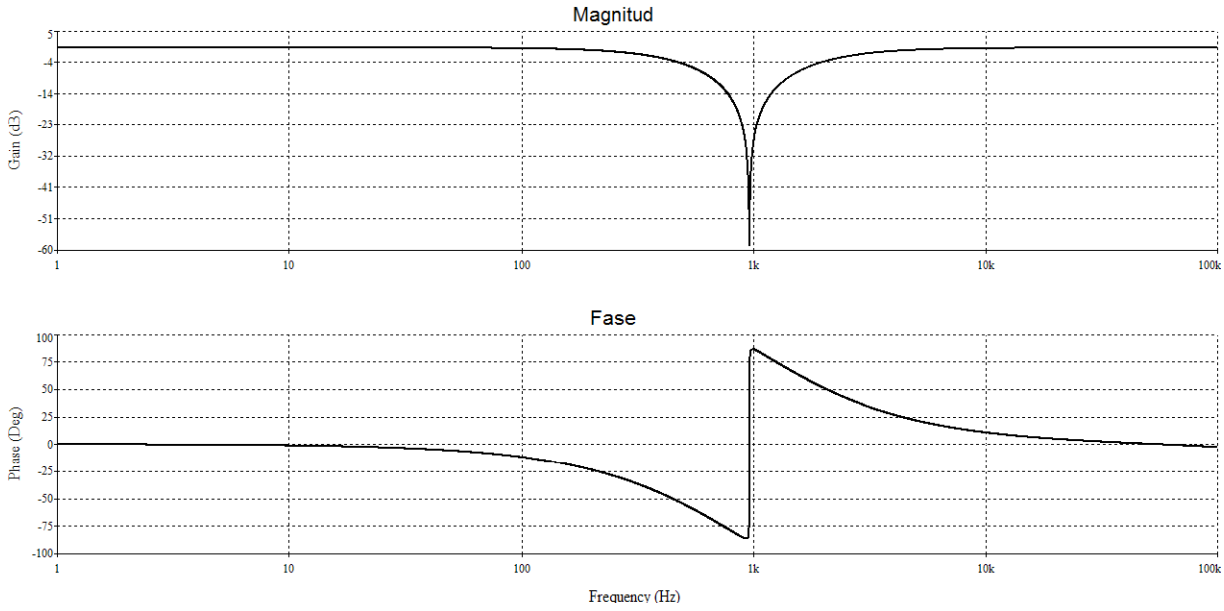


Figura 4.28. Simulación de la respuesta en frecuencia del filtro rechaza-banda Twen-Tee.

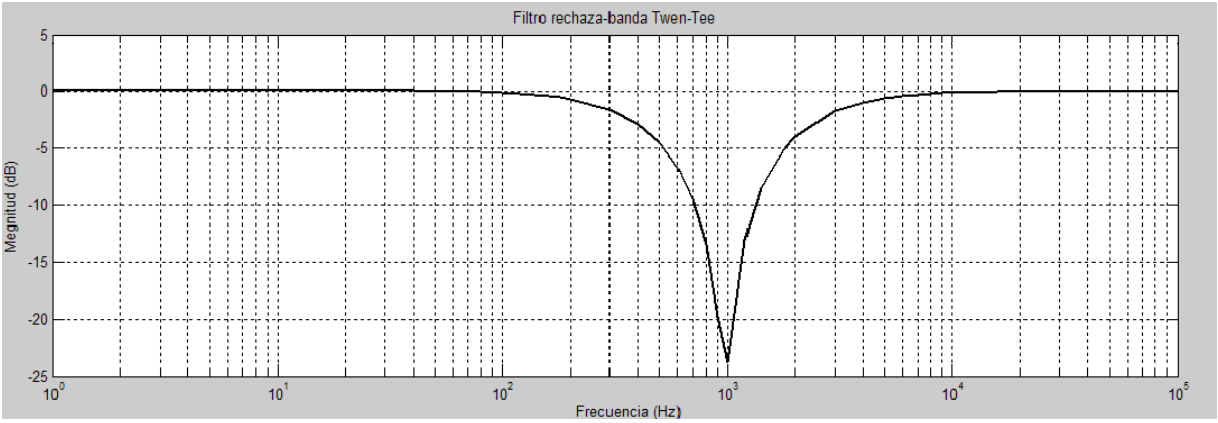


Figura 4.29. Respuesta en frecuencia de la magnitud del filtro rechaza-banda Twen-Tee generada por el Bode Plotter.

Tabla 4.12. Magnitud de ganancia del filtro rechaza-banda Twen-Tee.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la banda de paso baja			
5	0	0.0132	1.32
100	-0.189	-0.1604	15.13
300	-1.711	-1.602	6.37
Frecuencias en la banda de rechazo			
400	-3.061	-2.915	4.76
500	-4.852	-4.584	5.52
800	-14.88	-13.48	9.4
1000	-28.04	-23.84	0.53
1200	-13.265	-13.05	1.62
2000	-4.129	-4.082	1.13
2500	-2.624	-2.784	6.09
Frecuencias en la banda de paso alta			
3000	-1.8201	-1.795	1.37
15000	-0.174	-0.180	3.44
50000	-0.022	-0.026	18.18

En la figura 4.30 se muestra la respuesta en frecuencia del mismo filtro, sólo que esta vez se analizó en el rango configurable de frecuencias con un análisis en la frecuencia inicial de 100Hz hasta 10kHz con una resolución de 100Hz, se puede observar en la tabla 4.13 que el resultado es casi lo mismo a comparación de la tabla 4.12. No es un acercamiento a las frecuencias de 100Hz a 10kHz de la gráfica original, si no que es un análisis nuevo generado de manera configurable.

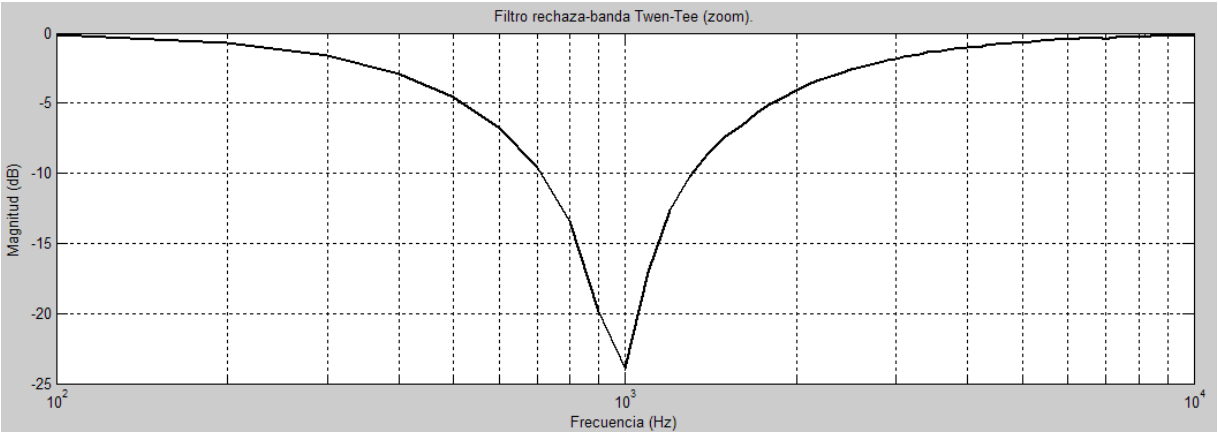


Figura 4.30. Respuesta en frecuencia de la magnitud del filtro Twen-Tee con un análisis configurado en 100Hz hasta 10kHz con una resolución de 100Hz.

Tabla 4.13. Magnitud de ganancia del filtro rechaza-banda Twen-Tee en un análisis configurado de 100Hz hasta 10kHz con una resolución de 100Hz

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]	Error %
Frecuencias en la banda de paso baja			
100	-0.189	-0.1604	15.13
300	-1.711	-1.642	4.03
Frecuencias en la banda de rechazo			
400	-3.061	-2.915	4.76
500	-4.852	-4.584	5.52
800	-14.88	-13.45	9.61
1000	-28.04	-23.85	14.94
1200	-13.265	-12.89	2.82
2000	-4.129	-4.093	0.87
2500	-2.624	-2.614	0.38
Frecuencias en la banda de paso alta			
3000	-1.8201	-1.828	0.43
15000	-0.174	-0.167	4.02

.

4.6.7 Filtro pasa-banda RLC para un análisis configurado

Este filtro fue creado para tres propósitos, el primero es que tan bien trabaja el algoritmo de los amplificadores PGA, se que ya creamos un filtro para ese propósito pero que más da uno más para ver si no tiene ningún fallo, y el segundo propósito es para analizar filtro de manera configurable y por tercer propósito ver si podemos

extender el análisis de los filtros electrónicos más allá de 100kHz y ver como funciona este tipo de análisis, en la figura 4.31 observamos el circuito del filtro y en la figura 4.32 su simulación con su respuesta en frecuencia. Esta prueba se realizó bajo el análisis con una frecuencia inicial de 1Hz, una frecuencia final de 1MHz y una resolución de 5kHz. Este filtro fue diseñado para una frecuencia de corte de 100kHz.

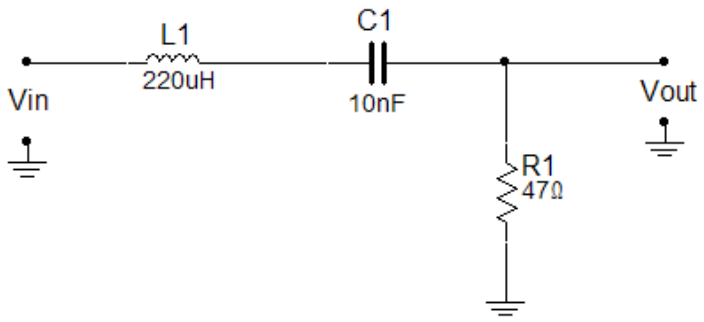


Figura 4.31. Filtro pasa-banda RLC.

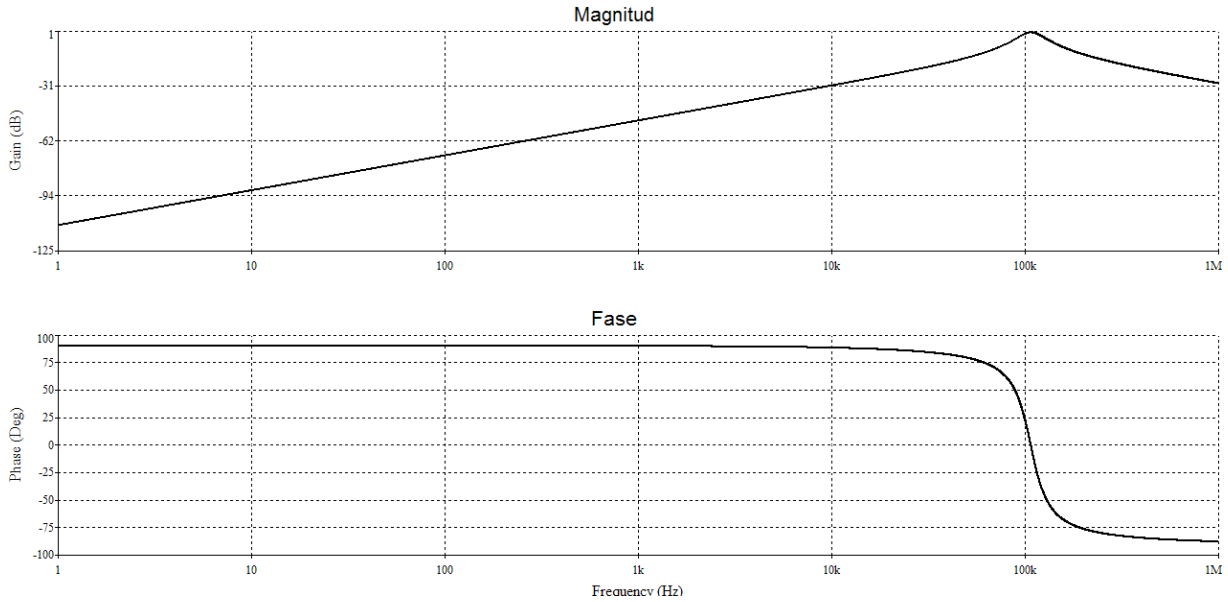


Figura 4.32. Simulación de la respuesta en frecuencia del filtro RLC.

En la figura 4.33 se muestra la respuesta en frecuencia de la magnitud del filtro pasabanda RLC generada por nuestro sistema y en la tabla 4.14 se muestran los valores de la magnitud de ganancia para el mismo filtro RLC.

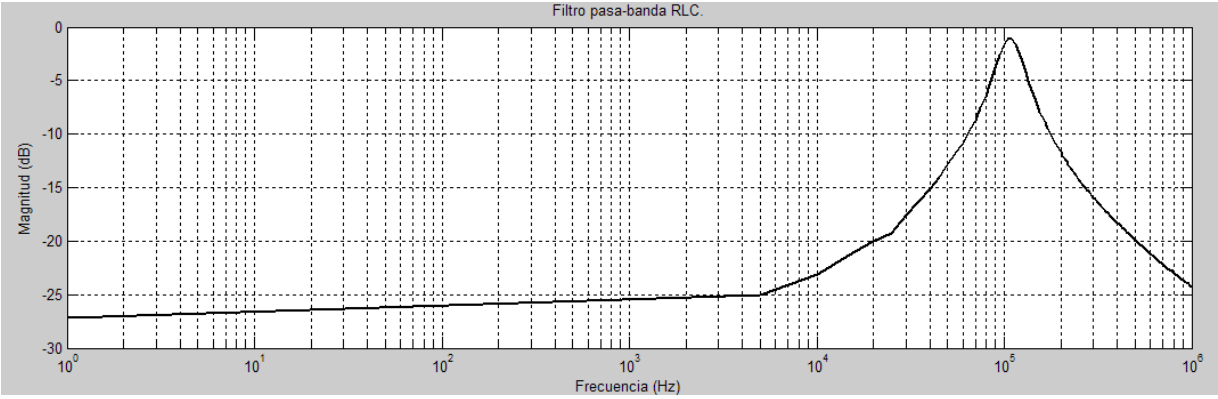


Figura 4.33. Respuesta en frecuencia de la magnitud del filtro pasa-banda RLC generada por el Bode Plotter.

Tabla 4.14. Magnitud de ganancia de la respuesta en frecuencia del filtro RLC pasa-banda.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]
Frecuencias en la banda de atenuación baja		
5000	-36.59	-25.07
20000	-24.283	-19.98
30000	-20.385	-17.52
40000	-17.33	-15.11
Frecuencias en la banda de paso		
50000	-14.64	-12.89
60000	-12.05	-10.84
70000	-9.406	-8.692
90000	-3.511	-3.792
100000	-0.785	-1.655
150000	-7.512	-7.294
200000	-12.68	-11.74
Frecuencias en la banda de atenuación alta		
500000	-22.96	-19.88
600000	-24.66	-21.3
1000000	-29.27	-24.33

4.6.8 Filtro rechaza-banda RLC para un análisis configurado

Este análisis tiene los mismos propósitos que el análisis pasado, la figura 4.34 muestra el circuito el cuál fue diseñado para una frecuencia de corte de 100kHz, mientras la figura 4.35 muestra la simulación de su respuesta en frecuencia, así

como las figuras 4.36 muestra su respuesta en frecuencia de la magnitud generada por nuestro sistema y su respectiva tabla mostrada en la tabla 4.15. El análisis para este filtro cuenta con la misma configuración que el análisis pasado, que tiene como frecuencia inicial de 1Hz, frecuencia final de 1MHz y con una resolución de 5kHz.

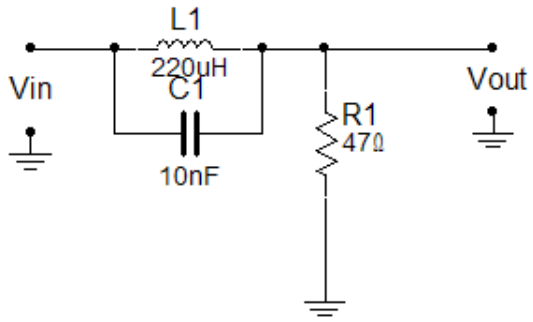


Figura 4.34. Filtro rechaza-banda RLC.

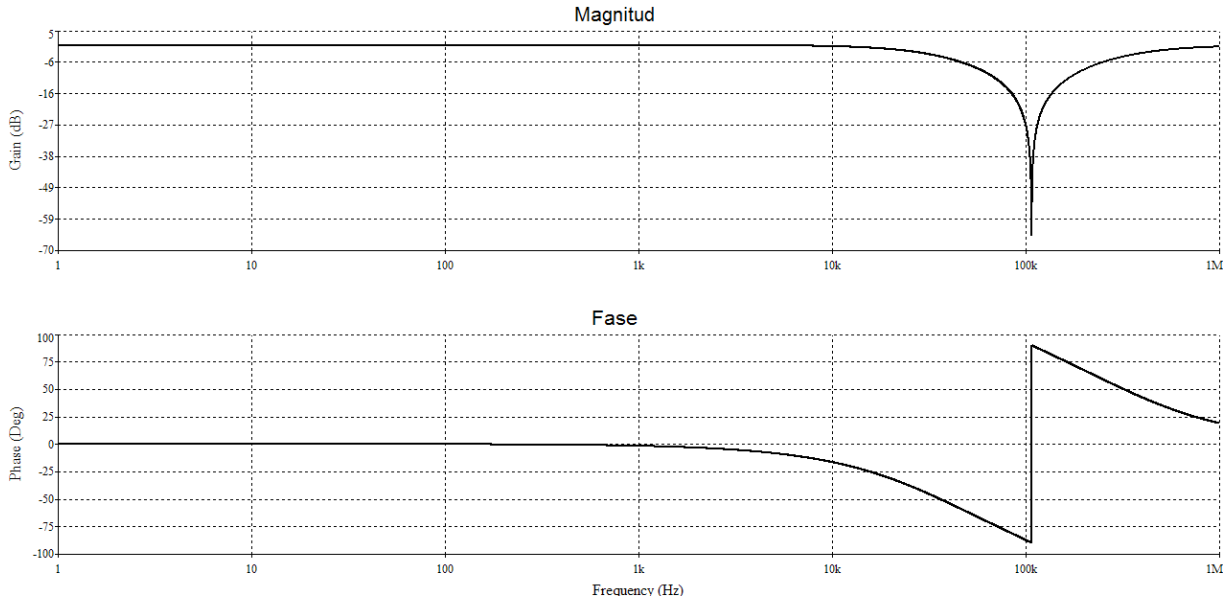


Figura 4.35. Simulación de la respuesta en frecuencia del filtro rechaza-banda RLC.

