DISEÑO DE FILTROS DIGITALES TIPO FIR, IIR Y ADAPTATIVOS UTILIZANDO FPGA

María Beatriz Muñoz Buitrón
Andrés Darío Dorado Peña

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Trabajo de Grado de Desarrollo
Línea de Desarrollo en Procesamiento Digital de Señales
Popayán, Febrero de 2011

DISEÑO DE FILTROS DIGITALES TIPO FIR, IIR Y ADAPTATIVOS UTILIZANDO FPGA

Trabajo de Grado

María Beatriz Muñoz Buitrón
Andrés Darío Dorado Peña

Director: PhD. Pablo E. Jojoa

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Trabajo de Grado de Desarrollo
Línea de Desarrollo en Procesamiento Digital de Señales
Popayán, Febrero de 2011

Tabla de Contenido

Lista de Figuras

Lista de Tablas

CAPITULO 1
INTRODUCCION

El procesamiento digital de señales es un área muy importante en la electrónica digital, razón por la cual la fundamentación teórica y práctica en este tema juega un papel muy importante en la formación de futuros Ingenieros Electrónicos y en Telecomunicaciones. La materia de Procesamiento Digital de Señales hace parte desde hace algunos años del pensum del programa de Ingeniería Electrónica y Telecomunicaciones, y en él se abordan temas como: Sistemas de Tiempo Discreto Lineales e Invariantes en el tiempo LIT[1], Representación en Frecuencia de señales de Tiempo Discreto, Transformada Z, Diseño de Filtro Digitales e Introducción a los Filtros Adaptativos, conceptos que son de vital importancia y que serían complementados con algunas prácticas de laboratorio en donde se haga uso de estos. Una de las aplicaciones más comunes y útiles que se fundamenta en procesamiento digital de señales es el filtrado digital. Por esta razón se planteó la elaboración de guías de laboratorio relacionadas con el diseño, simulación e implementación de filtros digitales tipo FIR, IIR y adaptativos.

Teniendo en cuenta que la Facultad de Ingeniería Electrónica y Telecomunicaciones cuenta con algunas FPGA de la compañía Xilinx®, se plantea como objetivo *Diseñar e implementar filtros digitales tipo FIR, IIR y adaptativos que trabajen con señales de audio utilizando FPGA Spartan 3A de Xilinx®*. Para alcanzar este objetivo es necesario definir los requerimientos tanto físicos como lógicos para la implementación de una etapa de adaptación de señales de audio a la entrada de la FPGA, así como la exploración de herramientas que permitan efectuar el diseño y la simulación de algoritmos y que además generen código HDL[2] a partir de estos diseños.

Las prácticas de laboratorio se dividen en dos grupos: el primero contiene las prácticas de diseño y la simulación, que se realizan en la herramienta Matlab® y en el entorno Simulink® para los filtros FIR, IIR y Adaptativo y el segundo contiene la práctica de implementación, donde se implementan en la FPGA los diseños realizados en las prácticas de diseño y simulación para lo cual se utiliza la herramienta *ISE Desing Suite* de Xilinx®.

Las prácticas implementadas son cinco: la primera relacionada con las técnicas de ventana para filtros FIR, la segunda relacionada con la técnica de rizado constante para filtro FIR, la tercera relacionada con la técnica de filtros analógicos para generación de filtros IIR, la cuarta relacionada con el filtrado adaptativo y en la quinta se encuentra contenida la implementación de los diseños anteriores de filtros FIR, IIR y adaptativo.

[1] LIT- Linear Time-Invariant: Lineal e invariante en el tiempo
[2] HDL- Hardware Description Language: Lenguaje de Descripción Hardware

El siguiente documento, que es el respaldo teórico de las prácticas de laboratorio, está organizado de la siguiente forma: en el capítulo 2 se exponen las diferentes estructuras de prácticas utilizadas en laboratorio por otras universidades [1]-[10] que sirven de soporte para la selección de la estructura adecuada para las prácticas implementadas; en el capítulo 3 se tratan los principales conceptos teóricos del filtrado digital y el proceso de diseño de filtros digitales; en el capítulo 4 se presentan algunas características de la FPGA Spartan 3A y los requerimientos de la etapa de adaptación necesaria para el tratamiento de audio con el conversor ADC[3], en el capítulo 5 se mostrarán las pruebas que fueron realizadas y resultados que fueron obtenidos de la simulación e implementación en la FPGA de los diseños realizados en Matlab® y para finalizar en el capítulo 6 se encontrarán las conclusiones y trabajos futuros del trabajo.

Adicionalmente en los anexos se encuentran las guías de laboratorio que serán utilizadas por los estudiantes.

[3] ADC- Analogic Digital Converter: Conversor Analógico Digital

CAPITULO 2
ESTRUCTURA DE PRÁCTICAS PARA EL DISEÑO DE FILTROS DIGITALES FIR, IIR Y ADAPTATIVO EN FPGA

Este capítulo contiene el proceso de escogencia del formato utilizado en las prácticas de laboratorio de filtros FIR, IIR y Adaptativos. Inicialmente se escoge la estructura que se va a aplicar para cada una de las 5 prácticas, para después explicar cada uno de los ítems que las conforman.

2.1 ESTRUCTURA DE LA PRÁCTICA

2.1.1 PRÁCTICAS DE LABORATORIO DE REFERENCIA

Para escoger una estructura adecuada de práctica de laboratorio, con un contenido y orden del mismo, coherente y claro para el entendimiento tanto del estudiante como del profesor, y que maximice el aprovechamiento del tiempo con el que se cuenta para desarrollar la práctica para que se puedan cumplir los objetivos propuestos con el menor número de pasos concretos posibles, se hizo una búsqueda en diferentes universidades del mundo, de prácticas de laboratorios relacionadas con filtros digitales para observar su estructura, y así establecer la forma organizacional que más se ajusta a las características antes descritas.

Algunas de las prácticas más relevantes que se utilizaron para este objetivo, se exponen en la tabla 2.1.

2.1.2 JUSTIFICACIÓN DE LA ESTRUCTURA ESCOGIDA

Analizando la organización de cada una de las prácticas enunciadas anteriormente, se decidió que la estructura adecuada para las prácticas de laboratorio, que fuera coherente entre sus partes y concreta en su contenido, es la mostrada en la tabla 2.2.

Nombre de la Practica	Universidad	Objetivos	Resumen	Introducción	Marco teórico	Trabajo Previo	Procedimiento de laboratorio	Resultados	Trabajo Complementario	Reporte	Conclusiones	Referencias
Practicas de diseño de filtros FIR e IIR. Laboratorio de Tratamiento digital de señales [1]	Universidad de Alcalá de España			★			★	★				
Digital Signal Processing Hardware. Lab Work 3. Digital Filters I: Low-Pass Filter [2]	Universidad, de Anadolu en Turquía	★		★	★	★	★	★				
J-DSP Lab 4: FIR and IIR Filter Design [3]	Universidad del Estado de Arizona.			★			★	★	★			
Lab5: Digital Filtering [4]	Universidad de Minnesota	★			★	★	★	★	★	★		
Filter Lab: Design of a 3rd order Butterworth Filter using the Universal Filter Chip UAF42 [5]	Universidad de Pennsylvania	★			★	★	★			★		★
Practica 3: Introducción al diseño de filtros activos [6]	Universidad Simón Bolívar de Venezuela		★	★	★		★				★	★
Lab #4: Digital Filters [7]	Universidad de Toronto	★			★		★	★	★	★		★
Lab #4: Fourier Transforms and Digital Filters [8]	Universidad de Connecticut	★			★	★	★	★		★		★
Laboratorio: Filtros Activos [9]	Universidad Nacional de San Luis de Argentina	★					★	★				★
Lab 4: FIR e IIR Filters in Matlab [10]	Universidad de Vermont	★		★			★	★	★			★
Laboratorios I y II de Sistemas de Telecomunicaciones [11]	Universidad del Cauca	★		★		★	★		★	★		★

Tabla 2. 1 ***Estructura de las prácticas de referencia***

Las guías de laboratorio están divididas en dos categorías: en la primera se encuentran las Prácticas de diseño y simulación en Matlab® y Simulink® de filtros FIR Káiser y con otro método de ventana, FIR Rizado constante, IIR Butterworth y Filtro Adaptativo; en la segunda se encuentra la Practica de implementación donde se mostrará la generación

de código VHDL[4] a partir los modelos en Simulink® obtenidos en las guías anteriores, para luego realizar la implementación de este código en la FPGA.

Practicas de simulación:	*Practica de Implementación:*
1. OBJETIVOS	1. OBJETIVOS
2. INTRODUCCION	2. INTRODUCCION
3. TEORIA	3. TEORIA
4. TRABAJO PREVIO	4. TRABAJO PREVIO
5. EJEMPLO PRACTICO	5. EJEMPLO PRACTICO
6. PROCEDIMIENTO DE LABORATORIO	6. PROCEDIMIENTO DE LABORATORIO PARA IMPLEMENTAR LOS MODELOS SIMULINK® DE LOS FILTROS
6.1 Diseño en Matlab®	6.1 Conversión a modelos de punto fijo
6.2 Simulación en Matlab®	6.2 Generación de código VHDL
6.3 Migración al Entorno Simulink®	6.3 Integración de los códigos generados con los conversores ADC y DAC
6.4 Resultados	6.4 Implementación del código en FPGA
	6.5 Análisis del desempeño
	6.6 Resultados
7. TRABAJO COMPLEMENTARIO	7. TRABAJO COMPLEMENTARIO
8. REPORTE	8. REPORTE
9. REFERENCIAS	9. REFERENCIAS

Tabla 2. 2 ***Estructura de Prácticas de Simulación y de Práctica de Implementación.***

2.2 EXPLICACIÓN DE LA ESTRUCTURA

- OBJETIVOS: contienen los propósitos de aprendizaje de la práctica, que es lo que se desea que el alumno adquiera de conocimiento al realizar la práctica.
- INTRODUCCIÓN: se realiza una breve presentación del contenido de la práctica.
- TEORÍA: síntesis de los conceptos teóricos que se manejarán en la práctica.
- TRABAJO PREVIO: conceptos y herramientas que deben ser conocidas previamente por parte del estudiante para aligerar el desarrollo de la practica.
- EJEMPLO PRÁCTICO: ejemplo completo del diseño y la simulación o de la implementación de filtros digitales en Matlab® y Simulink® para que sea una guía en el desarrollo del procedimiento del laboratorio.

[4] VHDL- **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage: Lenguaje de Descripción Hardware para Circuitos Integrados de Muy Alta Velocidad

- PROCEDIMIENTO DEL LABORATORIO: inicialmente se especifica con que software y/o equipo de laboratorio se va a trabajar dependiendo si la práctica es de simulación o implementación.
  En las prácticas de simulación, los ítems *Diseño y Simulación en Matlab®* piden realizar un diseño de un filtro y simularlo mediante las funciones de Matlab® y el ítem *Migración al entorno Simulink®* indica otra alternativa de simulación esta vez por medio de los bloques de Simulink®.
  En la práctica de implementación, los ítems *conversión a modelos de punto fijo, generación de código VHDL*, *integración de los códigos generados con los conversores ADC[5] y DAC[6], implementación del código en FPGA* y *análisis del desempeño,* piden adaptar los modelos Simulink® obtenidos de las guías anteriores para a partir de ellos generar código VHDL e integrarlo con el código de los conversores ADC y DAC, y ver su funcionamiento en tiempo real.
  Finalmente en *resultados*, se proponen algunas preguntas y tablas por llenar que se consideran necesarias para afirmar que los conceptos que se manejan en la práctica fueron correctamente entendidos y aplicados.
- REPORTE: comprende el desarrollo de preguntas, tabulación de datos, entrega de códigos y presentación al profesor del funcionamiento del filtro propuesto. Este reporte debe ser entregado al finalizar el tiempo de laboratorio por parte del estudiante al profesor.
- TRABAJO COMPLEMENTARIO: ejercicios adicionales similares a los planteados en el procedimiento de laboratorio, que buscan que el estudiante retome todo el tema de la práctica, valiéndose de lo que anteriormente hizo.
- REFERENCIAS: libros, tesis y documentos de los cuales se baso la práctica.

[5] ADC- Analogic Digital Converter: Conversor Analógico Digital
[6] DAC- Digital Analogic Converter: Conversor Digital Analógico

CAPITULO 3
DISEÑO DE FILTROS DIGITALES

En este capítulo se expondrán los conceptos fundamentales de la teoría de filtrado selectivo en frecuencia y adaptativo, además se presentará con detalle el proceso general de diseño de filtros que será utilizado para la elaboración de las prácticas de laboratorio.

3.1 FILTRADO

En una amplia variedad de aplicaciones resulta de interés cambiar las amplitudes relativas de las componentes de frecuencia de una señal, o quizás eliminar por completo algunas de estas, proceso conocido como filtrado. En procesamiento de señales, dejar pasar algunas frecuencias y atenuar de manera significativa o eliminar por completo otras, se conoce como el proceso de filtrado selectivo en frecuencia [12].

Los filtros pueden clasificase como analógicos o digitales, se prestará especial atención a los filtros digitales, los cuales se pueden representar como un algoritmo matemático que se puede implementar en hardware o en software, y que operan una señal digital de entrada para producir una señal digital de salida que cumpla con los objetivos del filtrado. Los filtros digitales se pueden clasificar como filtros de respuesta finita al impulso FIR[7] o filtros de respuesta infinita al impulso IIR[8].

3.2 FILTROS SELECTIVOS EN FRECUENCIA

Un filtro ideal selectivo en frecuencia es un sistema Lineal e Invariante en el Tiempo (LIT) que permite el paso de ciertas componentes de frecuencias sin ninguna modificación y atenúa completamente las otras. El rango de frecuencias que pasan sin atenuación se llama banda de paso del filtro, y el rango de frecuencias que son atenuadas se llama banda de supresión. De esta manera la magnitud de la respuesta ideal de un filtro está dada por $|H(\omega)| = 1$ en la banda de paso y $|H(\omega)| = 0$ en la banda de supresión.

Existen diferentes tipos de filtros selectivos en frecuencia y se exponen a continuación.
La característica de magnitud de la respuesta de un filtro ideal pasa bajo es ilustrada en la figura 3.1.a. Las regiones $0 \leq \omega \leq \omega_c$ y $\omega > \omega_c$ se refieren a la banda de paso y supresión respectivamente. Un filtro ideal pasa bajo debe tener la magnitud de su respuesta en frecuencia $|H(\omega)| = 1$ en el rango de frecuencias $0 \leq \omega \leq \omega_c$ y $|H(\omega)| = 0$ para $\omega > \omega_c$, permitiendo el paso de todas las componentes de frecuencia por debajo de ω_c y atenuando todas las componentes de frecuencia que se encuentren por encima. Un análisis similar se realiza para la

[7] FIR- Finite Impulse Response: Respuesta Finita al Impulso
[8] IIR- Infinite Impulse Response: Respuesta Infinita al Impulso

magnitud de la respuesta de un filtro ideal pasa alto ilustrada en la figuras 3.1.b, la magnitud de la respuesta de un filtro ideal pasa banda y la magnitud de la respuesta de un filtro ideal elimina banda ilustradas en la figura 3.1.c y 3.1.d respectivamente, donde las frecuencias ω_a y ω_b son llamadas frecuencia de borde inferior y frecuencia de borde superior respectivamente. Existe además un filtro conocido como filtro pasa todo, el cual presenta una magnitud de respuesta en frecuencia $|H(w)| = 1$ para todas las frecuencias, entonces todas las frecuencias pasan a través del filtro con igual ganancia. Este filtro no puede remover componentes de frecuencia, pero puede alterar la respuesta en fase. [13]

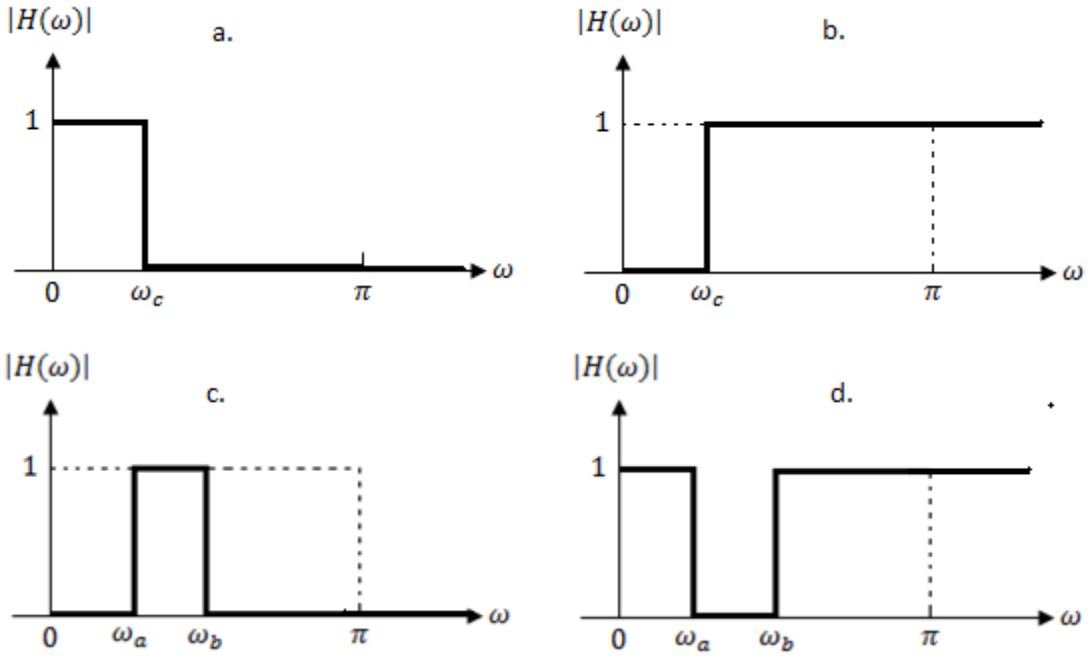


Figura 3. 1 ***Filtros ideales selectivos en frecuencia. a) Pasa bajo b) Pasa alto c) Pasa banda d) Elimina banda***

3.3 FILTROS FIR

Un filtro FIR de longitud M y orden $N = M - 1$, con entrada $x[n]$ y salida $y[n]$ es descrito por la ecuación de diferencias:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] = \sum_{k=0}^{N} b_k x\,[n-k]\,, \tag{1}$$

donde $\{b_k\},\ k = 0,1,\ldots,N$, es el conjunto de coeficientes. Alternativamente se puede expresar la secuencia de salida como la convolución de la respuesta del sistema al impulso unitario $h[n]$ con el espectro de la señal de entrada:

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]$$

(2)

Los límites superior e inferior de la ecuación (2) reflejan las características de causalidad[9] y duración finita del filtro. Claramente $b_k = h[k], \quad k = 0,1, \ldots, N$ [14].

3.3.1 RESPUESTA EN FRECUENCIA

La respuesta en frecuencia de un filtro FIR de orden N es dada por:

$$H(\omega) = \sum_{n=0}^{N} h[n] e^{-j\omega_n}$$

(3)

Recordando que la respuesta en frecuencia de un sistema digital esta generalmente en valores complejos que consisten en magnitud y fase, la función puede ser escrita como:

$$H(\omega) = A(\omega) e^{j\theta(\omega)}$$

(4)

en donde $A(\omega)$ es la característica de amplitud de la respuesta y $\theta(\omega)$ es la característica fase de la respuesta.

La respuesta en magnitud es dada por

$$M(\omega) = |H(\omega)| = |A(\omega)|$$

(5)

3.3.2 FILTROS DE FASE LINEAL

Una de las mayores ventajas de los filtros FIR es la facilidad con que pueden ser diseñados filtros de fase lineal exacta[10]. Un filtro con características de fase lineal no distorsionará la señal de entrada, lo que es deseable en un gran número de aplicaciones. Otras propiedades importantes de los filtros FIR es que estos son estables[11] y pueden ser implementados eficientemente en hardware de propósito general o específico [15].

Una gran variedad de aplicaciones requieren que los filtros digitales tengan fase lineal, particularmente es importante para las señales que son sensibles a la fase como el habla, la música y la transmisión de datos, en donde la fase no lineal puede generar distorsiones de frecuencia inaceptables [13].

Existen cuatro tipos de filtros FIR de fase lineal, dependiendo de si el orden N es par o impar y además de si la respuesta del sistema (filtro) al impulso $h[n]$ es simétrica o anti-simétrica [16]:

- **Tipo 1:** En este tipo de filtros la respuesta al impulso tiene longitud impar (N es par) y $h[n] = h[N - n]$. El espectro es periódico con periodo 2π.

[9] Un sistema es causal si su salida en cualquier instante de tiempo depende sólo de los valores de la entrada presentes y pasados. A menudo, a dicho sistema se llama no anticipativo, ya que la salida del sistema no anticipa valores futuros de la entrada [12].

[10] La Fase lineal ocurre cuando el desplazamiento de fase a la frecuencia ω es una función lineal de ω [12].

[11] Un filtro digital es estable si para toda señal de entrada acotada, la salida es acotada. Una señal $x[n]$ es acotada si su magnitud $|x[n]|$ no va a infinito. Puesto que los filtros FIR tienen un número finito de ceros en su función de transferencia, los filtros FIR son siempre estables.[13]

- **Tipo 2:** En este tipo de filtros la respuesta al impulso tiene longitud par (N es impar) y $h[n] = h[N-n]$. El espectro es también periódico con periodo de 4π. La respuesta en frecuencia debe ser cero en $\omega = \pi$. Estos filtros producen buenos filtros pasa bajos pero son inapropiados para diseños de filtros pasa altos.
- **Tipo 3:** En este tipo de filtros la respuesta al impulso tiene longitud impar y $h[n] = -h[N-n]$. El espectro es periódico con periodo 2π. La respuesta en frecuencia debe ser cero en $\omega = 0$ y $\omega = \pi$. No son apropiados para diseños de filtros pasa bajo y pasa alto. Además introducen un cambio de fase de 90 grados.
- **Tipo 4:** En este tipo de filtros la respuesta al impulso tiene longitud par y $h[n] = -h[N-n]$. El espectro es periódico con periodo 4π. La respuesta en frecuencia debe ser cero en $\omega = 0$ pero no necesariamente en $\omega = \pi$. No es recomendable usar este tipo de filtros para diseñar filtros pasa bajo, aunque se pueden realizar buenos filtros pasa altos. Como los filtros tipo 3, estos también introducen un cambio de fase de 90 grados.

Ahora se abordaran algunos métodos de diseño utilizados para obtener filtros FIR.

3.3.3 METODO DE VENTANAS

La explicación del método se inicia con la respuesta en frecuencia deseada $H_d(\omega)$ para luego determinar la correspondiente respuesta al impulso $h_d[n]$, que está relacionada con $H_d(\omega)$ mediante las ecuaciones de la transformada de Fourier:

$$H_d(\omega) = \sum_{n=0}^{\infty} h_d[n] e^{-j\omega n} \tag{6}$$

$$h_d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(\omega) e^{j\omega n} d\omega \tag{7}$$

En general, la respuesta al impulso $h_d[n]$ obtenida de la ecuación (7) es de infinita duración y debe ser truncada en algún punto, por ejemplo entre $0 \leq n \leq N$, para generar un filtro FIR de orden N y longitud $M = N + 1$. El truncamiento de $h_d[n]$ en una longitud M es equivalente a multiplicar $h_d[n]$ con una ventana rectangular definida como:

$$\mathrm{w}[n] = \begin{cases} 1, & \mathrm{n} = 0,1,\ldots,\mathrm{N} \\ 0 & \text{otros} \end{cases} \tag{8}$$

Entonces la respuesta del filtro FIR al impulso se convierte en

$$h[\mathrm{n}] = h_d[\mathrm{n}]w[\mathrm{n}]$$

(9)

$$h[\mathrm{n}] = \begin{cases} h_d[\mathrm{n}], & n = 0,1,..,N \\ 0, & otros \end{cases}$$

(10)

Es importante destacar que $h[\mathrm{n}]$ corresponde a una porción de $h_d[n]$ desplazada de modo que $h[\mathrm{n}]$ contenga la mayor cantidad de energía de $h_d[n]$. En el anexo D se muestran los $h[\mathrm{n}]$ para filtros selectivos en frecuencia.

Si la representación en el dominio de la frecuencia de la función ventana, es:

$$W(\omega) = \sum_{n=0}^{N} w[\mathrm{n}]e^{-j\omega n} \ ,$$

(11)

entonces la convolución de $H_d(\omega)$ con $\mathrm{W}(\omega)$ genera la respuesta en frecuencia del filtro FIR trucado.

El espectro de la magnitud de la respuesta en frecuencia de una función de ventana rectangular es mostrada en la figura 3.2, para $N = 31$ y $N = 61$. El ancho del lóbulo principal[12] de esta ventana es $\frac{4\pi}{N+1}$. Entonces, como se puede observar si se incrementa el orden N el lóbulo principal reduce su anchura [14].

[12] El ancho del lóbulo principal es el ancho medido hasta el primer cero de $\mathrm{W}(\omega)$ [14].

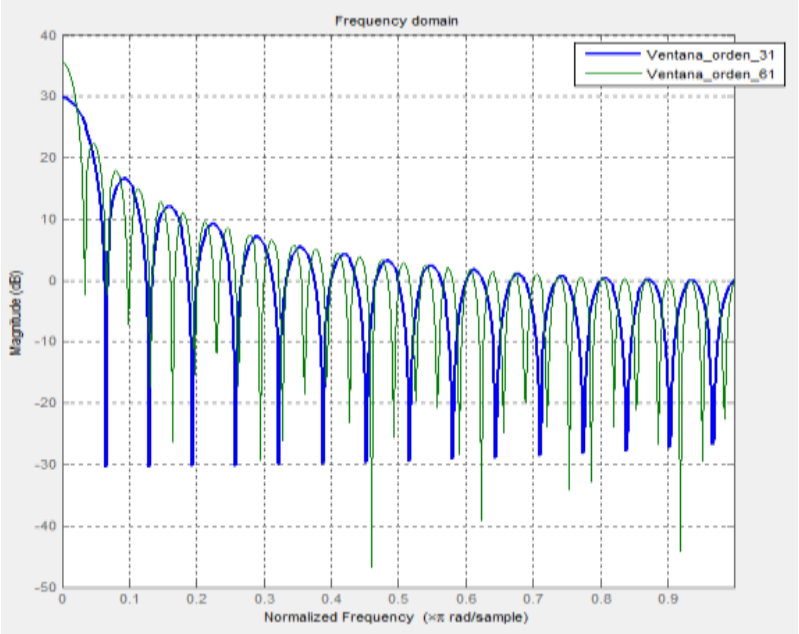


Figura 3. 2 ***Respuesta en frecuencia de la ventana rectangular***

Aunque el ancho de cada lóbulo lateral se reduce al incrementar N, el alto de cada lóbulo lateral se incrementa un poco al incrementar N, de tal manera que el área bajo los lóbulos laterales permanece invariante a los cambios de N. Las características de la ventana rectangular desempeñan un papel importante en la determinación de la respuesta en frecuencia resultante del filtro FIR obtenido al truncar $h_d[n]$ a longitud M. Específicamente la convolución de $H_d(\omega)$ con $\mathrm{W}(\omega)$ tiene el efecto de suavizar $H_d(\omega)$. Al incrementar N, $\mathrm{W}(\omega)$ reduce su anchura y la suavización que provee también es reducida. Por otra parte, los lóbulos laterales grandes de $\mathrm{W}(\omega)$ generan algunos efectos de oscilación o rizados indeseables en la respuesta en frecuencia del filtro FIR $H(\omega)$, estos efectos indeseables son mejor tratados por ventanas que no posean discontinuidades abruptas en sus características en el dominio del tiempo y que tengan lóbulos laterales pequeños en sus características en el dominio de la frecuencia. Las oscilaciones cerca de los bordes de las bandas se conocen con el nombre de fenómeno de Gibbs[13] [14].

Nombre de la ventana	**Secuencia en el dominio del tiempo, $h[n]$, $0 \leq n \leq N$**
Barlett	$1 - \frac{2\left\lvert n - \frac{N}{2}\right\rvert}{N-1}$
Blackman	$0.42 - 0.5cos\left(\frac{2\pi n}{N}\right) + 0.08\left(\frac{4\pi n}{N}\right)$
Hamming	$0.54 - 0.46cos\left(\frac{2\pi n}{N}\right)$

[13] Efecto de Gibbs: Nombrado por el físico Americano Josiah Willard Gibb. Ocurre cada vez que la representación en series de Fourier de una señal posee discontinuidades.

Hanning	$\frac{1}{2}\left(1-cos\left(\frac{2\pi n}{N}\right)\right)$
Kaiser	$\frac{I_o[\beta\sqrt{1-\left(1-\frac{2n}{N}\right)^2}]}{I_o\beta}$

Tabla 3. 1 ***Funciones ventana***

La Tabla 3.1 muestra algunas funciones de ventana que poseen características de respuesta en frecuencia deseables. Todas estas funciones de ventana tienen lóbulos laterales significativamente bajos comparados con los de la ventana rectangular. Sin embargo, para el mismo valor de N estas ventanas tienen el ancho del lóbulo principal más ancho que el de la ventana rectangular. Consecuentemente estas funciones de ventanas ofrecen mayor alisamiento y una región de transición más ancha para los filtros FIR. Para reducir el ancho de la región de transición se puede simplemente incrementar la longitud de la ventana lo que resulta en un filtro más largo. La Tabla 3.2 resume importantes características en el dominio de la frecuencia para algunas ventanas.

Tipo de Ventana	Ancho de transición aproximado del lóbulo principal	Mínima atenuación en la banda de supresión	Pico de los lóbulos laterales
Rectangular	$4\pi/M$	$21dB$	$-13dB$
Barlett	$8\pi/M$	$25dB$	$-27dB$
Hanning	$8\pi/M$	$44dB$	$-32dB$
Hamming	$8\pi/M$	$53dB$	$-43dB$
Blackman	$12\pi/M$	$74dB$	$-58dB$

Tabla 3. 2 ***Características importantes de algunas ventanas***

A continuación se presenta información de relevancia de las ventanas Hamming y Kaiser, en [16] [17] se encuentra más información para las otras ventanas.

3.3.3.1 VENTANA HAMMING

Dado que las discontinuidades en el tiempo de la función de ventana rectangular producen oscilaciones en la respuesta en frecuencia, esta se puede remplazar con una función ventana que decaiga suavemente en ambos lados. Esto puede reducir el efecto oscilatorio presentado en los bordes de las bandas del filtro. La ventana de Hamming es muy popular dentro de estas clases de funciones [16].

Esta ventana está definida matemáticamente como:

$$w[n] = 0.54 - 0.46cos\left(\frac{2\pi n}{N}\right), \quad n = 0,1,\dots,N, \tag{12}$$

donde N es el orden del filtro.

La respuesta al impulso de los filtros FIR diseñados usando la ventana es:

$$h[n] = w[n]h_d[n] \tag{13}$$

La figura 3.3 muestra la función ventana de Hamming para $N = 41$ y la figura 3.4 muestra la comparación de un filtro Pasa Bajo con $N = 41$ diseñado con ventana rectangular y con ventana Hamming:

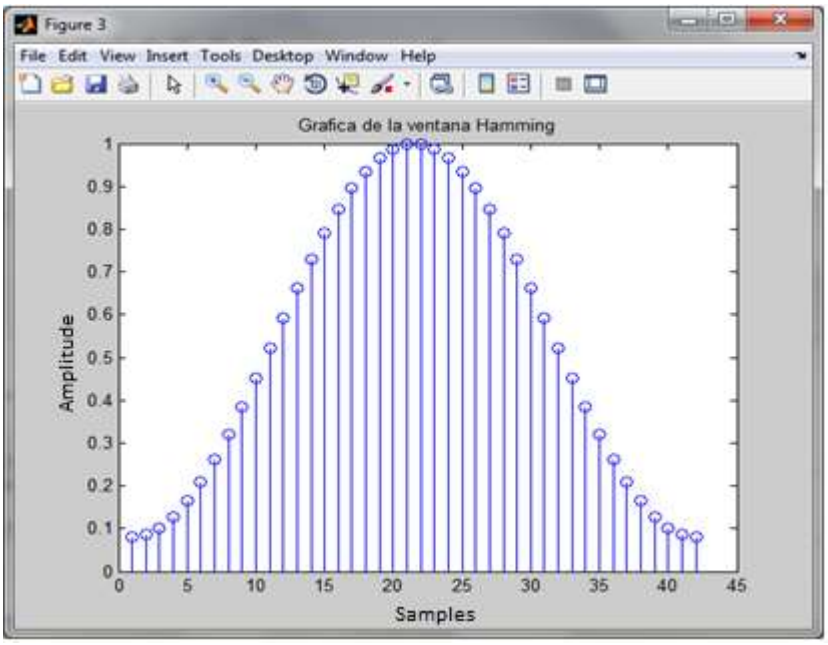


Figura 3. 3 ***Ventana Hamming con N=41***

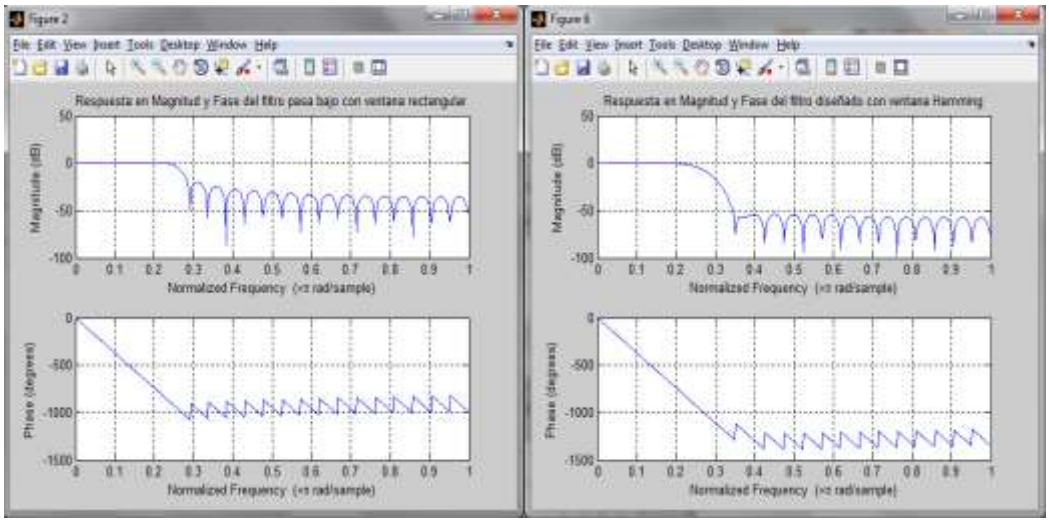


Figura 3. 4 ***Filtro Pasa Bajo N=41 diseñado con Ventana Rectangular y con Ventana Hamming*****.**

3.3.3.2 VENTANA DE KAISER

Esta es una de las más útiles y óptimas funciones para ventana, y está definida como:

$$w[n] = \frac{I_o[\beta\sqrt{1-\left(1-\frac{2n}{N}\right)^2}]}{I_o\beta}\ , \quad 0 \le n \le N\ , \tag{14}$$

en donde $I_o[\cdot]$ es la función de Bessel modificada de orden cero de tipo uno, y β es un parámetro que depende de N y puede ser escogido para generar varios anchos de transición y atenuación en la banda de supresión cercanos a los óptimos. Esta ventana puede generar diferentes anchos de transición para el mismo N, lo cual otras ventanas no ofrecen.

Debido a la complejidad involucrada en las funciones de Bessel, las ecuaciones de diseño para esta ventana no son fáciles de derivar. Afortunadamente Káiser[14] desarrollo estas ecuaciones de diseño empíricamente y se muestran a continuación [17].

Teniendo $\omega_p, \omega_s, A_p\ y\ A_s$ se puede determinar:

- Ancho de banda de transición normalizado $\Delta_f = \frac{\omega_s - \omega_p}{2\pi}$

(15)

- Orden del filtro $N \cong \frac{A_s - 7.95}{14.36\Delta_f} + 1$

(16)

- Parámetro $\beta = \begin{cases} 0.1102(A_s - 8.7), & A_s \ge 50 \\ 0.5842(A_s - 21)^{0.4} + 0.07886(A_s - 21), & 21 < A_s < 50 \\ 0, & A_s \le 21 \end{cases}$

(17)

La figura 3.5 muestra la función ventana de Káiser para $N = 41$ y la figura 3.6 muestra la comparación de un filtro Pasa Bajo con $N = 41$ diseñado con ventana rectangular y con ventana Káiser:

[14] Jim Kaiser descubridor de la ventana de Kaiser o ventana Kaiser-Bessel

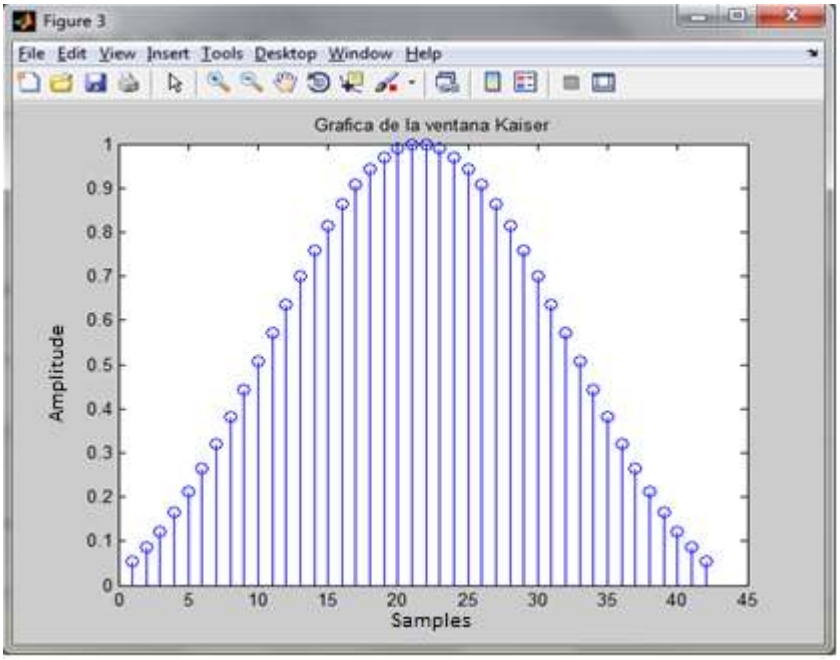


Figura 3. 5 ***Ventana Káiser con N=41***

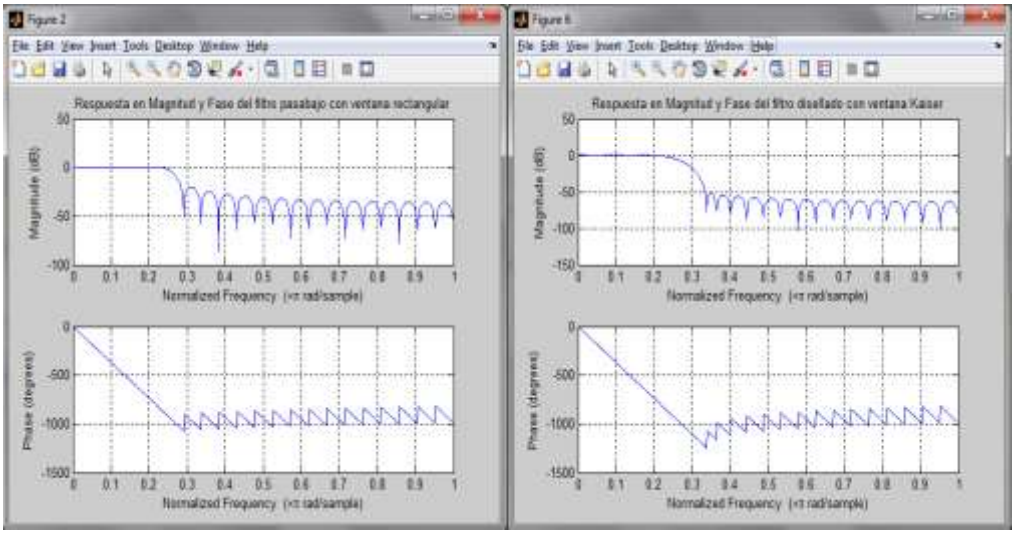


Figura 3. 6 ***Filtro Pasa Bajo N=41 diseñado con Ventana Rectangular y con Ventana Káiser y $\beta B = 4.53$***

3.3.4 METODO DE RIZADO CONSTANTE

Las técnicas de diseño de filtros lineales FIR de ventana y de muestreo en frecuencia son bastante fáciles de entender e implementar, pero tienen la desventaja de que la aproximación del error que es la diferencia entre la respuesta en frecuencia ideal y la real, no es uniformemente distribuida sobre las bandas, esta es mayor cerca de los bordes de las bandas y pequeña lejos de estas. Sin embargo distribuyendo el error uniformemente en las bandas, se pueden obtener filtros de menor orden que satisfagan las mismas especificaciones. Este problema es superado por la técnica de rizado [17].

Se define la función de error o error ponderado como la diferencia entre la respuesta en frecuencia ideal del filtro y la respuesta real, multiplicada por una función $W_{pond}(\omega)$ que puede ser expresada como:

$$E(\omega) = W_{pond}(\omega)[H_D(\omega) - H(\omega)], \tag{18}$$

en donde $H_D(\omega)$ es la respuesta en frecuencia ideal o deseada, $H(\omega)$ es la respuesta en frecuencia real y $W_{pond}(\omega)$ es una función de ponderación que permite escoger el tamaño relativo de los errores en las diferentes bandas de frecuencia, en particular es conveniente normalizar $W_{pond}(\omega)$ a la unidad en la banda de supresión e igualar $W_{pond}(\omega) = \frac{\delta_s}{\delta_p}$ en la banda de paso [15]. Estos conceptos son aclarados en la siguiente figura 3.7 donde se muestra la respuesta en frecuencia de un filtro pasa bajo con rizado constante junto con su respuesta ideal:

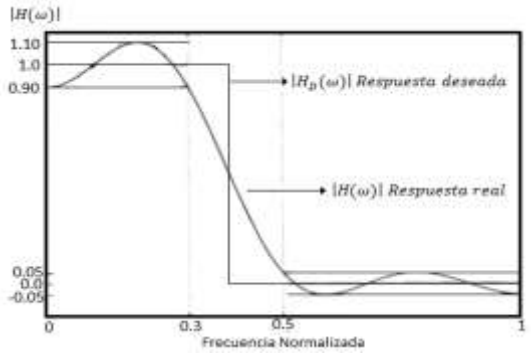


Figura 3. 7 ***Respuesta en frecuencia real e ideal de un filtro pasa bajo con rizado constante***

En la banda de paso la respuesta oscila entre 1 - δ_p y 1 + δ_p. En la banda de supresión la respuesta del filtro se sitúa entre 0 - δ_s y 0 + δ_s. La siguiente gráfica muestra la expresión $[H_D(\omega) - H(\omega)]$ para cada banda:

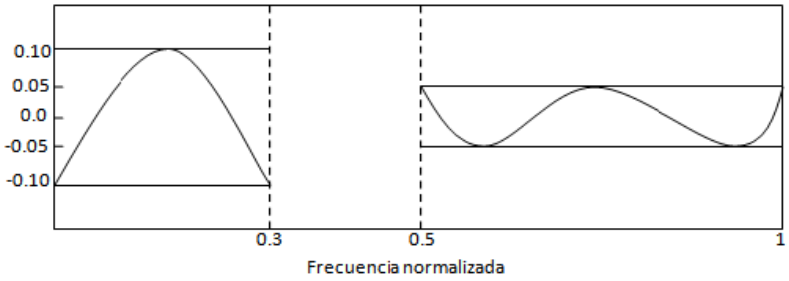


Figura 3. 8 ***Aproximación del error***

Ahora si se escoge:

$$W_{pond}(\omega) = \begin{cases} \dfrac{\delta_s}{\delta_p} = \dfrac{0.05}{0.1} = 0.5, en\ la\ banda\ de\ paso \\ 1, \quad en\ la\ banda\ de\ supresion \end{cases}$$

(19)

Entonces la función de error $E(\omega)$ es:

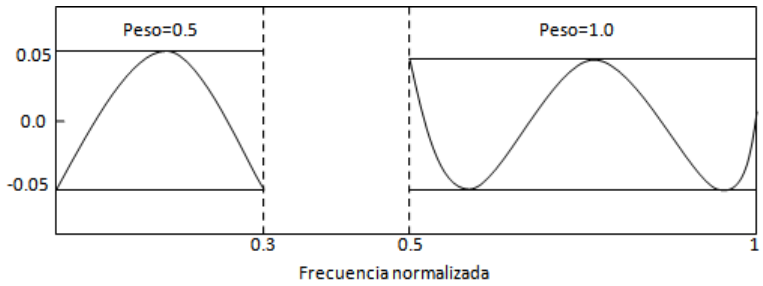


Figura 3. 9 ***Función del error***

El máximo error en las bandas de paso y de supresión llamado δ es igual a δ_s. Por lo tanto, se tiene éxito minimizando el máximo error ponderado a δ_s [17].

Esta técnica tiene el objetivo de determinar los coeficientes del filtro $h[n]$, de tal manera que el máximo error medido $|E(\omega)|$ esté minimizado en las bandas de paso y supresión [15]. Se ha establecido que cuando el $max|E(\omega)|$ se minimiza, la respuesta en frecuencia del filtro resultante tiene rizado constante en las bandas de paso y supresión. El cumplimiento de este objetivo lo ofrece el algoritmo de Parks-McClellan que provee una solución iterativa haciendo uso del algoritmo de intercambio de Remez para llegar a δ=δ_s. Cuando esta igualdad no se cumple entonces se debe incrementar M (si $\delta > \delta_s$) o decrementar M (si $\delta < \delta_s$) y usar el algoritmo Remez nuevamente para determinar el nuevo δ. Se repite el procedimiento hasta que $\delta \leq \delta_s$ [17].

Por otra parte, Káiser desarrollo la siguiente fórmula para determinar el orden N requerido:

$$N = \frac{-20log_{10}\sqrt{\delta_p\delta_s} - 13}{14.6\Delta_f} + 1$$

(20)

$$\Delta_f = \frac{\omega_s - \omega_p}{2\pi}$$

3.4 FILTROS IIR

Los filtros digitales de respuesta infinita al impulso IIR tienen en general una función de transferencia de la forma:

$$H(z) = \frac{\sum_{l=0}^{L-1} b_l \, z^{-l}}{1 + \sum_{m=1}^{M} a_m z^{-m}}$$

(21)

El problema de diseño consiste en encontrar los coeficientes b_l y a_m de modo que $H(z)$ satisfaga las condiciones dadas. Este filtro puede ser realizado también a través de una ecuación de diferencias:

$$y[n] = \sum_{l=0}^{L-1} b_l x[n-l] - \sum_{m=1}^{M} a_m y\,[n-m]$$

(22)

Dada la respuesta al impulso, la salida del filtro $y[n]$ también puede ser obtenida a través de una convolución lineal expresada como:

$$y[n] = x[n] * h[n] = \sum_{k=0}^{\infty} h[k]x[n-k],$$

(23)

la cual no es realizable computacionalmente porque esta hace uso de un número infinito de coeficientes. Por consiguiente se restringirá la atención en los filtros IIR que son descritos por la ecuación de diferencias dada en la ecuación (22) [13].

El diseño de filtros IIR usualmente parte de una función de transferencia de un filtro analógico $H(s)$. Ya que el diseño de filtros analógicos es un campo maduro y bien desarrollado, no es sorpresa que el diseño de un filtro IIR inicie en el dominio analógico y luego se convierta el diseño al dominio digital. El trabajo está en determinar un filtro digital $H(z)$ que se aproxime al rendimiento del filtro analógico deseado $H(s)$.

Se presenta a continuación el caso de diseño de filtro Butterworth. Para más información de otras aproximaciones como Chevyshev y Cauer ver [14] [17].

3.4.1 METODO DE DISEÑO DE BUTTERWORTH

Existen diferentes aproximaciones de los prototipos de filtros ideales pasa bajos en que se pueden basar los diseños, como son los filtros de Chevyshev tipo I y II, elípticos o de Cauer, de Bessel y de Butterworth. Esta última se explicara en detalle a continuación.

Este filtro se caracteriza porque presenta una respuesta de magnitud plana en las bandas de paso y rechazo es también llamado una aproximación todo polo[15] del filtro ideal. El cuadrado de la magnitud del filtro de orden N con frecuencia de corte Ω_c en rad/s, se determina como:

[15] Los filtros todo polo son aquellos que solo tiene términos en el denominador de su función de transferencia.

$$|H(\Omega)|^2 = \frac{1}{1 + (\Omega/\Omega_c)^{2N}} \ , \tag{24}$$

Ω es la variable que representa las frecuencias analógicas.

El filtro cumple con $|H(0)| = 1$ y $|H(\Omega_c)| = 1/\sqrt{2}$ o expresado como $20log_{10}|H(\Omega_c)| = -3dB$ para todos los valores de N, por lo que Ω_c se conoce como frecuencia de corte de 3dB.

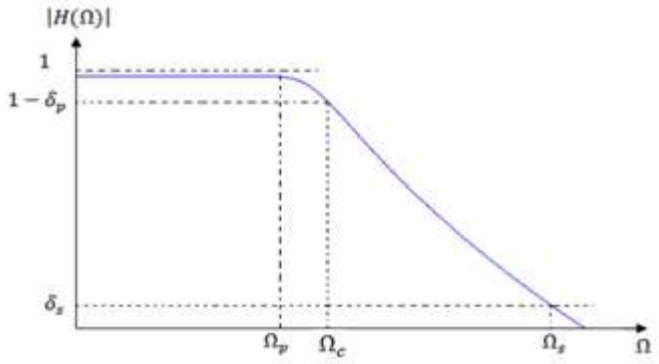


Figura 3. 10 ***Respuesta en magnitud de un filtro pasa bajo de Butterworth***

Una de las características del filtro es que la magnitud decrece monotónicamente tanto en la banda de paso como en la de supresión, además de tener una respuesta en magnitud totalmente plana en ambas bandas por lo que es llamado filtro máximamente plano (ver figura 3.10). La banda de paso plana es alcanzada a expensas de una región de transición que va desde Ω_p a Ω_s, la cual tiene una pendiente de caída demasiado pequeña. Además la respuesta en fase alrededor de la frecuencia de corte es no lineal como se puede observar en la Figura 3.11 [13].

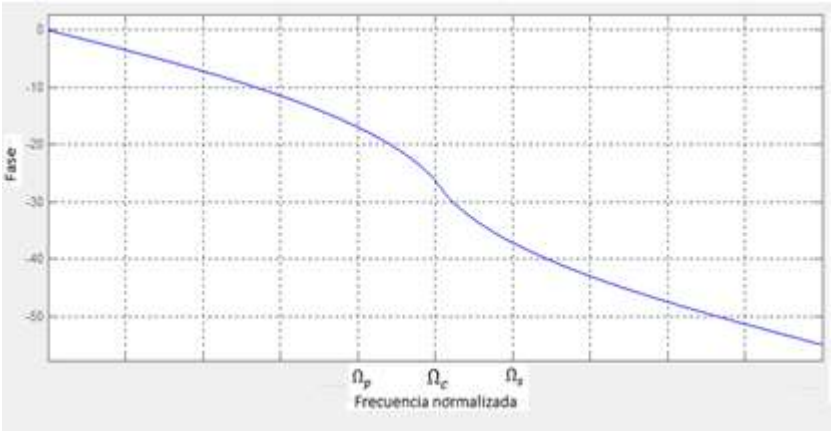


Figura 3. 11 ***Respuesta en fase de un filtro pasa bajo de Butterworth***

El filtro analógico Pasa Bajo es especificado por los parámetros Ω_p, A_p, Ω_s y A_s. Por lo tanto la esencia del diseño en el caso del filtro de Butterworth es encontrar el orden N y la frecuencia de corte Ω_c que satisfagan las especificaciones dadas. Se desea que:

- En $\Omega = \Omega_p$, $A_p = -10\log_{10}|H(\Omega)|^2$ o $A_p = -10\log_{10}\left(\frac{1}{1+\left(\frac{\Omega_p}{\Omega_c}\right)^{2N}}\right)$

(25)

- En $\Omega = \Omega_s$, $A_s = -10\log_{10}|H(\Omega)|^2$ o $A_s = -10\log_{10}\left(\frac{1}{1+\left(\frac{\Omega_s}{\Omega_c}\right)^{2N}}\right)$

(26)

Resolviendo estas dos ecuaciones se puede encontrar N y Ω_c. Entonces se tiene [17]:

$$N = \log_{10}\frac{\left[(10^{\frac{A_p}{10}}-1)/(10^{\frac{A_s}{10}}-1)\right]}{2\log_{10}(\Omega_p/\Omega_s)}$$

(27)

$$\Omega_c = \Omega_p[(1-\delta_p)^{-2}-1]^{-\frac{1}{2N}}$$

(28)

Generalmente N no es un entero, entonces si se quiere que N sea un entero se debe escoger el entero superior. Con esta información se calculan los polos de acuerdo a la siguiente ecuación:

$$s_k = -\Omega_c \sin\left[\frac{(2k+1)\pi}{2N}\right] + j\Omega_c \cos\left[\frac{(2k+1)\pi}{2N}\right], \quad 0 \le k \le N-1,$$

(29)

y finalmente se calcula la función de transferencia:

$$H(s) = \prod_{k=0}^{N-1}\frac{s-s_k}{s-p_k},$$

(30)

que para ser implementada en Matlab® se la lleva a la forma indicada por la ecuación (31):

$$H(s) = \frac{b_0 + b_1 s + \cdots + b_{N-1}s^{N-1}}{a_0 + a_1 s + \cdots + a_N s^N}$$

(31)

3.4.2 TRANSFORMACION BILINEAL

Después de realizar un acercamiento al diseño de filtros analógicos, estos se pueden transformar en filtros digitales. Este tipo de transformaciones son desarrolladas con el fin de preservar algunas características de los filtros analógicos. Por ejemplo la

transformación bilineal conserva las características de la función de transferencia del sistema al convertir el filtro del dominio analógico al dominio digital.

La transformación bilineal es un método en el dominio de la frecuencia que convierte la función de transferencia de un filtro analógico $\mathrm{H(s)}$ es una de un filtro digital $\mathrm{H(z)}$. La transformación es efectuada al realizar el siguiente cambio de variables:

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{32}$$

donde s es la variable en el plano complejo para señales continuas, z es la variable del plano complejo para señale discretas y T corresponde al periodo de muestreo.

Es llamada transformación bilineal por que tanto el numerador como el denominador de esta transformación son ecuaciones lineales.

Las frecuencias analógica y digital están relacionadas por medio de:

$$\Omega = \frac{2}{T}\tan\left(\frac{\omega}{2}\right), \tag{33}$$

en donde Ω es la frecuencia de borde del filtro analógico, ω es la frecuencia de borde deseada del filtro digital y T es el periodo de muestreo.

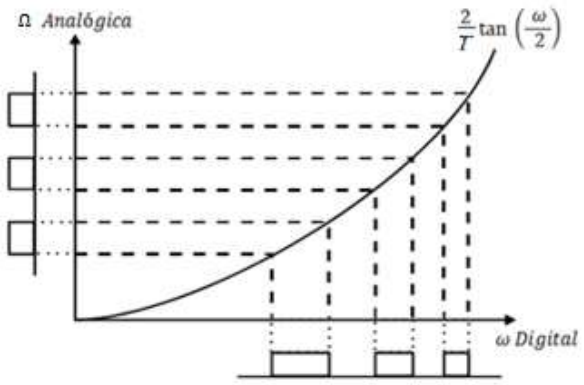


Figura 3. 12 ***Relación entre las frecuencias analógica y digital para transformación bilineal***

La ecuación (33) es también llamada pre-distorsión de las frecuencias analógicas. La figura 3.12 muestra que la relación entre Ω y ω es aproximadamente lineal para pequeños valores de ω pero se vuelve aceleradamente no lineal al incrementarse ω. Esta no linealidad genera una distorsión de la respuesta en frecuencia digital. Efecto

puede ser evitado a través de una pre-distorsión de las frecuencias analógicas para después aplicar la transformación [16].

Si se desean diseñar otro tipo de filtros selectivos en frecuencia a partir de un prototipo pasa bajo se deben utilizar las sustituciones mostradas en las tablas de Transformaciones en el dominio Analógico y de Transformaciones en el dominio Digital, mostradas en *Practical Digital Signal Processing for Engineers and Technicians* de *Edmund Lai* [16].

3.5 PROCESO DE DISEÑO DE FILTROS DIGITALES FIR E IIR

Los diseños que se realizan en las prácticas son de filtros selectivos en frecuencia, es decir, se pueden diseñar filtros pasa bajo, pasa altos, elimina banda y pasa banda, mediante diferentes técnicas de diseño de filtros digitales. Por facilidad se implementaran filtros tipo I, ya que su función de respuesta al impulso no tiene discontinuidades a menos que se deseen y son los que permiten implementar cualquier clase de filtro. Existen además otros tipos de filtros; como el tipo II el cual no es adecuado para el diseño filtros pasa alto o elimina banda, el tipo III el cual no es adecuado para el diseño de filtros pasa bajo, pasa alto o elimina banda y tipo IV el cual no se adecuado para el diseño de filtros pasa bajo o elimina banda. La Figura 3.13 muestra el diagrama de flujo general para el diseño de filtros digitales FIR e IIR:

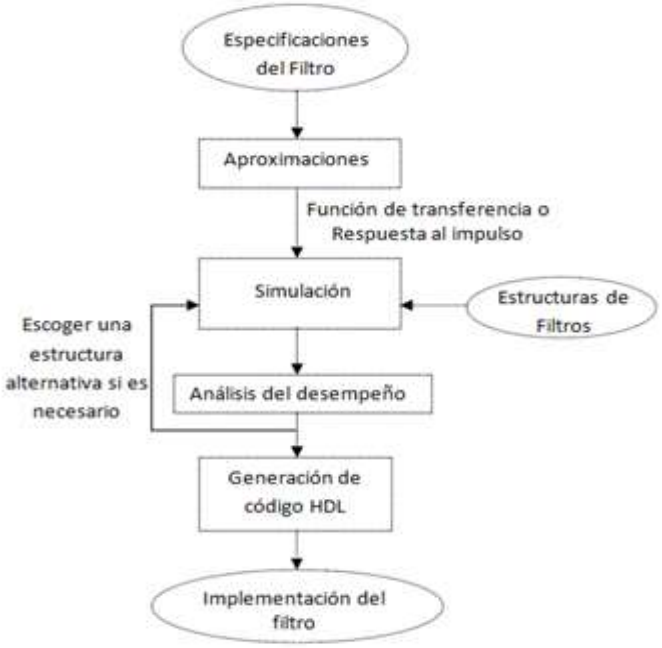


Figura 3. 13 ***Pasos en el diseño de filtros digitales FIR e IIR [16]***

3.5.1 ESPECIFICACIONES DE LOS REQUERIMIENTOS DEL FILTRO

Dado que no es necesario que la magnitud de la respuesta en frecuencia $|H(w)|$ sea constante en toda la banda de paso y sea cero en toda la banda de supresión, se

permite una pequeña cantidad de rizado tanto en la banda de paso como en la banda de supresión, como se muestra en la Figura 3.14a, donde se ilustra un filtro pasa bajo que será la base del siguiente análisis: La transición de la respuesta en frecuencia de la banda de paso a la banda de supresión se define como la banda de transición del filtro. La frecuencia ω_p delimita el final de la banda de paso, mientras que la frecuencia ω_s delimita el comienzo de la banda de supresión. Entonces el ancho de la banda de transición viene dado por $\omega_s - \omega_p$.

El rizado en la banda de paso se denota como δ_p, y la magnitud de la respuesta en frecuencia $|H(w)|$ varía entre $1 \pm \delta_p$. El rizado en la banda de supresión se denota como δ_s. Sin embargo para acomodar un rango dinámico amplio en la gráfica de la respuesta en frecuencia de un filtro es común en la práctica usar una escala logarítmica para la magnitud $|H(w)|$ [14]. Por lo tanto δ_s y δ_p que son llamadas las especificaciones absolutas se pasan a sus equivalentes en decibeles A_s y A_p que son llamadas las especificaciones relativas. Esto es:

$$dBscale = -20log_{10}\frac{\left|H\left(e^{j\omega}\right)\right|}{|H(e^{j\omega})|_{max}} \geq 0$$

(34)

Partiendo de que $\left|H(e^{j\omega})\right|_{max}$ en las especificaciones absolutas es igual a $(1+\delta_p)$, se tiene que:

$$A_p = -20log_{10}\frac{1-\delta_p}{1+\delta_p} > 0\ (\approx 0)$$

(35)

y

$$A_s = -20log_{10}\frac{\delta_s}{1+\delta_p} > 0\ (\gg 1)$$

(36)

La forma relativa se usa en la práctica para ambos filtros, tanto FIR como IIR; esta se muestra en la Figura 3.14b.

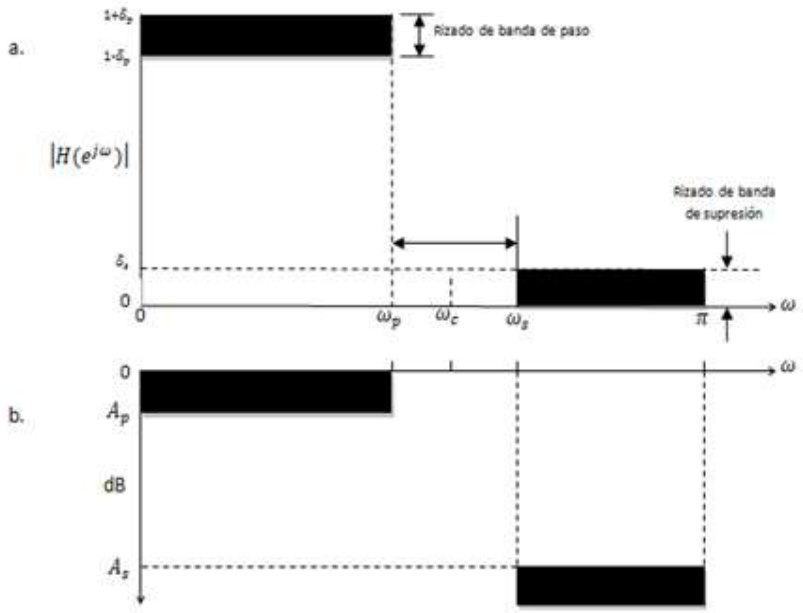


Figura 3. 14 ***Especificaciones de un filtro digital pasa bajo (a) absolutas, (b) relativas [17].***

En conclusión, las especificaciones de este filtro pasa bajo son:

- Banda de paso [0, ω_p], δ_p que es su correspondiente rizado en veces o A_p en dB
- Banda de supresión [ω_s , π], δ_s que es su correspondiente rizado en veces o A_s en dB
- Frecuencia de muestreo f_m [17].

Así como estas especificaciones fueron dadas para un filtro pasa bajo, especificaciones similares pueden ser también dadas para otros tipos de filtros selectivos en frecuencia. De igual manera, los parámetros de diseño más importantes para estos tipos de filtros son:

- La(s) frecuencia(s) de paso
- La(s) frecuencia(s) de supresión
- El máximo rizado permitido en la banda de paso
- El máximo rizado permitido en la banda de supresión
- Frecuencia de muestreo

3.5.2 APROXIMACION O CÁLCULO DE LA FUNCION DE TRANSFERENCIA DEL FILTRO

Este paso es llamado Aproximación debido a que lo que se está haciendo es en realidad buscar una función de transferencia que se aproxime a la respuesta ideal del filtro especificado. Los métodos para encontrar la solución al problema de aproximación de filtros digitales se pueden clasificar como directos o indirectos. Con los métodos directos

el problema es solucionado en el dominio discreto. Para los métodos indirectos primero se obtiene una función de transferencia de tiempo continuo usando uno de los métodos para el diseño de filtros analógicos, esta función de transferencia es transformada después en una función de transferencia de tiempo discreto, mediante los métodos de *impulso invariante* (ver [16]) o la *transformación bilineal*. Los métodos indirectos son usados comúnmente por los filtros IIR, mientras que los métodos directos son usados por los filtros FIR [16].

Teniendo en cuenta lo anterior, se usaron varios conceptos y teoría matemática estudiada en los cursos de procesamiento digital de señales, para iniciar con una descripción del filtro que se aproxime al conjunto dado de especificaciones. Los cálculos matemáticos fueron escritos en la herramienta Matlab®, con ayuda de algunas de sus funciones predefinidas. Para información sobre las herramientas de Matlab® necesarias para la realización de las Prácticas de laboratorio, consultar en Anexo A. La Figuras 3.15 y 3.16 muestran el diagrama de flujo general de cálculos para hallar la respuesta al impulso de los filtros FIR y el diagrama de flujo general de cálculos para hallar la respuesta al impulso de los filtros IIR, respectivamente.

3.5.3 SIMULACION DE FILTROS FIR E IIR

Para hacer la simulación de los algoritmos de filtrado descritos en MATLAB® se utilizaron algunos comandos dentro del mismo código de diseño así como también se usó la herramienta Simulink®. Se expone el segundo método ya que gracias a la herramienta Simulink® se logra representar el filtro diseñado en bloques y además facilita la generación de código VHDL, que posteriormente será implementado en la FPGA.

