UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN TECNOLOGÍAS DE INFORMACIÓN Y SISTEMAS

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

PROCESAMIENTO DE VOZ EN TIEMPO REAL EMPLEANDO UN PROCESADOR DIGITAL DE SEÑALES

T E S I S

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

PRESENTA:

JOSÉ MANUEL MARTÍNEZ OLGUÍN

ASESORES:

DR. LUIS ENRIQUE RAMOS VELASCO

ING. FRANCISCO MORALES JIMÉNEZ

PACHUCA HGO., MÉXICO 13 DE MARZO DE 2008

José Manuel Martínez Olguín

Procesamiento de voz en tiempo real empleando un procesador digital de señales

El presente trabajo, es una meta realizada,
esta dedicado a mis padres y hermanas,
quienes me han dado su cariño y comprensión,
me han dado los medios para concluir
de manera exitosa todos mis proyectos.
Por todo eso y mucho más, gracias.

Gracias a:

Doctor Luis Enrique por compartirme sus amplios conocimientos y experiencias en procesamiento de señales y teoría de control.

Ingeniero Francisco por apoyarme en la elaboración de esta tesis y aportar conocimientos valiosos en el área de la electrónica.

M. en C. María Angélica por aportar sus ideas en la elaboración de la tesis.

UAEH por los proyectos del programa anual de investigación con número PAI-31A, y al proyecto PROMEP P/CA-11 2006-14-18.

Resumen

Procesamiento de voz en tiempo real empleando un procesador digital de señales

En el presente trabajo de tesis se analiza la señal de voz, en tiempo real, su conversión del dominio del tiempo al dominio de la frecuencia con un procesador digital de señales, para esto, se utiliza un micrófono para sensar la señal de voz y obtener esta variable.

La señal de voz obtenida por el micrófono es procesada en tiempo real para obtener su espectro de frecuencia, con el fin de caracterizarla y construir un filtro tipo FIR.

El procesamiento de la voz, la transformada rápida de Fourier, se programan en un Procesador Digital de Señales (DSP) de la familia TMS320C6711 de Texas Instruments.

Índice general

Dedicatoria III
Resumen VII
Índice general IX
Índice de figuras XII

1. Introducción **1**
1.1. Planteamiento del problema 2
1.2. Propuesta de solución 2
1.3. Objetivos 3
1.4. Justificación 3
1.5. Antecedentes 3
1.6. Cuerpo de trabajo de tesis 4

2. Señales de sonido y voz **7**
2.1. Introducción 7
2.2. La voz 8
2.3. El sonido 12
2.4. Micrófono 13
2.5. Proceso de conversion analógica a digital 15
2.6. Sistemas 15
2.6.1. Clasificación de los sistemas discretos 16
2.7. Teoremas 19
2.7.1. Convolución 19
2.7.2. Convolución discreta 20
2.7.2.1. Suma de convolución. 20
2.8. Comentarios y referencias 22

3. Simulaciones **27**
3.0.1. Introducción 27
3.1. Herramientas definidas por MatLab 29
3.1.1. Funciones definidas por MatLab 30

3.2. Resultados de las simulaciones 30
3.2.1. Lectura de un archivo de voz 30
3.2.2. Realización ventaneo 32
3.2.2.1. Tipos de ventanas 32
3.2.2.2. Secuencia de eventos (ventaneo) 34
3.2.3. Normalización de resultados 38
3.2.4. Obtención de la transformada rápida de Fourier 38
3.3. Simulación: filtros 39
3.3.1. Generación del ruido de alta frecuencia 43
3.3.2. Señal muestreada 44
3.3.3. Suma de señales 45
3.3.4. Obtención señal filtrada 46
3.4. Comentarios y referencias 47

4. Implantación **49**
4.1. Introducción 49
4.2. Configuración de puertos 50
4.3. Proceso de implantación 52
4.4. Funciones definidas en el code composer studio(CCS) 52
4.4.1. Implantación de la FFT base-2 53
4.5. Comentarios y referencias 61

5. Conclusiones y trabajos futuros **63**

Bibliografía **65**

A. Herramientas del procesador digital de señales **67**
A.1. Arquitectura de los dispositivos TMS320 C62x/C67x 69
A.2. Interrupciones 69
A.3. Herramientas de hadware 70
A.3.1. El AD535 71
A.3.2. McBSP 73
A.3.3. EDMA 75
A.4. Herramientas de software 76
A.4.1. Entorno de desarrollo Integrado de Code Composer Studio (IDE) 76
A.4.1.1. Características del editor de código de programa . . 78
A.4.1.2. Características de construcción de aplicaciones 78
A.4.1.3. Características de depuración de aplicaciones 78
A.4.2. DSP/ BIOS plugs-ins 79
A.4.2.1. Configuración del DSP/BIOS 79
A.4.2.2. Módulos del DSP/BIOS 79

B. Wave **84**
B.1. Control de propiedades 85

C. Filtros **90**
C.1. Tipos de filtro 90
C.2. Filtros FIR 92
C.3. Filtros IIR 98
C.3.1. Diseño de filtros IIR a partir de filtros analógicos 100
C.3.1.1. Diseño de filtros IIR mediante la aproximación de derivadas 100
C.3.1.2. Diseño de filtros IIR mediante invarianza impulsional 101
C.3.1.3. Diseño de filtros IIR mediante la transformación bilineal 102

D. Algoritmos de programación **103**

Índice de figuras

1.1. Diagrama a bloques del procesamiento de voz. 4

2.1. Señal de voz: a) tiempo continuo, b) tiempo discreto. 8
2.2. Aparato fonador. 9
2.3. Corte transversal de la laringe. Movimiento del cartílago aritenoides y de los repliegues vocales (líneas discontinuas). 10
2.4. Zonas bucales. 11
2.5. Modelo funcional de generación de voz. 12
2.6. El sonido es la alteración física de un medio(líquido, gaseoso o sólido), en el cual se produce un movimiento ondulatorio de sus partículas y se propaga en forma de ondas. 13
2.7. Esquema de un micrófono. 23
2.8. Micrófono dinámico. 24
2.9. Señal de voz. a) Información original. b) Datos muestreados naturalmente. c) Muestras cuantizadas. d) Muestreo y retención. 24
2.10. Modelo de un sistema. 24
2.11. Esquema del muestreo Nyquist. 25
2.12. Muestreo en frecuencia. 25

3.1. Obtención de muestras de la señal de voz. 27
3.2. Almacenamiento de muestras de la señal de voz. 28
3.3. Almacenamiento y proceso de las muestras de voz. 28
3.4. Diagrama a bloques empleado para la simulación. 29
3.5. Señal de voz a ser procesada, el cual almacena la palabra “puerta”. . . 31
3.6. Respuesta en frecuencia de ventana rectangular y Hamming. 33
3.7. Proceso de ventaneo. 34
3.8. Ventaneo de la señal. 37
3.9. Espectro de magnitud normalizado por cada ventana. 40
3.10. Diagrama a bloques para el filtro. 41
3.11. Señal aleatoria y su espectro de frecuencia. 44

3.12. Respuesta en frecuencia del filtro. 45
3.13. Ruido de alta frecuencia y su espectro. 46
3.14. Señal de voz muestreada y su espectro. 47
3.15. Señal de voz con ruido y su espectro. 48
3.16. Respuesta de frecuencia del filtro. 48

4.1. Diagrama a bloques de la implantación 49
4.2. Diagrama de flujo para la configuración de puertos 50
4.3. Muestras obtenidas tras el ventaneo, en a) muestras impares y b) muestras pares 51
4.4. Diagrama de implantación de algoritmos. 52
4.5. Mariposa básica del algoritmo para la FFT. 53
4.6. Primera etapa para la FFT de diezmado en frecuencia. 55
4.7. Mezclado de datos. 56
4.8. Multiplicación de los coeficientes Hamming en tiempo real. 57
4.9. Ruido de alta frecuencia. 58
4.10. Espectro y fase del ruido de alta frecuencia 59
4.11. Espectro de Fourier en tiempo real de la señal de voz. 60

A.1. Diagrama a bloques de una interfaz analógica. 70
A.2. Diagrama a bloques del convertidor 71
A.3. Diagrama a bloques del McBSP. 74
A.4. Diagrama a bloques del EDMA. 75
A.5. Pantalla de inicio del Code Composer. 77
A.6. Ventana de configuración DSP/BIOS. 80
A.7. Camino de datos del TMS320C67x. 83

B.1. Interfaz de trabajo de GoldWave. 84
B.2. Modos gráficos de GoldWave. 86
B.3. Control de propiedades. 86
B.4. Ecualizador. 87
B.5. Lector de disco. 88
B.6. Plug-ins. 88
B.7. Valuador de expresión. 89
B.8. Filtro. 89

C.1. Estructura básica de un FIR. 93
C.2. Características de magnitud de filtros físicamente realizables. 97
C.3. Estructura básica de un IIR. 99

Acrónimos

Siglas	Descripción
AD	Analógico digital.
AIC	Circuito de interfaz analógica.
CCS	Code composer studio.
DA	Digital analógico.
DFT	Transformada discreta de Fourier.
DRR	Registro de recepción de datos.
DSP	Procesador digital de señales.
EDMA	Acceso directo mejorado a la memoria.
FFT	Transformada rápida de Fourier.
FIR	Filtro de respuesta finita al impulso.
IDE	Entorno integrado de desarrollo.
IIR	Filtro de respuesta infinita al impulso.
ISR	Rutina de servicio de interrupción.
LTI	Sistema lineal invariante en el tiempo.
McBSP	Puerto serial multicanal con buffer.
MFLOPS	Millones de instrucciones de punto flotante por segundo
MIPS	Millones de instrucciones por segundo
PAM	Modulación por amplitud de Pulso.
SDRAM	Memoria síncrona dinámica de acceso aleatorio.
STR	Sistema de tiempo real.

Notación

Símbolo	Descripción
$f(t) \longleftrightarrow F(w)$	$F(w)$ es la transformada directa de Fourier de $f(t)$ y $f(t)$ es la transformada inversa de Fourier de $F(w)$.
$\mathcal{F}[x(t)]$	Transformada de Fourier de la función $x(t)$.
$sen(x)$	Función seno de $\boldsymbol{x}$.
$cos(x)$	Función coseno de $\boldsymbol{x}$.
$\forall i$	Cuantificador a i.
$\mathfrak{D}$	Denota la derivada.
$f_1 * f_2$	Convolución de f_1 y f_2 .
∞	infinito.
W_N^{kn}	Factor revoloteo.
$\sum$	Sumatoria.
$\delta(t)$	Impulso unitario de t.
τ	Operador de transformación.
$x(n) \xrightarrow{z} X(z)$	$X(z)$ es la transformada z de $x(n)$.
$x[n] \xrightarrow{\tau} y[n]$	$y[n]$ es la salida del sistema τ a la entrada $x[n]$.

Capítulo 1

Introducción

Un sistema de tiempo real (STR) es un sistema informático en el que es significativo el tiempo en el que se producen las acciones. Las acciones deben realizarse dentro de un intervalo de tiempo determinado. Por ejemplo: un escenario habitual, el filtrado.

En la actualidad, el procesamiento de voz en tiempo real está alcanzando un nivel de calidad que posibilita su uso en aplicaciones tanto personales como dirigidas al público en general. Esta tecnología aunada al reconocimiento de voz en tiempo real el cual ofrece ventajas tales como: una manera más rápida de modificar datos, comodidad al no tener que utilizar el teclado o el ratón (mouse) para la captura de información, no es necesario utilizar el sentido visual, porque no se requiere estar de manera permanente delante de un monitor y además permite la realización de otras actividades manuales o visuales esto en el sentido de utilizar una computadora.

La enorme cantidad de posibilidades que la tecnología digital ofrece, basada en el desarrollo de microprocesadores, cada vez más potentes, hace que las aplicaciones de procesamiento digital de señales se multipliquen. Entre estas aplicaciones, las que involucran señales de voz han permitido disponer de un conjunto de servicios que hasta hace algunos años eran impensables. Redes de integración de voz y datos, diálogo hombre-máquina, síntesis a partir de texto, identificación/verificación de locutores, son algunos ejemplos de los logros alcanzados por el procesado digital de señales de voz.

Para diseñar un sistema en tiempo real se debe considerar:

- Un sistema operativo guiado por eventos, el cual sólo cambia de tarea cuando un evento necesita el servicio.

- Un diseño de compartición de tiempo provee cambios de tareas por interrupciones del reloj y por eventos.

Las interrupciones, son la forma más común de pasar información desde el mundo exterior al programa y son, por naturaleza, impredecibles. En un sistema de tiempo real estas interrupciones pueden informar diferentes eventos como la presencia de nueva información en un puerto de comunicaciones, de una nueva muestra de audio en un equipo de sonido o de un nuevo cuadro de imagen en una videograbadora digital.