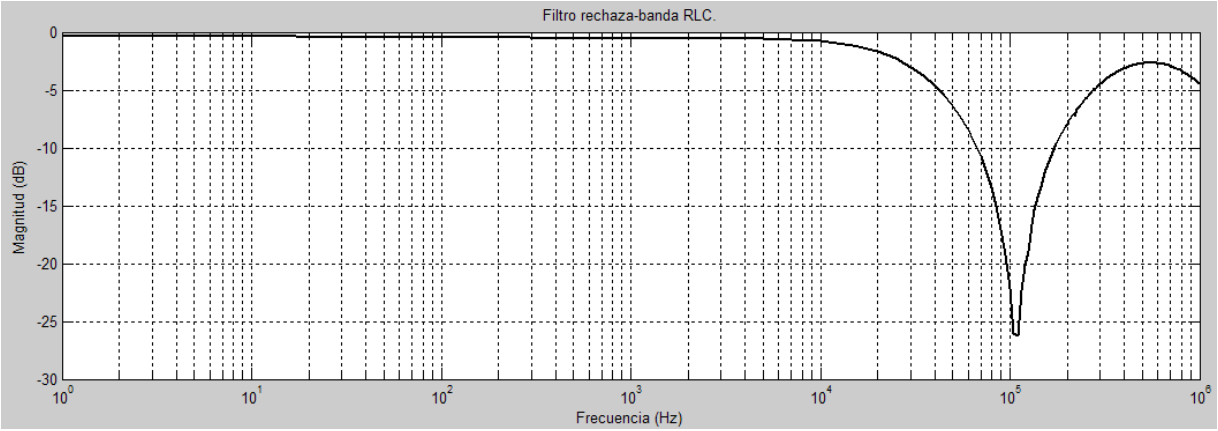


Figura 4.36. Respuesta en frecuencia de la magnitud del filtro rechaza-banda RLC generada por el Bode Plotter.

Tabla 4.15. Magnitud de ganancia de la respuesta en frecuencia del filtro rechaza-banda RLC.

Frecuencia (Hz)	Ganancia (dB) [Simulación]	Ganancia (dB) [Trazador]
Frecuencias en la banda de paso baja		
5000	-0.093	-0.503
20000	-1.371	-1.69
30000	-2.824	-3.088
40000	-4.573	-4.606
Frecuencias en la banda de rechazo		
50000	-6.556	-6.386
60000	-8.802	-8.456
70000	-11.413	-10.73
90000	-19.06	-17.01
100000	-27.001	-22.18
150000	-13.498	-12.54
200000	-8.2298	-7.903
Frecuencias en la banda de paso alta		
500000	-1.7726	-2.713
600000	-1.27	-2.704
1000000	-0.481	-4.472

4.7 Análisis de resultados

En cuanto en la generación de la señal de entrada se puede resumir en los siguientes puntos en comparación con la versión anterior [11, 12], los cuáles son:

- Las frecuencias generadas son muy exactas, tienen un error de $1.63x10^{-5}$ que es mucho menor, y en todas las frecuencias generadas tienen casi el mismo error.
- Los armónicos de la señal generada por el DDS AD9833 son mucho menores haciendo esto la señal más pura y por lo tanto un análisis más limpio del filtro electrónico en estudio.
- Debido a que la señal del DDS no se puede modificar por medio del mismo, es necesario una adecuación para esta señal, lo cuál hace que la gráfica sea un poco más grande en el rango positivo de los decibeles, haciendo que el sistema pueda medir mayor ganancia en un filtro.

Otro punto a tratar es la ampliación de la gráfica de magnitud de la respuesta en frecuencia haciendo que el filtro puedan ser analizados en rangos de magnitud más grande en el rango negativo de los decibeles, esto gracias a la ayuda de los amplificadores, PGA, que además de hacer la gráfica más grande se encargan de suavizar la gráfica evitando que se vean puntos o líneas rectas en la gráfica, cosa que pasaba en la versión anterior [11, 12].

En cuanto al análisis del filtro y la comparación con otros proyectos es muy difícil hacer una comparación total debido a que es imposible tener los mismos valores de los componentes pasivos como los son los capacitores y resistencias, incluso si se llegarán a tener los valores exactos de resistencias que la versión pasada esta también el hecho de que cada amplificador operacional trabaje de la misma manera que en la versión pasada [11, 12], incluso en los componentes pasivos es imposible que tengan la misma respuesta en frecuencia, y comparando con las simulaciones es también casi imposible que se pueda tener una comparación exacta del análisis de la

respuesta en frecuencia, es por eso que hay valores de las tablas que no se consideran debido a que es muy grande la diferencia.

Pero fuera de todo eso los resultados son alentadores y los resultados son muy cercanos a los valores esperados, eso sí, hay valores que no tienen mucha concordancia respecto a los OFFSET de los amplificadores, debido a la respuesta que tienen los componentes pasivos, es por eso que no se tomaron en cuenta aquellos valores con una diferencia muy grande con respecto a la simulación, estos errores se pueden generar por muchos factores uno de los cuáles son los amplificadores PGA, debido a que si algún componente genera un OFFSET, ya se un componente pasivo o un componente activo, los amplificadores PGA amplifican más el OFFSET haciendo que el error en el análisis del filtro electrónico sea erróneo.

Con respecto a la fase, se decidió no hacer mediciones de la fase debido a que el circuito que se tiene para la medición de la fase es muy poco factible debido a varios factores como: para medir la fase se necesita acerca de 40 segundos para que el filtro pasivo RC de 4to orden se estabilice, eso en cada frecuencia que se va a medir, otro factor es que cuando un filtro tiene un respuesta de un orden muy grande éste produce desfases muy grandes en la frecuencia de corte, esto quiere decir que para la medición de la fase para filtros de muy alto orden es necesario tener pasos de frecuencias muy chicos, lo que significa que el análisis tomaría mucho tiempo y eso es muy poco factible, pero fuera de esto el programa principal tiene la opción de decidir si se quiere medir la fase.

Para finalizar, los análisis de los filtros arrojaron una seria de tablas con sus respectivos errores comparados con la simulación de los mismos filtros, por cada análisis se tiene una tabla con sus respectivos errores, se sacaron los errores promedios de cada tabla y se calculó un error total de estos errores, estos errores fueron sacados de los mismos filtros que se probaron en la versión anterior [11, 12], esto para hacer una comparación con las dos versiones. En la tabla 4.16 se muestran los errores promedios de cada filtro analizado y su error total de cada

versión. Mencionando que hay valores que no se tomaron en cuenta debido a que la diferencia es muy grande, por diferentes factores mencionados con anterioridad.

Tabla 4.16. Tabla de errores promedio y el error total de cada versión.

Filtro	Error versión anterior (%)	Error versión actual (%)
Pasivo pasa-bajas de primer orden	18.19	18.17
Butterworth pasa-bajas de primer orden	9.08	1.480
Chebyshev pasa-bajas de segundo orden	15.90	7.210
Butterworth pasa-bajas de cuarto orden	9.13	7.702
Butterworth pasa-altas de cuarto orden	22.43	33.00
Doble T rechaza-banda	27.62	5.750
Promedio (%)	17.06	12.21

Se puede observar que el error es mucho menor por lo que se puede afirmar que se tiene un análisis más exacto, gracias a la ayuda de los amplificadores PGA y el ADC de 12bits con que cuenta el microcontrolador a comparación del pasado que era de 10bits, y análisis mejorado con todas las ventajas que se han mencionado antes, como el uso del puerto USB, los amplificadores PGA, el ADC del microcontrolador, varios tipos de análisis, entre otras.

Conclusiones y recomendaciones

En este proyecto se logró cumplir con la mayoría de los objetivos propuestos en el capítulo I, cumpliendo con el objetivo principal el cuál es un trazador de gráficas de bode con interfaz con la computadora usando protocolos de comunicación más rápidos y portables como lo son USB y el SPI.

Se realizaron muchas mejoras las cuales son:

- Se empleó el protocolo USB para la comunicación entre el sistema Bode Plotter y la computadora, un protocolo más rápido y a donde va dirigido el futuro de la comunicación entre dispositivos y las computadoras, incluso entre dispositivos sin necesidad de un host para realizar esta comunicación.

- Se utilizó un método más avanzado para la generación de la señal, utilizando uno de los circuitos integrados más avanzados el DDS AD9833, capaz de generar la señal que va hacia el filtro de manera muy exacta y pura.
- Se implementaron amplificadores de ganancia programable, los cuales aumentan el rango negativo de la medición de la señal de salida de los amplificadores.
- Los errores generados en las mediciones de los filtros son menores comparados con versiones anteriores [11, 12].
- Entre otras.

El proyecto fue realizado para que el cálculo de las sumas u operaciones complejas fueran realizadas por la computadora, debido a que es más rápida y tiene mucho más hardware que el microcontrolador.

El microcontrolador fue una herramienta muy valiosa para el desarrollo del proyecto, por que gracias a él se controla todos los elementos del proyecto como lo son: el DDS AD9833, los amplificadores de ganancia programable por medio del módulo SPI, los detectores de pico, también gracias al microcontrolador se capturan la magnitud de la ganancia y la fase, también se controla el Firmware para cumplir con los objetivos.

Gracias a los nuevos métodos y tecnología de síntesis digital directa conocida como DDS en circuitos integrados y al alcance de todos es posible realizar un análisis más exacto y fiable.

Recomendaciones.

En cuanto a las recomendaciones, se puede concluir que para que el proyecto quede totalmente terminado sólo es necesario concluir con los siguientes puntos:

- Es necesario contar con unas librerías para el graficado en la computadora y así pues dejar de utilizar el MATLAB el cuál realiza esta tarea, y nada más tener nuestro programa principal corriendo.
- Definir las frecuencias a analizar en los otros dos tipos de análisis de filtros: el análisis bajo y el alto.
- Es necesario también proporcionar ideas para crear un circuito que mida la fase de manera que sea lo más exacta posible, y así ver cuál de esas ideas es la más fiable.
- Una vez teniendo el circuito final para la medición de la fase, el siguiente paso sería ver cuál sería la opción más viable para alimentar al sistema, puede ser utilizando el mismo puerto USB para la alimentación de todo el sistema, si se alimentaría de manera externa o una combinación de ambos.
- Teniendo todo lo demás y para concluir el proyecto, el siguiente paso sería pasar todo el hardware del proyecto a PCB y hacer su contenedor.

Estas recomendaciones son para nuevos tesistas, o incluso para los alumnos que cursen la materia de prácticas profesionales en la institución. Y si se desea comercializar contar con las licencias necesarias para realizar este objetivo, como lo son las licencias para el uso y comercialización del USB, entre otras.

BIBLIOGRAFÍA Y REFERENCIAS

[1] AXELSON jan, "USB COMPLETE: Everything You Need to Develop Custom Peripherals", Tercera edición, Editorial Lakeview Research LLC Madison, WI53704, consultada en octubre del 2008.

[2] COMPAQ Computer Corporation, HEWLETT-PACKARD Company, INTEL Corporation, LUCENT Technologies Inc, MICROSOFT Corporation, NEC Corporation, Koninklijke PHILIPS Electronics N.V., "Universal Serial Bus, Revision 2.0", abril del 2000, consultada en octubre del 2008.

[3] MARTIN P. Bates, LUCIO Di Jasio, CHUCK Hellebuyck, DOGAN Ibrahim, JOHN Morton, D.W. Smith, JACK Smith, "PIC Microcontrollers: Know It All", Editorial Newnes, 2008, consultada en noviembre del 2008.

[4] JAVIER García de Jalón de la Fuente, JOSÉ Ignacio Rodríguez Garrido, ALFONSO Brazález Guerra, PATXI Funes Martínez, RUFINO Goñi Lasheras, RUBÉN Rodríguez Tamayo, "Aprenda lenguaje ANSI C como si estuviera en primero", Escuela Superior de Ingenieros Industriales, UNIVERSIDAD DE NAVARRA, Febrero 1998, consultada en octubre del 2008.

[5] STEPHEN Randy Davis, "C++ for DUMMIES", Quinta edición, Editorial Wiley Publishing, Inc., 2004, consultada en diciembre del 2008.

[6] C. BRITTON Rorabaugh, "Digital Filtes, Designer's Handbook: Featuring C Routines", Editorial TAB Books división de MCGraw-Hill, Inc., 1993, consultada en septiembre del 2008.

[7] JOUKO Vankka, "Direct Digital Synthesizers, Design and Applications", Helsinki University of Technology, Department of Electrical and Communications Engineering, noviembre 2000, consultada en febrero del 2009.

[8] ANALOG DEVICES, "A Technical Tutorial on Digital Signal Synthesis", 1999, documento web, http://www.analog.com/static/imported-

files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf, consultado en febrero del 2009.

[9] PAUL Smith, "Direct Digital Synthesis Theory and Applications", Analog Devices, 22 de abril del 2007, documento web, http://ewh.ieee.org/r5/dallas/aes//IEEE_AESS_22April07_DDS_Analog.ppt,, consultada en marzo del 2009.

[10] ANALOG DEVICES, "Direct Digital Synthesis (DDS)", página web, http://www.analog.com/en/rfif-components/direct-digital-synthesis-dds/products/index.html, consultada en agosto del 2008.

[11] JAIME Isaac Bojórquez Guerrero, "DESARROLLO DE SOFTWARE Y FIRMWARE PARA TRAZADOR DE GRÁFICAS DE BODE", Instituto Tecnológico de Sonora, mayo del 2007, consultada en mayo del 2009.

[12] ALBERTO Ulises Higuera López, "DESARROLLO DE HARDWARE E INTERFAZ PARA TRAZADOR DE GRÁFICAS DE BODE PARA FILTROS", Instituto Tecnológico de Sonora, mayo 2007, consultada en mayo del 2009.

[13] WIKIPEDIA, La Enciclopedia Libre, "Programación orientada a objetos", página web, http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos, consultada en febrero del 2009.

[14] MICROCHIP Technology Inc., "Datasheet PIC18F2458/2553/4458/4553", documento web, http://ww1.microchip.com/downloads/en/DeviceDoc/39887b.pdf, consultada en octubre del 2008.

[15] MICROSOFT, "MSDN; Microsoft Developer Network", página web, http://msdn.microsoft.com/es-es/default.aspx, consultada en diciembre del 2009.

[16] AXELSON Jan, "Lakeview Research", página web, http://www.lvr.com/, consultada en enero del 2009.

APÉNDICE A

Archivo MCHPWinUSBDevice.inf para el controlador del Bode Plotter

```
;Adapted from Jan Axelson's (of Lakeview Research: www.lvr.com) winusbdemo.inf file.
;Which in turn was adapted from the example INF in the Microsoft document:
;"How to Use WinUSB to Communicate with a USB Device"

[Version]
Signature = "$Windows NT$"
Class = CustomUSBDevices
ClassGuid= {a503e2d3-a031-49dc-b684-c99085dbfe92}

Provider = %MFGNAME%
DriverVer=05/21/2008,1.0.0.0
;CatalogFile=MCHPWinUSBDevice.cat ;CAT file used when obtaining a WHQL signature on the driver
package
;Otherwise isn't needed.
;------------------------------------------------------------------------
; ========== Manufacturer/Models sections ===========
;------------------------------------------------------------------------
[Manufacturer]
%MFGNAME% = MyDevice_WinUSB,NTx86,NTamd64
;------------------------------------------------------------------------
;Vendor and Product ID Definitions
;------------------------------------------------------------------------
; When developing your custom USB device, the VID and PID used in the PC side
; application program and the firmware on the microcontroller must match.
; Modify the below lines to use your VID and PID.  Use the format as shown below.
; Note: One INF file can be used for multiple devices with different VID and PIDs.
; For each supported device, append ",USB\VID_xxxx&PID_yyyy" to the end of the line.
; There is a maximum number of devices that can be supported per line however.
; If you append a large number of VID/PIDs to the end of the line, and get a:
; "The data area passed to a system call is too small." error when trying to install
; the INF, try removing some of the VIDs/PIDs.
;------------------------------------------------------------------------
[MyDevice_WinUSB.NTx86]
%DESCRIPTION% =USB_Install, USB\VID_04d8&PID_0021

[MyDevice_WinUSB.NTamd64]
%DESCRIPTION% =USB_Install, USB\VID_04d8&PID_0021
;=========================================================================
============
;ClassInstall32 and ClassInstall_AddReg sections used to make new device manager category.
;=========================================================================
============
[ClassInstall32]
AddReg=ClassInstall_AddReg
[ClassInstall_AddReg]
HKR,,,,%DEVICEMANAGERCATEGORY%
HKR,,Icon,,"-20"
; ================== Installation ==================
[USB_Install]
```

```
Include=winusb.inf
Needs=WINUSB.NT

[USB_Install.Services]
Include=winusb.inf
AddService=WinUSB,0x00000002,WinUSB_ServiceInstall

[WinUSB_ServiceInstall]
DisplayName     = %WinUSB_SvcDesc%
ServiceType     = 1
StartType       = 3
ErrorControl    = 1
ServiceBinary   = %12%\WinUSB.sys

[USB_Install.Wdf]
KmdfService=WINUSB, WinUsb_Install
UmdfServiceOrder=WINUSB

[WinUSB_Install]
KmdfLibraryVersion=1.7

[USB_Install.HW]
AddReg=Dev_AddReg

[Dev_AddReg]
HKR,,DeviceInterfaceGUIDs,0x10000,"{64AD9921-A458-43c9-97DA-1CF2D4CC8D30}"
;When editing the GUID (the big hex number with dashes inside the squiggly
;braces), make sure to write the intended PC application to use the same GUID.
;Otherwise the application won't be able to find the USB device properly.

[USB_Install.CoInstallers]
AddReg=CoInstallers_AddReg
CopyFiles=CoInstallers_CopyFiles

[CoInstallers_AddReg]
HKR,,CoInstallers32,0x00010000,"WinUSBCoInstaller.dll","WUDFUpdate_01007.dll","WdfCoInstaller0
1007.dll,WdfCoInstaller"

[CoInstallers_CopyFiles]
WinUSBCoInstaller.dll
WdfCoInstaller01007.dll
WUDFUpdate_01007.dll

[DestinationDirs]
CoInstallers_CopyFiles=11
; ================= Source Media Section =====================
[SourceDisksNames.x86]
1 = %DISK_NAME%,,,\i386
```

```
[SourceDisksNames.amd64]
2 = %DISK_NAME%,,,\amd64

[SourceDisksFiles.x86]
WinUSBCoInstaller.dll=1
WdfCoInstaller01007.dll=1
WUDFUpdate_01007.dll=1

[SourceDisksFiles.amd64]
WinUSBCoInstaller.dll=2
WdfCoInstaller01007.dll=2
WUDFUpdate_01007.dll=2
; Copy Files section
;---------------------------------------------------------------------------
[_CopyFiles_sys]
winusb.sys
;************************************************* ******************************
; Destination Directories
;---------------------------------------------------------------------------
[DestinationDirs]
DefaultDestDir = 12 ; %SystemRoot%\system32\drivers
_CopyFiles_sys = 12
; ================== Strings ==================
[Strings]
MFGNAME="ITSON - IE"
DESCRIPTION="Bode Plotter v3.0"
WinUSB_SvcDesc="WinUSB Device"
DISK_NAME="WinUSB Device Install Disk"
DEVICEMANAGERCATEGORY="Custom USB Devices"
```