El diagrama de bloques general que se desea implementar utilizando la herramienta Simulink® para simular los filtros FIR e IIR es el mostrado en la figura 3.17:

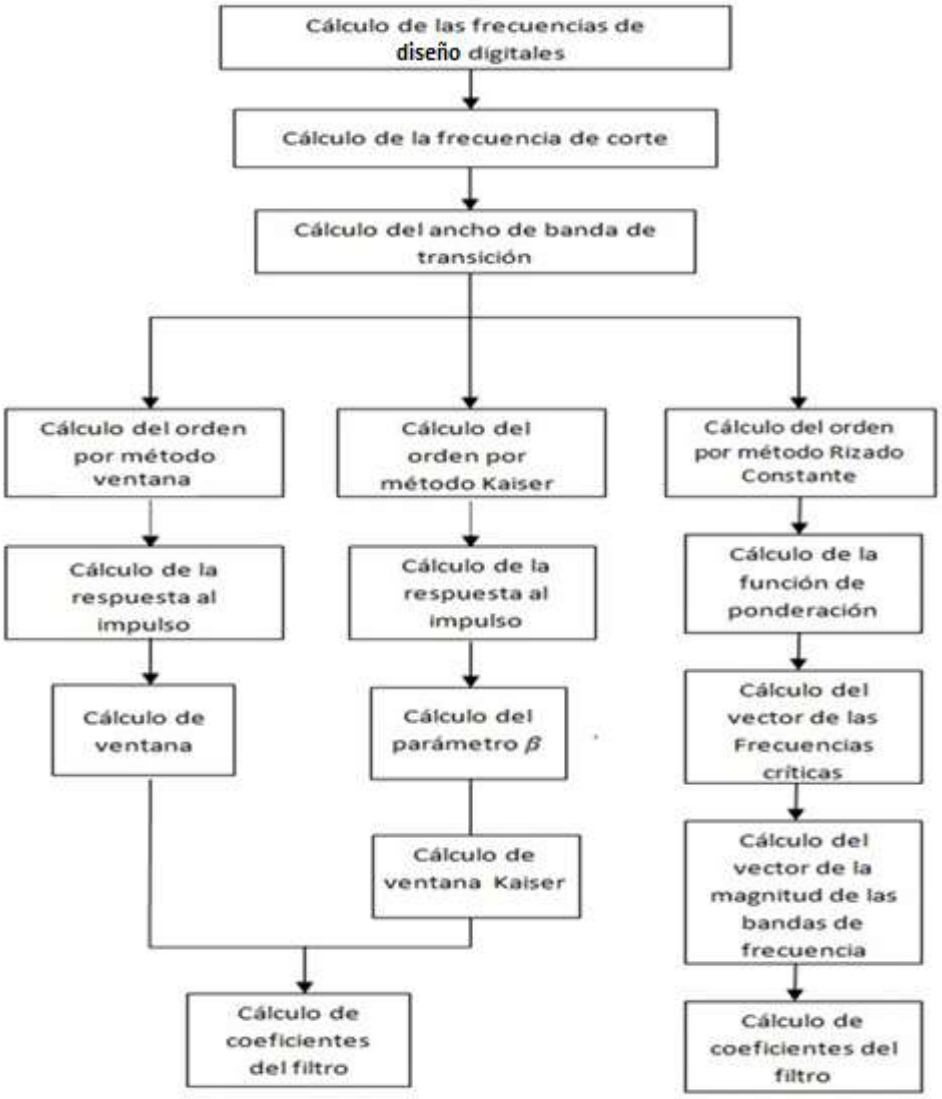


Figura 3. 15 ***Diagrama de flujo general de aproximaciones para filtros FIR***

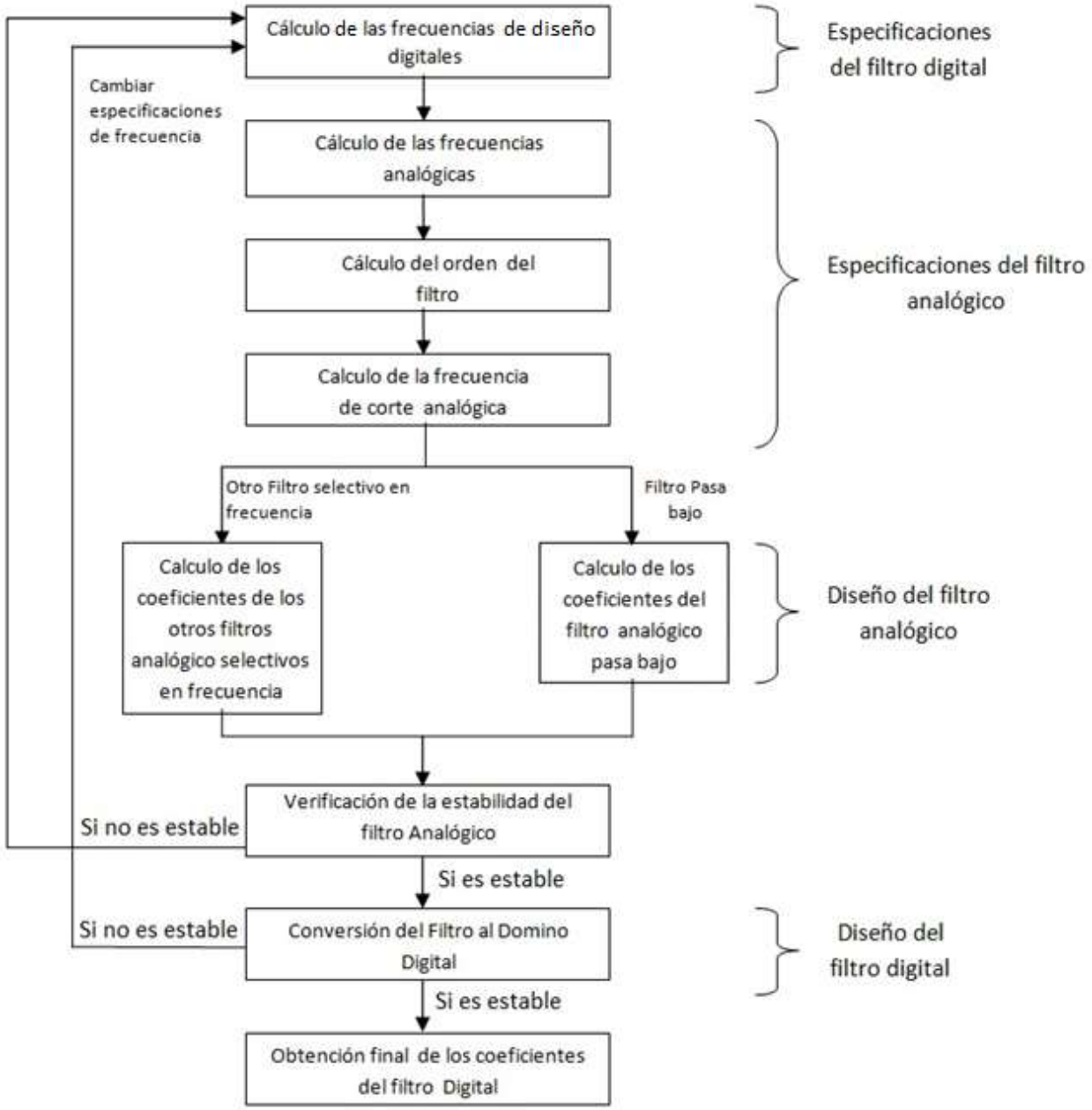


Figura 3. 16 ***Diagrama de flujo de las aproximaciones para filtro IIR***

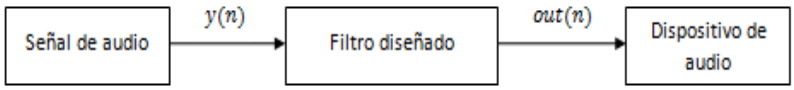


Figura 3. 17 ***Diagrama inicial de bloques del proceso general de simulación para filtros FIR e IIR***

Este diagrama de la figura 3.17 queda representado mediante los siguientes bloques en Simulink®:

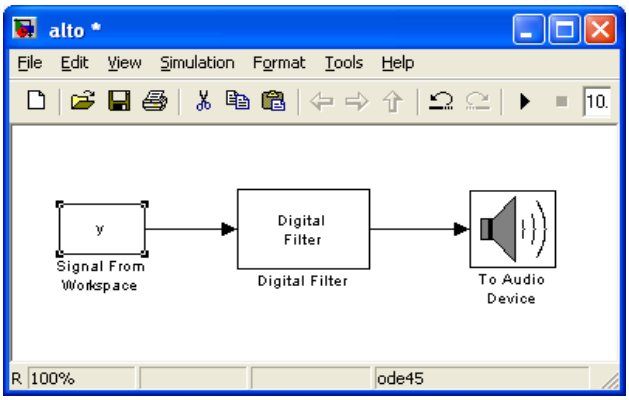


Figura 3. 18 ***Modelo en Simulink® para los filtros FIR e IIR diseñados***

A continuación se expondrán algunas características de cada uno de los bloques del modelo Simulink de la figura 3.18:

- ***Signal From Workspace***: Este bloque importa una señal desde el área de trabajo de MATLAB® a un modelo de Simulink®:

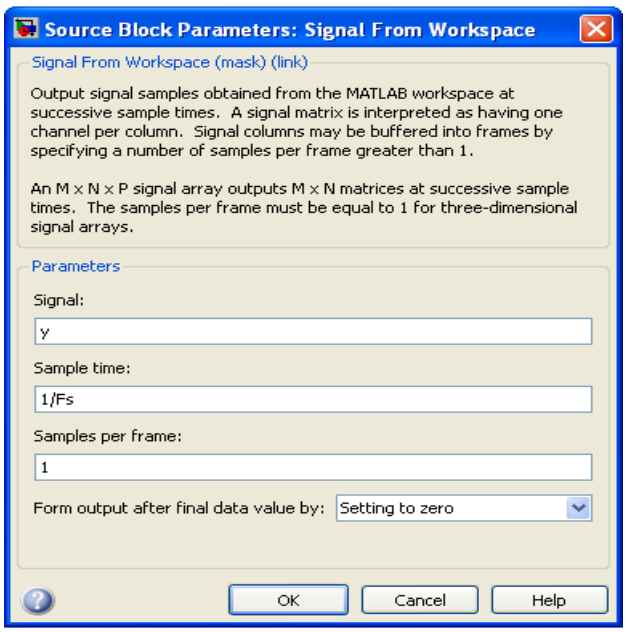


Figura 3. 19 ***Recuadro de dialogo de Bloque Signal From Workspace***

Signal: Es el nombre de una variable en el área de trabajo de Matlab® que contiene la señal a importar, o cualquier expresión válida de Matlab® que especifique la señal.

Sample time: es el período de muestreo de salida T_s de la señal.

Samples per frame: es el número de muestras Mo por cada trama de la señal.

Form output after final data value by: Especifica la salida después de que todas las muestras de la señal especificada han sido generadas. El bloque puede dar ceros de salida (`Setting to zero`), repetir la muestra final de datos (`Holding Final Value`) o repetir la señal entera desde el principio (`Cyclic Repetition`) si la señal es más corta que el tiempo de simulación.

- ***Digital Filter***: Este bloque se utiliza para implementar filtros de punto flotante o de punto fijo de los cuales se conozcan los coeficientes. El bloque de *Digital Filter* puede implementar filtros estáticos con coeficientes fijos así como filtros variantes en el tiempo con coeficientes que cambian a través del tiempo.
  El estado de la trama de salida y sus dimensiones son siempre las mismas de la señal de entrada que está siendo filtrada.

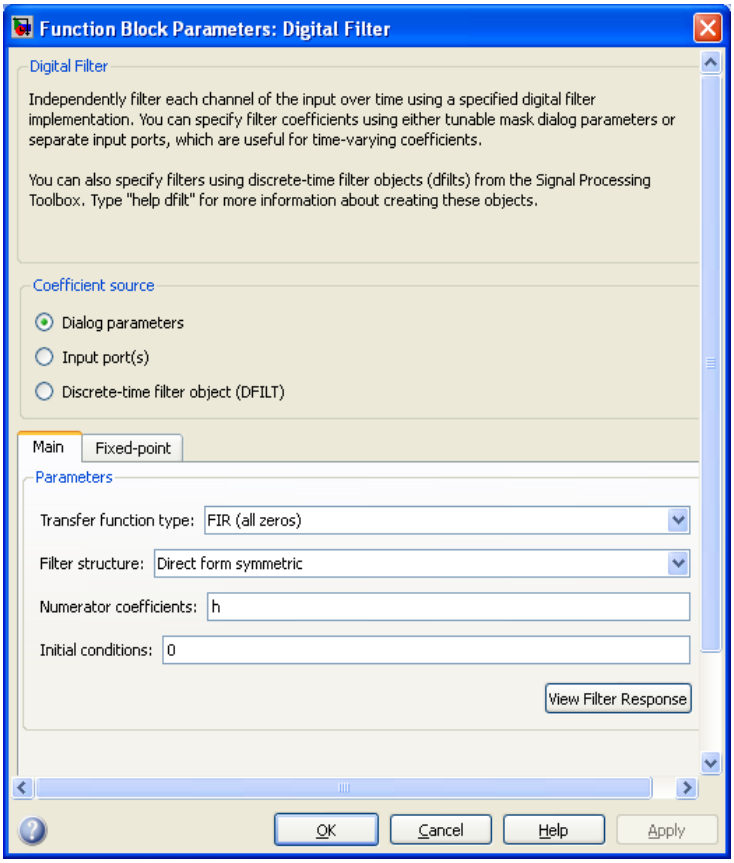


Figura 3. 20 ***Recuadro de Dialogo de bloque Digital Filter: Filtro FIR***

Coefficient source: El bloque *Digital Filter* puede operar en tres modos diferentes. Se selecciona el modo en este menú.

- ***Dialog parameters*****:** se debe digitar información sobre el filtro tales como la estructura y los coeficientes.
- ***Input port(s)****:* se debe digitar la estructura del filtro, y los coeficientes del filtro entran por uno o más puertos del bloque. Este modo es útil para especificar filtros variables en el tiempo.
- ***Discrete-time filter object (DFILT)***: se especifica el filtro usando un objeto `dfilt`[16].

Para el caso de las prácticas de laboratorio se utilizo el modo ***Dialog parameters*** así:

❖ **PARA FILTROS FIR**

Transfer function type: se selecciona el tipo de función de transferencia del filtro, en este caso FIR (todos ceros).

[16] dfilt: Es una función que permite crear un objeto de tipo dfilt (Filtro de tiempo discreto), cuenta con varias opciones de estructuras para diseñarlo y un grupo de métodos para acceder al objeto y modificar su información.

Filter structure: se selecciona la estructura del filtro. La selección de varias estructuras disponibles depende del tipo de función de transferencia. En este caso la estructura utilizada fue *Forma Simétrica Directa*. Esta reduce el consumo de recursos de la FPGA, lo que se puede observar el la gráfica 3.21, la cual fue obtenida de la interfaz de la herramienta *ISE Desing Suite* al implementar los diseños de los filtros con estructura directa y simétrica. La estructura directa puede también ser usada pues es la más común para realizar filtros no recursivos y su atractivo principal, es su simplicidad y facilidad de programación [15].

***Numerator coefficients*:** se especifica el vector de coeficientes en el numerador de la función de transferencia del filtro.

Initial Conditions: se especifican las condiciones iníciales de los elementos de retardo. Se asume que son cero que equivale a asumir que las entradas y salidas pasadas son cero. El bloque inicializa todos los elementos de retardo en el filtro en cero.

***View Filter Response*:** Este botón abre la Herramienta de Visualización del filtro *fvtool* de la *Signal Processing Toolbox* y muestra la respuesta en frecuencia del filtro FIR definido por el bloque.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	978	11,776	8%	
Number of 4 input LUTs	2,174	11,776	18%	
Number of occupied Slices	1,636	5,888	27%	
Number of Slices containing only related logic	1,636	1,636	100%	
Number of Slices containing unrelated logic	0	1,636	0%	
Total Number of 4 input LUTs	2,314	11,776	19%	
Number used as logic	2,174			
Number used as a route-thru	140			
Number of bonded IOBs	17	372	4%	
Number of BUFGMUXs	3	24	12%	
Number of MULT18X18SIOs	20	20	100%	
Average Fanout of Non-Clock Nets	1.67			

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	950	11,776	8%	
Number of 4 input LUTs	4,454	11,776	37%	
Number of occupied Slices	2,940	5,888	49%	
Number of Slices containing only related logic	2,940	2,940	100%	
Number of Slices containing unrelated logic	0	2,940	0%	
Total Number of 4 input LUTs	4,900	11,776	41%	
Number used as logic	4,440			
Number used as a route-thru	446			
Number used as Shift registers	14			
Number of bonded IOBs	17	372	4%	
Number of BUFGMUXs	3	24	12%	
Number of MULT18X18SIOs	20	20	100%	
Average Fanout of Non-Clock Nets	1.81			

Figura 3. 21 ***Consumo de Recursos en la FPGA de estructura Forma Directa Simétrica FIR (arriba) y Consumo de Recursos de Estructura Forma Directa FIR (abajo).***

❖ PARA FILTROS IIR

***Transfer Function Type*:** En este caso IIR (polos y ceros).

Filter Structure: En este caso se selecciona la estructura Forma Directa II Transpuesta SOS Bicuadrada, ya que para generar código VHDL a partir del modelo Simulink®, este tipo de estructura SOS (Secciones de segundo orden) es la única soportada por la herramienta *Simulink® VHDL Coder* que es la encargada de generar el código VDHL.

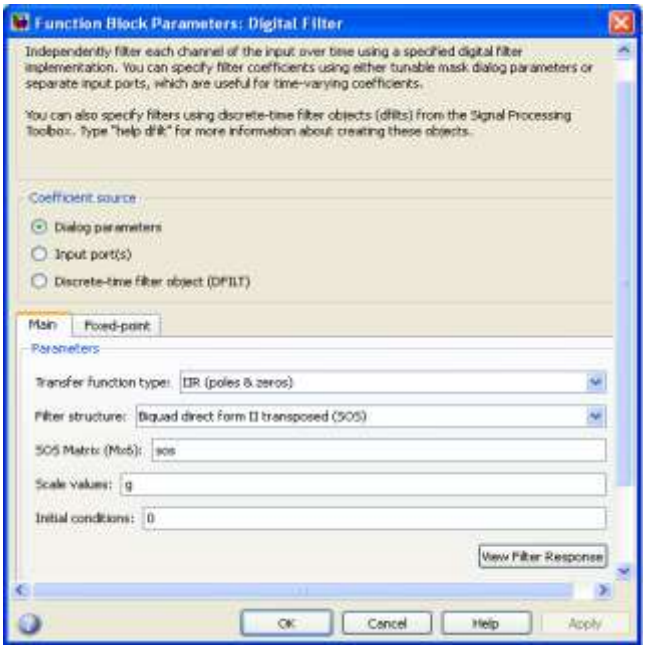


Figura 3. 22 ***Recuadro de dialogo de bloque Digital Filter: Filtro IIR***

SOS Matrix (Mx6)**:** se especifica la matriz `sos` que contiene los coeficientes del filtro de seccciones de segundo orden.

Scale Values**:** se especifica el valor escalar para ser aplicado antes y después de cada sección de un filtro bicuadrado.

Initial Conditions: se aplican las mismas condiciones de los filtros FIR

- ***To audio device***: Este bloque envía datos de audio a un dispositivo de audio del computador.

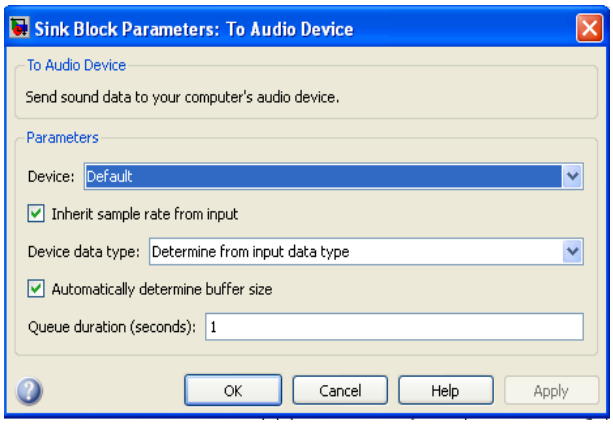


Figura 3. 23 ***Recuadro de dialogo de bloque To Audio Device***

Device: Se especifica el dispositivo al que se desea enviar los datos de audio.

Inherit sample rate from input: se selecciona esta casilla de verificación para que el bloque herede la frecuencia de muestreo de la señal de audio de la entrada al bloque.

Device data type: se especifica el tipo de datos de audio enviada al dispositivo.

Automatically determine buffer size: se selecciona esta casilla para permitir que el bloque calcule un tamaño del búfer.

Queue duration (seconds): se especifica el tamaño de la cola en segundos.

3.5.4 ANALISIS DEL DESEMPEÑO DE FILTROS FIR E IIR

Este paso comprende el análisis de resultados gráficos y auditivos. De no cumplirse los requisitos iníciales del diseño, se debe realizar cambios en los parámetros de diseño o en la configuración de los bloques de Simulink®.

3.5.5 GENERACIÓN DE CÓDIGO VHDL

En el hardware digital, los números son almacenados en palabras binarias. Una palabra binaria es una secuencia de longitud finita de bits (1's y 0's). La forma como los componentes hardware o las funciones software interpretan esta secuencias de 1's y 0`s se define por el tipo de dato. Los números binarios son representados como tipo de datos en punto fijo o punto flotante [18]. En la aritmética de punto fijo, el punto binario se encuentra siempre en la misma posición, es decir, existirán 1 bit para signo, m bits para la parte entera y n bits para la parte decimal. En algunos casos puede ocurrir que m = 0 (no existe parte entera) o bien, n = 0 (no existe parte decimal). En cambio, para la aritmética de punto flotante, la ubicación del punto binario puede variar, ya que existirán 1 bit para signo, 1 bit para signo del exponente, m bits para exponente, n bits para la parte real. Implementar por hardware una aritmética de punto fijo conlleva una mayor simplicidad, lo cual se traduce directamente en menores costos. Además ocupa menor superficie de silicio respecto a una unidad de punto flotante, lo que permite agregar al procesador más módulos y memoria. Entre las desventajas más significativas se encuentra el hecho de que, para lograr en punto fijo la misma precisión que se logra con punto flotante se necesitaría una cantidad muy grande de bits [19].

Debido a que la aritmética de punto fijo es la más sencilla de implementar y a que está soportada por la mayoría de la Toolboxes de Matlab®, será usada para realizar la respectiva representación de las variables de los filtros digitales. Además al generar código VHDL en punto flotante con la herramienta *Simulink® VHDL Coder* la declaración de los puertos de entrada y salida del filtro digital no es concurrente con las restricciones impuestas por Xilinx® en el XST[17], ya que están declaradas de tipo real y este tipo de dato solo es soportado para el cálculo de las constantes dentro de funciones [20].

[17] XST-Xilinx Synthesis Technology : La Tecnología de Síntesis Xilinx es una herramienta de Xilinx, que sintetiza diseños HDL para crear archivos netlist específicos de Xilinx llamados NGC. El archivo NGC es una lista de red que contiene los datos de diseño lógico y restricciones

Para que el diseño sea en punto fijo se configuran las opciones de la segunda pestaña del recuadro de diálogo del bloque *Digital Filter* (Figura 3.20) que corresponde a la parte de Fixed Point. Además se hace necesario la utilización de dos nuevos bloques llamados *Convert* y *Gain* (Figura 3.24):

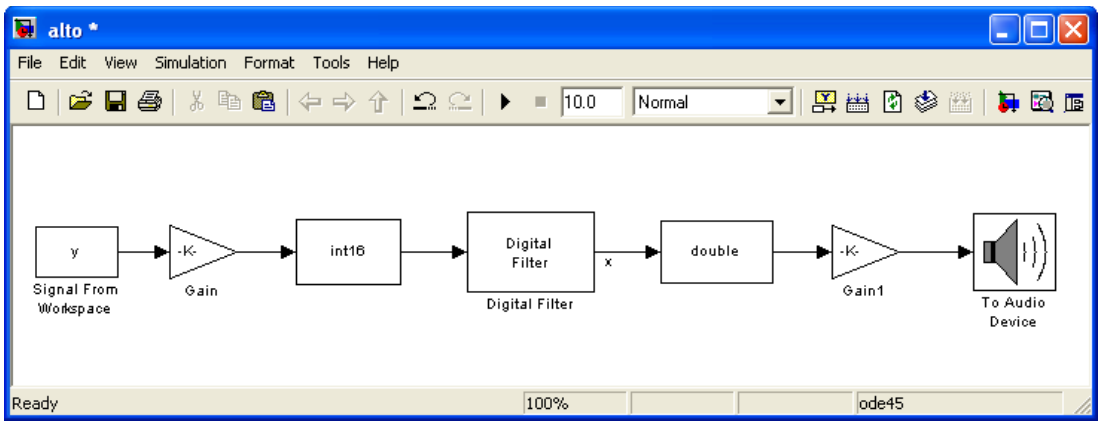


Figura 3. 24 ***Modelo Simulink® en Punto Fijo***

El bloque *Convert* es el encargado de convertir la señal de audio de entrada a punto fijo, luego al configurar la pestaña *Fixed Point* del bloque *Digital Filter,* este se encargará de convertir los coeficientes del filtro al mismo tipo de dato de la entrada que le está llegando, es decir a punto fijo, en este caso las variables fueron convertidas a int16[18].
Cuando se tiene la señal filtrada, se utiliza nuevamente el bloque *Convert,* para convertir las variables de int16 a tipo *double*[19] y así poder escuchar la señal filtrada en el bloque *To Audio Device*
En este tipo de modelos pequeños, es conveniente utilizar el bloque *Convert* para realizar la conversión de punto flotante a punto fijo, sin embargo cuando los modelos son mucho más grandes y complejos, existe una herramienta en Matlab®, que en estos casos es la adecuada para realizar la conversión de tipo de dato. Esta herramienta se llama *Fixed-Point Advisor*, la cual esta divida en 4 tareas principales, donde cada tarea atiende un aspecto del proceso de conversión. Para más información de esta herramienta consultar *Help* en Matlab®.

El bloque *Gain* se hace necesario dependiendo de si las amplitudes de la señal cargada en el bloque *Signal from Workspace* están distribuidas en un rango de enteros demasiado pequeño, pues esto ocasionaría que la conversión de *double* a *int16* se efectúe de manera incorrecta ya que muchas amplitudes serian aproximadas a un mismo entero y se perderían datos, por lo tanto se debe aplicar una ganancia a la señal para ampliar el rango en el que va a quedar definida. Por ejemplo, valores de amplitud que se encuentren entre ∓ 0.8 se aproximaran al entero más cercano dependiendo del

[18] int16: convierte un elemento a un entero de 16 bits con signo, el rango de valores de salida es de -32768 a 32767.
[19] double: tipo de dato de doble precisión.

método de redondeo utilizado en bloque *Convert,* es decir que esas amplitudes se aproximaran a 0 o a 1, lo cual generaría una simulación deficiente.

Luego de adicionar el bloque *Gain* y el bloque *Conver* al modelo inicial, éste estará listo para la generación de código VHDL utilizando la herramienta *Simulink® VHDL Coder*.

El *Simulink® VHDL Coder* genera varios archivos. El único archivo que se utilizara será el que tiene el nombre del bloque del filtro, es decir Digital_Filter.vhd, donde se declaran los puertos de entrada y salida y la arquitectura del filtro, cada puerto con nombre, modo y tipo y la arquitectura con las constante, las señales y los procesos que realiza internamente el filtro digital.

Para poder implementar los filtros en la FPGA, es necesario utilizar los conversores ADC y DAC que se explicaran en el Capítulo 4. Por lo tanto se realiza la programación de los conversores en lenguaje VHDL y se integran con el archivo Digital_Filter.vhd en el entorno de desarrollo *ISE Desing Suite*. Para la programación de los conversores se realizó la máquina de estados que se puede consultar en el Anexo B. Para más información consulte el *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®*.

Se debe tener en cuenta que cuando la señal analógica sale del conversor ADC, está representada por una palabra digital de 14bits longitud, 1 bit para signo, al llegar al filtro es necesario adicionarle dos bits “00” a la izquierda para completar la palabra de 16 bits con la que trabaja el filtro en la FPGA. Dado que el conversor DAC trabaja con 12 bits y la salida del filtro es una palabra de 16 bits se hace necesario remover 4 bits, para que el conversor pueda trabajar. Los bits removidos son los bits más significativos.

3.5.6 IMPLEMENTACIÓN DE FILTROS

En este paso final se implementaran los códigos VHDL generados para cada uno de los modelos Simulink®. Para este objetivo se utiliza una etapa de adecuación para la señal de audio de entrada a la FPGA que será explicada en el siguiente capítulo.

3.6 FILTROS ADAPTATIVOS

El diseño de filtros digitales con coeficientes fijos requiere que las especificaciones deseadas estén bien definidas, pero hay situaciones en las que estas especificaciones no están disponibles o son variantes con el tiempo. La solución en este caso es utilizar un filtro digital con coeficientes adaptables, conocido como filtro adaptativo.

Dado que las especificaciones no están disponibles, el algoritmo adaptativo que determina la actualización de los coeficientes del filtro, requiere información extra que usualmente es dada por una señal. Esta señal es llamada generalmente señal deseada o señal de referencia, y su elección depende de la aplicación.

Los filtros adaptativos son considerados sistemas no lineales, por lo tanto el análisis de su comportamiento es más complicado que el de los filtros de coeficientes fijos [21].
Un filtro adaptativo se constituye de dos partes distintas, un filtro digital, que ejecuta el procesamiento de la señal deseada y un algoritmo adaptativo, que ajusta los coeficientes del filtro. La forma general de un filtro adaptativo es mostrada en la figura 3.25, donde $d[n]$ es la señal deseada, $y[n]$ es la señal de salida del filtro digital y $e[n]$ la señal de error, la cual es la diferencia entre $d[n]$ y $y[n]$.

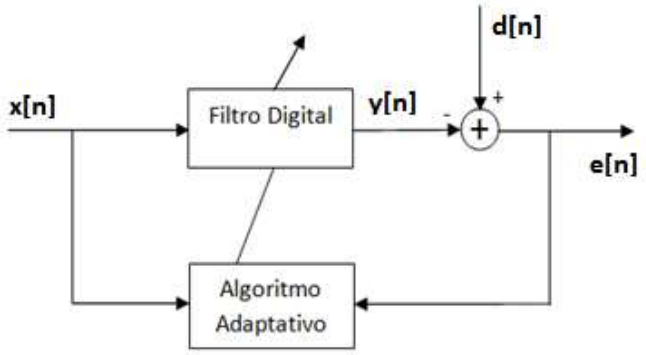


Figura 3. 25 ***Diagrama de bloques de un filtro adaptativo.***

La función del algoritmo adaptativo es ajustar los coeficientes del filtro para minimizar la señal $e[n]$, así como los coeficientes del filtro son actualizados, el error es progresivamente minimizado también muestra por muestra [13].

Tanto los filtros FIR como los IIR pueden ser usados para el filtrado adaptativo, los filtros FIR son sin embargo los más usados en la práctica. La razón de esta preferencia es simple, los filtros FIR tiene solamente ceros por ajustar y esto los hace libres de problemas de estabilidad, los cuales están asociados con los filtros adaptativos IIR, que tiene ceros y polos por ajustar. No por esto se puede concluir que los filtros FIR son siempre estables, por el contrario la estabilidad de los filtros depende críticamente del algoritmo adaptativo.

La estructura más ampliamente usada para la implementación de filtros adaptativos FIR es la estructura transversal, mostrada en la figura 3.26,

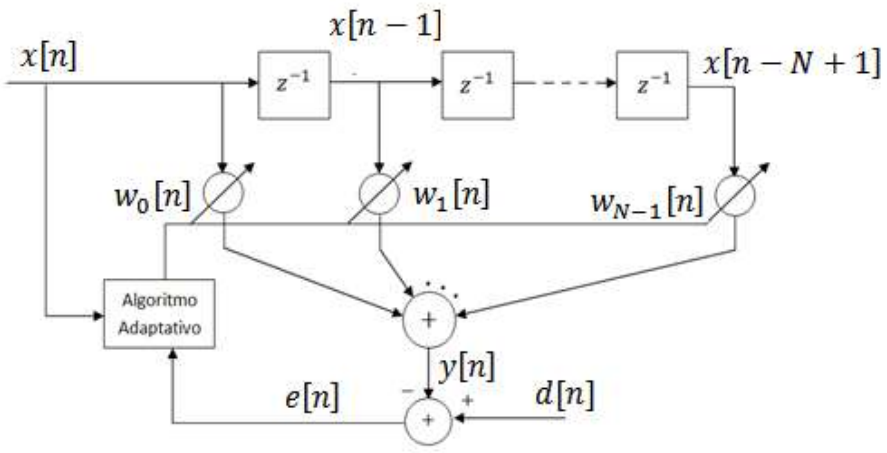


Figura 3. 26 ***Estructura filtro adaptativo transversal***

en la cual, el filtro adaptativo tiene una sola entrada $\boldsymbol{x}[n]$ y una salida $y[n]$; $d[n]$ es la señal deseada. La salida $y[n]$ es generada como una combinación lineal de muestras retrasadas de la secuencia de entrada $\boldsymbol{x}[n]$, de acuerdo con la siguiente ecuación:

$$y[n] = \sum_{i=0}^{N} \boldsymbol{w}_i[n]\boldsymbol{x}[n-i] \tag{37}$$

$\boldsymbol{w}_i[n]$ representa los coeficientes del filtro, $N+1$ es la longitud del filtro, $\boldsymbol{x}[n-i]$ representa las muestras de entrada del filtro. Los coeficientes $\boldsymbol{w}_i[n]$ son variables en el tiempo y son controlados por el algoritmo adaptativo [22].

Algunos algoritmos Adaptativos se exponen a continuación:

3.6.1 ALGORITMO LMS

El algoritmo LMS (Least Mean Square) es un algoritmo de filtrado adaptativo lineal, cuya simplicidad casi lo ha convertido en estándar; consta básicamente de dos procesos básicos:

I. Un proceso de filtrado, que consiste en:
   - Cálculo de la salida del filtro transversal $y[n]$ producida por un conjunto de muestras de entrada $\boldsymbol{x}[n]$.
   - Generar una estimación de error e$[n]$ mediante la comparación de $y[n]$con la señal deseada $d[n]$

II. Un proceso adaptativo, que consiste en el ajuste automático del vector de coeficientes del filtro $\boldsymbol{w}[n]$ de acuerdo con la estimación del error.

Luego la combinación de estos dos procesos trabajando juntos constituyen el lazo de realimentación alrededor del algoritmo LMS.

Las ecuaciones que describen este algoritmo son:

$$y[n] = \mathbf{w}^{\mathrm{T}}[n]\boldsymbol{x}[n] \tag{38}$$

$$\mathrm{e}[n] = d[n] - y[n] \tag{39}$$

$$\mathrm{w}[n+1] = \mathbf{w}[n] + \mu\boldsymbol{x}[n]e[n] \tag{40}$$

donde:
$\boldsymbol{x}[n]$: Vector de la señal de entrada.
$\mathbf{w}[n]$: Vector de coeficientes del filtro adaptativo.
$d[n]$: Escalar que corresponde a la señal deseada en el instante n.
$e[n]$: Escalar que corresponde al error de medida en el instante n.
μ : Parámetro de ajuste fijo [24].

3.6.2 ALGORITMO ACELERADOR REGRESIVO VERSIÓN γ

El algoritmo acelerador regresivo versión γ fue propuesto por P E Jojoa [25], en busca de disminuir la complejidad computacional del algoritmo ARCM[20]. Este algoritmo se ajusta mediante tres parámetros α, γ y m_1, por medio de los cuales se logra una buena velocidad de convergencia y paralelamente una considerable reducción del error de medida final.

Las ecuaciones que describen este algoritmo son:

$$e[n] = \boldsymbol{x}^T[n]\boldsymbol{w}[n-1] - d[n], \tag{41}$$

$$g[n] = \frac{e[n]+\gamma\boldsymbol{x}^T[n]\boldsymbol{w}[n-1]}{1+\alpha\gamma\boldsymbol{x}^T[n]M_1\boldsymbol{x}[n]}, \tag{42}$$

$$q[n] = \frac{\gamma}{\alpha+\gamma}\left[q[n-1] - \alpha g[n]M_1\boldsymbol{x}[n]\right], \tag{43}$$

$$\boldsymbol{w}[n] = \boldsymbol{w}[n-1] + \alpha q[n], \tag{44}$$

donde $d[n]$ corresponde a la señal deseada obtenida de la siguiente forma:

$$d[n] = \boldsymbol{x}^T[n]\boldsymbol{w_0} + r[n]\,, \tag{45}$$

$\boldsymbol{x}[n]$ es la señal de entrada, $\boldsymbol{w}[n]$ es el vector de coeficientes del filtro adaptativo, $d[n]$ es el escalar que corresponde a la señal de entrada en el instante n, $e[n]$ es el escalar que corresponde al error de medida en el instante n, $g[n]$ es el escalar auxiliar en el instante n, q$[n]$ es el vector auxiliar, $\boldsymbol{w_0}[n]$ es el vector de coeficientes óptimo, α, γ, m_1 son parámetros de ajuste fijo y M_1 es matriz definida positiva de tal forma que $M_1 = m_1 I$, donde I corresponde a la matriz identidad.

[20] ARCM: Es el Algoritmo Acelerador Regresivo Convencional Matricial que fue obtenido con el método de Euler regresivo a partir del algoritmo Acelerador de Tiempo Discreto (Jojoa 2009)

La ventaja que presenta este algoritmo es que los parámetros de ajuste se reducen a cantidades escalares (α, γ, m_1). De acuerdo al análisis de convergencia con respecto a la medida realizada P E Jojoa, se establece que el algoritmo converge para valores α, γ, m_1 positivos $(\alpha > 0, \gamma > 0, m_1 > 0)$. Así mismo, del análisis de Tracking del algoritmo ARγ, en ambiente no estacionarios se determino que éste presenta un mínimo error de ajuste cuando $\alpha\, \gamma\, m_1 \approx H$ (Criterio de mínimo error) donde H es una constante real positiva con un valor aproximado a 2 [24].

3.7 PROCESO DE DISEÑO DE FILTROS ADAPTATIVOS

El diseño realizado en la práctica es del algoritmo LMS. En la Figura 3.27 se muestra el diagrama de flujo general para el diseño del filtro adaptativo:

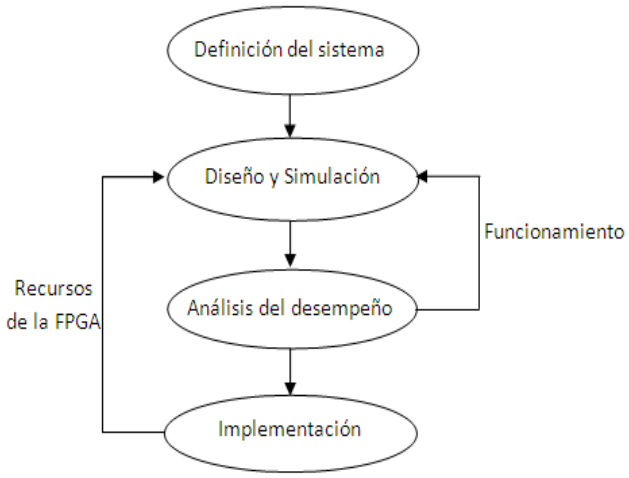


Figura 3. 277 ***Proceso General de diseño de Filtros Adaptativos***

3.7.1 DEFINICIÓN DEL SISTEMA

Los filtros adaptativos son sistemas que se auto-diseñan, lo que hace que se puedan adaptar a diferentes ambientes. Como resultado, los filtros adaptativos son útiles en campos tan diversos como el control, comunicaciones, radar y procesamiento de señales, cancelación de interferencias, el control activo del ruido, ingeniería biomédica, etc.

La característica común de estas aplicaciones es que se les imponen las mismas condiciones básicas de formulación de filtrado adaptativo, condiciones que implican un proceso de filtrado de la señal de entrada y la señal deseada. Como se sabe los parámetros del filtro se actualizan mediante una serie de mediciones de las señales subyacentes y esto lo realiza el algoritmo de filtrado adaptativo, el cual busca que la diferencia entre la salida del filtro y la respuesta deseada sea reducida al mínimo.

En este contexto, cuatro clases de aplicaciones básicas de filtrado adaptativo son reconocidas, estas son el modelado, el modelado inverso, la predicción lineal y cancelación de interferencia [26].

La aplicación de filtrado adaptativo definida como cancelación de interferencia, que se refiere a las situaciones en donde se requiere la cancelación de una señal de interferencia/ruido de una señal dada, la cual es una mezcla de la señal deseada y la señal de interferencia. El principio de cancelación de interferencia es obtener una estimación de la señal interferente y restarla de la señal contaminada. La viabilidad de esta idea depende de la disponibilidad de una fuente de referencia de la que se origina la señal interferente. La figura 3.28 representa el concepto de cancelación de interferencias en su forma más simple.

La entrada principal es la señal contaminada, es decir, la señal deseada más la señal de la referencia de entrada, la cual es generada por la fuente de interferencia solamente. El filtro adaptativo se ajusta para que una réplica de la señal de interferencia que está presente en la señal principal aparezca en su salida y(n). Restando esta salida de la entrada principal el resultado es una salida limpia de interferencias, de ahí el nombre de cancelación interferencia [26].

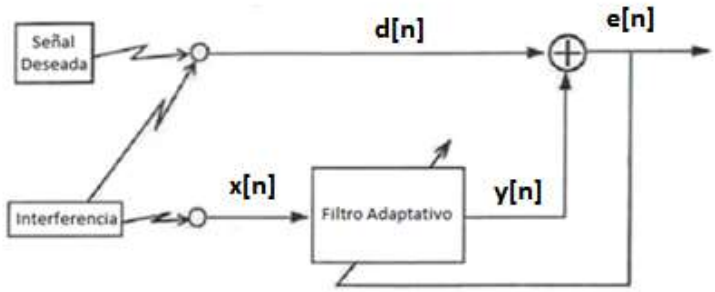


Figura 3. 288 ***Filtro Adaptativo Cancelador de Interferencia***

3.7.2 DISEÑO Y SIMULACIÓN DEL FILTRO ADAPTATIVO

El diseño del algoritmo adaptativo LMS se abstrajo del proyecto *Vectorized Adaptive noise canceler using LMS Filter* descargado de http://www.mathworks.com/Matlab®central/fileexchange/16278, el cual consta de dos diseños distintos del mismo algoritmo LMS, uno realizado con el bloque LMS Filter y el otro con el *Embededd Matlab® Subset*, que es un subconjunto del lenguaje Matlab®, que provee las herramientas para generar código C y VHDL eficiente para aplicaciones embebidas. Con este lenguaje trabaja el bloque *Embedded MATLAB® Function*, que permite adicionar funciones de Matlab® a modelos en Simulink®. Esta capacidad es útil para algoritmos que están mejor declarados en lenguaje textual de Matlab® que en lenguaje grafico de Simulink®.

El modelo que se utiliza en este trabajo es el desarrollado en el bloque *Embedded MATLAB® Function*. Para realizar el diseño y simulación del algoritmo en Simulink® se utilizaran los bloques mostrados en la figura 3.29 y a continuación se expondrán los más relevantes:

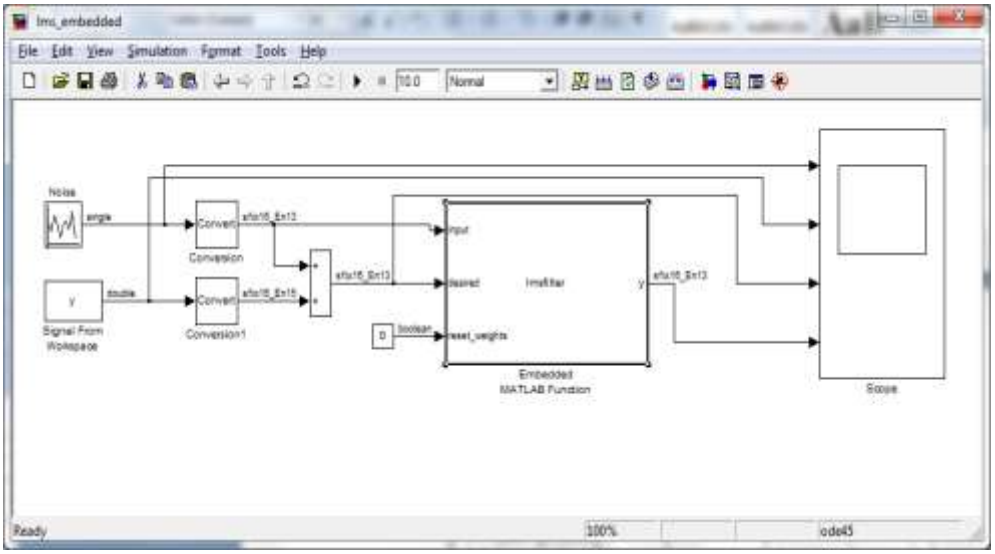


Figura 3. 299 ***Modelo en Simulink® del Algoritmo Adaptativo***

- ***Random Source***: Genera valores distribuidos aleatoriamente. El bloque genera una trama de M valores extraídos de una distribución uniforme o Gaussiana[21] pseudo-aleatoria, donde se especifica M en el parámetro de *Muestras por trama*.

[21] Distribución normal o gaussiana: Al iniciar el análisis estadístico de una serie de datos, y después de la etapa de detección y corrección de errores, un primer paso consiste en describir la distribución de las variables estudiadas. Una de las distribuciones teóricas más utilizadas en la práctica es la distribución normal, también llamada distribución gaussiana. Su importancia se debe fundamentalmente a la frecuencia con la que distintas variables asociadas a fenómenos naturales y cotidianos siguen, aproximadamente, esta distribución. La distribución de una variable normal está determinada por dos parámetros, su media y su desviación estándar, denotadas generalmente por μ y σ^2 . Con esta notación, la función de densidad de una característica x que sigue una distribución normal de media μ y varianza σ^2 viene dada por la siguiente ecuación que determina la curva en forma de campana que tiene esta distribución [27]:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left\{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}; \quad -\infty < x < \infty$$

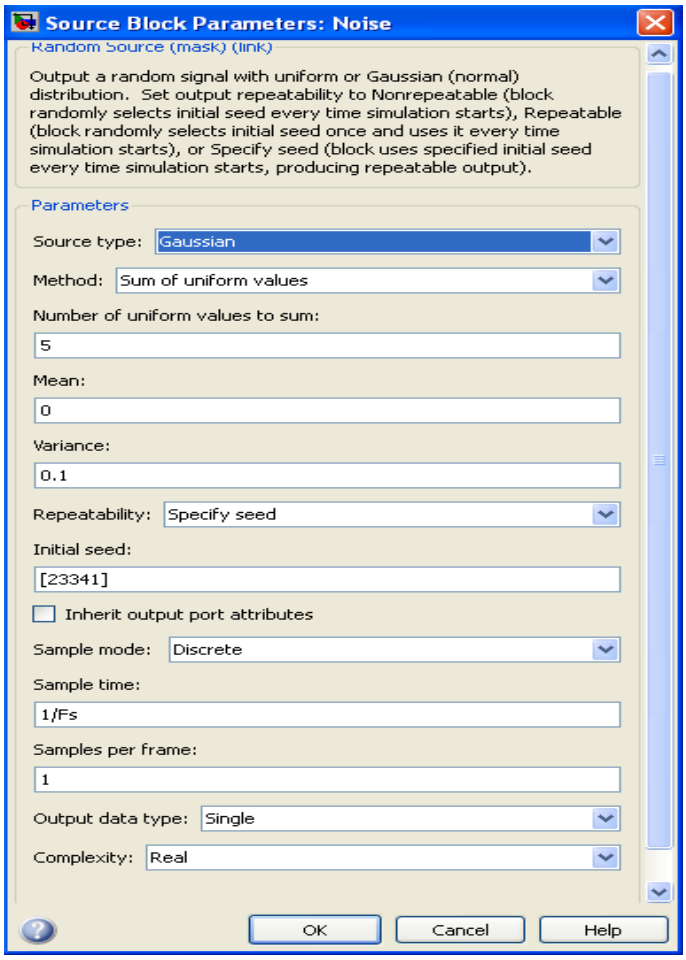


Figura 3. 30 ***Recuadro de Dialogo del Bloque Noise***

Source type: la distribución de la cual se extraerán los valores aleatorios, uniforme o gausiana.

Method: el método por el cual el bloque calcula los valores aleatorios gaussianos, Ziggurat o sum of uniform values.

Number of uniform values to sum: el número de valores aleatorios distribuidos uniformemente por sumar para calcular un solo número en una distribución aleatoria Gausiana.

Mean: la media de la distribución Gausiana normal.

Variance: la varianza de la distribución Gausiana normal.

Repeatability: la repetibilidad de la salida del bloque: Not repeatable, Repeatable o Specify seed. En las configuraciones Repeatable and Specify seed, el bloque saca la misma señal cada vez que se corre la simulación

Initial seed: son los números semilla iniciales que el generador de números aleatorios usa. El generador produce una secuencia idéntica de números aleatorios cada vez que es ejecutado con una semilla inicial particular.

Sample mode: se especifica el modo de muestreo continuo o discreto.

Sample time: es el periodo de muestreo Ts, de la secuencia de salida aleatoria

Sample per frame: es el número de muestras M, en cada trama de salida. Cuando el valor de este parámetro es 1, el bloque saca una señal basada en muestras.

Output data type: especifica el tipo de datos de salida de precisión única o doble precisión.

Complexity: especifica si la complejidad de la salida es real o compleja.

- ***Embedded MATLAB® Function Block****: E*ste bloque permite crear una función Matlab® dentro de un modelo Simulink®. Se puede especificar los datos de entrada y salida del bloque en la cabecera de la función como argumentos y valores de retorno. La función por defecto es `fcn` que tiene como argumento y valor de retorno `u` y `y` respectivamente, que se convierten en la entrada y salida del bloque por defecto. Este bloque soporta un subconjunto del lenguaje Matlab® para el cual se puede generar código embebido eficiente. La lista de las funciones soportadas se encuentra en el Help de Matlab®. Las funciones son editadas en el *Editor Embedded Matlab:*

Figura 3. 31 ***Ventana del Editor Embedded Matlab***

Para más información sobre el diseño del filtro adaptativo LMS, consultar la practica *No.4 SIMULACION DE FILTRO DIGITAL ADAPTATIVO LMS*

3.7.3 ANÁLISIS DE DESEMPEÑO

En este paso se realiza el análisis de resultados gráficos y auditivos. De no cumplirse los requisitos inícíales del diseño, se realizan cambios en el propio diseño o en los bloques que intervienen en el modelo.

3.7.4 IMPLEMENTACIÓN DEL FILTRO ADAPTATIVO

En este paso se implementan los códigos VHDL generados a partir del modelo Simulink®. Para este objetivo se utiliza una etapa de adecuación para la señal de audio de entrada a la FPGA que será explicada en el siguiente capítulo.

CAPITULO 4
INTRODUCCION A LA FPGA SPARTAN 3A DE XILINX®

En este capítulo se abordaran conceptos básicos relacionados con los dispositivos lógicos programables y las FPGA, haciendo un recorrido por la historia evolutiva de estos, para después hacer una descripción detallada de la FPGA Spartan 3A de Xilinx® y específicamente de los componentes usados en las prácticas de laboratorio. Para finalizar se hará un acercamiento al entorno de desarrollo *ISE Desing Suit.*

4.1 HISTORIA EVOLUTIVA DE LOS PLD[22]

Un dispositivo lógico programable es un circuito de propósito general o especifico, que posee una estructura interna que puede ser modificada por el usuario final para implementar una amplia gama de aplicaciones. El primer dispositivo que cumplió estas características fue una memoria PROM[23], pues se implementaron circuitos lógicos utilizando las líneas de direcciones como entradas de los circuitos lógicos y las líneas de datos como salidas.

Los primeros dispositivos diseñados específicamente para implementar funciones digitales programables fueron los PLA[24]. Los PLA cuentan fundamentalmente con un nivel de compuertas AND y otro de compuertas OR, ambos programables. Debido a los costos de fabricación y a la pobre velocidad de desempeño, surgieron dispositivos con un solo nivel de lógica programable denominados PAL[25], los PAL se basan en el mismo principio que las PLA, pero únicamente el nivel de lógica AND es programable, lo que significa que el nivel de lógica OR es fijo. Los dispositivos PAL fueron importantes porque cuando se introdujeron causaron un profundo efecto en el diseño de hardware digital. También son la base para algunas de las nuevas arquitecturas más sofisticadas [28]

Las variantes de la arquitectura básica PAL se ofrecen en varios productos conocidos por diferentes siglas. Todos los PLD pequeños, incluyendo PLA, PAL y dispositivos similares se agrupan en una sola categoría llamada SPLD[26], cuyas características más importantes son el bajo costo y la alta velocidad de rendimiento pin a pin.

Gracias al avance tecnológico se ha hecho posible la producción de dispositivos con mayor capacidad que los SPLD. La dificultad de aumentar la capacidad de una arquitectura SPLD radica en que la estructura de los planos lógico-programables crece demasiado rápido en tamaño a medida que el número de entradas se incrementa. La única forma viable para proporcionar dispositivos de gran capacidad basados en arquitecturas SPLD es integrar múltiples SPLD en un

[22] PLD: Programmable Logic Device – Dispositivo Lógico Programable
[23] PROM: Programmable Read-Only Memory – Memória Programable de Lectura solamente
[24] PLA:Programmable Logic Array –Arreglo Lógico Programable
[25] PAL: Programmable Array Logic –Programable Arreglo Lógico
[26] SPLD: Simple Programmable Logic Device – Dispositivo Lógico Programable Simple

solo chip y proporcionar opciones de interconexión entre ellos. Muchos FPD[27] comerciales, existentes en el mercado cuentan con esta estructura básica, y son llamados CPLD[28].
Un chip lógico de propósito general, de mayor capacidad es la MPGA[29] y consiste en una matriz de transistores prefabricados que se pueden personalizar mediante la conexión de los transistores con cables personalizados. La personalización se realiza durante la fabricación del chip mediante la especificación de la interconexión, esto hace que un usuario que haya recurrido a una MPGA tenga como consecuencias, grandes costos de montaje y largo tiempo de fabricación. La MPGA motivó el diseño del chip equivalente programable por el usuario: la FPGA[30] [29].

La FPGA es un dispositivo electrónico digital programable de muy alta escala de integración. Consiste en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad, lo que se programa en una FPGA son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques [30].

Los elementos básicos constituyentes de una FPGA como las de Xilinx® se pueden ver en la figura 4.1 y son los siguientes:

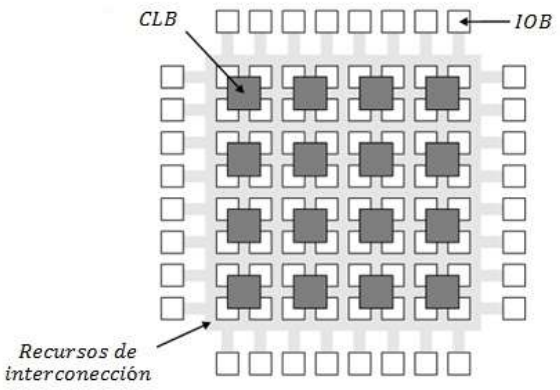


Figura 4. 1 ***Estructura general de una FPGA***

- CLB[31] : es la unidad básica en una FPGA, consisten en una matriz de conmutación configurable, multiplexores y flip-flops. La matriz de conmutación es altamente flexible a diversas configuraciones.

[27] FPD: Field Programmable Device – Dispositivo Programable por Campo: termino general para referirse a cualquier tipo de circuito integrado usado para implementar hardware digital, donde el chip puede se configurado por el usuario para realizar diferentes diseños
[28] CPLD: Complex Programmable Logic Device – Dispositivo Lógico Programable Complejo
[29] MPGA: Mask-Programmable Gate Array – Máscara de Arreglo de compuertas programables
[30] FPGA: *Field Programmable Gate Array – Matriz de Compuertas Programables por campo*
[31] CLB: Configurable logic block – Bloque Lógico Configurable

- IOB[32]: son grupos de bancos de entradas y salidas, en donde cada banco es capaz de soportar diferentes estándares de entradas y salidas.
- Recursos de interconexión cuya estructura y contenido se denomina arquitectura de rutado. En general, los recursos de interconexión son de tres tipos:
    - Conexiones directas: permiten la conexión de las salidas del CLB con sus vecinos más directos.
    - Interconexiones de propósito general: para distancias superiores a un CLB (mas allá del vecino). Son pistas horizontales y verticales del tamaño de un CLB, pero que se pueden empalmar para crear pistas más largas.
    - Líneas de largo recorrido: suelen cubrir lo ancho o largo de la pastilla. Permiten conexiones con un retardo mucho menor que uniendo las anteriores. El camino crítico de un circuito es el recorrido que, desde una entrada hasta una salida, presenta un retardo máximo [30].

4.2 ARQUITECTURA DE FPGA SPARTAN 3A DE XILINX®

La compañía Xilinx® fabrica una amplia gama de familias de FPGA, entre las cuales se destacan la Virtex y la Spartan.

En esta ocasión se utilizará una Tarjeta FPGA de la familia Spartan 3A.

La arquitectura de la FPGA Spartan 3A consta de 5 elementos funcionales programables:

- CLB: los bloques lógicos configurables desarrollan una amplia variedad de funciones lógicas como de almacenamiento de datos.
- IOB: los bloques de entrada/salida controlan el flujo de datos entre los pines de entrada y salida y la lógica interna del dispositivo. Estos soportan flujo de datos bidireccional.
- Bloque RAM: provee almacenamiento de datos en los bloques de puerto dual de 18Kbits
- Bloques Multiplicadores: aceptan dos números binarios de 18 bits como entrada y calculan el producto.
- Bloque manejador del reloj digital (DCM): provee su propia calibración y soluciones para la distribución, retardo, multiplicación, división y desplazamiento en fase de las señales de reloj.

[32] IOB: In/out blocks – Bloques de Entrada/Salida

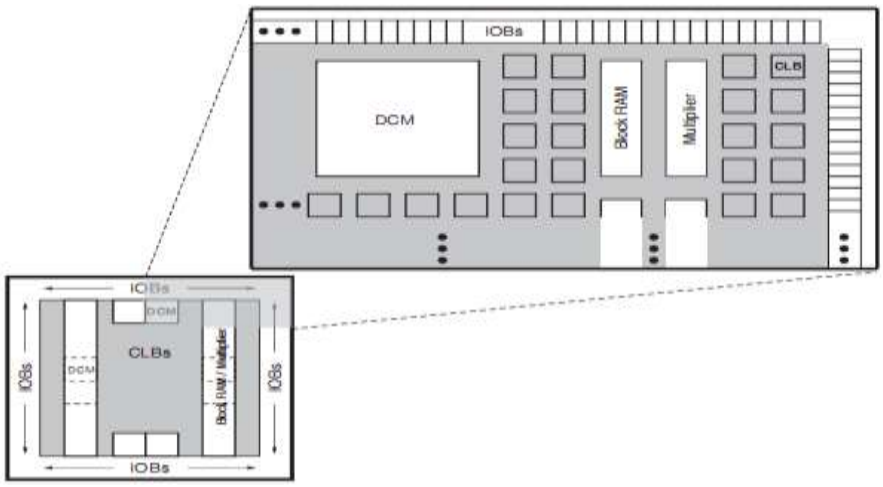


Figura 4. 2 ***Arquitectura de FPGA Spartan 3A***

Un anillo dual de IOBs rodea la matriz de CLBs. El dispositivo tiene dos columnas de RAM, cada una de ellas consiste de varios bloques RAM de 18Kbits. Cada bloque de RAM es asociado con un multiplicador dedicado. Los DCMs están en el centro, dos en la parte de arriba, dos abajo del dispositivo y dos en el medio de cada una de las columnas de bloques de RAM y multiplicadores.

La Spartan 3A dispone de una amplia red de rutas que interconectan los 5 elementos funcionales, transmitiendo señales entre ellos. Cada elemento funcional tiene una matriz de conmutación asociada que permite conexiones múltiples para el enrutamiento [31].

La tabla 4.1 resume los atributos principales de la FPGA Spartan 3A:

Dispositivo		*XC3S700A*
Compuertas de sistema		700K
	Equivalente en celdas lógicas	13.248
Arreglos de CLB	Filas	48
	Columnas	32
	CLB	1.472
	Sílices	5.888
RAM distribuida(bits)		92K
Bloques de RAM(bits)		360K
Multiplicadores dedicados		20
DCM		8
Máximo de E/S		372
Máximo de pares de E/S diferenciales		165

Tabla 4. 1 ***Atributos de Spartan 3A [32]***

4.3 COMPONENTES DE LA TARJETA SPARTAN 3A DE XILINX®

Los componentes de la Tarjeta Spartan-3A (XC3S700A-FG484) se observan en la grafica 4.3 [32]:

- Interruptores y pulsadores:
  1. Cuatro interruptores deslizables.
  2. Un interruptor de suspensión (si el modo suspensión esta activo en la aplicación, entonces la FPGA entra en el modo de suspensión siempre q el switch sea puesto en *suspend*, si el switch esta luego puesto en *run* la FPGA retoma la operación que estaba realizando antes de entrar en el modo *suspend*).
  3. Un Interruptor de encendido.
  4. Cuatro pulsadores.
  5. Un botón pulsador rotatorio.
  6. Un pulsador de reprogramación (fuerza a la FPGA a reprogramarse usando los datos almacenados en una memoria que es seleccionada por medio de la configuración de los jumpers).
- Leds:
  7. Ocho leds indicadores de salida.
  8. Un led de indicación de programación exitosa.
  9. Dos leds opcionales de entrada y/o salida. (dependiendo de las características de la aplicación, estos leds pueden ser usados como de entrada y/o salida).
  10. Un led de indicación de encendido.
  11. Un led de indicación de conexión de interfaz Jtag USB[33]
- Relojes:
  12. Un oscilador integrado de 50MHz
  13. Un oscilador auxiliar de 133MHz, que es sustituible opcionalmente por un oscilador de 8 pines.
  14. Conector SMA[34] para proveer una fuente de reloj externa.
- Salidas /entradas:
  15. Una pantalla LCD de 2 líneas de 16 caracteres.
  16. Un conector hembra de 15 pines VGA estándar.
  17. Dos puestos seriales RS-232 (uno hembra y uno macho).
  18. Un puerto para ratón o teclado PS/2 con un conector estándar DIN[35] de seis pines.
  19. Una interface RJ45 Ethernet 10/100.
  20. Un circuito de captura analógico de dos canales, consta de un preamplificador programable a escala y un conversor análogo digital ADC.

[33] Jtag: Joint Test Action Group- Test aplicado a un circuito impreso utilizando escaneo de limites

[34] SMA: SubMiniature versión A - conector roscado para cable coaxial

[35] DIN connector: conectores eléctricos de alta frecuencia, multi-pin, que fueron estandarizados por el *Deutsches Institut für Normung*

21. Un conversor digital analógico DAC de 4 canales de 12 bits.
22. Dos interface Jtag, una USB y otra paralela.
23. Un conector estéreo de audio digital.

- Memorias:
  24. Memoria flash PROM de acceso paralelo (NOR) de 32Mbit
  25. Dos interfaces flash SPI[36], una memoria flash PROM de 16Mbit y una memoria data-flash de 16Mbit.
  26. Memoria DDR2 SDRAM de 512Mbit (32M*16)
  27. Memoria flash PROM de 4Mbit
- Expansión
  28. Un conector de borde 100 pines
  29. Dos conectores diferenciales de 43 pines con capacidad de transmisión-recepción 600Mbps por cada par.
  30. Dos conectores de seis pines que pueden ser usados como entradas o salidas
- Otros
  31. Conector de adaptador de corriente.

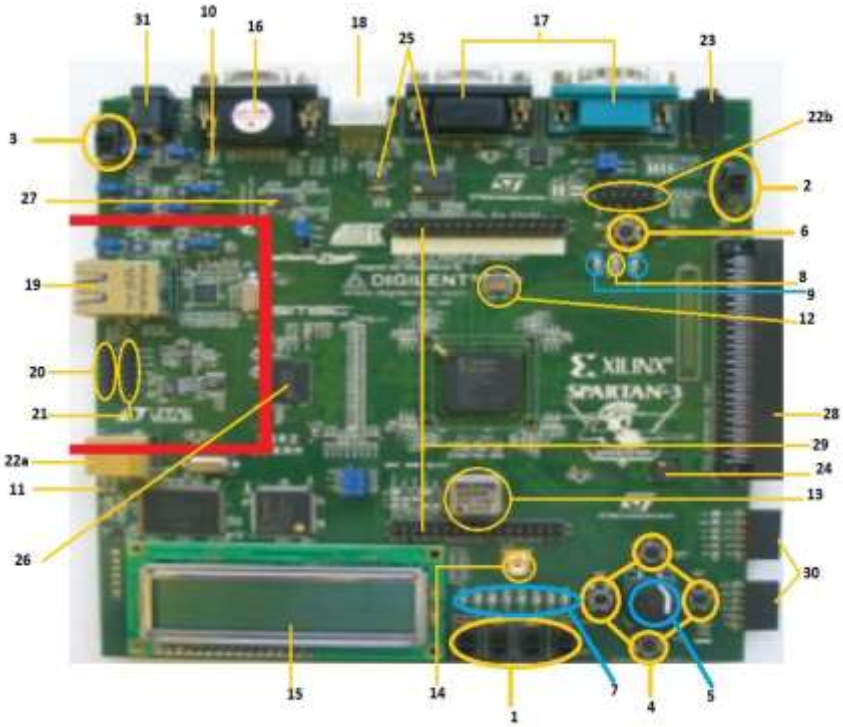


Figura 4. 3 ***Componentes de la FPGA***

[36] SPI: Serial Peripheral Interface – Es un bus estándar de comunicaciones.