Para propósito general un procesador moderno suele ser más rápido, para programación en tiempo real deben utilizarse procesadores lo más predecibles posibles, sin caché, sin paginación, sin predicción de saltos, etc. Todos estos factores añaden una aleatoriedad que hace que sea difícil demostrar que el sistema es viable, es decir, que cumple con los plazos en el tiempo.

1.1. Planteamiento del problema

¿Cómo obtener las características espectrales de señal de voz en tiempo real?

1.2. Propuesta de solución

El ingeniero en telecomunicaciones ha dedicado el mayor tiempo de sus estudios en analizar señales en tiempo continuo y discreto. Concluye que las señales de mayor importancia son las señales digitales, ya que ofrecen mayor número de ventajas frente a las analógicas. Pero aún existe un problema, y es el buscar un dispositivo electrónico que sea veloz, ejecute un mayor número de instrucciones, ahorre energía y presente precisión en la respuesta a su sistema.

Dado que la señal de voz cambia de acuerdo a la situación, por ejemplo si una persona habla por micrófono y pronuncia una palabra, ésta puede ser procesada, con el propósito de analizar sus características espectrales, una vez teniendo esto, se procede a realizar una segunda prueba con la misma persona y la misma palabra para obtener otra muestra, resultando que ambas muestras son diferentes, esto puede ser debido a muchas circunstancias una de ellas puede ser que la persona habló con mayor rapidez y pronunció la palabra más fuerte. Por lo tanto la propuesta de solución al problema planteado es la siguiente:

- Capturar la señal de voz a través de un micrófono.
- Emplear un DSP para el procesamiento de señal en tiempo real.
- Implantar algoritmos de procesamiento digital para analizar la señal como puede ser la transformada de Fourier, convolución, filtros digitales, etc.

1.3. Objetivos

GENERAL

1. Implantar en un DSP algoritmos para el procesamiento digital de voz en tiempo real y en un DSP para caracterizar a la señal de voz.

PARTICULARES

1. Estudiar las propiedades de la transformada de Fourier y convolución, para la implantación de filtros digitales.

2. Estudiar el manejo del DSP, para hacer posible la programación en el procesamiento de voz.

3. Analizar señales de voz para conocer sus características y manipular cada señal examinada.

1.4. Justificación

El desarrollo de nuevas tecnologías en nuestro mundo necesita de la implementación de teoremas y postulados matemáticos (en forma de algoritmos) en tiempo real que ayuden a solventar problemas para la comodidad del ser humano. Obtener las características espectrales de una señal de voz es una tarea importante en la actualidad; pueden ser utilizadas en el área de domótica, ya que con sus características espectrales se pueden diseñar sistemas de control [**?**].

1.5. Antecedentes

En 1963 General Electric, el MIT, y AT&T Bell Laboratories comienzan a trabajar en el proyecto Multics. El objetivo de este proyecto es la construcción de un sistema operativo de propósito general de memoria compartida, multiprocesamiento y tiempo compartido. Edsger Dijkstra describe y nombra el Problema de las Regiónes Críticas. Mucho del trabajo posterior en sistemas concurrentes es dedicado a encontrar formas eficientes y seguras de manejar regiones críticas.

En 1965 James W. Cooley y John W. Tukey describen el Algoritmo de la Transformada Rápida de Fourier(FFT); que es uno de los más grandes algoritmos que utilizan subrutinas de punto flotante.

En el año 1977, la transformada rápida de Fourier es simplemente un algoritmo rápido para la evaluación numérica de integrales de Fourier desarrollado en los laboratorios de IBM, y su importancia radica en la rapidez de cálculo conseguida, toma

su importancia en: ecualización y filtrado en equipos de audio/video en tiempo real, comunicaciones, etc. Evidentemente se hace uso del mismo en el programa para obtener rápidamente el espectro de la señal a partir de la señal temporal de entrada, aunque se podría haber hecho a partir de la integral discreta de Fourier, siendo en este caso necesario mucho más tiempo de cálculo.

En 1986, Karlheinz Brandenburg, director de tecnologías de medios electrónicos del Instituto Fraunhofer IIS, realizó una investigación junto con Thomson Multimedia y registra la primer patente del formato MP3. Es un formato de audio digital comprimido, en este formato comprimido utilizan la transformada de Fourier para la elaboración de estos archivos.

En 1998, el departamento de sistemas y computadoras de la universidad politécnica de Valencia, implementa "Transformada Rápida de Fourier(FFT) e interpolación en tiempo real" por el Doctor Juan Luis Posadas Yague. El cual diseña e implementa un sistema en tiempo real que obtiene la FFT basado en la función SINC.

1.6. Cuerpo de trabajo de tesis

Un diagrama a bloques en general del proyecto para la elaboración de la transformada de Fourier en tiempo real es:

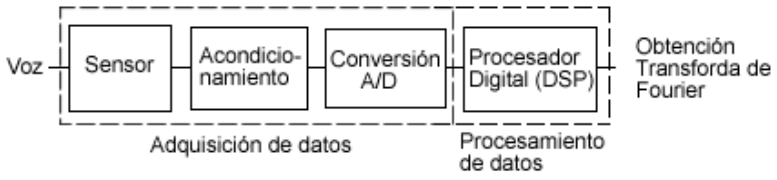

Figura 1.1: Diagrama a bloques del procesamiento de voz.

A continuación se describen los capítulos que conforman el documento y que están relacionados con el diagrama a bloques que se muestra en la Figura 1.1.

- El **Capítulo 1** presenta brevemente los objetivos generales, objetivos específicos, la justificación, el planteamiento del problema de esta tesis así también como antecedentes, haciendo descripción de un sistema real.
- El **Capítulo 2** explica como se genera la **voz**, como se propaga hasta el **sensor** y el proceso de **conversión** de la señal analógica a formato digital.
- En el **Capítulo 3** se presentan los resultados de simulaciones realizadas al aplicar el filtro FIR y la FFT al procesamiento digital de voz.

- En el **Capítulo 4** se exponen los resultados de laboratorio logrados con el procesador digital de señales de *Texas Instruments* TMS320C6711.
- Finalmente, en el **Capítulo 5** se describen las conclusiones de este trabajo.

También el documento contiene tres apéndices los cuales están organizados de la siguiente manera:

- El **Apéndice A** comprende las especificaciones técnicas del **procesador digital de señales** TMS320C6711.
- El **Apéndice B** contiene una breve descripción del simulador GoldWave.
- El **Apéndice C** abarca la clasificación de filtros, explicando ampliamente los filtros digitales y el diseño de filtros caracterizados por el tipo de respuesta a una entrada unitaria (FIR e IIR).
- El **Apéndice D** se presentan los algoritmos diseñados en el desarrollo de esta tesis.

Capítulo 2

Señales de sonido y voz

Tomando como referencia el diagrama bloques de la Figura 1.1, se comenzará por analizar la señal de voz, ¿Cómo se genera? y ¿Cuáles son los organismos que participan en la generación de la voz? (sección 2.2). Continuando con el diagrama, estudiaremos el medio entre la voz y el sensor (sección2.3); siguiendo con el diagrama se verá el dispositivo electrónico que ayuda a convertir las ondas sonoras a eléctricas (sección 2.4); una vez convertida en señal eléctrica analógica se pasa a formato digital para que pueda ser manejada por el DSP(procesador digital de señales), (sección 2.5); una vez comprendido algunos concepto, se da una clasificación de los sistemas de acuerdo a la temática del procesamiento digital de señales (sección 2.6), y por último; se explican algunos teoremas de importancia para el desarrollo de esta tesis (sección 2.7).

2.1. Introducción

Una señal es una variable física que contiene o transporta información. Por ejemplo:

- Señales eléctricas — tensiones y corrientes en un circuito.
- Señales acústicas — señales de lenguaje o de audio (analógicas o digitales).
- Señales de video — variaciones de intensidad en una imagen.
- Señales biológicas — secuencia de bases en un generador.

Una posible clasificación de las señales es la siguiente :

1. Señales analógicas.

   Cuando se habla de señales analógicas se hace referencia a señales que son funciones continuas, ya sea del tiempo o de la frecuencia. Así entonces una señal

analógica puede describirse mediante una expresión matemática o gráficamente por medio de una curva o incluso por un conjunto de valores tabulados. Como se puede ver en la Figura 2.1a.

2. Señales discretas.

   Por otro lado las señales discretas son aquellas señales que se conocen sólo en ciertos instantes de tiempo y pueden surgir naturalmente o como consecuencia de hacer un muestreo de señales continúas. Una señal muestreada o discreta $x[k]$ es justamente una sucesión ordenada de valores correspondientes al índice entero k que engloba la historia del tiempo de la señal.

   Una señal discreta $x[k]$ se gráfica como líneas contra el índice k. Como es posible observar en la Figura 2.1b.

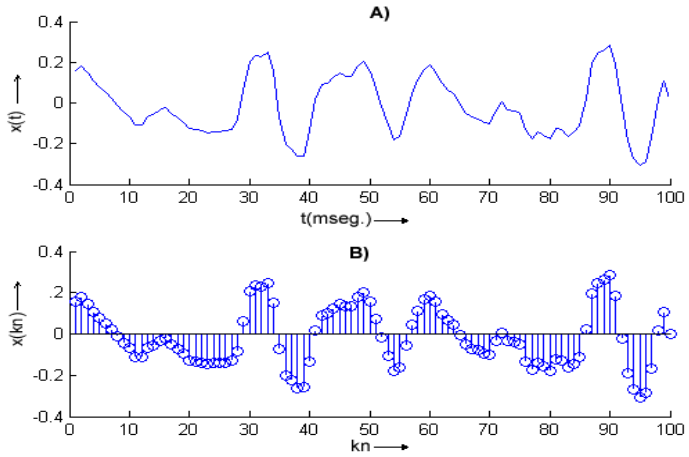

Figura 2.1: Señal de voz: a) tiempo continuo, b) tiempo discreto.

En este trabajo es de vital importancia la señal de voz, por lo cual se presentarán sus principales características.

2.2. La voz

Definición

La voz es el sonido producido por el aparato fonador humano [8].

Aparato fonador

El aparato fonador se puede dividir en tres grandes partes: las cavidades infraglóticas, la cavidad glótica y las cavidades supraglóticas. Cada una de ellas realiza una misión distinta en la fonación, pero todas ellas son imprescindibles en la misma. En la figura siguiente se presenta una descripción.

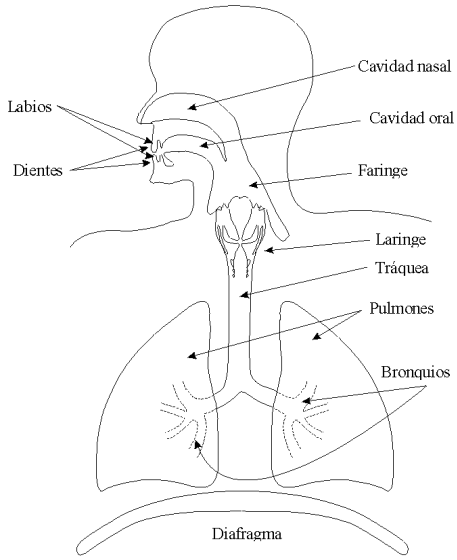

Figura 2.2: Aparato fonador.

Cavidades infraglóticas

Tienen como misión la corriente de aire espirada necesaria para producir el sonido. Están compuestas por diafragma, pulmones, bronquios y tráquea.

El diafragma es un músculo situado por debajo de los pulmones y con forma de cúpula. Su misión es controlar el despliegue e hinchado de la cavidad pulmonar o su reducción y vaciado junto con los músculos pectorales, y con ello la respiración. Cuando se contrae el diafragma se ensancha la cavidad torácica, produciéndose la inspiración de aire; al reflejarse se reduce la cavidad, produciéndose la espiración del aire contenido en los pulmones.

Los bronquios y tráquea son tubos cartilaginosos que conducen el aire entre los pulmones y la laringe. Su función en la fonación es la de simples canales de transmisión de flujo aéreo.

Cavidad glótica

Está formada por la laringe. La característica mas interesante es la presencia de las cuerdas vocales, que son las responsables de la producción de la vibración básica para la generación de la voz. Aunque se llaman tradicionalmente cuerdas vocales, en realidad se trata de dos marcados pliegues musculosos. Cuando el aire sale de los pulmones pasa por la hendidura glótica (la glotis es el espació triangular que queda entre las cuerdas vocales), haciéndose vibrar. La vibración producida que se emite se puede variar en frecuencia e intensidad según varíe la masa, longitud y tensión de las cuerdas vocales.

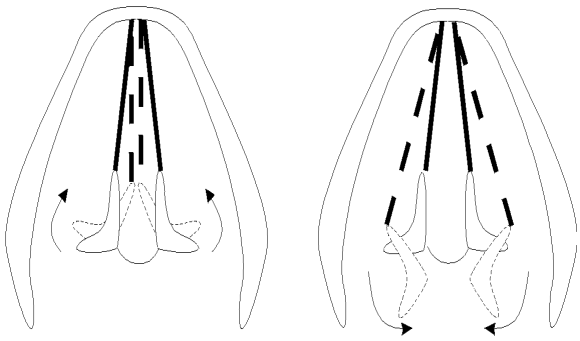

Figura 2.3: Corte transversal de la laringe. Movimiento del cartílago aritenoides y de los repliegues vocales (líneas discontinuas).