Código fuente del archivo main.c del Firmware del Bode plotter

```
/********************************************************************
 FileName: main.c
 Dependencies: See INCLUDES section
 Processor: PIC18 or PIC24 USB Microcontrollers
 Hardware: The code is natively intended to be used on the following hardware
platforms:

 PICDEM™ FS USB Demo Board,
PIC18F87J50 FS USB Plug-In Module, or Explorer 16 + PIC24 USB PIM.  The
firmware may be modified for use on other USB platforms by editing the
HardwareProfile.h file.
Complier: Microchip C18 (for PIC18) or C30 (for PIC24)
Company: Microchip Technology, Inc.

Software License Agreement:

The software supplied herewith by Microchip Technology Incorporated
(the "Company") for its PIC® Microcontroller is intended and
supplied to you, the Company's customer, for use solely and
exclusively on Microchip PIC Microcontroller products. The
software is owned by the Company and/or its supplier, and is
protected under applicable copyright laws. All rights are reserved.
Any use in violation of the foregoing restrictions may subject the
user to criminal sanctions under applicable laws, as well as to
civil liability for the breach of the terms and conditions of this
license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO
WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY,
INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,  IN ANY
CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

********************************************************************
File Description:

Change History:
Rev Date Description
1.0   11/19/2004 Initial release
2.1   02/26/2007 Updated for simplicity and to use common coding style

* Programa final del Bode Plotter v3.0 con conexion USB (Conectado como
*
* puerto USB no como COM) con conexión con Bode Plotter v3.0
*
* creado en MS Visual C++
*
* Descripcion del Archivo:
*
* Creado el xx de xxxxxx del 2009
*
* Autor: Arturo Aganza Torres
*
********************************************************************/

/** INCLUDES *******************************************************/
#include        <p18f2553.h>
#include        <portb.h>
#include        <spi.h>
#include        <pconfig.h>
#include        <delays.h>
#include        <Compiler.h>
#include        <usb_config.h>
#include        <GenericTypeDefs.h>
#include        <usb_device.h>
#include        <usb.h>
#include        <usb_function_generic.h>
#include        <define.h>

/** CONFIGURATION**************************************************/

// Configuration bits for PICDEM FS USB Demo Board (based on PIC18F4550)
#pragma config PLLDIV   = 10  // (40 MHz crystal on PICDEM FS USB board)
#pragma config CPUDIV   = OSC1_PLL2
#pragma config USBDIV   = 2        // Clock source from 96MHz PLL/2
#pragma config FOSC     = ECPLLIO_EC
#pragma config FCMEN    = OFF
#pragma config IESO     = OFF
#pragma config PWRT     = OFF
#pragma config BOR      = ON
#pragma config BORV     = 3
#pragma config VREGEN   = ON      //USB Voltage Regulator
#pragma config WDT      = OFF
#pragma config WDTPS    = 32768
#pragma config MCLRE    = ON
#pragma config DEBUG      = OFF
#pragma config LPT1OSC  = OFF
#pragma config PBADEN   = OFF
#pragma config CCP2MX   = ON
#pragma config STVREN   = ON
#pragma config LVP      = OFF
#pragma config XINST    = OFF      // Extended Instruction Set
#pragma config CP0      = OFF
#pragma config CP1      = OFF
#pragma config CP2      = OFF
#pragma config CP3      = OFF
#pragma config CPB      = OFF
#pragma config CPD      = OFF
#pragma config WRT0     = OFF
#pragma config WRT1     = OFF
#pragma config WRT2     = OFF
#pragma config WRT3     = OFF
#pragma config WRTB     = OFF      // Boot Block Write Protection
#pragma config WRTC     = OFF
#pragma config WRTD     = OFF
#pragma config EBTR0    = OFF
#pragma config EBTR1    = OFF
#pragma config EBTR2    = OFF
#pragma config EBTR3    = OFF
#pragma config EBTRB    = OFF

/**VARIABLES*****************************************************/

#if defined(__18F14K50) || defined(__18F13K50) || defined(__18LF14K50) ||
defined(__18LF13K50)
#pragma udata usbram2
#elif defined(__18F2455) || defined(__18F2550) || defined(__18F4455) ||
defined(__18F4550)\
    || defined(__18F4450) || defined(__18F2450)\
    || defined(__18F2458) || defined(__18F2453) || defined(__18F4558) ||
defined(__18F4553)
#pragma udata USB_VARIABLES=0x500
#else
   #pragma udata
#endif

unsigned char OUTPacket[64]; //User application buffer for receiving and holding
OUT packets sent from the host

unsigned char INPacket[64]; //User application buffer for sending IN packets to the
host

#pragma udata
BOOL blinkStatusValid;
unsigned char TIEMPO = 0;  //Variable BOOL para el bucle del tiempo transitorio.

unsigned char DELAY_PKD01 = 0;
unsigned char LTIEMPO = 0;
unsigned char HTIEMPO = 0;
unsigned short ADC_NOISE = 0;
USB_HANDLE USBGenericOutHandle;
USB_HANDLE USBGenericInHandle;
unsigned long FREQREG = 0x00000000;
unsigned char TEMPA = 0;
unsigned char f, b, d, a; //Variables para el Loop for.
unsigned short ba = 0;
unsigned long g = 0;
float fRESOLU;
float BRESOLU;
union FRECUENCIA
{
                unsigned long VAL;
                unsigned char v[4];
}FINICIAL, FFINAL, RESOLU, RESTAA;

#pragma idata A_VARIABLES=0x600

const unsigned long AMEDIOPacket[62] = {10, 21, 32, 42, 53, 64, 75, 85, 96, 107,
128, 150, 171, 193, 214, 322, 429, 536, 644, 751, 858, 966, 1073, 1288, 1503, 1717,
1932, 2147, 3221, 4294, 5368, 6442, 7516, 8589, 9663, 10737, 12884, 15032, 17179,
19327, 21474, 32212, 42949, 53687, 64424, 74161, 85899, 96636, 107374, 128849,
150323, 171798, 193273, 214748, 322122, 429496, 536870, 644245, 751619, 858993,
966367, 1073741};
union PRUEBA{
                        unsigned short VAL;
                        unsigned char v[2];
                } ADCBUFFER, UBLOQUES;
union FASE{
                        signed short VAL;
                        unsigned char v[2];
            } FADCBUFFER, F2ADCBUFFER;



/**PRIVATEPROTOTYPES*********************************************/

static void InitializeSystem(void);
void USBDeviceTasks(void);
void YourHighPriorityISRCode(void);
void YourLowPriorityISRCode(void);
void ProcessIO(void);
void BlinkUSBStatus(void);
void SEND_DDS(unsigned long FRECUENCIA);
void INICIA_DDS(void); //Inicia el DDS a frecuencia de 0 Hz.
void INICIA_AMPLIS(void); //Inicia los LTC6912 Amplificadores a 0 dB, o sea
ganancia 1.
```

```
void SEND(void);
void WRITE_AMP(unsigned char STEP); //Manda el contro al los LTC6912.
void TIMERS(void); //Inicializa el timer para la interrupción del
USBDevicesTasks.

void IINTERRUPCION(void); //Inicia todos los registros de las interrupciones
del proyecto.

unsigned char MAKE8(unsigned long var, unsigned char offset);
void TIEMPO_A(void);
void TIEMPO_R(void);
void LEER_ADC(void);
void CONTROL_LTC(void);
//void INT_FASE(void);

/*VECTOR REMAPPING ***************************************/

#if defined(__18CXX)
        //On PIC18 devices, addresses 0x00, 0x08, and 0x18 are used for
        //the reset, high priority interrupt, and low priority interrupt
        //vectors.  However, the current Microchip USB bootloader
        //examples are intended to occupy addresses 0x00-0x7FF or
        //0x00-0xFFF depending on which bootloader is used.  Therefore,
        //the bootloader code remaps these vectors to new locations
        //as indicated below.  This remapping is only necessary if you
        //wish to program the hex file generated from this project with
        //the USB bootloader.  If no bootloader is used, edit the
        //usb_config.h file and comment out the following defines:
//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
//#define
PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOA
DER
#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
  #define REMAPPED_RESET_VECTOR_ADDRESS     0x1000
  #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS 0x1008
  #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS 0x1018
#elif defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
  #define REMAPPED_RESET_VECTOR_ADDRESS             0x800
  #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x808
  #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS     0x818
#else
  #define REMAPPED_RESET_VECTOR_ADDRESS             0x00
  #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x08
  #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS     0x18
#endif

#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PR
OGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
  extern void _startup (void);        // See c018i.c in your C18 compiler dir
  #pragma code REMAPPED_RESET_VECTOR =
REMAPPED_RESET_VECTOR_ADDRESS
        void _reset (void)
        {
            _asm goto _startup _endasm
        }
#endif
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR =
REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
        void Remapped_High_ISR (void)
        {
            _asm goto YourHighPriorityISRCode _endasm
        }
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR =
REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
        void Remapped_Low_ISR (void)
        {
            _asm goto YourLowPriorityISRCode _endasm
        }

#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PR
OGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
//Note: If this project is built while one of the bootloaders has
//been defined, but then the output hex file is not programmed with
//the bootloader, addresses 0x08 and 0x18 would end up programmed with
//0xFFFF.
//As a result, if an actual interrupt was enabled and occured, the PC would jump
//to 0x08 (or 0x18) and would begin executing "0xFFFF" (unprogrammed
//space).  This
//executes as nop instructions, but the PC would eventually reach the
REMAPPED_RESET_VECTOR_ADDRESS
//(0x1000 or 0x800, depending upon bootloader), and would execute the "goto
//_startup".  This
//would effective reset the application.
//To fix this situation, we should always deliberately place a
//"goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS" at address
//0x08, and a
//"goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS" at address
//0x18.  When the output
//hex file of this project is programmed with the bootloader, these sections do
//not
//get bootloaded (as they overlap the bootloader space).  If the output hex file is //not
//programmed using the bootloader, then the below goto instructions do get
//programmed,
//and the hex file still works like normal.  The below section is only required to //fix
this
//scenario.
#pragma code HIGH_INTERRUPT_VECTOR = 0x08
        void High_ISR (void)
        {
            _asm goto
REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
        }
        #pragma code LOW_INTERRUPT_VECTOR = 0x18
        void Low_ISR (void)
        {
            _asm goto
REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
        }
#endif
//end of "#if
//defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(P///RO/
//GRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOA///DER)"

#pragma code


//These are your actual interrupt handling routines.
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode()
{
  if(PIR1bits.TMR1IF)
    {
        TIMER1 = APAGADO;
        USBDeviceTasks();
        TMR1L = 0x6A;
        Nop();
        TMR1H = 0xFF;
        PIR1bits.TMR1IF = 0;
        TIMER1 = ENCENDIDO;
    }
} //This return will be a "retfie fast", since this is in a #pragma interrupt section

#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode()
{
   if(INTCONbits.TMR0IF)
    {
        TMR_TRANS = APAGADO;
        TIEMPO = FALSE;
        TMR0L = LTIEMPO;
        TMR0H = HTIEMPO;
        INTCONbits.TMR0IF = 0;
    }
   if(PIR1bits.TMR2IF)
    {
        TMR_PKD01 = APAGADO;
        DELAY_PKD01 = FALSE;
        TMR2 = 0xFF;
        PR2 = 0x69;
        PIR1bits.TMR2IF = 0;
   }
} //This return will be a "retfie", since this is in a #pragma interruptlow section

#elif defined(__C30__)
  #if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
    /*
     *ISR JUMP TABLE
     *
     *It is necessary to define jump table as a function because C30 will
     *not store 24-bit wide values in program memory as variables.
     *
     *This function should be stored at an address where the goto instructions
     *line up with the remapped vectors from the bootloader's linker script.
     *
     *  For more information about how to remap the interrupt vectors,
     *  please refer to AN1157.  An example is provided below for the T2
     *  interrupt with a bootloader ending at address 0x1400
     */
//      void __attribute__ ((address(0x1404))) ISRTable(){
//
//          asm("reset"); //reset instruction to prevent runaway code
//          asm("goto %0"::"i"(&_T2Interrupt));  //T2Interrupt's address
//      }
 #endif
#endif    //of "#if defined(__18CXX)"
/* DECLARATIONS ******************************************************/
#pragma code
/*********************************************************************
* Function: void main(void)
* PreCondition:    None
* Input:           None
* Output:          None
* Side Effects:    None
 * Overview: Main program entry point.
```

```
 * Note: None
 *******************************************************************/

void main(void)
  {
     InitializeSystem();
while(1)
    {
                //USBDeviceTasks();
                ProcessIO();


    }// Find de while(1)
  }//Fin de main


/******************************************************************
* Function:       static void InitializeSystem(void)
* PreCondition:    None
*
* Input:          None
* Output:         None
* Side Effects:    None
* Overview: InitializeSystem is a centralize initialization
*               routine. All required USB initialization routines
*               are called from here.
*
*               User application initialization routine should
*               also be called from here.
* Note:           None
******************************************************************/
static void InitializeSystem(void)
{
  ADCON1 = 0x0C; //Configura el PORTA como I/O digitales
  ADCON0 = 0x01;
  ADCON2 = 0xFE;
  TRISA = 0x07;
  ClosePORTB(); //Deshabilita las interrupciones y las pull-ups internas del
//PORTB
  DDS_CS=1;
  TRISBbits.TRISB1 = 0;     //SCK del SPI
  TRISCbits.TRISC7 = 0;     //SDO del SPI
  TRISCbits.TRISC6 = 0;     //Habilitador del DDS
  TRISBbits.TRISB7 = 0;     //Para estado del USB
  TRISBbits.TRISB6 = 0;     //Para estado del USB
  TRISBbits.TRISB5 = 0;     //Para estado del USB
  TRISBbits.TRISB4 = 1;     //Detectar si es conectado y empezar
  //TRISBbits.TRISB0 = 1;
  TRISBbits.TRISB2 = 1;
  TRISCbits.TRISC2 = 1;     //Configura el Pin RC2 como entrada, para
//inicializar el USB
  PORTBbits.RB7=0;
  PORTBbits.RB6=0;
  PORTBbits.RB5=0;
  AMP1_CS = 1;
  AMP2_CS = 1;
  PEAKOUT = RESET;
  PEAKIN =  RESET;
  INICIA_DDS();
  INICIA_AMPLIS();
  IINTERRUPCION();
  TIMERS();
  OUTPacket[0] = ESPERAR;
  OUTPacket[1] = 0;
//The USB specifications require that USB peripheral devices must never source
//current onto the Vbus pin.  Additionally, USB peripherals should not source
//current on D+ or D- when the host/hub is not actively powering the Vbus line.
//When designing a self powered (as opposed to bus powered) USB peripheral
//device, the firmware should make sure not to turn on the USB module and D+
//or D- pull up resistor unless Vbus is actively powered.  Therefore, the
//firmware needs some means to detect when Vbus is being powered by the host.
//A 5V tolerant I/O pin can be connected to Vbus (through a resistor), and
//can be used to detect when Vbus is high (host actively powering), or low
//(host is shut down or otherwise not supplying power).  The USB firmware
//can then periodically poll this I/O pin to know when it is okay to turn on
//the USB module/D+/D- pull up resistor.  When designing a purely bus powered
//peripheral device, it is not possible to source current on D+ or D- when the
//host is not actively providing power on Vbus. Therefore, implementing this
//bus sense feature is optional.  This firmware can be made to use this bus
//sense feature by making sure "USE_USB_BUS_SENSE_IO" has been defined
//in the
//HardwareProfile.h file.
    #if defined(USE_USB_BUS_SENSE_IO)
    tris_usb_bus_sense = INPUT_PIN; // See HardwareProfile.h
    #endif

//If the host PC sends a GetStatus (device) request, the firmware must respond
//and let the host know if the USB peripheral device is currently bus powered
//or self powered.  See chapter 9 in the official USB specifications for details
//regarding this request.  If the peripheral device is capable of being both
//self and bus powered, it should not return a hard coded value for this request.
//Instead, firmware should check if it is currently self or bus powered, and
//respond accordingly.  If the hardware has been configured like demonstrated
//on the PICDEM FS USB Demo Board, an I/O pin can be polled to determine the
//currently selected power source.  On the PICDEM FS USB Demo Board, "RA2"
//is used for     this purpose.  If using this feature, make sure
//"USE_SELF_POWER_SENSE_IO"
//has been defined in HardwareProfile.h, and that an appropriate I/O pin has been
//mapped
//to it in HardwareProfile.h.
    #if defined(USE_SELF_POWER_SENSE_IO)
    tris_self_power = INPUT_PIN; // See HardwareProfile.h
    #endif

USBGenericOutHandle = 0;
USBGenericInHandle = 0;
USBDeviceInit(); //usb_device.c.  Initializes USB module SFRs and firmware
//variables to known states.
T1CONbits.TMR1ON = 1; //Temporizador para la actualización del USB
blinkStatusValid = TRUE;

}//fin InitializeSystem

/******************************************************************
* Function: void ProcessIO(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:     This function is a place holder for other user routines.
*               It is a mixture of both USB and non-USB tasks.
* Note: None
*******************************************************************/
void ProcessIO(void)
{
if(blinkStatusValid)
    {
      BlinkUSBStatus();
    }
while(OUTPacket[0] == ESPERANDO)
  {
USBGenericOutHandle=USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,
USBGEN_EP_SIZE);
  }
if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;

if(!USBHandleBusy(USBGenericOutHandle))
  {
     if(OUTPacket[2] == EMPEZAR)
      {
                if(OUTPacket[1] == AMEDIO)
                {
                }//FIN if(OUTPacket[1] == AMEDIO)
                if(OUTPacket[1] == ABAJO)
                {
                }//FIN if(OUTPacket[1] == ABAJO)
                if(OUTPacket[1] == AALTO)
                {
                        PORTBbits.RB7 = 1;
                        PORTBbits.RB6 = 1;
                        F2ADCBUFFER.VAL = 0;
                        d = 2; f = 0; b = 0; a = 0;
                        ADCON0 = CANALVIN;
                        SEND_DDS(10737);
                        TIEMPO = 1;
                        TMR_TRANS = ENCENDIDO;
                        while(TIEMPO);
                        PEAKIN = DETECT;
                        DELAY_PKD01 = TRUE;;
                        TMR_PKD01 = ENCENDIDO;
                        while(DELAY_PKD01);
                        LEER_ADC();
                        PEAKIN = RESET;
                        SEND_DDS(0);
                        INPacket[62] = ADRESL;
                        INPacket[63] = ADRESH;
                        INICIA_AMPLIS();
                        ADCON0 = CANALVOUT;
                        SEND_DDS(AMEDIOPacket[0]);
                        LTIEMPO = 0xE4;
                        HTIEMPO = 0x48;
                        TMR0L = 0xE4;
                        TMR0H = 0x48;
                        if(OUTPacket[21])
                          {
                            while(a <= 40)
                            {
                                TIEMPO = 1;
                                TMR_TRANS = ENCENDIDO;
                                while(TIEMPO);
                                TIEMPO = 1;
                                TMR_TRANS = ENCENDIDO;
                                while(TIEMPO);
                                a++;
                            }
                          }
                        for(; f <= 61; f++)
```

```
                                {
                                  TIEMPO_A();
                                  SEND_DDS(AMEDIOPacket[f]);
                                  Nop(); Nop(); Nop();
                                  TIEMPO = 1;
                                  TMR_TRANS = ENCENDIDO;
                                  while(TIEMPO);
                                  PEAKOUT = DETECT;
                                  DELAY_PKD01 = TRUE;;
                                  TMR_PKD01 = ENCENDIDO;
                                  while(DELAY_PKD01);
                                  TIEMPO = 1;
                                  TMR_TRANS = ENCENDIDO;
                                  while(TIEMPO);
                                  LEER_ADC();
                                  PEAKOUT = RESET;
                                  DELAY_PKD01 = TRUE;
                                  TMR_PKD01 = ENCENDIDO;
                                  while(DELAY_PKD01);
                                  CONTROL_LTC();
                                  PEAKOUT = DETECT;
                                  DELAY_PKD01 = TRUE;;
                                  TMR_PKD01 = ENCENDIDO;
                                  while(DELAY_PKD01);
                                  TIEMPO = 1;
                                  TMR_TRANS = ENCENDIDO;