4.4 COMPONENTES UTILIZADOS DE LA FPGA Y ETAPA DE ADAPTACION

- El circuito de captura analógica de dos canales, está compuesto por un pre-amplificador programable a escala LTC6912-1 que provee dos amplificadores independientes con ganancia programable, la salida del preamplificador es conectada a un conversor analógico digital LTC1407A-1, la salida del conversor es una representación de 14 bits con complemento a dos de la señal de entrada. Por consiguiente se pueden representar valores entre -2^{13} y $2^{13}-1$, la máxima taza de muestreo es aproximadamente 1.5MHz. Los componentes y las entradas del circuito de captura analógica se muestran en la siguiente figura:

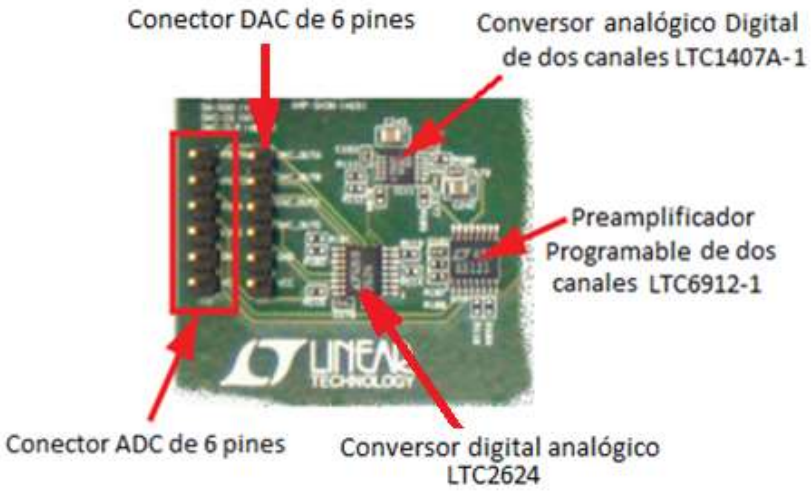


Figura 4. 4 ***Componentes y entradas de Circuito ADC [32]***

- El Circuito de conversión digital analógico es un LTC2624, de cuatro canales seriales, con resolución de 12-bit sin signo cada uno. El circuito DAC y sus salidas se muestran en la figura 4.4.

Es necesaria una etapa de adaptación para evitar que los valores de entrada al conversor ADC no excedan los límites establecidos de voltaje. Estos límites de voltaje de entrada dependen de la ganancia del amplificador.

La ganancia de este amplificador es programable de -1 a -100. Para el caso de las prácticas la ganancia escogida será -1, por lo tanto los valores de voltaje deben estar entre 0,4v y 2,9v. Según las especificaciones de *Spartan-3A FPGA Starter Kit Board User Guide* se debe adicionar un nivel DC de 1,65 a la señal alterna de entrada, de tal manera que los límites de voltaje para esta ganancia específica sean 1.65±1.25 [32].

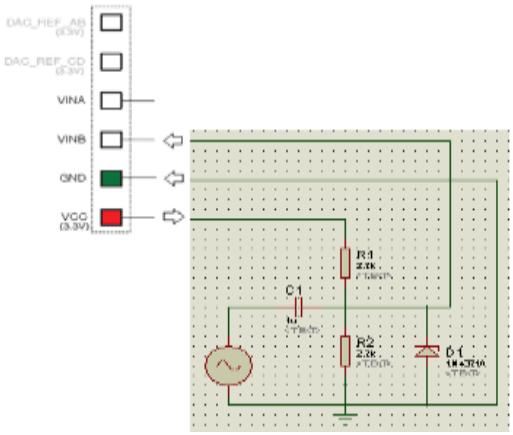


Figura 4. 5 ***Diagrama circuital de etapa de adaptación para filtro FIR e IIR***

El circuito de la grafica 4.5 corresponde a un divisor del voltaje de referencia de 3.3v, que sale del pin VCC del conversor ADC de la FPGA, para obtener 1.65v y sumarlos a la señal alterna de entrada. Luego un diodo Zener recorta los niveles de voltaje de esta señal que estén por encima de los permitidos y la envía al canal 2 del conversor analógico-digital (VINB).

Sin embargo, en señales de audio, el hecho de recortar los picos de la señal implica perdida de datos perceptibles al oído, por lo tanto al escuchar modificaciones en la señal de salida de los conversores, se debe disminuir el volumen con que sale del equipo de audio, para disminuir las amplitudes de voltaje de entrada al ADC y que de esta manera no sean recortadas por el Zener.

Para el filtro Adaptativo LMS que utiliza dos señales de entrada a la FPGA, se diseñó la etapa de adaptación mostrada en la figura 4.6, la cual realiza el mismo proceso de adaptación de los voltajes de la señal explicado anteriormente, para cada una de las dos señales de entrada:

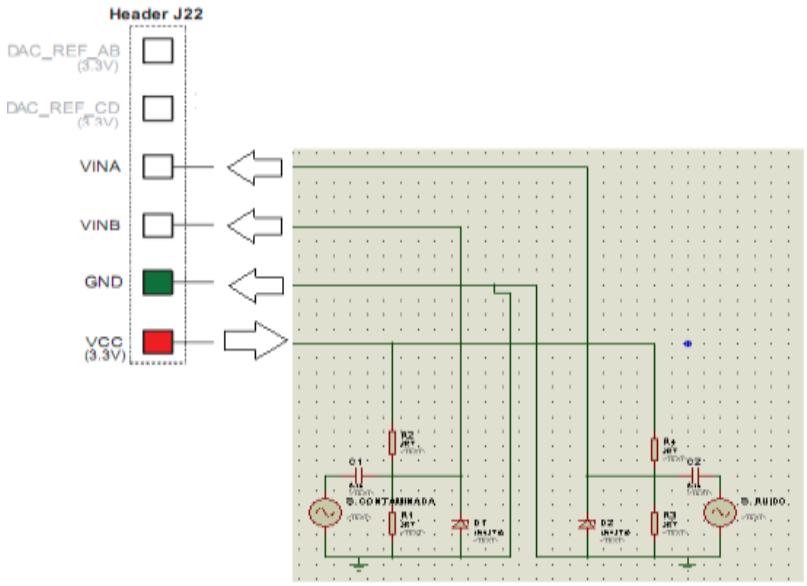


Figura 4. 6 ***Diagrama circuital de etapa de adaptación para filtro Adaptativo LMS***

En la figura 4.6, la señal de ruido entra por el canal 1 (VINA) y la señal contaminada entra por el canal 2(VINB) del conversor ADC de la FPGA.

La siguiente figura 4.7, muestra una foto tomada a la FPGA integrada con la etapa de adaptación, donde se observa el cable de entrada de audio y el cable de salida de la FPGA a los parlantes:

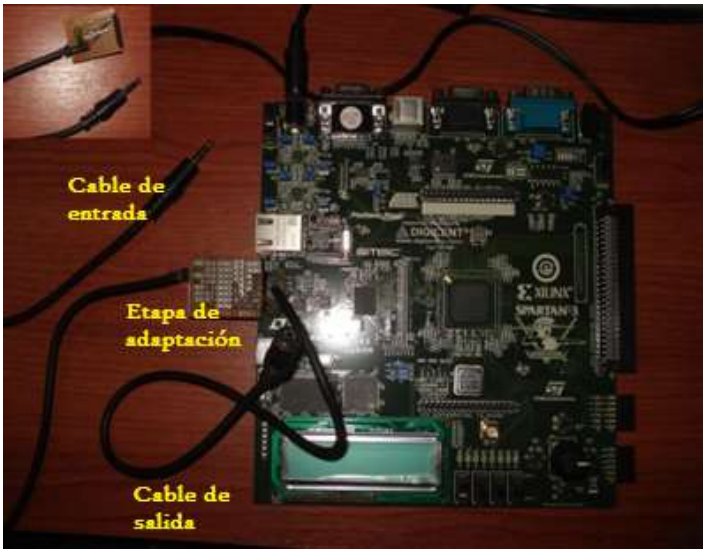


Figura 4. 7 ***Integración de FPGA con etapa de adaptación y los parlantes***

Para información sobre la conexión de la Etapa de Adaptación con la FPGA consulte *Práctica de implementación de filtros digitales FIR por método de Ventanas, FIR Rizado Constante, IIR Butterworth y Adaptativo sobre FPGA Spartan 3A.*

4.5 ENTORNO DE DESARROLLO

La ayuda de un entorno con herramientas que asistan en el proceso de diseño, simulación, síntesis del resultado y configuración del hardware, favorece el propósito y la solución de problemas complejos en dispositivos reconfigurables como la FPGA. Uno de estos entornos es el software de la compañía Xilinx® denominado ISE™ (Integrated Software Environment). Esta herramienta se puede clasificar como un entorno EDA[37]

El *Project Navigator* es la interfaz grafica de usuario de la herramienta ISE, que permite el acceso a todos los componentes del proyecto. Los diseños de usuario se pueden introducir mediante diferentes formatos: los esquemáticos, los grafos de estados y las descripciones hardware en VHDL o Verilog. Una vez compilados los diseños se puede simular su comportamiento a nivel funcional o a nivel temporal. A nivel funcional no tiene en cuenta los retardos provocados por el hardware y a nivel temporal se simula el diseño teniendo en cuenta cómo se va a configurar el hardware.

El flujo de diseño que se debe seguir en el ISE, es descrito por las siguientes secciones [33]:

- Creación del diseño

[37] EDA: Electronic Desing Automation - Automatización del Diseño electrónico.

Durante la creación del diseño, se debe crear un proyecto ISE y luego crear o agregar archivos a este. El proyecto ISE puede contener muchos tipos de archivos de código fuente y módulos de diseño.

- Síntesis
  Durante la síntesis, el motor de síntesis XST compila el diseño para transformar las fuentes HDL en una arquitectura específica de diseño lista de red o *Netlist*.
- Simulación
  En varios momentos durante el flujo del diseño se puede verificar la funcionalidad del diseño utilizando la herramienta de simulación. Esta herramienta que viene integrada con el ISE es conocida como *ISim* o simuladores *ModelSim*.
- Restricciones de entrada
  Usando las restricciones de diseño se puede especificar el tiempo, el momento y algunos otros requerimientos de diseño. El software ISE proporciona editores para facilitar la especificación de restricciones para limitaciones de tiempo, así como pines de entrada y salida y las limitaciones de diseño.
- Implementación
  Después de la síntesis se corre la aplicación de diseño, la cual convierte el diseño lógico en un archivo de formato físico que pueda ser descargado en un dispositivo seleccionado. Usando el Proyect Navigator se puede modificar las propiedades de los procesos para controlar la implementación y así optimizar el diseño. Para tratar de cumplir los objetivos de diseño más rápido, se puede utilizar *SmartXplorer* para automatizar múltiples implementaciones corriendo con diferentes propiedades en los procesos.
- Análisis de implementación
  Después de la implementación se puede analizar el rendimiento del diseño frente a las limitaciones, la utilización de los recursos del dispositivo, rendimiento temporal y utilización de energía. Se pueden ver los resultados en archivos de informes, mirando la implementación actual del dispositivo y con herramientas con interfaces gráficas como el *PlanAhead*. Además es posible analizar de forma interactiva los resultados de tiempo y energía usando el *Analizador de tiempo* y las herramientas de análisis *Xpower*. Finalmente se puede realizar una depuración del sistema haciendo uso de la herramienta *ChipScope Pro*.
- Mejora de implementación
  Con base en el análisis de los resultados de diseño se pueden realizar cambios en las fuentes del diseño, las propiedades de los procesos o la especificación de restricciones para después volver a ejecutar la síntesis y la implementación.
- Configuración de dispositivos y Programación
  Después de generar un archivo de programa se configura el dispositivo. Durante la configuración, se generan dos archivos que deben ser descargados desde un computador a un dispositivo de Xilinx®.

La versión del ISE que se usará será la 12.1, para consultar detalles sobre los requerimientos del sistema y la instalación de la herramienta consultar el Anexo C

CAPITULO 5
PRUEBAS Y RESULTADOS

En este capítulo se muestran los resultados del diseño y la simulación en Matlab® y Simulink® de los filtros digitales propuestos en los Ejemplos Prácticos de cada una de las 4 prácticas. Además se muestran los resultados de la implementación de estos diseños en la FPGA.

5.1 RESULTADOS DE LA PRÁCTICA 1: SIMULACION DE FILTROS DIGITALES TIPO FIR POR EL METODO DE VENTANA KAISER Y OTRO METODO DE VENTANA

5.1.1 Para filtro FIR Hamming Elimina Banda, con especificaciones:

```
fp1=600Hz    %frecuencia de paso 1
fs1=1200Hz   %frecuencia de supresión 1
fs2=2700Hz   %frecuencia de supresión 2
fp2=3200Hz   %frecuencia de paso 2
fm=8000Hz    %frecuencia de muestreo
As=50dB      %Atenuación de supresión
```

- **RESULTADOS DEL DISEÑO**

```
Frecuencia de corte1 Fc1=(fp1+fs1)/(2fm)= 0.11250 ciclos/muestra
Frecuencia de corte2 Fc2=(fp2+fs2)/(2fm)= 0.36875 ciclos/muestra
```

Como es un filtro elimina banda se tienen dos anchos de banda de transición:

```
Ancho de banda1 BW1=abs(2*pi*(Fp1-Fs1))= 0.4712
Ancho de banda2 BW2=abs(2*pi*(Fp2-Fs2))= 0.3926
```

por lo cual dependiendo del diseñador, se selecciona uno de ellos, en este caso se consideró el de menor ancho de banda buscando mayor selectividad :

```
Ancho de banda de transición BW= 0.3926
```

Teniendo en cuenta la atenuación de diseño `As=50dB`, de acuerdo con la tabla 3.2. La ventana que puede ser usada es la ventana de Hamming, ya que su máxima atenuación permitida es de 53dB.

```
Orden del filtro N=ceil(((8*pi)/BW)-1)= 63
```

Recordando que el filtro diseñado es tipo I, el orden de este debe ser par, por lo tanto se suma 1 al resultado anterior, con lo cual se obtiene `N=64`

Al graficar la función de la ventana Hamming `w =.54-.46*cos(2*pi*n/(N))`, el resultado es el mostrado en la figura 5.1:

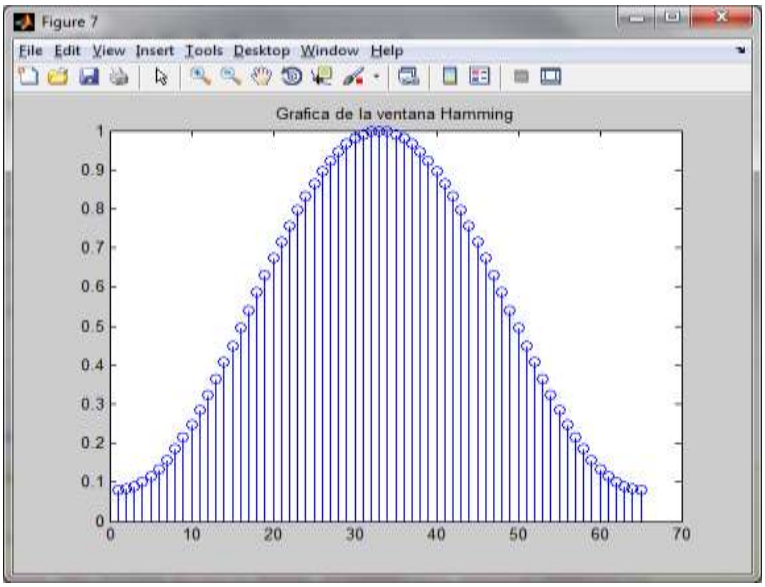


Figura 5. 1 ***Grafica de la ventana Hamming con N=64***

Luego de obtener la ventana `w`, esta se aplica a la función de transferencia del filtro Elimina Banda `he`, de la siguiente forma: `h=he.*w`, donde `h` es la función de transferencia del filtro Elimina Banda diseñado con ventana Hamming.
Para información sobre las funciones de transferencia de los filtros selectivos en frecuencia, consulte el anexo D.

En la grafica 5.2 se muestra la respuesta en frecuencia del filtro diseñado con ventana Hamming, donde se observa que se cumple con los requisitos de diseño, es decir, que la atenuación deseada en la banda de supresión y las frecuencias de diseño son logradas.

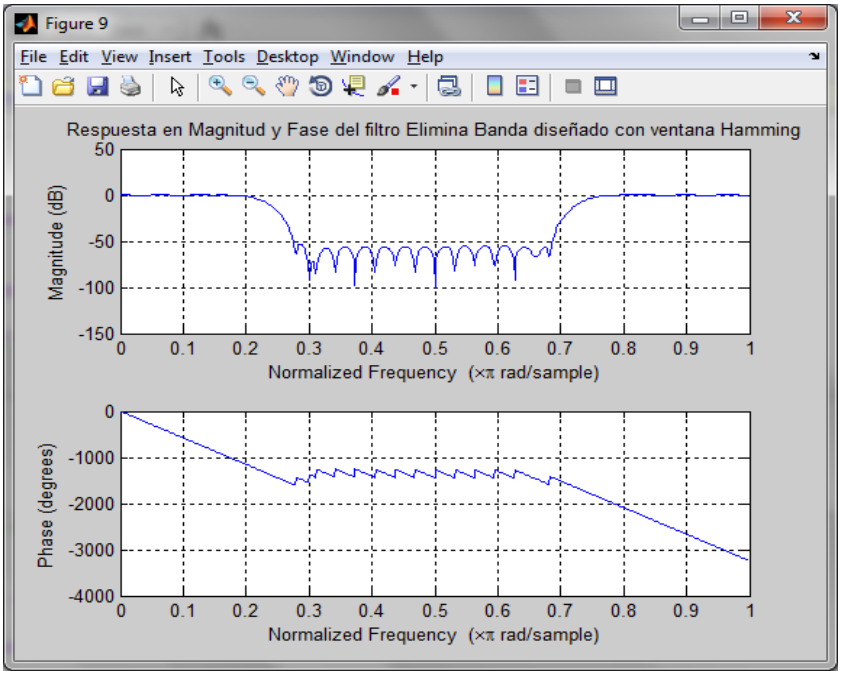


Figura 5. 2 ***Grafica de la respuesta en frecuencia del filtro Elimina Banda usando ventana Hamming***

- **RESULTADOS DE LA SIMULACION EN MATLAB®**

  La señal que va a ser filtrada, es la señal *handel* predefinida por Matlab®, la cual carga los datos en la variable `y`, de tamaño 73113x1 con amplitudes que van desde -0.8 hasta 0.8 (ver figura 5.3.a) y carga su frecuencia de muestreo en la variable `Fs=8192(Hz)`. Esta señal `y` es aplicada al filtro diseñado con ventana Hamming dando como resultado la señal almacenada en la variable `out` (ver figura 5.3.b.)

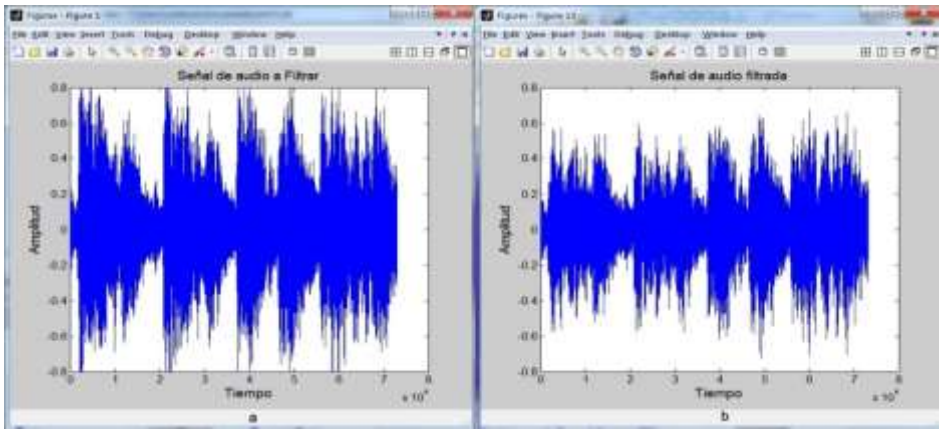


Figura 5. 3 ***Grafica Amplitud vs. Muestras de la señal a la entrada y a la salida del filtro Elimina Banda diseñado con ventana Hamming***

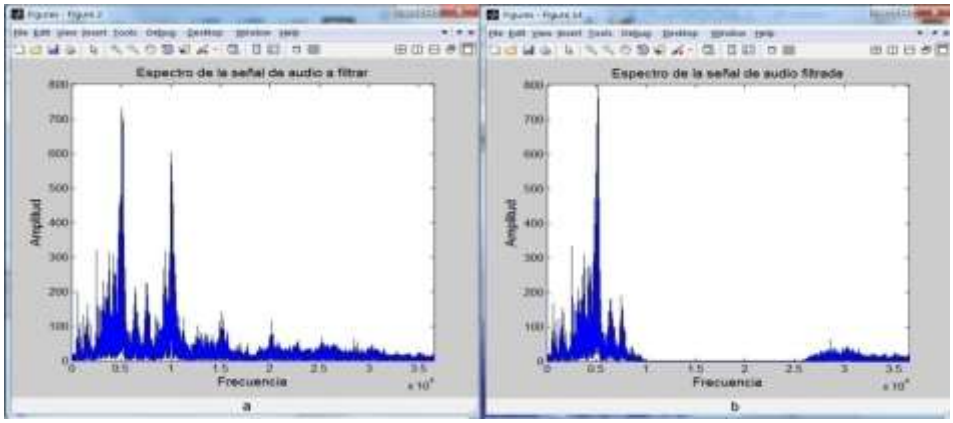


Figura 5. 4 ***Grafica del espectro de la señal de entrada y del espectro de la señal de salida del filtro Elimina Banda diseñado con ventana Hamming respectivamente***

El análisis espectral de las señales de entrada (figura 5.4.a) y de salida del filtro (figura 5.4.b) permite observar como son atenuadas las frecuencias que corresponden a las especificaciones indicadas.

- **RESULTADOS DE LA SIMULACION EN SIMULINK®**

  Siguiendo las indicaciones de la Práctica para la creación del modelo en Simulink® y utilizando la misma señal *handel* para ser filtrada, los resultados auditivos de la simulación en el entorno Simulink® fueron exitosos al escuchar cambios en la señal de audio de salida en comparación con la señal de entrada original. (El modelo Simulink y las señales que intervienen se encuentran en el CD)

- **RESULTADOS DE LA IMPLEMENTACION EN FPGA**

  Para realizar este paso, se siguieron las indicaciones de la Práctica No.5 *IMPLEMENTACION DE FILTROS DIGITALES FIR POR MÉTODO DE VENTANA Y FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A,* que muestra cómo pasar el modelo Simulink® de punto flotante a punto fijo y como generar a partir de este modelo el código VHDL. Además el *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®* también es necesario ya que muestra como implementar el código generado junto con el código de los conversores ADC y DAC.

  El plan de pruebas para la implementación del filtro FIR Hamming, consiste en montar el sistema como se indica en la figura 4.7 (Integración de FPGA con etapa de adaptación y los parlantes), escuchar la señal de audio de entrada (señal *handel*) y la señal de audio salida de la FPGA, gracias a la utilización de un interruptor de la FPGA (ver Practica No.5), además, capturar y observar estas señales por medio de Matlab®. La captura se hizo posible gracias a la función de Matlab® `wavrecord(N, Fs)`, donde `N` es el numero de muestras grabadas que para el caso se consideró guardar 100000 muestras y `Fs` es la frecuencia de muestreo utilizada para la grabación, en este caso se utilizó Fs= 11025(Hz). Las señales de entrada y de salida de la FPGA capturadas se presentan en las figuras 5.5.a y 5.5.b respectivamente:

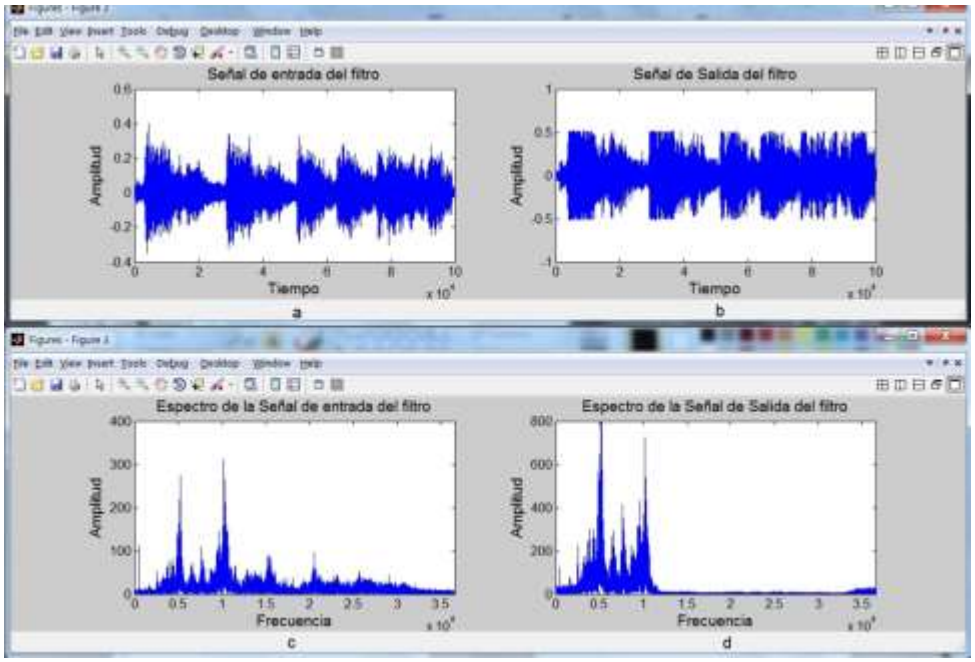


Figura 5. 5 ***Señal a la entrada y a la salida de la FPGA para el Filtro FIR Elimina Banda diseñado con ventana Hamming***

Al igual que la simulación en Matlab®, se observa mediante el análisis espectral que se cumple con las especificaciones de diseño: figura 5.5.c. Espectro señal de entrada y figura 5.5.d Espectro de señal de salida con atenuación del filtro elimina banda.

5.1.2 Para un filtro FIR Káiser Pasa Banda, con especificaciones:

```
fs1=300Hz;    %frecuencia de supresión 1
fs2=2100Hz;   %frecuencia de supresión 2
fp1=800Hz;    %frecuencia de paso 1
fp2=1500Hz;   %frecuencia de paso 2
fm = 8000Hz; %frecuencia de muestreo
As1=50dB;     %atenuación en la banda de supresión 1
Ap=1dB;       %atenuación en la banda de paso
As2=50dB;     %atenuación en la banda de supresión 2
```

- **RESULTADOS DEL DISEÑO**

```
Frecuencia de corte1 Fc1=(fp1+fs1)/(2fm)= 0.06875 ciclos/muestra
Frecuencia de corte2 Fc2=(fp2+fs2)/(2fm)= 0.22500 ciclos/muestra
```

Como es un filtro pasa banda se tienen dos anchos de banda de transición:

```
Ancho de banda1 BW1=abs(2*pi*(Fp1-Fs1))= 0.3927
Ancho de banda2 BW2=abs(2*pi*(Fp2-Fs2))= 0.4712
```

por lo cual dependiendo del diseñador, se selecciona uno de ellos, en este caso se consideró el de menor ancho de banda buscando mayor selectividad:

```
Ancho de banda de transición BW= 0.3927

Orden del filtro N=ceil((As1-7.95)/(2.285*BW))=47
```

Recordando que el filtro diseñado es tipo I, el orden de este debe ser par, por lo tanto se suma 1 al resultado anterior, con lo cual se obtiene `N=48`

Dependiendo de la atenuación de supresión dada en dB, se calcula el parámetro Beta. Para la atenuación `As1=50`:

```
Parámetro beta B= 0.5842*((As1-21)^0.4)+0.07886*(As1-21)= 4.53351
```

Los valores hallados para el parámetro Beta y el orden del filtro, son necesarios para hallar la función de la ventana Káiser:

`w= kaiser(M,B)'`, donde `M` es la longitud del filtro, es decir `N+1`

La siguiente figura 5.6 muestra la grafica de la función de ventana Káiser `w`:

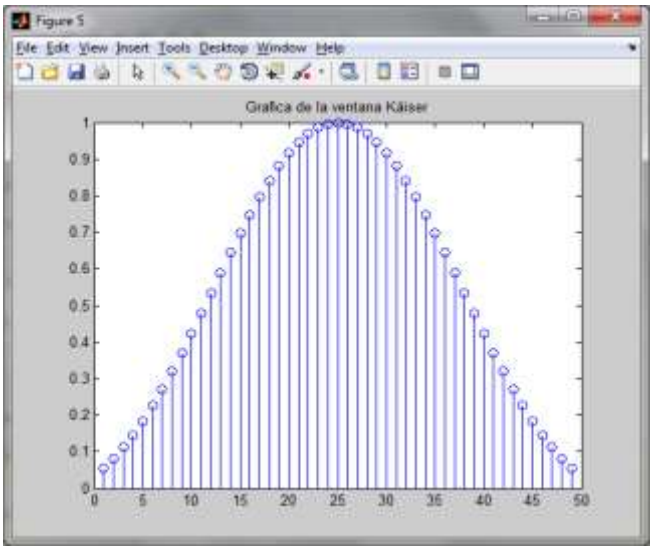


Figura 5. 6 ***Grafica de la ventana Kaiser con N=48***

Luego de obtener la ventana `w`, esta se aplica a la función de transferencia del filtro Pasa Banda `hp`, de la siguiente forma: `h=hp.*w`, donde `h` es la función de transferencia del filtro Pasa Banda diseñado con ventana Káiser.
Para información sobre las funciones de trasferencia de los filtros selectivos en frecuencia, consulte el anexo D

La figura 5.7 presentada a continuación, muestra la respuesta en frecuencia del filtro diseñado con ventana Káiser, donde se puede notar que se cumple con los requisitos de diseño, es decir, que las atenuaciones deseadas en las bandas de supresión y las frecuencias de borde son logradas.

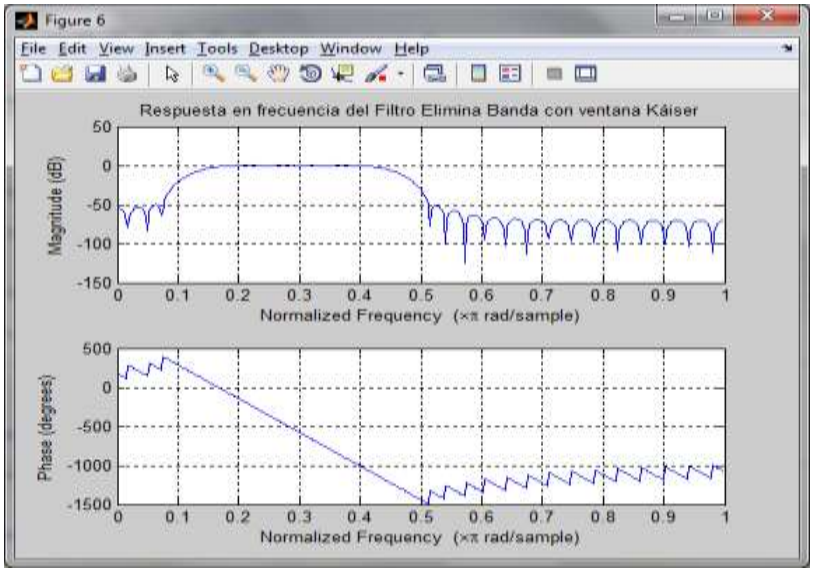


Figura 5. 7 ***Grafica de la respuesta en frecuencia del filtro Pasa Banda usando ventana Káiser***

- **RESULTADOS DE LA SIMULACION EN MATLAB®**

  La señal que va a ser filtrada, es la señal *handel* predefinida por Matlab®, la cual carga los datos en la variable `y`, de tamaño 73113x1 con amplitudes que van desde -0.8 hasta 0.8 (ver figura 5.8.a) y carga su frecuencia de muestreo en la variable `Fs=8192(Hz)`. Esta señal `y` es aplicada al filtro diseñado con ventana Káiser dando como resultado la señal almacenada en la variable `out` (ver figura 5.8.b.)

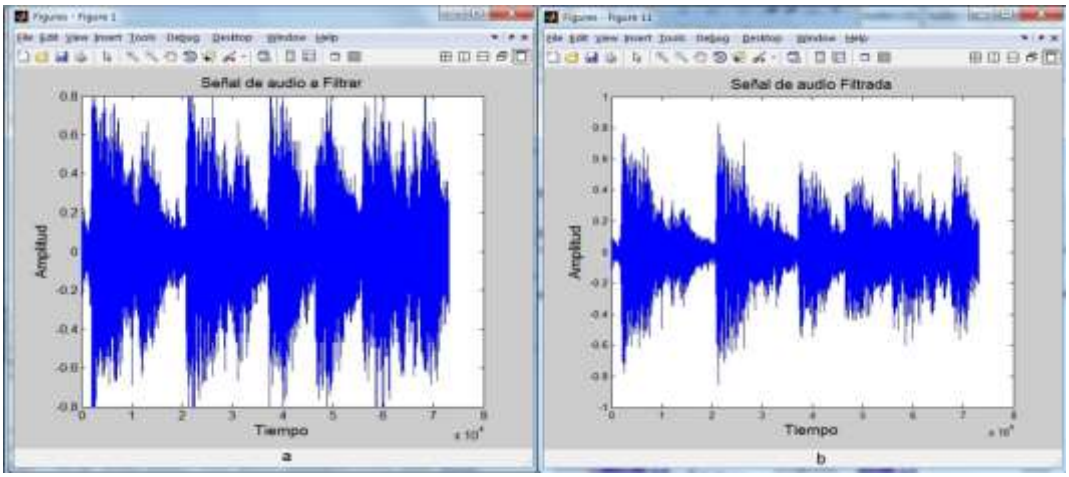


Figura 5. 8 ***Grafica Amplitud vs. Muestras de la señal a la entrada y a la salida del filtro Pasa Banda diseñado con ventana Káiser***

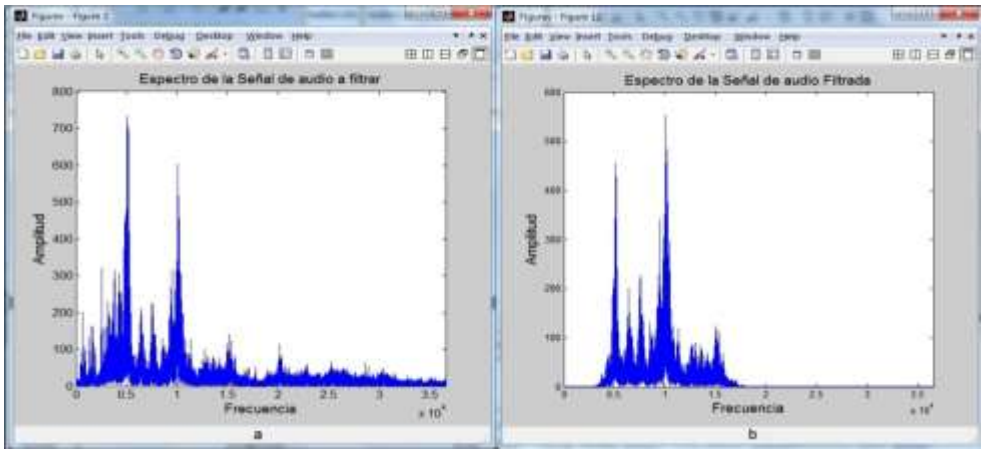


Figura 5. 9 ***Grafica del espectro de la señal de entrada y del espectro de la señal de salida del Filtro Pasa Banda diseñado con ventana Káiser respectivamente.***

La figura 5.8 muestra que la señal de audio de salida ha cambiado respecto a la señal original. La figura 5.9 muestra el espectro de la señal de entrada así como el espectro de la señal de salida donde se puede observar que solo pasan las frecuencias comprendidas en la banda de paso, las demás se atenúan.

- **RESULTADOS DE LA SIMULACION EN SIMULINK®**

  Siguiendo las indicaciones de la Práctica para la creación del modelo en Simulink® y utilizando la misma señal *handel* para ser filtrada, los resultados auditivos de la simulación en el entorno Simulink® fueron exitosos al escuchar cambios en la señal de audio de salida en comparación con la señal de entrada original, debido a las frecuencias eliminadas. (El modelo Simulink y las señales que intervienen se encuentran en el CD)

- **RESULTADOS DE LA IMPLEMENTACION EN FPGA**

Para realizar este paso, se siguieron las indicaciones de la Práctica No.5 *IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANA, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A y* del *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®*.

Aplicando el mismo plan de pruebas del filtro FIR diseñado con ventana Hamming, se capturan las señales de entrada *handel* (figura 5.10.a) y de salida de la FPGA (figura 5.10.b) como se indico anteriormente. Al graficarlas se obtuvieron las siguientes figuras para el filtro Pasa Banda Káiser:

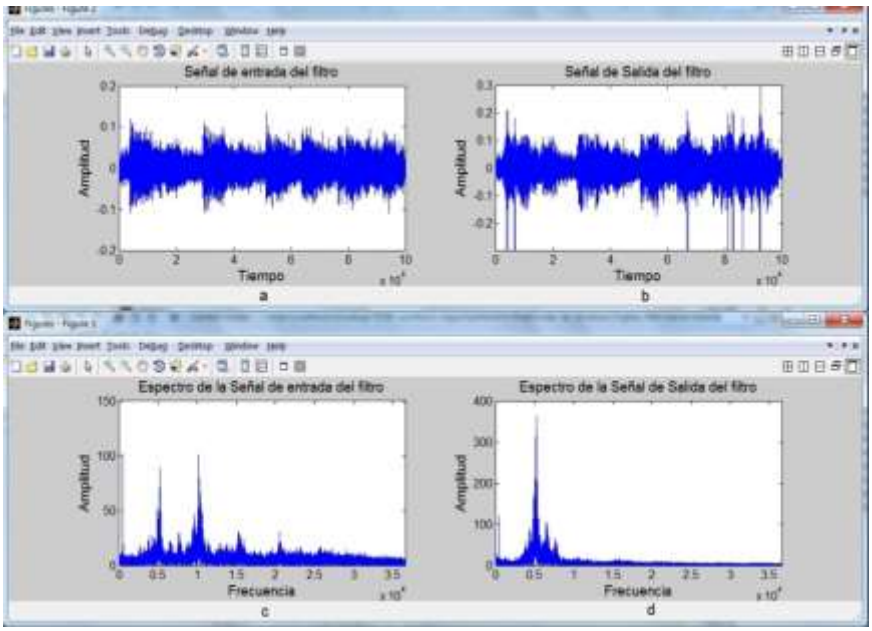


Figura 5. 10 ***Señal a la entrada y a la salida de la FPGA para el Filtro FIR Pasa Banda diseñado con ventana Káiser***

Al igual que la simulación en Matlab®, se observa mediante el análisis espectral que se cumple con las especificaciones de diseño: figura 5.10.c. Espectro señal de entrada y figura 5.10.d Espectro de señal de salida con atenuación del filtro pasa banda.

5.2 RESULTADOS DE LA PRÁCTICA 2: SIMULACION DE FILTROS DIGITALES TIPO FIR-METODO DE RIZADO CONSTANTE

5.2.1 Para un filtro FIR con Rizado constante Pasa Alto, con especificaciones:

```
Ap=2dB;      %Atenuación de banda de paso
As=50dB;     %Atenuación de banda de supresión
fm=8000Hz;   %frecuencia de muestreo
fp=2350Hz;   %frecuencia de paso
fs=1800Hz;   %frecuencia de supresión
```

- **RESULTADOS DEL DISEÑO**

```
Atenuación de banda de supresión en unidades naturales
ds=(1+dp)*(10^(-As/20))= 0.003524
Atenuación de banda de paso en unidades naturales
```

```
dp=(10^(Ap/20)-1)/(10^(Ap/20)+1)= 0.114623

Ancho de banda de transición BWf=abs((fp-fs)/fm)= 0.06875
Orden del filtro N=ceil((-20*log10(sqrt(ds*dp))-13)/(14.6*BWf))
=21
```

Recordando que el filtro diseñado es tipo I, el orden de este debe ser par, por lo tanto se suma 1 al resultado anterior, con lo cual se obtiene `N=22`

Se hallan los vectores `f`, `a` y `Wpond` utilizados por la función `firpm` para hallar la función de transferencia del filtro Pasa Alto con rizado constante:

```
Vector de frecuencias criticas normalizadas f=[0 ws/pi wp/pi 1]=
[0, 0.45, 0.587, 1]
Vector de magnitudes de las bandas de frecuencia a= [0, 0, 1, 1]
Función de ponderación Wpond=[1 ds/dp]= [1, 0.03075]
```

Con los valores de `f`, `a`, `Wpond` y `N` calculados, se obtiene el filtro diseñado con rizado constante:

```
h=firpm(N, f, a, Wpond);
```

La figura 5.11 muestra la respuesta en frecuencia del filtro con rizado constante, donde se puede notar que los requerimientos son logrados:

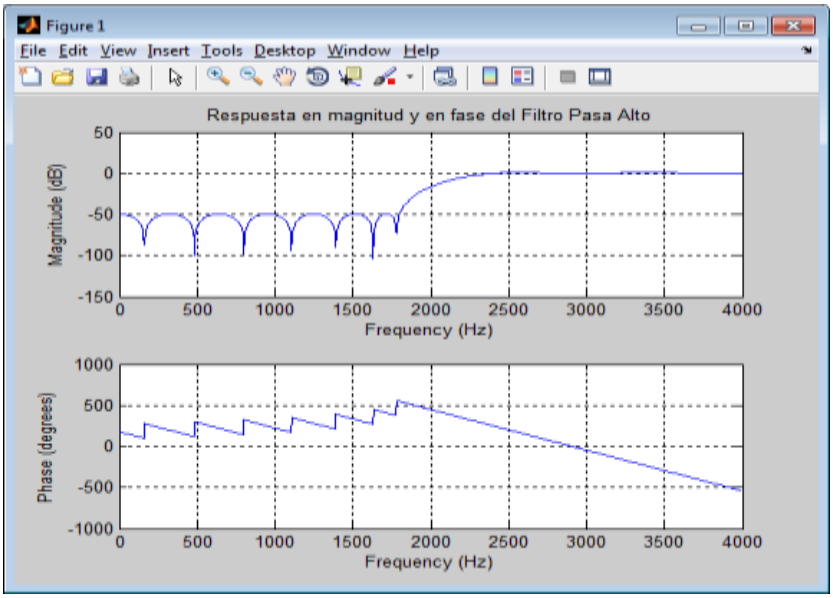


Figura 5. 11 ***Grafica de la respuesta en frecuencia del filtro Pasa Alto con Rizado constante con N=22.***

- **RESULTADOS DE LA SIMULACION EN MATLAB®**

La señal que va a ser filtrada, es la señal *handel* predefinida por Matlab®, la cual carga los datos en la variable `y`, de tamaño 73113x1 con amplitudes que van desde -0.8 hasta 0.8 (ver figura 5.12.a) y carga su frecuencia de muestreo en la variable `Fs=8192(Hz)`. Esta señal `y` es aplicada al filtro diseñado con rizado constante dando como resultado la señal almacenada en la variable `out` (ver figura 5.12.b.)

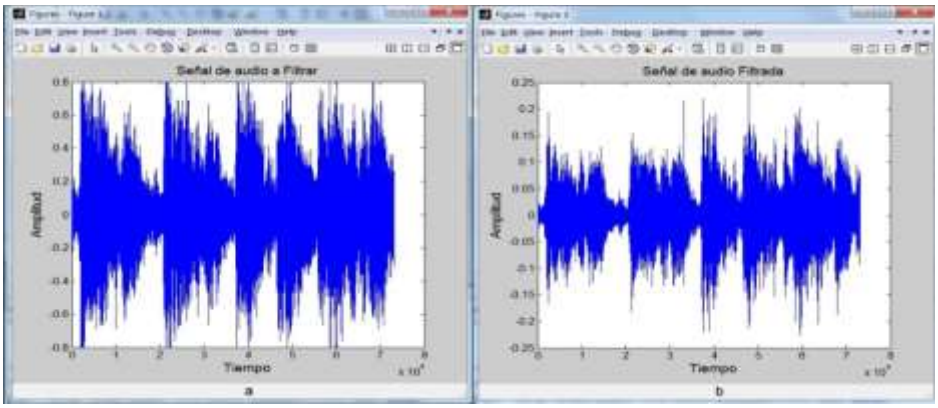


Figura 5. 12 ***Grafica Amplitud vs. Muestras de la señal a la entrada y a la salida del filtro Pasa Alto diseñado con Rizado constante***

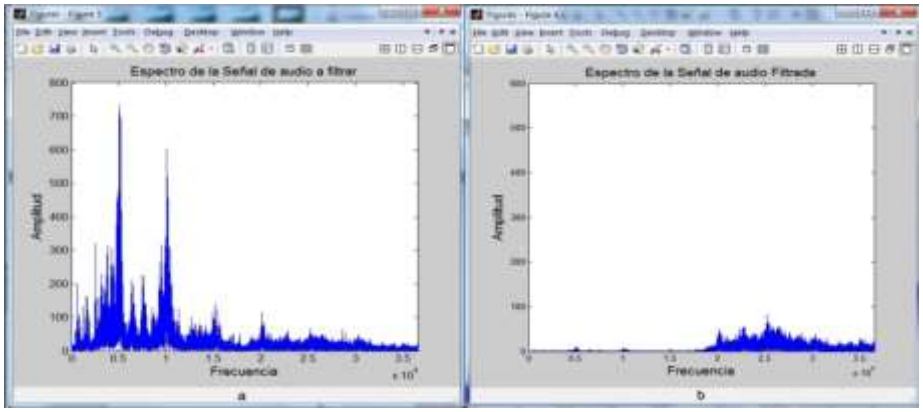


Figura 5. 13 ***Grafica del espectro de la señal de entrada y del espectro de la señal de salida del filtro Pasa Alto con Rizado constante respectivamente***

La figura 5.12 muestra que la señal de audio de salida ha cambiado respecto a la señal original. La figura 5.13 muestra el espectro de la señal de entrada así como el espectro de la señal de salida donde se puede observar que son atenuadas las frecuencias bajas.

- **RESULTADOS DE LA SIMULACION EN SIMULINK®**

  Siguiendo las indicaciones de la Práctica para la creación del modelo en Simulink® y utilizando la misma señal *handel* para ser filtrada, los resultados auditivos de la simulación en el entorno Simulink® fueron exitosos al escuchar cambios en la señal de audio de salida en comparación con la señal de entrada original, debido a las frecuencias eliminadas. (El modelo Simulink y las señales que intervienen se encuentran en el CD)

- **RESULTADOS DE LA IMPLEMENTACION EN FPGA**

  Para realizar este paso, se siguieron las indicaciones de la Práctica No.5 *IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANAS, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A y* del *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®*.

  Aplicando el mismo plan de pruebas del filtro FIR diseñado con ventana Hamming, se capturan las señales de entrada *handel* (figura 5.14.a) y de salida de la FPGA (figura 5.14.b)

como se indico anteriormente. Al graficarlas se obtuvieron las siguientes figuras para el filtro Pasa Alto con rizado constante:

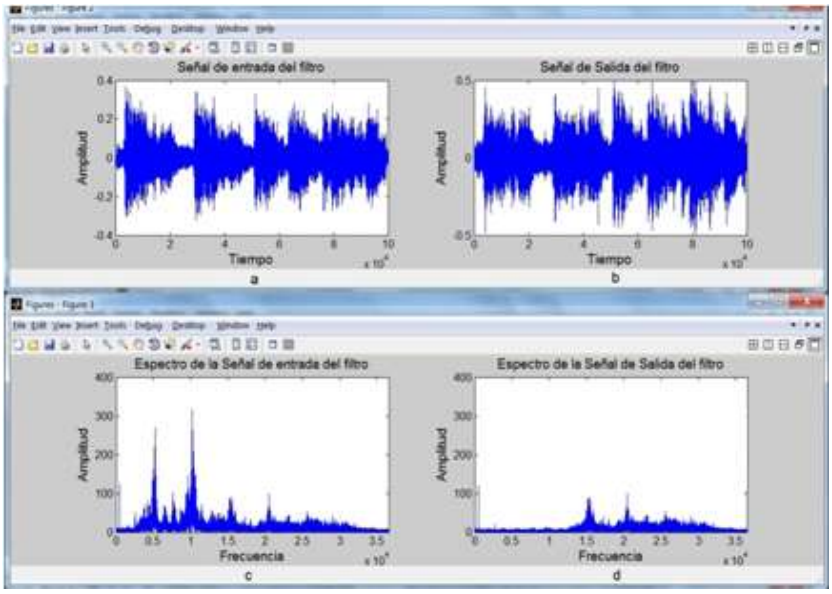


Figura 5. 14 ***Señal a la entrada y a la salida de la FPGA para el Filtro FIR Pasa Alto diseñado con Rizado Constante***

Al igual que la simulación en Matlab®, se observa mediante el análisis espectral que se cumple con las especificaciones de diseño: figura 5.14.c. Espectro señal de entrada y figura 5.14.d Espectro de señal de salida con atenuación del filtro pasa alto.

5.3 RESULTADOS DE LA PRÁCTICA 3: SIMULACION DE FILTROS DIGITALES TIPO IIR POR METODO DE BUTTERWORTH

5.3.1 Para un filtro IIR Butterworth Pasa bajo, con especificaciones:

```
fs=1700Hz;  %frecuencia de supresión
fp=1300Hz;  %frecuencia de paso
ap=1dB;     %Atenuación banda de paso
as=50dB;    %Atenuación banda de supresión
fm=8000Hz;  %frecuencia de muestreo
```

- **RESULTADOS DEL DISEÑO**

```
Frecuencia de paso analógica op=tan(wp/2)= 0.15838
Frecuencia de supresión analógica os=tan(ws/2)= 0.72654
Orden del filtro n=ceil(log10((10^(ap/10)-1)/(10^(as/10)-1))/
2*log10(op/os)) =4
Frecuencia de corte analógica oc= op*((((1-dp)^-2)-1)^(-1/(2*n)))= 0.186
```

Con los valores de `n` y `oc` calculados, se obtiene la función de transferencia del filtro IIR Pasa Bajo analógico Butterworth:

`[b, a]= butter(n,oc,'low','s')= h(s)=b(s)/a(s)=[0.00119]/[`s^4`+0.486`s^3`+ 0.1181`s^2`+ 0.0168`s^1`+0.00119]`

Con los valores de `b` y `a` calculados, se obtiene la función de transferencia del filtro IIR Pasa Bajo digital:

[bdd, add]=bilinear(b,a,fm)= b(z)/a(z) =[7.387e-04+ 0.0029z^{-1}+ 0.0044z^{-2}+0.00295z^{-3}+7.3871e-04z^{-4}]/[1+(-3.041z^{-1})+3.556z^{-2}+(-1.883z^{-3})+0.379z^{-4}]

La figura 5.15 muestra como las frecuencias `op` y `os` del filtro analógico Pasa Bajo, calculadas predistorsionando las frecuencias de diseño del filtro digital se cumplen y también las atenuaciones dadas para cada banda delimitada por estas frecuencias, además se puede observar la condición de fase no lineal que posee el filtro:

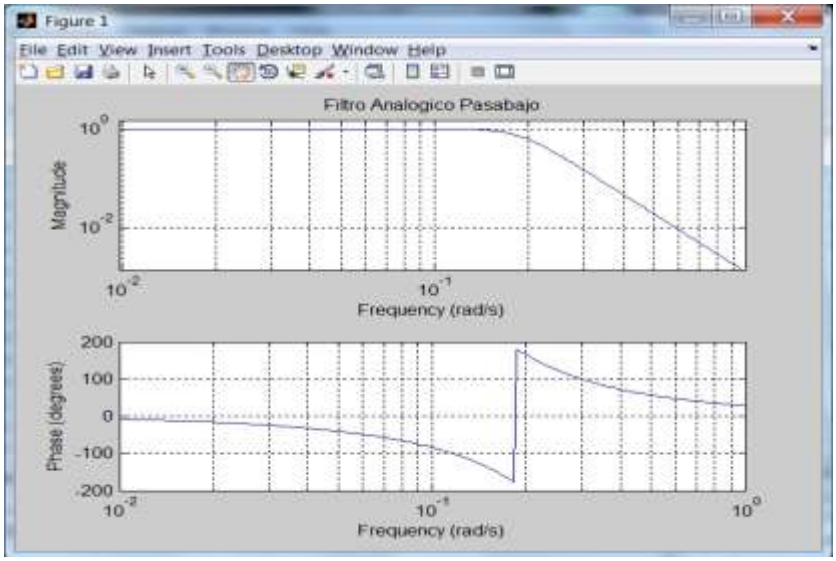


Figura 5. 15 ***Gráfica de la Respuesta en frecuencia de filtro Analógico IIR Butterworth Pasa Bajo***

La figura 5.16 muestra la respuesta en frecuencia del filtro digital, donde se observa que se cumple con las especificaciones de diseño para el filtro digital:

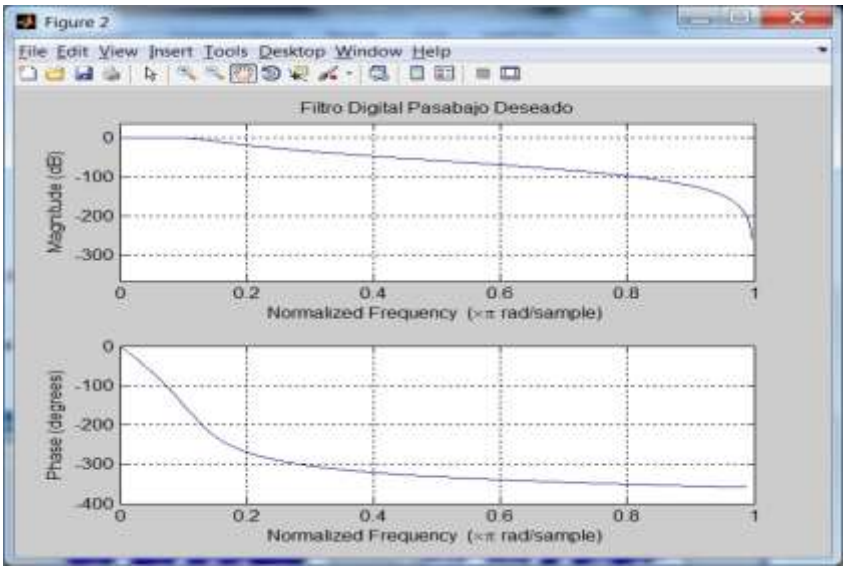


Figura 5. 16 ***Gráfica de la Respuesta en frecuencia de filtro Digital Pasa Bajo***

- **RESULTADOS DE LA SIMULACION EN MATLAB®**

  La señal que va a ser filtrada, es la señal *handel* predefinida por Matlab®, la cual carga los datos en la variable `y`, de tamaño 73113x1 con amplitudes que van desde -0.8 hasta 0.8 (ver figura 5.17.a) y carga su frecuencia de muestreo en la variable `Fs=8192(Hz)`. Esta señal `y` es aplicada al filtro diseñado con rizado constante dando como resultado la señal almacenada en la variable `out` (ver figura 5.17.b):

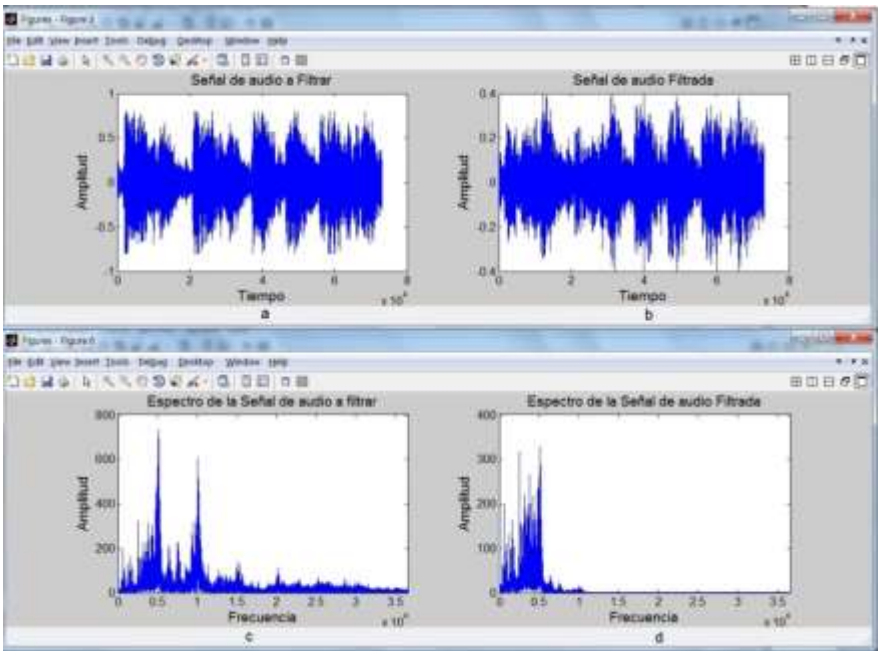


Figura 5. 17 ***Grafica Amplitud vs. Muestras y Grafica del espectro de la señal a la entrada y a la salida del filtro Pasa Bajo diseñado con Método de Butterworth***

La figura 5.17 muestra la señal de audio de salida en comparación con la de entrada, donde se destaca el cambio que hubo en la señal, además el espectro de la señal de salida también permite observar que las frecuencias altas son atenuadas

- **COMPROBACION DE LA ESTABILIDAD DEL FILTRO**

  Para comprobar la estabilidad del filtro analógico, se utiliza la función `zplane(b,a)`, que da como resultado la siguiente grafica:

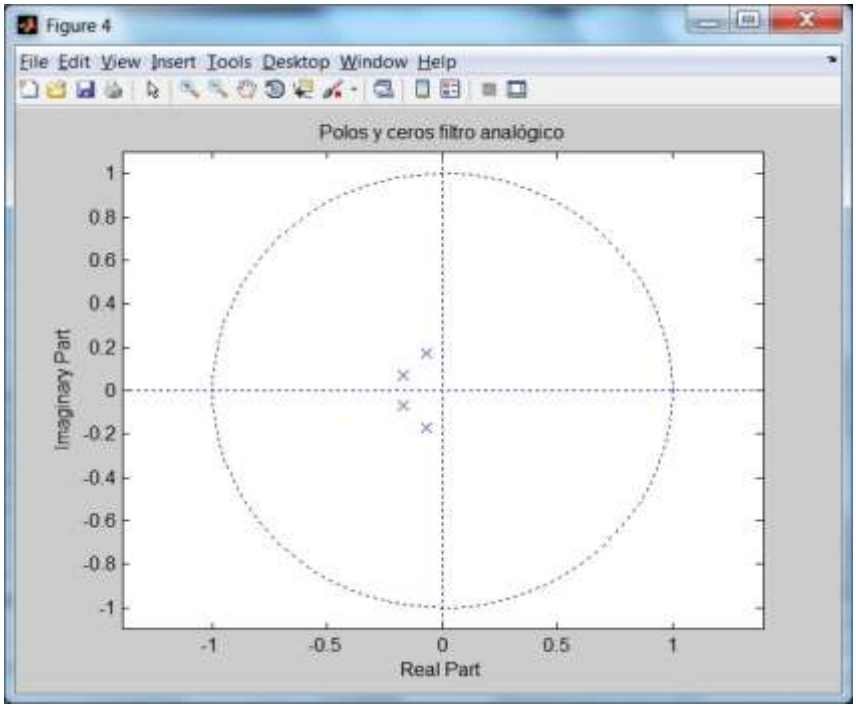


Figura 5. 18 ***Ubicación de los polos y los ceros del filtro analógico***

Para comprobar la estabilidad del filtro digital diseñado, se utiliza la función `zplane(bdd,add)`, que da como resultado la siguiente grafica:

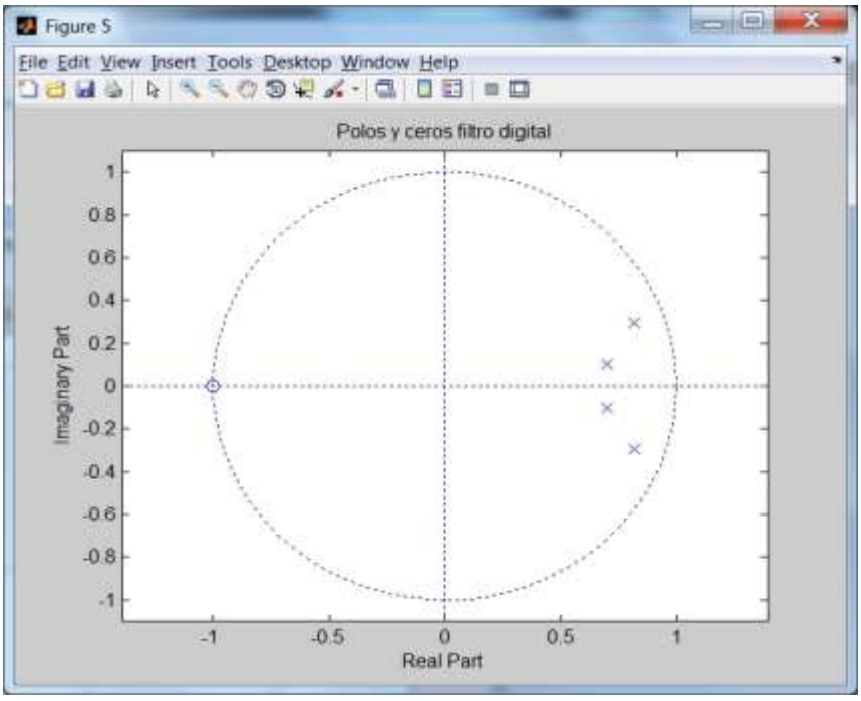


Figura 5. 19 ***Ubicación de los polos y los ceros del filtro Digital***

Las figuras 5.18 y 5.19 muestran la estabilidad de los filtros analógico, pues los polos y ceros están ubicados en el semiplano izquierdo y digital, ya que tanto los polos como los ceros están dentro del círculo de radio unidad.

- **RESULTADOS DE LA SIMULACION EN SIMULINK®**

  Siguiendo las indicaciones de la Práctica para la creación del modelo en Simulink® y utilizando la misma señal *handel* para ser filtrada, los resultados auditivos de la simulación en el entorno Simulink® fueron exitosos al escuchar cambios en la señal de audio de salida en comparación con la señal de entrada original, debido a las frecuencias eliminadas. (El modelo Simulink y las señales que intervienen se encuentran en el CD)

- **RESULTADOS DE LA IMPLEMENTACION**

  Para realizar este paso, se siguieron las indicaciones de la Práctica No.5 *IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANAS, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A y* del *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®*.