Cavidades supraglóticas

Existen cuatro: faringe, nasal, bucal y labial.

Mas arriba de la laringe, lo primero que se encuentra es la faringe, de donde arranca la raíz de la lengua. Aparece el primer obstáculo móvil: la úvula; es el apéndice final del paladar blando o velo del paladar. Cuando está unida a la pared faríngea, la corriente de aire sale exclusivamente por la boca, produciendo sonidos orales. Si el velo del paladar está caído, también se expulsará aire por la cavidad nasal. La cavidad nasal carece de elementos móviles, por lo cual su función es pasiva en la producción del habla.

La lengua es el órgano más móvil de la boca, registrando una actividad elevada durante el habla. Se divide en tres partes: raíz, dorso y ápice. En la fonética tradicionalmente no no se le prestó toda la atención que se le merecía, pero recientemente se ha demostrado que el perfil que adopta en cada movimiento es causa de un resonado acústico y, por lo tanto, el timbre del sonido será diferente según la forma sea cóncava, convexa o plana, o que se situé en la parte anterior, central o posterior.

Dentro la cavidad bucal tenemos los dientes y alvéolos. Los dientes son órganos pasivos en la medida que están insertos en los maxilares; los inferiores son móviles para estar engarzados en la mandíbula inferior, siendo esta actividad en la articulación. El paladar es una amplia zona que va desde los alvéolos hasta la úvula. En ella se distingue el paladar duro, situado sobre el hueso palatino, y el paladar blando o velo del paladar que acaba en la úvula.

Finalmente tenemos los labios, elemento que posee bastante movilidad y, que por lo tanto permite modificar los sonidos.

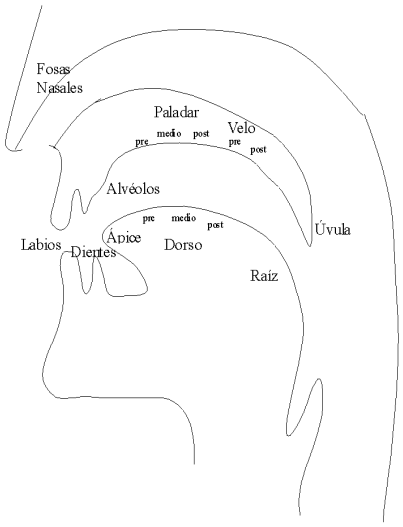

Figura 2.4: Zonas bucales.

Para producción del habla se dan los siguientes elementos

- Una fuente de energía, proporcionada por el aire a presión que se expulsa en la espiración.
- Un órgano vibratorio: las cuerdas vocales.
- Una caja de resonancia: las fosas nasales, la cavidad bucal y la faringe.
- Un sistema de articulación del sonido: lengua, labios, dientes y úvula.

El proceso se inicia con la espiración del aire, al pasar a través de las cuerdas vocales las hace vibrar a una frecuencia determinada que depende de la tensión de las mismas; a esta frecuencia se le conoce como frecuencia fundamental. El tono está relacionado con la frecuencia fundamental: cuando el tono es grave indica que

la frecuencia es baja y cuando es agudo que la frecuencia es alta, según como se encuentre articulados los órganos se formará una caja de resonancia distinta, la cual potenciará un conjunto de frecuencia y atenuará el resto. Aunque articulemos de forma similar los distintos fonemas, según la distancia, forma, dureza. Etc. De los órganos, aparecen características especiales de cada individuo, que es el timbre. Finalmente sale al exterior la voz. Este proceso explica el conjunto de fonemas sonoros.

El resto de fonemas se producen por fricciones y explosiones de aire.

Modelo funcional

En la Figura 2.6 se muestra un modelo funcional del sistema de generación de voz, en el que se observa con mayor claridad los, órganos y cavidades involucradas en el proceso de generación de voz.

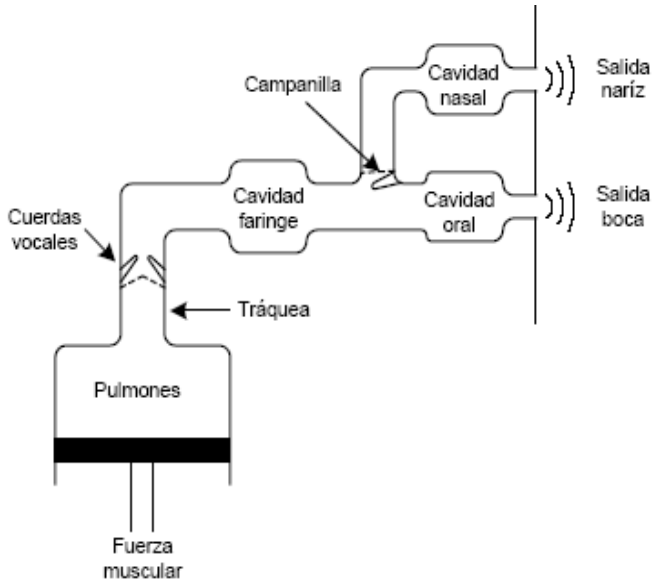

Figura 2.5: Modelo funcional de generación de voz.

2.3. El sonido

Cuando un cuerpo vibrante consigue desplazar las moléculas de aire hacia adelante y hacia atrás, y este movimiento se expande como un conjunto de burbujas concéntricas y llega hasta nuestros oídos, la sensación que percibimos se llama sonido, Figura 2.6.

Figura 2.6: El sonido es la alteración física de un medio(líquido, gaseoso o sólido), en el cual se produce un movimiento ondulatorio de sus partículas y se propaga en forma de ondas.

todos los cuerpos son capaces de emitir sonido si son golpeados, frotados o en general excitados para que vibren. A pesar de esto, para que el sonido se transmita desde el cuerpo vibrante hasta el micrófono, es necesario que exista un medio elástico ya sea sólido, liquido o gaseoso. Sin tal medio el movimiento vibrante no se transmitirá.

2.4. Micrófono

El micrófono es un dispositivo que transforma la energía sonora en energía eléctrica procesable. De este dispositivo depende que la calidad de la reproducción de la voz para que se realice con absoluta fidelidad. La voz humana se mueve teóricamente entre

Figura 2.7: Esquema de un micrófono.

los 20Hz y los 20khz. Es por ello que a la hora de elegir un micrófono el rango de frecuencia abarcables por éste, ha de ser lo más amplio posible, siendo ideal un rango de frecuencias mencionados para la voz humana.

Características

Algunas de las características de un micrófono son:

Sensibilidad se trata de la capacidad que tiene el micrófono en captar el sonido más bajo que se puede producir, es decir, si tenemos cuatro micrófonos, y colocamos delante de ellos una emisión sonora, como puede ser una señal constante a 1000 hertzios, y en el conector de cada micrófono conectamos un voltímetro, veremos que cada uno marcará un voltaje distinto. Se mide en decibelios (dBs).

Respuesta de frecuencia Se trata de la sensibilidad que tiene el micrófono para todas las frecuencias.

Ruido propio Se trata del nivel de ruido que tiene un micrófono sin aplicarle ninguna fuente sonora, es decir, el ruido que por naturaleza y construcción genera el micrófono.

Relación señal/ruido se trata nada más ni nada menos que de la resta entre el nivel sonoro que le apliquemos al micrófono (también de dBs) y el ruido propio del micrófono.

Impedancia es la capacidad que tiene un elemento eléctrico o electrónico (en nuestro caso un micrófono) en impedir el paso de corriente eléctrica. La impedancia de mide en Ohmios.

Tipos de micrófonos según su construcción

Micrófonos dinámicos están formados por un imán permanente acomodado dentro de una pieza cilíndrica de plástico o metal que sirve como estructura. Sobre el imán se encuentra una bobina móvil, y, a su vez, encima de ésta un diafragma que consiste en una pequeña lámina de plástico muy ligera, Figura 2.8.

Las ondas sonoras, que han viajado por el aire, impulsan con movimientos vibratorios al diafragma; éste, a su vez, transmite el movimiento vibrante a la bobina móvil que, al desplazarse de forma continua dentro del campo magnético producido por el imán permanente, induce una corriente eléctrica que sigue las variaciones de frecuencia e intensidad de la onda sonora; de esta manera se genera una señal eléctrica que representa la forma de la onda de sonido.

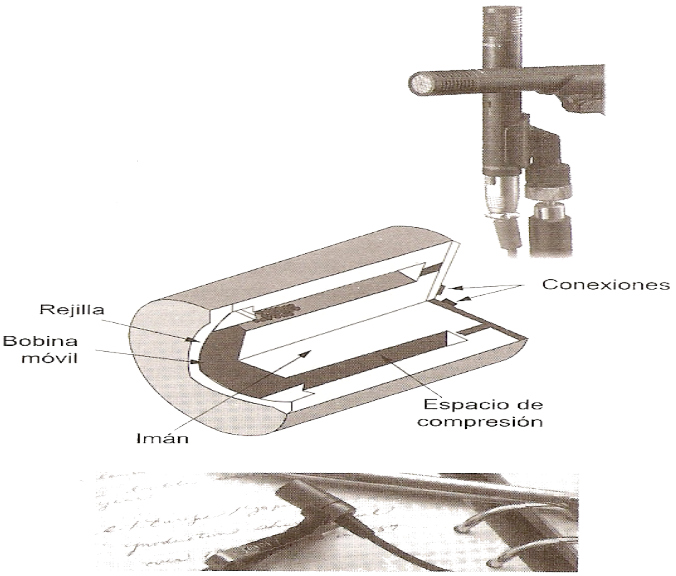

Figura 2.8: Micrófono dinámico.

Micrófonos de cristal Su principio de operación se basa en el llamado efecto piezoeléctrico, que es sinónimo de electricidad por presión y se refiere a la propiedad que tiene ciertos cristales(cuarzo) de producir en sus extremos una pequeña diferencia de potencial(voltaje) cuando se les aplica en el cuerpo una fuerza extrema.

Los micrófonos de cristal tienen estratégicamente colocada, entre un par de placas metálicas (una fijas y otra móvil), una oblea de cristal; estos tres elementos mantienen contacto entren sí.

Cuando al micrófono llega una onda sonora, ésta provoca que la placa móvil vibrante, y entonces ejerza presión sobre la oblea de cristal. Dicha presión hace que en la superficie de la oblea aparezca un voltaje, el cual sigue las variaciones del movimiento de la onda sonora. Finalmente, este voltaje se transforma en una corriente eléctrica variable que representa a la misma onda sonora.

Micrófonos de carbón En la actualidad ya no se usan, estaban formados por una pieza cilíndrica de carbón granulado, y tenía una placa metálica sobre cada una de sus caras planas. Cuando las ondas de sonido chocaban contra las placas, éstas ejercían mayor o menor presión sobre los gránulos de carbón;y de esta manera, se provocaba que la resistencia total entre las placas variara en proporción con la onda de sonido.

Se puede decir que cuanto más sensible sea en la percepción de las ondas sonoras, más fiel será la señal eléctrica que proporcione.

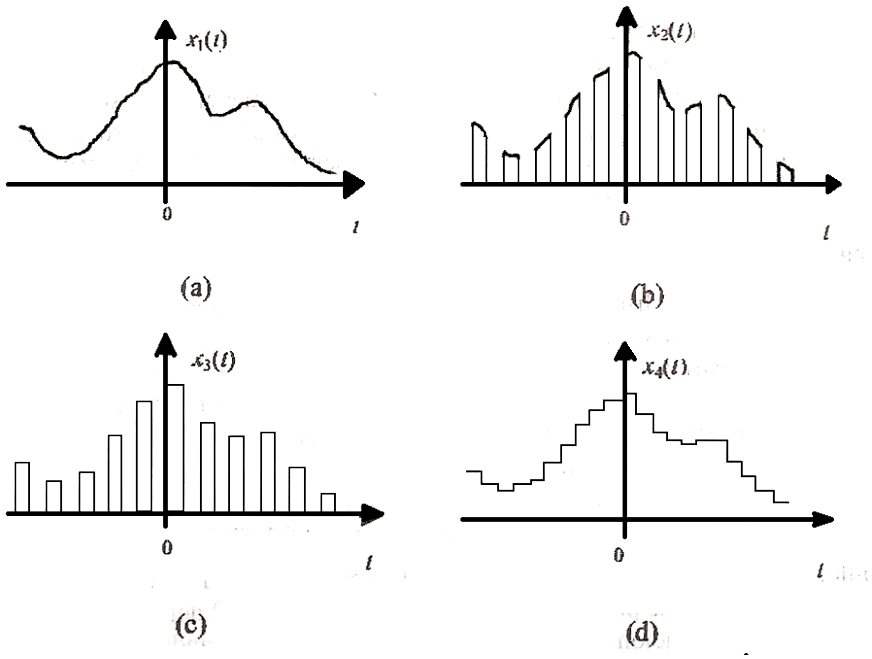

Figura 2.9: Señal de voz. a) Información original. b) Datos muestreados naturalmente. c) Muestras cuantizadas. d) Muestreo y retención.