                                  while(TIEMPO);
                                  Delay1KTCYx(50);
                                  TIEMPO = 1;
                                  TMR_TRANS = ENCENDIDO;
                                  while(TIEMPO);
                                  LEER_ADC();
                                if((ADCBUFFER.VAL == 0) && (TEMPA == 12))
                                   ADCBUFFER.VAL == 1;
                                   PEAKOUT = RESET;
if((ADCBUFFER.VAL >= RANGOADCBAJO) && (ADCBUFFER.VAL <=
RANGOADCALTO))
  {
                INPacket[d++] = TEMPA;
                INPacket[d++] = ADCBUFFER.v[0];
                INPacket[d++] = ADCBUFFER.v[1];
                if(OUTPacket[21])
                  {
                    ADCON0 = CANALFASE;
                    TMR0L = 0xE4;
                    TMR0H = 0x48;
                    LTIEMPO = 0xE4;
                    HTIEMPO = 0x48;
                    a = 0;
                while(a <= 1)
                  {
                                TIEMPO = 1;
                                TMR_TRANS = ENCENDIDO;
                                while(TIEMPO);
                                TIEMPO = 1;
                                TMR_TRANS = ENCENDIDO;
                                while(TIEMPO);
                                a++;
                  }
                LEER_ADC();
                FADCBUFFER.v[0] = ADCBUFFER.v[0];
                FADCBUFFER.v[1] = ADCBUFFER.v[1];
if(((FFASE) && (FADCBUFFER.VAL <= F2ADCBUFFER.VAL)) ||
((!FFASE) && (FADCBUFFER.VAL >= F2ADCBUFFER.VAL)))
  {
                F2ADCBUFFER.VAL = FADCBUFFER.VAL;
                FADCBUFFER.VAL *= 1;
  }
else if(((FFASE) && (FADCBUFFER.VAL >= F2ADCBUFFER.VAL)) ||
((!FFASE) && (FADCBUFFER.VAL <= F2ADCBUFFER.VAL)))
  {

                F2ADCBUFFER.VAL = FADCBUFFER.VAL;
                FADCBUFFER.VAL *= (-1);
  }
INPacket[d++] = FADCBUFFER.v[0];
INPacket[d++] = FADCBUFFER.v[1];
ADCON0 = CANALVOUT;
}//Fin de if(OUTPacket[21])
                else
                {
                                INPacket[d++] = 0;
                                INPacket[d++] = 0;
                }
//TEMPA = 0;
//d += 2;
}//END if((ADCBUFFER.VAL >= RANGOADCBAJO) && (ADCBUFFER
//<= RANGOADCALTO))
if(d == 62)
  {
     TIMER1 = APAGADO;
      if(!USBHandleBusy(USBGenericInHandle))
        {
                USBGenericInHandle=USBGenWrite(USBGEN_EP_NUM,(BYTE*)&
INPacket,USBGEN_EP_SIZE);
        }
TIMER1 = ENCENDIDO;
b += 1;
d = 2;
}//END if(d == 62)
if((f == 61) && (b == 5)) //Mandar el ultimo bloque (bloque 6)
  {
                TIMER1 = APAGADO;
                if(!USBHandleBusy(USBGenericInHandle))
                  {
                USBGenericInHandle=USBGenWrite(USBGEN_EP_NUM,(BYTE*)&
                INPacket,USBGEN_EP_SIZE);
                  }
TIMER1 = ENCENDIDO;
OUTPacket[1] = TERMINADO;
OUTPacket[2] = TERMINADO;
OUTPacket[0] = ESPERAR;
INICIA_AMPLIS();
SEND_DDS(0);
PORTBbits.RB7 = 0;
PORTBbits.RB6 = 0;
  }//END if((f == 61) && (b == 5)) //Mandar el ultimo bloque (bloque 6)
}//END for(; f<=61; f++)
}//FIN if(OUTPacket[1] == AALTO)

if(OUTPacket[1] == ARANGO)
  {
                PORTBbits.RB7 = 1;
                PORTBbits.RB6 = 1;
                ADCON0 = CANALVIN;
                SEND_DDS(10737);
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                PEAKIN = DETECT;
                DELAY_PKD01 = TRUE;;
                TMR_PKD01 = ENCENDIDO;
                while(DELAY_PKD01);
                LEER_ADC();
                PEAKIN = RESET;
                SEND_DDS(0);
                INPacket[62] = ADRESL;
                INPacket[63] = ADRESH;
                ADCON0 = CANALVOUT;
                a = 0;
                FINICIAL.v[0] = OUTPacket[3];
                FINICIAL.v[1] = OUTPacket[4];
                FINICIAL.v[2] = OUTPacket[5];
                FINICIAL.v[3] = OUTPacket[6];
                FFINAL.v[0] = OUTPacket[7];
                FFINAL.v[1] = OUTPacket[8];
                FFINAL.v[2] = OUTPacket[9];
                FFINAL.v[3] = OUTPacket[10];
                RESOLU.v[0] = OUTPacket[11];
                RESOLU.v[1] = OUTPacket[12];
                RESOLU.v[2] = OUTPacket[13];
                RESOLU.v[3] = OUTPacket[14];
                UBLOQUES.v[0] = OUTPacket[15];
                UBLOQUES.v[1] = OUTPacket[16];
                RESTAA.v[0] = OUTPacket[17];
                RESTAA.v[1] = OUTPacket[18];
                RESTAA.v[2] = OUTPacket[19];
                RESTAA.v[3] = OUTPacket[20];
fRESOLU = ((float)268435455 * ((float)RESOLU.VAL)) / ((float)25000000);
                d = 2; f = 0; g = 0x00000000; ba = 0x0000;
                INICIA_AMPLIS();
                if(OUTPacket[21])
                  {
                                F2ADCBUFFER.VAL = 0;
                                SEND_DDS(FINICIAL.VAL);
                                LTIEMPO = 0xE4;
                                HTIEMPO = 0x48;
                                TMR0L = 0xE4;
                                TMR0H = 0x48;
                                while(a <= 40)
                                {
                                        TIEMPO = 1;
                                        TMR_TRANS = ENCENDIDO;
                                        while(TIEMPO);
                                        TIEMPO = 1;
                                        TMR_TRANS = ENCENDIDO;
                                        while(TIEMPO);
                                        a++;
                                }
                  }
for( ; FINICIAL.VAL <= FFINAL.VAL; g++)
{
                TIEMPO_R();
                SEND_DDS(FINICIAL.VAL);
                Nop(); Nop(); Nop(); Nop();
                TIEMPO = 1;
```

```
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                PEAKOUT = DETECT;
                DELAY_PKD01 = TRUE;;
                TMR_PKD01 = ENCENDIDO;
                while(DELAY_PKD01);
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                LEER_ADC();
                PEAKOUT = RESET;
                DELAY_PKD01 = TRUE;;
                TMR_PKD01 = ENCENDIDO;
                while(DELAY_PKD01);
                CONTROL_LTC();
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                PEAKOUT = DETECT;
                DELAY_PKD01 = TRUE;;
                TMR_PKD01 = ENCENDIDO;
                while(DELAY_PKD01);
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                Delay1KTCYx(50);
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                LEER_ADC();
                if((ADCBUFFER.VAL == 0) && (TEMPA == 12))
                   ADCBUFFER.VAL == 1;
                PEAKOUT = RESET;
if((ADCBUFFER.VAL >= RANGOADCBAJO) && (ADCBUFFER.VAL <=
RANGOADCALTO))
{
                INPacket[d++] = TEMPA;
                INPacket[d++] = ADCBUFFER.v[0];
                INPacket[d++] = ADCBUFFER.v[1];
                if(OUTPacket[21])
                {
                        ADCON0 = CANALFASE;
                        TMR0L = 0xE4;
                        TMR0H = 0x48;
                        LTIEMPO = 0xE4;
                        HTIEMPO = 0x48;
                        a = 0;
                        while(a <= 1)
                        {
                          TIEMPO = 1;
                          TMR_TRANS = ENCENDIDO;
                          while(TIEMPO);
                          TIEMPO = 1;
                          TMR_TRANS = ENCENDIDO;
                          while(TIEMPO);
                          a++;
                        }
                LEER_ADC();
                FADCBUFFER.v[0] = ADCBUFFER.v[0];
                FADCBUFFER.v[1] = ADCBUFFER.v[1];
if(((FFASE) && (FADCBUFFER.VAL <= F2ADCBUFFER.VAL)) ||
((!FFASE) && (FADCBUFFER.VAL >= F2ADCBUFFER.VAL)))
{
                F2ADCBUFFER.VAL = FADCBUFFER.VAL;
                FADCBUFFER.VAL *= 1;
}
else if(((FFASE) && (FADCBUFFER.VAL >= F2ADCBUFFER.VAL)) ||
((!FFASE) && (FADCBUFFER.VAL <= F2ADCBUFFER.VAL)))
{
                F2ADCBUFFER.VAL = FADCBUFFER.VAL;
                FADCBUFFER.VAL *= (-1);
}
INPacket[d++] = FADCBUFFER.v[0];
INPacket[d++] = FADCBUFFER.v[1];
ADCON0 = CANALVOUT;
}//Fin de if(OUTPacket[21])
else
{
                INPacket[d++] = 0;
                INPacket[d++] = 0;
}
if(g == (RESTAA.VAL - 2))
   FINICIAL.VAL = FFINAL.VAL;
else
{
                BRESOLU = ((float)FINICIAL.VAL + fRESOLU);
                FINICIAL.VAL = (unsigned long) BRESOLU;
}
} //END if((ADCBUFFER.VAL >= RANGOADCBAJO) && (ADCBUFFER
//<= RANGOADCALTO))
if(d == 62)
{
                TIMER1 = APAGADO;
                if(!USBHandleBusy(USBGenericInHandle))
                {

USBGenericInHandle=USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket,USB
GEN_EP_SIZE);
                }
TIMER1 = ENCENDIDO;
ba += 1; d = 2;
}//END if(d == 62)
if((g == (RESTAA.VAL - 1)) && (ba == (UBLOQUES.VAL - 1)))
//Mandar el ultimo bloque
{
                TIMER1 = APAGADO;
                if(!USBHandleBusy(USBGenericInHandle))
                {
                        USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket,USBGEN_EP_SIZE);
                }
TIMER1 = ENCENDIDO;
OUTPacket[1] = TERMINADO;
OUTPacket[2] = TERMINADO;
OUTPacket[0] = ESPERAR;
INICIA_AMPLIS();
SEND_DDS(0);
PORTBbits.RB7 = 0;
PORTBbits.RB6 = 0;
}//END if((g == (RESTAA.VAL - 1)) && (ba == (UBLOQUES.VAL - 1)))
}//END for( ; FINICIAL <= FFINAL; g++)
}// FIN if(OUTPacket[1] == ARANGO)

if(OUTPacket[1] == CALIBRACION)
{
                SEND();
                PEAKOUT = DETECT;
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                LEER_ADC();
                PEAKOUT = RESET;
                CONTROL_LTC();
                OUTPacket[1] = TERMINADO;
                OUTPacket[2] = TERMINADO;
                OUTPacket[0] = ESPERAR;
}//FIN if(OUTPacket[1] == CALIBRACION)

if(OUTPacket[1] == AUTO)
{
                SEND();
                Nop(); Nop(); Nop(); Nop();
                PEAKIN = DETECT;
                DELAY_PKD01 = 1;
                TMR_PKD01 = ENCENDIDO;
                while(DELAY_PKD01);
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                ADCON0 = CANALVIN;
                ADCGO_DONE = GO;
                while(ADCGO_DONE);
                INPacket[1] = ADRESL;
                INPacket[2] = ADRESH;
                TIMER1 = APAGADO;
                if(!USBHandleBusy(USBGenericInHandle))
                {
                USBGenericInHandle=USBGenWrite(USBGEN_EP_NUM,(BYTE*)&
                INPacket,USBGEN_EP_SIZE);
                }
                TIMER1 = ENCENDIDO;
                PEAKIN = RESET;
                SEND_DDS(0);
                OUTPacket[0] = ESPERAR;
                OUTPacket[2] = TERMINADO;
                OUTPacket[1] = TERMINADO;
}//FIN if(OUTPacket[1] == AUTO)

if(OUTPacket[1] == OFFSET)
{
                PEAKOUT = RESET;
                ADCON0 = CANALVOUT;
                TIEMPO = 1;
                TMR_TRANS = ENCENDIDO;
                while(TIEMPO);
                ADCGO_DONE = GO;
                while(ADCGO_DONE);
                INPacket[1] = ADRESL;
                TIMER1 = APAGADO;
                if(!USBHandleBusy(USBGenericInHandle))
                {
                USBGenericInHandle=USBGenWrite(USBGEN_EP_NUM,(BYTE*)&
                INPacket,USBGEN_EP_SIZE);
                }
                TIMER1 = ENCENDIDO;
                OUTPacket[0] = ESPERAR;
                OUTPacket[2] = TERMINADO;
                OUTPacket[1] = 0;
```

```
}//FIN if(OUTPacket[1] == OFFSET)
}//FIN if (OUTPacket[2] == 1)
}
}// END ProcessIO

/********************************************************************
* Function: void BlinkUSBStatus(void)
* PreCondition:    None
* Input:           None
* Output:          None
* Side Effects:    None
* Overview:        BlinkUSBStatus turns on and off LEDs
*                  corresponding to the USB device state.
* Note:            mLED macros can be found in HardwareProfile.h
*                  USBDeviceState is declared and updated in
*                  usb_device.c.
********************************************************************/
void BlinkUSBStatus(void)
{
   static WORD led_count=0;
   if(led_count == 0)led_count = 10000U;
   led_count--;
   if(USBSuspendControl == 1)
   {
      if(led_count==0)
      {
         PORTBbits.RB7 = !PORTBbits.RB7;
         PORTBbits.RB7 = PORTBbits.RB6 ;        // Both blink at the same time
      }//end if
   }
   else
   {
      if(USBDeviceState == DETACHED_STATE)
      {
         PORTBbits.RB7 = 0;
         PORTBbits.RB6 = 0;
      }
      else if(USBDeviceState == ATTACHED_STATE)
      {
         PORTBbits.RB7 = 1;
         PORTBbits.RB6 = 1;
      }
      else if(USBDeviceState == POWERED_STATE)
      {
         PORTBbits.RB7 = 1;
         PORTBbits.RB6 = 0;
      }
      else if(USBDeviceState == DEFAULT_STATE)
      {
         PORTBbits.RB7 = 0;
         PORTBbits.RB6 = 1;
      }
      else if(USBDeviceState == ADDRESS_STATE)
      {
         if(led_count == 0)
         {
            PORTBbits.RB7 = !PORTBbits.RB7;
            PORTBbits.RB6 = 0;
         }//end if
      }
      else if(USBDeviceState == CONFIGURED_STATE)
      {
         if(led_count==0)
         {
            PORTBbits.RB7 = !PORTBbits.RB7;
            PORTBbits.RB6 = !PORTBbits.RB7;       // Alternate blink
         }//end if
      }//end if(...)
   }//end if(UCONbits.SUSPND...)

}//end BlinkUSBStatus

//******************************************************************
// ** USB Callback Functions**************************************
//******************************************************************
// The USB firmware stack will call the callback functions USBCBxxx() in
//response to certain USB related events.  For example, if the host PC is
//powering down, it will stop sending out Start of Frame (SOF) packets to your -
//device.  In response to this, all USB devices are supposed to decrease their
//power consumption from the USB Vbus to <2.5mA each.  The USB module
//detects this condition (which according to the USB specifications is 3+ms of no
//Bus activity/SOF packets) and then calls the USBCBSuspend() function.  You
//should modify these callback functions to take appropriate Actions for each of
//these conditions.  For example, in the USBCBSuspend(), you may wish to add
//code that will decrease power consumption from Vbus to <2.5mA (such as by
//clock switching, turning off LEDs, putting the microcontroller to sleep, etc.).
//Then, in the USBCBWakeFromSuspend() function, you may then wish to add
//code that undoes the power saving things done in the USBCBSuspend()
//Function.
// The USBCBSendResume() function is special, in that the USB stack will not
//automatically call this function.  This function is meant to be called from the
//application firmware instead.  See the additional comments near the function.

/********************************************************************
* Function: void USBCBSuspend(void)
* PreCondition:    None
* Input:           None
* Output:          None
* Side Effects:    None
* Overview:        Call back that is invoked when a USB suspend is detected
* Note:            None
********************************************************************/
void USBCBSuspend(void)
{
//Example power saving code.  Insert appropriate code here for the desired
//application behavior.  If the microcontroller will be put to sleep, a
//process similar to that shown below may be used:
//ConfigureIOPinsForLowPower();
//SaveStateOfAllInterruptEnableBits();
//DisableAllInterruptEnableBits();
//EnableOnlyTheInterruptsWhichWillBeUsedToWakeTheMicro();
//should enable at least USBActivityIF as a wake source
//Sleep();
//RestoreStateOfAllPreviouslySavedInterruptEnableBits();
//Preferrably, this should be done in the USBCBWakeFromSuspend() function instead.
//RestoreIOPinsToNormal();
//Preferrably, this should be done in the USBCBWakeFromSuspend() function instead.
//IMPORTANT NOTE: Do not clear the USBActivityIF (ACTVIF) bit here.  This bit
//is cleared inside the usb_device.c file.  Clearing USBActivityIF here will cause
//things to not work as intended.
   #if defined(__C30__)
   #if 0
      U1EIR = 0xFFFF;
      U1IR = 0xFFFF;
      U1OTGIR = 0xFFFF;
      IFS5bits.USB1IF = 0;
      IEC5bits.USB1IE = 1;
      U1OTGIEbits.ACTVIE = 1;
      U1OTGIRbits.ACTVIF = 1;
      TRISA &= 0xFF3F;
      LATAbits.LATA6 = 1;
      Sleep();
      LATAbits.LATA6 = 0;
   #endif
   #endif
}

/********************************************************************
* Function: void _USB1Interrupt(void)
* PreCondition:    None
* Input:           None
* Output:          None
* Side Effects:    None
* Overview:    This function is called when the USB interrupt bit is set
*              In this example the interrupt is only used when the device
*              goes to sleep when it receives a USB suspend command
* Note:            None
********************************************************************/
#if 0
void __attribute__ ((interrupt)) _USB1Interrupt(void)
{
   #if !defined(self_powered)
      if(U1OTGIRbits.ACTVIF)
      {
         IEC5bits.USB1IE = 0;
         U1OTGIEbits.ACTVIE = 0;
         IFS5bits.USB1IF = 0;

         //USBClearInterruptFlag(USBActivityIFReg,USBActivityIFBitNum);
         USBClearInterruptFlag(USBIdleIFReg,USBIdleIFBitNum);
         //USBSuspendControl = 0;
      }
   #endif
}
#endif

/********************************************************************
* Function: void USBCBWakeFromSuspend(void)
* PreCondition:    None
* Input:           None
* Output:          None
* Side Effects:    None
* Overview:     The host may put USB peripheral devices in low power
*               suspend mode (by "sending" 3+ms of idle).  Once in suspend
*               mode, the host may wake the device back up by sending non-
*               idle state signalling.
*               This call back is invoked when a wakeup from USB suspend
*               is detected.
* Note:            None
********************************************************************/
void USBCBWakeFromSuspend(void)
{
// If clock switching or other power savings measures were taken when
// executing the USBCBSuspend() function, now would be a good time to
// switch back to normal full power run mode conditions.  The host allows
// a few milliseconds of wakeup time, after which the device must be
// fully back to normal, and capable of receiving and processing USB
```

```
// packets.  In order to do this, the USB module must receive proper
// clocking (IE: 48MHz clock must be available to SIE for full speed USB
// operation).
}
/********************************************************************
* Function: void USBCB_SOF_Handler(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:      The USB host sends out a SOF packet to full-speed
*                  devices every 1 ms. This interrupt may be useful
*                  for isochronous pipes. End designers should
*                  implement callback routine as necessary.
* Note:           None
********************************************************************/
void USBCB_SOF_Handler(void)
{
  // No need to clear UIRbits.SOFIF to 0 here.
  // Callback caller is already doing that.
}
/********************************************************************
* Function: void USBCBErrorHandler(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview: The purpose of this callback is mainly for
*               debugging during development. Check UEIR to see
*               which error causes the interrupt.
* Note:           None
********************************************************************/
void USBCBErrorHandler(void)
{
   // No need to clear UEIR to 0 here.
   // Callback caller is already doing that.
   // Typically, user firmware does not need to do anything special
   // if a USB error occurs.  For example, if the host sends an OUT
   // packet to your device, but the packet gets corrupted (ex:
   // because of a bad connection, or the user unplugs the
   // USB cable during the transmission) this will typically set
   // one or more USB error interrupt flags.  Nothing specific
   // needs to be done however, since the SIE will automatically
   // send a "NAK" packet to the host.  In response to this, the
   // host will normally retry to send the packet again, and no
   // data loss occurs.  The system will typically recover
   // automatically, without the need for application firmware
   // intervention.
   // Nevertheless, this callback function is provided, such as
   // for debugging purposes.
}

/********************************************************************
* Function: void USBCBCheckOtherReq(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:   When SETUP packets arrive from the host, some
*                  firmware must process the request and respond
*                  appropriately to fulfill the request.  Some of
*                  the SETUP packets will be for standard
*                  USB "chapter 9" (as in, fulfilling chapter 9 of
*                  the official USB specifications) requests, while
*                  others may be specific to the USB device class
*                  that is being implemented.  For example, a HID
*                  class device needs to be able to respond to
*                  "GET REPORT" type of requests.  This
*                  is not a standard USB chapter 9 request, and
*                  therefore not handled by usb_device.c.  Instead
*                  this request should be handled by class specific