  Aplicando el mismo plan de pruebas del filtro FIR diseñado con ventana Hamming, se capturan las señales de entrada *handel* (figura 5.20.a) y de salida de la FPGA (figura 5.20.b) como se indico anteriormente. Al graficarlas se obtuvieron las siguientes figuras para el filtro IIR Pasa Bajo Butterworth:

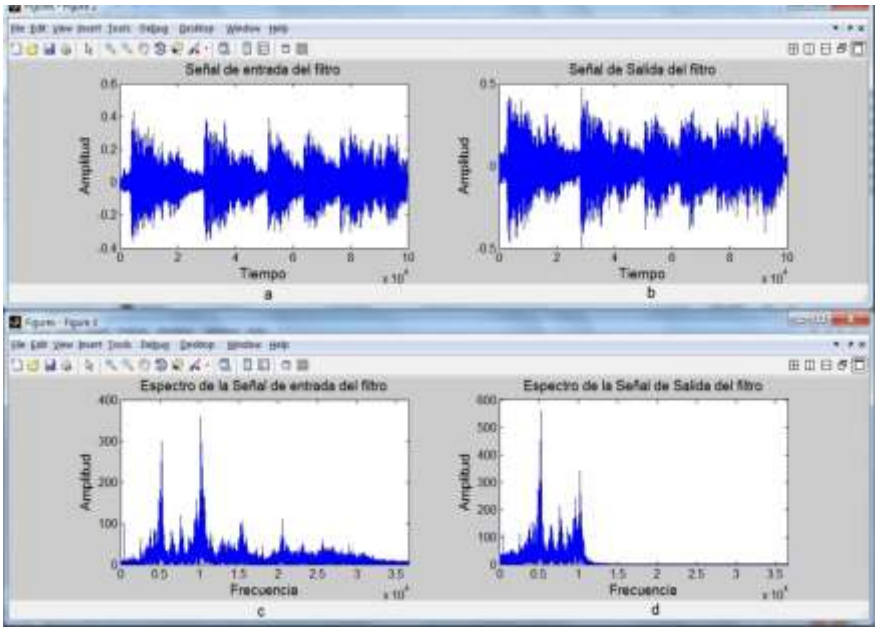


Figura 5. 20 ***Señal a la entrada y a la salida de la FPGA para el Filtro IIR Pasa Bajo diseñado método de Butterworth***

Al igual que la simulación en Matlab®, se observa mediante el análisis espectral que se cumple con las especificaciones de diseño: figura 5.20.c. Espectro señal de entrada y figura 5.20.d Espectro de señal de salida con atenuación del filtro pasa bajo

5.4 RESULTADOS DE LA PRÁCTICA 4: SIMULACION DE FILTRO DIGITAL ADAPTATIVO LMS

- **RESULTADOS DE LA SIMULACIÓN PARA FILTRO ADAPTATIVO LMS DE 6 COEFICIENTES.**
  Para realizar este paso, se siguen las indicaciones de la práctica, en la cual se muestra cómo crear el modelo en Simulink® de la figura 3.29 con el bloque *Embedded MATLAB® Function* y modificar en el *Editor Embedded Matlab®* las funciones que conforman el algoritmo adaptativo LMS, descargadas de http://www.mathworks.com/Matlab®central/fileexchange/16278, para luego ser compiladas y finalmente simular el modelo resultante. La señal de audio utilizada fue la señal *handel* nombrada anteriormente (figura 5.21.b), *input* es la señal de ruido aplicado, en este caso fue ruido blanco gaussiano de media 0 y con varianza de 0.15 (figura, 5.21.a), *desired* es la señal de audio contaminada (figura 5.21.c) y *y* es la señal de salida o señal de error, es decir la señal de audio original (figura 5.21.d):

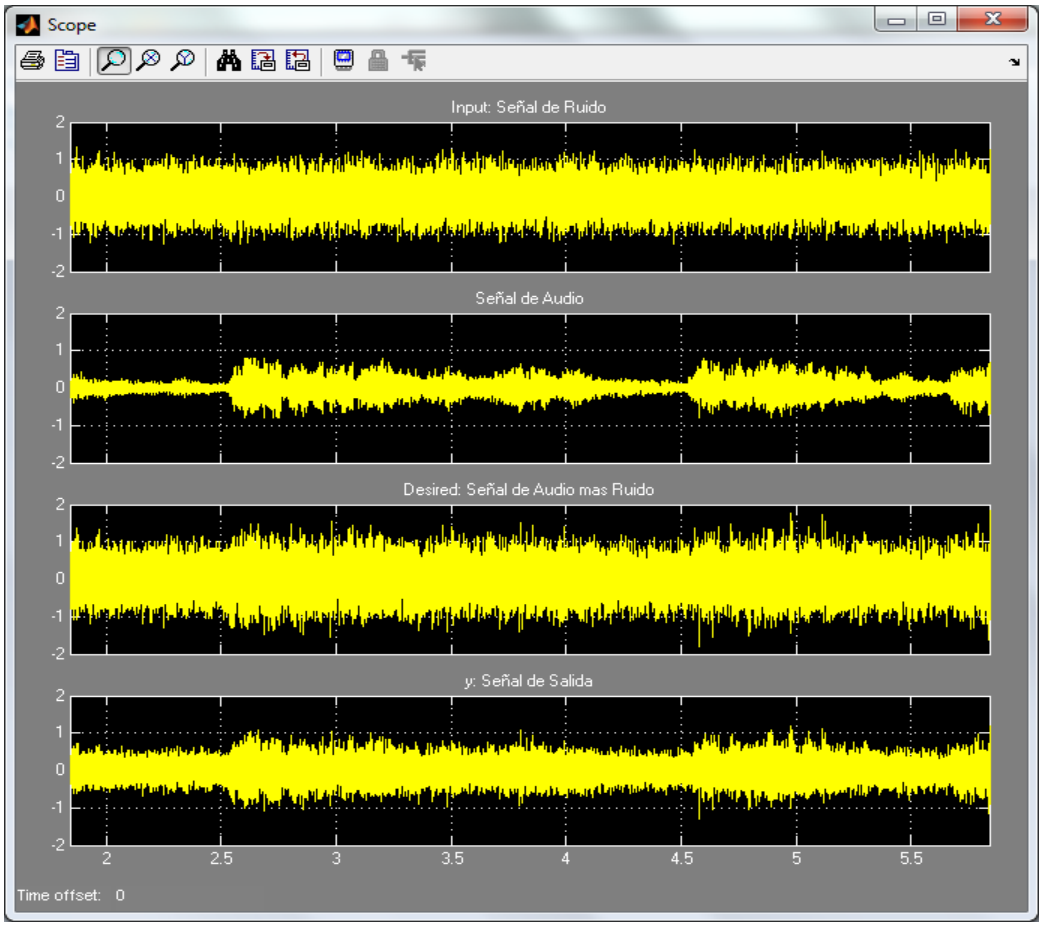


Figura 5. 21 ***Resultados de la Simulación***

(El modelo Simulink y las señales que intervienen se encuentran en el CD)

- **RESULTADOS DE LA IMPLEMENTACION**

Para realizar este paso, se siguieron las indicaciones de la Práctica No.5 *IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANA, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A,* que muestra cómo generar a partir del modelo Simulink® el código VHDL y las indicaciones del *Manual de Usuario para la Implementación de Filtros Digitales en FPGA Spartan 3A de Xilinx®* que muestra como implementar el código generado junto con el código de los conversores ADC y DAC.

El plan de pruebas de la implementación del filtro Adaptativo LMS sobre la FPGA, consiste en montar el sistema como se indica en la figura 4.7 (Integración de FPGA con etapa de adaptación y los parlantes), escuchar la señal de ruido, la señal contaminada y la señal de salida (señal de error) gracias a la utilización de dos interruptores de la FPGA (ver Practica No.5), además, capturar y observar estas señales por medio de Matlab®. La captura se hizo posible gracias a la función de Matlab® `wavrecord(N, Fs)`, donde `N` es el numero de muestras grabadas que para el caso se consideró guardar 100000 muestras y `Fs` es la frecuencia de muestreo utilizada para la grabación, en este caso se utilizó Fs= 11025(Hz).

Capturando y graficando la señal de ruido, se obtuvo la siguiente grafica para el filtro Adaptativo LMS:

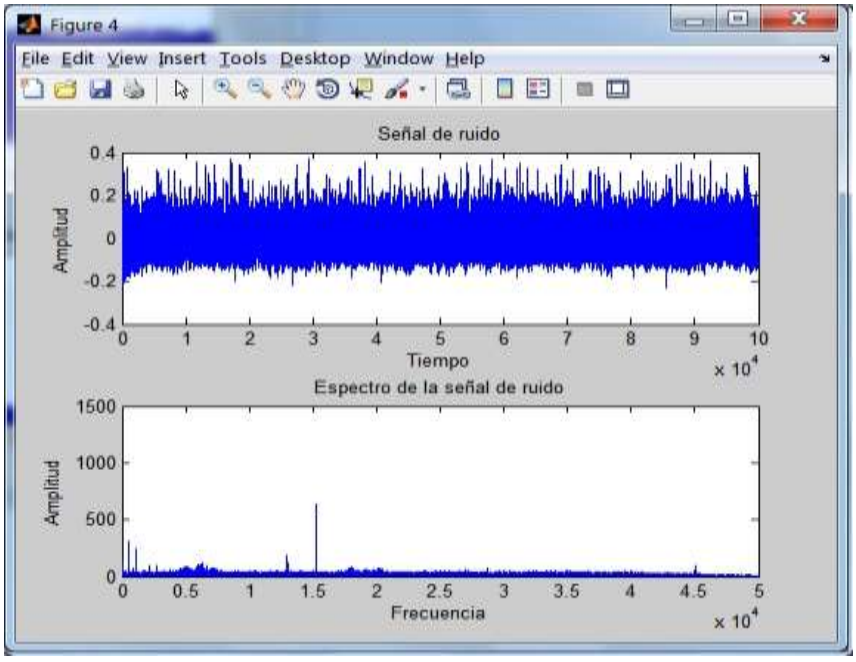


Figura 5. 22 ***Señal de ruido***

Capturando y graficando la señal de audio contaminada, se obtuvo la siguiente grafica para el filtro Adaptativo LMS:

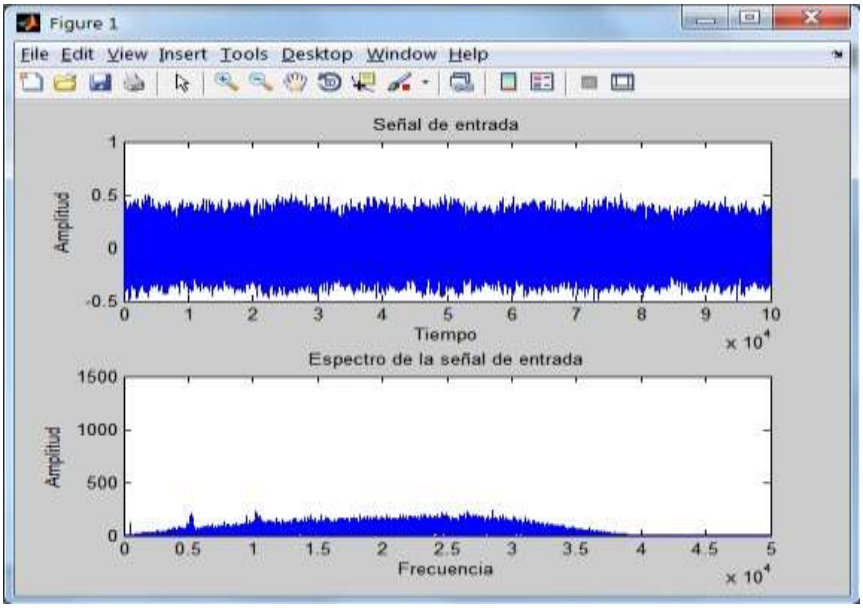


Figura 5. 23 ***Señal de audio contaminada***

Capturando y graficando la señal salida, se obtuvo la siguiente grafica para el filtro Adaptativo LMS:

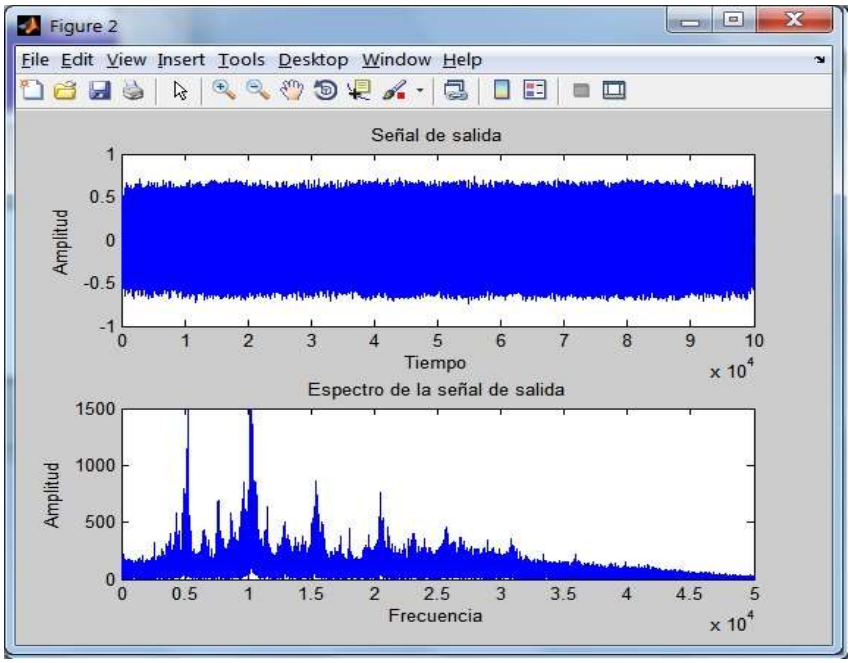


Figura 5. 24 ***Señal de salida***

La señal de entrada sin interferencia se muestra a continuación:

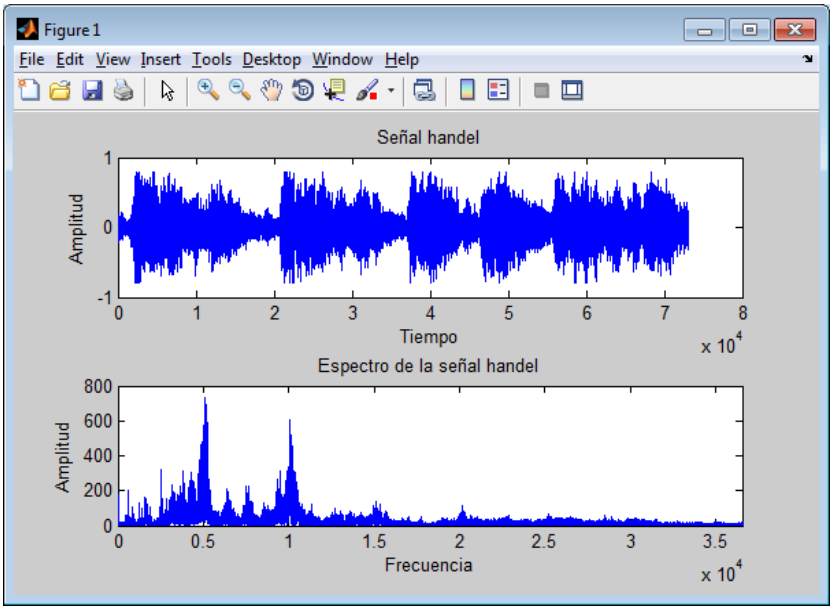


Figura 5. 25 ***Señal de audio original*****.**

Se puede concluir al observar las figuras 5.24 y 5.25 que el filtro adaptativo cumple la función para la que fue diseñado, ya que a pesar de que a la entrada hay una señal demasiado contaminada (figura 5.23), a la salida de la FPGA se puede ver el espectro de la señal *handel* con una pequeña cantidad de ruido (figura 5.24).

El anexo E contiene los formatos de evaluación de las prácticas de laboratorio, diligenciados por estudiantes de ingeniería electrónica de la FIET, que desarrollaron las practicas adjuntas a este trabajo.

CAPITULO 6
CONCLUSIONES Y TRABAJOS FUTUROS

1. *Matlab®* como entorno de desarrollo, facilita el análisis, el desarrollo y la detección y corrección de errores que se presentaron al generar los algoritmos de los filtros digitales tipo FIR, IIR y adaptativo. Gracias a la claridad del lenguaje, a las múltiples opciones de realizar un mismo proceso y a la sección de *Help*, se logra aplicar varias funciones que permitieron cumplir los objetivos de diseño con éxito. Además, el poder simular los diseños tanto en el entorno Matlab® como en el entorno *Simulink®*, permite comprobar el funcionamiento de forma visual y auditiva, detectando así de manera más contundente cualquier error que se presente.

2. La herramienta *Simulink® HDL Coder* de Matlab® permite generar código VHDL de una manera rápida, eficaz y automática, a partir de un modelo Simulink® en punto flotante o en punto fijo. Características que son muy importantes a la hora de trabajar con FPGA, pues se ahorra bastante tiempo en la generación de códgio hdl y así se puede centrar la atención en la integración de este código con los puertos de entrada y salida de cualquier tarjeta, en este caso los conversores analógico a digital y digital a analógico.

3. Se comprobó que la técnica de diseño de rizado constante genera filtros de orden menor con las mismas especificaciones que las demás técnicas de diseño de filtro FIR, como esta especificado en la literatura.

4. Se comprobó la funcionalidad de la herramienta *Embedded Matlab®*, necesaria cuando los algoritmos no cuentan con un bloque de Simulink® específico dedicado a su funcionamiento o cuando están mejor declarados en lenguaje textual de Matlab® que en lenguaje grafico de Simulink®. Esta herramienta se utilizo para la implementación del filtro Adaptativo LMS

5. Los recursos de la FPGA Spartan 3A fueron limitantes en el desarrollo del proyecto, ya que al tener anchos de banda de transición muy pequeños, los filtros generados son más largos, es decir tienen un mayor número de coeficientes, que al tratar de ser implementados en la FPGA producen el siguiente error: *the design is too large for the given device and package.* La técnica del rizado constante fue más tolerante a este limitante ya que permitió que se implementaran filtros con pequeños anchos de banda de transición cuyo orden no generaba este error.

6. El algoritmo adaptativo versión gama, no pudo ser implementado, ya que luego de generar el código VHDL a partir del modelo en Simulink® realizado con el bloque *Embedded Matlab® Function,* al tratar de implementar el código generado en el entorno *ISE* de Xilinx®, el XST que es la tecnología de síntesis de Xilinx®, genero el error: *Operator <INVALID OPERATOR> must have constant operands or first operand must be power of 2*, este error se debe a que el XST tiene entre sus restricciones que no debe haber una división donde el denominador sea una variable y

este algoritmo en una de sus ecuaciones posee una división donde el termino del denominador es la señal de audio retrasada.

7. La tecnología de síntesis de Xilinx® XST, no permitió la implementación de los filtros en punto flotante ya que al generar código VHDL a partir de modelos en punto flotante con la herramienta *Simulink® VHDL Coder* la declaración de los puertos de entrada y salida del filtro digital no es concurrente con las restricciones impuestas por Xilinx® en el XST, ya que están declaradas de tipo real y este tipo de dato solo es soportado para el cálculo de las constantes dentro de funciones.

8. La herramienta *ISE Proyect Navigator* ofrece una interfaz agradable al usuario, pues tiene los archivos y los procesos ordenados en ventanas correctamente clasificadas, lo cual hace que el trabajo al implementar algún diseño sea realizado sin demasiadas dificultades.

9. La mayoría de los filtros generados con la herramienta *Simulink® HDL Coder* consumen gran parte de los recursos de la FPGA Spartan 3A, haciendo que sea complicado combinarlos con diseños más grandes. Tal vez la ineficiencia se debe a que Matlab® no incluye a Xilinx® dentro de las opciones en las cuales se escoge el dispositivo para el cual va a ser generado el código VHDL.

TRABAJOS FUTUROS:

10. Ajustar el Algoritmo Adaptativo Acelerador Regresivo Versión Gama para que se adecue a la FPGA de Xilinx®.

11. Hacer uso del conector estéreo de audio digital que posee la tarjeta en conjunto con el conversor analágico digital para implementar cualquier diseño de filtro digital. Debido a que con la utilización del conversor digital analógico para el caso de este trabajo, hubo una pérdida de bits, se esperaría que al utilizar el conector estéreo, se obtenga una mayor calidad de audio a la salida, dado que esta pérdida no se presentaría.

12. Dado que el filtro adaptativo programado en la tarjeta posee pocos coeficientes, esto debido a las restricciones de la misma, puede que su funcionamiento no sea el óptimo, por lo que sería una buena práctica implementarlo en una tarjeta con más capacidades.

13. Realizar proyectos en los cuales se haga uso de otros puertos de la FPGA, dado que se cuenta con el manual de usuario y con las guías de laboratorio que exponen la creación de un proyecto desde el proceso de simulación hasta el proceso de implementación, lo cual facilita cualquier proceso.

BIBLIOGRAFIA

[1] http://agamenon.tsc.uah.es/Asignaturas/it/tds/apuntes/practicas_iir_fir_2op.pdf. [Consultado: Marzo 12, 2010]

[2] http://home.anadolu.edu.tr/~sgorgulu/dsphw/Lab3/Lab3.pdf. [Consultado: Marzo 13, 2010]

[3] http://jdsp.engineering.asu.edu/EXAMPLES/lab4.pdf. [Consultado: Marzo 10, 2010]

[4] http://www.me.umn.edu/courses/me4231/labs/Lab5.pdf. [Consultado: Marzo 12, 2010]

[5] http://www.seas.upenn.edu/~ese206/labs/LabButterworth/ButterworthFilter.pdf. [Consultado: Marzo 13, 2010]

[6] http://www.labc.usb.ve/gceglia/Ec1168/GUIAS/Filtros%20activos_files/Filtros%20Activos.pdf . [Consultado: Marzo 12, 2010]

[7] http://www.comm.utoronto.ca/~bkf/ECE431/Lab04_Filters.pdf. [Consultado: Marzo 12, 2010]

[8] http://www.bme.uconn.edu/courses/BME%203400/Labs/LAB%204.pdf. [Consultado: Marzo 12, 2010]

[9] http://linux0.unsl.edu.ar/~rvilla/c3m09/prt05.pdf. [Consultado: Marzo 10, 2010]

[10] http://www.cems.uvm.edu/~mirchand/classes/EE275/2007/Matlab®/Lab4.pdf. [Consultado: Marzo 12, 2010]

[11] Universidad del Cauca. Laboratorio de Sistemas de Telecomunicaciones I

[12] OPPENHEIM A. WILLSKY A. SISTEMAS Y SEÑALES. Segunda edición

[13] KUO S. LEE B. REAL TIME DIGITAL SIGNAL PROCESSING-2001

[14] PROAKIS J. MONOLAKIS D. DIGITAL SIGNAL PROCESSING. Third Edition-1996.

[15] IFEACHOR E. JERVIS B. DIGITAL SIGNAL PROCESSING. A PRACTICAL APPROACH. Second Edition

[16] LAI E. PRACTICAL DIGITAL SIGNAL PROCESSING FOR ENGINEERS AND TECHNICIANS. Primera edición-2003

[17] INGLE V. PROAKIS J. DIGITAL SIGNAL PROCESSING, USING MATLAB® V4. Northeasterm University-1997

[18] Help Matlab®

[19] HERRAMIENTAS DE DISEÑO Y ARITMETICA DE PUNTO FIJO-PROCESAMIENTO DIGITAL DE SEÑALES. LabDSP. Universidad Nacional de Córdoba

[20] XST User Guide-2005

[21] DINIZ P. ADAPTIVE FILTERING-ALGORITHMS AND PRACTICAL IMPLEMENTATION. Tercera Edicion-2008

[22] BOROUJENY B. ADAPTIVE FILTERS-THEORY AND APPLICATIONS. 1998

[23] HAIKIN S. ADAPTIVE FILTER THEORY. Tercera Edicion.

[24] REALPE J. ESTUDIO DEL EFECTO DE LA VARIACION TEMPORAL DE PARAMETROS DEL ALGORITMO ACELERADOR REGRESIVO VERSION γ (ARγ). Universidad del Cauca. Popayan-2009.

[25] JOJOA P. UM ALGORITMO ACELERADOR DE PARAMETROS. Escuela Politécnica de Universidad de Sao Paulo. Sao Paulo-2003

[26] B.FARHANG-BOROUJENY. ADAPTIVE FILTERS, THEORY AND APPLICATIONS - National University of Singapore

[27] DIAZ P. FERNANDEZ P. LA DISTRIBUCION NORMAL. Unidad de Epidemiologia Clínica y Bioestadística. Complexo Hospitalario Juan Canalejo-2001
[28] XILINX®. PROGRAMMABLE LOGIC DESIGN - Quick Start Handbook
[29] BROWN S. ROSE J. ARCHITECTURE OF FPGAs AND CPLDs: TUTORIAL. Department of Electrical and Computer Engineering. University of Toronto
[30] VALLEJO M. AYALA J. FPGA: NOCIONES BASICAS E IMPLEMENTACION. Departamento de Ingeniería Electrónica. Universidad Politécnica de Madrid-Abril 2004.
[31] SPARTAN-3A FPGA FAMILY: DATASHEET- Marzo 6, 2009
[32] SPARTAN-3A FPGA STARTED KIT BOARD USER GUIDE. For Revision C Board -Junio 21, 2007
[33] Help de ISE Proyect Navigator

ANEXO A

Toolbox necesarias para realización de las prácticas

Para realizar el diseño, simulación e implementación de los filtros digitales se hizo uso de la herramienta Matlab® versión 7.9 (R2009b) en conjunto con algunas de sus herramientas adicionales, las cuales se listan a continuación:

Para realizar el diseño:

- **Signal Processing Toolbox Versión 6.12 (R2009b)** es una colección de herramientas basadas en el entorno Matlab®. La Toolbox soporta un amplio rango de operaciones de procesamiento de la señal, desde la generación de formas de onda hasta el diseño y la implementación de filtros, modelado paramétrico y análisis espectral. Las funciones utilizadas de esta Toolbox son:
    - Sinc
    - Freqz
    - Kaiser
    - Firpm
    - Butter
    - Bilinear
    - Freqs
    - tf2sos
    - fvtool

- **Embedded MATLAB® (R2009b)** Embedded MATLAB® es un subconjunto del lenguaje MATLAB®. El subconjunto Embedded MATLAB® soporta la generación de código eficiente para crear un prototipo e implementarlo en sistemas embebidos. Se compone de más de 270 operadores y funciones de MATLAB® y más de 90 funciones de *Toolbox Fixed Point*. Esta Toolbox se utilizo para el diseño y simulación del algoritmo LMS.
- Las demás funciones utilizadas en las practicas forman parte de las funciones básicas de Matlab®

Para realizar la simulación:

- **Simulink® Versión 7.4 (R2009b)** modela, simula y analiza sistemas dinámicos. Permite plantear una pregunta acerca de un sistema, modelar el sistema, y ver qué pasa.
- **Signal Processing Blockset Versión 6.10 (R2009b)** proporciona algoritmos y herramientas para el diseño y la simulación de sistemas de procesamiento de la señal. Se puede desarrollar algoritmos DSP para el procesamiento de el habla y el audio, detección de señales, seguimiento por radar, comunicaciones de banda base, y otras aplicaciones. La mayoría de los algoritmos y

herramientas están disponibles como objetos del Sistema (para uso en MATLAB®) y bloques (para uso en Simulink®). Se debe tener previamente instalados Matlab®, Simulink® y la Signal Processing Toolbox para que la Signal Processing Blockset funcione correctamente. Los bloques utilizados de esta herramienta son:

- Signal From Workspace
- Digital Filter
- To Audio Device
- Time Scope

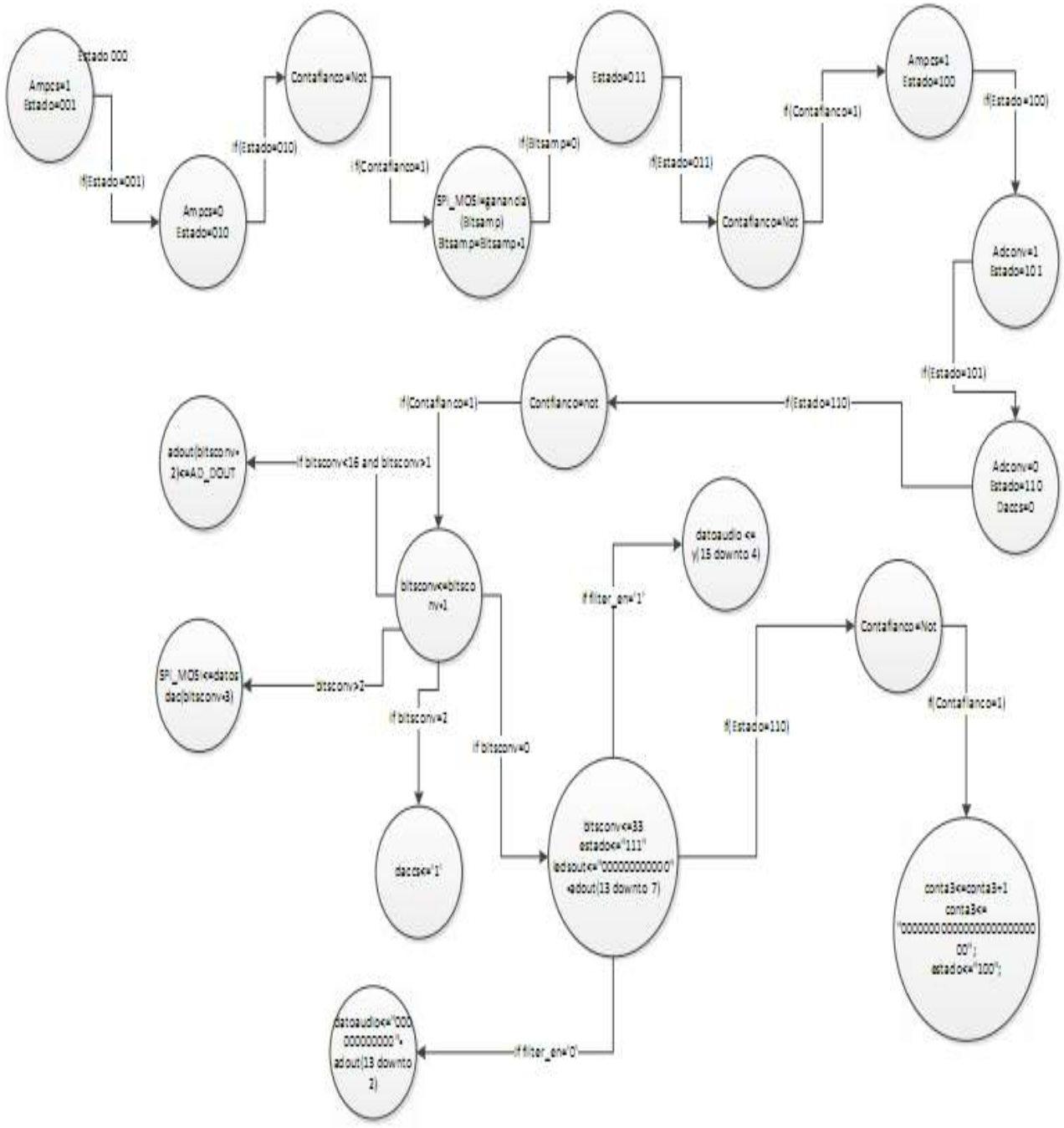
Para realizar la implementación:

- **Fixed Point Versión 3.0 (R2009b)** proporciona datos de tipo punto fijo y permite el desarrollo de algoritmos proporcionando aritmética de punto fijo. Permite la interoperabilidad con Simulink® y la Signal Processing Blockset
- **Simulink® Fixed Point 6.2 (R2009b)** habilita las capacidades intrínsecas de punto fijo de los siguientes productos a través de la familia de productos Simulink®:
  - Simulink®
  - Signal Processing Blockset
- **Simulink® HDL Coder Versión 1.6 (R2009b)**, luego de usar el Simulink® Fixed Point, esta herramienta software le permite generar códigos del lenguaje de descripción hardware (HDL) basados en modelos de Simulink® y máquinas de estados finitos Stateflow. Esta Herramienta necesita la previa de la Instalación de Matlab®, Simulink®, Fixed Point Toolbox, Simulink® Fixed Point.

ANEXO B

Maquina de estados de los conversores ADC y DAC

A continuación se muestra la máquina de estados que fue la base para la generación de código VHDL que configura y permite el uso de los conversores ADC y DAC:



ANEXO C

Requerimientos e instalación de la herramienta software ISE versión 12

Requerimientos de instalación para ISE Desing Suite 12.

- Sistemas operativos
    - ISE Desing Suite 12 soporta las versiones de sistema operativo XP SP1/SP2/SP3, Vista Business y Windows 7 de Microsoft, en sus versiones de 32 y 64 bits. Además soporta otros sistemas operativos como Red Hat Enterprise Linux y SUSE Linux Enterprise.
- Requerimientos del sistema
    - Requerimientos para la instalación de los cables:
      Para instalar la plataforma del cable USB II el sistema debe tener al menos un puerto USB 1.1. La recomendación de Xilinx® es usar un puerto USB 2.0 para mejor funcionamiento.
    - Requerimientos de Memoria:
      Para el correcto funcionamiento del ISE Desing Suite 12, que va a trabajar con una Spartan 3A- XC3S700A es necesario contar con 420Mb de memoria RAM si el sistema es de 32bits y con 610Mb de memoria RAM si el sistema operativo es de 64bits. Nótese que esta es la cantidad de memoria usada únicamente por la aplicación, por lo que en realidad es necesario contar con por lo menos 2Gb de memoria en el ordenador.
- Equipo y requisitos de permisos:
    - Permisos de directorio: permisos de escritura deben existir para todos los directorios que contienen archivos de diseño que haya que modificar.
    - Monitor VGA de 16-bits a color con una resolución mínima recomendada de 1024 x 768 píxeles.
    - DVD-ROM físico o virtual
    - Para programar los dispositivos, debe tener un cable paralelo disponible o un cable USB.

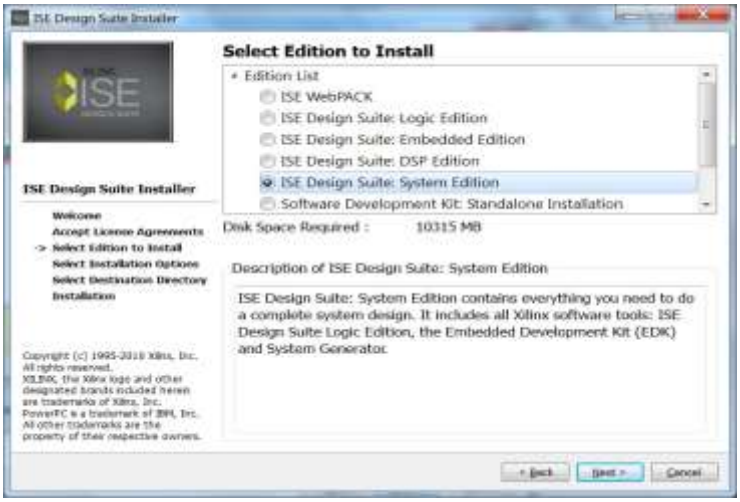
Instrucciones de instalación:

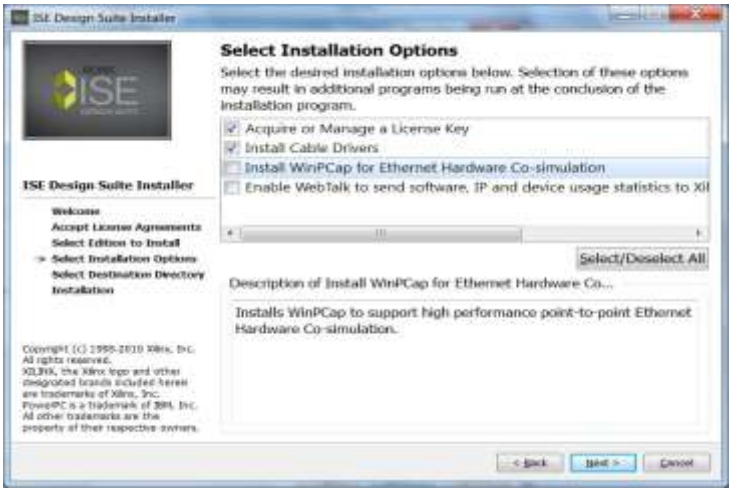
- Cerrar todos los programas antes de empezar la instalación
- Asegurarse que el equipo cumple con los requerimientos descritos antes.
- Correr el programa Setup y seguir las instrucciones sobre la pantalla para instalar el software:

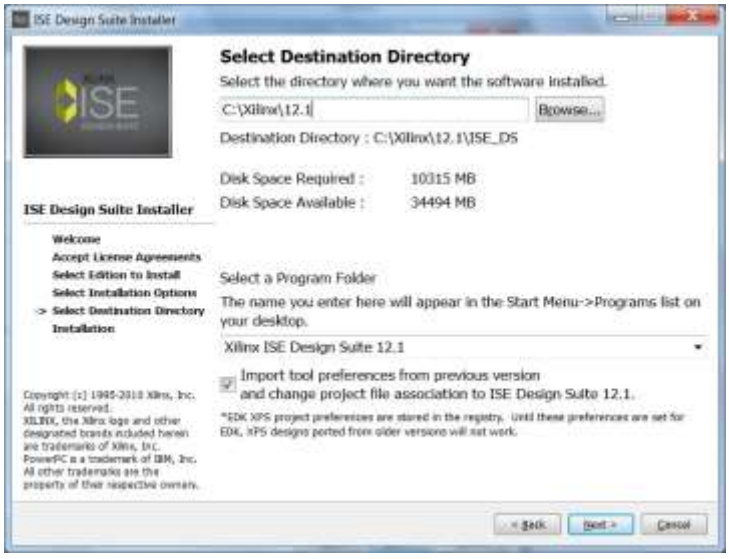

ISE Design Suite Installer
Welcome
We are glad you've chosen Xilinx as your platform development partner. This program will install ISE WebPACK, one of the four ISE Design Suite Editions or one of our two standalone products. The installation process will consist of the steps listed to the left.
You will need to have administrator privileges in order to install this software on Windows operating systems. To reduce installation time, we recommend that you disable any anti-virus software before continuing.
For the product you select to install, we also recommend that you install to a new directory. If you choose to install your selected product into a directory with an older installation of ISE Design Suite, we will require the older version to be uninstalled before proceeding.
ISE Design Suite Installer
-> Welcome
Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc.
All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
Next >
Cancel

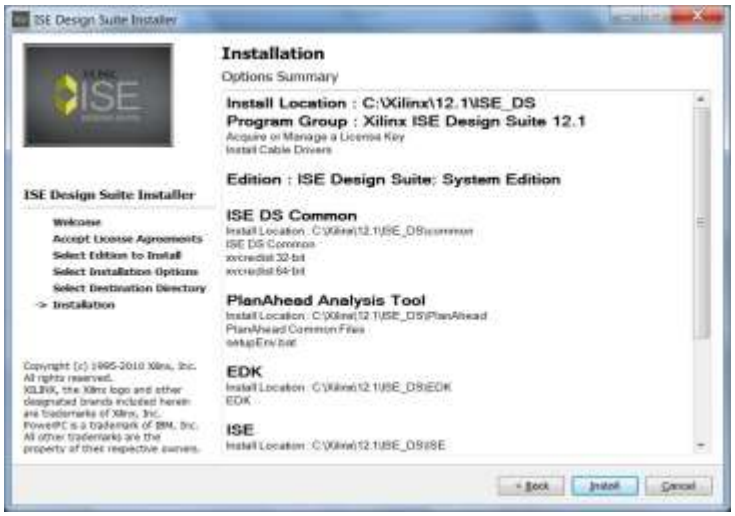

ISE Design Suite Installer
Accept License Agreements (1 of 2)
Please read the following Software License Agreement and click the checkbox below it to indicate that you accept the terms of the license agreement.
CAREFULLY READ THIS COLLECTION OF INFORMATION AND LICENSE AGREEMENTS. BY CLICKING THE "ACCEPT" OR "AGREE" BUTTON, OR OTHERWISE ACCESSING, DOWNLOADING, INSTALLING OR USING THE SOFTWARE, YOU AGREE ON BEHALF OF LICENSEE TO BE BOUND BY THIS INFORMATION AND LICENSE AGREEMENTS (TO THE EXTENT APPLICABLE TO THE SPECIFIC SOFTWARE YOU OBTAIN AND THE SPECIFIC MANNER IN WHICH YOU USE SUCH SOFTWARE).
IF LICENSEE DOES NOT AGREE TO ALL OF THE INFORMATION AND LICENSE AGREEMENTS BELOW, DO NOT CLICK THE "ACCEPT" OR "AGREE" BUTTON OR ACCESS, DOWNLOAD, INSTALL OR USE THE SOFTWARE; AND IF LICENSEE HAS ALREADY OBTAINED THE SOFTWARE FROM AN AUTHORIZED SOURCE, PROMPTLY RETURN IT FOR A REFUND.
Part One: Overview.
I accept the terms of this software license.
ISE Design Suite Installer
Welcome
-> Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc.
All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
< Back
Next >
Cancel


ISE Design Suite Installer
Accept License Agreements (2 of 2)
Please read the following Software License Agreement and click the checkbox below it to indicate that you accept the terms of the license agreement.
XILINX, INC.
END USER LICENSE AGREEMENT
CAREFULLY READ THIS END USER LICENSE AGREEMENT ("AGREEMENT"). BY CLICKING THE "ACCEPT" OR "AGREE" BUTTON, OR OTHERWISE ACCESSING, DOWNLOADING, INSTALLING OR USING THE SOFTWARE, YOU AGREE ON BEHALF OF LICENSEE TO BE BOUND BY THIS AGREEMENT.
IF LICENSEE DOES NOT AGREE TO ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT CLICK THE "ACCEPT" OR "AGREE" BUTTON OR ACCESS, DOWNLOAD, INSTALL OR USE THE SOFTWARE; AND IF LICENSEE HAS ALREADY OBTAINED THE SOFTWARE FROM AN AUTHORIZED SOURCE, PROMPTLY RETURN IT FOR A REFUND.
1. Definitions
I accept the terms of this software license.
ISE Design Suite Installer
Welcome
-> Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc.
All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
< Back
Next >
Cancel

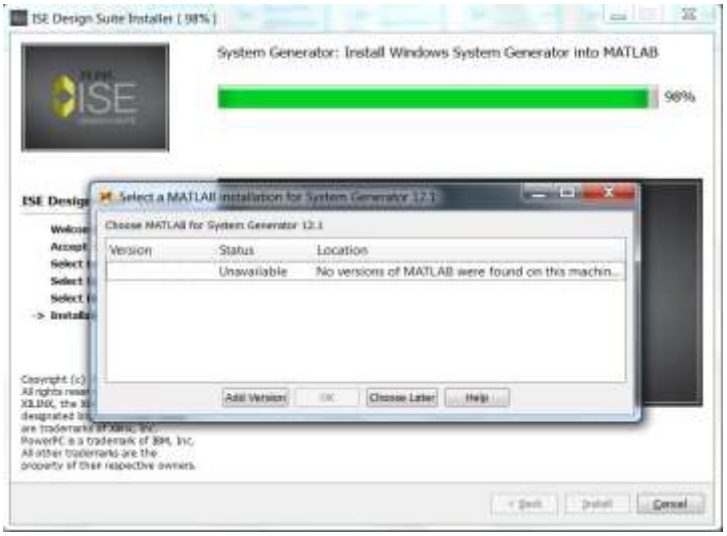

ISE Design Suite Installer
Select Edition to Install
Edition List
ISE WebPACK
ISE Design Suite: Logic Edition
ISE Design Suite: Embedded Edition
ISE Design Suite: DSP Edition
ISE Design Suite: System Edition
Software Development Kit: Standalone Installation
Disk Space Required : 10315 MB
Description of ISE Design Suite: System Edition
ISE Design Suite: System Edition contains everything you need to do a complete system design. It includes all Xilinx software tools: ISE Design Suite Logic Edition, the Embedded Development Kit (EDK) and System Generator.
ISE Design Suite Installer
Welcome
Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc. All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
< Back
Next >
Cancel

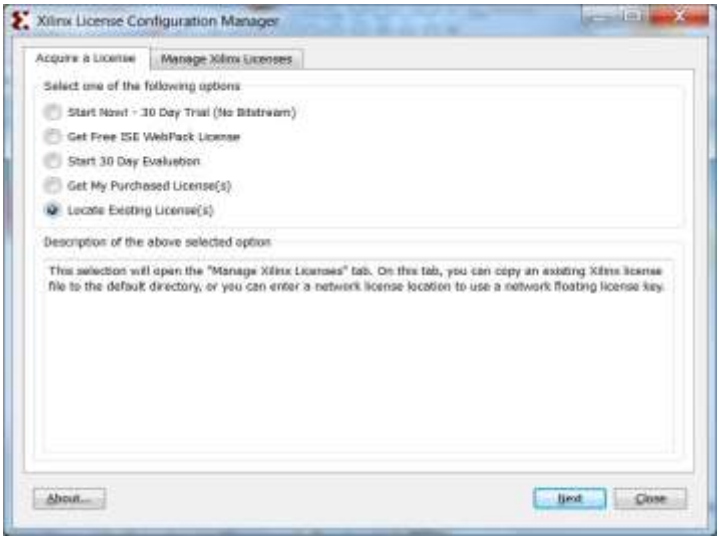

ISE Design Suite Installer
Select Installation Options
Select the desired installation options below. Selection of these options may result in additional programs being run at the conclusion of the installation program.
Acquire or Manage a License Key
Install Cable Drivers
Install WinPCap for Ethernet Hardware Co-simulation
Enable WebTalk to send software, IP and device usage statistics to Xil
Select/Deselect All
Description of Install WinPCap for Ethernet Hardware Co...
Installs WinPCap to support high performance point-to-point Ethernet Hardware Co-simulation.
ISE Design Suite Installer
Welcome
Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc. All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
< Back
Next >
Cancel

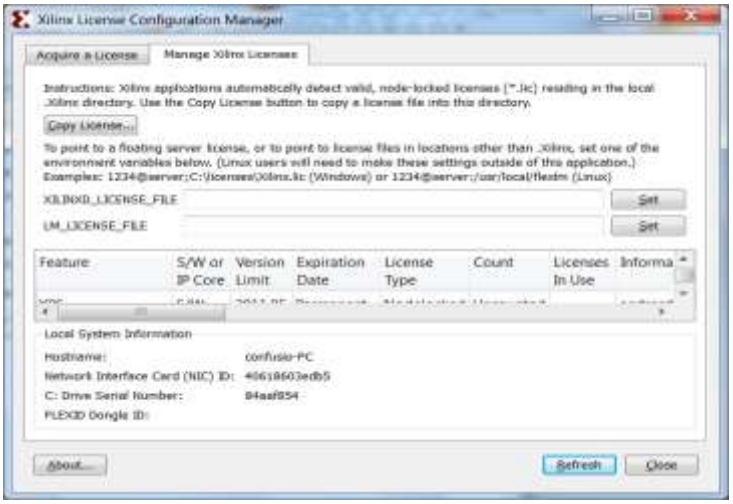

ISE Design Suite Installer
Select Destination Directory
Select the directory where you want the software installed.
C:\Xilinx\12.1
Browse...
Destination Directory : C:\Xilinx\12.1\ISE_DS
Disk Space Required : 10315 MB
Disk Space Available : 34494 MB
Select a Program Folder
The name you enter here will appear in the Start Menu->Programs list on your desktop.
Xilinx ISE Design Suite 12.1
Import tool preferences from previous version and change project file association to ISE Design Suite 12.1.
*EDK XPS project preferences are stored in the registry. Until these preferences are set for EDK, XPS designs ported from older versions will not work.
ISE Design Suite Installer
Welcome
Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
Installation
Copyright (c) 1995-2010 Xilinx, Inc. All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the property of their respective owners.
< Back
Next >
Cancel

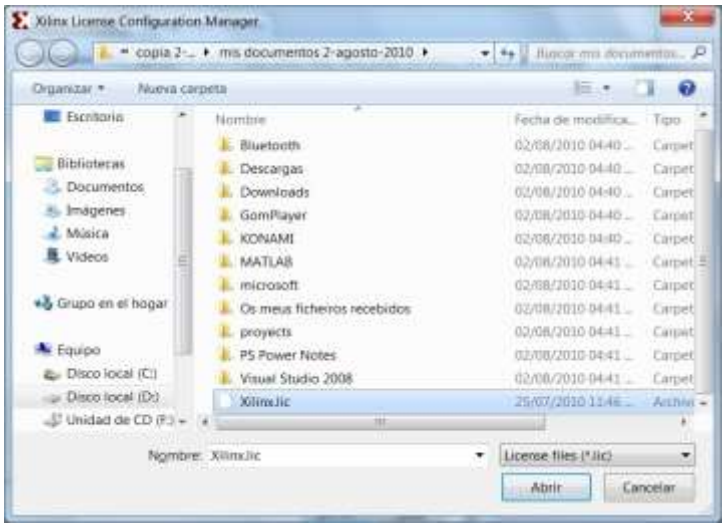

ISE Design Suite Installer
Installation
Options Summary
Install Location : C:\Xilinx\12.1\ISE_DS
Program Group : Xilinx ISE Design Suite 12.1
Acquire or Manage a License Key
Install Cable Drivers
Edition : ISE Design Suite: System Edition
ISE DS Common
Install Location: C:\Xilinx\12.1\ISE_DS\common
ISE DS Common
xvcredist 32-bit
xvcredist 64-bit
PlanAhead Analysis Tool
Install Location: C:\Xilinx\12.1\ISE_DS\PlanAhead
PlanAhead Common Files
setupEnv.bat
EDK
Install Location: C:\Xilinx\12.1\ISE_DS\EDK
EDK
ISE
Install Location: C:\Xilinx\12.1\ISE_DS\ISE
ISE Design Suite Installer
Welcome
Accept License Agreements
Select Edition to Install
Select Installation Options
Select Destination Directory
-> Installation
Copyright (c) 1995-2010 Xilinx, Inc.
All rights reserved.
XILINX, the Xilinx logo and other
designated brands included herein
are trademarks of Xilinx, Inc.
PowerPC is a trademark of IBM, Inc.
All other trademarks are the
property of their respective owners.
< Back
Install
Cancel


ISE Design Suite Installer (4%)
ISE DS Common: xvcredist 32-bit
4%
Microsoft Visual C++ 2005 SP1 Redistributable Packag...
Please read the following license agreement. Press the PAGE DOWN key to see the rest of the agreement.
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT VISUAL C++ 2005 RUNTIME LIBRARIES
These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft
* updates,
* supplements,
* Internet-based services, and
* support services
for this software, unless other terms accompany those items. If so, those terms apply.
By using the software, you accept these terms. If you do not accept them, do not use the software.
Do you accept all of the terms of the preceding License Agreement? If you choose No, Install will close. To install you must accept this agreement.
Yes
No
ISE Design Suite
Welcome
Accept License
Select Edition t
Select Installat
Select Destinat
-> Installation
XILINX
Cancel


ISE Design Suite Installer (98%)
System Generator: Install Windows System Generator into MATLAB
98%
Select a MATLAB installation for System Generator 12.1
Choose MATLAB for System Generator 12.1
Version
Status
Location
Unavailable
No versions of MATLAB were found on this machin...
Add Version
OK
Choose Later
Help
ISE Design
Welcome
Accept
Select I
Select I
Select I
-> Installa
PowerPC is a trademark of IBM, Inc.
All other trademarks are the
property of their respective owners.
Back
Install
Cancel


Xilinx License Configuration Manager
Acquire a License
Manage Xilinx Licenses
Select one of the following options
Start Now! - 30 Day Trial (No Bitstream)
Get Free ISE WebPack License
Start 30 Day Evaluation
Get My Purchased License(s)
Locate Existing License(s)
Description of the above selected option
This selection will open the "Manage Xilinx Licenses" tab. On this tab, you can copy an existing Xilinx license file to the default directory, or you can enter a network license location to use a network floating license key.
About...
Next
Close


Xilinx License Configuration Manager
Acquire a License
Manage Xilinx Licenses
Instructions: Xilinx applications automatically detect valid, node-locked licenses (*.lic) residing in the local .Xilinx directory. Use the Copy License button to copy a license file into this directory.
Copy License...
To point to a floating server license, or to point to license files in locations other than .Xilinx, set one of the environment variables below. (Linux users will need to make these settings outside of this application.)
Examples: 1234@server;C:\licenses\Xilinx.lic (Windows) or 1234@server:/usr/local/flexlm (Linux)
XILINXD_LICENSE_FILE
Set
LM_LICENSE_FILE
Set
Feature
S/W or IP Core
Version Limit
Expiration Date
License Type
Count
Licenses In Use
Informa
Local System Information
Hostname: confusio-PC
Network Interface Card (NIC) ID: 40618603edb5
C: Drive Serial Number: 84aaf854
PLEXID Dongle ID:
About...
Refresh
Close


Xilinx License Configuration Manager
copia 2 ... mis documentos 2-agosto-2010
Buscar mis documentos
Organizar
Nueva carpeta
Escritorio
Bibliotecas
Documentos
Imágenes
Música
Videos
Grupo en el hogar
Equipo
Disco local (C:)
Disco local (D:)
Unidad de CD (F:)
Nombre
Fecha de modifica...
Tipo
Bluetooth
Descargas
Downloads
GomPlayer
KONAMI
MATLAB
microsoft
Os meus ficheiros recebidos
proyects
PS Power Notes
Visual Studio 2008
Xilinx.lic
25/07/2010 11:46 ...
Archivo
Nombre: Xilinx.lic
License files (*.lic)
Abrir
Cancelar

Si el sistema operativo es Windows XP o Windows 7, al conectar la FPGA y estar encendida, aparecerán los Wizard que ayudaran a instalar los controladores.

ANEXO D

Funciones de respuesta al impulso para filtros selectivos en frecuencia tipo I

Para realizar el diseño de los filtros digitales pasa bajo, pasa alto y elimina banda, se parte de un filtro pasa banda así:

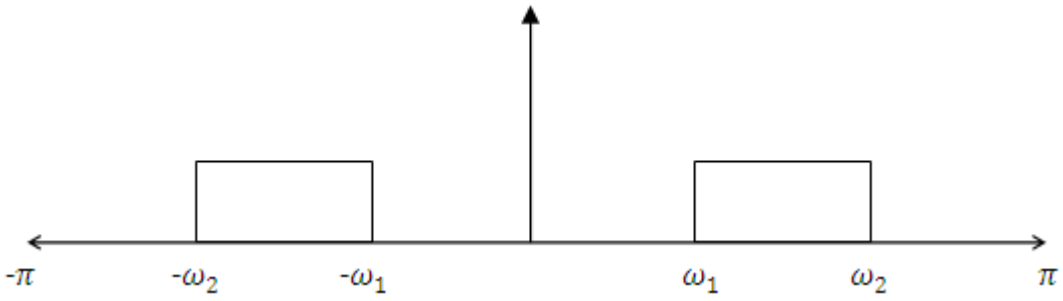


Figura D. 1 ***Filtro pasa banda***

Donde la función de respuesta al impulso está dada por:

$$h(n) = \frac{\omega_2}{\pi} sinc\left(\frac{\omega_2(n-\frac{N}{2})}{\pi}\right) - \frac{\omega_1}{\pi} sinc\left(\frac{\omega_1(n-\frac{N}{2})}{\pi}\right)$$

- Analizando la figura D.1, para conseguir un filtro pasa bajo: $\omega_1, -\omega_1 = 0$

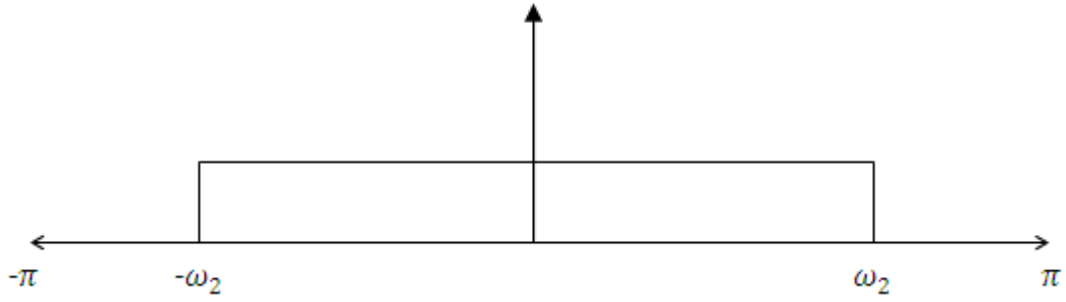


Figura D. 2 ***Filtro pasa bajo***

Donde la función de respuesta al impulso está dada por:

$$h(n) = \frac{\omega_2}{\pi}\, Sinc\left(\frac{\omega_2\left(n-\frac{N}{2}\right)}{\pi}\right) - \frac{0}{\pi}\, Sinc\left(\frac{0\,\left(n-\frac{N}{2}\right)}{\pi}\right)$$

$$h(n) = \frac{\omega_2}{\pi}\, Sinc\left(\frac{\omega_2\left(n-\frac{N}{2}\right)}{\pi}\right)$$

- Analizando la figura D.1, para conseguir un filtro pasa alto: $\omega_{2,} - \omega_2 = \pi$

Figura D. 3 ***Filtro pasa alto***

Donde la función de respuesta al impulso está dada por:

$$h(n) = \frac{\pi}{\pi}\, Sinc\left(\frac{\pi\left(n-\frac{N}{2}\right)}{\pi}\right) - \frac{\omega_1}{\pi}\, Sinc\left(\frac{\omega_1\left(n-\frac{N}{2}\right)}{\pi}\right)$$

$$h(n) = \mu\left(n-\frac{N}{2}\right) - \frac{\omega_1}{\pi}\, Sinc\left(\frac{\omega_1\left(n-\frac{N}{2}\right)}{\pi}\right)$$

- Analizando las figuras D.2 y D.3, para conseguir un filtro elimina banda, se suma un filtro pasa alto con un filtro pasa bajo así:

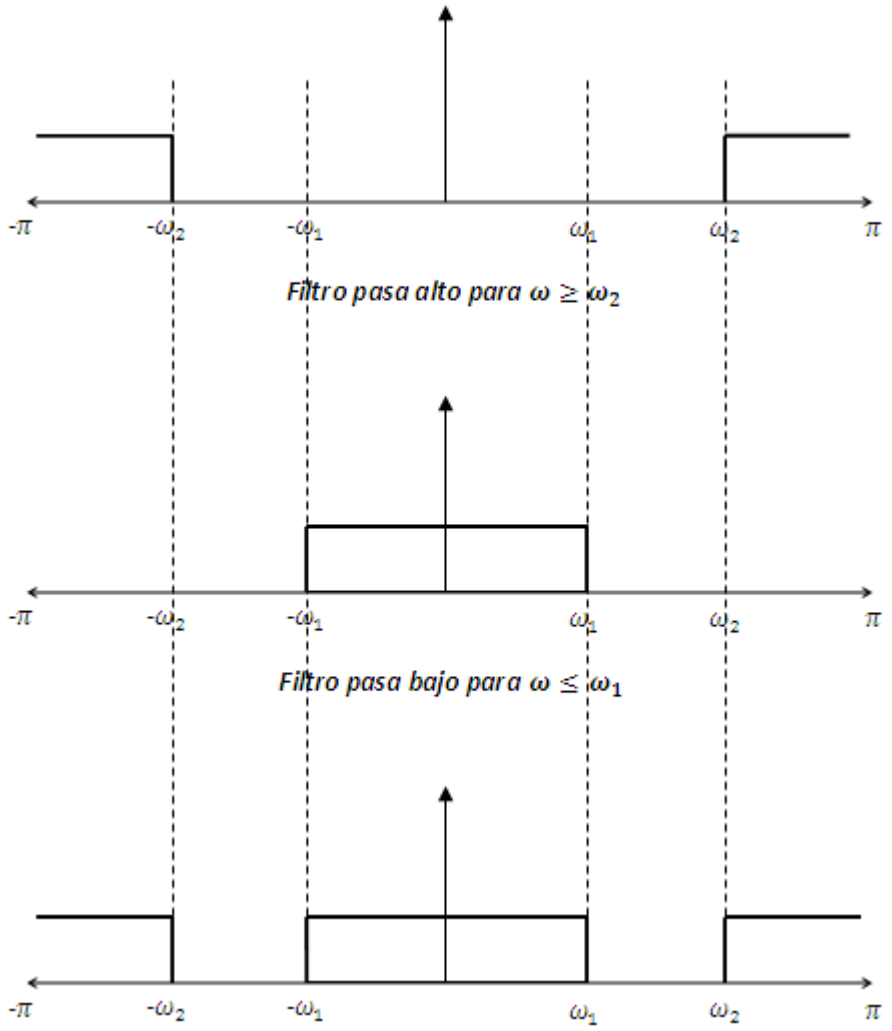


Figura D. 4 ***Filtro elimina banda para*** $\omega_1 \leq \omega \leq \omega_2$

Donde la función de respuesta al impulso está dada por:

$$h(n) = \frac{\omega_1}{\pi}\, Sinc\left(\frac{\omega_1\left(n-\frac{N}{2}\right)}{\pi}\right) + \mu\left(n-\frac{N}{2}\right) - \frac{\omega_2}{\pi}\, Sinc\left(\frac{\omega_2\left(n-\frac{N}{2}\right)}{\pi}\right)$$

ANEXO E

Formatos de Evaluación para las prácticas de laboratorio.

FORMATO PARA: PRACTICA No 1. SIMULACION DE FILTROS DIGITALES TIPO FIR POR EL METODO DE VENTANA KAISER Y OTRO METODO DE VENTANA.

Claridad: Grado de Dificultad:

E: Excelente A: Alto

B: Bueno M: Medio

R: Regular B: Bajo

M: Malo

Parte de la practica	Claridad				Grado de Dificultad		
	E	B	R	M	A	M	B
Objetivos	X					x	
Introducción	X					x	
Marco Teórico	X					x	
Trabajo Previo	X					x	
Ejemplo Practico 1 (Hamming)	X					x	
Procedimiento de Laboratorio (Hamming)	X					x	
Diseño en Matlab		x				x	
Simulación en Matlab	X					x	
Migración al entorno Simulink	X					X	
Resultados		X				X	
Ejemplo Practico 2 (Kaiser)	X					x	
Procedimiento de Laboratorio (Kaiser)	X					x	
Diseño en Matlab	X						x
Simulación en Matlab	X						x
Migracion al entorno Simulink	X						X
Resultados	X						x
Trabajo complementario	X						X
Reporte	X					x	

FORMATO PARA: PRÁCTICA No 2. SIMULACION DE FILTROS DIGITALES TIPO FIR POR EL METODO DE RIZADO CONSTANTE.

Parte de la practica	Claridad				Grado de Dificultad		
	E	B	R	M	A	M	B
Objetivos	X					x	
Introducción	X					x	
Marco Teórico	X					x	
Trabajo Previo	X				x		

Ejemplo Practico	X					x	
Procedimiento de laboratorio para Filtro con rizado constante	X					x	
Diseño en Matlab	X					x	
Simulación en Matlab	X					x	
Migración al entorno Simulink	X					x	
Resultados	X					X	
Trabajo Complementario		x				x	
Reporte		x				x	

FORMATO PARA: PRÁCTICA No 3. SIMULACION DE FILTROS DIGITALES IIR POR METODO DE BUTTERWORTH

Parte de la practica	Claridad				Grado de Dificultad		
	E	B	R	M	A	M	B
Objetivos	X					X	
Introducción	X					X	
Marco Teórico	X					X	
Trabajo Previo	X				X		
Ejemplo Practico	X				X		
Procedimiento de laboratorio para Filtro IIR Butterworth		x				X	
Diseño en Matlab		x				X	
Simulación en Matlab		x				X	
Migración al entorno Simulink		x				X	
Resultados		X				X	
Trabajo Complementario		X				X	
Reporte		x				X	

FORMATO PARA: PRÁCTICA No 4. SIMULACION DE FILTRO DIGITAL ADAPTATIVO LMS

Parte de la practica	Claridad				Grado de Dificultad		
	E	B	R	M	A	M	B
Objetivos	X					X	
Introducción	X					X	
Marco Teórico	X					X	
Trabajo Previo	X				X		
Ejemplo Practico	X				X		
Procedimiento de laboratorio para Filtro Adaptativo LMS		x				X	
Diseño y Simulación para filtro LMS		x			X		
Resultados		X				X	
Trabajo Complementario		X				X	
Reporte		x				X	

FORMATO PARA: PRÁCTICA No 5. IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANAS, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A

Parte de la practica	Claridad				Grado de Dificultad		
	E	B	R	M	A	M	B
Objetivos	X					X	
Introducción	X					X	
Marco Teórico	x					X	
Trabajo Previo	x				X		
Ejemplo Practico			x		X		
Procedimiento de laboratorio para implementar los modelos Simulink de los Filtros			x			X	
Conversión a modelos de punto fijo	x				x		
Generación de código VHDL	x						x
Integración de los códigos generados con los conversores ADC y DAC			x		x		
Implementación del código			x		x		
Análisis del desempeño	x				X		
Resultados		X				X	
Trabajo Complementario		X				X	
Reporte		x				X	

MANUAL DE USUARIO PARA LA IMPLEMENTACIÓN DE FILTROS DIGITALES EN FPGA SPARTAN 3A DE XILINX

✓ Generalidades

Para empezar es necesario conocer más acerca del archivo de proyecto, su extensión es .xise y es un archivo XML que contiene los datos releventes para el proyecto como son: información acerca de la versión de software ISE, la lista de archivos contenidos en el proyecto, configuración de fuente incluyendo propiedades del diseño y de los procesos. La información que no está contenida en el archivo de proyecto es: la información del estado de los procesos, la historia de comandos y las restricciones de datos.

Es importante notar que el archivo .ucf (ver más adelante) que debe estar incluido en el proyecto, no está incluido en la información de restricciones de datos.

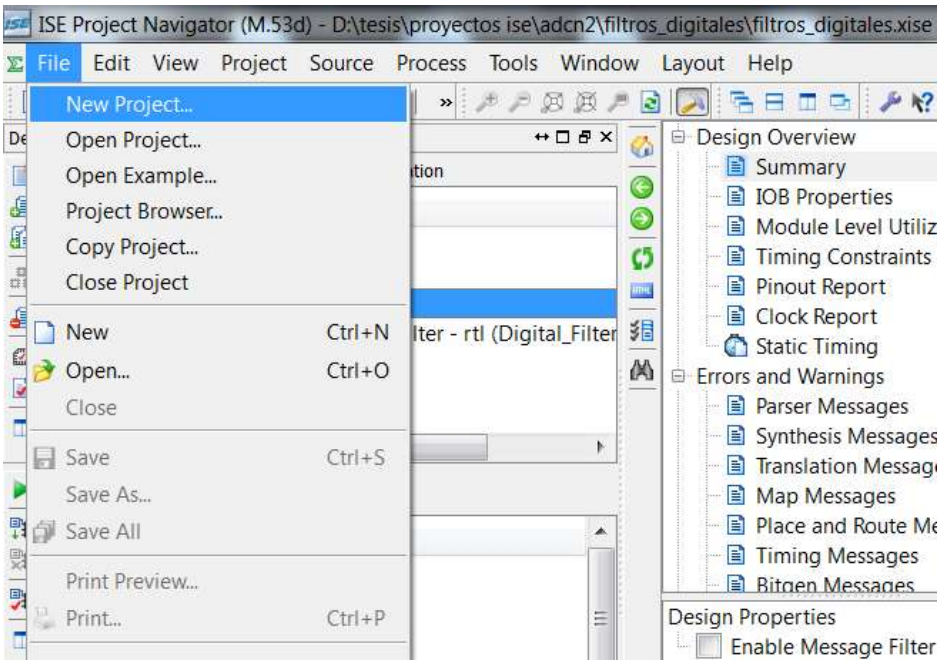
Un archivo de extensión .gise también existe y contiene los datos de los estados de los procesos, pero no es necesario interactuar directamente con este archivo.