2.5. Proceso de conversion analógica a digital

Existen cuatro formas de onda que podemos describir en el proceso de conversión de la voz en una señal digital, Figura 2.9. La referencia es la forma de onda original de voz, Figura 2.9a. La Figura 2.9b representa una versión muestreada de la señal original, conocida como datos muestreado naturalmente o señal PAM (Modulación por Amplitud de Pulso). No obstante, tal señal es incompatible con un sistema digital, ya que es continua en amplitud y un sistema digital trata con un número finito de símbolos. La Figura 2.9c ilustra las muestras cuantizadas, es decir, cada pulso es expresado como un nivel a partir de un conjunto finito de niveles. en este caso, a cada subnivel le es asignado un símbolo de un alfabeto finito. Los pulsos de la Figura 2.9c son conocidos como muestras cuantizadas; tal formato es la elección obvia para un sistema digital ya que la señal es representada por amplitudes discretas. El formato de la Figura 2.9d es construido a partir de la salida de un circuito de muestreo y retención.

Cuando los valores de las muestras son discretas, tal formato puede ser utilizado por un sistema digital. En todos los casos mencionados, la señal analógica original puede ser aproximadamente reconstruida a partir de las muestras cuantizadas. No obstante, la fidelidad de la reconstrucción puede mejorarse al incrementar el número de niveles de cuantización (requiriéndose un mayor ancho de banda en el sistema digital).

2.6. Sistemas

Un sistema es un conjunto interconectado de elementos que procesan una señal. Se caracteriza por tener una o más entradas y una o más salidas.

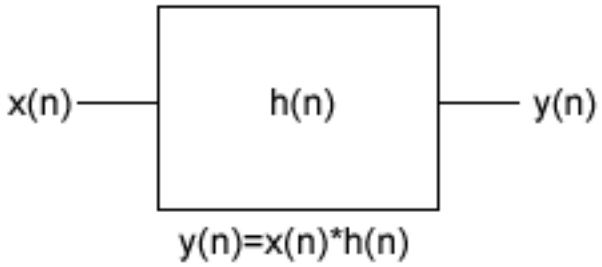

Figura 2.10: Modelo de un sistema.

2.6.1. Clasificación de los sistemas discretos

Tanto en el análisis como en el diseño de sistemas es conveniente realizar una clasificación de los mismos según las propiedades que satisfacen. Las técnicas matemáticas que se aplican para analizar y diseñar sistemas en tiempo discreto dependen fuertemente de las características generales de los sistemas considerados. Para que un sistema disponga de una propiedad determinada, ésta debe cumplirse para cada posible señal de entrada al sistema. Si una propiedad se satisface para algunas señales de entrada pero no para otras, el sistema no posee tal propiedad, es decir, para demostrar que el sistema tiene alguna propiedad, debe comprobarse que esta propiedad se cumple para cualquier señal de entrada posible[**?**].

Sistemas estáticos y sistemas dinámicos

Un sistema en tiempo discreto se denomina *estático* o sin memoria si su salida en cualquier instante n depende a la suma de la muestra de entrada en ese mismo ins-

tante, pero no de las muestras pasadas o futuras de la entrada, como en las siguientes ecuaciones:

$$y(n) = ax(n), \tag{2.1}$$
$$y(n) = nx(n) + bx^3(n), \tag{2.2}$$

En cualquier otro caso, se dice que el sistema es *dinámico* o con memoria, por ejemplo:

$$y(n) = x(n) + 3x(n-1), \tag{2.3}$$
$$y(n) = \sum_{k=0}^{n} x\,(n-k), \tag{2.4}$$
$$y(n) = \sum_{k=0}^{\infty} x\,(n-k), \tag{2.5}$$

Sistemas invariantes en el tiempo y sistemas variantes en el tiempo

La clase general de sistemas puede subdividirse en dos categorías amplias: sistemas invariantes en el tiempo y sistemas variantes en el tiempo. Un sistema se dice invariante en el tiempo si sus características de entrada-salida no cambian con el tiempo. Por ejemplo, un sistema τ en reposo que, cuando se excita con una señal $x(n)$, produce una señal de salida $y(n)$. Si se tiene:

$$y(n) = \tau\,[x(n)] \tag{2.6}$$

Supóngase ahora que esa misma entrada es retardada k unidades de tiempo para dar lugar a $x(n-k)$, y de nuevo se aplica al mismo sistema. Si las características del sistema no cambian con el tiempo, la salida del sistema en reposo será $y(n-k)$, es decir, la salida será la misma que la correspondiente a la entrada $x(n)$, excepto en que estará retardada las mismas k unidades de tiempo que se retardó la entrada. Esto conduce a definir un sistema invariante en el tiempo o invariante ante desplazamientos usando el teorema que se explica a continuación.

Teorema.

Un sistema en reposo τ es *invariante en el tiempo* o *invariante a desplazamientos* si y sólo si

$$x(n)\tau y(n),$$

$$x(n-k) \xrightarrow{\tau} y(n-k), \tag{2.7}$$

para toda señal de entrada $x(n)$ y todo desplazamiento temporal k.

Sistemas lineales frente a sistemas no lineales

Un sistema lineal es aquel que satisface el *principio de superposición*, el cual exige que la respuesta del sistema a una suma ponderada de señales sea igual a la correspondiente suma ponderada de las salidas a cada una de las señales de entrada. Por tanto, se tiene la siguiente definición de linealidad.

Teorema.

Un sistema es lineal si y sólo si

$$\tau\left[a_1x_1(n) + a_2x_2(n)\right] = a_1\tau\left[x_1(n)\right] + a_2\tau\left[x_2(n)\right]. \tag{2.8}$$

Para cuales quiera secuencias arbitrarias de entrada $x_1(n)$ y $x_2(n)$, y cuales quiera constantes arbitrarias a_1 y a_2.

Sistemas causales frente a sistemas no causales

Teorema.

Se dice que un sistema es *causal* si su salida en cualquier instante n (es decir, $y(n)$) depende sólo de las entradas presentes y pasadas (es decir, $x(n), x(n-1), x(n-2), \ldots$) pero no de las futuras (es decir, $x(n+1), x(n+2), \ldots$). En términos matemáticos, la salida de un sistema causal verifica una ecuación de la forma

$$y(n) = F[x(n), x(n-1), x(n-2), \ldots], \tag{2.9}$$

donde $F[\cdot]$ es una función arbitraria. Si un sistema no satisface esta definición se dice que es *no causal*. En un sistema de este tipo, la salida depende no sólo de las entradas pasadas y presentes sino también de las futuras. Es evidente que en un sistema en tiempo real no se dispone de las entradas futuras de la señal y, por lo tanto, un sistema no causal no puede ser realizado físicamente (es decir, no puede ser implementado). Por otra parte, si la señal se graba de manera que el procesamiento no se realiza en tiempo real, es posible implementar sistemas no causales, ya que todos los valores de la señal se encuentran disponibles en el momento del procesado.

Sistemas estables frente a sistemas inestables

La estabilidad es una propiedad muy importante que debe ser considerada en cualquier aplicación práctica de un sistema. Sistemas inestables presentan un comportamiento errático y extremo que es causa de desbordamiento del sistema en aplicaciones prácticas[9].

Teorema.

Un sistema arbitrario en reposo se dice de entrada acotada-salida acotada (*BIBO, bounded input-bounded output*), si y sólo si toda entrada acotada produce una salida

acotada. Matemáticamente, el acotamiento de las secuencias de entrada y salida, $x(n)$ e $y(n)$, se traduce en la existencia de un par de números finitos, digamos M_x y M_y, tales que

$$|x(n)| \leq M_x < \infty, \qquad |y(n)| \leq M_y < \infty, \tag{2.10}$$

para todo n. Si, para alguna entrada acotada $x(n)$ la salida no está acotada (es infinita), el sistema se clasifica como inestable. Por ejemplo, en el sistema no lineal descrito mediante la ecuación de entrada-salida

$$y(n) = y^2(n-1) + x(n)$$

Como entrada se selecciona la señal acotada

$$x(n) = C\delta(n)$$

donde C es una constante y $\delta(n)$ es la función impulso. Suponiendo que $y(-1) = 0$. Entonces la secuencia de salida es

$$y(0) = C, \quad y(1) = C^2, \quad y(2) = C^4, \ldots, y(n) = C^{2^n}.$$

Claramente, la salida no está acotada si $1 < |C| < \infty$. Por lo tanto, el sistema es inestable dado que una entrada acotada ha producido una salida no acotada.

2.7. Teoremas

Muestreo Nyquist

El muestreo de una señal con frecuencia de muestreo $fs > 2fb$ es llamado muestreo Nyquist. El esquema de la Figura 2.11 muestra el proceso. El filtro analógico pasabajas(2.11a) limita el ancho de banda de la señal de entrada a $fs/2$. El siguiente circuito muestreo - retenedor muestrea el ancho de banda limitado a una tasa de muestreo fs.

La amplitud en función del tiempo sobre un periodo de muestreo $Ts = \frac{1}{fs}$ es convertido a una secuencia x(n) por un cuantizador(Figura 2.11b). Esta secuencia de números llega al procesador digital de señales (DSP), el cual, a la señal se le aplican algoritmos matemáticos. La secuencia de salida $y(n)$ es entregada al convertidor DA, el cual da una señal de salida como el de la Figura 2.11c. Siguiendo con el diagrama un filtro pasabajas da una señal analógica $y(t)$(Figura 2.11d). En la Figura 2.12 se demuestra pasa a paso la conversión AD / DA en el dominio de la frecuencia. Cada espectro de la Figura 2.12a...d corresponde a la salida $a...d$ de la Figura 2.11.

Después de limitar el ancho de banda y muestrearla(2.12a), un espectro periódico con periodo fs de la señal muestreada es obtenida en la Figura 2.12b. Resumiendo

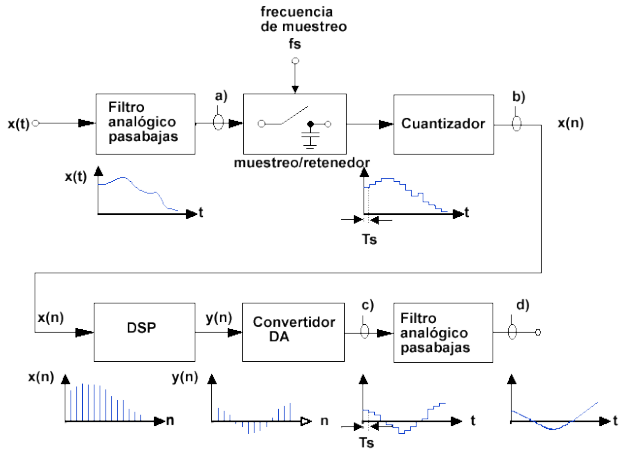

Figura 2.11: Esquema del muestreo Nyquist.

el error de cuantización $e(n)$ es estáticamente independiente de cada una. El ruido tiene un espectro uniforme cuya distribución en la frecuencia es $0 \leq f \leq fs$. La salida del convertidor DA tiene un espectro periódico. Sin embargo es echa con la función $sinc(sinc = \frac{\text{sen}(x)}{x})$ de un circuito retenedor ($Figura 2.12c$). Los ceros de la función $sinc$ son múltiplos de la frecuencia de muestreo fs. Como resultado se reconstruye la señal(2.12d), el espectro imagen es eliminado por un filtro pasabajas.

2.7.1. Convolución

Una convolución es un operador matemático que transforma dos funciones x y h en una tercera función que en cierto sentido representa la magnitud en la que se superponen, x y una versión trasladada e invertida de h. La convolución de x y h se denota $x * h$. Una convolución es un tipo de cierto promedio en movimiento, como se puede observar si una de las funciones es tomada la función característica de un intervalo.

Propiedades de la convolución

Las propiedades de los diferentes operadores de convolución son las siguientes:
Conmutativa.

$$x * h = h * x.$$

*Nota:*Esta propiedad se puede perder si no se pide que la función sea reflejada.
Asociativa.