*                  firmware, such as that contained in usb_function_hid.c.
* Note:           None
********************************************************************/
void USBCBCheckOtherReq(void)
{
}//end
/********************************************************************
* Function: void USBCBStdSetDscHandler(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:    The USBCBStdSetDscHandler() callback function is
*                  called when a SETUP, bRequest: SET_DESCRIPTOR request
*                  arrives.  Typically SET_DESCRIPTOR requests are
*                  not used in most applications, and it is
*                  optional to support this type of request.
* Note:           None
********************************************************************/
void USBCBStdSetDscHandler(void)
{
   // Must claim session ownership if supporting this request
}  //end

/********************************************************************
* Function:       void USBCBInitEP(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:    This function is called when the device becomes
*                  initialized, which occurs after the host sends a
*                  SET_CONFIGURATION (wValue not = 0) request.  This
*                  callback function should initialize the endpoints
*                  for the device's usage according to the current
*                  configuration.
* Note:           None
********************************************************************/
void USBCBInitEP(void)
{
USBEnableEndpoint(USBGEN_EP_NUM,USB_OUT_ENABLED|USB_IN_ENABL
ED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP);
USBGenericOutHandle =
USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,USBGEN_EP_SIZE);
}
/********************************************************************
* Function:       void USBCBSendResume(void)
* PreCondition:    None
* Input:          None
* Output:          None
* Side Effects:    None
* Overview:   The USB specifications allow some types of USB
*                  peripheral devices to wake up a host PC (such
*                  as if it is in a low power suspend to RAM state).
*                  This can be a very useful feature in some
*                  USB applications, such as an Infrared remote
*                  control Receiver.  If a user presses the "power"
*                  button on a remote control, it is nice that the
*                  IR receiver can detect this signalling, and then
*                  send a USB "command" to the PC to wake up.
*
*                  The USBCBSendResume() "callback" function is used
*                  to send this special USB signalling which wakes
*                  up the PC.  This function may be called by
*                  application firmware to wake up the PC.  This
*                  function should only be called when:
*
*                  1.  The USB driver used on the host PC supports
*                  the remote wakeup capability.
*                  2.  The USB configuration descriptor indicates
*                  the device is remote wakeup capable in the
*                  bmAttributes field.
*                  3.  The USB host PC is currently sleeping,
*                  and has previously sent your device a SET
*                  FEATURE setup packet which "armed" the
*                  remote wakeup capability.
*
*                  This callback should send a RESUME signal that
*                  has the period of 1-15ms.
*
* Note:        Interrupt vs. Polling
*              -Primary clock
*              -Secondary clock ***** MAKE NOTES ABOUT THIS *******
*               > Can switch to primary first by calling USBCBWakeFromSuspend()
*              The modifiable section in this routine should be changed
*              to meet the application needs. Current implementation
*              temporary blocks other functions from executing for a
*              period of 1-13 ms depending on the core frequency.
*
*              According to USB 2.0 specification section 7.1.7.7,
*              "The remote wakeup device must hold the resume signaling
*              for at lest 1 ms but for no more than 15 ms."
*              The idea here is to use a delay counter loop, using a
*              common value that would work over a wide range of core
*              frequencies.
*              That value selected is 1800. See table below:
*
*              ==========================================================
*              Core Freq(MHz)      MIP        RESUME Signal Period (ms)
*              ==========================================================
*                        48            12         1.05
*                         4             1         12.6
*              ==========================================================
*              * These timing could be incorrect when using code
*                optimization or extended instruction mode,
*                or when having other interrupts enabled.
*                Make sure to verify using the MPLAB SIM's Stopwatch
*                and verify the actual signal on an oscilloscope.
********************************************************************/
void USBCBSendResume(void)
{
   static WORD delay_count;
   USBResumeControl = 1;          // Start RESUME signaling
   delay_count = 1800U;           // Set RESUME line for 1-13 ms
   do
   {
```

```
delay_count--;
   }while(delay_count);
   USBResumeControl = 0;
}
/** EOF main.c *************************************************/
// ***********************************************************************
// ************* FUNCIONES DEL FIRMWARE *********************
// ***********************************************************************
/***********************************************************************
* Función: void INICIA_DDS(void)
* PreCondción: Ninguna
* Entrada: Ninguna
* Salida: Ninguna
* Efectos Secundarios: Ninguna
* Descripción General: Función la cuál inicializa el DDS.
* Nota: Ninguna
***********************************************************************/
void INICIA_DDS(void)
{
		OpenSPI(DDS);
		DDS_CS=0;
		WriteSPI(RESETM);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(RESETL);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1;
		DDS_CS=0;
		WriteSPI(0b01000000); /*MSBs de los LSBs del Registro de
		FREQ0*/
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(0b00000000);/*LSbs de los LSBs del REgistro de
		FREQ0*/
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1;
		DDS_CS=0;
		WriteSPI(0b01000000);/*MSBs de los MSBs del regitro de
		FREQ0*/
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(0b00000000);/*LSBs de los MSbs del registro de
		FREQ0*/
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1;
		DDS_CS=0;
		WriteSPI(0b11000000);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(0b00000000);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1;
		DDS_CS=0;
		WriteSPI(RESETT);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(RESETL);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1;
}
/***********************************************************************
* Función: void SEND(void)
* PreCondión:		Ninguna
* Entrada:		Ninguna
* Salida:		Ninguna
* Efectos Secundarios:	Ninguna
* Descripción General:	Funcion la cual envia el dato al DDS, dato el cual
*			viene directo de la PC con los datos de control y
*			dividido en BYTES.
* Nota: Funcion la cual probablemente se elimine, fue solo para prueba.
 ***********************************************************************/
void SEND(void)
{
		OpenSPI(DDS);
		DDS_CS=0; //Habilita el DDS para recibir datos
		WriteSPI(OUTPacket[5]);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(OUTPacket[4]);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS = 1; //Deshabilita el DDS
		DDS_CS = 0; //Habilita el DDS
		WriteSPI(OUTPacket[7]);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(OUTPacket[6]);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1; //Deshabilita el DDS
}
/***********************************************************************
* Función:		void SEND_DDS(unsigned long FRECUENCIA)
* PreCondión:		Ninguna
* Entrada:		unsigned long FRECUENCIA
* Salida:		Ninguna
* Efectos Secundarios:	Ninguna
* Descripción General:	Función que se encarga de enviar datos al REGISTRO de
*			frecuencia al DDS para controlar la frecuencia de salida.
*			Con sus respectivos bits de configuración. Manda 4bytes
*			de uno en uno, ya que recibe 16 bits en 16bits con los 2
*			MSB de control para el registro.
* Nota:			Ninguna
 ***********************************************************************/
void SEND_DDS(unsigned long FRECUENCIA)
{
		unsigned char LDEM = 0;
		unsigned char MDEM = 0;
		unsigned char LDEL = 0;
		unsigned char MDEL = 0;
		LDEL = MAKE8(FRECUENCIA,0);
		FRECUENCIA >>= 6;
		MDEL = MAKE8(FRECUENCIA,0);
		MDEL >>= 2;
		MDEL = MDEL + 0b01000000;
		LDEM = MAKE8(FRECUENCIA,1);
		MDEM = MAKE8(FRECUENCIA,2);
		MDEM = MDEM + 0b01000000;
		OpenSPI(DDS);
		DDS_CS=0; //Habilita el DDS para recibir datos.
		WriteSPI(MDEL);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(LDEL);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(MDEM);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		WriteSPI(LDEM);
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		DDS_CS=1; //Deshabilita el DDS.
}//Fin de SEND
/***********************************************************************
* Función:	unsigned char MAKE8(unsigned long var, unsigned char offset)
* PreCondión:		Ninguna
* Entrada:		unsigned long var, unsigned char offset
* Salida:		Returna x
* Efectos Secundarios: Ninguna
* Descripción General: Función que se encarga de hacer una variable unsigned longen
*			una variable de 8 Bits.
* Nota:			Ninguna
***********************************************************************/
 unsigned char MAKE8(unsigned long var, unsigned char offset)
{
		unsigned char x;
		x = ((var >> (offset*8)) & 0xFF);
		return x;
}

/***********************************************************************
* Función:		void TIMERS(void)
* PreCondión:		Ninguna
* Entrada:		Ninguna
* Salida:		Ninguna
* Efectos Secundarios:	Ninguna
* Descripción General:	Función que se encarga de inicializar los timers
*			para diferentes tareas, como:
*			- Actualizar el bus USB con USBDeviceTask cada
*			  100us po lo menos.
*			- Para el ciclo finito de la variable para el tiempo
*			  transitorio que se le deben de dar a los filtros.
*			- Para el tiempo de estabilizacion de los PKD01.
* Nota:			Ninguna
***********************************************************************/
void TIMERS(void)
{
		T1CON = 0b10110000;
		TMR1L = 0x69;
		Nop();
		TMR1H = 0xFF;
		T0CON = 0b00000110;
		TMR0L = 0xE4;
		TMR0H = 0x48;
		T2CON = 0b00001011;
		TMR2  = 0xFF;
		PR2   = 0x69;
}
/***********************************************************************
* Función:	void IINTERRUPCION(void)
* PreCondión:	Ninguna
* Entrada:	Ninguna
```

```
* Salida:		Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que se encarga de inicializar las
*				interrupciones de los timers para las diferentes
*				tareas, como:
*
*				- Actualizar el bus USB con USBDeviceTask cada
*				100us po lo menos.
*				- Para el ciclo finito de la variable para el tiempo
*				  transitorio que se le deben de dar a los filtros.
*				- Para el tiempo de estabilizacion de los PKD01.
 * Nota:			Ninguna
 ********************************************************************/
void IINTERRUPCION(void)
{
				RCONbits.IPEN = 1;
				INTCON2bits.TMR0IP = 0;
				IPR1bits.TMR1IP = 1;
				IPR1bits.TMR2IP = 0;
				INTCON = 0b11100000;
				PIR1bits.TMR1IF = 0;
				PIR1bits.TMR2IF = 0;
				PIE1bits.TMR1IE = 1;
				PIE1bits.TMR2IE = 1;
}
/*********************************************************************
* Función:		void INICIA_AMPLIS(void)
* PreCondión:		Ninguna
* Entrada:		Ninguna
* Salida:		Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que se encarga de inicializar los amplifica
*				dores LTC6912 para una ganancia de 0 dB o 1.
* Nota:			Ninguna
*********************************************************************/
void INICIA_AMPLIS(void)
{
		OpenSPI(LTC);
		AMP1_CS = 0;
		AMP2_CS = 0;
		WriteSPI(0b00010001);//17
		while(!PIR1bits.SSPIF);
		PIR1bits.SSPIF = 0;
		AMP1_CS = 1;
		AMP2_CS = 1;
		TEMPA = 0x00;
}
/*********************************************************************
* Función:		void WRITE_AMP(unsigned char STEP)
* PreCondión:		Ninguna
* Entrada:		unsigned char STEP
* Salida:		Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que manda al los amplificadores la
*				ganancia a programar, son 2 IC LTC con 2
*				amplificadores cada uno.
* Nota:			Ninguna
*********************************************************************/
void WRITE_AMP(unsigned char STEP)
{
		OpenSPI(LTC);
		if((TEMPA == 0) && (STEP == UP))
		{
				AMP1_CS = 0;
				WriteSPI(0b00010111);//23
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP1_CS = 1;
				//TEMPA++;
				TEMPA = 3;
		}
		else if((TEMPA == 1) && (STEP == DOWN))
		{
				AMP1_CS = 0;
				WriteSPI(0b00010001);//17
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP1_CS = 1;
				TEMPA--;
		}
		else if ((TEMPA == 3) && (STEP == UP))
		{
				AMP1_CS = 0;
				WriteSPI(0b01110111);//119
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP1_CS = 1;
				//TEMPA++;
				TEMPA = 6;
		}
		else if ((TEMPA == 2) && (STEP == DOWN))
		{
				AMP1_CS = 0;
				WriteSPI(0b00010111);//23
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP1_CS = 1;
				TEMPA--;
		}
		else if ((TEMPA == 6) && (STEP == UP))
		{
				AMP2_CS = 0;
				WriteSPI(0b00010111);//23
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP2_CS = 1;
				//TEMPA++;
				TEMPA = 9;
		}
		else if ((TEMPA == 3) && (STEP == DOWN))
		{
				AMP2_CS = 0;
				WriteSPI(0b00010001);//17
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP2_CS = 1;
				TEMPA--;
		}
		else if ((TEMPA == 9) && (STEP == UP))
		{
				AMP2_CS = 0;
				WriteSPI(0b01110111);//119
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP2_CS = 1;
				//TEMPA++;
				TEMPA = 12;
		}
		else if ((TEMPA == 4) && (STEP == DOWN))
		{
				AMP2_CS = 0;
				WriteSPI(0b00010111);//23
				while(!PIR1bits.SSPIF);
				PIR1bits.SSPIF = 0;
				AMP2_CS = 1;
				TEMPA--;
		}
		else
				Nop();
}
/*********************************************************************
* Función:		void TIEMPO_A(void)
* PreCondión:		Ninguna
* Entrada:		Ninguna
* Salida:		Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que manda al timer los tiempos para los tiempos
*				de diferentes frecuencias y que se pueda estabilizar
*				el voltaje proveniente del PKD01 del filtro.
* Nota:			Ninguna
*********************************************************************/
void TIEMPO_A(void)
{
		if(f <= 10)
		{
				TMR0L = 0xE4;
				TMR0H = 0x48;
				LTIEMPO = 0xE4;
				HTIEMPO = 0x48;
		}
		else if(f <= 20)
		{
				TMR0L = 0x83;
				TMR0H = 0x6D;
				LTIEMPO = 0x83;
				HTIEMPO = 0x6D;
		}
		else if(f <= 30)
		{
				TMR0L = 0x22;
				TMR0H = 0x92;
				LTIEMPO = 0x22;
				HTIEMPO = 0x92;
		}
		else if(f <= 40)
		{
				TMR0L = 0x71;
				TMR0H = 0xA4;
				LTIEMPO = 0x71;
				HTIEMPO = 0xA4;
		}
		else if(f <= 50)
		{
				TMR0L = 0xC1;
				TMR0H = 0xB6;
				LTIEMPO = 0xC1;
				HTIEMPO = 0xB6;
```

```
		}
		else if(f <= 61)
		{
			TMR0L = 0x60;
			TMR0H = 0xDB;
			LTIEMPO = 0x60;
			HTIEMPO = 0xDB;
		}
}
/*********************************************************************
* Función:			void TIEMPO_R(void)
* PreCondión:			Ninguna
* Entrada:			Ninguna
* Salida:			Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que manda al timer los tiempos para los
*				tiempos de diferentes frecuencias y que se pueda
*				estabilizar el voltaje proveniente del PKD01 del
*				filtro.
* Nota:				Hace lo mismo que la funcion void
*				TIEMPO_B(void) salvo que esta es para el analisis
*				del rango el cual no tiene las frecuencias definidas.
*********************************************************************/
void TIEMPO_R(void)
{
		if(FINICIAL.VAL <= 128)
		{
			TMR0L = 0xE4;
			TMR0H = 0x48;
			LTIEMPO = 0xE4;
			HTIEMPO = 0x48;
		}
		else if(FINICIAL.VAL <= 1073)
		{
			TMR0L = 0x83;
			TMR0H = 0x6D;
			LTIEMPO = 0x83;
			HTIEMPO = 0x6D;
		}
		else if(FINICIAL.VAL <= 10737)
		{
			TMR0L = 0x22;
			TMR0H = 0x92;
			LTIEMPO = 0x22;
			HTIEMPO = 0x92;
		}
		else if(FINICIAL.VAL <= 96636)
		{
			TMR0L = 0x71;
			TMR0H = 0xA4;
			LTIEMPO = 0x71;
			HTIEMPO = 0xA4;
		}
		else if(FINICIAL.VAL <= 214748)
		{
			TMR0L = 0xC1;
			TMR0H = 0xB6;
			LTIEMPO = 0xC1;
			HTIEMPO = 0xB6;
		}
		else if(FINICIAL.VAL <= 1073741)
		{
			TMR0L = 0x60;
			TMR0H = 0xDB;
			LTIEMPO = 0x60;
			HTIEMPO = 0xDB;
		}
}
/*********************************************************************
* Función:			void LEER_ADC(void)
* PreCondión:			Ninguna
* Entrada:			Ninguna
* Salida:			Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que se utiliza para reducir los ruidos
*				que se puedan generar en la lectura del ADC.
* Nota:
*********************************************************************/
void LEER_ADC(void)
{
		a = 0x00;
		ADC_NOISE = 0x0000;
		for(; a <= 15; a++)
		{
			ADCGO_DONE = GO;

			while(ADCGO_DONE);
			ADCBUFFER.v[0] = ADRESL;
			ADCBUFFER.v[1] = ADRESH;
			ADC_NOISE += ADCBUFFER.VAL;
		}
		ADC_NOISE >>= 4;
		ADCBUFFER.VAL = ADC_NOISE;
}

/*********************************************************************
* Función:			void CONTROL_LTC(void)
* PreCondión:			Ninguna
* Entrada:			Ninguna
* Salida:			Ninguna
* Efectos Secundarios:		Ninguna
* Descripción General:		Función que se utiliza para el control de los LTC
*				para que el ADC tenga mas resolucion en la PC.
* Nota:
*********************************************************************/
void CONTROL_LTC(void)
{
		OpenSPI(LTC);
		if(ADCBUFFER.VAL >= RANGOADCALTO)
		{
			INICIA_AMPLIS();
			Nop(); Nop(); Nop();
			PEAKOUT = DETECT;
			DELAY_PKD01 = TRUE;;
			TMR_PKD01 = ENCENDIDO;
			while(DELAY_PKD01);
			TIEMPO = 1;
			TMR_TRANS = ENCENDIDO;
			while(TIEMPO);
			TIEMPO = 1;
			TMR_TRANS = ENCENDIDO;
			while(TIEMPO);
			LEER_ADC();
			PEAKOUT = RESET;
			DELAY_PKD01 = TRUE;;
			TMR_PKD01 = ENCENDIDO;
			while(DELAY_PKD01);
		}
		else if((ADCBUFFER.VAL == 0) && (TEMPA == 12))
		{
			ADCBUFFER.VAL = 1;
		}
		else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 11))
		{
			AMP2_CS = 0;
			WriteSPI(0b01110111);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 12;
		}
		else if((ADCBUFFER.VAL <= CUATRO2) && (TEMPA == 10))
		{
			AMP2_CS = 0;
			WriteSPI(0b01100110);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 11;
		}
		else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 9))
		{
			AMP2_CS = 0;
			WriteSPI(0b01010101);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 10;
		}
		else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 8))
		{
			AMP2_CS = 0;
			WriteSPI(0b01000100);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 9;
		}
		else if((ADCBUFFER.VAL <= CUATRO2) && ( TEMPA == 7))
		{
			AMP2_CS = 0;
			WriteSPI(0b00110011);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 8;
		}
		else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 6))
		{
			AMP2_CS = 0;
			WriteSPI(0b00100010);
			while(!PIR1bits.SSPIF);
			PIR1bits.SSPIF = 0;
			AMP2_CS = 1;
			TEMPA = 7;
		}
		else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 5))
		{
			AMP1_CS = 0;
```

```
                WriteSPI(0b01110111);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 6;
        }
        else if((ADCBUFFER.VAL <= CUATRO2) && (TEMPA == 4))
        {
                AMP1_CS = 0;
                WriteSPI(0b01100110);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 5;
        }
        else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 3))
        {
                AMP1_CS = 0;
                WriteSPI(0b01010101);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 4;
        }
        else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 2))
        {
                AMP1_CS = 0;
                WriteSPI(0b01000100);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 3;
        }
        else if((ADCBUFFER.VAL <= CUATRO2) && (TEMPA == 1))
        {
                AMP1_CS = 0;
                WriteSPI(0b00110011);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 2;
        }
        else if((ADCBUFFER.VAL == 0) && (TEMPA == 0))
        {
                WRITE_AMP(UP);
                while(!ADCBUFFER.VAL)
                {
                        WRITE_AMP(UP);
                        LEER_ADC();
                        if(TEMPA == 12)
                        {
                           ADCBUFFER.VAL = 1;
                           break;
                        }
                }
        }
        else if((ADCBUFFER.VAL <= CUATROC) && (TEMPA == 0))
        {
                AMP1_CS = 0;
                WriteSPI(0b01010101);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 4;
        }
        else if ((ADCBUFFER.VAL <= CIEN) && (TEMPA == 0))
        {
                AMP1_CS = 0;
                WriteSPI(0b01000100);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 3;
        }
        else if((ADCBUFFER.VAL <= VCINCO) && (TEMPA == 0))
        {
                AMP1_CS = 0;
                WriteSPI(0b00110011);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 2;
        }
        else if((ADCBUFFER.VAL <= CUATRO) && (TEMPA == 0))
        {
                AMP1_CS = 0;
                WriteSPI(0b00100010);
                while(!PIR1bits.SSPIF);
                PIR1bits.SSPIF = 0;
                AMP1_CS = 1;
                TEMPA = 1;
        }
}
```