✓ Implementación de filtros selectivos en frecuencia FIR e IIR

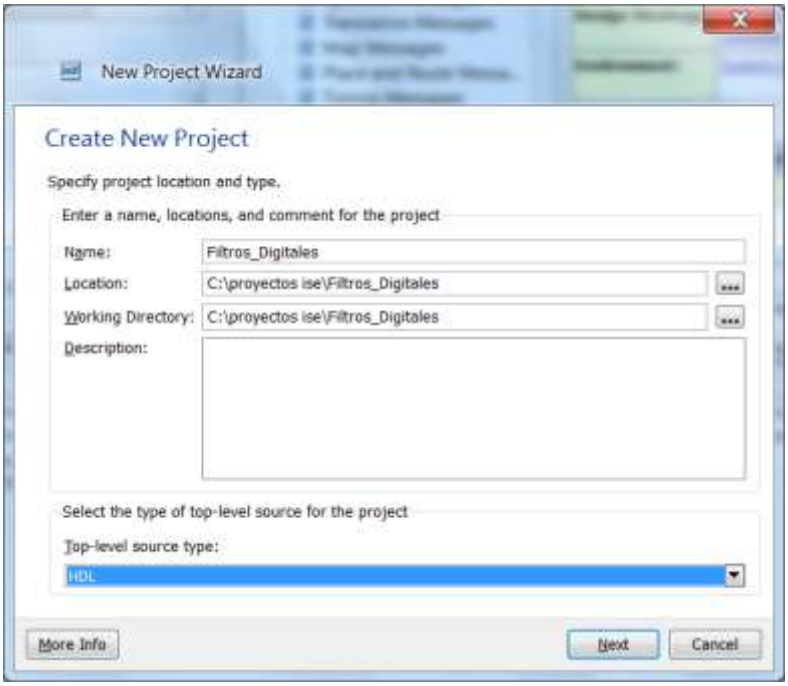
- Creación de un proyecto

Primero se debe crear un nuevo proyecto para luego agregar los archivos fuente y establecer las propiedades de los procesos. Ahora siga los pasos especificados por las siguientes figuras:

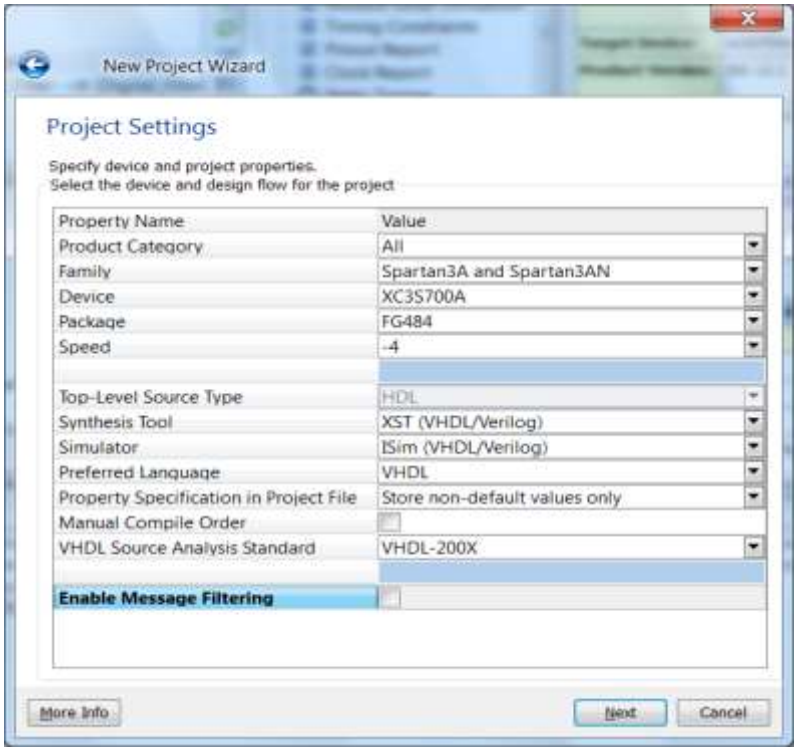
1. Ejecute el *ISE Proyec Navigator*  y Seleccione *File ->New Project*



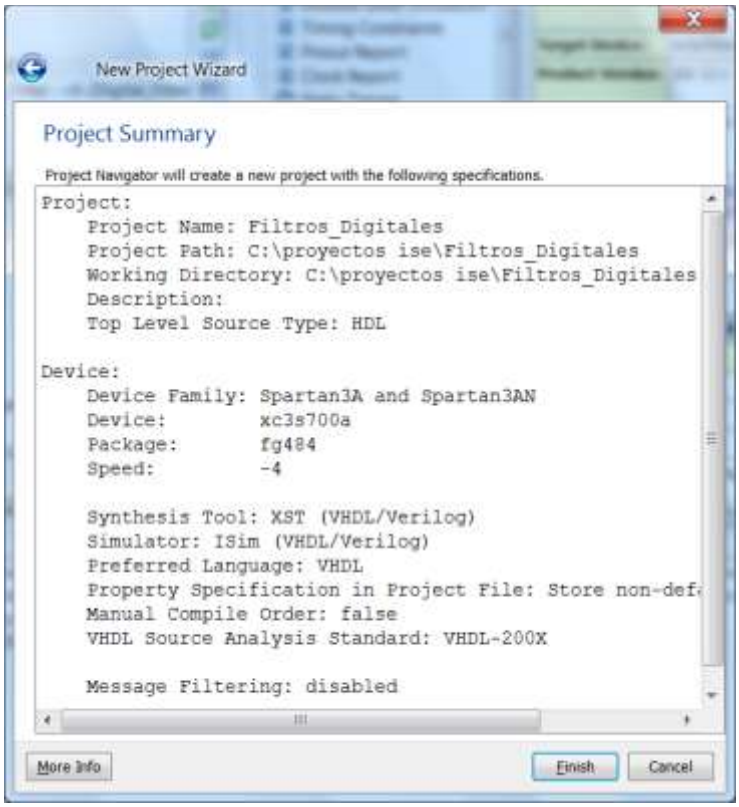
2. Proporcione el nombre, la ubicación y el directorio de trabajo que desea para el proyecto. Además se pueden agregar algunos comentarios. El tipo de fuente de máximo nivel con el que se trabajara en este manual es HDL.



3. Especifique el dispositivo y las propiedades del proyecto.



Después de realizar los pasos anteriores aparecerá la siguiente ventana. Que indica las especificaciones del proyecto a crear. Para finalizar la creación presionar *Finish*.

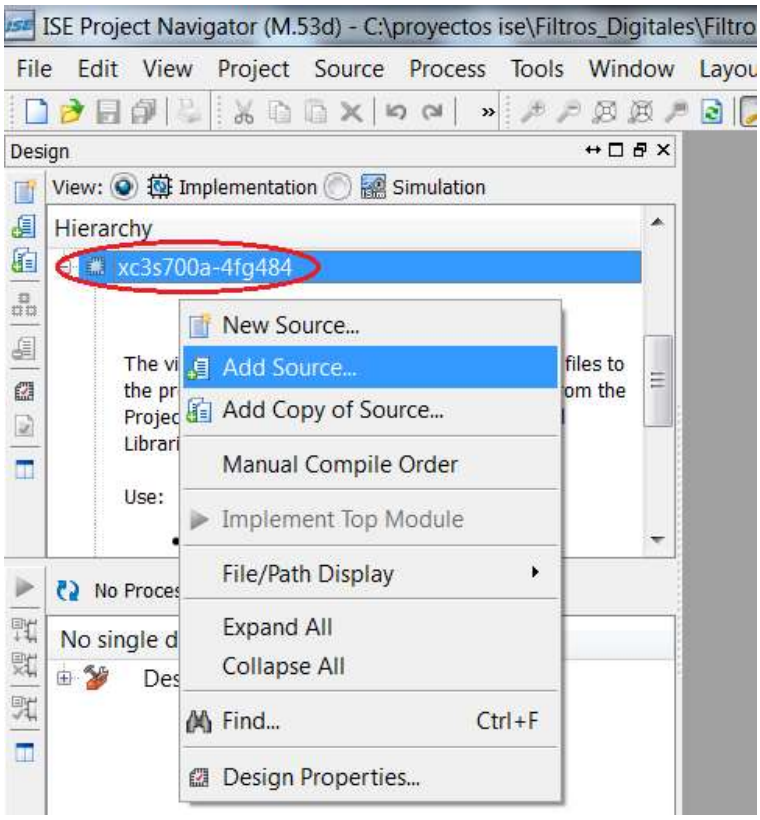


Ahora se procederá a adicionar los archivos fuente necesarios para la implementación filtros digitales. Los archivos que se adicionan fueron previamente creados y adecuados para su correcto funcionamiento. El archivo de máximo nivel para filtros FIR e IIR es llamado adc.vhd el cual tiene defino dentro de él un componente llamado Digital_FIlter.vhd. El archivo adc.vhd fue generado de una manera manual, siguiendo las instrucciones y restricciones impuestas en los *datasheets* de los conversores ADC y DAC que están integrados en la FPGA Spartan 3A. El archivo Digital_Filter.vhd fue generado automáticamente a partir de un modelo en Simulink con la ayuda de la herramienta *Simulink VHDL coder*.

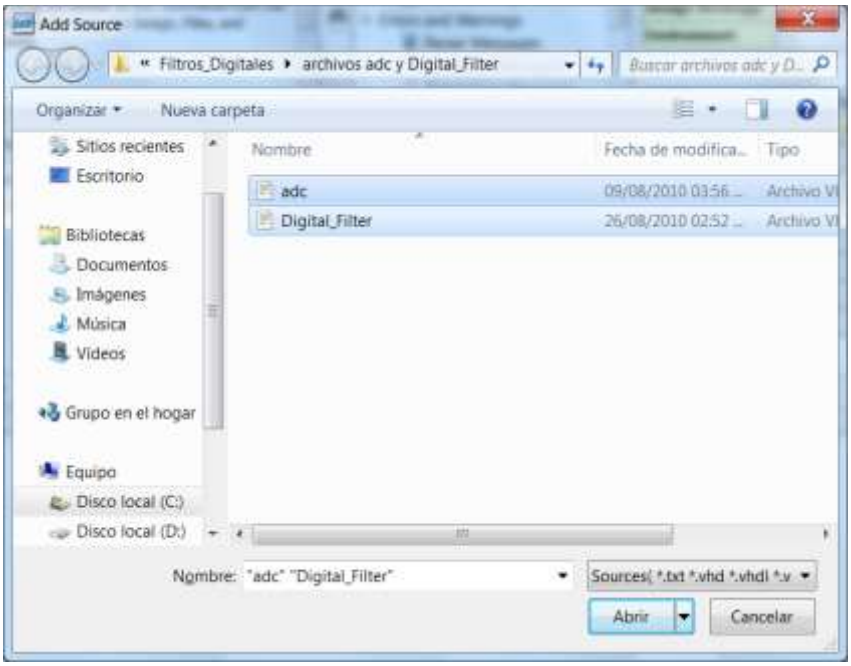
Nótese que el archivo adc.vhd es reutilizable, pues lo que se cambiaria seria únicamente el archivo Digital_Filter.vhd, para cada tipo de filtro FIR e IIR que se desee implementar. Por lo tanto al generar código a partir de cualquier filtro en Simulink se debe tener en cuenta que el archivo debe siempre llamarse Digital_Filter.vhd

Para agregar estos archivos realice lo mostrado en las siguientes figuras:

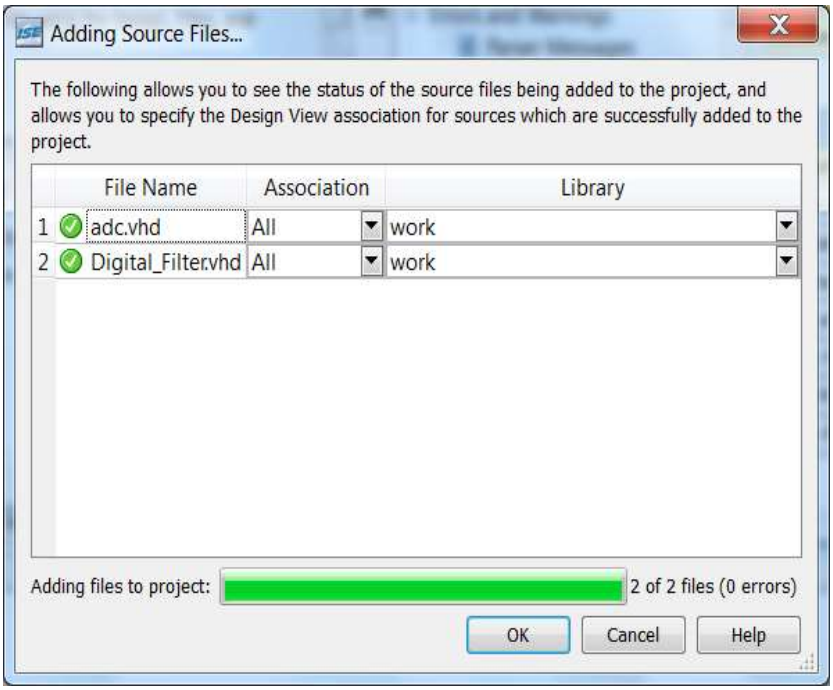
4. Dar click derecho donde se indica con el ovalo, para que se despliegue la ventana que esta sobrepuesta. En esta ventada seleccione *Add Source*



5. Busque la carpeta donde se encuentran los archivos adc.vhd y Digital_Filter.vhd. Seleccione ambos archivos y presione abrir:

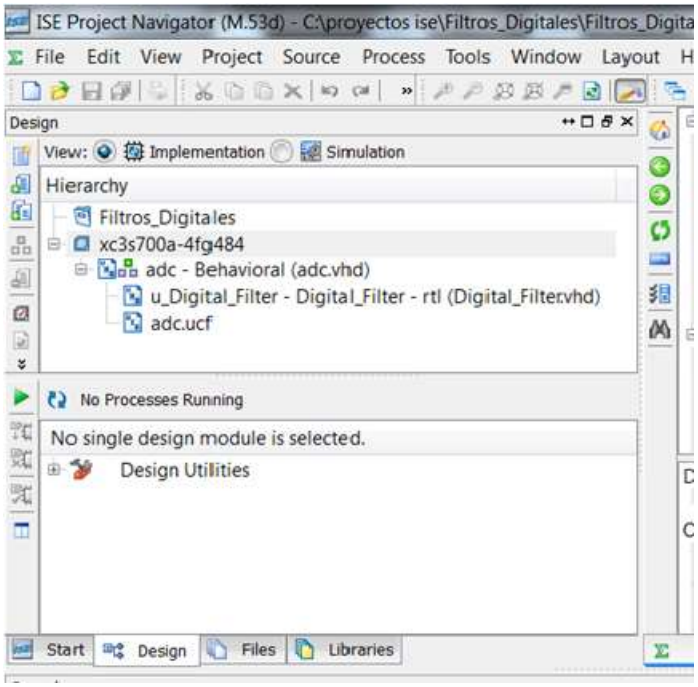


6. Se desplegará una ventana en donde se indica el estado de los archivos agregados.



7. Para este manual el archivo .ucf fue previamente elaborado y debe ser también agregado al proyecto de la misma forma como se agregaron los archivos anteriores. El archivo de restricciones de usuario UCF especifica la ubicación, la ejecución, el nombramiento, la dirección de señal y consideraciones para el análisis de tiempo y para la implementación del diseño.

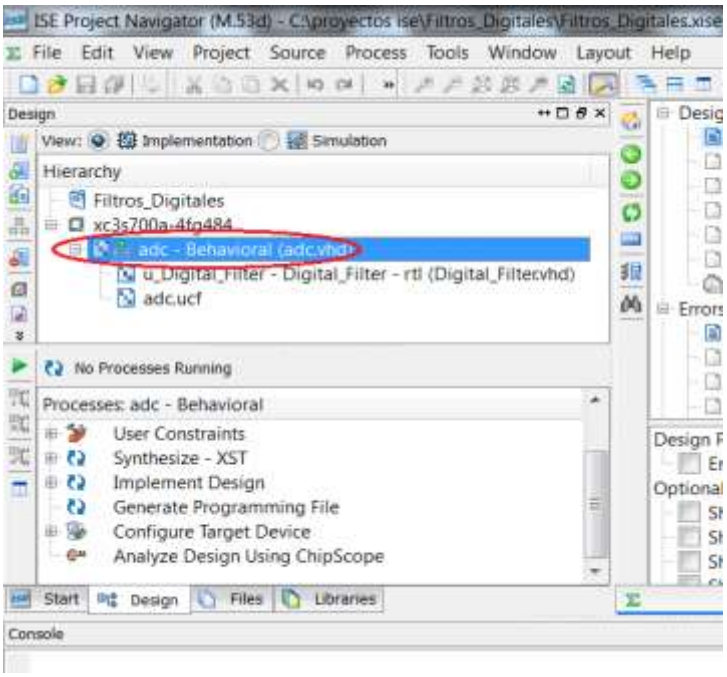
Después de agregar estos archivos la estructura jerárquica debe lucir como la de la siguiente figura:



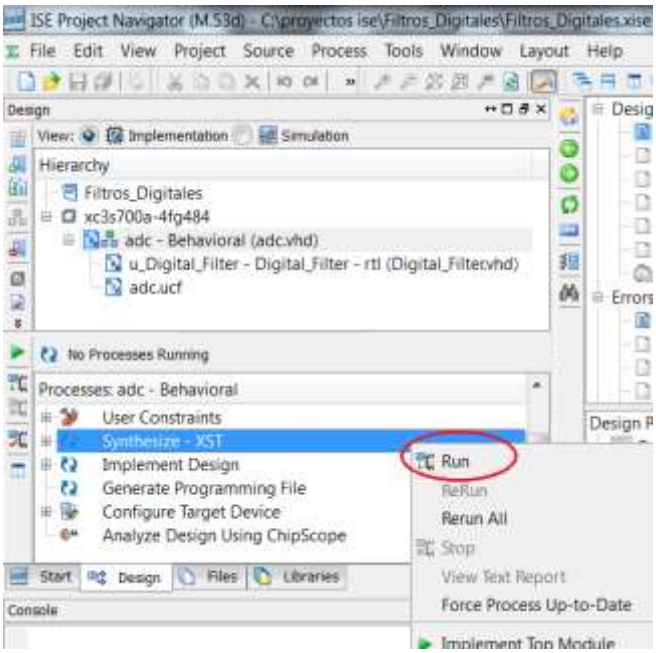
- Síntesis del proyecto.

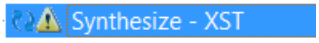
  En este paso realice lo mostrado en las siguientes figuras:

  8. Seleccionar el archivo adc.vhd, en la ventana de jerarquía de archivos



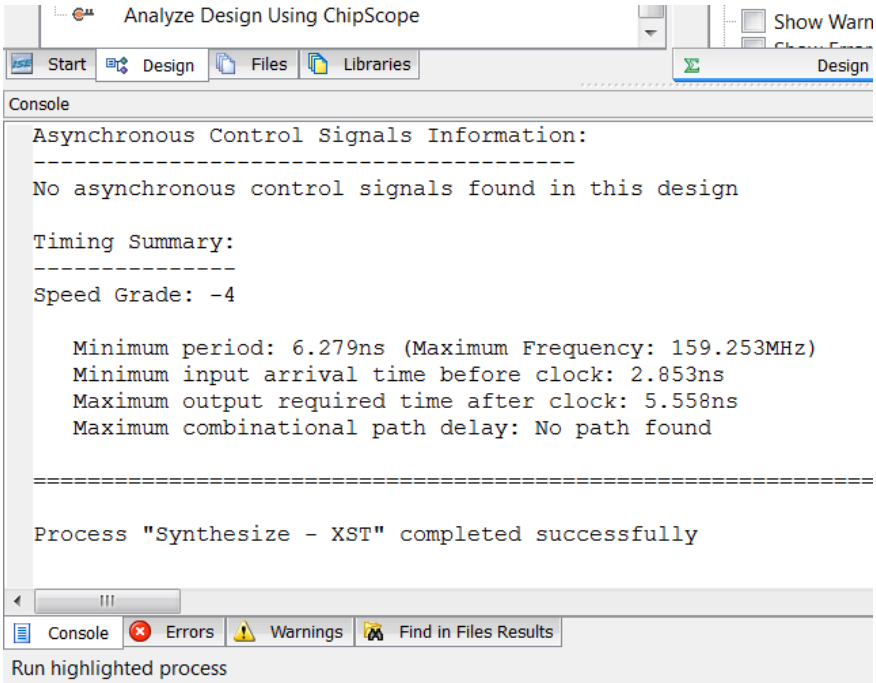
  9. Pulsar click derecho en el proceso *Synthesize-XST* en la ventana de procesos y luego dar click en *RUN*, como se muestra enseguida.



Al finalizar la ejecución aparecerá un indicador de alerta al lado del proceso *Synthesize-XST*, así: 

. Esto es debido a algunos *warnings* provenientes del archivo Digital_Filter.vhd

Para seguir más detalladamente la ejecución de este proceso mirar la Consola de la herramienta. Que al final lucirá así.



- Implementación del diseño

Una vez el proyecto ha sido sintetizado se puede proceder a la ejecución del proceso de *Implement Desing*, el cual contiene dentro de si los siguientes pasos:

I. *Traslate*

Fusiona el archivo de ruta de red *netlist* con las restricciones en un archivo de diseño de Xilinx.

II. *Map*

Ajusta el diseño a los recursos disponibles del dispositivo destino y, opcionalmente ubica el diseño.

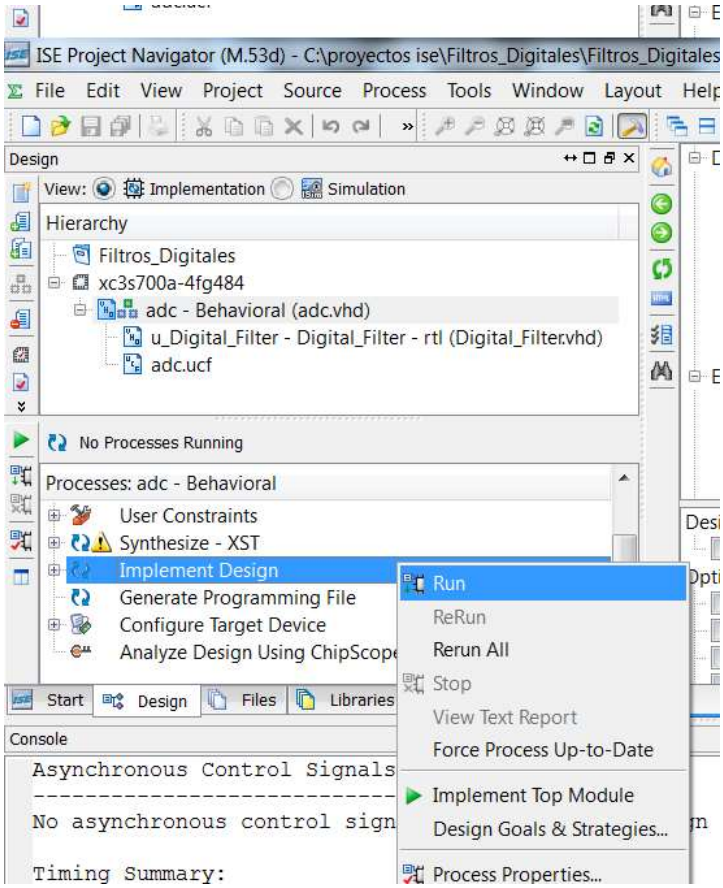
III. *Place and Route*

Ubica y encamina el diseño de acuerdo a las restricciones temporales.

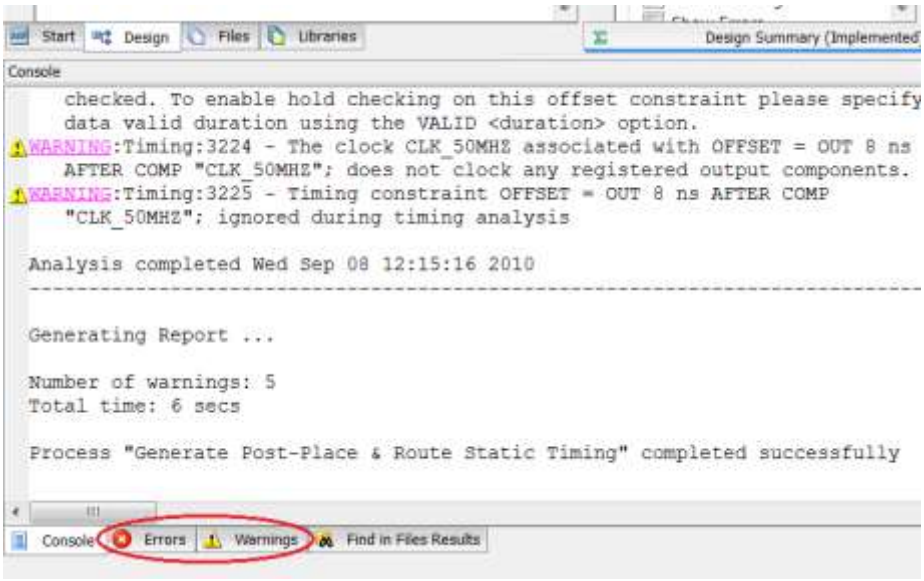
IV. *Generate Programming File*

Crea un archivo de secuencia de bits que se puede descargar al dispositivo destino.

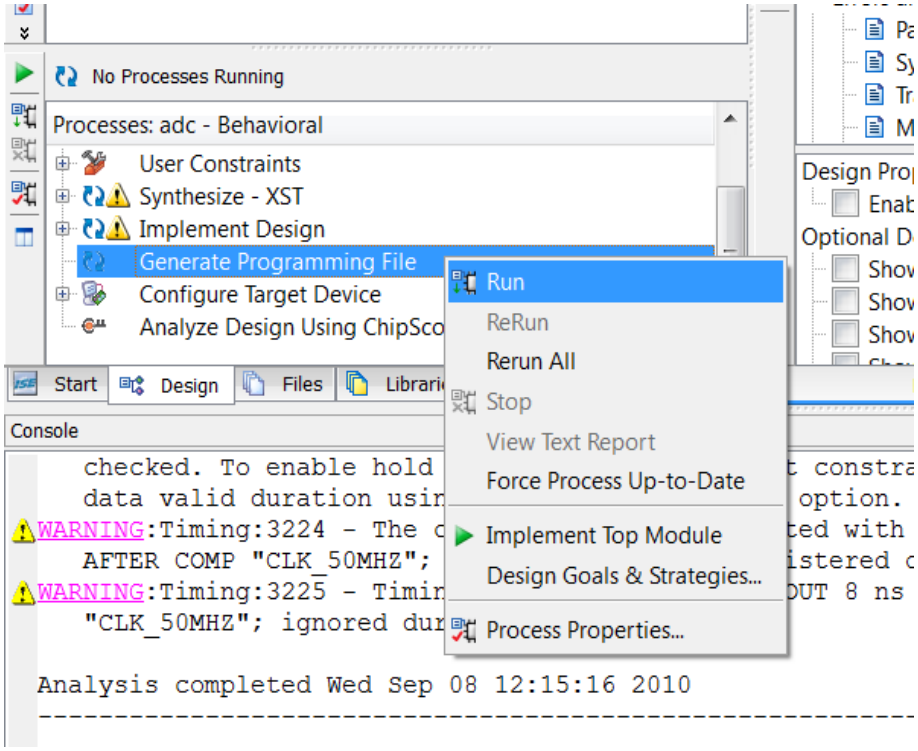
10. Para ejecutar este proceso es también necesario seleccionar el archivo adc, en la ventana de jerarquía de archivos, como se mostró anteriormente. Pulsar click derecho en el proceso *Implement Desing* en la ventana de procesos y luego dar click en RUN, como se muestra enseguida:



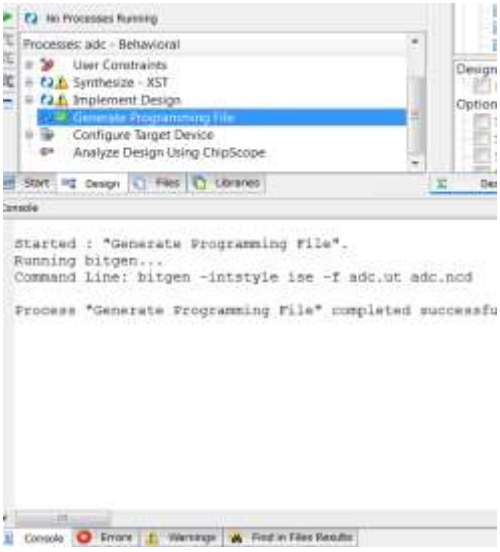
Es de importancia que en este paso se tenga en cuenta la ventana de warnings y errores, pues algunas veces el diseño a implementar necesita hacer uso de más del 100% de los recursos de la tarjeta, lo que es un inconveniente para seguir con el flujo de trabajo.



11. Después de la ejecución de este proceso se procede a la Generación del archivo de programa. Como se muestra a continuación:

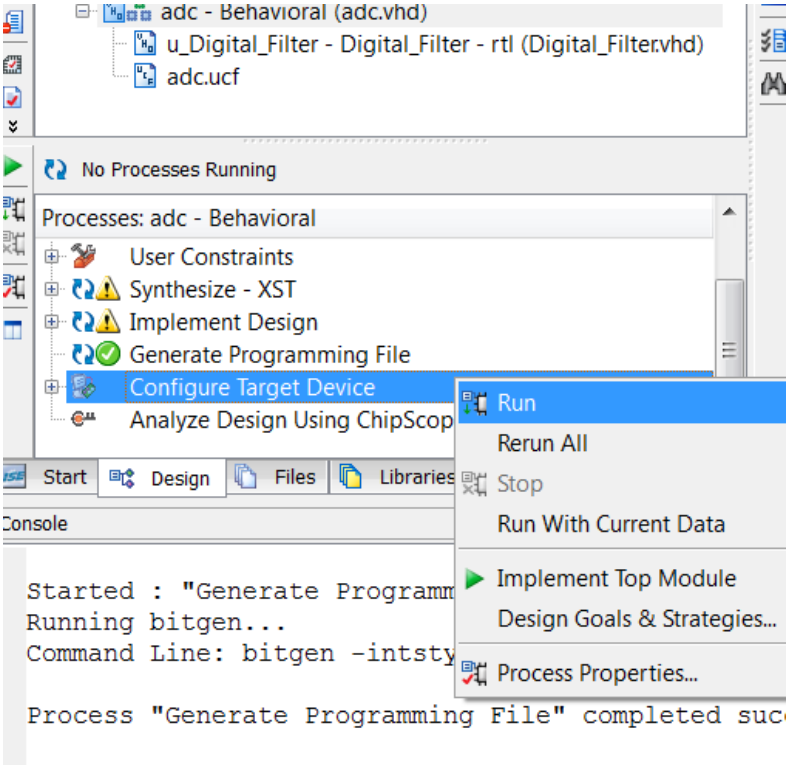


Al finalizar aparecerá:

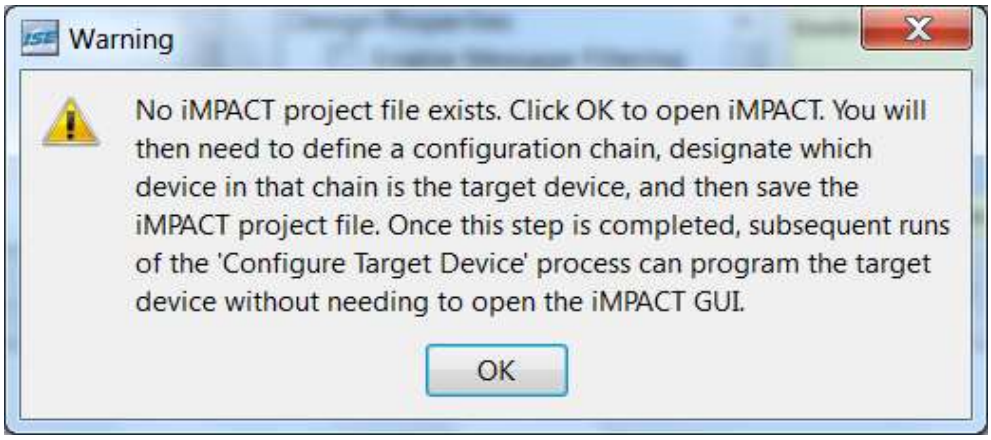


- Configuración y programación del dispositivo
  Después de la generación del archivo de programa, se puede configurar el dispositivo. Esto se puede realizar directamente desde un computador usando la herramienta iMPACT y el cable de descargas.

12. Siga lo mostrado en las siguientes figuras:

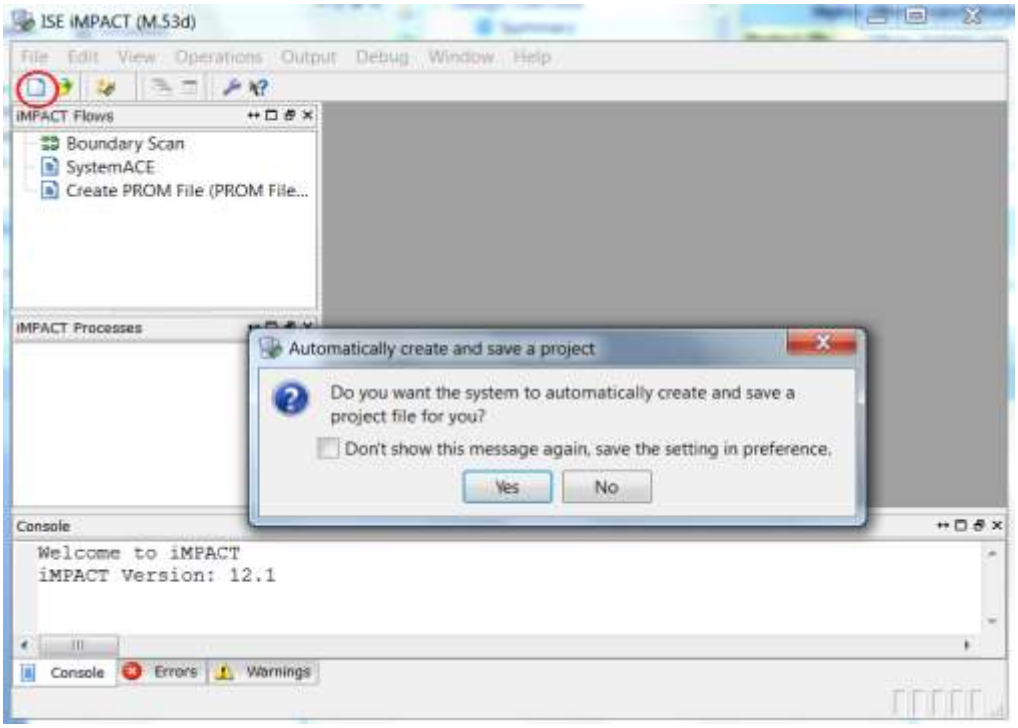


Dado que la herramienta iMPACT no ha sido usada, o no se han guardados los archivos generados con esta, es necesario presionar OK para que se abra la herramienta y poder crear un archivo de este tipo.



Aparecerá la siguiente interfaz, la cual representa a la herramienta iMPACT. Para crear el archivo nuevo seguir los siguientes pasos:

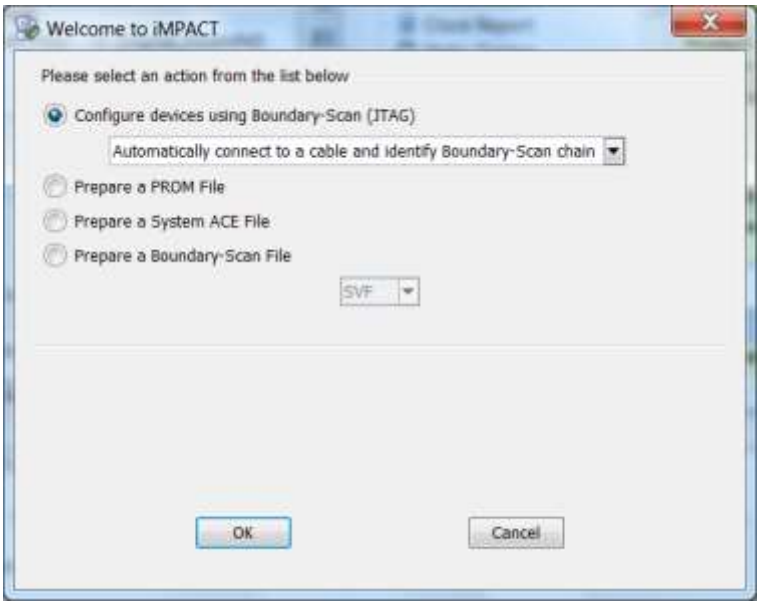
13. Presione click en donde se indica con el círculo de color rojo. Luego aparecerá la opción de autoguardado, se debe pulsar *Yes*.



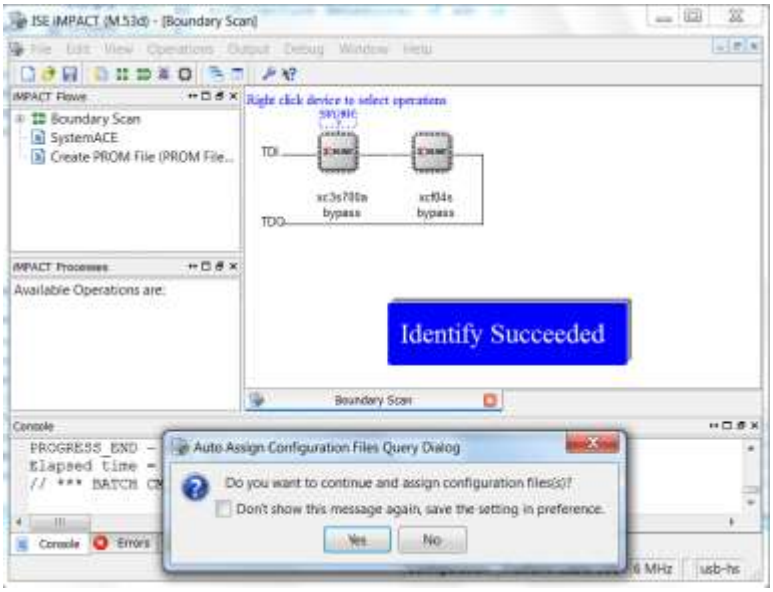
Al llegar a este paso asegúrese que el USB esté conectado tanto a la tarjeta como al computador y además que la tarjeta este conectada a la alimentación y encendida, de

lo contrario se genera una ventana que detiene la ejecución del proceso y le sugiere revisar la configuración del cable.

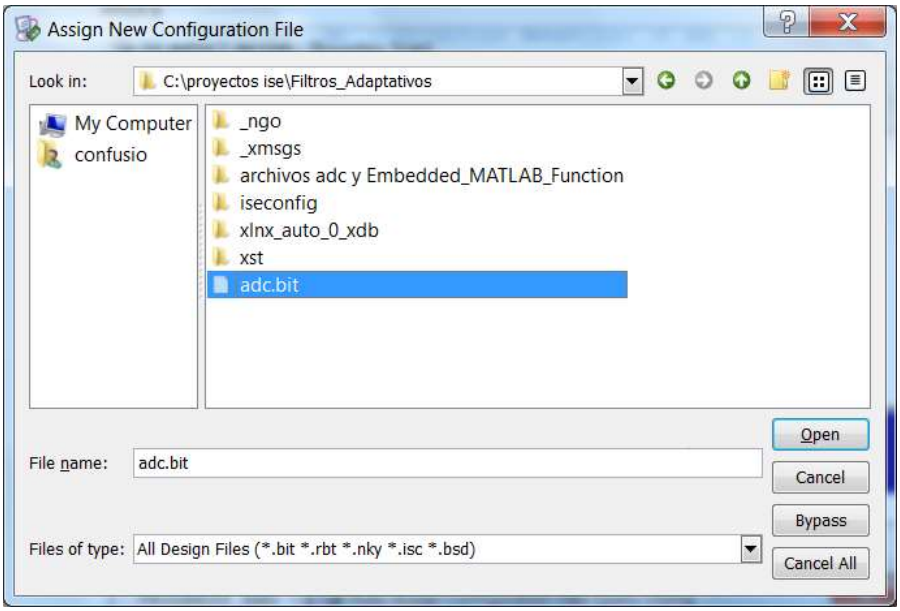
Aparece la siguiente ventana en donde se selecciona que la acción *Configure devices using Boundary-Scan* realice las operaciones automáticamente. Se pulsa *OK*



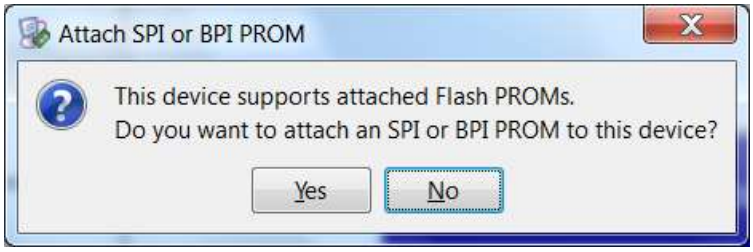
14. Ahora se desplegara la siguiente ventana que indicara que los dispositivos programables xc3s700a y xcf04s han sido identificados, y aparece la opción de seleccionar un archivo de configuración para que sea programado en uno de estos dispositivos. Se pulsa *Yes.*



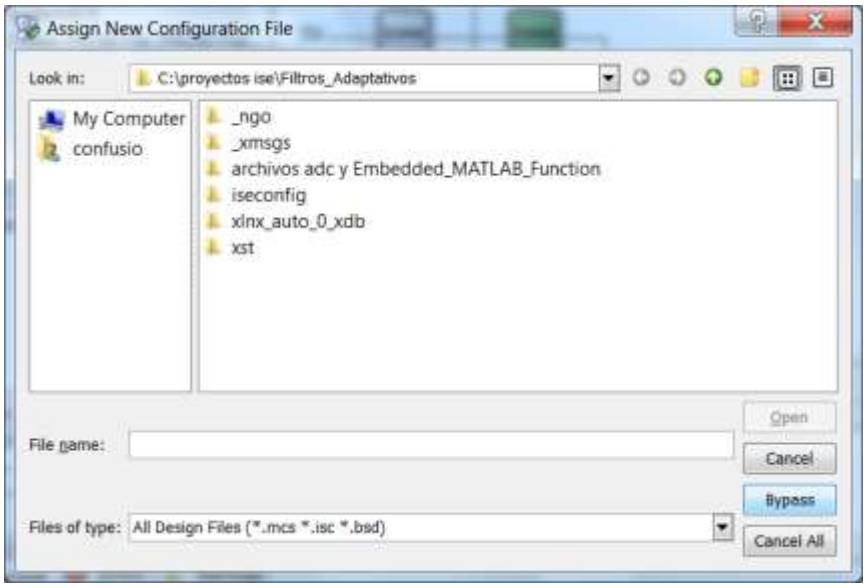
15. Asignación de archivo de configuración para el dispositivo xc3s700a: Se debe seleccionar el archivo .bit que ha sido generado dentro de la carpeta del proyecto actual. Se selecciona y se pulsa *open*



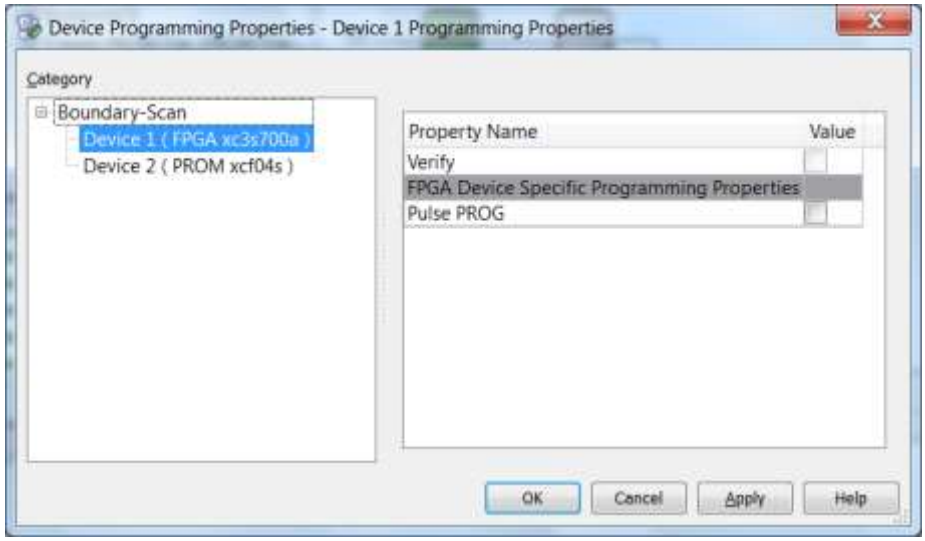
16. Ahora se pregunta que si se desea configurar el dispositivo xcf04s. Se debe pulsar *No*.



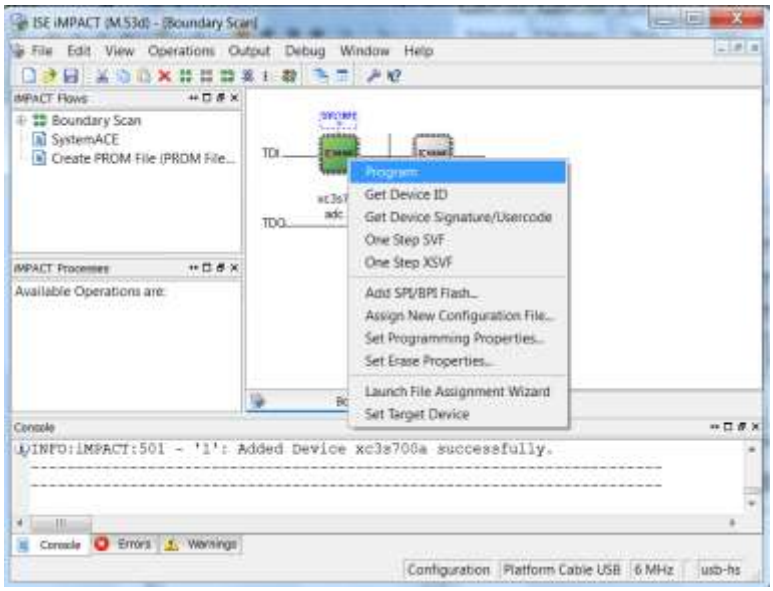
Se desplegara la siguiente ventana, en la cual se pulsara *Bypass:*



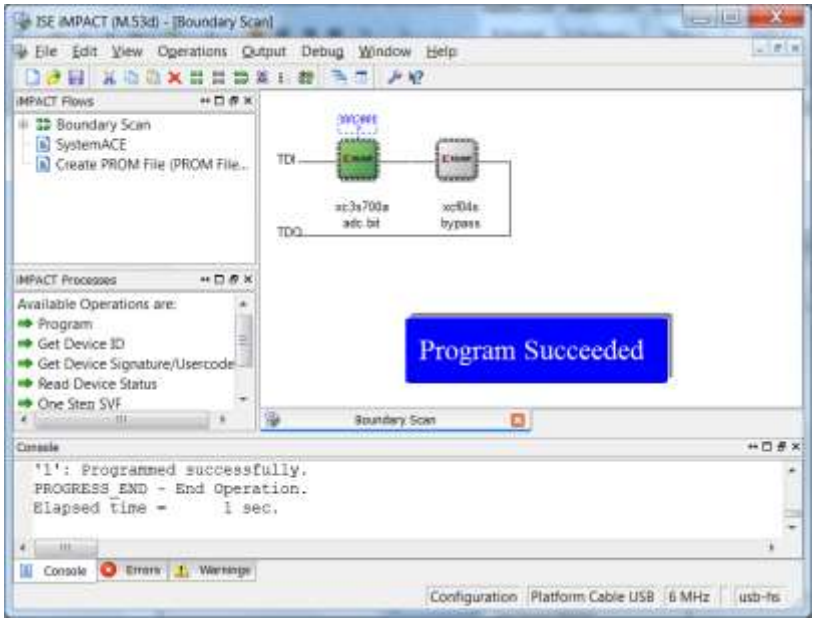
17. Ahora se pide configurar las propiedades de la programación de los dispositivos, se deja todo por defecto y se pulsa *OK.* Opcionalmente se puede seleccionar *Pulse PROG* para que un programa de instrucciones sea enviado al dispositivo y borre la configuración de memoria del dispositivo previamente a iniciar la secuencia configuración.



Ahora se debe dar *click derecho* encima del diagrama que identifica al dispositivo xc3s700a y posteriormente pulsar *Program* como se ve en la siguiente figura:



Al finalizar debe aparecer una imagen como la siguiente, la cual nos indica el estado de la programación. En este caso indica que la programación ha sido satisfactoria:

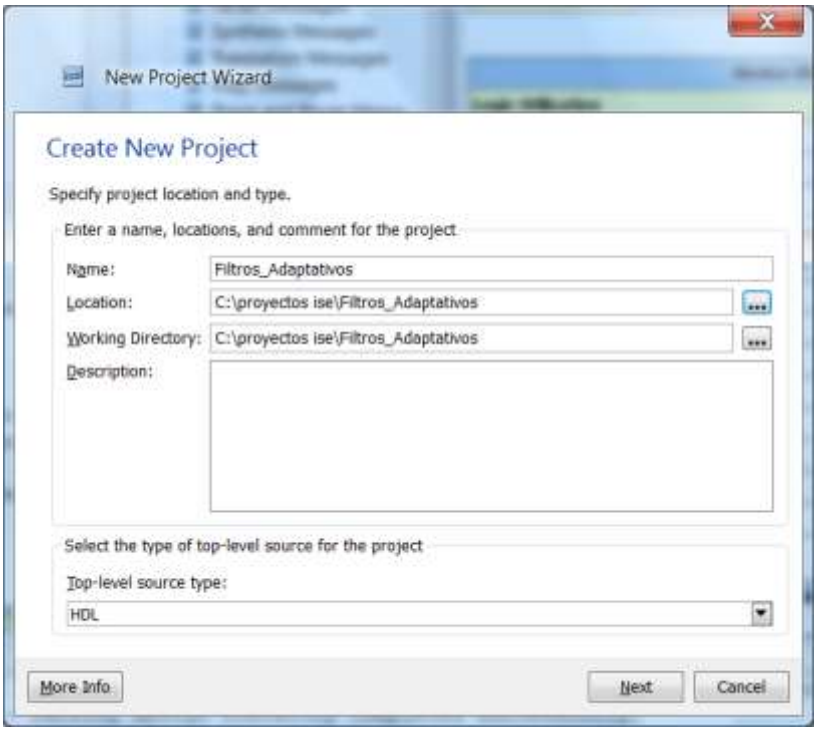


- ✓ **Implementación de filtros adaptativos**
  - Creación de un proyecto

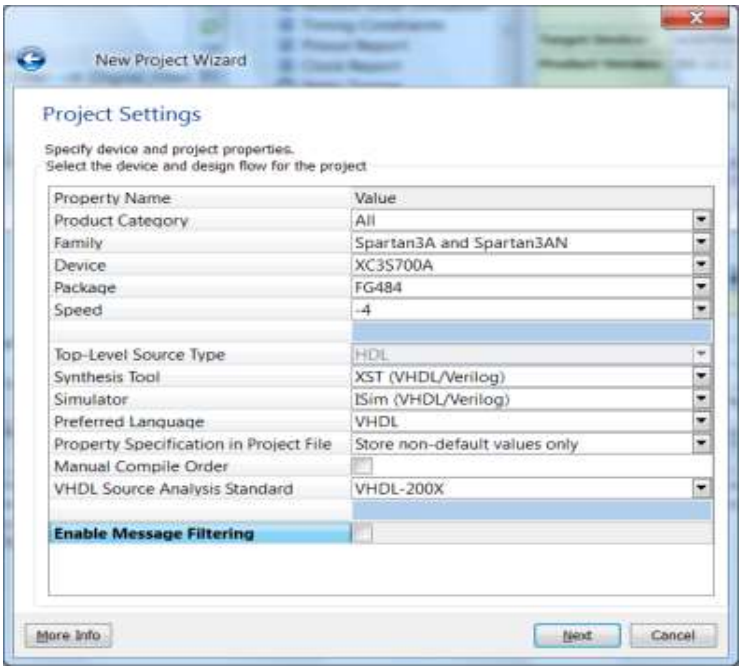
    Primero se debe crear un nuevo proyecto para luego agregar los archivos fuente y establecer las propiedades de los procesos. Ahora siga los pasos especificados por las siguientes figuras:

1. Ejecute el *ISE Proyec Navigator*  y Seleccione *File ->New Project*

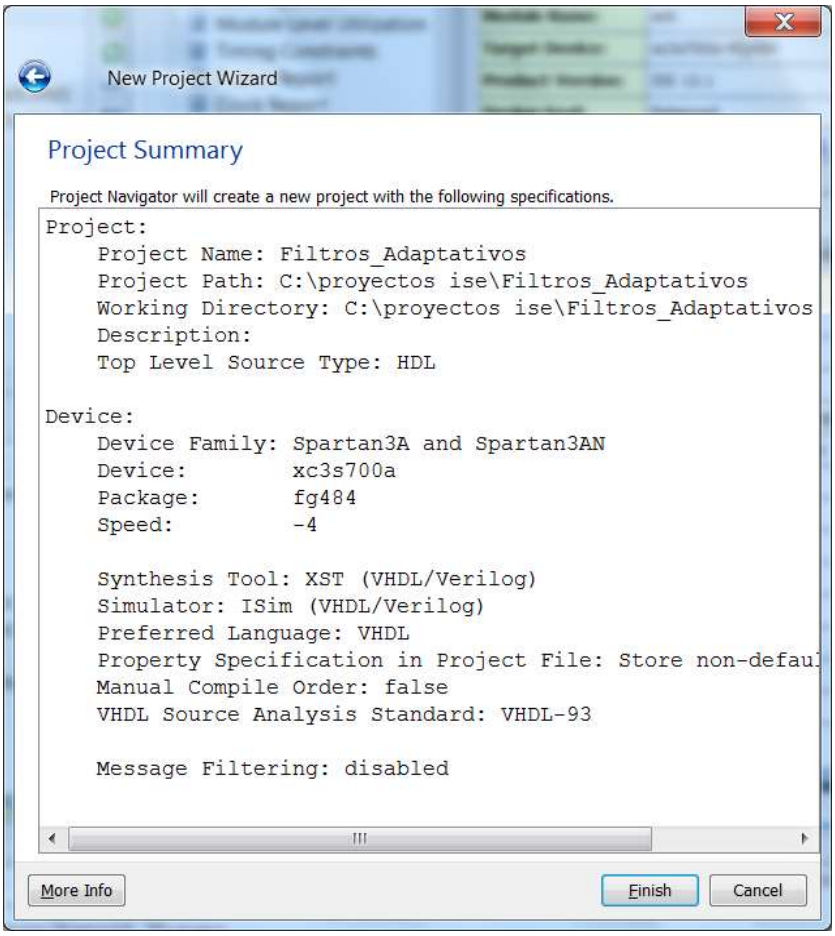
2. Proporcione el nombre, la ubicación y el directorio de trabajo que desea para el proyecto. Además se pueden agregar algunos comentarios. El tipo de fuente de máximo nivel con el que se trabajara en este manual es HDL.



3. Especifique el dispositivo y las propiedades del proyecto.



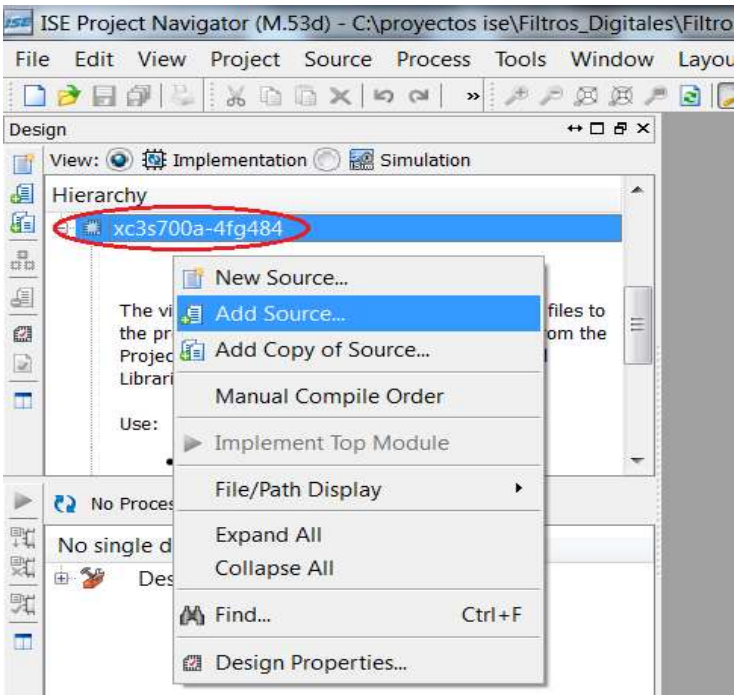
Después de realizar los pasos anteriores aparecerá la siguiente ventana. Que indica las especificaciones del proyecto a crear. Para finalizar la creación presionar *Finish*.



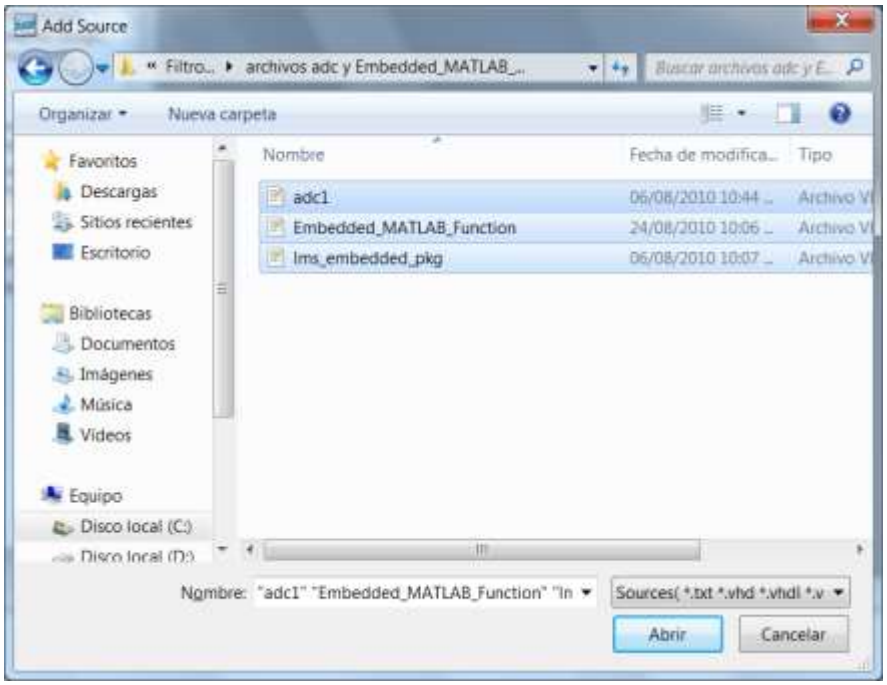
Ahora se procederá a adicionar los archivos fuente necesarios para la implementación de filtros adaptativos. Los archivos que se adicionan fueron previamente creados y adecuados para su correcto funcionamiento. El archivo de máximo nivel es llamado adc1.vhd el cual tiene defino dentro de él un componente llamado Embedded_MATLAB_Function.vhd que a su vez trae definido dentro de sí el archivo lms_embedded_pkg.vhd. El archivo adc1.vhd fue generado de una manera manual, siguiendo las instrucciones y restricciones impuestas en los *datasheets* de los conversores ADC y DAC que están integrados en la FPGA Spartan 3A. Los archivos Embedded_MATLAB_Function.vhd y lms_embedded_pkg.vhd.vhd fueron generados automáticamente a partir de un modelo en Simulink con la ayuda de la herramienta *Simulink VHDL coder*.

Para agregar estos archivos realice lo mostrado en las siguientes figuras:

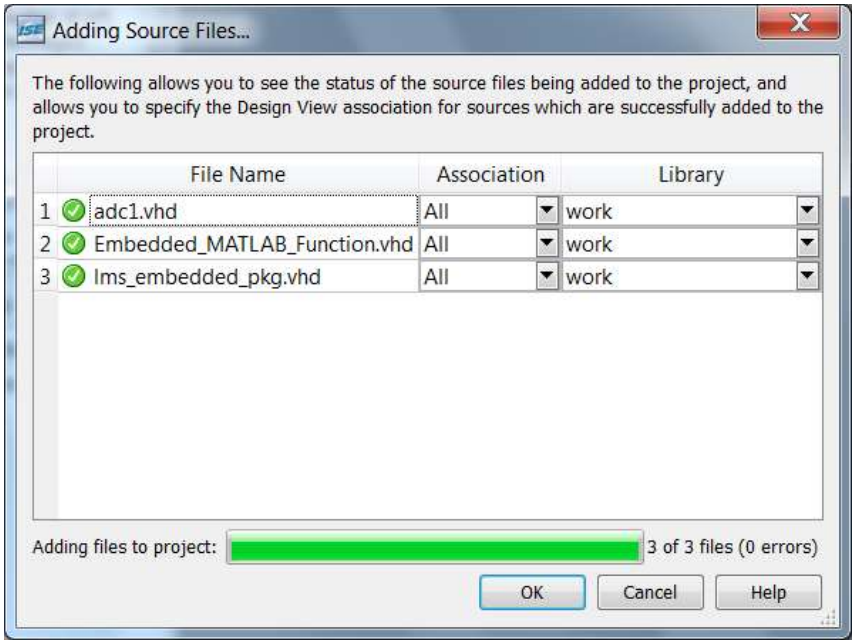
4. Dar click derecho donde se indica con el ovalo, para que se despliegue la ventana que esta sobrepuesta. En esta ventada seleccione *Add Source:*



5. Busque la carpeta donde se encuentran los archivos adc1.vhd, Embedded_MATLAB_Function.vhd y lms_embedded_pkg.vhd.vhd. Se seleccionan los archivos y se presiona abrir:

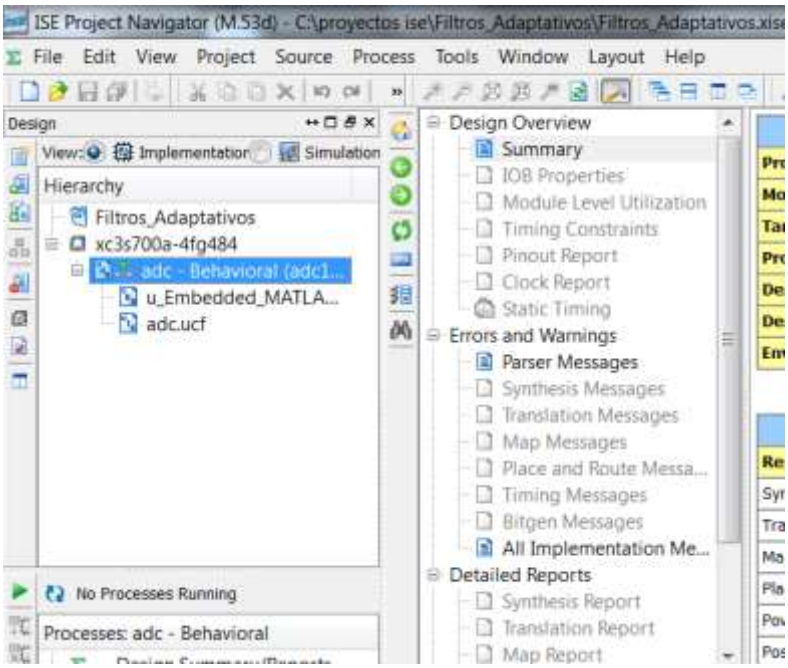


6. Se desplegará una ventana en donde se indica el estado de los archivos agregados:



7. Para este manual el archivo .ucf fue previamente elaborado y debe ser también agregado al proyecto de la misma forma como se agregaron los archivos anteriores.