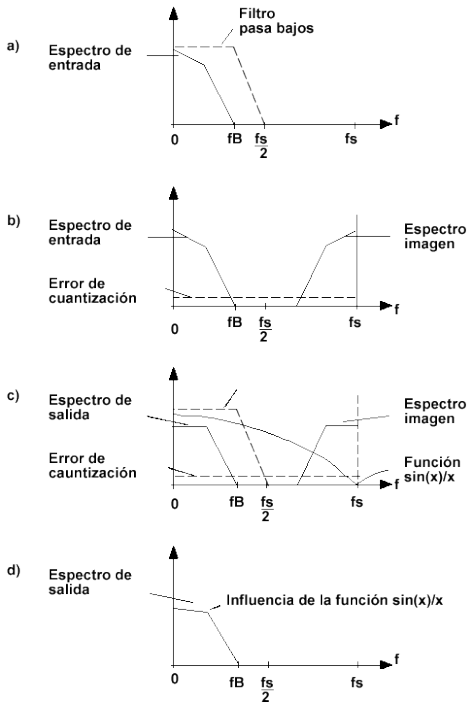

Figura 2.12: Muestreo en frecuencia.

$$x * (h * f) = (x * h) * f.$$

Distributiva.

$$x * (h + f) = (x * h) + (x * f).$$

Asociativa con multiplicación escalar.

$$a(x * h) = (ax) * h = x * (ah).$$

Para todo número complejo o real a.

Regla de derivación.

$$\mathfrak{D}(x * h) = \mathfrak{D}x * h = x * \mathfrak{D}h,$$

donde $\mathfrak{D}$ denota la derivada de x.

2.7.2. Convolución discreta

Convolución es un concepto que se extiende a todos los sistemas que son lineales e invariantes en el tiempo (LTI - Linear Time Invariant). La convolución es un instrumento poderoso al determinar el resultado de un sistema después de saber la entrada arbitraria y la respuesta al impulso del sistema.

2.7.2.1. Suma de convolución.

La suma de convolución provee una manera matemáticamente concisa para expresar el resultado de un sistema LTI, basado en una entrada arbitraria para una señal discreta y también el saber la respuesta del sistema. La suma de convolución es expresada como

$$y[n] = \sum_{k=-\infty}^{\infty} (x[k]\, h[n-k]). \tag{2.11}$$

Tanto en tiempo discreto como en tiempo continuo, la convolución es representada por el símbolo $*$, y puede ser escrita como

$$y[n] = x[n] * h[n].$$

Las señales discretas pueden ser representadas por la suma de impulsos discretos que están desplazados y escalados. Como se asume que el sistema es lineal e invariante con el tiempo, es razonable decir que la entrada de la señal esta formada por impulsos que también están escalados y desplazados, esto en turno daría como resultado del sistema una suma de respuesta de impulsos que también están escalados y desplazados. Esto es exactamente lo que ocurre en convolución[5].

A continuación se explica una manera matemática de ver esta derivación

$$\begin{aligned} y[n] &= H\big[x[n]\big], \\ &= H\Big[\sum_{k=-\infty}^{\infty} \big(x[k]\,\delta[n-k]\big)\Big], \\ &= \sum_{k=-\infty}^{\infty} \Big(H\big[x[k]\delta[n-k]\big]\Big), \\ &= \sum_{k=-\infty}^{\infty} \Big(x[k]\,H\big[\delta[n-k]\big]\Big), \\ &= \sum_{k=-\infty}^{\infty} \Big(x[k]\,h[n-k]\Big), \end{aligned} \tag{2.12}$$

donde H denota un sistema LTI, la ecuación (2.12) es la suma de convolución. Se utiliza la propiedad de desplazamiento para reescribir la función $x[n]$ como una suma

de funciones multiplicada por una suma unitaria. Después, se mueve el operador H y la sumatoria es lineal en el sistema. Por esta linealidad, y por el hecho que $x[k]$ es constante, se pueden extraer las constantes ya mencionadas y únicamente multiplicar la ecuación por H. Finalmente, se usa el dato que H es invariante con el tiempo para llegar a la ecuación deseada, la suma de convolución. Si se quiere calcular la salida del sistema en un instante determinado, por ejemplo $n = n_0$. De acuerdo con (2.12), la respuesta en $n = n_0$ vendrá dada por

$$y(n_0) = \sum_{k=-\infty}^{\infty} \Big(x[k]\, h[n_0 - k] \Big). \tag{2.13}$$

El cálculo de la convolución entre $x[k]$ y $h[k]$ supone la realización de los siguientes cuatro pasos:

1. *Reflexión.* Se refleja $h[k]$ respecto de $k = 0$ para producir $h[-k]$.

2. *Desplazamiento.* Se desplaza $h[-k]$, n_0 hacia la derecha (izquierda) si n_0 es positivo (negativo), para obtener $h[n_0 - k]$.

3. *Multiplicación.* Se multiplica $x[k]$ por $h[n_0 - k]$ para obtener la secuencia producto $v_{n_0}[k] \equiv x[k]h[n_0 - k]$.

4. *Suma.* Se suman todos los valores de la secuencia producto $v_{n_0}[k]$ y se obtiene el valor de la salida en el instante $n = n_0$.

Con este procedimiento se obtiene la salida del sistema en un instante determinado, esto es $n = n_0$. En general, se busca determinar la salida del sistema en cualquier instante de tiempo $-\infty < n < \infty$. En consecuencia, los pasos del 2 al 4 del procedimiento descrito deberán repetirse para todos los posibles valores del desplazamiento $-\infty < n < \infty$.

2.8. Comentarios y referencias

El aspecto más importante de este capítulo ha sido la caracterización de señales y sistemas. De particular importancia es la clase de los sistemas lineales invariantes en el tiempo (LTI), ampliamente usados en la práctica para el diseño e implementación de sistemas de procesado digital. Se han caracterizado los sistemas LTI mediante su respuesta impulsional y analizado la fórmula de la convolución tanto discreta como continua que permite determinar la respuesta del sistema a cualquier señal de entrada.

Para obtener las características espectrales de voz, se trabajará con señales digitalizadas y sistemas en tiempo discreto lineales invariantes en el tiempo; la convolución

será una herramienta imprescindible debido a que las salida de un sistema lineal representa una convolución entre la entrada y la respuesta del sistema a un impulso.

Existen varios libros sobre señales y sistemas discretos. Por ejemplo los de McGillem y Cooper (1984), Oppenheim y Willsky (1983), Siebert (1986), Hildebrand (1952) y Levy y Lessman (1961). Muchas aplicaciones prácticas del procesado digital de señales de voz, de imagen, de radar, sonar y geofísicas son tratadas en el libro editado por Oppenheim (1978).

Capítulo 3

Simulaciones

Este capítulo tiene como objetivo presentar los resultados obtenidos en las simulaciones al aplicar el filtro FIR y la FFT al procesamiento digital de voz, para la realización de la simulación se empleó MatLab versión siete.

3.0.1. Introducción

Antes de comenzar a realizar las simulaciones existe una etapa previa que consiste en obtener muestras de voz empleando las técnicas de muestreo, cuantización y codificación. En forma general el proceso de obtención de muestras de voz, parte de una señal analógica producida por el hablante y el proceso de muestreo realizado por un convertidor analógico/digital:

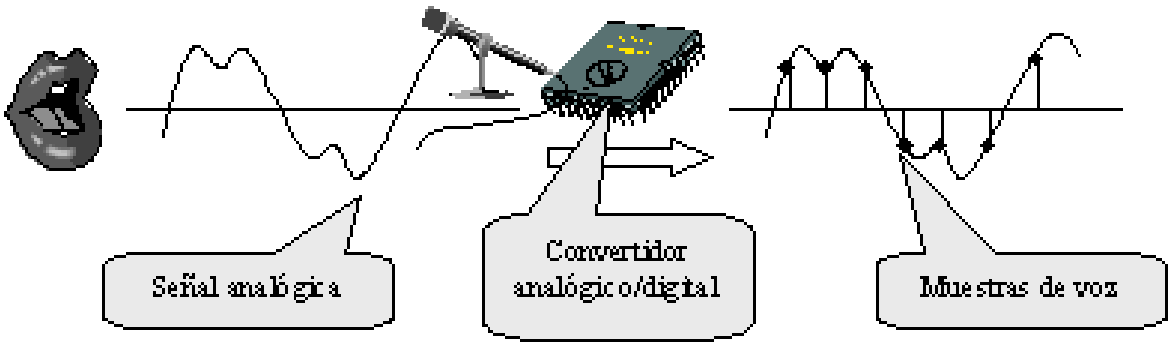

Figura 3.1: Obtención de muestras de la señal de voz.

Mientras se realiza el proceso de muestreo, los valores numéricos de cada muestra (que indican amplitudes de la señal) se almacenan en la memoria de un computador o de un hadware específico de tratamiento de señal para que puedan ser procesados. En esta fase se suelen emplear dispositivos específicos de entrada/salida par realizar el traspaso del convertidor analógico/digital a la memoria.

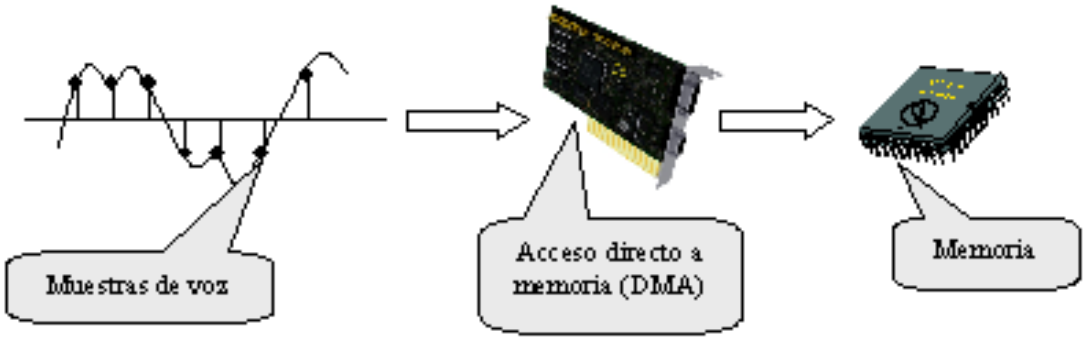

Figura 3.2: Almacenamiento de muestras de la señal de voz.

Las muestras en memoria pueden ser procesadas directamente por el computador para aplicaciones en tiempo real, o bien pueden ser almacenadas en un archivo para su uso posterior.

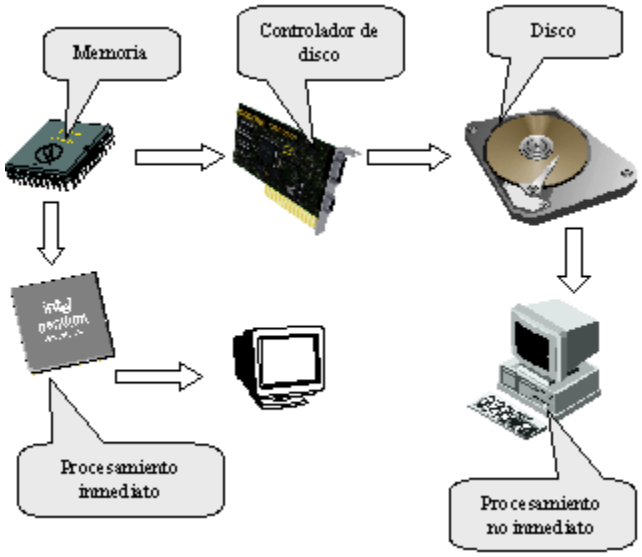

Figura 3.3: Almacenamiento y proceso de las muestras de voz.

Por otra parte, los pasos para la simulación son:

- Leer un archivo de voz.
- Secuencia de eventos(ventaneo).
- Obtener la transformada rápida de Fourier.
- Normalizar los resultados.

- Graficar los resultados.

Cuyo diagrama a bloques se presenta en la Figura 3.4

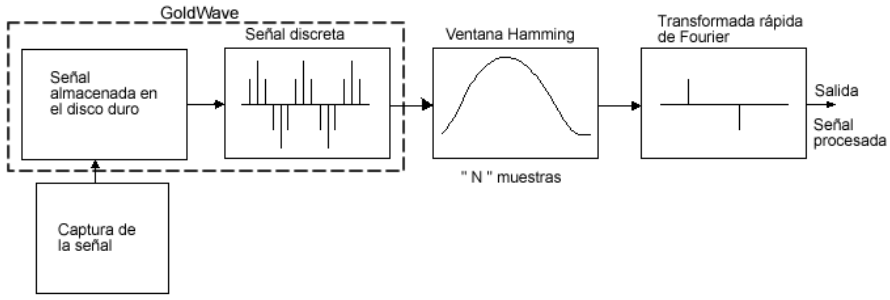

Figura 3.4: Diagrama a bloques empleado para la simulación.

Para realizar el primer paso, se hace uso del software GoldWave, la cual es una herramienta de audio que permite seleccionar la frecuencia de muestreo, el tipo de ventana, y además permite visualizar el espectro en frecuencia [2]. Otra ventaja que se obtine al usar este software, es su flexibilidad en la manipulación de los datos adquiridos a través de un micrófono, ya que permite procesar los datos de un archivo.

Observe que en el diagrama a bloques tiene una etapa de captura de datos, esta etapa es la obtención de muestras de voz, explicada en la primera parte de la introducción.

3.1. Herramientas definidas por MatLab

MatLab es un entorno de computación y desarrollo de aplicaciones totalmente integrado orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos [7].