ANEXO 1

Hoja de especificaciones del DDS AD9833

Low Power 20 mW 2.3 V to 5.5 V Programmable Waveform Generator

AD9833

FEATURES

Digitally Programmable Frequency and Phase
20 mW Power Consumption at 3 V
0 MHz to 12.5 MHz Output Frequency Range
28-Bit Resolution (0.1 Hz @ 25 MHz Ref Clock)
Sinusoidal/Triangular/Square Wave Outputs
2.3 V to 5.5 V Power Supply
No External Components Required
3-Wire SPI® Interface
Extended Temperature Range: –40°C to +105°C
Power-Down Option
10-Lead MSOP Package

APPLICATIONS

Frequency Stimulus/Waveform Generation
Liquid and Gas Flow Measurement
Sensory Applications—Proximity, Motion, and Defect Detection
Line Loss/Attenuation
Test and Medical Equipment
Sweep/Clock Generators
TDR

GENERAL DESCRIPTION

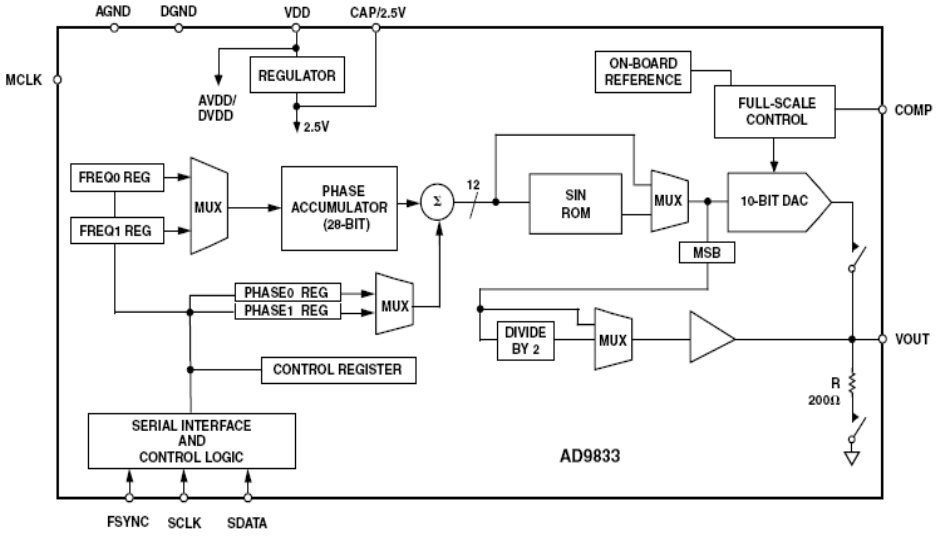
The AD9833 is a low power programmable waveform generator capable of producing sine, triangular, and square wave outputs. Waveform generation is required in various types of sensing, actuation, and time domain reflectometry applications. The output frequency and phase are software programmable, allowing easy tuning. No external components are needed. The frequency registers are 28 bits; with a 25 MHz clock rate, resolution of 0.1 Hz can be achieved. Similarly, with a 1 MHz clock rate, the AD9833 can be tuned to 0.004 Hz resolution.

The AD9833 is written to via a 3-wire serial interface. This serial interface operates at clock rates up to 40 MHz and is compatible with DSP and microcontroller standards. The device operates with a power supply from 2.3 V to 5.5 V.

The AD9833 has a power-down function (SLEEP). This allows sections of the device that are not being used to be powered down, thus minimizing the current consumption of the part, e.g., the DAC can be powered down when a clock output is being generated.

The AD9833 is available in a 10-lead MSOP package.

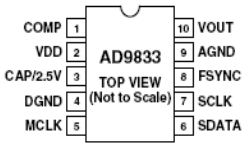
FUNCTIONAL BLOCK DIAGRAM



REV. A

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703

PIN CONFIGURATION



PIN FUNCTION DESCRIPTIONS

Pin Number	Mnemonic	Function
Power Supply		
2	VDD	Positive Power Supply for the Analog and the Digital Interface Sections. The on-board 2.5 V regulator is also supplied from VDD. VDD can have a value from 2.3 V to 5.5 V. A 0.1 µF and a 10 µF decoupling capacitor should be connected between VDD and AGND.
3	CAP/2.5 V	The digital circuitry operates from a 2.5 V power supply. This 2.5 V is generated from VDD using an on-board regulator (when VDD exceeds 2.7 V). The regulator requires a decoupling capacitor of typically 100 nF, which is connected from CAP/2.5 V to DGND. If VDD is equal to or less than 2.7 V, CAP/2.5 V should be tied directly to VDD.
4	DGND	Digital Ground.
9	AGND	Analog Ground.
Analog Signal and Reference		
1	COMP	A DAC Bias Pin. This pin is used for decoupling the DAC bias voltage.
10	VOUT	Voltage Output. The analog and digital output from the AD9833 is available at this pin. An external load resistor is not required because the device has a 200 Ω resistor on board.
Digital Interface and Control		
5	MCLK	Digital Clock Input. DDS output frequencies are expressed as a binary fraction of the frequency of MCLK. The output frequency accuracy and phase noise are determined by this clock.
6	SDATA	Serial Data Input. The 16-bit serial data-word is applied to this input.
7	SCLK	Serial Clock Input. Data is clocked into the AD9833 on each falling SCLK edge.
8	FSYNC	Active Low Control Input. This is the frame synchronization signal for the input data. When FSYNC is taken low, the internal logic is informed that a new word is being loaded into the device.

AD9833

TERMINOLOGY

Integral Nonlinearity
This is the maximum deviation of any code from a straight line passing through the endpoints of the transfer function. The endpoints of the transfer function are zero scale, a point 0.5 LSB below the first code transition (000 . . . 00 to 000 . . . 01), and full scale, a point 0.5 LSB above the last code transition (111 . . . 10 to 111 . . . 11). The error is expressed in LSBs.

Differential Nonlinearity
This is the difference between the measured and ideal 1 LSB change between two adjacent codes in the DAC. A specified differential nonlinearity of ±1 LSB maximum ensures monotonicity.

Output Compliance
The output compliance refers to the maximum voltage that can be generated at the output of the DAC to meet the specifications. When voltages greater than that specified for the output compliance are generated, the AD9833 may not meet the specifications listed in the data sheet.

Spurious-Free Dynamic Range
Along with the frequency of interest, harmonics of the fundamental frequency and images of these frequencies are present at the output of a DDS device. The spurious-free dynamic range (SFDR) refers to the largest spur or harmonic present in the band of interest. The wideband SFDR gives the magnitude of the largest harmonic or spur relative to the magnitude of the fundamental frequency in the 0 to Nyquist bandwidth. The narrow-band SFDR gives the attenuation of the largest spur or harmonic in a bandwidth of ±200 kHz about the fundamental frequency.

Total Harmonic Distortion
Total harmonic distortion (THD) is the ratio of the rms sum of harmonics to the rms value of the fundamental. For the AD9833, *THD* is defined as

$$\text{THD} = 20 \log \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 + V_5^2 + V_6^2}}{V_1}$$

where V_1 is the rms amplitude of the fundamental and V_2, V_3, V_4, V_5, and V_6 are the rms amplitudes of the second through sixth harmonics.

Signal-to-Noise Ratio (SNR)
SNR is the ratio of the rms value of the measured output signal to the rms sum of all other spectral components below the Nyquist frequency. The value for SNR is expressed in decibels.

Clock Feedthrough
There will be feedthrough from the MCLK input to the analog output. Clock feedthrough refers to the magnitude of the MCLK signal relative to the fundamental frequency in the AD9833's output spectrum.