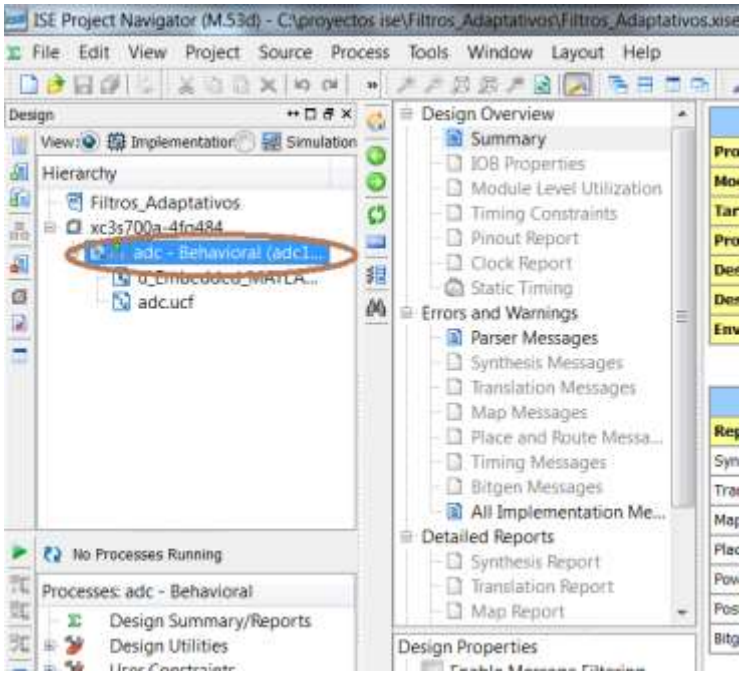
Después de agregar estos archivos la estructura jerárquica debe lucir como la de la siguiente figura:



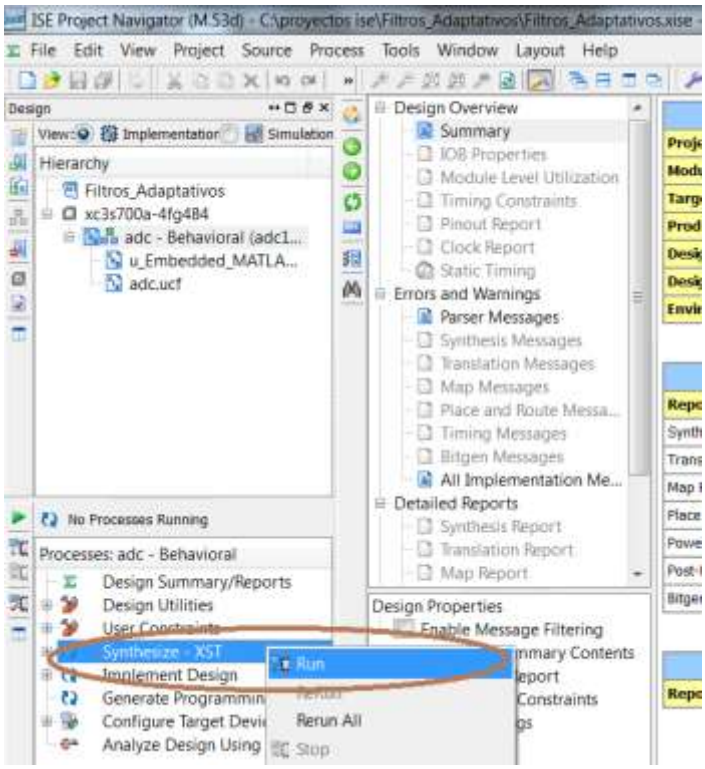
- Síntesis del proyecto.

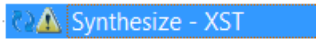
  En este paso realice lo mostrado en las siguientes figuras:

  8. Seleccionar el archivo adc1.vhd, en la ventana de jerarquía de archivos:



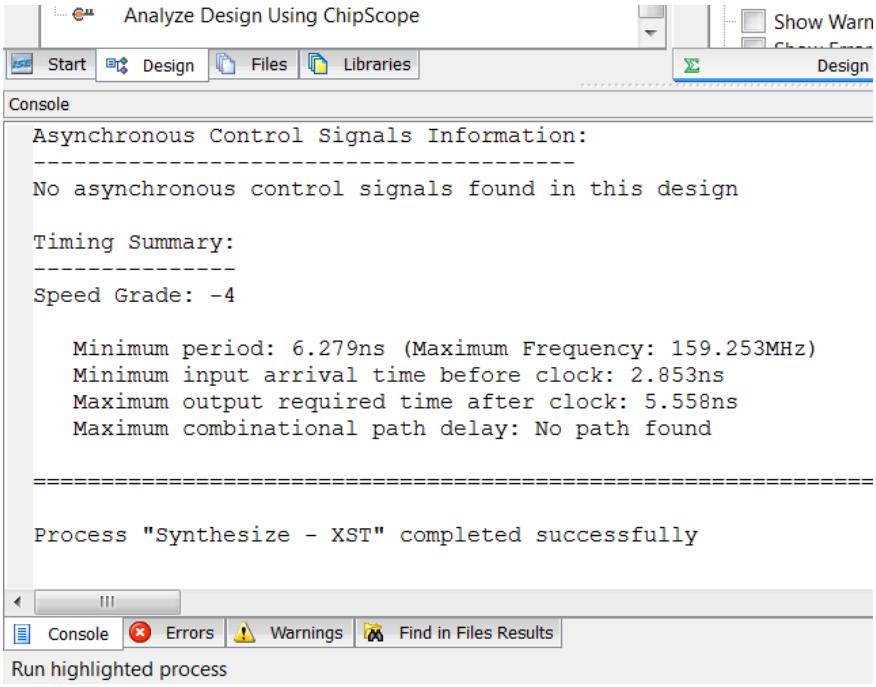
  9. Pulsar click derecho en el proceso *Synthesize-XST* en la ventana de procesos y luego dar click en *RUN*, como se muestra enseguida:



Al finalizar la ejecución aparecerá un indicador de alerta al lado del proceso *Synthesize-XST*, así: 

. Esto es debido a algunos *warnings* provenientes del archivo Embedded_MATLAB_Function.vhd

Para seguir más detalladamente la ejecución de este proceso mirar la Consola de la herramienta. Que al final lucirá así.



- Implementación del diseño:

  Una vez el proyecto ha sido sintetizado se puede proceder a la ejecución del proceso de *Implement Desing*, el cual contiene dentro de si los siguientes pasos:

  I. *Traslate*

  Fusiona el archivo de ruta de red *netlist* con las restricciones en un archivo de diseño de Xilinx.

  II. *Map*

  Ajusta el diseño a los recursos disponibles del dispositivo destino y, opcionalmente ubica el diseño.

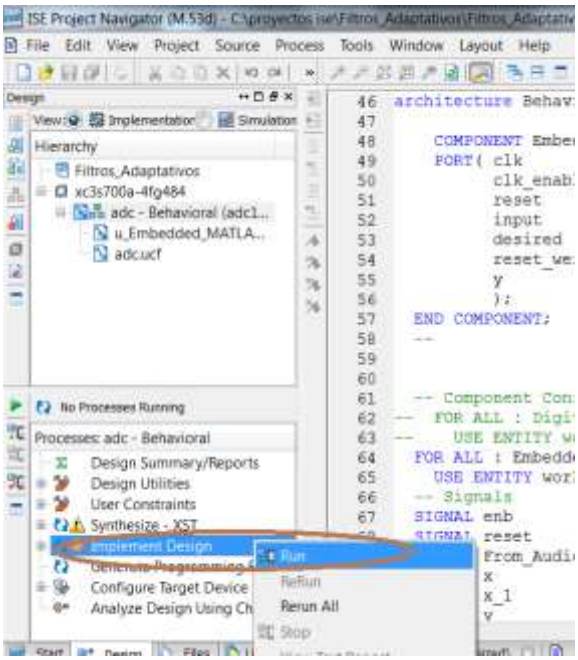
  III. *Place and Route*

  Ubica y encamina el diseño de acuerdo a las restricciones temporales.

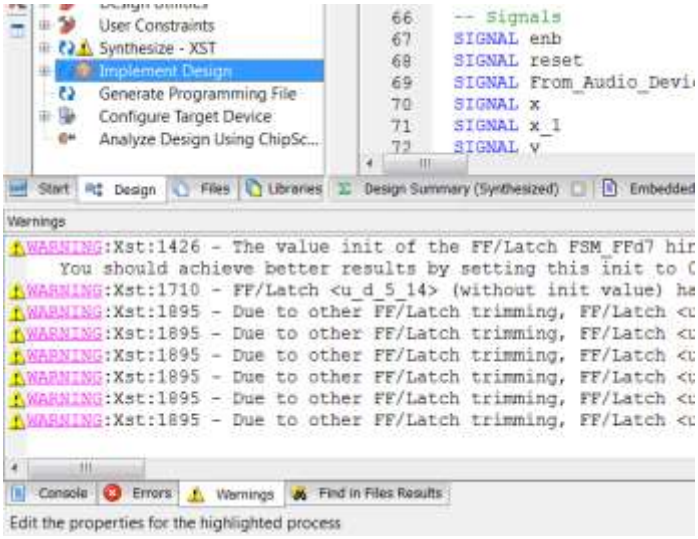
  IV. *Generate Programming File*

  Crea un archivo de secuencia de bits que se puede descargar al dispositivo destino.

10. Para ejecutar este proceso es también necesario seleccionar el archivo adc1.vhd, en la ventana de jerarquía de archivos, como se mostró anteriormente. Pulsar click derecho en el proceso *Implement Desing* en la ventana de procesos y luego dar click en RUN, como se muestra enseguida:



Es de importancia que en este paso se tenga en cuenta la ventana de warnings y errores, pues algunas veces el diseño a implementar necesita hacer uso de más del 100% de los recursos de la tarjeta, lo que es un inconveniente para seguir con el flujo de trabajo.



11. Siga los pasos 11, 12, 13, 14,15, 16 y 17 expuestos en el apartado **Implementación de filtros selectivos en frecuencia FIR e IIR**

PRACTICA No 1. SIMULACION DE FILTROS DIGITALES TIPO FIR POR EL METODO DE VENTANA KAISER Y OTRO METODO DE VENTANA.

1. OBJETIVOS

- Diseñar Filtros FIR con el método de ventana Káiser y otro método de ventana, en Matlab.
- Simular los filtros FIR diseñados con el método de ventana Káiser y otro método de ventana en Matlab y Simulink

2. INTRODUCCION

En esta práctica se presentan los conceptos básicos para el diseño de filtros digitales tipo FIR usando el método de ventanas, además se muestra en un ejemplo práctico, la utilización de estas técnicas de ventaneo en el diseño y la simulación de filtros en la herramienta Matlab. El proceso de laboratorio desarrollado por el estudiante tiene como guía el ejemplo práctico enunciado anteriormente, pero los requisitos de diseño y algunos otros parámetros serán definidos por el profesor.

3. MARCO TEORICO

Un filtro es un sistema continuo o discreto que modifica el espectro de una señal de entrada de acuerdo a ciertas especificaciones para producir una señal de salida. Para sistemas lineales e invariantes en el tiempo el espectro de la señal salida es igual al espectro de la señal de entrada multiplicado por la respuesta en frecuencia del sistema.

Los filtros pueden clasificarse como analógico y digitales. Un filtro digital se puede definir como un proceso computacional implementado con circuitos y/o programación, en el cual una secuencia numérica de entrada se transforma en otra secuencia numérica de salida con características predeterminadas. Matemáticamente, un filtro digital se representa por una ecuación de diferencias, y para su implementación se hace uso de sumadores y multiplicadores binarios y bloques de atraso [1].

Existen dos tipos de filtros digitales, IIR (respuesta infinita al impulso) y FIR (respuesta finita al impulso). En esta práctica abordaremos solamente los filtros tipo FIR.

Las diferentes configuraciones que se pueden realizar de filtros digitales FIR o IIR son mostradas en la figura 1:

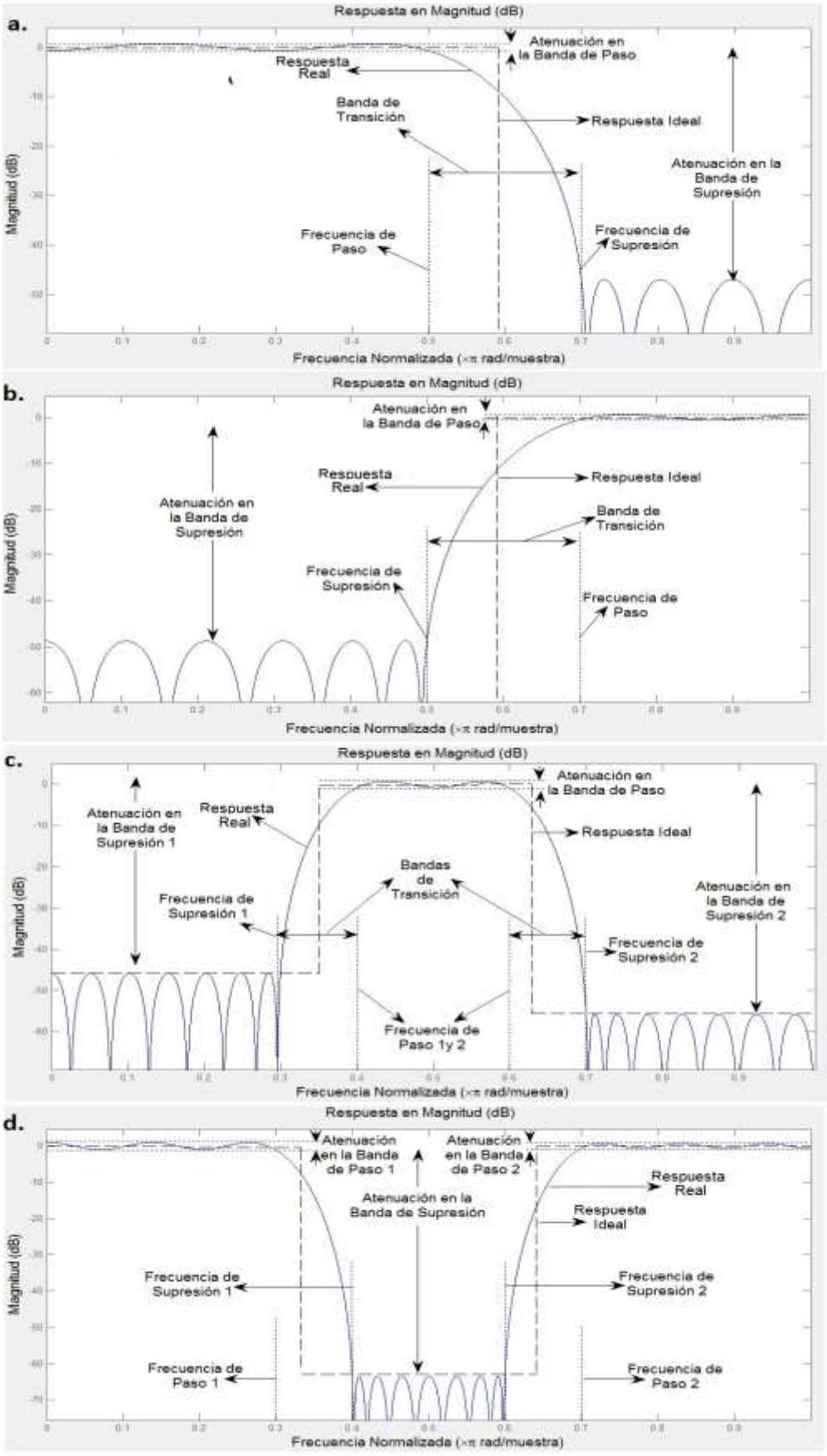


Figura 1. Configuración: a. Pasa Bajo, b. Pasa Alto, c. Pasa banda, d. Elimina banda.

Filtros FIR

Un filtro FIR con longitud M, cuyo orden es $M-1=N$, con entrada $x(n)$ y salida $y(n)$ se describe por la ecuación de diferencias:

$$y(n) = \sum_{K=0}^{N} b_k\, x(n-k) \tag{1}$$

Donde $\{b_k\}$ son los coeficientes del filtro.

Es posible representar la secuencia de salida como la convolución de la respuesta al impulso del sistema $h(n)$ con la señal de entrada. Entonces se tiene:

$$y(n) = \sum_{K=0}^{N} h(k)x(n-k) \tag{2}$$

Donde los limites superior e inferior de la sumatoria reflejan las características de causalidad y duración finita del filtro. La ecuación 1 y la ecuación 2 son iguales en su forma, por lo tanto, se puede decir que $b_k = h(k),\ k = 0,1,2,\ldots\ldots,N$ [2].

La transformada Z de la respuesta al impulso $h(n)$ produce la función transformada $H(z)$, la cual tiene, en el caso de los filtros causales FIR, N ceros cuya posición está determinada por los coeficientes b_k y N polos en el origen. Entonces, un filtro FIR es siempre estable. Debido al hecho de que la trasformada de la función de un filtro es exclusivamente determinada por la posición de los ceros, los filtros FIR son llamados filtros "todo-ceros" [3].

$$H(z) = \sum_{k=0}^{N} h(k)z^{-k} \tag{3}$$

Diseño de filtros FIR usando ventanas

La respuesta al impulso $h_d(n)$ para un filtro ideal es de duración infinita y debe ser truncada en algún punto para su manejo en sistemas reales. Para producir un filtro FIR de orden N es necesario truncarla en $n=N$. Este truncamiento es equivalente a la multiplicación de la respuesta al impulso ideal del filtro $h_d(n)$ con una ventana rectangular, definida como [2]:

$$w(n) = \begin{cases} 1, & n = 0,1,2,\ldots,N \\ 0, & otro\ caso \end{cases} \tag{4}$$

Por lo tanto la respuesta al impulso de un filtro FIR se convierte en:

$$h(n) = h_d(n)w(n) \tag{5}$$

$$h(n) = \begin{cases} h_d(n), & n = 0,1,\ldots,N \\ 0, & otro\ caso \end{cases} \tag{6}$$

Este truncamiento afecta la respuesta del filtro, apareciendo oscilaciones en los bordes de las bandas de paso y supresión, además de una disminución en la pendiente de caída del filtro. Estas características pueden alterarse utilizando otro tipo de ventanas [4]. Las ecuaciones de diseño de estas ventanas se muestran en la tabla 1 y en la tabla 2 se muestran las características más relevantes de cada una de ellas.

Ventana	Secuencia en el dominio temporal $n = 0,1,\dots,N$	Función en Matlab
Barlett	$1 - \frac{2\left\lvert n - \frac{N}{2}\right\rvert}{N-1}$	W=BARTLETT(N)
Hanning	$\frac{1}{2}\left(1 - cos\left(\frac{2\pi n}{N}\right)\right)$	W=HANN(N)
Hamming	$0.54 - 0.46\cos\frac{2\pi n}{N}$	W=HAMMING(N)
Blackman	$0.42 - 0.5cos\left(\frac{2\pi n}{N}\right) + 0.08\left(\frac{4\pi n}{N}\right)$	W=BLACKMAN(N)
Káiser	$\frac{I_0\left(\beta\sqrt{1-\left(1-\frac{2n}{N}\right)^2}\right)}{I_0(\beta)}$	W = KAISER(N)

Tabla 1. Ecuaciones de diseño de ventanas.

Ventana	Ancho de transición aproximado del lóbulo principal	Pico de lóbulos laterales (dB)	Atenuación Mínima de la banda de supresión (dB)
Rectangular	$4\pi/M$	-13	-21
Barlett	$8\pi/M$	-27	-25
Hanning	$8\pi/M$	-32	-44
Hamming	$8\pi/M$	-43	-53
Blackman	$12\pi/M$	-58	-74

Tabla 2. Características relevantes de las ventanas existentes

A continuación se presenta un caso específico de diseño con técnicas de ventana, en este caso diseño con ventana Hamming:

Método de ventana Hamming.

La ventana Hamming es construida adicionando un periodo de una función coseno a la ventana rectangular. El beneficio de adicionar los segmentos de coseno es la reducción de los picos de los lóbulos laterales. El precio por este beneficio es que el lóbulo principal se doble en anchura [5]. Por lo tanto, la ventana Hamming comparada con la ventana rectangular disminuye el rizado en los bordes de ambas bandas, de paso y supresión, sin embargo se tiene un mayor ancho de banda de transición [6].

Para aplicar en determinado diseño, la técnica de ventana Hamming se debe tener en cuenta que la atenuación esperada en la banda de supresión (especificaciones de diseño), pueda ser cumplida por esta ventana, guiándose por la tabla 2.

El orden del filtro para el método de ventana Hamming se calculo como:

$$N = \frac{8\pi}{\Delta\omega} - 1 \tag{7}$$

Donde $\Delta\omega$ que es el ancho de transición se define como:

$$\Delta\omega = \left| \omega_p - \omega_s \right| \tag{8}$$

ω_p es la frecuencia de paso y ω_s es la frecuencia de supresión.

La siguiente tabla muestra las ecuaciones necesarias para diseñar un filtro selectivo en frecuencia por el método de ventana Hamming. En la parte derecha se muestra una posible forma de escribir las formulas en Matlab:

Proceso	Ecuaciones de Diseño	Código Matlab
Cálculo de las frecuencias de paso y supresión normalizadas ω_p, ω_s: Con las especificaciones dadas de *fs* (frecuencia de supresión en Hz), *fp* (frecuencia de paso en Hz) y *fm* (frecuencia de muestreo en Hz), se calcula:	Frecuencia de supresión/paso normalizada: $F_{s,p} = \frac{f_{s,p}}{f_m}$ $\omega_{s,p} = 2\pi F_{s,p}$	`Fp=fp/fm;` `Wp=2*pi*Fp;`
Cálculo de la(s) frecuencia(s) de corte F_c :	Filtro Pasa bajo y Pasa alto: $\omega_c = \frac{\omega_p + \omega_s}{2}$ $F_c = \frac{\omega_c}{2*pi}$	`Wc=(Wp+Ws)/2;` `Fc=Wc/(2*pi);`
	Filtro Elimina banda y Pasa banda: $\omega_{c1} = \frac{\omega_{p1} + \omega_{s1}}{2}$, $\omega_{c2} = \frac{\omega_{p2} + \omega_{s2}}{2}$ $F_{c1} = \frac{\omega_{c1}}{2*pi}$, $F_{c2} = \frac{\omega_{c2}}{2*pi}$	`Wc1=((Ws1+Wp1)/2);` `Wc2=((Wp2+Ws2)/2);` `Fc1= Wc1/(2*pi);` `Fc2= Wc2/(2*pi);`
Cálculo del ancho de banda de transición $\Delta\omega$:	Filtro Pasa bajo y Pasa alto: $\Delta\omega = \lvert\omega_s - \omega_p\rvert$	`BW=abs(Ws-Wp);`
	Filtro Elimina banda y Pasa banda: $\Delta\omega_1 = \lvert\omega_{s1} - \omega_{p1}\rvert$ $\Delta\omega_2 = \lvert\omega_{p2} - \omega_{s2}\rvert$ Se escoge el menor entre ambos	`BW1=abs(Ws1-Wp1);` `BW2=abs(Wp2-Ws2);` `BW=min(BW1, BW2);`
Cálculo del orden del filtro N :	$N = \frac{k\pi}{\Delta\omega} - 1$, donde $k = 8$ para ventana Hamming (ver tabla 2)	`N= ceil(((8*pi)/BW)-1);` `if rem(N,2)~=0` `  N=N+1;` `end` *La función* `ceil` *redondea el resultado al entero más cercano en dirección de los infinitos positivos.* *La función* `rem` *devuelve el residuo de la división de* `N` *entre* `2`*. La rutina* `if` *al ser el*

		residuo diferente de cero le suma 1 a N, para que sea par y cumplir con las condiciones de filtro tipo I.
Cálculo de la respuesta al impulso unitario del filtro $h_1(n)$:	Filtro Pasa bajo: $2 * F_c * sinc\left(2 * F_c * \left(n - \frac{N}{2}\right)\right)$	`n=0:N;` `h1 = 2*Fc2*sinc(2*Fc2*(n-(N/2)))`
	Filtro Pasa alto: $\delta\left(n - \frac{N}{2}\right) - 2F_c sinc\left(2F_c\left(n - \frac{N}{2}\right)\right)$	`M=N+1;` `n=0:N;` `hpb = 2*Fc*sinc(2*Fc*(n-(N/2)))` `delta = zeros(1,M);` `delta((M+1)/2)=1;` `h1=delta-hpb;`
	Filtro Elimina banda => Filtro pasa alto + filtro pasa bajo: $\delta\left(n - \frac{N}{2}\right)$ $- 2F_{c2} sinc\left(2F_{c2}\left(n - \frac{N}{2}\right)\right)$ $+ 2F_{c1} sinc\left(2F_{c1}\left(n - \frac{N}{2}\right)\right)$	`M=N+1;` `n=0:N;` `h1 = 2*Fc2*sinc(2*Fc2*(n-(N/2)));` `h2= 2*Fc1*sinc(2*Fc1*(n-(N/2)));` `delta = zeros(1,M);` `delta((M+1)/2)=1;` `hhp=delta-h1;` `he=hhp+h2;` *La respuesta al impulso de un primer filtro pasa bajo se representa por `h1` con frecuencia de corte `Fc2` y longitud `M`, se utiliza para formar un filtro pasa alto `hhp` con frecuencia de corte `Fc2` y longitud `M`, un segundo filtro pasa bajo se representa por `h2` con frecuencia de corte `Fc1` y longitud `M`. Luego con la suma de `hhp` y `h1` se obtiene un filtro elimina banda `he` con frecuencias de corte `Fc1` y `Fc2` y longitud `M`.*
	Pasa banda => Filtro pasa bajo2 - Filtro pasa bajo1: $2F_{c2} sinc\left(2F_{c2}\left(n - \frac{N}{2}\right)\right)$ $- 2F_{c1} sinc\left(2F_{c1}\left(n - \frac{N}{2}\right)\right)$	`n=0:N;` `h1 = 2*Fc2*sinc(2*Fc2*(n-(N/2)));` `h2= 2*Fc1*sinc(2*Fc1*(n-(N/2)));` `hp=h2-h1;` *La respuesta al impulso de un filtro pasa bajo `h2` con frecuencia de corte `Fc2` y longitud `M` se resta con la respuesta al impulso de un segundo filtro pasa bajo `h1` con frecuencia de corte `Fc1` y longitud `M`, que da como resultado un filtro pasa banda con frecuencias de corte `Fc1` y `Fc2` de longitud `M`.*
Cálculo de la ventana $w(n)$:	$w(n) = 0.54 - 0.46\cos(2\pi\frac{n}{N})$	`w = .54 - .46*cos(2*pi*n/(N));`
Cálculo de la nueva respuesta al impulso $h(n)$ aplicando la ventana $w(n)$:	$h(n) = h_1(n)w(n)$	`h=h1.*w;`

Tabla 3. Ecuaciones para filtro FIR diseñado con ventana Hamming

Método de ventana Káiser

La ventana de káiser está basada en las funciones de Bessel, donde I_0 es la función de Bessel modificada de primer tipo de orden cero, el parámetro β que depende de N, provee un control continuo sobre el cambio entre el nivel del lóbulo lateral y el ancho del lóbulo principal. Valores grandes de β generan niveles bajos en los lobulos laterales, pero el precio es una anchura mayor del lóbulo principal [5]. Esta ventana puede proveer diferentes anchos de transición para una misma longitud de ventana M, que es algo de lo que carecen las otras [7].

El parámetro β se calcula como:

$$\beta = \begin{cases} 0.1102(A-8.7), & A > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 \leq A \leq 50 \\ 0, & A < 21 \end{cases} \tag{9}$$

Donde A es la atenuacion deseada en la banda de supresión.

El orden N del filtro se estima según:

$$N = \frac{A - 7.95}{2.285\Delta\omega}, \qquad A > 21\ dB \tag{10}$$

Donde el ancho de banda de transición se expresa como:

$$\Delta\omega = \left| \omega_p - \omega_s \right| \tag{11}$$

ω_p es la frecuencia de paso y ω_s es la frecuencia de supresión.

La siguiente tabla muestra las ecuaciones necesarias para diseñar un filtro selectivo en frecuencia por el método de ventana Káiser. En la parte derecha se muestra una posible forma de escribir las formulas en Matlab:

Proceso	Ecuaciones de Diseño	Código Matlab
Cálculo de las frecuencias de paso y supresión normalizadas ω_p, ω_s	De la misma forma anterior para filtro con ventana Hamming (Tabla 3)	
Cálculo de la(s) frecuencia(s) de corte F_c	De la misma forma anterior para filtro con ventana Hamming (Tabla 3)	
Cálculo del ancho de banda de transición $\Delta\omega$:	De la misma forma anterior para filtro con ventana Hamming (Tabla 3)	
Cálculo del orden N:	$N = \frac{A_s - 7.95}{2.285\Delta\omega_{min}}$	`N=ceil((As-7.95)/(2.285*BW));` `if rem(N,2)~=0` `  N=N+1;`

		`End`
Calculo del parámetro β :	$\beta = \begin{cases} 0.1102(A_s - 8.7), & A_s > 50 \\ 0.5842(A_s - 21)^{0.4} + 0.07886(A_s - 21), & 21 \le A_s \le 50 \\ 0, & A_s < 21 \end{cases}$	`if (A>50)` `B=0.1102*(A-8.7);` `elseif (21<=A<=50)` `B=0.5842*((A-21)^0.4)+0.07886*(A-21)` `elseif (A<21)` `B=0;` `End`
Cálculo de la respuesta al impulso unitario del filtro $h_1(n)$:	Dependiendo del tipo de filtro selectivo, se utiliza las ecuaciones usadas en Cálculo de la respuesta al impulso unitario del filtro $h_1(n)$ para el método de diseño por ventana Hamming (Tabla 3).	
Cálculo de la ventana w:	$w = \frac{I_0\left(\beta\sqrt{1-\left(1-\frac{2n}{N}\right)^2}\right)}{I_0(\beta)}$	`w= kaiser(M,B)'` En este caso se utilizo la función predefinida de MATLAB para ventana Kaiser de longitud `M` y parámetro `B`. El resultado `w` tiene la forma `Nx1` y debe ser traspuesto para que tenga la forma `1xN` y pueda ser multiplicado con el filtro $h_1(n)$ que tiene la forma `1xN`.
Cálculo de la nueva respuesta al impulso unitario $h(n)$ utilizando la ventana w:	$h(n) = h_1(n)w(n)$	`h=h1.*w;`

Tabla 4. Ecuaciones para diseño de filtro FIR con ventana Káiser

4. TRABAJO PREVIO

El estudiante debe fundamentarse en los siguientes temas para poder alcanzar los objetivos de esta práctica:

- Filtros digitales FIR
- Método de diseño de filtros FIR mediante ventanas
- Efecto Gibbs
- Comandos básicos para el tratamiento de señales en Matlab
- Entorno Simulink

5. EJEMPLO PRACTICO:

El siguiente ejemplo muestra las operaciones realizadas en Matlab y la simulación en el entorno Simulink para un filtro Elimina Banda tipo 1 diseñado con el método de ventana Hamming.

Creación de un archivo.m con el siguiente código :

```
%Diseño de filtro digital Elimina Banda tipo I por método de ventana
Hamming
clear all;
close all;
%% Carga de señal para simular
load handel
```

```
%% Especificaciones del Filtro
fp1=600;   %frecuencia de paso 1
fs1=1200;  %frecuencia de supresión 1
fs2=2700;  %frecuencia de paso 2
fp2=3200;  %frecuencia de supresión 2
fm=8000; %frecuencia de muestreo
As=50;     %atenuación de supresión

%%  Aproximaciones
%Cálculo de las frecuencias normalizadas y de corte.
Fp1=fp1/fm;
Fp2=fp2/fm;
Wp1=2*pi*Fp1; %frecuencia de paso 1 normalizada
Wp2=2*pi*Fp2; %frecuencia de paso 2 normalizada
Fs1=fs1/fm;
Fs2=fs2/fm;
Ws1=2*pi*Fs1; %frecuencia de supresión 1 normalizada
Ws2=2*pi*Fs2; %frecuencia de supresión 2 normalizada
Wc1=((Ws1+Wp1)/2);
Wc2=((Wp2+Ws2)/2);
Fc1= Wc1/(2*pi); %frecuencia de corte 1
Fc2= Wc2/(2*pi); %frecuencia de corte 2

%% Calculo del ancho de transición y el orden
% como el método de ventana que vamos a utilizar es Hamming
entonces
% ancho del lóbulo principal=(8*pi)/(N+1)
BW1=abs(Wp1-Ws1);
BW2=abs(Wp2-Ws2);
BW=min(BW1, BW2); %ancho de banda de transición
N= ceil(((8*pi)/BW)-1);% orden del filtro
if rem(N,2)~=0
    N=N+1;
end
M=N+1;  %longitud del filtro

%% Calculo de la respuesta al impulso
n=0:N;
% Respuesta en frecuencia filtro pasa bajo h1
h1 = 2*Fc2*sinc(2*Fc2*(n-(N/2)));
% Respuesta en frecuencia filtro pasa bajo h2
h2= 2*Fc1*sinc(2*Fc1*(n-(N/2)));
figure(1)
freqz(h1);title('Respuesta en Magnitud y Fase del filtro
pasabajo 1');
figure(2)
freqz(h2);title('Respuesta en Magnitud y Fase del filtro
pasabajo 2');
%funcion delta
delta = zeros(1,M);
delta((M+1)/2)=1;
figure(3)
stem(delta)

%% Generación de filtro Elimina Banda
hhp=delta-h1;%ecuación de un filtro Pasa alto sin ventana
figure(4)
freqz(hhp);title('Respuesta en Magnitud y Fase del filtro Pasa
Alto');
he=hhp+h2; %Suma de Pasa Alto + Pasa Bajo
```

```
figure(5)
freqz(he);title('Respuesta en Magnitud y Fase del filtro Elimina
Banda');
figure(6)
stem(he);title('Respuesta al impulso de filtro Elimina Banda');


%% Calculo de la Ventana Hamming
w = .54 - .46*cos(2*pi*n/(N));
figure(7)
stem(w);title('Grafica de la ventana Hamming');

%% Respuesta al impulso del filtro Elimina Banda con ventana
Hamming
h=he.*w;
figure(8)
stem(h);title('Respuesta al impulso del filtro Elimina Banda
diseñado con ventana Hamming');
figure(9)
freqz(h);title('Respuesta en Magnitud y Fase del filtro Elimina
Banda diseñado con ventana Hamming');

%% Simulación
sound(y,Fs)
out = conv(y,h);
%sound(out, Fs)
%%
figure(10)
subplot(2,1,1)
plot(y);title('Señal a Filtrar');
subplot(2,1,2)
plot(out);title('Señal Filtrada')
%%
figure(11)
subplot(2,1,1)
yx=fft(y);
plot(abs(yx));title('Modulo de la TF in');
axis([0 36555 0 800])
outx=fft(out);
subplot(2,1,2)
plot(abs(outx));title('Modulo de la TF out');
axis([0 36555 0 800])
```

-Para comprobar los resultados de manera auditiva, primero se elimina el indicador de comentario (%) de la función `sound(y,Fs)` *y se coloca el indicador en la función* `sound(out, Fs)`*, así se podrá escuchar la señal a filtrar, luego se hará el proceso inverso para escuchar la señal filtrada, volviendo a ejecutar la misma celda.*

Creación del modelo en Simulink:

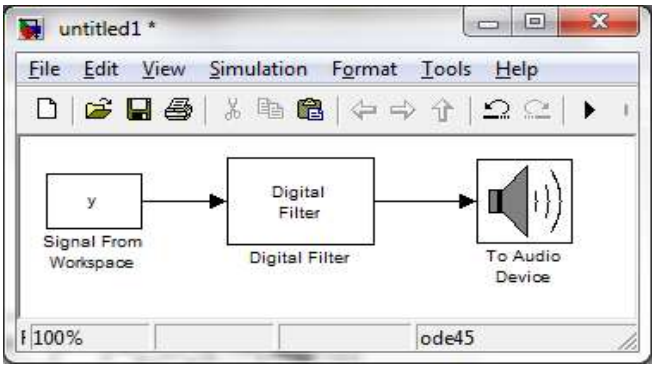


Figura 2. Diagrama de bloques del proceso de simulación

- " *Signal From Workspace* " : Este bloque importa una señal desde el área de trabajo de MATLAB a un modelo de Simulink.

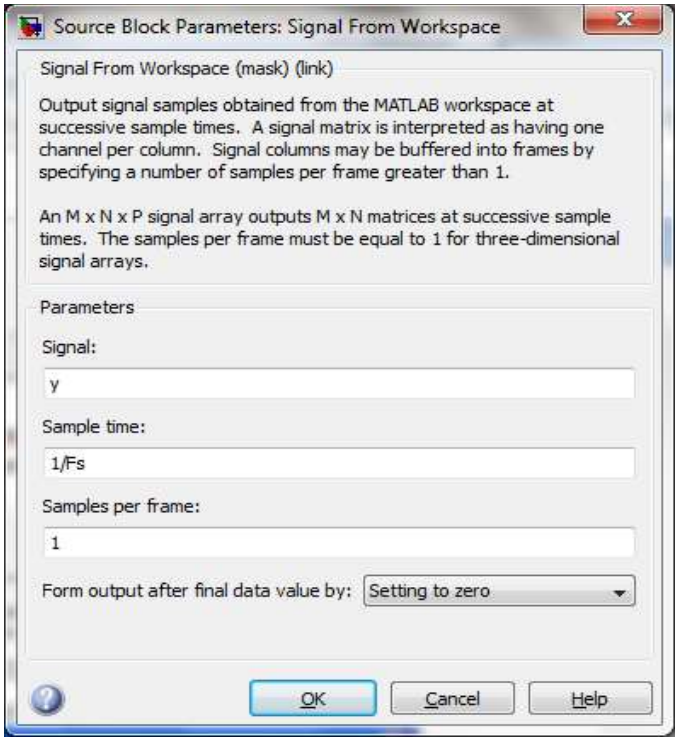


Figura 3. Recuadro de dialogo del bloque Signal From Workspace

Signal: Es el nombre de una variable en el área de trabajo de Matlab que contiene la señal a importar, o cualquier expresión válida de Matlab que especifique la señal. Para este caso, la señal utilizada es una señal de audio predefinida por Matlab denominada *handel*, la cual carga los datos en la variable 'y' y la frecuencia de muestreo en la variable 'Fs'. La variable 'y' tiene un tamaño 73113x1, con un rango de amplitudes que van desde -0.8 a 0.8

Sample time: es el período de muestreo de salida T_s de la señal. Para este caso el tiempo de muestreo es 1/Fs = 1/8192

Samples per frame: es el número de muestras Mo por cada trama de la señal. Para este caso las muestras por trama es 1.

Form output after final data value by: Especifica la salida después de que todas las muestras de la señal especificada han sido generadas. El bloque puede dar ceros de salida (`Setting to zero`), repetir la muestra final de datos (`Holding Final Value`) o repetir la señal entera desde el principio (`Cyclic Repetition`) si la señal es más corta que el tiempo de simulación. En esta práctica de laboratorio se escogió la forma `Setting to zero`.

- "*Digital Filter*": Este bloque se utiliza para implementar filtros de punto flotante o de punto fijo de los cuales se conozcan los coeficientes. El bloque de *Digital Filter* puede implementar filtros estáticos con coeficientes fijos así como filtros variantes en el tiempo con coeficientes que cambian a través del tiempo.
  El estado de la trama de salida y sus dimensiones son siempre las mismas de la señal de entrada que está siendo filtrada.

Figura 4. Recuadro de dialogo del bloque Digital Filter

Fuentes de coeficientes:

El bloque *Digital Filter* puede operar en tres modos diferentes. Se selecciona el modo en el menú **Coefficient source**:

- **Dialog parameters** (Parámetros de dialogo), se debe digitar información sobre el filtro tales como la estructura y los coeficientes.
- **Input port(s)** (puerto(s) de entrada), se debe digitar la estructura del filtro, y los coeficientes del filtro entran por uno o más puertos del bloque. Este modo es útil para especificar filtros variables en el tiempo.

- **Discrete-time filter object (DFILT)** (objeto de filtro discreto en el tiempo (DFILT)), se especifica el filtro usando un objeto `dfilt`[1].

Para este caso se utilizo el modo **Dialog parameters:**

Transfer function type: se selecciona el tipo de función de transferencia del filtro, en este caso FIR (todos ceros).

Filter structure: se selecciona la estructura del filtro. En este caso la estructura utilizada fue *Forma Simétrica Directa*; esta estructura fue escogida atendiendo la sugerencia de Matlab para la reducción del consumo de recursos de la FPGA. La estructura directa puede también ser usada pues es la más común para realizar filtros no recursivos y su atractivo principal, es su simplicidad y facilidad de programación [8].

Numerator coefficients: se especifica el vector de coeficientes en el numerador de la función de transferencia del filtro. En este caso, los coeficientes en el numerador fueron obtenidos desde el *workspace* por medio del vector *h*, que representa la respuesta al impulso unitario del filtro en el código Matlab anterior.

Initial Conditions: se especifican las condiciones iníciales de los elementos de retardo. Se asume que son cero que equivale a asumir que las entradas y salidas pasadas son cero. El bloque inicializa todos los elementos de retardo en el filtro en cero.

View Filter Response: Este botón abre la Herramienta de Visualización del filtro *fvtool* de la *Signal Processing Toolbox* y muestra la respuesta en frecuencia del filtro FIR definido por el bloque. Este botón sólo está disponible cuando el *Filter Design Toolbox* está instalado.

- "*To audio device*": Este bloque envía datos de audio a un dispositivo de audio de la computadora.

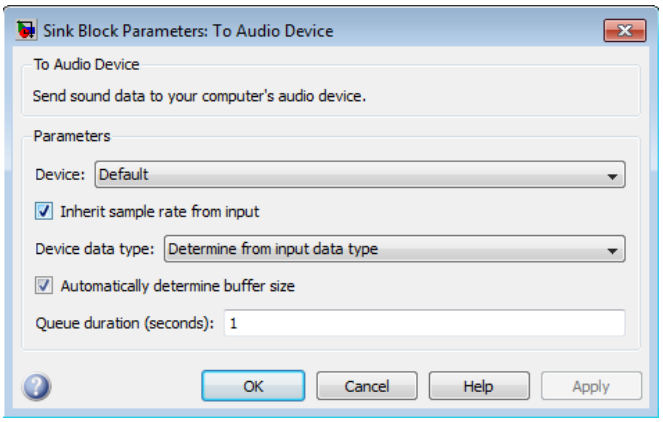


Figura 5. Recuadro de dialogo del Bloque To Audio Device

[1] dfilt: Es una función que permite crear un objeto de tipo dfilt (Filtro de tiempo discreto), cuenta con varias opciones de estructuras para diseñarlo y un grupo de métodos para acceder al objeto y modificar su información.

Device: Se especifica el dispositivo al que se desea enviar los datos de audio. Si se conecta o desconecta un dispositivo de audio del sistema, digitar `clear mex` en los comandos de Matlab para actualizar la lista.

Inherit sample rate from input: se selecciona esta casilla de verificación para que el bloque herede la frecuencia de muestreo de la señal de audio de la entrada al bloque.

Device data type: se especifica el tipo de datos de audio enviada al dispositivo.

Automatically determine buffer size: se selecciona esta casilla para permitir que el bloque calcule un tamaño del búfer.

Queue duration (seconds)): se especifica el tamaño de la cola en segundos, su valor es 1 por defecto.

Una vez hechos estos cambios, se inicia la simulación dando click en *Start Simulation* para comprobar el funcionamiento del filtro.

6. PROCEDIMIENTO DE LABORATORIO PARA EL METODO DE VENTANAS

- Software: Matlab, Simulink
- Equipo: parlantes, computador

6.1 Diseño en Matlab. Diseñar un filtro (Pasa bajo, Pasa Alto, Pasa Banda o Elimina Banda):_____ con la técnica de ventaneo:_____ con las siguientes especificaciones: _____

6.2 Simulación en Matlab. Aplicar el filtro a la señal de audio utilizada.

6.3 Migración al entorno Simulink. En la ventana de comandos, escribir *Simulink* para abrir esta herramienta, en donde se buscaran y se llevaran a una hoja de nuevo modelo los bloques *Signal From Workspace*, *To audio device* y *Digital Filter*, como el modelo de la figura 2. Realizar los ajustes necesarios de acuerdo a las indicaciones del profesor.

6.4 Resultados

- Comprobar si la respuesta en frecuencia del filtro diseñado coincide con las especificaciones dadas de frecuencias de paso y supresión.

  _____

  _____

- Comprobar si la respuesta en frecuencia del filtro diseñado coinciden con la especificación dada de atenuación en la banda de supresión.

  _____

  _____

- Según la grafica de Respuesta en Frecuencia para el filtro diseñado, ¿Cuál es la atenuación mínima de la banda de supresión para el filtro con la técnica de ventana utilizada?*

  _____

- Tabule los resultados obtenidos para este método en la siguiente tabla:

Ventana	Pico del primer lóbulo lateral (dB)	Ancho de banda de transición (xπ)	Ancho del lóbulo principal de la ventana (xπ)	Rizado de banda de paso (dB)	Atenuación de banda de supresión (dB)*
Káiser					
Otra ventana					

Tabla 5. Resultados de simulación en Matlab

- ¿Qué efectos realiza el filtro diseñado con la técnica de ventana escogida a la señal?

  _____
  _____
  _____

- Al disminuir el ancho de transición entre las bandas, ¿Que sucede con el orden del filtro?

  _____
  _____
  _____
  _____

- De la simulación, ¿Qué puede concluir al escuchar la señal de salida?

  _____
  _____

- En el modelo en Simulink, colocar un osciloscopio y observar la señal de entrada y salida. ¿Qué conclusiones puede sacar? Al comparar las graficas obtenidas en la parte de diseño en el entorno Matlab con las observadas en el osciloscopio que puede concluir.

  _____
  _____
  _____
  _____

*Considere atenuación mínima al valor de atenuación del primer lóbulo lateral.

7. EJEMPLO PRACTICO DE VENTANA KAISER:

El siguiente ejemplo muestra las operaciones realizadas en Matlab y la simulación en el entorno Simulink para un filtro Pasa Banda tipo 1 diseñado con el método de ventana Káiser.

Creación de un archivo.m con el siguiente código :

```
%Diseño de filtro FIR Pasa Banda tipo I con un método de ventana
Káiser
clear all;
close all;
%% Carga de señal para simular
load handel

%% Especificaciones
fs1=300;   %frecuencia de supresión 1
fs2=2100;  %frecuencia de supresión 2
fp1=800;   %frecuencia de paso 1
fp2=1500;  %frecuencia de paso 2
fm = 8000; %frecuencia de muestreo
```

```
As1=50;    %atenuación en la banda de supresión 1
Ap=1;      %atenuación en la banda de paso
As2=50;    %atenuación en la banda de supresión

%% Aproximaciones
Fp1=fp1/fm;
Wp1=2*pi*Fp1; %frecuencia de paso 1 normalizada
Fp2=fp2/fm;
Wp2=2*pi*Fp2; %frecuencia de paso 2 normalizada
Fs1=fs1/fm;
Ws1=2*pi*Fs1; %frecuencia de supresión 1 normalizadas
Fs2=fs2/fm;
Ws2=2*pi*Fs2; %frecuencia de supresión 2 normalizada
Wc1=((Ws1+Wp1)/2);
Wc2=((Wp2+Ws2)/2);
Fc1= Wc1/(2*pi); %frecuencia de corte 1
Fc2= Wc2/(2*pi); %frecuencia de corte 2

%% Calculo de ancho de banda transición y el orden del filtro
BW1=abs(Wp1-Ws1);
BW2=abs(Wp2-Ws2);
BW=min(BW1, BW2); %ancho de banda transición
N=ceil((As1-7.95)/(2.285*BW));
if rem(N,2)~=0
    N=N+1;
end
M=N+1; %longitud del filtro

%% Calculo del parámetro Beta
if (As1>50)
   B=0.1102*(As1-8.7);
elseif(21<=As1<=50)
   B= 0.5842*((As1-21)^0.4)+0.07886*(As1-21);
elseif(As1<21)
   B=0;
end

%% Generación del filtro Pasa Banda
n=0:N;
h1= 2*Fc2*sinc(2*Fc2*(n-(N/2)));
h2= 2*Fc1*sinc(2*Fc1*(n-(N/2)));
figure(1)
freqz(h1);title('Respuesta en Magnitud y Fase del Filtro Pasa
Bajo 1');
figure(2)
freqz(h2);title('Respuesta en Magnitud y Fase del Filtro Pasa
Bajo 2');
hp=h1-h2;
figure(3)
freqz(hp);title('Respuesta en Magnitud y Fase del Filtro Pasa
Banda');
figure(4)
stem(hp);title('Respuesta al impulso de Filtro Pasa Banda');

%% Ventana Kaiser
w= kaiser(M,B)';
figure(5)
stem(w);title('Grafica de la ventana Káiser');

%% Filtro Elimina Banda con ventana Káiser
h=hp.*w;
```

```
figure(6)
freqz(h);title('Respuesta en frecuencia del Filtro Elimina Banda
con ventana Káiser');

%% Simulacion
sound(y,Fs)
out = conv(y,h);
%sound(out,Fs)
%%
figure(7)
subplot(2,1,1)
plot(y);title('Señal a Filtrar');xlabel('Muestras');
ylabel('amplitud')
subplot(2,1,2)
plot(out);title('Señal Filtrada');xlabel('Muestras');
ylabel('amplitud')
%%
figure(8)
subplot(2,1,1)
yx=fft(y);
plot(abs(yx));title('Modulo de la TF in');
axis([0 36555 0 800])
outx=fft(out);
subplot(2,1,2)
plot(abs(outx));title('Modulo de la TF out');
axis([0 36555 0 800])
```

-Para comprobar los resultados de manera auditiva, primero se elimina el indicador de comentario (%) de la función `sound(y,Fs)` *y se coloca el indicador en la función* `sound(out, Fs)`*, así se podrá escuchar la señal a filtrar, luego se hará el proceso inverso para escuchar la señal filtrada, volviendo a ejecutar la misma celda.*

Creación del modelo en Simulink:

La simulación del filtro Káiser en la herramienta Simulink utiliza el mismo modelo usado para el filtro con ventana Hamming del ejemplo práctico anterior (Punto 6). Se debe ejecutar el código Matlab de la ventana Káiser para que el valor de la variable `h` que contiene los coeficientes del filtro Hamming anterior sea actualizado a los coeficientes del filtro actual. Para cerciorarse que fueron actualizados correctamente, dar click en *View Filter Response* del panel de dialogo del bloque *Digital Filter* para observar la grafica de respuesta en frecuencia del filtro Káiser. Dar click en *Start Simulation* para iniciar la simulación del modelo.

8. PROCEDIMIENTO DE LABORATORIO PARA VENTANA KAISER

8.1 Diseño en Matlab. Diseñar un filtro (Pasa Bajo, Pasa Alto, Elimina Banda o Pasa Banda): _____ con la técnica de ventaneo Káiser, con las siguientes especificaciones: _____

8.2 Simulación en Matlab. Aplicar el filtro a la señal de audio utilizada.

8.3 Migración al entorno Simulink. En la ventana de comandos, escribir *Simulink* para abrir esta herramienta, en donde se buscaran y se llevaran a una hoja de nuevo modelo los bloques *Signal From Workspace*, *To audio device* y *Digital Filter*, como el modelo de la figura 2. Realizar los ajustes necesarios de acuerdo a las indicaciones del profesor.

8.4 Resultados

- ¿Cuál es la atenuación de la banda suprimida al aplicar ventana Káiser*?

  _____

- Comprobar si la respuesta en frecuencia del filtro diseñado coincide con las especificaciones dadas de frecuencias de paso y supresión.

  _____

  _____

- Comprobar si la respuesta en frecuencia del filtro diseñado coinciden con la especificación dada de atenuación en la banda de supresión.

  _____

  _____

- Varié valores de β y observe que pasa con la respuesta en frecuencia del filtro. Escriba los resultados observados en los anchos de banda de transición, el orden del filtro y la atenuación en la banda de supresión.
  Valores menores de β:

  _____

  _____

  _____

  Valores mayores de β:

  _____

  _____

  _____

- De la simulación, ¿Qué puede concluir al escuchar la señal de salida?

  _____

  _____

- En el modelo en Simulink, colocar un osciloscopio y observar la señal de entrada y salida. ¿Qué conclusiones puede sacar? Al comparar las graficas obtenidas en la parte de diseño en el entorno Matlab con las observadas en el osciloscopio que puede concluir.

  _____

  _____

  _____

- Llenar la tabla 1.
  *Considere atenuación mínima al valor de atenuación del primer lóbulo lateral.

9. TRABAJO COMPLEMENTARIO

Diseñar y simular un filtro Pasa Alto, utilizando la ventana adecuada, que cumpla las siguientes especificaciones:

```
fs=2300;    %frecuencia de supresión
fp=2800;    %frecuencia de paso
fm=8000;    %frecuencia de muestreo
As=60;      %atenuación de la banda eliminada
Ap=2;       %atenuación de la banda de paso
```

10. REPORTE

El estudiante debe entregar las respuestas del ítem Resultados, junto con el diseño en Matlab(.m) y el modelo en Simulink(.mdl) de los filtros diseñados.

11. BILBIOGRAFIA

[1] GARCIA ORTEGA M. INTRODUCCION A FILTROS DIGTALES. Instituto Politécnico Nacional – Mexico.2003.

[2] PROAKIS J. MONOLAKIS D. DIGITAL SIGNAL PROCESSING. Third Edition-1996.

[3] FELDBAUER C. DIGITAL FILTER IMPLEMENTATION I. Laboratory handout, http://www.spsc.tugraz.at/courses/dsplab/filt1/filt1.pdf.

[4] RIVERA M. FILTROS DIGITALES FIR E IIR. Universidad Técnica de Federico Santa María. Valparaiso-2009.

[5] SMITH J. SPECTRAL AUDIO SIGNAL PROCESSING. Marzo 2007

[6] LAI E. PRACTICAL DIGITAL SIGNAL PROCESSING FOR ENGINEERS AND TECHNICIANS. Singapore

[7] INGLE V. PROAKIS J. DIGITAL SIGNAL PROCESSING, USING MATLAB V4. Northeasterm University-1997

[8] IFEACHOR E. JERVIS B. DIGITAL SIGNAL PROCESSING. A PRACTICAL APPROACH. Second Edition

PRACTICA No 2. SIMULACION DE FILTROS DIGITALES TIPO FIR-METODO DE RIZADO CONSTANTE

1. OBJETIVO

- Diseñar Filtros FIR con el método de Rizado Constante.
- Simular los filtros FIR diseñados con el método de Rizado Constante en Matlab y Simulink

2. INTRODUCCION

En esta práctica se presentan los conceptos básicos para el diseño de filtros digitales tipo FIR usando el método de Rizado Constante, además se muestra en un ejemplo práctico, la utilización de esta técnica en el diseño y la simulación de filtros en la herramienta Matlab. El proceso de laboratorio desarrollado por el estudiante tiene como guía el ejemplo práctico enunciado anteriormente, pero los requisitos de diseño y algunos otros parámetros serán definidos por el profesor.

3. MARCO TEORICO

Un filtro es un sistema continuo o discreto que modifica el espectro de una señal de entrada de acuerdo a ciertas especificaciones para producir una señal de salida. Para sistemas lineales e invariantes en el tiempo el espectro de la señal salida es igual al espectro de la señal de entrada multiplicado por la respuesta en frecuencia del sistema.

Los filtros pueden clasificarse como analógico y digitales. Un filtro digital se puede definir como un proceso computacional implementado con circuitos y/o programación, en el cual una secuencia numérica de entrada se transforma en otra secuencia numérica de salida con características predeterminadas. Matemáticamente, un filtro digital se representa por una ecuación de diferencias, y para su implementación se hace uso de sumadores y multiplicadores binarios y bloques de atraso [1].

Existen dos tipos de filtros digitales, IIR (respuesta infinita al impulso) y FIR (respuesta finita al impulso). En esta práctica abordaremos solamente los filtros tipo FIR.

Las diferentes configuraciones que se pueden realizar de filtros digitales FIR o IIR son mostradas en la figura 1:

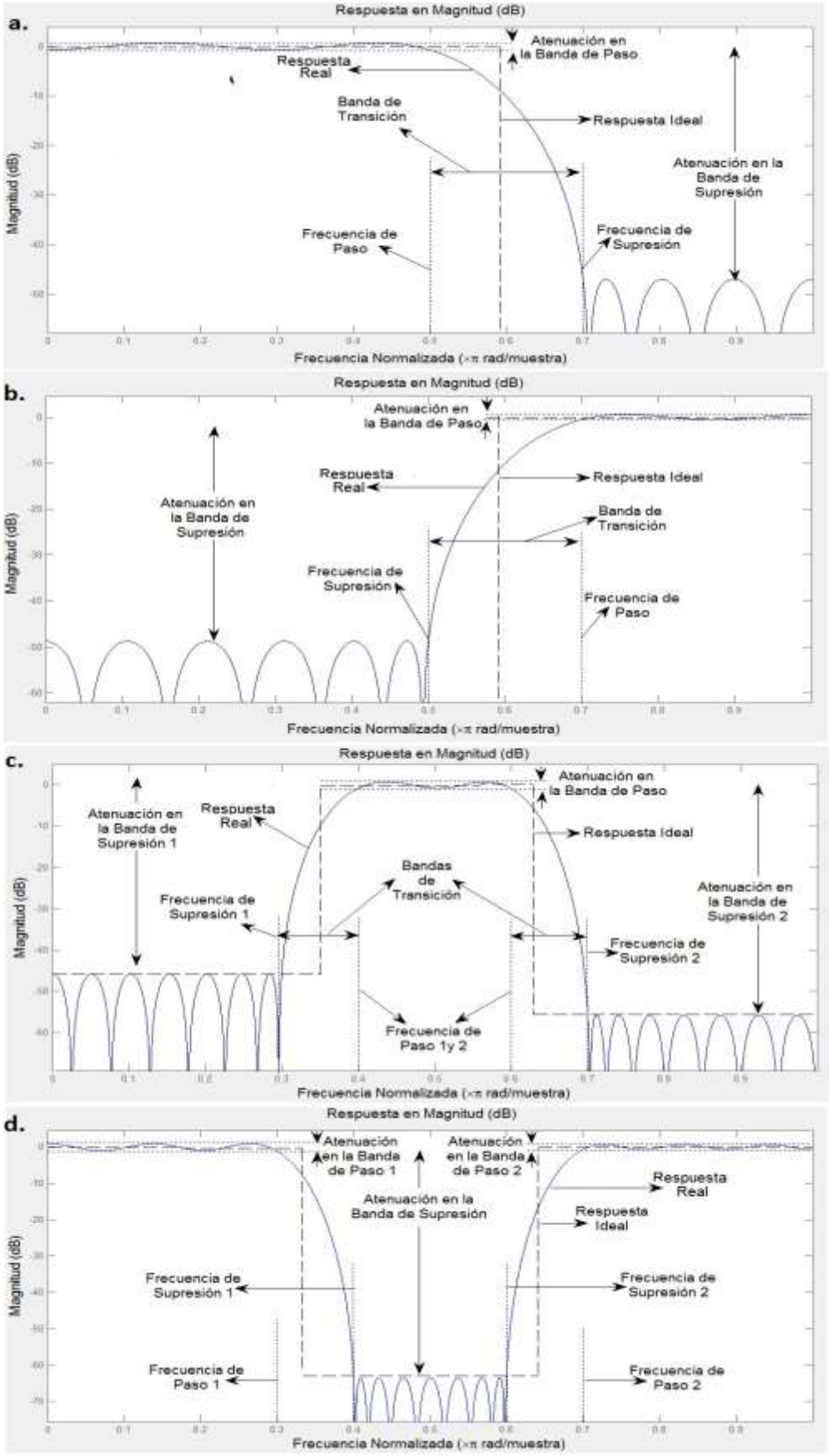


Figura 1. Configuración: a. Pasa Bajo, b. Pasa Alto, c. Pasa banda, d. Elimina banda.

Filtros FIR

Un filtro FIR con longitud M, cuyo orden es $M-1=N$, con entrada $x(n)$ y salida $y(n)$ se describe por la ecuación de diferencias:

$$y(n) = \sum_{K=0}^{N} b_k\, x(n-k) \tag{1}$$

Donde $\{b_k\}$ son los coeficientes del filtro.

Es posible representar la secuencia de salida como la convolución de la respuesta al impulso del sistema $h(n)$ con la señal de entrada. Entonces se tiene:

$$y(n) = \sum_{K=0}^{N} h(k)x(n-k) \tag{2}$$

Donde los limites superior e inferior de la sumatoria reflejan las características de causalidad y duración finita del filtro. La ecuación 1 y la ecuación 2 son iguales en su forma, por lo tanto, se puede decir que $b_k = h(k),\ k = 0,1,2,\ldots\ldots,N$ [2].

La transformada Z de la respuesta al impulso $h(n)$ produce la función transformada $H(z)$, la cual tiene, en el caso de los filtros causales FIR, N ceros cuya posición está determinada por los coeficientes b_k y N polos en el origen. Entonces, un filtro FIR es siempre estable. Debido al hecho de que la trasformada de la función de un filtro es exclusivamente determinada por la posición de los ceros, los filtros FIR son llamados filtros "todo-ceros" [3].

$$H(z) = \sum_{k=0}^{N} h(k)z^{-k} \tag{3}$$

Diseño de filtros FIR usando método de rizado constante

Los métodos de diseño de filtros FIR de ventaneo y de muestreo en frecuencia tienen la desventaja de que la aproximación del error que es la diferencia entre la respuesta en frecuencia ideal y la real del filtro no es uniformemente distribuida sobre todos los intervalos de banda. Esta distribución es mayor cerca a los bordes de las bandas y menor lejos de estos. Sin embargo distribuyendo el error uniformemente, como lo hace el método del rizado constante, se puede obtener un filtro de menor orden comparado con los otros métodos, que satisfaga las mismas especificaciones [4].

Se define la función de error o error ponderado como la diferencia entre la respuesta en frecuencia ideal del filtro y la respuesta real, multiplicada por una función $W_{pond}(\omega)$ que puede ser expresada como:

$$E(\omega) = W_{pond}(\omega)[H_D(\omega) - H(\omega)] \tag{4}$$

En donde $H_D(\omega)$ es la respuesta en frecuencia ideal o deseada, $H(\omega)$ es la respuesta en frecuencia real y $W_{pond}(\omega)$ es una función de ponderación que permite escoger el tamaño relativo de los errores en las diferentes bandas de frecuencia, en particular es conveniente normalizar $W_{pond}(\omega)$ a la unidad en la banda de supresión e igualar

$W_{pond}(\omega) = \frac{\delta_s}{\delta_p}$ en la banda de paso, para que el máximo error en ambas bandas de paso y de supresión llamado δ se minimice a δ_s .

Se ha establecido que cuando el $max|E(\omega)|$ se minimiza, la respuesta en frecuencia del filtro resultante tiene rizado constante en las bandas de paso y supresión. El cumplimiento de este objetivo lo ofrece el algoritmo de Parks-McClellan que provee una solución iterativa haciendo uso del algoritmo de intercambio de Remez para llegar a δ=δ_s. Cuando esta igualdad no se cumple entonces se debe incrementar M (si $\delta > \delta_s$) o decrementar M (si $\delta < \delta_s$) y usar el algoritmo Remez nuevamente para determinar el nuevo δ. Se repite el procedimiento hasta que $\delta \leq \delta_s$ [4].