MatLab integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo, donde los problemas y sus soluciones son expresadas del mismo modo en que se escribirían usualmente, sin necesidad de hacer uso de la programación tradicional.

MatLab dispone también en la actualidad de un amplio abanico de programas de apoyo especializados, denominados Toolboxes, que extienden significativamente el número de funciones incorporadas en el programa principal. Estos Toolboxes cubren en la actualidad prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el toolbox de proceso de señales y sistemas, que es un entorno de cálculo técnico, que se ha convertido en estándar de la industria, con capacidades no superadas en computación y visualización numérica.

MatLab tiene una gran colección de funciones para el procesamiento de señal en el Signal Processing Toolbox. Este incluye funciones para:

- Análisis de filtros digitales incluyendo respuesta en frecuencia, retardo de grupo, retardo de fase.
- Implementación de filtros, tanto directo como usando técnicas en el dominio de la frecuencia basadas en la FFT.
- Diseño de filtros IIR, incluyendo Butterworth, Chebyschev tipo I, Chebyschev tipo II y elíptico.
- Diseño de filtros FIR mediante el algoritmo óptimo de Parks-McClellan.
- Procesamiento de la transformada rápida de Fourier FFT, incluyendo la transformación para potencias de dos y su inversa, y transformada para no potencias de dos.

3.1.1. Funciones definidas por MatLab

Algunas de las funciones a utilizar para la elaboración de la primera simulación son:

fft() Realiza la transformada rápida de Fourier.

ifft() Realiza la transformada inversa de Fourier.

hamming() Da los coeficientes de una ventana Hamming de longitud $N - 1$.

wavread('nombre') Lee un archivo de audio wav y lo almacena sus valores en una variable.

zeros(size(1:n)) Realiza una matriz de ceros tamaño n.

3.2. Resultados de las simulaciones

El mayor esfuerzo de esta tesis se centra en el análisis espectral. Resulta más conveniente aplicar el análisis a porciones de voz, ya que nuestro interés es observar la evolución de los distintos parámetros calculados; por ello se procesan porciones o ventanas de la señal. Tomando los pasos que se hicieron referencia en la introducción se procede.

3.2.1. Lectura de un archivo de voz

Nuestra señal a analizar es una señal de voz, esta señal fue grabada mediante GoldWave como un archivo mono a una frecuencia de muestreo de 11025Hz, la cual se muestra en la Figura 3.5.

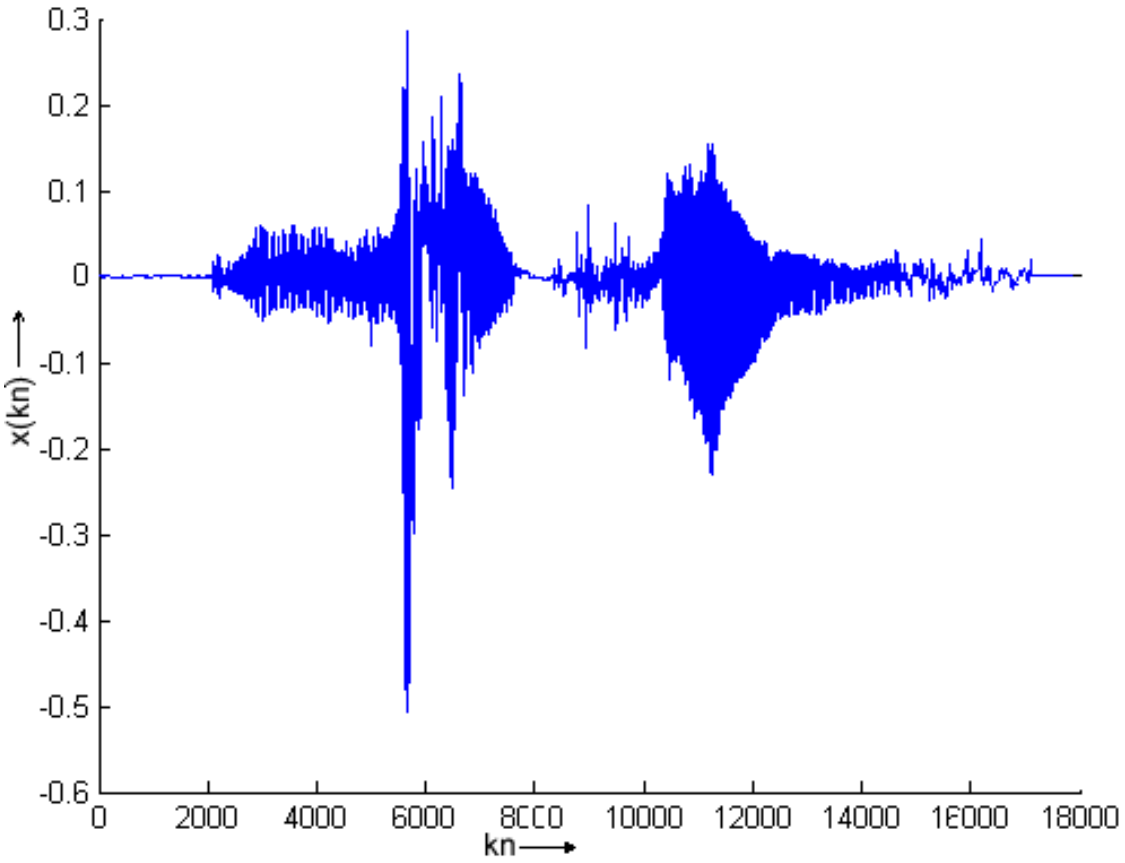

Figura 3.5: Señal de voz a ser procesada, el cual almacena la palabra "puerta".

Una vez que ya se tiene la señal almacenada en un archivo, se procede a leer el archivo y almacenarla en una variable. La secuencia de comandos para la lectura mediante MatLab es:

MATLAB 3.1: Adquisición de la señal de voz

```
[x,Fs,Nb]=wavread(\lq puerta');  % Lee un archivo de audio y almacena
[ns,nc]=size(x);
```

Los valores de amplitud del comando $wavread('nombre')$, se almacenan en la variable y; el valor de la frecuencia de muestreo en Fs y el número de bits en Nb.

En el comando $size(variable)$, ns corresponde al número de filas y nc al número de columnas.

3.2.2. Realización ventaneo

Antes de realizar la secuencia se deben definir los tipos ventanas a utilizar.

3.2.2.1. Tipos de ventanas

Una ventana rectangular se define como:

$$h(t) = \begin{cases} 1, & |t| \leq \frac{T_o}{2}, \\ 0, & |t| > \frac{T_o}{2}, \end{cases}$$

el comando que permite realizar una ventana rectangular es:

MATLAB 3.2: Ventana rectangular

```
w=ones(n,1);
plot(w);
```

Una ventana Hamming, en términos matemáticos se define como:

$$h(t) = \begin{cases} 0.54 + 0.46 \cos \frac{2\pi t}{T_o}, & |t| \leq \frac{T_o}{2}, \\ 0, & |t| > \frac{T_o}{2} \end{cases}$$

el comando que permite realizar una ventana Hamming es:

MATLAB 3.3: Ventana hamming.

```
w=hamming(n);
plot(w);
```

Donde n para ambas ventanas es la longitud. Su respuesta en de frecuencia para este tipos de ventanas es, ver Figura 3.6

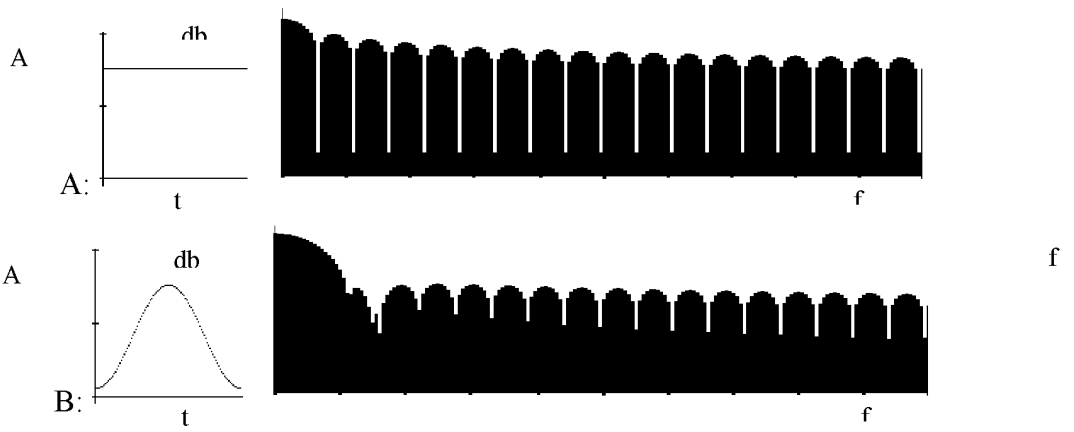

Figura 3.6: Respuesta en frecuencia de ventana rectangular y Hamming.

3.2.2.2. Secuencia de eventos (ventaneo)

El mecanismo de ventaneo se ilustra en la Figura 3.7. A cada porción de señal de voz(tamaño deseado) se le asigna una ventana, de tal forma que las muestras queden ponderadas con los valores de la función escogida (ventana Hamming). En este caso, las muestras que se encuentran en los extremos de la ventana tienen un peso mucho menor que los que se hallan en el medio, lo cual es adecuado para evitar que características de los extremos del bloque varíen interpolar la la función de ventana Hamming.

Si las ventanas son muy pequeñas nos proporcionan excesivos detalles de la evolución, y si son muy grandes podemos perder detalles significativos. Una interpretación matemática es:

$$T_v = I_v + S_v. \tag{3.1}$$

Donde:

Tv Tamaño de la ventana.

Iv Incremento de la ventana.

Sv Solapamiento de la ventana.

Se puede deducir, si no se aplicaran el corrimiento de ventanas el sistema se volvería lento y tardaría mucho tiempo en ejecutar todas las muestras. Es por ello que se debe aplicar el sistema de ventaneo. El código para la secuencia de la señal señal es:

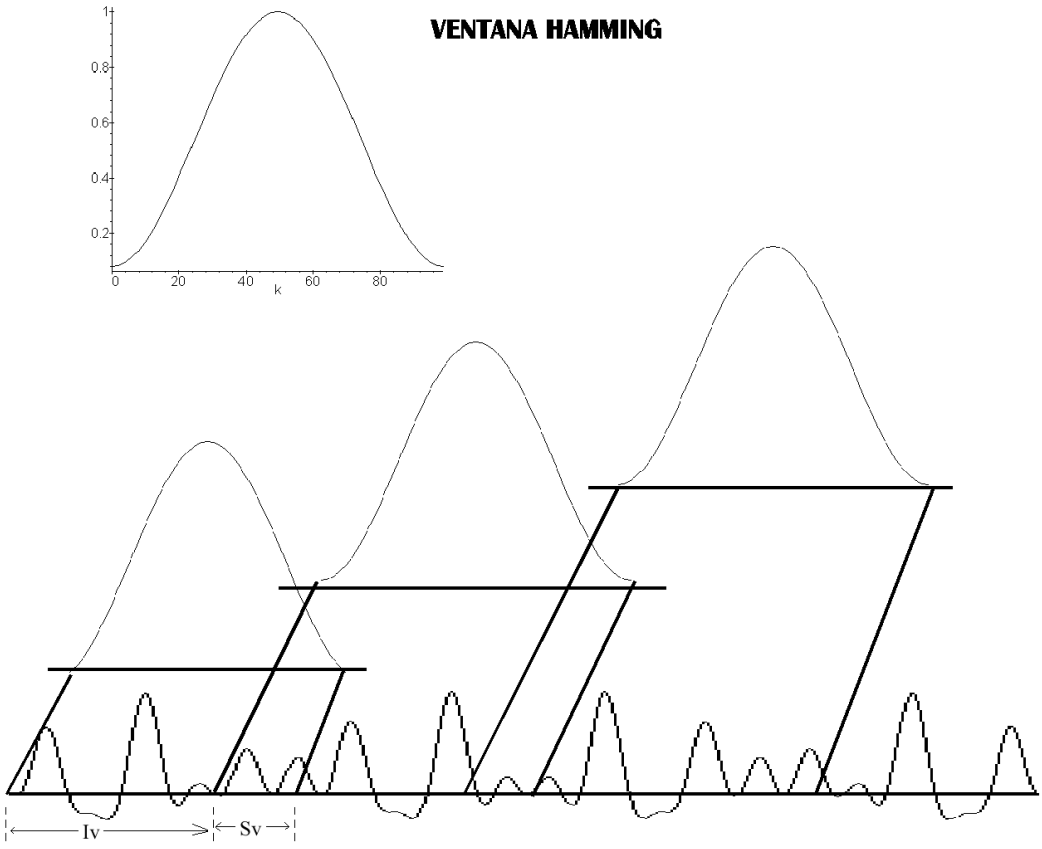

Figura 3.7: Proceso de ventaneo.