THEORY OF OPERATION

Sine waves are typically thought of in terms of their magnitude form a(t) = sin(ωt). However, these are nonlinear and not easy to generate except through piecewise construction. On the other hand, the angular information is linear in nature. That is, the phase angle rotates through a fixed angle for each unit of time. The angular rate depends on the frequency of the signal by the traditional rate of $\omega = 2\pi f$.

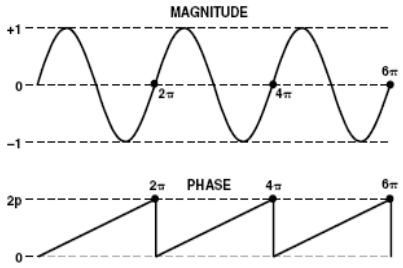


Figure 4. Sine Wave

Knowing that the phase of a sine wave is linear and given a reference interval (clock period), the phase rotation for that period can be determined.

$$\Delta Phase = \omega \Delta t$$

Solving for ω

$$\omega = \Delta Phase / \Delta t = 2\pi f$$

Solving for *f* and substituting the reference clock frequency for the reference period $\left(1/f_{MCLK} = \Delta t\right)$

$$f = \Delta Phase \times f_{MCLK} / 2\pi$$

The AD9833 builds the output based on this simple equation. A simple DDS chip can implement this equation with three major subcircuits: numerically controlled oscillator + phase modulator, SIN ROM, and digital-to-analog converter.

Each of these subcircuits is discussed in the following section.

CIRCUIT DESCRIPTION

The AD9833 is a fully integrated direct digital synthesis (DDS) chip. The chip requires one reference clock, one low precision resistor, and decoupling capacitors to provide digitally created sine waves up to 12.5 MHz. In addition to the generation of this RF signal, the chip is fully capable of a broad range of simple and complex modulation schemes. These modulation schemes are fully implemented in the digital domain, allowing accurate and simple realization of complex modulation algorithms using DSP techniques.

The internal circuitry of the AD9833 consists of the following main sections: a numerically controlled oscillator (NCO), frequency and phase modulators, SIN ROM, a digital-to-analog converter, and a regulator.

Numerically Controlled Oscillator Plus Phase Modulator
This consists of two frequency select registers, a phase accumulator, two phase offset registers, and a phase offset adder. The main component of the NCO is a 28-bit phase accumulator. Continuous time signals have a phase range of 0 to 2π. Outside this range of numbers, the sinusoid functions repeat themselves in a periodic manner. The digital implementation is no different. The accumulator simply scales the range of phase numbers into a multibit digital word. The phase accumulator in the AD9833 is implemented with 28 bits. Therefore, in the AD9833, $2\pi = 2^{28}$. Likewise, the $\Delta Phase$ term is scaled into this range of numbers $0 < \Delta Phase < 2^{28} - 1$. With these substitutions, the previous equation becomes

$$f = \Delta Phase \times f_{MCLK} / 2^{28}$$

where $0 < \Delta Phase < 2^{28} - 1$.

The input to the phase accumulator can be selected either from the FREQ0 register or FREQ1 register, and is controlled by the FSELECT bit. NCOs inherently generate continuous phase signals, thus avoiding any output discontinuity when switching between frequencies.

Following the NCO, a phase offset can be added to perform phase modulation using the 12-bit phase registers. The contents of one of these phase registers is added to the most significant bits of the NCO. The AD9833 has two phase registers; their resolution is $2\pi/4096$.

SIN ROM
To make the output from the NCO useful, it must be converted from phase information into a sinusoidal value. Since phase information maps directly into amplitude, the SIN ROM uses the digital phase information as an address to a look-up table and converts the phase information into amplitude. Although the NCO contains a 28-bit phase accumulator, the output of the NCO is truncated to 12 bits. Using the full resolution of the phase accumulator is impractical and unnecessary, as this would require a look-up table of 2^{28} entries. It is necessary only to have sufficient phase resolution such that the errors due to truncation are smaller than the resolution of the 10-bit DAC. This requires that the SIN ROM have two bits of phase resolution more than the 10-bit DAC.

The SIN ROM is enabled using the MODE bit (D1) in the control register. This is explained further in Table XI.

Digital-to-Analog Converter
The AD9833 includes a high impedance current source 10-bit DAC. The DAC receives the digital words from the SIN ROM and converts them into the corresponding analog voltages.

The DAC is configured for single-ended operation. An external load resistor is not required since the device has a 200 Ω resistor on board. The DAC generates an output voltage of typically 0.6 V p-p.

Regulator
VDD provides the power supply required for the analog section and the digital section of the AD9833. This supply can have a value of 2.3 V to 5.5 V.

The internal digital section of the AD9833 is operated at 2.5 V. An on-board regulator steps down the voltage applied at VDD to 2.5 V. When the applied voltage at the VDD pin of the AD9833 is equal to or less than 2.7 V, the CAP/2.5 V and VDD pins should be tied together, thus bypassing the on-board regulator.

FUNCTIONAL DESCRIPTION

Serial Interface
The AD9833 has a standard 3-wire serial interface that is compatible with SPI®, QSPI™, MICROWIRE™, and DSP interface standards.

Data is loaded into the device as a 16-bit word under the control of a serial clock input, SCLK. The timing diagram for this operation is given in Figure 3.

The FSYNC input is a level triggered input that acts as a frame synchronization and chip enable. Data can be transferred into the device only when FSYNC is low. To start the serial data transfer, FSYNC should be taken low, observing the minimum FSYNC to SCLK falling edge setup time, t_7. After FSYNC goes low, serial data will be shifted into the device's input shift register on the falling edges of SCLK for 16 clock pulses. FSYNC may be taken high after the 16th falling edge of SCLK, observing the minimum SCLK falling edge to FSYNC rising edge time, t_8. Alternatively, FSYNC can be kept low for a multiple of 16 SCLK pulses and then brought high at the end of the data transfer. In this way, a continuous stream of 16-bit words can be loaded while FSYNC is held low, FSYNC only going high after the 16th SCLK falling edge of the last word loaded.

The SCLK can be continuous, or alternatively the SCLK can idle high or low between write operations but must be high when FSYNC goes low (t_{11}).

Powering Up the AD9833
The flow chart in Figure 7 shows the operating routine for the AD9833. When the AD9833 is powered up, the part should be reset. This will reset appropriate internal registers to zero to provide an analog output of midscale. To avoid spurious DAC outputs while the AD9833 is being initialized, the RESET bit should be set to 1 until the part is ready to begin generating an output. RESET does not reset the phase, frequency, or control registers. These registers will contain invalid data, and therefore should be set to a known value by the user. The RESET bit should then be set to 0 to begin generating an output. The data will appear on the DAC output eight MCLK cycles after RESET is set to 0.

Latency
Associated with each asynchronous write operation in the AD9833 is a latency. If a selected frequency/phase register is loaded with a new word, there is a delay of seven to eight MCLK cycles before the analog output will change. (There is an uncertainty of one MCLK cycle, as it depends on the position of the MCLK rising edge when the data is loaded into the destination register.)

AD9833

Control Register

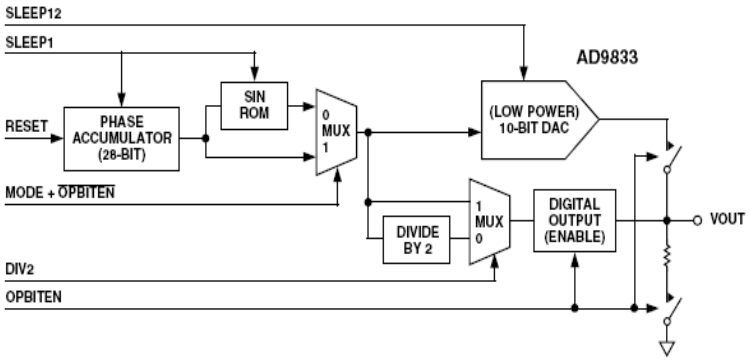
The AD9833 contains a 16-bit control register that sets up the AD9833 as the user wants to operate it. All control bits, except MODE, are sampled on the internal negative edge of MCLK.

Table II describes the individual bits of the control register. The different functions and the various output options from the AD9833 are described in more detail in the section following Table II.

To inform the AD9833 that the contents of the control register will be altered, D15 and D14 must be set to 0 as shown below.

Table I. Control Register

D15	D14	D13 D0
0	0	CONTROL BITS



DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	B28	HLB	FSELECT	PSELECT	0	RESET	SLEEP1	SLEEP12	OPBITEN	0	DIV2	0	MODE	0

Figure 5. Function of Control Bits

Table II. Description of Bits in the Control Register

Bit	Name	Function
D13	B28	Two write operations are required to load a complete word into either of the frequency registers. B28 = 1 allows a complete word to be loaded into a frequency register in two consecutive writes. The first write contains the 14 LSBs of the frequency word, and the next write will contain the 14 MSBs. The first two bits of each 16-bit word define the frequency register to which the word is loaded and should, therefore, be the same for both of the consecutive writes. Refer to Table IV for the appropriate addresses. The write to the frequency register occurs after both words have been loaded, so the register never holds an intermediate value. An example of a complete 28-bit write is shown in Table V. When B28 = 0, the 28-bit frequency register operates as two 14-bit registers, one containing the 14 MSBs and the other containing the 14 LSBs. This means that the 14 MSBs of the frequency word can be altered independent of the 14 LSBs, and vice versa. To alter the 14 MSBs or the 14 LSBs, a single write is made to the appropriate frequency address. The control bit D12 (HLB) informs the AD9833 whether the bits to be altered are the 14 MSBs or 14 LSBs.
D12	HLB	This control bit allows the user to continuously load the MSBs or LSBs of a frequency register while ignoring the remaining 14 bits. This is useful if the complete 28-bit resolution is not required. HLB is used in conjunction with D13 (B28). This control bit indicates whether the 14 bits being loaded are being transferred to the 14 MSBs or 14 LSBs of the addressed frequency register. D13 (B28) must be set to 0 to be able to change the MSBs and LSBs of a frequency word separately. When D13 (B28) = 1, this control bit is ignored. HLB = 1 allows a write to the 14 MSBs of the addressed frequency register. HLB = 0 allows a write to the 14 LSBs of the addressed frequency register.
D11	FSELECT	The FSELECT bit defines whether the FREQ0 register or the FREQ1 register is used in the phase accumulator.
D10	PSELECT	The PSELECT bit defines whether the PHASE0 register or the PHASE1 register data is added to the output of the phase accumulator.
D9	Reserved	This bit should be set to 0.
D8	RESET	RESET = 1 resets internal registers to 0, which corresponds to an analog output of midscale. RESET = 0 disables RESET. This function is explained further in Table IX.
D7	SLEEP1	When SLEEP1 = 1, the internal MCLK clock is disabled, the DAC output will remain at its present value as the NCO is no longer accumulating. When SLEEP1 = 0, MCLK is enabled. This function is explained further in Table X.

Table II. Description of Bits in the Control Register (continued)

Bit	Name	Function
D6	SLEEP12	SLEEP12 = 1 powers down the on-chip DAC. This is useful when the AD9833 is used to output the MSB of the DAC data. SLEEP12 = 0 implies that the DAC is active. This function is explained further in Table X.
D5	OPBITEN	The function of this bit, in association with D1 (MODE), is to control what is output at the VOUT pin. This is explained further in Table XI. When OPBITEN = 1, the output of the DAC is no longer available at the VOUT pin. Instead, the MSB (or MSB/2) of the DAC data is connected to the VOUT pin. This is useful as a coarse clock source. The bit DIV2 controls whether it is the MSB or MSB/2 that is output. When OPBITEN = 0, the DAC is connected to VOUT. The MODE bit determines whether it is a sinusoidal or a ramp output that is available.
D4	Reserved	This bit must be set to 0.
D3	DIV2	DIV2 is used in association with D5 (OPBITEN). This is explained further in Table XI. When DIV2 = 1, the MSB of the DAC data is passed directly to the VOUT pin. When DIV2 = 0, the MSB/2 of the DAC data is output at the VOUT pin.
D2	Reserved	This bit must be set to 0.
D1	MODE	This bit is used in association with OPBITEN (D5). The function of this bit is to control what is output at the VOUT pin when the on-chip DAC is connected to VOUT. This bit should be set to 0 if the control bit OPBITEN = 1. This is explained further in Table XI. When MODE = 1, the SIN ROM is bypassed, resulting in a triangle output from the DAC. When MODE = 0, the SIN ROM is used to convert the phase information into amplitude information, which results in a sinusoidal signal at the output.
D0	Reserved	This bit must be set to 0.

Frequency and Phase Registers

The AD9833 contains two frequency registers and two phase registers, which are described in Table III.

Table III. Frequency/Phase Registers

Register	Size	Description
FREQ0	28 Bits	Frequency Register 0. When the FSELECT bit = 0, this register defines the output frequency as a fraction of the MCLK frequency.
FREQ1	28 Bits	Frequency Register 1. When the FSELECT bit = 1, this register defines the output frequency as a fraction of the MCLK frequency.
PHASE0	12 Bits	Phase Offset Register 0. When the PSELECT bit = 0, the contents of this register are added to the output of the phase accumulator.
PHASE1	12 Bits	Phase Offset Register 1. When the PSELECT bit = 1, the contents of this register are added to the output of the phase accumulator.

The analog output from the AD9833 is

$$f_{MCLK}/2^{28} \times FREQREG$$

where *FREQREG* is the value loaded into the selected frequency register. This signal will be phase shifted by

$$2\pi/4096 \times PHASEREG$$

where *PHASEREG* is the value contained in the selected phase register. Consideration must be given to the relationship of the selected output frequency and the reference clock frequency to avoid unwanted output anomalies.

The flow chart in Figure 9 shows the routine for writing to the frequency and phase registers of the AD9833.

Writing to a Frequency Register

When writing to a frequency register, Bits D15 and D14 give the address of the frequency register.

Table IV. Frequency Register Bits

D15	D14	D13		D0
0	1	MSB	14 FREQ0 REG Bits	LSB
1	0	MSB	14 FREQ1 REG Bits	LSB

If the user wants to change the entire contents of a frequency register, two consecutive writes to the same address must be performed since the frequency registers are 28 bits wide. The first write will contain the 14 LSBs, while the second write will contain the 14 MSBs. For this mode of operation, the control bit B28 (D13) should be set to 1. An example of a 28-bit write is shown in Table V.

Table V. Writing 00FC00 to FREQ0 REG

SDATA Input	Result of Input Word
0010 0000 0000 0000	Control Word Write (D15, D14 = 00), B28 (D13) = 1, HLB (D12) = X
0100 0000 0000 0000	FREQ0 REG Write (D15, D14 = 01), 14 LSBs = 0000
0100 0000 0011 1111	FREQ0 REG Write (D15, D14 = 01), 14 MSBs = 003F

In some applications, the user does not need to alter all 28 bits of the frequency register. With coarse tuning, only the 14 MSBs are altered, while with fine tuning, only the 14 LSBs are altered. By setting the control bit B28 (D13) to 0, the 28-bit frequency register operates as two, 14-bit registers, one containing the 14 MSBs and the other containing the 14 LSBs. This means that the 14 MSBs of the frequency word can be altered independent of the 14 LSBs, and vice versa. Bit HLB (D12) in the control register identifies which 14 bits are being altered. Examples of this are shown in Tables VI and VII.

Table VI. Writing 3FFF to the 14 LSBs of FREQ1 REG

SDATA Input	Result of Input Word
0000 0000 0000 0000	Control Word Write (D15, D14 = 00), B28 (D13) = 0; HLB (D12) = 0, i.e. LSBs
1011 1111 1111 1111	FREQ1 REG Write (D15, D14 = 10), 14 LSBs = 3FFF

Table VII. Writing 00FF to the 14 MSBs of FREQ0 REG

SDATA Input	Result of Input Word
0001 0000 0000 0000	Control Word Write (D15, D14 = 00), B28 (D13) = 0, HLB (D12) = 1, i.e., MSBs
0100 0000 1111 1111	FREQ0 REG Write (D15, D14 = 01), 14 MSBs = 00FF

Writing to a Phase Register
When writing to a phase register, Bits D15 and D14 are set to 11. Bit D13 identifies which phase register is being loaded.

Table VIII. Phase Register Bits

D15	D14	D13	D12	D11		D0
1	1	0	X	MSB	12 PHASE0 Bits	LSB
1	1	1	X	MSB	12 PHASE1 Bits	LSB

RESET Function
The RESET function resets appropriate internal registers to 0 to provide an analog output of midscale. RESET does not reset the phase, frequency, or control registers. When the AD9833 is powered up, the part should be reset. To reset the AD9833, set the RESET bit to 1. To take the part out of reset, set the bit to 0. A signal will appear at the DAC to output eight MCLK cycles after RESET is set to 0.

Table IX. Applying RESET

RESET Bit	Result
0	No Reset Applied
1	Internal Registers Reset

SLEEP Function
Sections of the AD9833 that are not in use can be powered down to minimize power consumption. This is done using the SLEEP function. The parts of the chip that can be powered down are the internal clock and the DAC. The bits required for the SLEEP function are outlined in Table X.

Table X. Applying the SLEEP Function

SLEEP1 Bit	SLEEP12 Bit	Result
0	0	No Power-Down
0	1	DAC Powered Down
1	0	Internal Clock Disabled
1	1	Both the DAC Powered Down and the Internal Clock Disabled

DAC Powered Down
This is useful when the AD9833 is used to output the MSB of the DAC data only. In this case, the DAC is not required so it can be powered down to reduce power consumption.

Internal Clock Disabled
When the internal clock of the AD9833 is disabled, the DAC output will remain at its present value as the NCO is no longer accumulating. New frequency, phase, and control words can be written to the part when the SLEEP1 control bit is active. The synchronizing clock is still active, which means that the selected frequency and phase registers can also be changed using the control bits. Setting the SLEEP1 bit to 0 enables the MCLK. Any changes made to the registers while SLEEP1 was active will be seen at the output after a certain latency.

VOUT Pin
The AD9833 offers a variety of outputs from the chip, all of which are available from the VOUT pin. The choice of outputs are the MSB of the DAC data, a sinusoidal output, or a triangle output.