Por otra parte, Káiser desarrollo la siguiente fórmula para determinar el orden N requerido:

$$N = \frac{-20log_{10}\sqrt{\delta_p\delta_s} - 13}{14.6\Delta_f} + 1$$

(5).

$$\Delta_f = \frac{\omega_s - \omega_p}{2\pi}$$

La siguiente tabla muestra las ecuaciones necesarias para diseñar un filtro FIR selectivo en frecuencia por el método de Rizado Constante. En la parte derecha se muestra una posible forma de escribir las formulas en Matlab:

Proceso	Ecuaciones de Diseño	Código Matlab
Cálculo de las frecuencias de paso y supresión normalizadas ω_p, ω_s: Con las especificaciones dadas de *fs* (frecuencia de supresión en Hz), *fp* (frecuencia de paso en Hz) y *fm* (frecuencia de muestreo en Hz), se calcula:	Frecuencia de supresión/paso normalizada: $F_{s,p} = \frac{f_{s,p}}{f_m}$ $\omega_{s,p} = 2\pi F_{s,p}$	`Fp=fp/fm;` `Wp=2*pi*Fp;`
Cálculo de la(s) frecuencia(s) de corte F_c :	Filtro Pasa bajo y Pasa alto: $\omega_c = \frac{\omega_p+\omega_s}{2}$ $F_c = \frac{\omega_c}{2*pi}$	`Wc=(Wp+Ws)/2;` `Fc=Wc/(2*pi);`
	Filtro Elimina banda y Pasa banda: $\omega_{c1} = \frac{\omega_{p1}+\omega_{s1}}{2}$, $\omega_{c2} = \frac{\omega_{p2}+\omega_{s2}}{2}$ $F_{c1} = \frac{\omega_{c1}}{2*pi}$, $F_{c2} = \frac{\omega_{c2}}{2*pi}$	`Wc1=((Ws1+Wp1)/2);` `Wc2=((Wp2+Ws2)/2);` `Fc1= Wc1/(2*pi);` `Fc2= Wc2/(2*pi);`
	Filtro Pasa bajo y Pasa alto: $\Delta\omega = \|\omega_s - \omega_p\|$	`BW=abs(Ws-Wp);`

Cálculo del ancho de banda de transición $\Delta\omega$:	Filtro Elimina banda y Pasa banda: $\Delta\omega_1 = \lvert\omega_{s1} - \omega_{p1}\rvert$ $\Delta\omega_2 = \lvert\omega_{p2} - \omega_{s2}\rvert$ Se escoge el menor entre ambos	`BW1=abs(Ws1-Wp1);` `BW2=abs(Wp2-Ws2);` `BW=min(BW1, BW2);`
Cálculo de las atenuaciones en unidades naturales δ_s y δ_p: Este paso se realiza si las especificaciones están dadas de forma relativa (dB)	$\delta_p = \frac{1 - 10^{-\frac{A_p}{20}}}{1 + 10^{-\frac{A_p}{20}}} = \frac{10^{\frac{A_p}{20}} - 1}{10^{\frac{A_p}{20}} + 1}$ $\delta_s = 10^{-\frac{A_s}{20}} * (1 + \delta_p)$	`dp=(10^(Ap/20)-1)/(10^(Ap/20)+1);` `ds=(1+dp)*(10^(-As/20));`
Cálculo del orden del filtro N:	$N = \frac{-20log_{10}\left(\sqrt{\delta_s\delta_p}\right) - 13}{14.6\Delta f}$	`N=ceil((-20*log10(sqrt(ds*dp))-13)/(14.6*BWf))`
Cálculo de la función de ponderación W_{pond}: Este vector tiene el valor de $\frac{\delta_s}{\delta_p}$ para la banda de paso y 1 para la banda suprimida, así:	Filtro Pasa bajo $W_{pond} = \left[\frac{\delta_s}{\delta_p} \quad 1\right]$	`Wpond=[ds/dp 1];`
	Filtro Pasa alto $W_{pond} = \left[1 \quad \frac{\delta_s}{\delta_p}\right]$	`Wpond=[1 ds/dp];`
	Filtro Elimina banda $W_{pond} = \left[\frac{\delta_s}{\delta_{p1}} \quad 1 \quad \frac{\delta_s}{\delta_{p2}}\right]$	`Wpond=[ds/dp1 1 ds/dp2];`
	Filtro Pasa banda $W_{pond} = \left[1 \quad \frac{\delta_s}{\delta_p} \quad 1\right]$	`Wpond=[1 ds1/dp 1];`
Cálculo del vector de las Frecuencias críticas normalizadas $F[\ ]$: Este vector contiene los valores de las frecuencias críticas normalizadas en orden ascendente de 0 a 1.	Filtro Pasa bajo $F = \left[0 \quad \frac{\omega_p}{\pi} \quad \frac{\omega_s}{\pi} \quad 1\right]$	`f=[0 wp/pi ws/pi 1];`
	Filtro Pasa alto $F = \left[0 \quad \frac{\omega_s}{\pi} \quad \frac{\omega_p}{\pi} \quad 1\right]$	`f=[0 ws/pi wp/pi 1];`
	Filtro Elimina banda $F = \left[0 \quad \frac{\omega_{p1}}{\pi} \quad \frac{\omega_{s1}}{\pi} \quad \frac{\omega_{s2}}{\pi} \quad \frac{\omega_{p2}}{\pi} \quad 1\right]$	`f=[0 wp1/pi ws1/pi ws2/pi wp2/pi 1]`
	Filtro Pasa banda $F = \left[0 \quad \frac{\omega_{s1}}{\pi} \quad \frac{\omega_{p1}}{\pi} \quad \frac{\omega_{p2}}{\pi} \quad \frac{\omega_{s2}}{\pi} \quad 1\right]$	`f=[0 ws1/pi wp1/pi wp2/pi ws2/pi 1]`
Cálculo del vector de las magnitudes de las bandas de frecuencia $A_o[\ ]$: Este vector tiene el valor de 1 para la banda de paso y 0 para la banda suprimida. Debe tener el mismo tamaño del vector de las frecuencias críticas normalizadas F[ ]	Filtro Pasa bajo: $A_o = [1 \ 1 \ 0 \ 0]$	`a=[1 1 0 0];`
	Filtro Pasa alto: $A_o = [0 \ 0 \ 1 \ 1]$	`a=[0 0 1 1];`
	Filtro Elimina banda: $A_o = [1 \ 1 \ 0 \ 0 \ 1 \ 1]$	`a=[1 1 0 0 1 1];`
	Filtro Pasa banda: $A_o = [0 \ 0 \ 1 \ 1 \ 0 \ 0]$	`a=[0 0 1 1 0 0];`

Cálculo de la respuesta al impulso del filtro $h(n)$:	Algoritmo de Parks-McClellan	**`h=firpm(N, F, Ao, Wpond)`** *Esta función entrega un filtro digital con longitud `N+1` que dependiendo de `F`, `Ao` y `Wpond` puede ser cualquiera de los 4 filtros selectivos en frecuencia.*

Tabla 1. Ecuaciones para diseño de filtro FIR con método de Rizado Constante

4. TRABAJO PREVIO

El estudiante debe fundamentarse en los siguientes temas para poder alcanzar los objetivos de esta práctica:

- Filtros digitales FIR
- Método de diseño de filtros FIR mediante rizado constante
- Efecto Gibbs
- Comandos básicos para el tratamiento de señales en Matlab
- Entorno Simulink
- Función `firpm` de Matlab

5. EJEMPLO PRACTICO

El siguiente ejemplo muestra las operaciones realizadas en Matlab y la simulación en el entorno Simulink para un filtro Pasa Alto tipo 1 diseñado con el método de Rizado Constante

Creación de un archivo.m con el siguiente código :

```
%Diseño de filtro FIR Pasa Alto de tipo I mediante el método de
rizado constante
clear all;
close all;
%% Señal a cargar
load handel

%% Especificaciones
Ap=2;      %Rizado de banda de paso
As=50;     %Atenuación de banda de supresión
fm=8000;   %frecuencia de muestreo
fp=2350;   %frecuencia de paso
fs=1800; %frecuencia de supresión

%% Aproximaciones
Fs1=fs/fm;
Fp=fp/fm;
ws=2*pi*Fs1; %frecuencia de supresión normalizada
wp=2*pi*Fp;  %frecuencia de paso normalizada

%% Calculo de las atenuaciones en unidades naturales
dp=(10^(Ap/20)-1)/(10^(Ap/20)+1); %rizado banda de paso en
unidades naturales
ds=(1+dp)*(10^(-As/20));%rizado banda de supresión en unidades
naturales

%% Calculo de la función de ponderación del error y el orden del
filtro
Wpond=[1 ds/dp];     %función de ponderación
BWf=(wp-ws)/(2*pi); %ancho de banda de transición
N=ceil((-20*log10(sqrt(ds*dp))-13)/(14.6*BWf));%orden del filtro
```

```
if rem(N,2)~=0
    N=N+1;
end

%% Calculo del vector de frecuencias críticas normalizadas
f=[0 ws/pi wp/pi 1]; %frecuencias criticas normalizadas
%% Calculo del vector de las magnitudes de las bandas de
frecuencia

a=[0 0 1 1]; %magnitudes de las bandas de frecuencia

%% Calculo de los coeficientes del filtro
h=firpm(N, f, a, Wpond);

%% Respuesta en frecuencia del filtro Pasa Alto diseñado
figure(1)
freqz(h,1,1024,fm);title('Respuesta en magnitud y en fase del
Filtro Pasa Alto');

%% Simulacion
%sound(y, Fs)
out = conv(y,h);
sound(out,Fs)
%%
figure(2)
subplot(2,1,1)
plot(y);title('Señal a Filtrar');
subplot(2,1,2)
plot(out);title('Señal filtrada');

figure(3)
subplot(2,1,1)
yx=fft(y);
plot(abs(yx));title('Modulo de la TF in');
axis([0 36555 0 800])
outx=fft(out);
subplot(2,1,2)
plot(abs(outx));title('Modulo de la TF out');
axis([0 36555 0 800])
```

-El vector f y el vector m deben ser del mismo tamaño
-La función firpm realiza el algoritmo de Parks-McClellan y retorna un filtro FIR de longitud N+1 el cual tiene la mejor aproximación a la respuesta en frecuencia descrita por f y m
-Para comprobar los resultados de manera auditiva, primero se elimina el indicador de comentario (%) de la función `sound(y,Fs)` *y se coloca el indicador en la función* `sound(out, Fs)`*, así se podrá escuchar la señal a filtrar, luego se hará el proceso inverso para escuchar la señal filtrada, volviendo a ejecutar la misma celda*

Creación del modelo en Simulink:

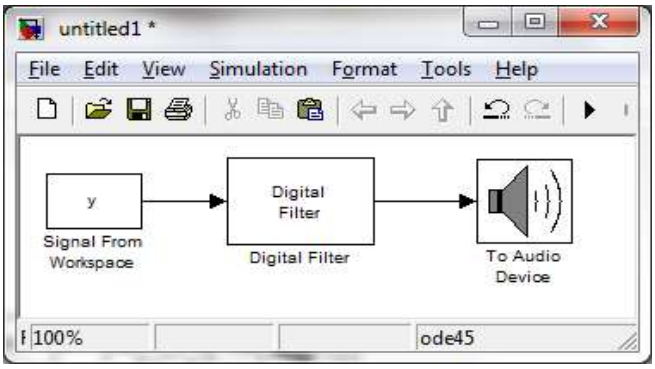


Figura 2. Diagrama de bloques del proceso de simulación

- " *Signal From Workspace* " : Este bloque importa una señal desde el área de trabajo de MATLAB a un modelo de Simulink.

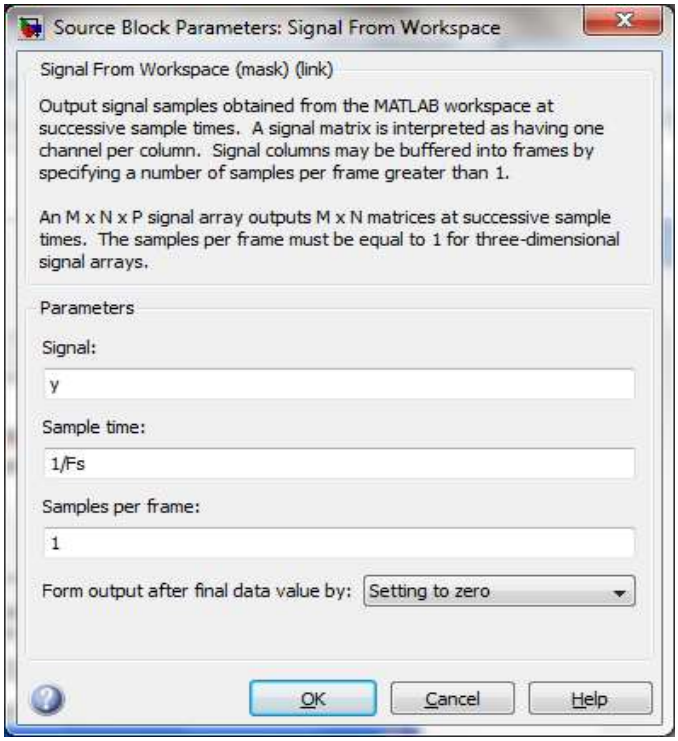


Figura 3. Recuadro de dialogo del bloque Signal From Workspace

Signal: Es el nombre de una variable en el área de trabajo de Matlab que contiene la señal a importar, o cualquier expresión válida de Matlab que especifique la señal. Para este caso, la señal utilizada es una señal de audio predefinida por Matlab denominada *handel*, la cual carga los datos en la variable 'y' y la frecuencia de muestreo en la variable 'Fs'. La variable 'y' tiene un tamaño 73113x1, con un rango de amplitudes que van desde -0.8 a 0.8

Sample time: es el período de muestreo de salida T_s de la señal. Para este caso el tiempo de muestreo es 1/Fs = 1/8192

Samples per frame: es el número de muestras Mo por cada trama de la señal. Para este caso las muestras por trama es 1.

Form output after final data value by: Especifica la salida después de que todas las muestras de la señal especificada han sido generadas. El bloque puede dar ceros de salida (`Setting to zero`), repetir la muestra final de datos (`Holding Final Value`) o repetir la señal entera desde el principio (`Cyclic Repetition`) si la señal es más corta que el tiempo de simulación. En esta práctica de laboratorio se escogió la forma `Setting to zero`.

- "*Digital Filter*": Este bloque se utiliza para implementar filtros de punto flotante o de punto fijo de los cuales se conozcan los coeficientes. El bloque de *Digital Filter* puede implementar filtros estáticos con coeficientes fijos así como filtros variantes en el tiempo con coeficientes que cambian a través del tiempo.
  El estado de la trama de salida y sus dimensiones son siempre las mismas de la señal de entrada que está siendo filtrada.

Figura 4. Recuadro de dialogo del bloque Digital Filter

Fuentes de coeficientes:

El bloque *Digital Filter* puede operar en tres modos diferentes. Se selecciona el modo en el menú **Coefficient source**:

- **Dialog parameters** (Parámetros de dialogo), se debe digitar información sobre el filtro tales como la estructura y los coeficientes.
- **Input port(s)** (puerto(s) de entrada), se debe digitar la estructura del filtro, y los coeficientes del filtro entran por uno o más puertos del bloque. Este modo es útil para especificar filtros variables en el tiempo.

- **Discrete-time filter object (DFILT)** (objeto de filtro discreto en el tiempo (DFILT)), se especifica el filtro usando un objeto `dfilt`[1].

Para este caso se utilizo el modo **Dialog parameters:**

Transfer function type: se selecciona el tipo de función de transferencia del filtro, en este caso FIR (todos ceros).

Filter structure: se selecciona la estructura del filtro. En este caso la estructura utilizada fue *Forma Simétrica Directa*; esta estructura fue escogida atendiendo la sugerencia de Matlab para la reducción del consumo de recursos de la FPGA. La estructura directa puede también ser usada pues es la más común para realizar filtros no recursivos y su atractivo principal, es su simplicidad y facilidad de programación [5].

Numerator coefficients: se especifica el vector de coeficientes en el numerador de la función de transferencia del filtro. En este caso, los coeficientes en el numerador fueron obtenidos desde el *workspace* por medio del vector *h*, que representa la respuesta al impulso unitario del filtro en el código Matlab anterior.

Initial Conditions: se especifican las condiciones iníciales de los elementos de retardo. Se asume que son cero que equivale a asumir que las entradas y salidas pasadas son cero. El bloque inicializa todos los elementos de retardo en el filtro en cero.

View Filter Response: Este botón abre la Herramienta de Visualización del filtro *fvtool* de la *Signal Processing Toolbox* y muestra la respuesta en frecuencia del filtro FIR definido por el bloque. Este botón sólo está disponible cuando el *Filter Design Toolbox* está instalado.

- "*To audio device*": Este bloque envía datos de audio a un dispositivo de audio de la computadora.

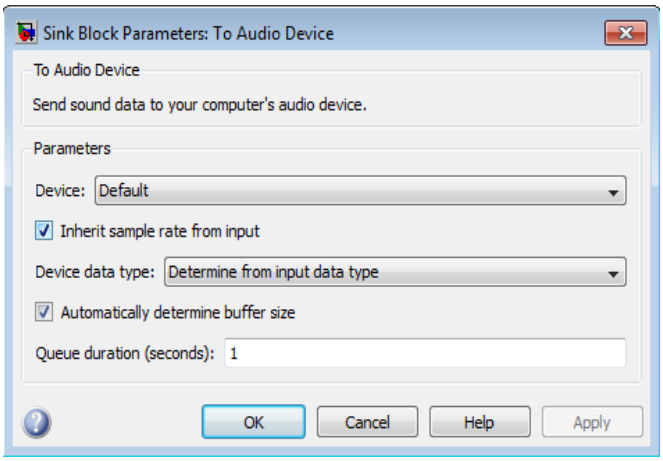


Figura 5. Recuadro de dialogo del bloque To Audio Device

[1] dfilt: Es una función que permite crear un objeto de tipo dfilt (Filtro de tiempo discreto), cuenta con varias opciones de estructuras para diseñarlo y un grupo de métodos para acceder al objeto y modificar su información.

Device: Se especifica el dispositivo al que se desea enviar los datos de audio. Si se conecta o desconecta un dispositivo de audio del sistema, digitar `clear mex en los comandos de Matlab para actualizar la lista.`

Inherit sample rate from input: se selecciona esta casilla de verificación para que el bloque herede la frecuencia de muestreo de la señal de audio de la entrada al bloque.

Device data type: se especifica el tipo de datos de audio enviada al dispositivo.

Automatically determine buffer size: se selecciona esta casilla para permitir que el bloque calcule un tamaño del búfer.

Queue duration (seconds)): se especifica el tamaño de la cola en segundos, su valor es 1 por defecto.

Una vez hechos estos cambios, se inicia la simulación dando click en *Start Simulation* para comprobar el funcionamiento del filtro.

6. PROCEDIMIENTO DE LABORATORIO PARA RIZADO CONSTANTE

- Software: Matlab, Simulink
- Equipo: parlantes, computador

6.1 Diseño en Matlab. Diseñar un filtro (Pasa Alto, Pasa Bajo, Elimina Banda o Pasa Banda):_____ con el método de rizado constante, con las siguientes especificaciones:
_____

6.2 Simulación en Matlab. Aplicar el filtro a la señal de audio utilizada.

6.3 Migración al entorno Simulink. En la ventana de comandos, escribir *Simulink* para abrir esta herramienta, en donde se buscaran y se llevaran a una hoja de nuevo modelo los bloques *Signal From Workspace*, *To audio device* y *Digital Filter*, como el modelo de la figura 2. Realizar los ajustes necesarios de acuerdo a las indicaciones del profesor.

6.4 Resultados

- Observando la Respuesta en frecuencia del filtro diseñado, ¿cómo es la atenuación obtenida respecto a la atenuación deseada de banda de supresión? Varié N para obtener los resultados esperados. Anote los cambios que observa. ¿Con que valor de N se obtiene la atenuación de supresión pedida en el diseño?
  _____
  _____
  _____
- De la simulación, ¿Qué puede concluir al escuchar la señal de salida?
  _____
  _____
- En el modelo en Simulink, colocar un osciloscopio y observar la señal de entrada y salida. ¿Qué conclusiones puede sacar? Al comparar las graficas obtenidas en la parte de diseño en el entorno Matlab con las observadas en el osciloscopio que puede concluir.

_____
_____
_____

- Un filtro Pasa bajo Hamming con `fm=8000`, `fp=800`, `fs=1300`, `Ap=3` y `As=50`, genera `As=57` y `N=64`. Genere un filtro Pasa bajo con el método de rizado constante con estas mismas especificaciones y escriba que cambios se presentan.

  _____
  _____
  _____
  _____

7. TRABAJO COMPLEMENTARIO

Diseñar, simular e implementar un filtro FIR Pasa banda mediante el método del rizado constante, con las siguientes especificaciones:

```
Ap=3; %Rizado de banda de paso
As1=40; %atenuación de banda de supresión 1
As2=45; %atenuación de banda de supresión 2
fm=8000;% frecuencia de muestreo
fs1=500;%frecuencia de supresión 1
fp1=600;%frecuencia de paso 1
fp2=1500;%frecuencia de paso 2
fs2=1600;%frecuencia de supresión 2
```

8. REPORTE

El estudiante debe entregar las respuestas del ítem Resultados, junto con el diseño en Matlab(.m) y el modelo en Simulink(.mdl) de los filtros diseñados y del filtro del Trabajo Complementario

9. BILBIOGRAFIA

[1] GARCIA ORTEGA M. INTRODUCCION A FILTROS DIGTALES. Instituto Politécnico Nacional – Mexico.2003.

[2] PROAKIS J. MONOLAKIS D. DIGITAL SIGNAL PROCESSING. Third Edition-1996.

[3] FELDBAUER C. DIGITAL FILTER IMPLEMENTATION I. Laboratory handout, http://www.spsc.tugraz.at/courses/dsplab/filt1/filt1.pdf.

[4] INGLE V. PROAKIS J. DIGITAL SIGNAL PROCESSING, USING MATLAB V4. Northeasterm University-1997

[5] IFEACHOR E. JERVIS B. DIGITAL SIGNAL PROCESSING. A PRACTICAL APPROACH. Second Edition

PRACTICA No 3. SIMULACION DE FILTROS DIGITALES TIPO IIR POR METODO DE BUTTERWORTH

1. OBJETIVO

- Diseñar Filtros IIR con la aproximación Butterworth.
- Simular los filtros IIR diseñados con la aproximación Butterworth en Matlab y Simulink

2. INTRODUCCION

En esta práctica se presentan los conceptos básicos para el diseño de filtros digitales tipo IIR diseñados con la aproximación Butterworth. Para aplicar esta teoría, se muestra en un ejemplo práctico, la utilización de esta técnica en el diseño y la simulación de filtros en la herramienta Matlab. El proceso de laboratorio desarrollado por el estudiante tiene como guía el ejemplo práctico enunciado anteriormente, pero los requisitos de diseño y algunos otros parámetros serán definidos por el profesor.

3. MARCO TEORICO

Un filtro es un sistema continuo o discreto que modifica el espectro de una señal de entrada de acuerdo a ciertas especificaciones para producir una señal de salida. Para sistemas lineales e invariantes en el tiempo el espectro de la señal salida es igual al espectro de la señal de entrada multiplicado por la respuesta en frecuencia del sistema.

Los filtros pueden clasificarse como analógico y digitales. Un filtro digital se puede definir como un proceso computacional implementado con circuitos y/o programación, en el cual una secuencia numérica de entrada se transforma en otra secuencia numérica de salida con características predeterminadas. Matemáticamente, un filtro digital se representa por una ecuación de diferencias, y para su implementación se hace uso de sumadores y multiplicadores binarios y bloques de atraso [1].

Existen dos tipos de filtros digitales, IIR (respuesta infinita al impulso) y FIR (respuesta finita al impulso). En esta práctica abordaremos solamente los filtros tipo IIR.

Las diferentes configuraciones que se pueden realizar de filtros digitales FIR o IIR son mostradas en la figura 1:

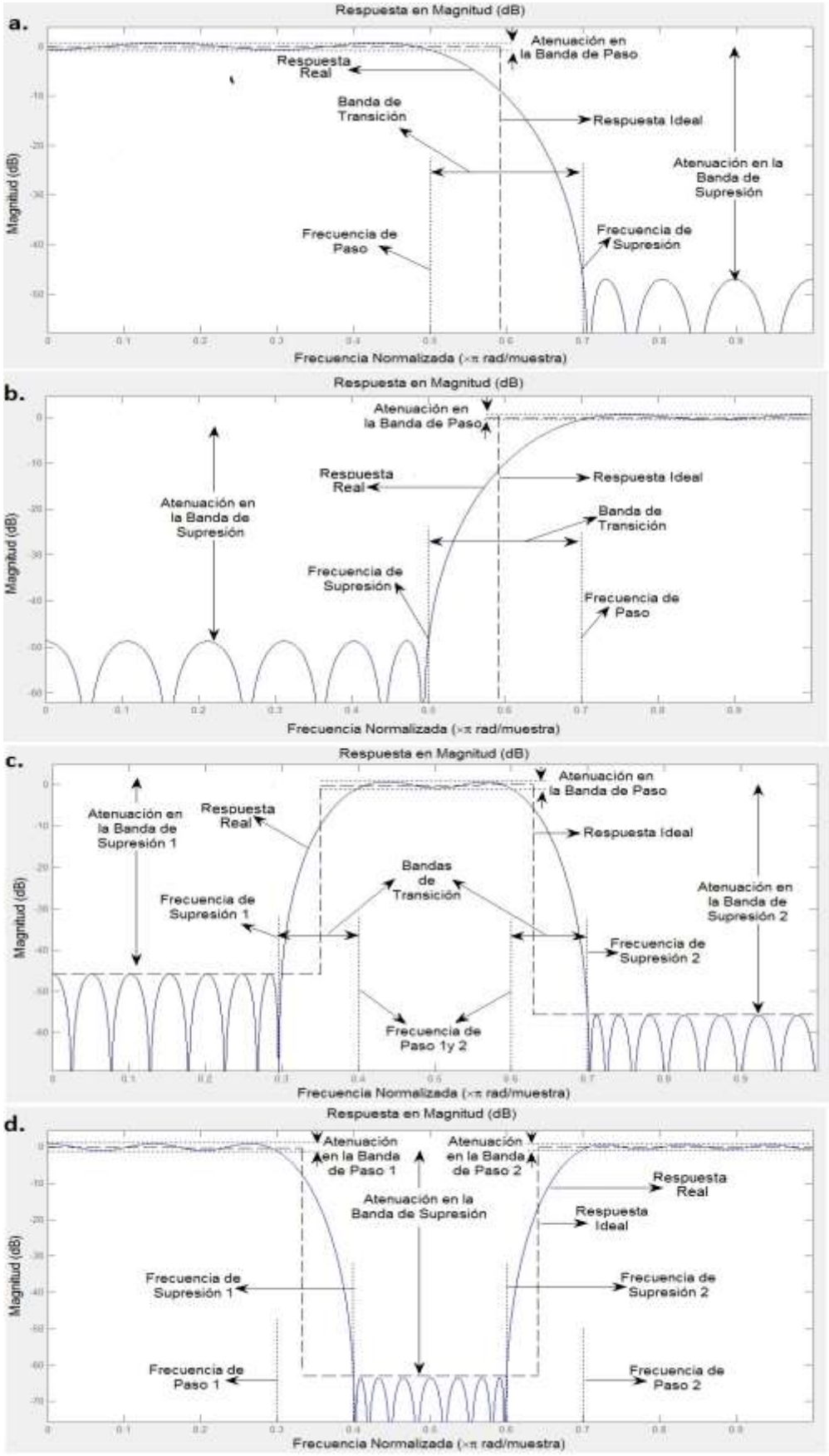


Figura 1. Configuración: a. Pasa Bajo, b. Pasa Alto, c. Pasa banda, d. Elimina banda.

Filtros IIR

Los filtros digitales con respuesta infinita al impulso IIR son recursivos, ya que la salida no solo depende de la entrada actual sino además de valores pasados de la salida (Filtros con realimentación).

La ecuación diferencia que implementa un filtro IIR es :

$$y(n) = -\sum_{K=1}^{N} a(k)y(n-k) + \sum_{K=0}^{M} b(k)x(n-k) \tag{1}$$

Donde $x(n)$ es la entrada y $y(n)$ es la salida, el orden del filtro es N si $N \geq M$ [1]

La respuesta al impulso $h(n)$ del filtro IIR es la salida que se produce cuando la entrada es el impulso. Conocida la respuesta al impulso, la salida del filtro $y(n)$, también puede ser obtenida por convolución lineal expresada como sigue:

$$y(n) = x(n) * h(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \tag{2}$$

Sin embargo la ecuación (2), no es computacionalmente realizable porque utiliza un número infinito de coeficientes. Por lo tanto, se limita la atención a los filtros IIR que se describen por la ecuación (1).

Los filtros digitales IIR pueden obtenerse fácilmente a partir del diseño de un filtro analógico y, a continuación, utilizar una técnica de mapeo para transformarlo desde el plano s al plano z.

Existen diversos métodos de diseño de filtros analógicos pasa bajos, en los que se encuentra el método de diseño de Butterworth, Chebyshev tipo I y II, elíptico y por último el método de Bessel. En este laboratorio se hará uso del método de diseño de Butterworth, el cual se explicará más adelante.

Hay dos métodos para el diseño de filtros digitales IIR sobre la base de filtros analógicos IIR existentes, el método de impulso invariante y la transformación bilineal; para no diseñar el filtro IIR digital directamente, estos métodos mapean el filtro digital en un filtro analógico equivalente, que puede ser diseñado por uno de los métodos de diseño de filtros analógicos antes mencionados.

El método de impulso invariante preserva la respuesta al impulso del filtro analógico original mediante la digitalización de la respuesta al impulso de filtros analógicos, pero no su respuesta en frecuencia (magnitud). Debido al aliasing inherente, el desempeño de este método es inapropiado para filtros paso alto o elimina banda. La transformación bilineal genera filtros muy eficientes, y está bien estudiada para el diseño de filtros selectivos en frecuencia. Los filtros digitales resultantes de la

transformación bilineal preservan las características de respuesta en magnitud de los filtros analógicos, pero no las propiedades de dominio del tiempo [2]. El método usado en el laboratorio es la transformación bilineal.

Si se desea diseñar otros tipos de filtros selectivos en frecuencia, como paso alto, paso banda y elimina banda se debe aplicar la transformación del eje de frecuencia (o banda) de un filtro de paso bajo a fin de que se comporte como otro filtro selectivo en frecuencia.

Método de Butterworth

El filtro pasa bajo de Butterworth es una aproximación todo polo[1] del filtro ideal, que se caracteriza por la respuesta en magnitud al cuadrado:

$$|H(\Omega)|^2 = \frac{1}{1 + (\Omega/\Omega_c)^{2N}} \tag{3}$$

En donde N es el orden del filtro, Ω es la variable que representa las frecuencias analógicas y Ω_c es la frecuencia de corte analógica.

El filtro analógico Pasa Bajo es especificado por los parámetros $\Omega_\mathrm{p}, \mathrm{A_p}, \Omega_\mathrm{s}$ y $\mathrm{A_s}$. Por lo tanto en el caso del filtro de Butterworth se debe encontrar el orden N y la frecuencia de corte Ω_c. Estas variables se encuentran usando las siguientes ecuaciones:

$$\mathrm{N} = \log_{10} \frac{\left[(10^{\frac{\mathrm{A_p}}{10}} - 1)/(10^{\frac{\mathrm{A_s}}{10}} - 1)\right]}{2\log_{10}(\Omega_\mathrm{p}/\Omega_\mathrm{s})} \tag{4}$$

$$\Omega_\mathrm{c} = \Omega_\mathrm{p}[\left(1 - \delta_p\right)^{-2} - 1]^{-\frac{1}{2N}} \tag{5}$$

Donde N es el orden del filtro, Ω_c es la frecuencia de corte, δ_p y δ_s son los valores de los rizados de las bandas de paso y supresión expresados en veces. Generalmente N no es un entero, entonces si se quiere que N sea un entero se debe escoger el entero superior [3].

Para realizar la conversión de las frecuencias del dominio digital al dominio analógico se usa la siguiente ecuación.

[1] Los filtros todo polo son aquellos que solo tiene términos en el denominador de su función de transferencia.

$$\Omega_i = \frac{2}{T_s} tan\left(\frac{\omega_i}{2}\right)$$

(6)

en donde Ω es la frecuencia de borde del filtro analógico, ω es la frecuencia de borde del filtro digital y T es el periodo de muestreo.

Transformación Bilineal

La transformación bilineal es un método que opera en el dominio de la frecuencia, el cual convierte la función de transferencia de un filtro analógico H(s) y una digital H(z).

La transformación es ejecutada realizando el siguiente cambio de variables.

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

(7)

Es llamada bilineal porque tanto el numerador como el denominador de la ecuación son ecuaciones lineales. Esta transformada es reversible, esto es que H(s) puede ser obtenida a partir de H(z), realizando la siguiente sustitución [4].

$$z = \frac{2/T + s}{2/T - s}$$

(8)

Debe tenerse en cuenta que para que la transformación sea exitosa, el filtro analógico a ser transformado debe ser completamente estable, es decir que todos sus polos y ceros estén en el semiplano izquierdo del plano S, para que así el filtro digital obtenido sea estable, es decir que todos los polos y ceros estén dentro de la circunferencia unitaria definida por el plano Z.

La siguiente tabla muestra las ecuaciones necesarias para diseñar un filtro IIR Butterworth. En la parte derecha se muestra una posible forma de escribir las formulas en Matlab:

Proceso	Ecuaciones de Diseño	Código Matlab
Cálculo de las frecuencias de paso y supresión normalizadas ω_p, ω_s: Con las especificaciones dadas de *fs* (frecuencia de supresión en Hz), *fp* (frecuencia de paso en Hz) y *fm* (frecuencia de muestreo en Hz), se	Frecuencia de supresión/paso normalizada: $F_{s,p}= \frac{f_{s,p}}{f_m}$ $\omega_{s,p} = 2\pi F_{s,p}$	`Fp=fp/fm;` `Wp=2*pi*Fp;`

calcula:		
Cálculo de las frecuencias analógicas Ω_p y Ω_s	$\Omega_p = tan\frac{\omega_p}{2}$ $\Omega_s = tan\frac{\omega_s}{2}$	`op=tan(wp/2);` `os=tan(ws/2);`
-PARA FILTRO PASA BAJO:		
Cálculo del orden del filtro N :	$aa = log_{10}\frac{10^{\frac{A_p}{10}}-1}{10^{\frac{A_s}{10}}-1}$ $dd = 2log_{10}\frac{\Omega_p}{\Omega_s}$ $N = \frac{aa}{bb}$	`aa= log10( (10^(ap/10)-1)/(10^(as/10)-1) );` `dd= 2*log10(op/os);` `n=ceil(aa/dd);`
Cálculo de la frecuencia de corte analógica Ω_c: con la atenuación dada `Ap` en dB, se halla su equivalente en veces para aplicar la fórmula.	$\delta_p = \frac{10^{\frac{A_p}{20}}-1}{10^{\frac{A_p}{20}}+1}$ $\Omega_c = \Omega_p\left[(1-\delta_p)^{-2}-1\right]^{-\frac{1}{2N}}$	`dp=(10^(ap/20)-1)/(10^(ap/20)+1);` `oc= op*((((1-dp)^-2)-1)^(-1/(2*n)));`
Cálculo de los coeficientes b/a de un filtro Pasa bajo:	Aproximación Butterworth	**`[b a]=butter(n, oc, 'low','s');`** *Esta función diseña un filtro analógico pasa bajo Butterworth de orden `n`, con frecuencia de corte `oc`, donde `b` es el vector que contiene los coeficientes del numerador y `a` es el vector que contiene los coeficientes del denominador*
Cálculo del filtro digital Pasa bajo:	Transformación Bilineal	**`[bdd add]=bilinear(b,a, Fs);`** *Este comando convierte los vectores `b` y `a` que contienen los coeficientes de la función de transferencia del filtro pasa bajo en el dominio `s` a una función discreta equivalente en el dominio `z`, donde el vector `bdd` es el que contiene los coeficientes del numerador y el vector `add` es el que contiene los coeficientes del denominador. El ultimo campo corresponde a la frecuencia de muestreo `Fs`.*
Cálculo de las secciones de segundo orden:	Secciones de segundo orden SOS	`[sos,g] = tf2sos(bdd,add);` *La función convierte los vectores de coeficientes `bdd, add` de la función de transferencia del filtro digital a una representación equivalente de secciones de segundo orden `sos` con ganancia `g`.* *Este paso es necesario para la generación de código vhdl a partir del modelo en Simulink.*
	Esta función grafica los	`zplane(b,a)`

Verificación de la estabilidad del filtro Pasa Bajo:	polos y ceros del filtro analógico diseñado. Si el resultado es un filtro analógico inestable se deben modificar las especificaciones de diseño.	
	Esta función grafica los polos y ceros del filtro digital diseñado. Si el resultado es un filtro digital inestable se deben modificar las especificaciones de diseño.	`zplane(bdd,add)`
-PARA FILTRO PASA ALTO:		
Cálculo del orden N y la frecuencia de corte Ω_c del filtro:	Filtro Analógico Pasa Alto:	`[n, oc]=buttord(op, os, ap, as, 's')` *Esta función retorna la frecuencia de corte analógica* `oc` *y el orden* `n` *más bajo de un filtro analógico Butterworth que pierde no más de* `ap(dB)` *de atenuación en la banda de paso y tiene por lo menos* `as(dB)` *de atenuación en la banda de supresión.* `op` *y* `os` *son las frecuencias analógicas de paso y de supresión.*
Cálculo de los coeficientes b/a del filtro:	Diseño de filtro Butterworth Pasa Alto	`[b a]=butter(n, oc, 'high','s');`
Cálculo del filtro digital:	Transformación bilineal	`[bdd add]=bilinear(b, a, 0.5);`
Cálculo de las secciones de Segundo orden:	Secciones de segundo orden `sos` con ganancia `g`	`[sos,g] = tf2sos(bdd,add);`
Verificación de la estabilidad del filtro:	Estas funciones grafican los polos y ceros del filtro analógico y digital diseñado. Si el resultado es un filtro inestable se deben modificar las especificaciones de diseño.	`zplane(b,a)` `zplane(bdd,add)`
-PARA FILTROS PASA BANDA Y ELIMINA BANDA:		
Cálculo de la frecuencia de corte y ancho de transición del filtro Analógico:	Filtro Pasa banda y Elimina banda	`Wo=sqrt(op1*op2);` `Bw = op2-op1;`
Cálculo del orden N del filtro analógico:	Filtro Analógico Pasa Banda y Elimina banda:	`[n, oc]=buttord(op, os, ap, as, 's')` Donde op=[op1, op2], os=[os1,

		os2]
Cálculo del prototipo del filtro Pasa bajo Butterworth:	Esta función retorna los ceros `z1`, polos `p1` y la ganancia `k1` de un prototipo de filtro Butterworth Pasa bajo normalizado de orden `n`. el resultado es un filtro que tiene `n` polos alrededor de la circunferencia de radio uno en el semiplano izquierdo y no tiene ceros.	`[z1,p1,k1]=buttap(n);`
Calculo de la función de transferencia del Prototipo del filtro Analógico:	Esta función retorna la función de transferencia a partir de la ubicación de los polos y los ceros y la ganancia	`[b,a]=zp2tf(z1,p1,k1)`
Calculo de la representación estado-espacio del prototipo:	Esta función calcula la representación estado-espacio a partir de la ubicación de los polos y los ceros y la ganancia.	`[A,B,C,D] = zp2ss(z1,p1,k1);`
Transformación del Prototipo Pasa Bajo al filtro Deseado	Filtro Pasa Banda: Esta función transforma el prototipo analógico pasa bajo con frecuencia de corte igual a 1 a un filtro pasa banda centrado en frecuencia `Wo` y ancho de banda de transición `Bw`.	`[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw);`
	Filtro Elimina Banda: Esta función transforma el prototipo analógico pasa bajo con frecuencia de corte igual a 1 a un filtro elimina banda centrado en frecuencia `Wo` y ancho de banda de transición `Bw`.	`[At,Bt,Ct,Dt] = lp2bs(A,B,C,D,Wo,Bw);`
Cálculo de los polos, ceros y ganancia del filtro Analógico Deseado	Calculo los polos, ceros y ganancia del filtro Analógico Deseado que se encuentra en representación estado-espacio.	`[zt,pt,kt] = ss2zp(At,Bt,Ct,Dt);`
Calculo de la función de transferencia del filtro Analógico Deseado:	Esta función retorna la función de transferencia a partir de la ubicación de los polos y los ceros y la ganancia	`[b1t,a1t]=zp2tf(zt,pt,kt)`
Verificación de la		

estabilidad del filtro Analógico Deseado:	Grafica los polos y ceros del filtro Analógico Deseado	`zplane(b1t,a1t)`
Calculo del filtro Digital Deseado:	Transformación Bilineal en términos de estado-espacio	`[Ad,Bd,Cd,Dd] = bilinear(At,Bt,Ct,Dt,0.5);`
Cálculo de los polos, ceros y ganancia del filtro Deseado Digital:	Calculo los polos, ceros y ganancia del filtro Digital Deseado que se encuentra en representación estado-espacio.	`[zd,pd,kd] = ss2zp(Ad,Bd,Cd,Dd);`
Cálculo de las secciones de Segundo orden:	Secciones de segundo orden `sos` con ganancia `g`	`[sos,g]=zp2sos(zd,pd,kd);`
Verificación de la estabilidad del filtro digital Deseado:	Grafica los polos y ceros del filtro Digital Deseado	`zplane(zd,pd);`
Calculo de la función de transferencia del filtro digital Deseado:	Esta función retorna la función de transferencia a partir de la ubicación de los polos y los ceros y la ganancia	`[b11,a11]=zp2tf(zd,pd,kd)`

Tabla 1. Ecuaciones para diseño de filtro IIR Butterworth

4. TRABAJO PREVIO

El estudiante debe fundamentarse en los siguientes temas para poder alcanzar los objetivos de esta práctica:

- Filtros digitales IIR
- Métodos de diseño de filtros IIR
- Comandos básicos para el tratamiento de señales en Matlab
- Entorno Simulink

5. EJEMPLO PRACTICO

El siguiente ejemplo muestra las operaciones realizadas en Matlab y la simulación en el entorno Simulink para un filtro IIR Pasa Bajo tipo 1 diseñado con el método de Butterworth.

Creación de un archivo.m con el siguiente código :

```
%Diseño de filtro IIR Pasa bajo Butterworth
clear all;
close all;

%% Carga de señal para simular
load handel

%% Especificaciones del filtro
```

```
fp=400;%frecuencia de paso
fs=1600;%frecuencia de supresión
ap=2;%atenuación banda de paso dB
as=50;% atenuación banda de supresión dB
fm=8000;% frecuencia de muestreo

%% Aproximaciones
%Cálculo de las frecuencias normalizadas.
Fp=fp/fm;
Fs1=fs/fm;
wp=2*pi*Fp;
ws=2*pi*Fs1;

%% Calculo de las frecuencias analógicas
op=tan(wp/2); %frecuencia de paso analógica
os=tan(ws/2); %frecuencia de supresión analógica

%% Calculo del orden n y frecuencia de corte analógica
aa= log10(  (10^(ap/10)-1)/(10^(as/10)-1) );
dd= 2*log10(op/os);
n=ceil(aa/dd);
dp=(10^(ap/20)-1)/(10^(ap/20)+1);
oc= op*((((1-dp)^-2)-1)^(-1/(2*n))); % frecuencia de corte

%% Calculo del filtro analógico Pasa Bajo
[b a]=butter(n, oc, 'low','s');
figure(1)
freqs(b,a);title('Filtro Analógico Pasa Bajo');

%% Calculo de filtro digital Pasa Bajo
[bdd add]=bilinear(b, a, 0.5);
[sos,g] = tf2sos(bdd,add);% secciones de segundo orden
figure(2)
freqz(bdd,add);title('Filtro Digital Pasa Bajo Deseado');

%% Simulación
%sound(y,Fs)
out=filter(bdd,add,y);
sound(out,Fs)
%%
figure(3)
subplot(2,1,1)
plot(y);title('Señal a Filtrar');xlabel('Muestras');
ylabel('amplitud')
subplot(2,1,2)
plot(out);title('Señal Filtrada');xlabel('Muestras');
ylabel('amplitud')
%%
figure(4)
subplot(2,1,1)
yx=fft(y);
plot(abs(yx));title('Modulo de la TF in');
axis([0 36555 0 800])
outx=fft(out);
subplot(2,1,2)
plot(abs(outx));title('Modulo de la TF out');
axis([0 36555 0 800])
%%
figure(5)
```

```
zplane(b,a);title('Polos y ceros del Filtro Analógico Pasa
Bajo');
figure(6)
zplane(bdd,add);title('Polos y ceros del Filtro Digital Pasa
Bajo');
```

-Para comprobar los resultados de manera auditiva, primero se elimina el indicador de comentario (%) de la función `sound(y,Fs)` *y se coloca el indicador en la función* `sound(out, Fs)`*, así se podrá escuchar la señal a filtrar, luego se hará el proceso inverso para escuchar la señal filtrada, volviendo a ejecutar la misma celda.*

Creación del modelo en Simulink:

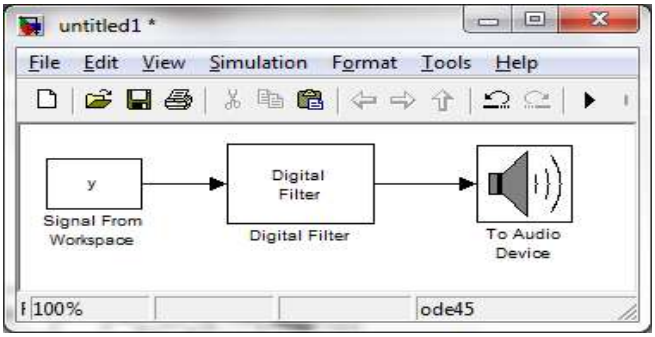


Figura 2. Diagrama de bloques del proceso de simulación

- " *Signal From Workspace* " : Este bloque importa una señal desde el área de trabajo de MATLAB a un modelo de Simulink.

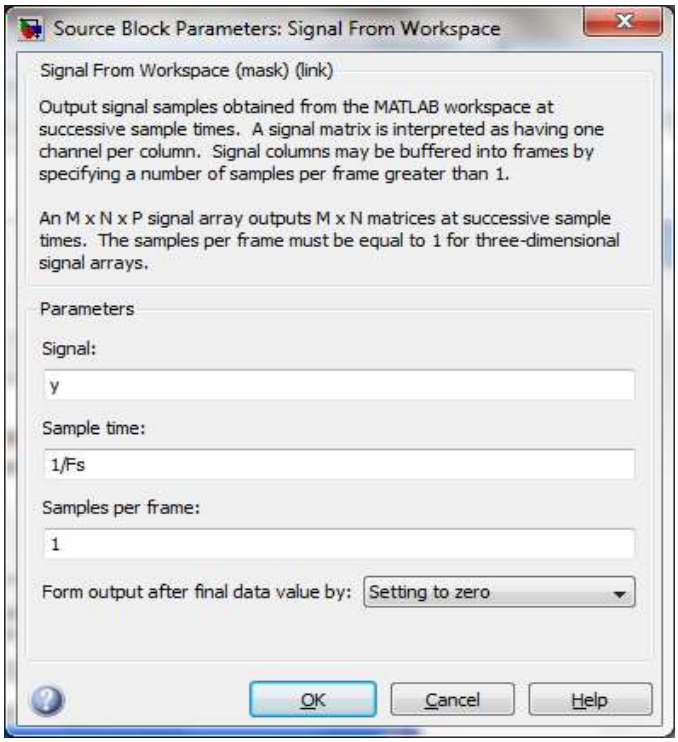


Figura 3. Recuadro de dialogo del bloque Signal From Workspace

Signal: Es el nombre de una variable en el área de trabajo de Matlab que contiene la señal a importar, o cualquier expresión válida de Matlab que especifique la señal. Para este caso, la señal utilizada es una señal de audio predefinida por Matlab denominada *handel*, la cual carga los datos en la variable 'y' y la frecuencia de muestreo en la

variable ‘Fs’. La variable ‘y’ tiene un tamaño 73113x1, con un rango de amplitudes que van desde -0.8 a 0.8

Sample time: es el período de muestreo de salida T_s de la señal. Para este caso el tiempo de muestreo es 1/Fs = 1/8192

Samples per frame: es el número de muestras Mo por cada trama de la señal. Para este caso las muestras por trama es 1.

Form output after final data value by: Especifica la salida después de que todas las muestras de la señal especificada han sido generadas. El bloque puede dar ceros de salida (`Setting to zero`), repetir la muestra final de datos (`Holding Final Value`) o repetir la señal entera desde el principio (`Cyclic Repetition`) si la señal es más corta que el tiempo de simulación. En esta práctica de laboratorio se escogió la forma `Setting to zero`.

- “*Digital Filter*”: Este bloque se utiliza para implementar filtros de punto flotante o de punto fijo de los cuales se conozcan los coeficientes. El bloque de *Digital Filter* puede implementar filtros estáticos con coeficientes fijos así como filtros variantes en el tiempo con coeficientes que cambian a través del tiempo.
  El estado de la trama de salida y sus dimensiones son siempre las mismas de la señal de entrada que está siendo filtrada.

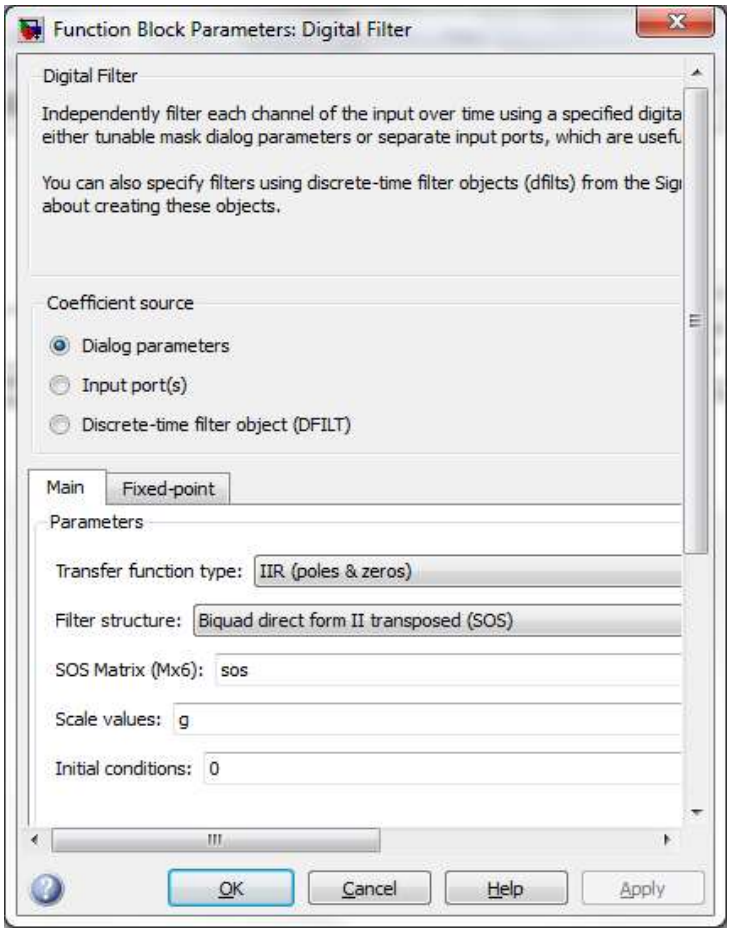


Figura 4. Recuadro de dialogo del bloque Digital Filter

Fuentes de coeficientes:

El bloque *Digital Filter* puede operar en tres modos diferentes. Se selecciona el modo en el menú **Coefficient source**:

- **Dialog parameters** (Parámetros de dialogo), se debe digitar información sobre el filtro tales como la estructura y los coeficientes.
- **Input port(s)** (puerto(s) de entrada), se debe digitar la estructura del filtro, y los coeficientes del filtro entran por uno o más puertos del bloque. Este modo es útil para especificar filtros variables en el tiempo.
- **Discrete-time filter object (DFILT)** (objeto de filtro discreto en el tiempo (DFILT)), se especifica el filtro usando un objeto `dfilt`[2].

Para este caso se utilizo el modo **Dialog parameters:**

Transfer function type: se selecciona el tipo de función de transferencia del filtro, en este caso IIR (polos y ceros).

Filter structure: se selecciona la estructura del filtro. En este caso la estructura seleccionada fue la Forma Directa II Transpuesta SOS Bicuadrada

SOS Matrix (Mx6): se especifica la matriz `sos` que contiene los coeficientes del filtro de secciones de segundo orden.

Scale Values: se especifica el valor escalar para ser aplicado antes y después de cada sección de un filtro bicuadrado. En este caso el valor es `g`

Initial Conditions: se especifican las condiciones iníciales de los elementos de retardo. Se asume que son cero que equivale a asumir que las entradas y salidas pasadas son cero. El bloque inicializa todos los elementos de retardo en el filtro en cero.

View Filter Response: Este botón abre la Herramienta de Visualización del filtro *fvtool* de la *Signal Processing Toolbox* y muestra la respuesta en frecuencia del filtro FIR definido por el bloque. Este botón sólo está disponible cuando el *Filter Design Toolbox* está instalado.

- "*To audio device*": Este bloque envía datos de audio a un dispositivo de audio de la computadora.

[2] dfilt: Es una función que permite crear un objeto de tipo dfilt (Filtro de tiempo discreto), cuenta con varias opciones de estructuras para diseñarlo y un grupo de métodos para acceder al objeto y modificar su información.

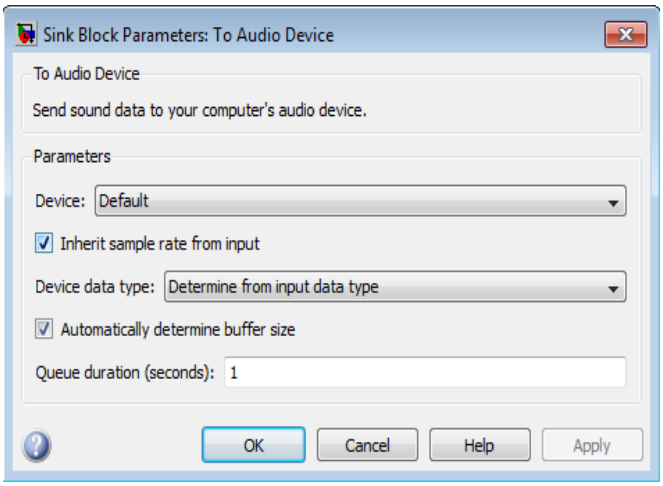


Figura 5. Recuadro de dialogo del bloque To Audio Device

Device: Se especifica el dispositivo al que se desea enviar los datos de audio. Si se conecta o desconecta un dispositivo de audio del sistema, digitar `clear mex en los comandos de Matlab para actualizar la lista.`

Inherit sample rate from input: se selecciona esta casilla de verificación para que el bloque herede la frecuencia de muestreo de la señal de audio de la entrada al bloque.

Device data type: se especifica el tipo de datos de audio enviada al dispositivo.

Automatically determine buffer size: se selecciona esta casilla para permitir que el bloque calcule un tamaño del búfer.

Queue duration (seconds)): se especifica el tamaño de la cola en segundos, su valor es 1 por defecto.

Una vez hechos estos cambios, se inicia la simulación dando click en *Start Simulation* para comprobar el funcionamiento del filtro.

6. PROCEDIMIENTO DE LABORATORIO PARA FILTRO DIGITAL IIR

- Software: Matlab, Simulink
- Equipo: parlantes, computador

6.1 Diseño en Matlab. Diseñar un filtro IIR Butterworth (Pasa Alto, Pasa Bajo, Elimina Banda o Pasa Banda): _____, con las siguientes especificaciones:

_____

6.2 Simulación en Matlab. Aplicar el filtro a la señal de audio utilizada.

6.3 Migración al entorno Simulink. En la ventana de comandos, escribir *Simulink* para abrir esta herramienta, en donde se buscaran y se llevaran a una hoja de nuevo modelo los bloques *Signal From Workspace*, *To audio device* y *Digital Filter*, como el modelo de la figura 2. Realizar los ajustes necesarios de acuerdo a las indicaciones del profesor.

6.4 Resultados

- Comprobar por medio de las graficas, si las respuestas en frecuencia de los filtros coinciden con las especificaciones dadas de frecuencias de paso y supresión tanto analógicas como digitales.

  _____
  _____

- Comprobar por medio de la graficas, si las respuestas en frecuencia de los filtros coinciden con la especificación dada de atenuación en la banda de supresión.

  _____
  _____

- Al pasar del dominio analógico al dominio digital, ¿Qué sucede con las atenuaciones en las bandas de paso y supresión? Justifique su respuesta.

  _____
  _____
  _____
  _____
  _____

- Al pasar del dominio analógico al dominio digital, ¿Qué sucede con la respuesta en fase del filtro? Justifique su respuesta.

  _____
  _____
  _____
  _____

- Compruebe el funcionamiento del filtro digital IIR, haciendo uso de las graficas de los espectros de la señal de audio antes y después del filtrado ¿Que concluye?

  _____
  _____
  _____

- De la simulación, ¿Qué puede concluir al escuchar la señal de salida?

  _____
  _____

- En el modelo en Simulink, colocar un osciloscopio y observar la señal de entrada y salida. ¿Qué conclusiones puede sacar? En comparación con las graficas de la parte de diseño en el entorno Matlab, ¿Cómo son las observadas en el osciloscopio?

  _____
  _____
  _____
  _____

- El filtro FIR Pasa bajo diseñado con el método del rizado constante con `fm=8000`, `fs=1200`, `fp=800`, `Ap=3` y `As=50`, genera `N=28` y `As=49`. Realice un filtro IIR Pasa bajo Butterworth con estas mismas especificaciones y escriba que cambios observa.

  _____
  _____
  _____

7. TRABAJO COMPLEMENTARIO:

Diseñe y simule un filtro digital IIR elimina banda con las siguientes especificaciones:

```
fp1=500; %frecuencia de paso 1
fs1=600;%frecuencia de supresión 1
fs2=1100;%frecuencia de supresión 2
fp2=1200;%frecuencia de paso 2
fm=8000; %frecuencia de muestreo
ap1=2; %atenuación en la banda de paso 1
as=50; %atenuación en la banda de supresión
ap2=2; %atenuación en la banda de paso 2
```

8. REPORTE:

El estudiante debe entregar las respuestas del ítem Resultados, junto con el diseño en Matlab (.m) y el modelo en Simulink (.mdl) de los filtros diseñados

9. BIBLIOGRAFIA

[1] GARCIA ORTEGA M. INTRODUCCION A FILTROS DIGTALES. Instituto Politécnico Nacional – Mexico.2003.

[2] KUO S. LEE B. REAL-TIME DIGITAL SIGNAL PROCESSING. 2001

[3] INGLE V. PROAKIS J. DIGITAL SIGNAL PROCESSING, USING MATLAB V4. Northeasterm University-1997

[4] OPPENHEIM A. SCHAFER R. BUCK J. DISCRETE-TIME SIGNAL PROCESSING. 1998

PRACTICA No 4. SIMULACION DE FILTRO DIGITAL ADAPTATIVO LMS

1. OBJETIVO

Simular el algoritmo LMS, utilizando el *Embedded Matlab Subset*[1] en el entorno Simulink

2. INTRODUCCION

En esta práctica se presentan los conceptos principales del filtrado adaptativo y del algoritmo adaptativo LMS. Más adelante, se muestra el diseño y la simulación de un modelo Simulink para el algoritmo LMS diseñado mediante *Embedded Matlab Subset.* El proceso de laboratorio desarrollado por el estudiante consiste en modificar parámetros de diseño a petición del profesor del modelo inicial presentado.

MARCO TEORICOruido/

iEl algoritmo LMS (Least Mean Square) es un algoritmo de filtrado adaptativo lineal, cuya simplicidad casi lo ha convertido en estándar; consta básicamente de dos procesos básicos:

I. Un proceso de filtrado, que consiste en:
   - Calculo de la salida del filtro transversal $y(n)$ producida por un conjunto de muestras de entrada $x(n)$.
   - Generar una estimación de error e(n) mediante la comparación de $y(n)$ con la señal deseada $d(n)$

II. Un proceso adaptativo, que consiste en el ajuste automático del vector de coeficientes del filtro $w(n)$ de acuerdo con la estimación del error.

Luego la combinación de estos dos procesos trabajando juntos constituyen el lazo de realimentación alrededor del algoritmo LMS, como se muestra en la Figura 1.

[1] Embedded Matlab Subset -Subconjunto Embedded Matlab: es un subconjunto del lenguaje Matlab, que provee las herramientas para generar código C eficiente para aplicaciones embebidas. Con este lenguaje trabaja el bloque *Emdedded MATLAB Function Block*, que permite adicionar funciones en Matlab a modelos en Simulink.

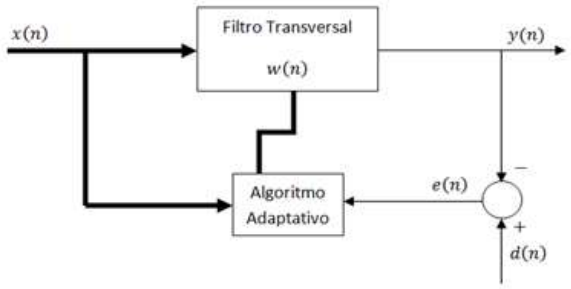


Figura 1. Lazo de realimentación algoritmo LMS [1]

En esta práctica de laboratorio, se algoritmo LMS para de ruido. La grafica siguiente describe este tipo de configuración:

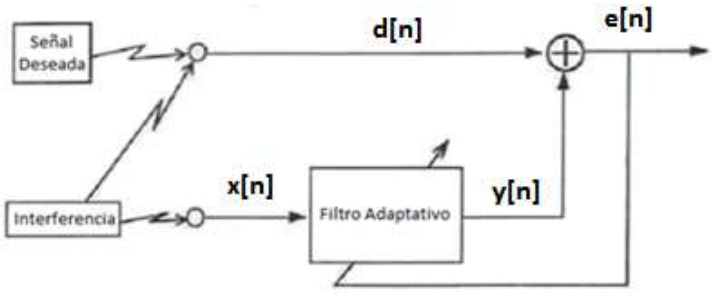


Figura 2. Filtro Adaptativo Cancelador de Interferencia [2]

Las señales y ecuaciones que describen este algoritmo son:

- $\boldsymbol{w}[n]$: Vector de coeficientes
- $\boldsymbol{x}[n]$: Vector de ruido
- $\boldsymbol{d}[n]$: Vector de señal contaminada
- $y[n]$: Salida del filtro
- $\boldsymbol{w}[n+1]$: Vector de coeficientes actualizados
- Filtrado: $y[n] = w^T[n]x[n]$
- Error de estimación: $e[n] = \boldsymbol{d}[n] - y[n]$
- Adaptación del vector de coeficientes: $\boldsymbol{w}[n+1] = \boldsymbol{w}[n] + 2\mu e(n)\boldsymbol{x}[n]$
  Donde μ, es un parámetro de ajuste fijo [3]

3. TRABAJO PREVIO

El estudiante debe fundamentarse en los siguientes temas para poder alcanzar los objetivos de esta práctica:

- Filtrado Adaptativo
- Algoritmo LMS
- Comandos básicos de Matlab

- Embedded Matlab Subset y Embedded Matlab Function Block
- Entorno Simulink

5. EJEMPLO PRACTICO

a. Diseño y simulación en Matlab

En el Proyecto *Vectorized Adaptive noise canceler using LMS Filter*, que se descarga de http://www.mathworks.com/matlabcentral/fileexchange/16278, se encuentran las funciones que se utilizan para simular el algoritmo LMS, mediante el *Embedded Matlab Function Block*. Estas funciones se modificaron para un filtro de 6 coeficientes.

Creación del modelo en Simulink con el bloque Embedded MATLAB Function:

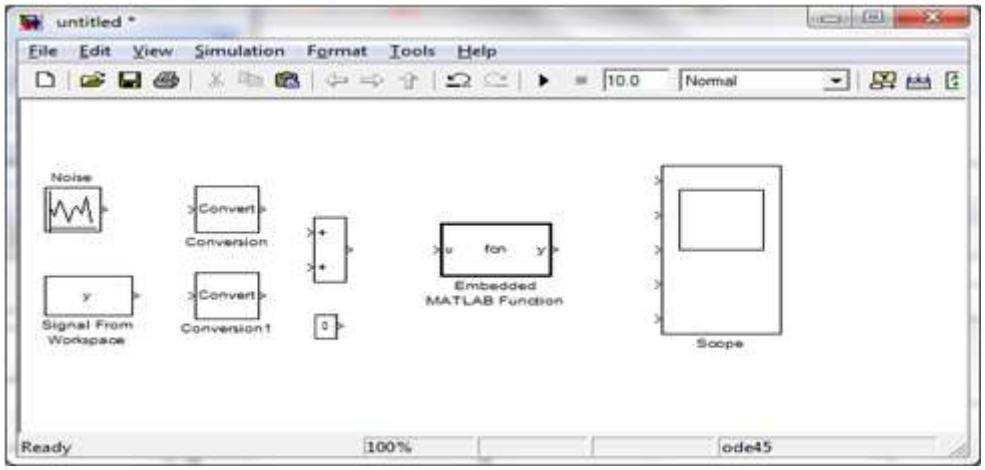


Figura 3. Modelo inicial del algoritmo LMS en Simulink

- " *Random Source* ": Genera valores distribuidos aleatoriamente. El bloque genera una trama de M valores extraídos de una distribución uniforme o Gaussiana[2] pseudo-aleatoria, donde se especifica M en el parámetro de *Muestras por trama*.

[2] Distribución normal o gaussiana: Al iniciar el análisis estadístico de una serie de datos, y después de la etapa de detección y corrección de errores, un primer paso consiste en describir la distribución de las variables estudiadas. Una de las distribuciones teóricas más utilizadas en la práctica es la distribución normal, también llamada distribución gaussiana. Su importancia se debe fundamentalmente a la frecuencia con la que distintas variables asociadas a fenómenos naturales y cotidianos siguen, aproximadamente, esta distribución. La distribución de una variable normal está determinada por dos parámetros, su media y su desviación estándar, denotadas generalmente por μ y σ^2 . Con esta notación, la función de densidad de una característica x que sigue una distribución normal de media μ y varianza σ^2 viene dada por la siguiente ecuación que determina la curva en forma de campana que tiene esta distribución [4]:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left\{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}; \quad -\infty < x < \infty$$

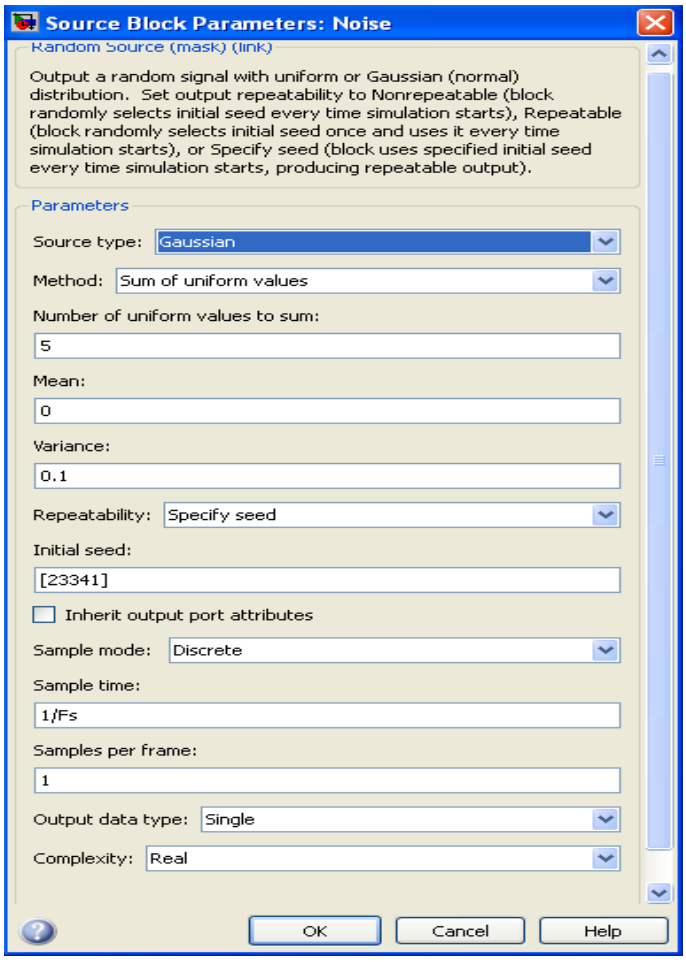


Figura 4. Recuadro de dialogo del bloque Random Source

Source type: la distribución de la cual se extraen los valores aleatorios, uniforme o gausiana.