MATLAB 3.4: Secuencia ventaneo de la señal, primera parte.

```
iv=2756;                          %Define valores para desplazar ventana
sv=0;
tv=iv+sv;

h=hamming(tv);                    %Define ventana Hamming
x=zeros(size(1:tv));
va=zeros(size(1:tv));
m=zeros(size(1:4));

i=1;
j=0;
nv=0;
for k=1:ns;                          %Secuencia ventaneo

    if(i==(iv+1))
      j=1;
    end

     if(i<=tv)
       va(i)=y(k);
       if(i==tv)           %inicia ventanas impares
         nv=nv+1;

         for i =1:tv ;     %Coeficientes ventana Hamming
           x(i) = h(i) * va(i) ;
         end

         subplot(4,1,nv)
         plot(x)


       end
       i=1+i;
     end


     if((j>=1)&&(j<=tv))
       ba(j)=y(k);
       if(tv==j)        %inicia ventanas pares
         nv=nv+1;

         for i =1:tv ; %Coeficientes ventana Hamming
           x(i) = h(i) * ba(i) ;
         end

         subplot(4,1,nv)
         plot(x)

       end
```

Este algoritmo se aplica la secuencia de ventaneo a la señal de la Figura 3.5. Con un incremento de $i_v = 2756 \quad muestras$, valor de solapamiento $s_v = 0 \quad$, por lo que da un tamaño de ventana igual a $T_v = 2756 \quad muestra$, en total son cuatro ventanas. Graficando da como resultado ver Figura 3.8.

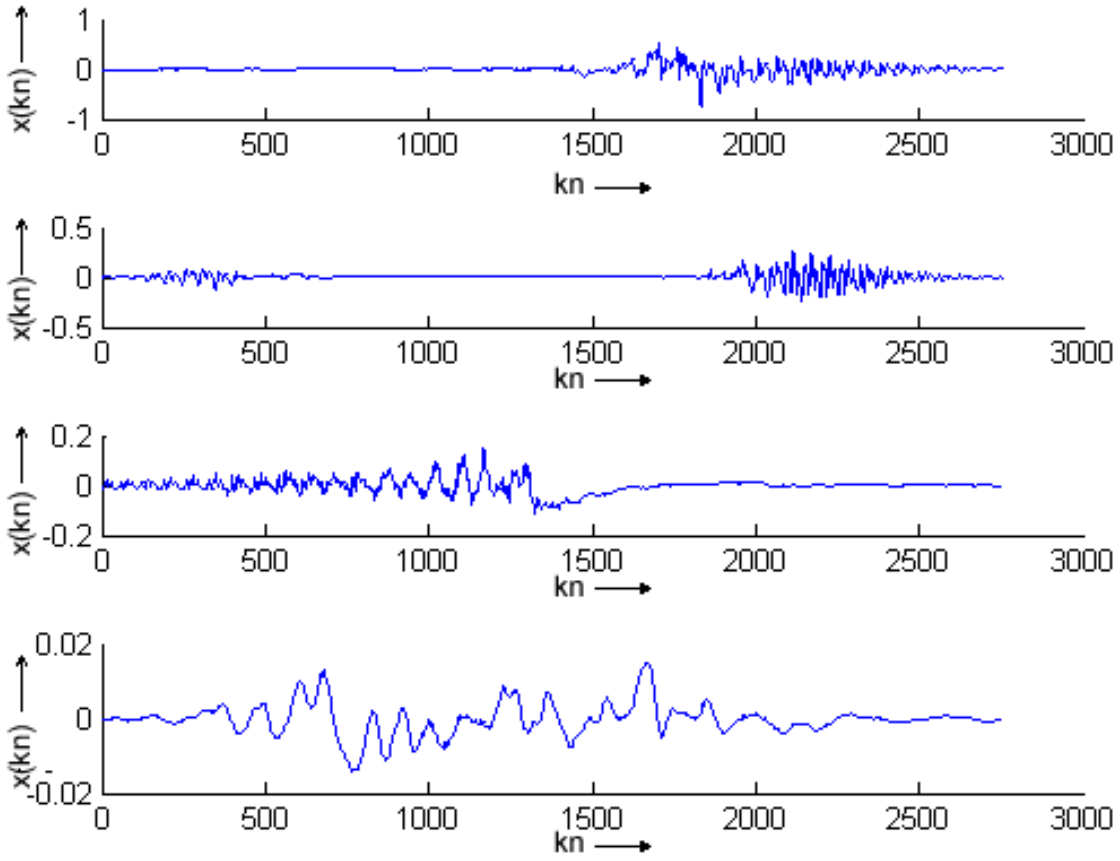

Figura 3.8: Ventaneo de la señal.

`Nota:` En el desarrollo de este algoritmo, el comando *zeros(size(1:n)* dentro MatLab, construye una matriz de ceros, donde *size* es el tamaño de la matriz.

MATLAB 3.5: Secuencia ventaneo de la señal, segunda parte.

```
          j=j+1;
          if(sv==0)
           i=i+1;
          end

        end

        if(j==(iv+1))
          i=1;
        end

        if((sv==0)&&(j==(tv+1)))
          j=0;
        end

  end
```

3.2.3. Normalización de resultados

Para normalizar los resultados se debe convertir el valor máximo de la función calculada en el máximo permitido (H), manteniendo la proporción con el resto de valores. Si f_{max} fuera el valor máximo entre todos los valores de $f(i)$ y H la altura de la ventana se normalizaría mediante:

$$f(i) = parte\_entera(H\frac{f(i)}{f_{max}}), \quad \forall i \tag{3.2}$$

A modo de ejemplo se normalizarán las siguientes operaciones matemáticas.

3.2.4. Obtención de la transformada rápida de Fourier

El siguiente paso es aplicar la transformada discreta de Fourier base dos, de secuencia finita x(n), matemáticamente se puede expresarlo mediante la ecuación:

$$X(K) = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \qquad k = 0, 1, 2, ..., N-1. \tag{3.3}$$

donde:

$$W_N = e^{-2j\frac{\pi}{N}}, \qquad k = 0, 1, 2, ..., N-1. \tag{3.4}$$

Al aplicar la transformada de Fourier con el ventaneo se convierte en la transformada de tiempo corto de Fourier, la secuencia de ventaneo

$$x_m(n),$$

se define como:

$$x_m(n) = x(n)w(m-n), \tag{3.5}$$

donde $w(n)$ es la ventana que determina el bloque de la secuencia discreta donde se aplica la transformada discreta de Fourier para un índice de tiempo m. La secuencia quedaría como:

$$X_m(k) = \sum_{n=0}^{N-1} w(m-n)x(n)W_n^{kn}, \qquad k = 0, 1, 2, ..., N-1. \tag{3.6}$$

La longitud de la ventana $w(n)$ determina la resolución de la transformada discreta de Fourier en tiempo corto. Dependiendo de la longitud de la ventana, corta o larga. El espectro del sonido se clasifica en banda ancha y en banda angosta. Si se tiene una ventana angosta se obtiene un espectro de banda ancha que se caracteriza por una mala resolución en frecuencia, y una buena resolución en tiempo; si se utiliza una ventana larga, se obtiene un espectro de banda angosta el cual se caracteriza por tener una buena resolución en frecuencia, pero un decremento en la resolución temporal.

Los comandos para graficar el espectro de frecuencia es MatLab es:

MATLAB 3.6: Obtención de la transformada rápida de Fourier.

```
X=fft(señal);
f=abs(X);
plot(f)
```

Este código de MatLab se inserta después que se a multiplicado por la venta par e impar, se puede ver la Figura 3.9.

3.3. Simulación: filtros

Para la simulación de cualquier tipo de filtro es necesario conocer el espectro de frecuencia de la señal, éste se obtiene mediante la transformada Fourier descrita en la sección anterior. Una vez obtenido las componentes de frecuencia de la señal, se debe tomar la decisión si el filtro es pasabajas, pasa alta y pasa banda, según los requerimientos.

Continuando con la simulación de algoritmos, se añade ruido a la señal como se muestra en el siguiente diagrama:

El objetivo de añadir ruido a la señal es comprobar que realmente este trabajando el filtro digital.

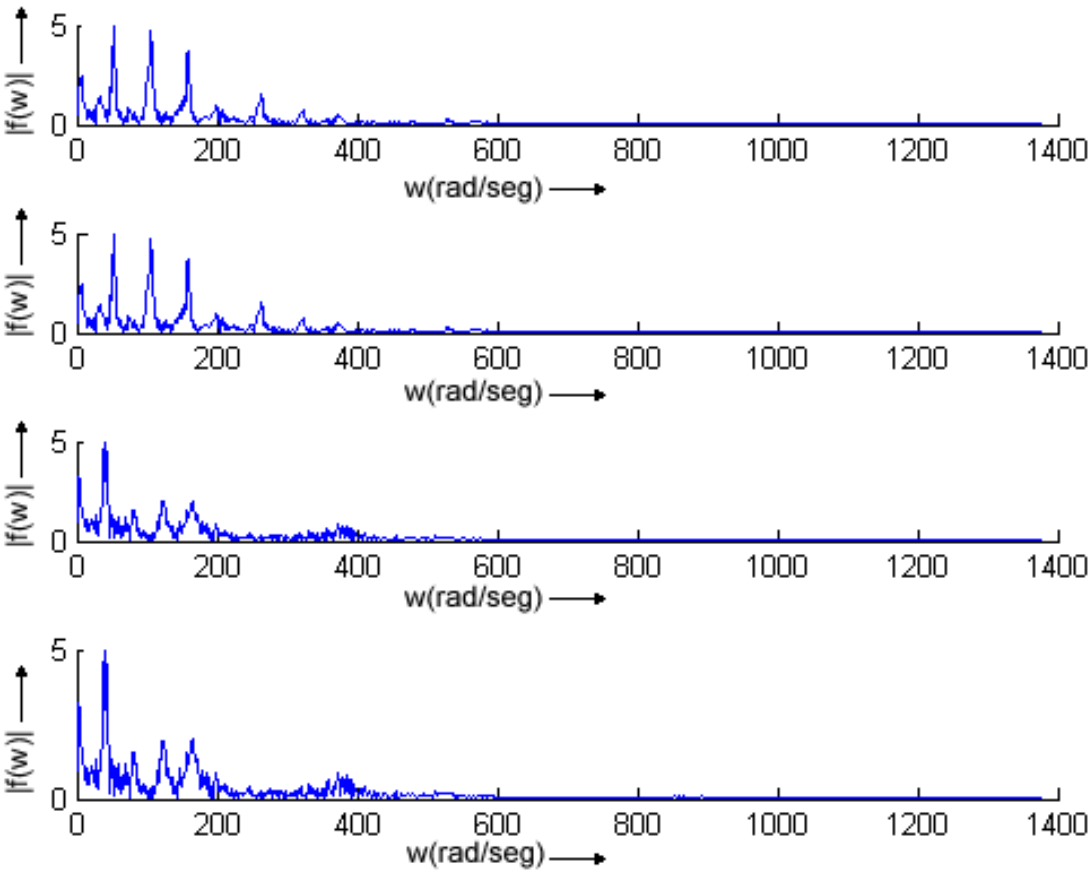

Figura 3.9: Espectro de magnitud normalizado por cada ventana.

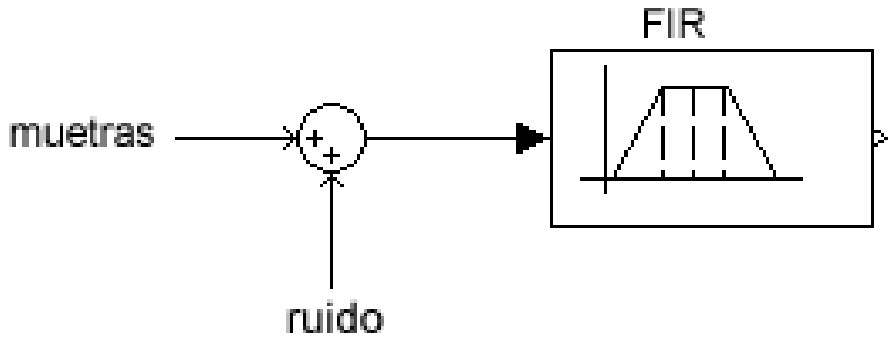

Figura 3.10: Diagrama a bloques para el filtro.

Funciones de MatLab para realizar filtros FIR

MatLab dispone de funciones que facilitan el diseño de filtros digitales. Algunas de las funciones en MatLab son:

- **FIR1** `>> B = fir1(N,Wn,type,window);`

  Diseña un filtro FIR pasobajo de orden N (longitud $N+1$) y frecuencia de corte Wn (normalizada con respecto a la frecuencia de Nyquists, $0 \leq Wn \leq 1$). Se pueden especificar otro tipo de filtros de la misma forma que con los filtros IIR mediante el parámetro type. Por ejemplo, para un filtro parabanda:

  ```
  >> B = fir1(N,[W1 W2],'stop');
  ```

  Por defecto la función FIR usa la ventana de Hamming. Otro tipo de ventanas pueden también especificarse:

- **FIR2**

  ```
  >> B = fir2(N,F,M,window);
  ```

  Diseña un filtro FIR utilizando el método del muestreo frecuencial. Los parámetros de entrada es el orden del filtro N (longitud $N+1$) y dos vectores F y M

que especifican la frecuencia y la magnitud, de forma que `plot(F,M)` es una gráfica de la respuesta deseada del filtro.