The OPBITEN (D5) and MODE (D1) bits in the control register are used to decide which output is available from the AD9833. This is explained further below and also in Table XI.

MSB of the DAC Data
The MSB of the DAC data can be output from the AD9833. By setting the OPBITEN (D5) control bit to 1, the MSB of the DAC data is available at the VOUT pin. This is useful as a coarse clock source. This square wave can also be divided by two before being output. The DIV2 (D3) bit in the control register controls the frequency of this output from the VOUT pin.

Sinusoidal Output
The SIN ROM is used to convert the phase information from the frequency and phase registers into amplitude information that results in a sinusoidal signal at the output. To have a sinusoidal output from the VOUT pin, set the MODE (D1) bit to 0 and the OPBITEN (D5) bit to 0.

Triangle Output
The SIN ROM can be bypassed so that the truncated digital output from the NCO is sent to the DAC. In this case, the output is no longer sinusoidal. The DAC will produce a 10-bit linear triangular function. To have a triangle output from the VOUT pin, set the MODE (D1) bit = 1.

Note that the SLEEP12 bit must be 0 (i.e., the DAC is enabled) when using this pin.

Table XI. Various Outputs from VOUT

OPBITEN Bit	MODE Bit	DIV2 Bit	VOUT Pin
0	0	X	Sinusoid
0	1	X	Triangle
1	0	0	DAC Data MSB/2
1	0	1	DAC Data MSB
1	1	X	Reserved

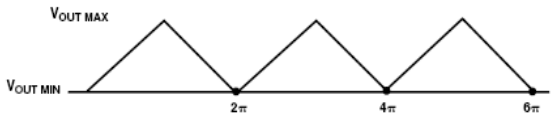


Figure 6. Triangle Output

APPLICATIONS
Because of the various output options available from the part, the AD9833 can be configured to suit a wide variety of applications.

One of the areas where the AD9833 is suitable is in modulation applications. The part can be used to perform simple modulation, such as FSK. More complex modulation schemes, such as GMSK and QPSK, can also be implemented using the AD9833.

In an FSK application, the two frequency registers of the AD9833 are loaded with different values. One frequency will represent the space frequency, while the other will represent the mark frequency. Using the FSELECT bit in the control register of the AD9833, the user can modulate the carrier frequency between the two values.

The AD9833 has two phase registers; this enables the part to perform PSK. With phase shift keying, the carrier frequency is phase shifted, the phase being altered by an amount that is related to the bit stream being input to the modulator.

The AD9833 is also suitable for signal generator applications. Because the MSB of the DAC data is available at the VOUT pin, the device can be used to generate a square wave.

With its low current consumption, the part is suitable for applications in which it can be used as a local oscillator.

GROUNDING AND LAYOUT
The printed circuit board that houses the AD9833 should be designed so that the analog and digital sections are separated and confined to certain areas of the board. This facilitates the use of ground planes that can be separated easily. A minimum etch technique is generally best for ground planes since it gives the best shielding. Digital and analog ground planes should be joined in one place only. If the AD9833 is the only device requiring an AGND to DGND connection, then the ground planes should be connected at the AGND and DGND pins of the AD9833. If the AD9833 is in a system where multiple devices require AGND to DGND connections, the connection should be made at one point only, a star ground point that should be established as close as possible to the AD9833.

Avoid running digital lines under the device as these will couple noise onto the die. The analog ground plane should be allowed to run under the AD9833 to avoid noise coupling. The power supply lines to the AD9833 should use as large a track as possible to provide low impedance paths and reduce the effects of glitches on the power supply line. Fast switching signals, such as clocks, should be shielded with digital ground to avoid radiating noise to other sections of the board. Avoid crossover of digital and analog signals. Traces on opposite sides of the board should run at right angles to each other. This will reduce the effects of feedthrough through the board. A microstrip technique is by far the best but is not always possible with a double-sided board. In this technique, the component side of the board is dedicated to ground planes, while signals are placed on the other side.

Good decoupling is important. The AD9833 should have supply bypassing of 0.1 µF ceramic capacitors in parallel with 10 µF tantalum capacitors. To achieve the best from the decoupling capacitors, they should be placed as close as possible to the device, ideally right up against the device.

Proper operation of the comparator requires good layout strategy. The strategy must minimize through proper layout of the PCB the parasitic capacitance between V_{IN} and the SIGN BIT OUT pin by adding isolation using a ground plane. For example, in a 4-layer board, the C_{IN} signal could be connected to the top layer and the SIGN BIT OUT connected to the bottom layer, so that isolation is provided by the power and ground planes between.

Hoja de especificaciones del microcontrolador PIC18F2553

PIC18F2458/2553/4458/4553
Data Sheet

28/40/44-Pin High-Performance, Enhanced Flash, USB Microcontrollers with 12-Bit A/D and nanoWatt Technology

MICROCHIP

PIC18F2458/2553/4458/4553

28/40/44-Pin High-Performance, Enhanced Flash, USB Microcontrollers with 12-Bit A/D and nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 Endpoints (16 bidirectional)
- 1-Kbyte Dual Access RAM for USB
- On-Chip USB Transceiver with On-Chip Voltage Regulator
- Interface for Off-Chip USB Transceiver
- Streaming Parallel Port (SPP) for USB Streaming Transfers (40/44-pin devices only)

Power-Managed Modes:

- Run: CPU On, Peripherals On
- Idle: CPU Off, Peripherals On
- Sleep: CPU Off, Peripherals Off
- Idle mode Currents Down to 5.8 μA Typical
- Sleep mode Currents Down to 0.1 μA Typical
- Timer1 Oscillator: 1.1 μA Typical, 32 kHz, 2V
- Watchdog Timer: 2.1 μA Typical
- Two-Speed Oscillator Start-up

Special Microcontroller Features:

- C Compiler Optimized Architecture with Optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash Program Memory Typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory Typical
- Flash/Data EEPROM Retention: > 40 Years
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 41 ms to 131s
- Programmable Code Protection
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via Two Pins
- In-Circuit Debug (ICD) via Two Pins
- Optional Dedicated ICD/ICSP Port (44-pin TQFP package only)
- Wide Operating Voltage Range (2.0V to 5.5V)

Flexible Oscillator Structure:

- Four Crystal modes, Including High-Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal Oscillator Block:
  - 8 user-selectable frequencies, from 31 kHz to 8 MHz
  - User-tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Dual Oscillator Options allow Microcontroller and USB module to Run at Different Clock Speeds
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if any clock stops

Peripheral Highlights:

- High-Current Sink/Source: 25 mA/25 mA
- Three External Interrupts
- Four Timer modules (Timer0 to Timer3)
- Up to 2 Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 5.2 ns (TCY/16)
  - Compare is 16-bit, max. resolution 83.3 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bits
- Enhanced Capture/Compare/PWM (ECCP) module:
  - Multiple output modes
  - Selectable polarity
  - Programmable dead time
  - Auto-shutdown and auto-restart
- Enhanced USART module:
  - LIN bus support
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I^2C™ Master and Slave modes
- 12-Bit, up to 13-Channel Analog-to-Digital Converter module (A/D) with Programmable Acquisition Time
- Dual Analog Comparators with Input Multiplexing

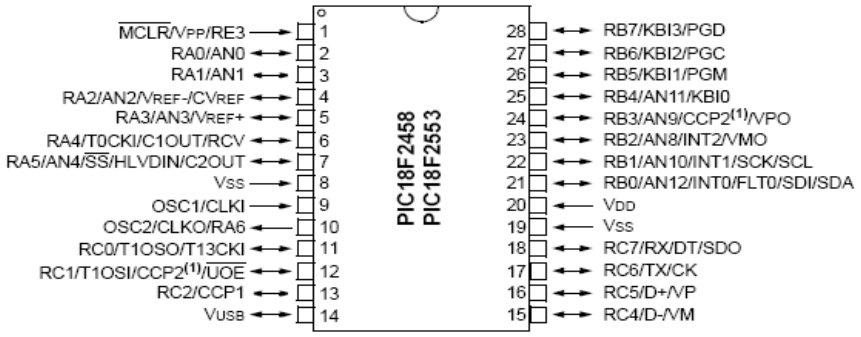
Note: This document is supplemented by the *"PIC18F2455/2550/4455/4550 Data Sheet"* (DS39632). See **Section 1.0 "Device Overview"**.

Device	Program Memory		Data Memory		I/O	12-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comp.	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™			
PIC18F2458	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2553	32K	16384											
PIC18F4458	24K	12288			35	13	1/1	Yes					
PIC18F4553	32K	16384											

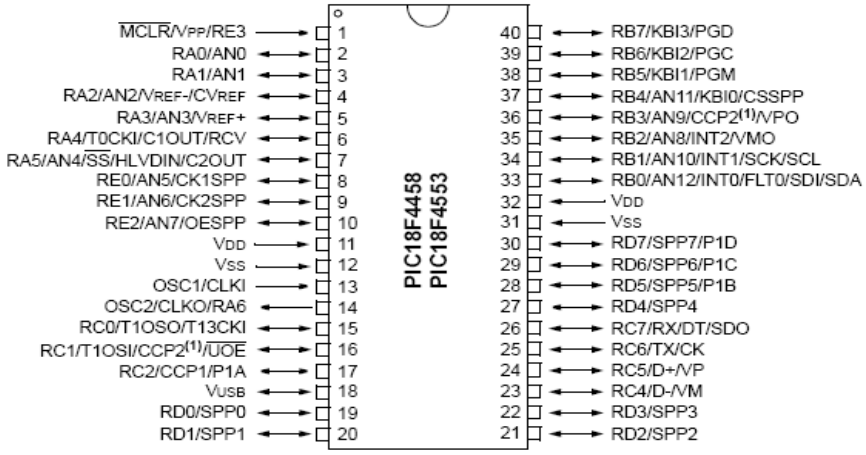
PIC18F2458/2553/4458/4553

Pin Diagrams

28-Pin SPDIP, SOIC



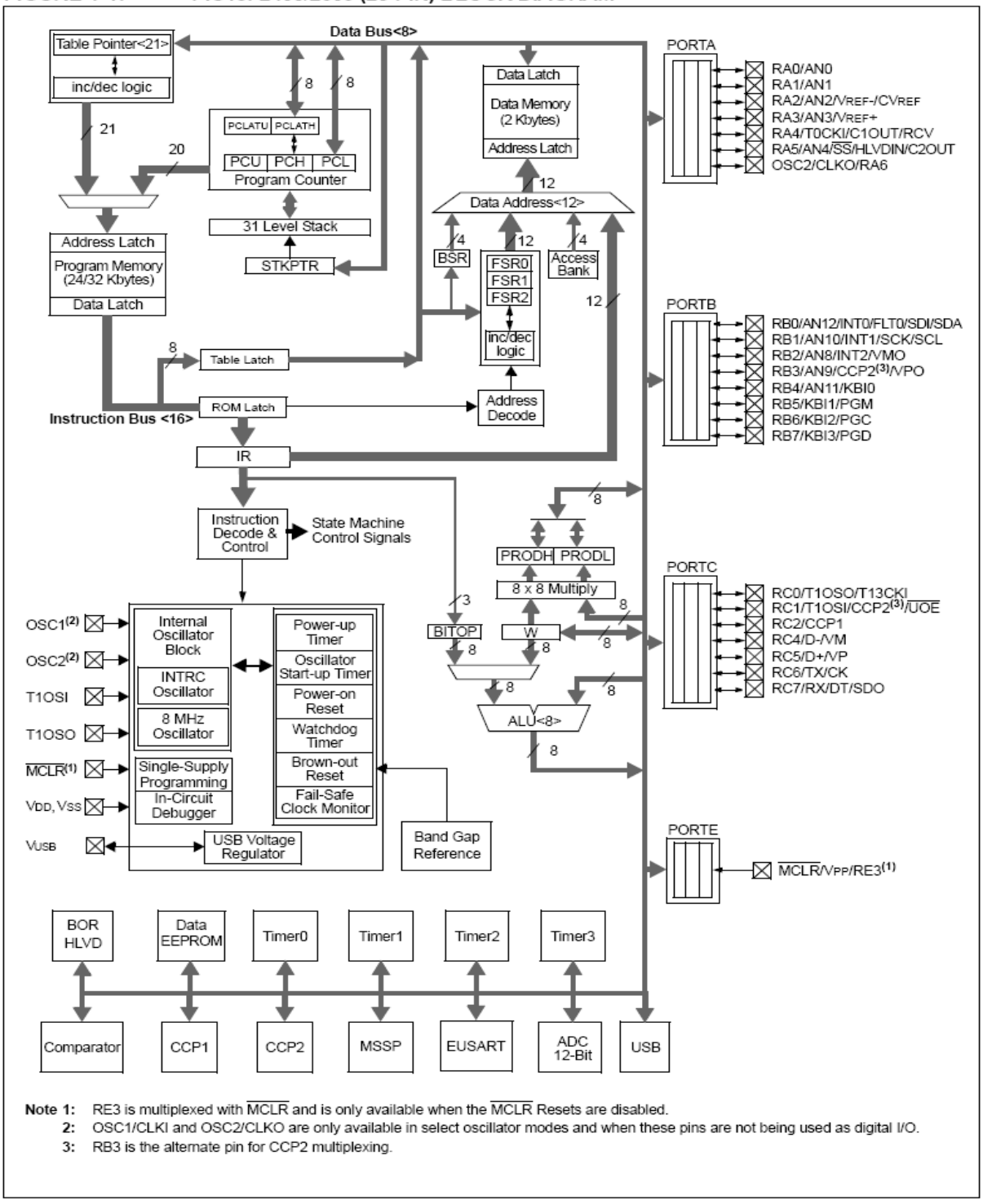
40-Pin PDIP



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

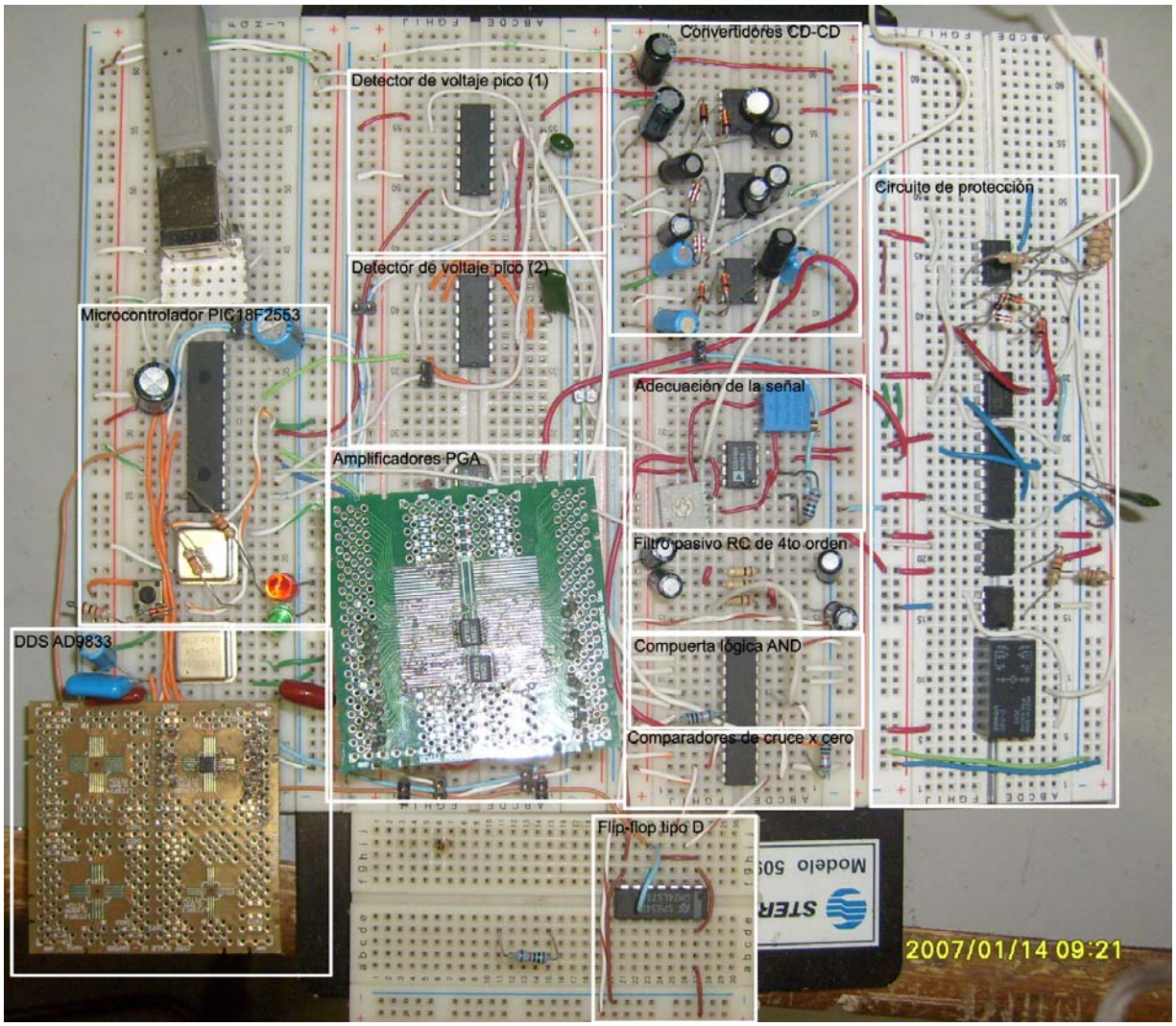
PIC18F2458/2553/4458/4553

FIGURE 1-1: PIC18F2458/2553 (28-PIN) BLOCK DIAGRAM



Note 1: RE3 is multiplexed with MCLR and is only available when the MCLR Resets are disabled.
2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O.
3: RB3 is the alternate pin for CCP2 multiplexing.

Imagen del circuito en protoboard del Bode Plotter


Convertidores CD-CD
Detector de voltaje pico (1)
Circuito de protección
Detector de voltaje pico (2)
Microcontrolador PIC18F2553
Adecuación de la señal
Amplificadores PGA
Filtro pasivo RC de 4to orden
DDS AD9833
Compuerta lógica AND
Comparadores de cruce x cero
Flip-flop tipo D
2007/01/14 09:21