Method: el método por el cual el bloque calcula los valores aleatorios gaussianos, `Ziggurat` o `sum of uniform values`.

Number of uniform values to sum: el número de valores aleatorios distribuidos uniformemente por sumar para calcular un solo número en una distribución aleatoria Gausiana.

Mean: la media de la distribución Gausiana normal.

Variance: la varianza de la distribución Gausiana normal.

Repeatability: es la repetibilidad de la salida del bloque: `Not repeatable`, `Repeatable` o `Specify seed`. En las configuraciones `Repeatable` and `Specify seed`, el bloque saca la misma señal cada vez que se corre la simulación

Initial seed: son los números semilla iniciales que el generador de números aleatorios usa. El generador produce una secuencia idéntica de números aleatorios cada vez que es ejecutado con una semilla inicial particular.

Sample mode: se especifica el modo de muestreo `continuo` o `discreto`.

Sample time: es el periodo de muestreo `Ts`, de la secuencia de salida aleatoria

Sample per frame: es el número de muestras `M`, en cada trama de salida. Cuando el valor de este parámetro es 1, el bloque saca una señal basada en muestras.

Output data type: especifica el tipo de datos de salida de precisión única o doble precisión.

Complexity: especifica si la complejidad de la salida es real o compleja.

- " *Signal From Workspace* " : Este bloque importa una señal desde el área de trabajo de MATLAB a un modelo de Simulink.

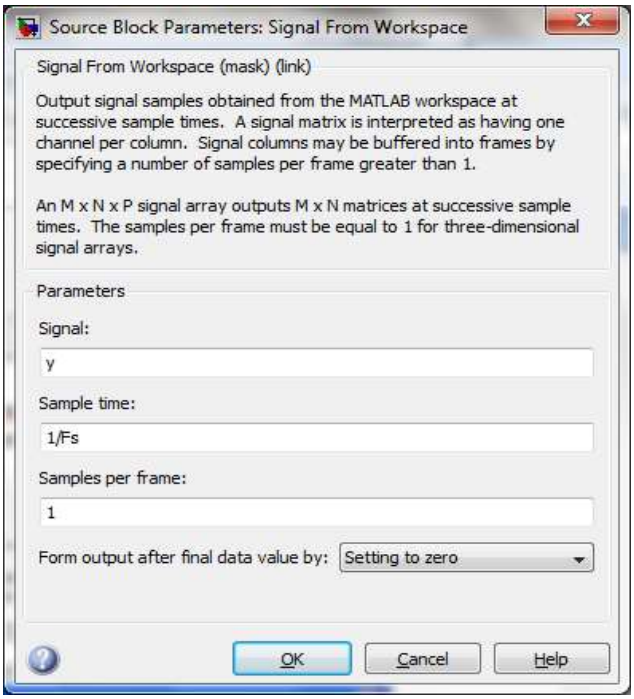


Figura 5. Recuadro de dialogo del bloque Signal From Workspace

Signal: Es el nombre de una variable en el área de trabajo de Matlab que contiene la señal a importar, o cualquier expresión válida de Matlab que especifique la señal. Para este caso, la señal utilizada es una señal de audio predefinida por Matlab denominada *handel*, la cual carga los datos en la variable 'y' y la frecuencia de muestreo en la variable 'Fs'. La variable 'y' tiene un tamaño 73113x1, con un rango de amplitudes que van desde -0.8 a 0.8

Sample time: es el período de muestreo de salida T_s de la señal. Para este caso el tiempo de muestreo es 1/Fs = 1/8192

Samples per frame: es el número de muestras Mo por cada trama de la señal. Para este caso las muestras por trama es 1.

Form output after final data value by: Especifica la salida después de que todas las muestras de la señal especificada han sido generadas. El bloque puede dar ceros de salida (`Setting to zero`), repetir la muestra final de datos (`Holding Final Value`) o repetir la señal entera desde el principio (`Cyclic Repetition`) si la señal es más corta que el tiempo de simulación. En esta práctica de laboratorio se escogió la forma `Setting to zero`.

- " *Convert* ": convierte la señal de entrada en un tipo de dato especifico. En este caso la conversión se hace a punto fijo.

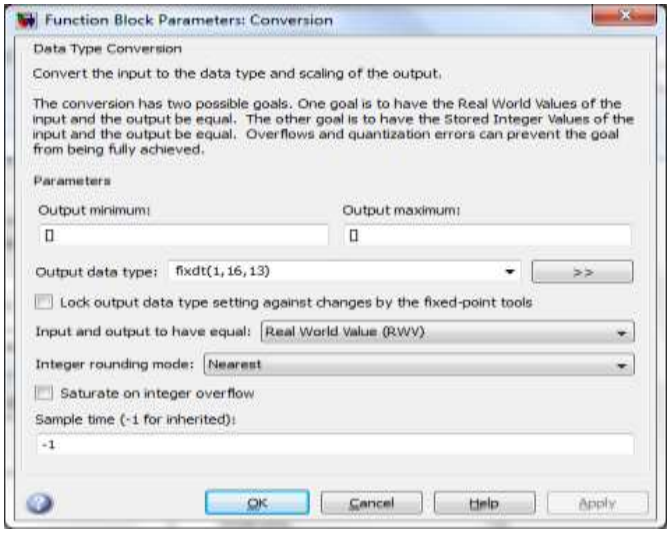


Figura 6. Recuadro de dialogo del bloque Convert

- " *Sum* " : Suma o resta señales de entrada. En este caso suma la señal de ruido del bloque *Random Source* con la señal de audio del bloque *Signal From Workspace*

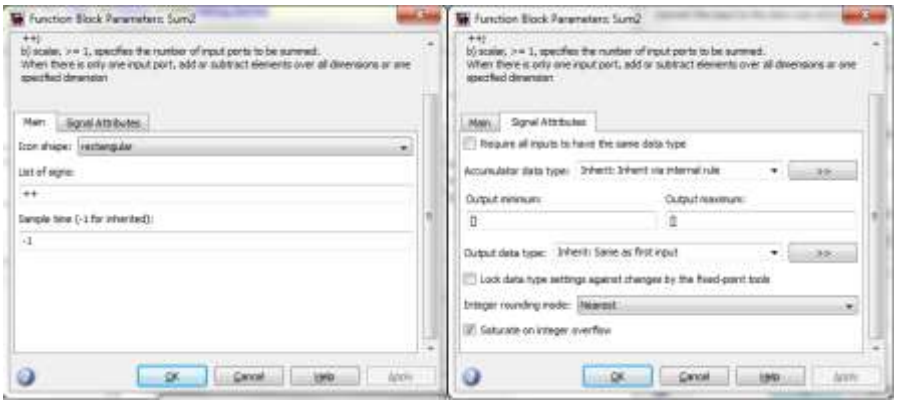


Figura 7. Recuadro de dialogo del bloque Sum

- ***" Constant " :*** este bloque debe ser siempre cero para que los pesos del filtro adaptativo se vayan actualizando.

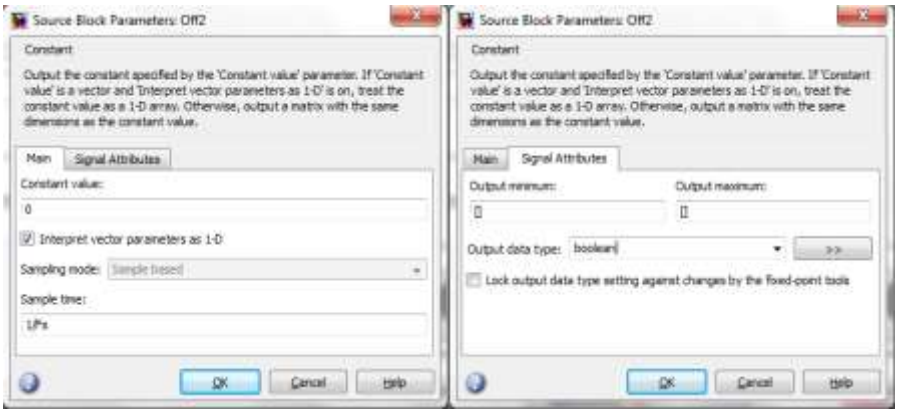


Figura 8. Recuadro de dialogo del bloque Constant

- ***" Embedded MATLAB Function Block " :*** Este bloque permite crear una función Matlab dentro de un modelo Simulink. Se puede especificar los datos de entrada y salida del bloque en la cabecera de la función como argumentos y valores de retorno. La función por defecto es *fcn* que tiene como argumento y valor de retorno *u* y *y* respectivamente, que se convierten en la entrada y salida del bloque por defecto. Este bloque soporta un subconjunto del lenguaje Matlab para el cual se puede generar código embebido eficiente. La lista de las funciones soportadas se encuentra en el *Help* de Matlab.

  Embedded Matlab Editor es el editor del bloque, donde se generan las funciones del algoritmo:

Figura 9. Ventana del Editor Embedded Matlab

El Proyecto *Vectorized Adaptive noise canceler using LMS Filter* se descarga en una carpeta de nombre lms_eml, que contiene el modelo Simulink y las funciones en Matlab del algoritmo LMS. La función principal del algoritmo denominada `lms_filter` que es la encargada de llamar a las demás funciones para que realizan los procesos matemáticos del algoritmo se encuentra en el modelo Simulink *eml_vectorized_lms_7b.mdl*, en el subsistema `lms_eml_hardware`, en el bloque *Embedded MATLAB Function*. Las funciones `tap_delay_fcn`, `treesum_fcn` y `update_weight_fcn` son las encargadas de realizar las sumas, los retrasos y la actualización de los coeficientes. De las funciones `lmsfilter y` se elimina como parámetro de entrada `step_size` y en la función `update_weight_fcn` se elimina como parámetro de entrada `step_size` y se copian las siguientes líneas al inicio de esta función:

```
persistent step_size;
step_size=fi(0.002);
```

Las funciones editadas en *Embedded Matlab Editor* se pueden observar en las siguientes tablas:

Proceso	Codigo Matlab
Se crea una función con nombre `lmsfilter`, tiene una salida `y` y tres entradas `input`, `desired` y `reset_weights`. `input` : la señal de ruido. `desired`: la señal que incluye la señal de audio original y la señal de ruido `reset_weights`: para reiniciar los pesos del filtro.	`function y = lmsfilter(input, desired, reset_weights)`
Indica que la función debe ser compila con *Embedded Matlab*.	`%#eml`
Crea un objeto de tipo fimath con las mismas características de `input`.	`fm = fimath(input);`

Se crea la variable `filter_coefficients`, si no contiene ningún valor esta variable se convierte en un objeto de punto fijo con valor ceros (1, 6), con signo (1), longitud de palabra de 16bits, longitud de la fracción de 16 bits y con las características del objeto `fm`.	`persistent filter_coefficients;` `if isempty(filter_coefficients)` `  filter_coefficients =` `fi(zeros(1, 6), 1, 16, 16, fm);` `end`
La función `tapped_delay_fcn` retrasa la señal y la devuelve en forma de vector.	`delayed_signal =` `tapped_delay_fcn(input);`
Se crea `weight_applied` como objeto de punto fijo, con valor `delayed_signal.*` `filter_coefficients`, con signo (1), longitud de palabra de 32bits, longitud de la fracción de 20 bits y con las características del objeto `fm`.	`weight_applied = fi(delayed_signal .* filter_coefficients, 1, 32, 20, fm);`
Se llama a la función `treesum_fcn` para sumar los resultados obtenidos al multiplicar la señal retrasada con los coeficientes del filtro y obtener la señal filtrada.	`filtered_signal =` `treesum_fcn(weight_applied);`
Se crea `td` como objeto de punto fijo, con valor `desired`, con signo (1), longitud de palabra de 16bits, longitud de la fracción de 13 bits y con las características del objeto `fm`. Esta variable representa a la señal deseada.	`td = fi(desired, 1, 16, 13, fm);`
Se crea `tf` como objeto de punto fijo con valor `filtered_signal`, con signo (1), longitud de palabra de 16bits, longitud de la fracción de 13 bits y con las características del objeto `fm`. Esta variable representa a la señal filtrada.	`tf = fi(filtered_signal,1,16,13, fm)`
$\mathrm{e}(n) = d(n) - y(n)$ Se crea `err_sig` como objeto de punto fijo, con valor `td - tf`, con signo (1), longitud de palabra de 16bits, longitud de la fracción de 13 bits y con las características del objeto `fm`. Esta variable representa a la señal de error.	`err_sig = fi(td-tf, 1, 16, 13, fm);`
Se asigna `err_sig` a la variable de salida del filtro	`y = err_sig;`
Se llama a la función `updated_weight_fcn1` para calcular los nuevos coeficientes del filtro.	`updated_weight =` `update_weight_fcn(err_sig,` `delayed_signal,` `filter_coefficients,` `reset_weights);`
Se actualizan los coeficientes del filtro	`filter_coefficients =` `updated_weight;`

Tabla 1. Ecuaciones de la función lmsfilter

Proceso	Codigo Matlab
Se crea una función con nombre `tapped_delay_fcn`, tiene una salida `tap_delay` y una entrada `input`. `input` representa a la señal de entrada del filtro	`function tap_delay = tapped_delay_fcn(input)`
Se crea la variable `u_d`, si no contiene ningún valor esta variable se convierte en un objeto de punto fijo con valor ceros (1, 6), con tipo numérico igual al de `input` y con las características del objeto `fimath(input)`	`persistent u_d;` `if isempty(u_d)` `  u_d = fi(zeros(1,6),` `numerictype(input),` `fimath(input));` `end`
Se copia `input` en la posición 1 y se desplaza sucesivamente.	`u_d = [u_d(2:6), input]`
Se hace `u_d` igual a la variable de retorno de la función `tap_delay`	`tap_delay = u_d;`

Tabla 2. Ecuaciones de la función tap_delay

Proceso	Codigo Matlab
Se crea una función con nombre `treesum_fcn`, tiene una salida `y` y una entrada `u`. `u` representa a la variable pesos aplicados. Esta función hace uso de la función `vsum`	`function y = treesum_fcn(u)`
Se suman todos los valores del vector de entrada, hasta que el resultado es un vector de una sola posición. En este caso como el filtro tiene una longitud de 6 coeficientes entonces son necesarios 3 niveles.	`level1 = vsum(u);` `level2 = vsum(level1);` `level3 = vsum(level2);`
Se hace `level3` igual a la variable de retorno de la función `y`	`y = level3;`
Se crea una función con nombre `vsum`, tiene una salida `output` y una entrada `input`. `input` representa a la `levelx`, la cual le es enviada por la función `treesum_fcn`.	`function output = vsum(input)`
Se crea `nt`, variable que guarda el tipo numérico de `input` y `fm representa` objeto de tipo `fimath` con las mismas características de `input`	`nt = numerictype(input);` `fm = fimath(input);` .
`nd` contiene la mitad del número de elementos de `input`	`nd = numel(input)/2`
`vt` contiene las posiciones impares de input	`vt = input(1:2:end)`
En este ciclo se suman las posiciones del vector de entrada por parejas, entonces la variable de salida contendrá la mitad del	`for i = int32(1:nd)` `  k = int32(i*2);` `  vt(i) = fi(vt(i) + input(k),`

número de elementos que contenía la entrada	`nt, fm);` `end`
Se hace `vt` igual a la variable de retorno de la función `output`	`output = vt;`

Tabla 3. Ecuaciones de la función treesum_fcn

Proceso	Codigo Matlab
Se crea una función con nombre `update_weight_fcn1`, tiene una salida `weights` y cuatro entradas `err_sig, delayed_signal, filter_coefficients, reset_weights`	`function weights = update_weight_fcn(err_sig, delayed_signal, filter_coefficients, reset_weights)`
`persistent` crea una variable local, su valores son mantenidos en memoria entre las llamadas a la función	`persistent step_size;` `step_size=fi(0.002);`
`fimath(input)` crea un objeto de tipo `fimath` con las mismas características de `step_size`.	`fm = fimath(step_size);`
$step\_sig = \mu e(n)$ $correction\_factor = step\_sig * x(n)$ $\mathrm{w}(n+1) = \mathrm{w}(n) + \mu x(n) e(n)$ Se calculan los nuevos coeficientes del filtro `updated_weight` con ayuda de las variables `step_sig y correction_factor` y la variable de entrada `filter_coefficients`. Las estas variables temp y updated_weight son representadas como objetos de punto fijo, con diferentes valores cada uno, con signo(1), longitud de palabra de 16bits, longitud de la fracción de 13 bits y con las características del objeto `fm`.	`step_sig = fi(step_size .* err_sig, 1, 32, 20, fm);` `correction_factor = fi(delayed_signal .* step_sig, 1, 32, 20, fm);` `updated_weight = fi(correction_factor + filter_coefficients, 1, 16, 16, fm);`
Se comprueba la variable de entrada `reset_weights`. Si esta en alto esta variable se convierte en un objeto de punto fijo con valor ceros (1, 40), con signo (1), longitud de palabra de 16bits, longitud de la fracción de 16 bits y con las características del objeto `fm`. Si esta en bajo se hace `updated_weight` igual a la variable de retorno de la función `weights`.	`if reset_weights` `  weights = fi(zeros(1,40), 1, 16, 16, fm);` `else` `  weights = updated_weight;` `end`

Tabla 4. Ecuaciones de la función update_weight_fcn

Al cerrar el editor, el bloque *Embedded MATLAB Function* se configura para tener las mismas entradas y salidas que las de la función `lmsfilter`.

Al realizar las conexiones que muestra la siguiente figura, el modelo está listo para ser simulado dando click en *Start Simulation*

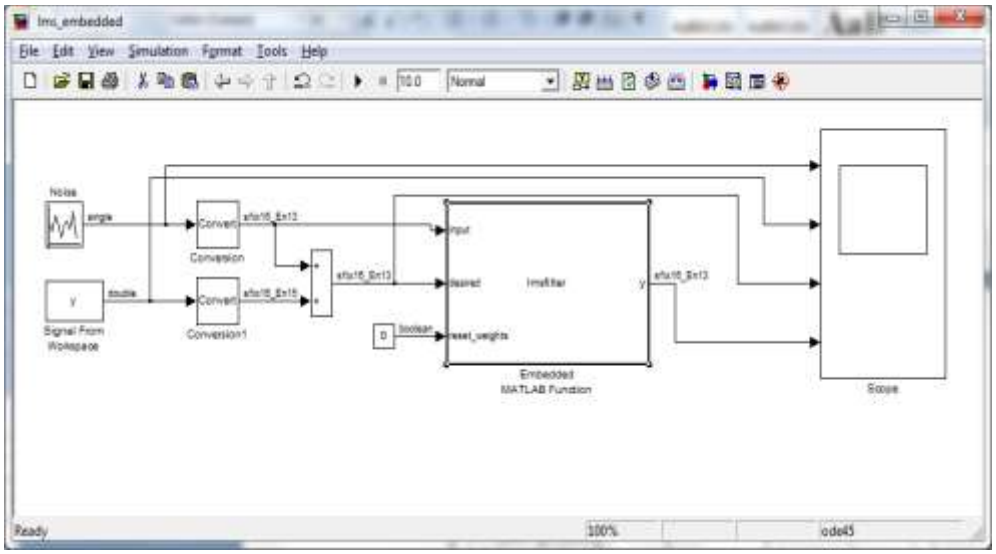


Figura 10. Modelo final de Simulink para algoritmo LMS

b. Análisis del desempeño del Algoritmo LMS: Para observar las señales que intervienen en el diseño, el osciloscopio se configura para tener 4 entradas: la primera es la *señal de ruido*, la segunda es la *señal de audio*, la tercera es la *señal de audio con ruido* y la cuarta es la *señal de salida* del filtro LMS.

Para escuchar las señales que intervienen en el diseño se realizan las siguientes configuraciones:

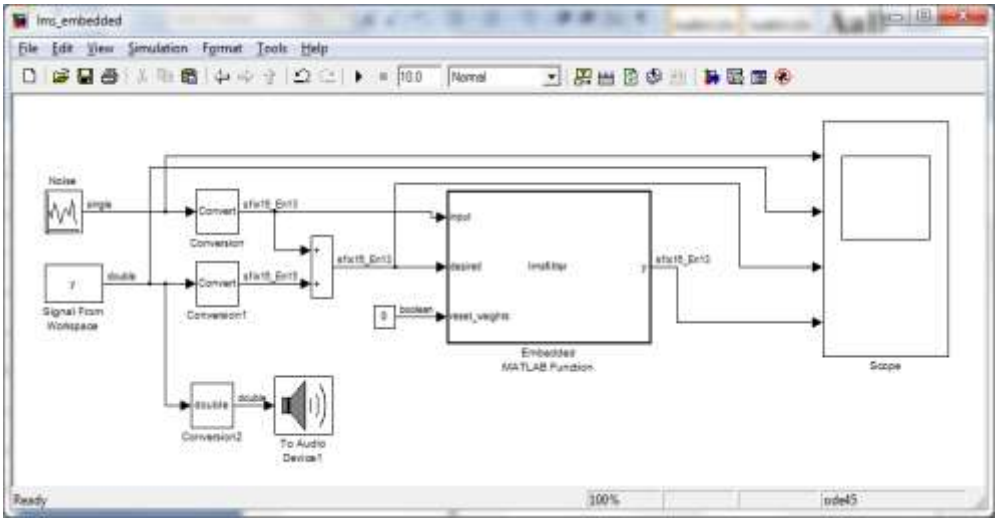


Figura 11. Configuración del Modelo Simulink para escuchar la señal de audio original

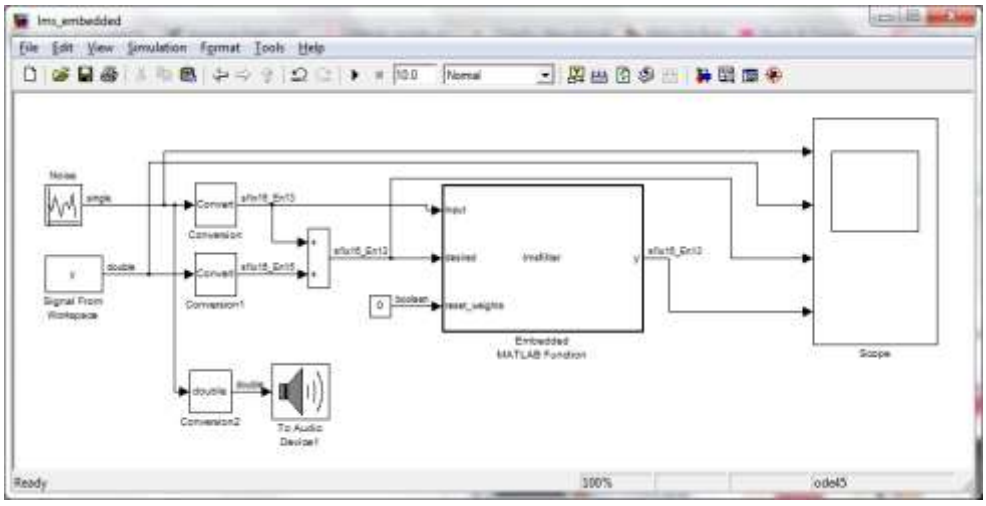


Figura 12. Configuración del Modelo en Simulink para escuchar la señal de ruido

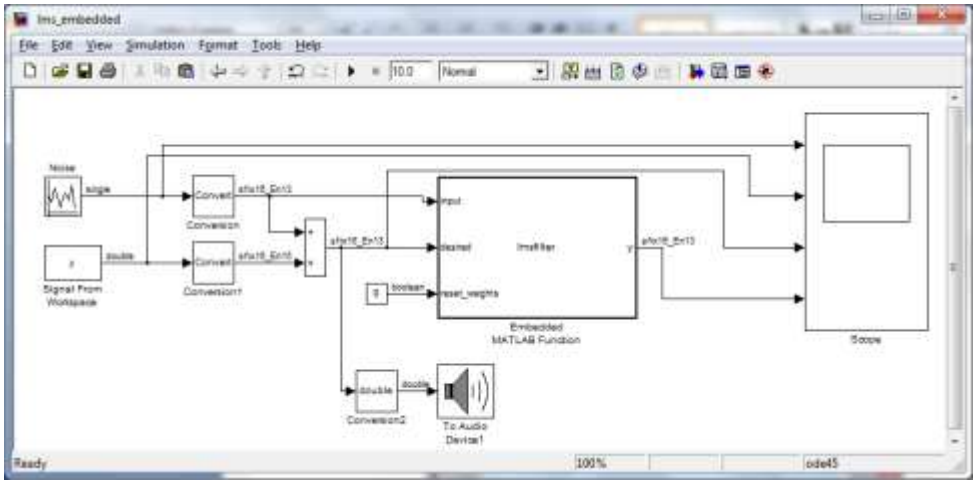


Figura 13. Configuración del Modelo en Simulink para escuchar la señal de audio contaminada

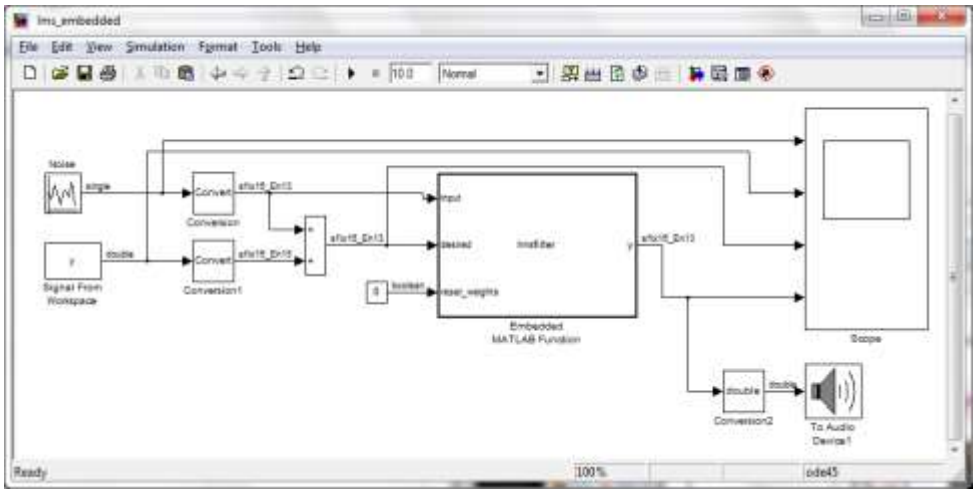


Figura 14. Configuración del Modelo en Simulink para escuchar la señal de salida

6. PROCEDIMIENTO DE LABORATORIO PARA ALGORTIMO LMS

- Software: Matlab, Simulink
- Equipo: parlantes, computador

6.1 Diseño y simulación de filtro LMS: Realizar los cambios en los parámetros que indique el profesor

6.2 Resultados:

- Según los resultados del osciloscopio, ¿cómo es la señal de audio comparada con la señal de audio con ruido?

  _____

  _____

- Según los resultados del osciloscopio, ¿cómo es la señal de audio contaminada con ruido comparada con la señal de salida? Concluya.

  _____

  _____

- Según los resultados auditivos, ¿cómo es la señal de audio contaminada respecto a la señal de salida del filtro?

  _____

  _____

- ¿Qué resultados se obtienen en la señal de salida al cambiar el nivel de potencia del ruido?

  _____

  _____

- ¿Considera que el trabajo efectuado por el algoritmo es útil? Justifique

  _____

  _____

  _____

7. TRABAJO COMPLEMENTARIO

Disminuir el número de coeficientes del filtro. Observar que ocurre con los resultados mostrados en el osciloscopio.

8. REPORTE

El estudiante debe entregar las respuestas de ítem Resultados, junto con el diseño en Matlab(.m) y el modelo en Simulink(.mdl) del algoritmo LMS diseñado y del Trabajo Complementario

9. BIBLIOGRAFIA

[1] HAIKIN S. ADAPTIVE FILTER THEORY. Tercera Edicion

[2] B.FARHANG-BOROUJENY. ADAPTIVE FILTERS, THEORY AND APPLICATIONS - National University of Singapore

[3]

[4] DIAZ P. FERNANDEZ P. LA DISTRIBUCION NORMAL. Unidad de Epidemiologia Clínica y Bioestadística. Complexo Hospitalario Juan Canalejo-2001

PRACTICA No 5. IMPLEMENTACION DE FILTROS DIGITALES FIR POR METODO DE VENTANAS, FIR RIZADO CONSTANTE, IIR BUTTERWORTH Y ADAPTATIVO SOBRE FPGA SPARTAN 3A.

1. OBJETIVO

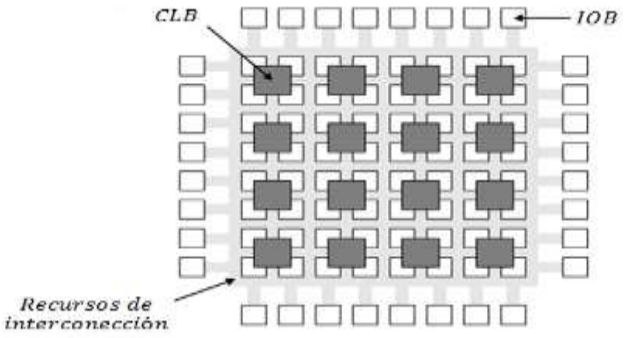
Implementar diseños de Filtros Digitales tipo FIR, IIR y Adaptativo diseñados y simulados en Matlab®, para observar su funcionamiento en FPGA 3A Xilinx®

2. INTRODUCCION

En esta práctica se presentan las características básicas de la FPGA Spartan 3A y los componentes que se utilizan en el desarrollo de la práctica. Más adelante, se explica por medio de un ejemplo práctico, cómo adecuar los modelos obtenidos en Simulink® en las cuatro prácticas anteriores para generar a partir de ellos el código VHDL, que será implementado en la FPGA. El proceso de laboratorio desarrollado por el estudiante puede seguir como guía el ejemplo práctico enunciado anteriormente para adecuar los diseños que se desee implementar.

3. MARCO TEORICO

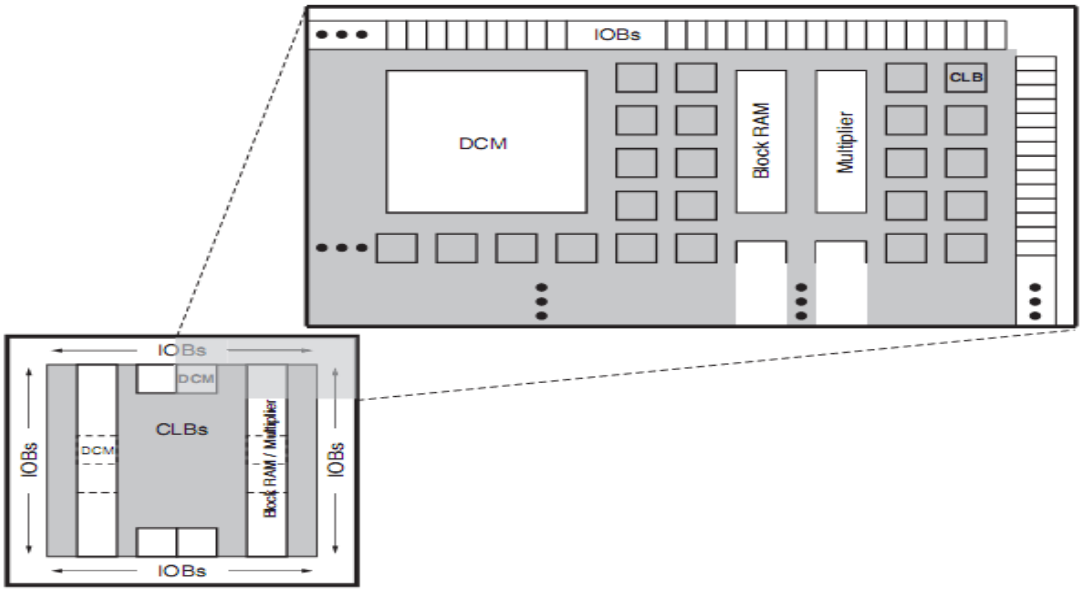
La FPGA es un dispositivo electrónico digital programable de muy alta escala de integración. Consiste en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad, lo que se programa en una FPGA son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques [1].



Gráfica 1. Estructura General de una FPGA

La compañía Xilinx contiene una amplia gama de familias de FPGA, entre las cuales se destacan la Virtex y la Spartan.
En esta ocasión se utilizará una Tarjeta FPGA de la familia Spartan 3A.
La arquitectura de la FPGA Spartan 3A consiste en 5 elementos funcionales programables:

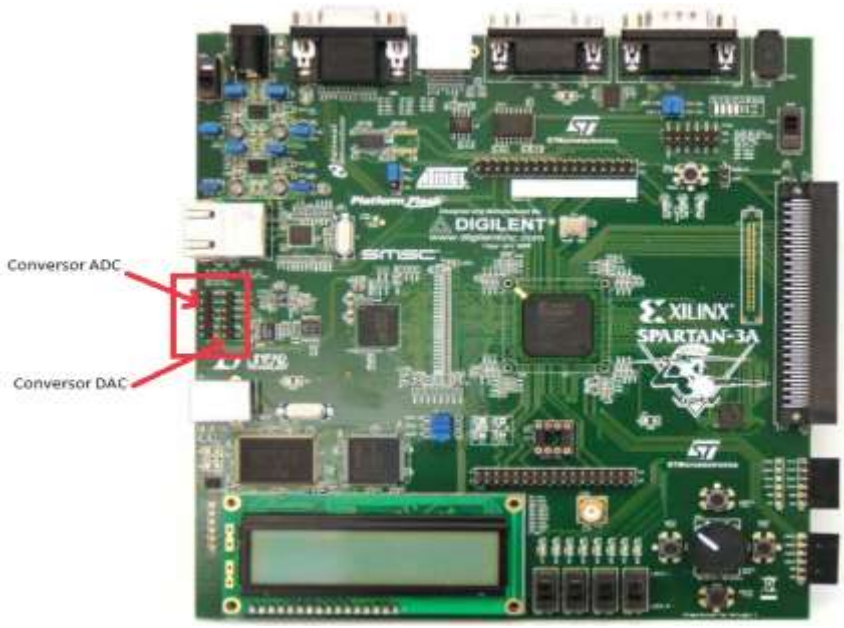
- CLB: los bloques lógicos configurables desarrollan una amplia variedad de funciones lógicas como de almacenamiento de datos.
- IOB: los bloques de entrada/salida controlan el flujo de datos entre los pines de entrada y salida y la lógica interna del dispositivo. Estos soportan flujo de datos bidireccional.
- Bloque RAM: provee almacenamiento de datos en los bloques de puerto dual de 18Kbits
- Bloques Multiplicadores: aceptan dos números binarios de 18 bits como entrada y calculan el producto.
- Bloque manejador del reloj digital (DCM): provee su propia calibración y soluciones para la distribución, retardo, multiplicación, división y desplazamiento en fase de las señales de reloj [2].



Gráfica 2. Arquitectura de FPGA Spartan 3A

Los componentes de la tarjeta que se utilizan en la práctica son:

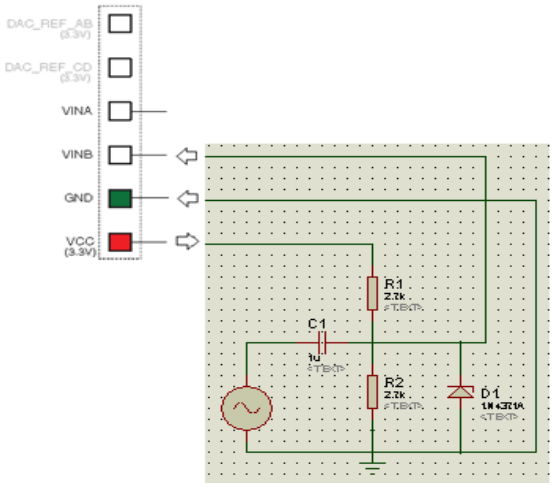
- Conversor ADC: el circuito de captura analógica de dos canales, está compuesto por un pre-amplificador programable a escala LTC6912-1 que provee dos amplificadores independientes con ganancia programable, la salida del preamplificador es conectada a un conversor análogo digital LTC1407A-1, la salida del conversor es una representación de 14 bits con complemento a dos de la señal de entrada. Por consiguiente se pueden representar valores entre -2^{13} y $2^{13}-1$, la máxima taza de muestreo es aproximadamente 1.5MHz.
- Conversor DAC: El Circuito de conversión digital analógico es un LTC2624, de cuatro canales seriales, con resolución de 12-bit sin signo cada uno. El circuito DAC y sus salidas se muestran en la figura 3



Gráfica 3. FPGA Spartan 3A y los componentes a utilizar en la práctica [3].

Además se hace necesario una etapa de adaptación para evitar que los valores de entrada al conversor ADC no excedan los límites establecidos de voltaje. Estos límites de voltaje de entrada dependen de la ganancia del amplificador.

La ganancia de este amplificador es programable de -1 a -100. Para el caso de las prácticas la ganancia escogida será -1, por lo tanto los valores de voltaje deben estar entre 2,9 y 0,4 v. Según las especificaciones de *Spartan-3A FPGA Starter Kit Board User Guide* se debe adicionar un nivel DC de 1,65 a la señal alterna de entrada, de tal manera que los límites de voltaje para esta ganancia específica sean 1.65±1.25.

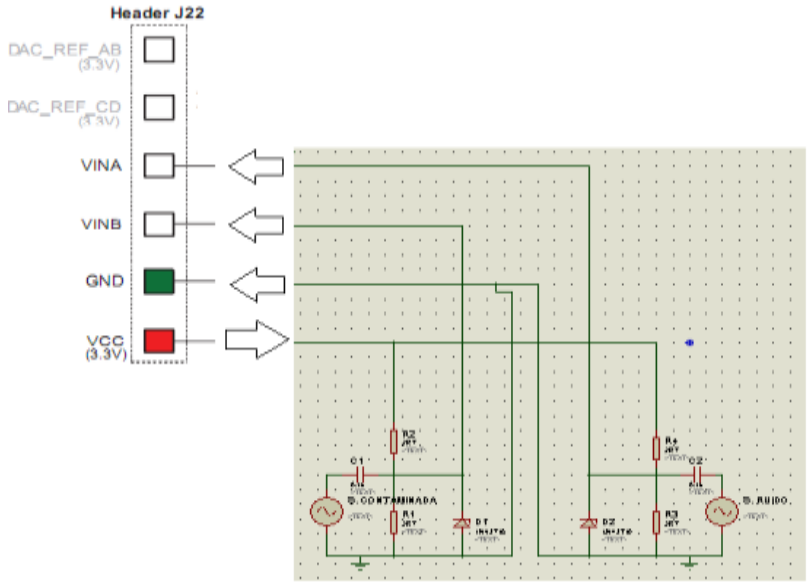


Gráfica 4. Diagrama Circuital de Etapa de Adaptación para filtro FIR e IIR

El circuito de la grafica 4 realiza un divisor del voltaje de referencia de 3.3v, que sale del pin VCC del conversor ADC de la FPGA, para obtener 1.65v y sumarlos a la señal alterna de entrada. Luego un diodo Zener recorta los niveles de voltaje de esta señal que estén por encima de los permitidos y la envía al canal 2 del conversor análogo-digital (VINB).

Sin embargo, en señales de audio, el hecho de recortar los picos de la señal implica perdida de datos perceptibles al oído, por lo tanto al escuchar vacios en la señal de salida de los conversores, se debe disminuir el volumen con que sale del computador, para disminuir las amplitudes de voltaje de entrada al ADC y que de esta manera no sean recortadas por el Zener.

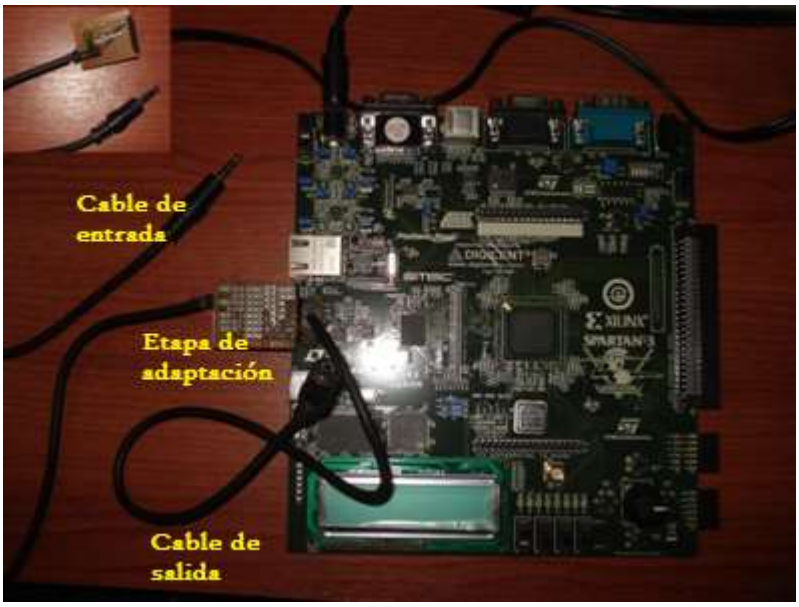
Para el filtro Adaptativo LMS que utiliza dos señales de entrada a la FPGA, se diseñó la etapa de adaptación mostrada en la figura 5, la cual realiza el mismo proceso de adaptación de los voltajes de la señal explicado anteriormente, para cada una de las dos señales de entrada.



Gráfica 5. Diagrama circuital de etapa de adaptación para filtro Adaptativo LMS

En la figura 5, la señal de ruido entra por el canal 1 (VINA) y la señal contaminada entra por el canal 2(VINB) del conversor ADC de la FPGA.

La siguiente figura 6, muestra una foto tomada a la FPGA integrada con la etapa de adaptación, donde se observa el cable de entrada de audio y el cable de salida de la FPGA a los parlantes:



Gráfica 6. Etapa de Adaptación integrada a la FPGA

La siguiente grafica 7 muestra como se debe conectar el cable de salida a la FPGA. El cable de entrada está sujeto a la etapa de adaptación y esta solo tiene una forma de ser conectada a la FPGA como se muestra en la grafica 6.

Gráfica 7. Conexión del Cable de salida a la FPGA

4. TRABAJO PREVIO

El estudiante debe fundamentarse en los siguientes temas para poder alcanzar los objetivos de esta práctica:

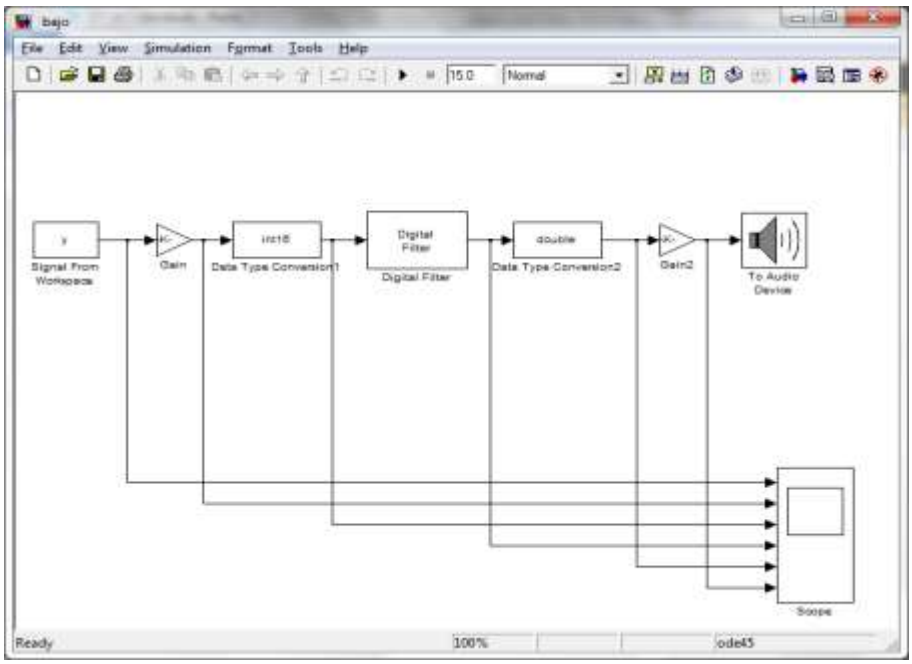
- Manual de usuario para la implementación de filtros digitales en la FPGA Spartan 3A de Xilinx
- Filtros Digitales FIR y Adaptativos
- Lenguaje VHDL
- Comando Básicos Matlab®
- Aritmética de Punto Flotante y Punto Fijo
- Entorno Simulink®

5. EJEMPLO PRÁCTICO: Implementación de filtro diseñado con ventana Hamming

Mediante los siguientes pasos se implementa el modelo en Simulink® del filtro Elimina Banda expuesto como Ejemplo Práctico (ítem 5) de la Practica No.1: "Simulación de filtros digitales tipo FIR por el método de ventana Kaiser y por otro método de ventana".

a. Conversión a modelo en punto fijo: El modelo obtenido en la práctica No.1 realiza las operaciones en aritmética de Punto Flotante, y estas deben ser llevadas a aritmética de punto fijo ya que la generación de código VHDL a partir de un modelo en punto flotante no es concurrente con las restricciones impuestas por Xilinx en el XST[1]. Además la aritmética de punto fijo es más fácil de implementar.

Para convertir el modelo de punto flotante a punto fijo, se adicionan los bloques *Convert* y *Gain* en el diseño final obtenido en la Práctica1 así:



Gráfica 8. Modelo en Simulink® a implementar

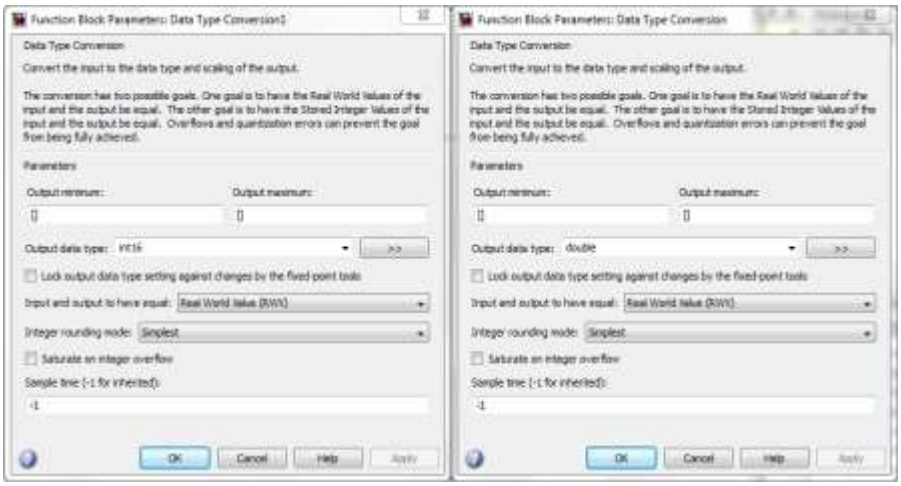
El bloque *Convert* es el encargado de convertir la señal de entrada a punto fijo; luego al configurar la pestaña *Fixed Point* del bloque *Digital Filter,* éste se encargará de convertir los coeficientes al mismo tipo de dato de la entrada que le está llegando, es decir entrada en punto fijo. En este caso las variables fueron convertidas a int16[2].

[1] XST-Xilinx Synthesis Technology : La Tecnología de Síntesis Xilinx es una herramienta de Xilinx, que sintetiza diseños HDL para crear archivos netlist específicos de Xilinx llamados NGC. El archivo NGC es una lista de red que contiene los datos de diseño lógico y restricciones.

[2] int16: convierte un elemento a un entero de 16 bits con signo, el rango de valores de salida es de -32768 a 32767.

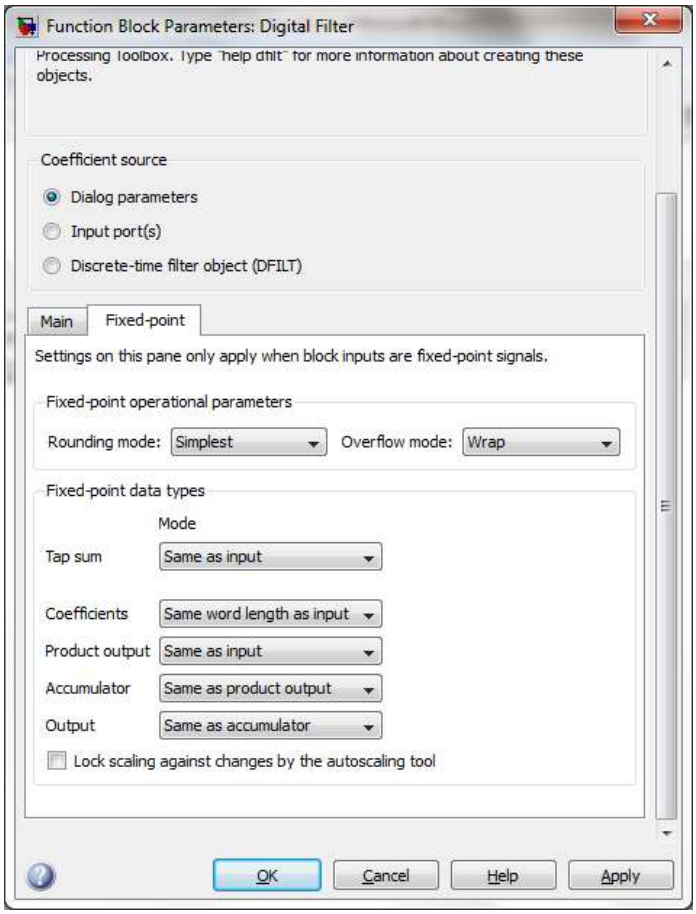
Cuando se tiene la señal filtrada, se utiliza nuevamente el bloque *Convert,* para convertir las variables de int16 a tipo *double*[3] y así poder escuchar la señal filtrada en el bloque *To Audio Device*

Se debe tener en cuenta que cuando la señal sale del conversor ADC está representada por una palabra digital de 14bits de longitud, 1 bit para signo, al llegar al filtro es necesario adicionarle dos bits "00" a la izquierda para completar la palabra de 16 bits con la que trabaja el filtro en la FPGA. Dado que el conversor DAC trabaja con 12 bits y la salida del filtro es una palabra de 16 bits se hace necesario remover 4 bits, para que el conversor pueda trabajar. Los bits removidos son los bits más significativos.



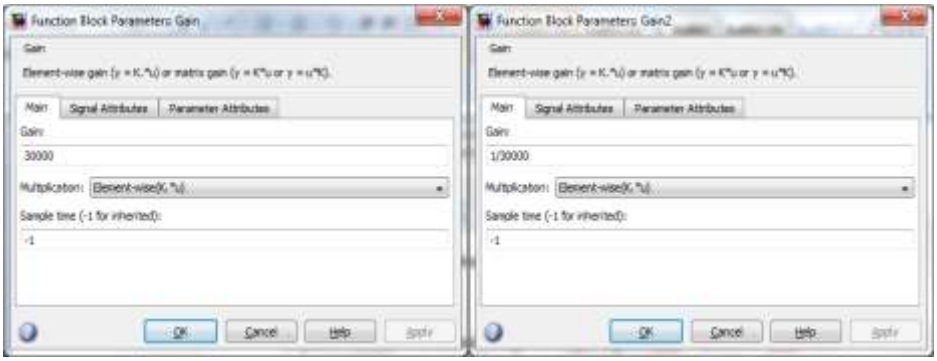
Gráfica 9. Recuadro de Dialogo del bloque Convert de entrada y de salida respectivamente

[3] double: tipo de dato de doble precisión.



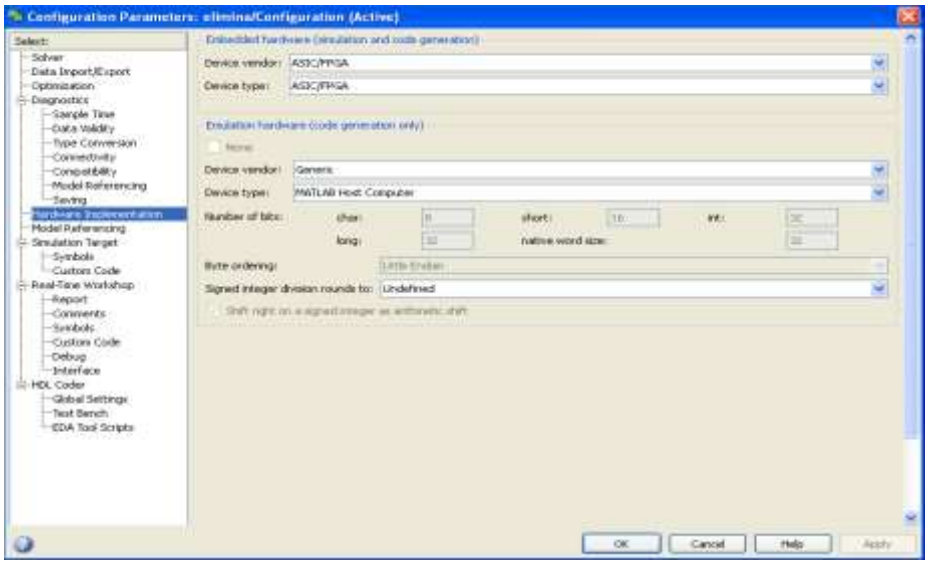
Gráfica 10. Recuadro de Dialogo de bloque Digital Filter. Pestaña de Fixed Point

El bloque *Gain* es necesario ya que las amplitudes de la señal cargada en el bloque *Signal from Workspace* tienen como máximo valor ∓ 0.8, lo cual es un inconveniente, ya que para convertir esta señal a *int16* se aproximan los valores al entero más cercano dependiendo del método de redondeo utilizado en bloque *Convert,* es decir que esas amplitudes se aproximaran a 0 o a 1. Esto provocará una simulación deficiente, por lo que es necesario aplicar una ganancia a esta señal antes de convertirla. La ganancia aplicada es 30000:



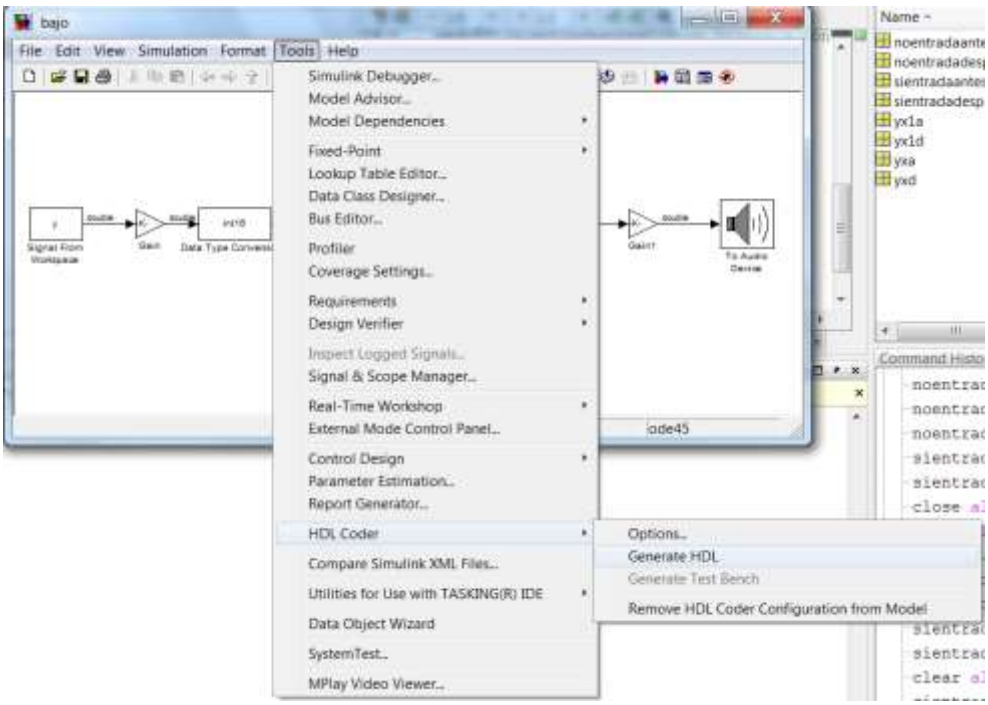
Gráfica 11. Recuadro de Dialogo de bloque Gain de entrada y de salida respectivamente

Finalmente, antes de generar el código VHDL, se define el dispositivo en el cual se va a implementar el diseño, en la ruta: *Simulation=>ConfigureParameters=>Hardware Implementation*



Gráfica 12 Asignación de Hardware para la implementación

***b. Generación de código VHDL*:** Luego de tener el modelo en Simulink® en punto fijo, se genera el código VHDL automáticamente utilizando la herramienta *Simulink HDL Coder* así:



Gráfica 13. Generación de código VHDL con Simulink HDL Coder

c. Integración de los códigos generados con los conversores ADC y DAC: La generación de un nuevo proyecto y los primeros pasos con la herramienta ISE de

Xilinx debe ser consultados en el *Manual de usuario para la implementación de filtros digitales en la FPGA Spartan 3A de Xilinx*.

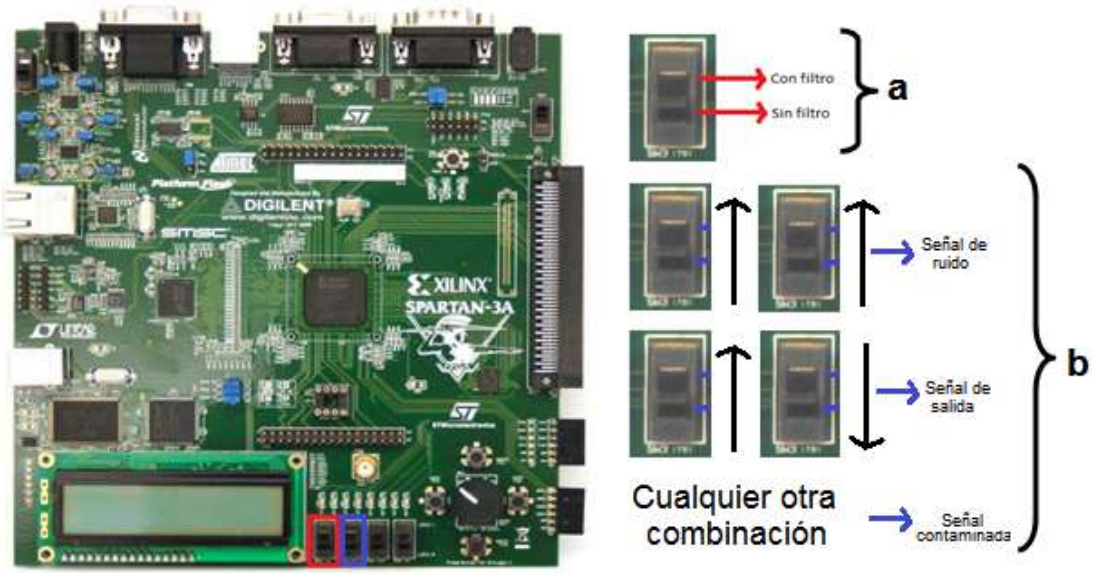
Por otra parte Simulink HDL Coder genera tres archivos del modelo Filtro Elimina Banda Hamming, de estos, el único necesario, es llamado Digital_Filter.vhd, que contiene la declaración de la entidad, es decir de qué tipo y cuáles son los puertos de entrada y salida del filtro y además el comportamiento del filtro mediante retrasos, multiplicaciones y sumas. La integración de este código con los demás archivos que contiene el proyecto, debe también consultarse en el Manual de Usuario.

d. ***Implementación del código:*** Para implementar el código de todo el proyecto se debe seguir la indicaciones recomendadas en el *Manual de usuario para la implementación de filtros digitales en la FPGA Spartan 3A de Xilinx*:

e. ***Análisis del desempeño:*** en este paso se escuchan y observan los resultados obtenidos en tiempo real y se compararan con los resultados de simulación. Para poder escuchar las señales en diferentes puntos es hace uso de algunos interruptores como se muestra a continuación.

Para el caso de los filtros selectivos en frecuencias trabajados en las tres primeras guías se hace uso del interruptor encerrado con color rojo, para escuchar la señal pasando y sin pasar por el filtro; como se ve en la figura 14.a.

En el caso de filtrado adaptativo se hacen uso de los dos interruptores, encerrados con rojo y azul, como se ve en la figura 14.b:



Gráfica 14. Interruptor de activación del filtro (rojo) e interruptor para escuchar señal de ruido y señal de audio contaminada (azul)

En este caso, para el filtro Hamming, solo se utiliza el interruptor en rojo para escuchar las señales sin filtrar y filtrada y capturarlas por medio de la función

`wavrecord(M, Fs)` de Matlab® donde `M` es el numero de muestras q en este caso serán 100000 y `Fs` es la frecuencia de muestreo que en este caso es 11025. Deben estar sincronizados lo mejor posible, el instante en el que se manda la señal de audio a la FPGA y el instante en el que se ejecuta la función `wavrecord` en Matlab®.

Para el caso de este ejemplo práctico, se diseño el código mostrado más adelante para realizar las siguientes capturas:

- `sinfilsinaud =wavrecord(100000,11025)` es la captura de la salida sin señal de audio y sin filtro.
- `confilsinaud =wavrecord(100000,11025)` es la captura de la salida sin señal de audio con filtro.
- `sinfilconaud =wavrecord(100000,11025)` es la captura de la señal de salida con señal de audio y sin filtro.
- `confilconaud= wavrecord(100000,11025)` es la captura de la señal de salida con señal de audio con filtro.

Se aplica el siguiente código para graficar los resultados de la implementación:

```
%%
figure(1)
subplot(2, 2, 1)
plot(sinfilconaud);title('Señal de Salida del filtro con señal
de audio y sin filtro')
subplot(2, 2, 2)
plot(confilconaud);title('Señal de Salida del filtro con señal
de audio y con filtro')
subplot(2, 2, 3)
inx=fft(sinfilconaud);
plot(abs(inx));title('Espectro de Señal de Salida del filtro
con señal de audio y sin filtro')
subplot(2, 2, 4)
outx=fft(confilconaud);
plot(abs(outx));title('Espectro de Señal de Salida del filtro
con señal de audio y con filtro')
%%
figure(2)
subplot(2, 2, 1)
plot(sinfilsinaud);title('Salida del filtro sin señal de audio
y sin filtro')
subplot(2, 2, 2)
plot(confilsinaud);title('Salida del filtro sin señal de audio
y con filtro')
subplot(2, 2, 3)
inx1=fft(sinfilsinaud);
plot(abs(inx1));title('Espectro de Salida del filtro sin señal
de audio y sin filtro')
subplot(2, 2, 4)
outx1=fft(confilsinaud);
plot(abs(outx1));title('Espectro de Salida del filtro sin
señal de audio y con filtro')
```

6. PROCEDIMIENTO DE LABORATORIO PARA IMPLEMENTAR LOS MODELOS SIMULINK® DE LOS FITLROS.

- Software: Matlab®, ISE 12.1
- Equipo: FPGA, parlantes, circuito Adaptador, computador

Los modelos Simulink® a implementar serán definidos por el profesor:_____
_____
_____

6.1 Conversión a modelos de punto fijo: Adicionar los bloques *Convert* y *Gain* siguiendo la grafica 8. Configurar la pestaña *Fixed Point* de bloque *Digital Filter* de acuerdo a la grafica 10. Se debe recordar que el modelo del filtro Adaptativo LMS obtenido en la Practica No.4 ya es un modelo en punto fijo y no requiere este paso.

6.2 Generación de código VHDL: seguir la grafica 13

6.3 Integración de los códigos generados con los conversores ADC y DAC: para realizar este paso se debe seguir como guía el *Manual de usuario para la implementación de filtros digitales en la FPGA Spartan 3A de Xilinx*

6.4 Implementación del código : para realizar este paso se debe seguir como guía el *Manual de usuario para la implementación de filtros digitales en la FPGA Spartan 3A de Xilinx*

6.5 Análisis del desempeño: Capturar las señales que defina el profesor y graficar los resultados.

6.6 Resultados:

- **¿Cómo son los resultados auditivos obtenidos en la simulación comparados con los resultados auditivos obtenidos en la implementación?**
  _____
  _____
- **En la implementación, ¿Cómo es la señal de audio a la salida sin pasar por el filtro? ¿Cómo es después de pasar por el filtro?**
  _____
  _____
  _____
- **¿Cumple con los objetivos de diseño este filtro?**
  _____
  _____
  _____
- **Compara los resultados gráficos obtenidos en la simulación, con los resultados gráficos obtenidos en la implementación. ¿Que concluye?**
  _____
  _____
  _____

7. REPORTE

El estudiante debe mostrar la implementación de los filtros y entregar la solución de todas las preguntas anteriores

8. REFERENCIAS:

[1] VALLEJO M. AYALA J. FPGA: NOCIONES BASICAS E IMPLEMENTACION. Departamento de Ingeniería Electrónica. Universidad Politécnica de Madrid- Abril 2004.

[2] SPARTAN-3A FPGA FAMILY: DATASHEET- Marzo 6, 2009

[3] SPARTAN-3A FPGA STARTED KIT BOARD USER GUIDE. For Revision C Board - Junio 21, 2007