Se pueden indicar saltos bruscos en la respuesta frecuencial duplicando el valor de la frecuencia de corte.

F debe estar entre 0 y 1, en orden creciente, siendo el primer elemento igual a 0 y el último 1. El parámetro window indica el tipo de ventana a utilizar. Por defecto, usa la ventana de Hamming.

- **FIRLS**

  `>> B = firls(N,F,M);`

  Diseño de filtros FIR usando la minimización del error por mínimos cuadrados. Los argumentos de entrada son el orden del filtro N, y dos vectores F y M, cuyo formato difiere de los análogos en la función `fir2`. El filtro obtenido es la mejor aproximación a (F, M) por mínimos cuadrados.

  F es un vector que indica los límites de las bandas de frecuencia en parejas (por tanto el tamaño de F debe ser par), y en orden ascendente entre 0 y 1. M es un vector del mismo tamaño que F que indica la magnitud deseada para cada banda de frecuencias. La respuesta deseada es la línea que conecta los puntos $(F(k), M(k))$ y $(F(k+1), M(k+1))$ para k impar.

  `>> B = firls(N,F,M,W);`

  Donde B para cada caso anterior son los coeficientes del filtro.

- **freqz**

  `>> B = freqz(b,a,N);`

  Calcula N puntos de la respuesta de frecuencia del filtro definido por b y a

  `>> h = freqz(b,a,f,fs);`

  Regresa la respuesta de frecuencia en el vector h , donde f es la frecuencia de la señal, fs es la frecuencia de muestreo y $b - a$ son los coeficientes del filtro.

3.3.1. Generación del ruido de alta frecuencia

Como se mostró en el diagrama de bloques se añade ruido, este ruido debe ser de alta frecuencia, es decir mayor al espectro de frecuencia de la señal original, para obtener ruido de alta frecuencia, se siguió:

1. Generación de señal aleatoria.
2. Diseño de un filtro digital pasa alta.
3. Obtención de la señal.

Cuyo código en MatLab es:

MATLAB 3.7: Generación de la señal aleatoria

```
%Parámetros
Fs=11050;     %Frecuencia de muestreo
T=1/Fs;       %Periodo de muestra
L=1024;       %Longitud de la señal
t=(0:L-1)*T; %Tiempo del vector


%Señal aleatoria
y=0.5*randn(size(t));
```

Ver Figura 3.11,

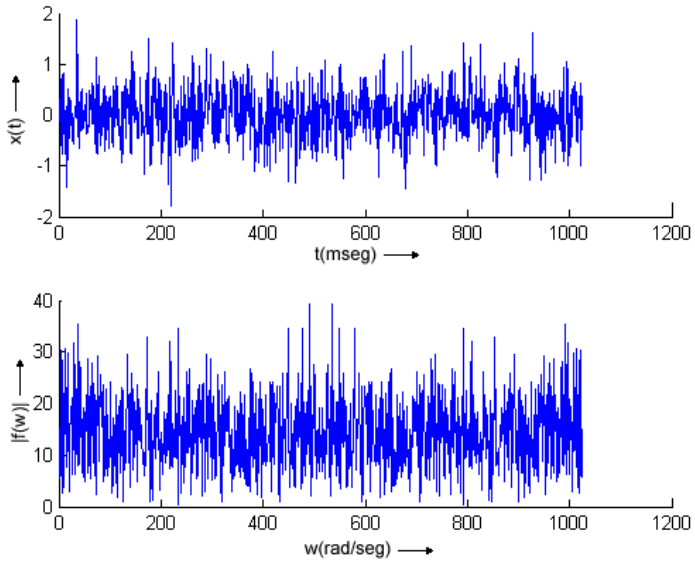

Figura 3.11: Señal aleatoria y su espectro de frecuencia.

MATLAB 3.8: Diseño del filtro digital pasa alta

```
%filtro pasa altas
N=44;  %orden del filtro

Fny=Fs/2;
Bfir1=fir1(N,5000/Fny,'high');
F=0:10:5000;
H=abs(freqz(Bfir1,1,F,Fs));

semilogy(F,H,'r')

title('Respuesta de frecuencia');
```

Ver Figura 3.12,

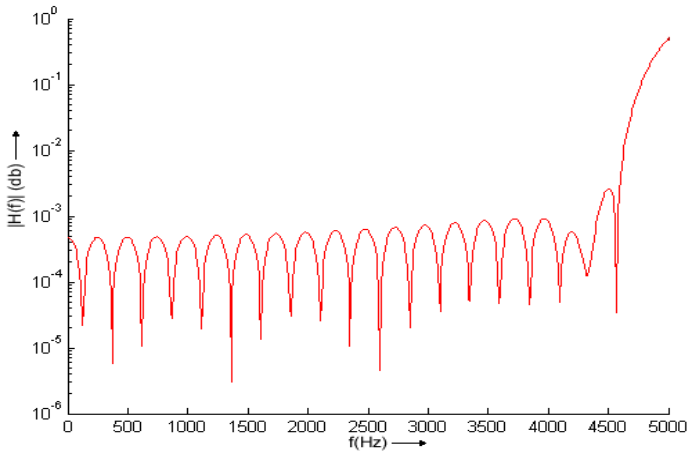

Figura 3.12: Respuesta en frecuencia del filtro.

MATLAB 3.9: Obtención ruido alta frecuencia

```
%comprobación filtro pasa altas
r = conv(Bfir1,y);

plot(r);
title('ruido de alta frecuencia')
```

ver Figura 3.13.

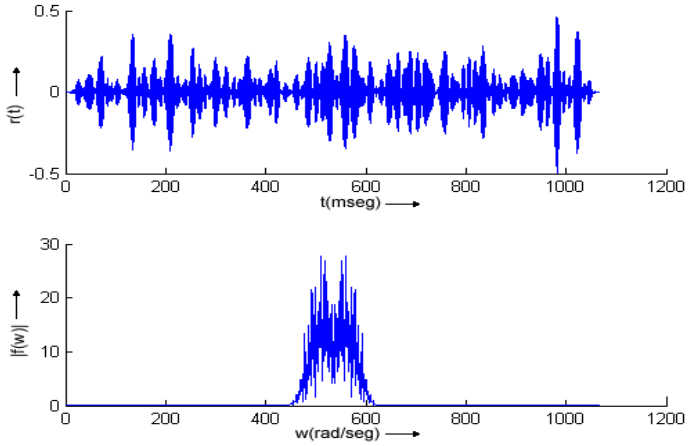

Figura 3.13: Ruido de alta frecuencia y su espectro.

3.3.2. Señal muestreada

La señal muestreada es una porción de nuestra primera señal, Figura 3.5, solo que aquí se toman 1024 muestras, el código en MatLab es:

MATLAB 3.10: Señal muestreada y espectro

```
%señal de voz capturada
[z,Fs,Nb]=wavread('a',1024);

%espectro de la señal de voz
esvoz=fft(z);

plot(abs(esvoz));
```

Ver Figura 3.14

3.3.3. Suma de señales

El siguiente paso es sumar las señales: señal ruido de alta frecuencia y señal de la voz,

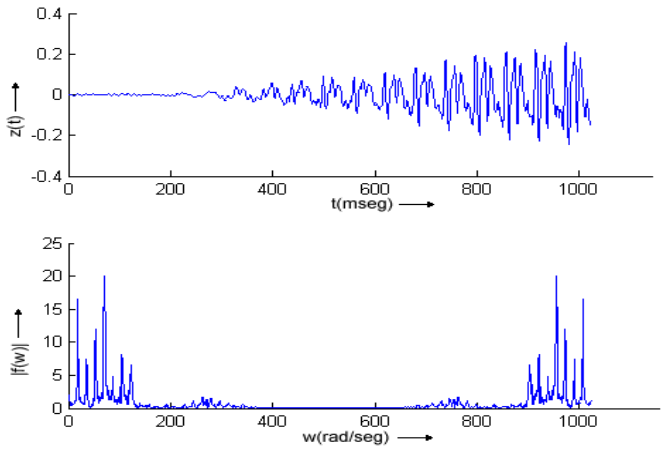

Figura 3.14: Señal de voz muestreada y su espectro.

MATLAB 3.11: Suma de señales

```
%suma señales
for i =1:1024;
     sx(i) = r(i) + z(i) ;
end
```

Ver la Figura 3.15,

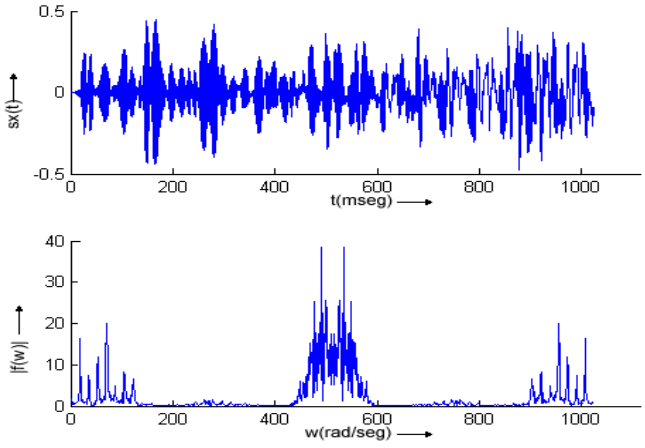

Figura 3.15: Señal de voz con ruido y su espectro.

3.3.4. Obtención señal filtrada

Para el caso de la señal de voz, el rango de frecuencias donde se encuentra la información para su estudio, es de $0.3khz$ a $3.4khz$. Se va a diseñar un filtro digital con estas características:

MATLAB 3.12: Diseño del filtro digital para voz

```
%filtro paso de banda
Bfira=fir1(N,[30 3500]/Fny);

Hfir2=abs(freqz(Bfira,1,F,Fs));

semilogy(F,Hfir2,'r')
title('Respuesta de frecuencia');
```

ver Figura 3.16;

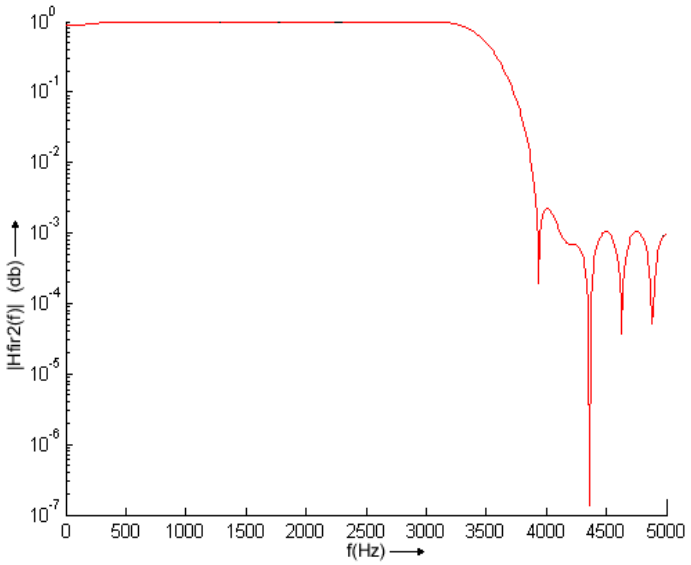

Figura 3.16: Respuesta de frecuencia del filtro.

una vez que ya se ha diseñado este tipo de filtro FIR, hay que comprobar su funcionamiento. Esto se hace con la convolución los coeficientes FIR obtenidos con la señal de voz - ruido:

MATLAB 3.13: Comprobación del filtro

```
%comprobación del filtro digital
m=conv(Bfira,sh);
```

3.4. Comentarios y referencias

MatLab ofrece muchas herramientas para realizar la funcionalidad indispensable en procesamiento de señales, tales como Transformadas Rápidas Fourier y Transformadas Rápidas Inversas de Fourier. La visualización de datos de procesamiento de señales está soportada por funciones tales como gráficos stem y periodogramas. El lenguaje de MatLab, inherentemente orientado a matrices hace que la expresión de coeficientes de filtros y demoras de buffers sean muy simples de expresar y comprender. Un tutorial completo acerca de MatLab es el del autor Professor Tilbury.

La siguiente bibliografía es muy útil para el procesamiento digital de señales: Vinay, K. I., Proakis, J. G. Digital signal processor using MatLab y PWS Publishing Company.

Capítulo 4

Implantación

El objetivo de este capítulo es presentar los resultados que se obtienen de la implantación de los algoritmos diseñados para el procesamiento digital de voz, empleando un DSP.

Para la elaboración del programa deben tomarse eventos, que permitan convertir la función en finita[1].

4.1. Introducción

El diagrama a bloques para la implantación se muestra en la Figura 4.1.

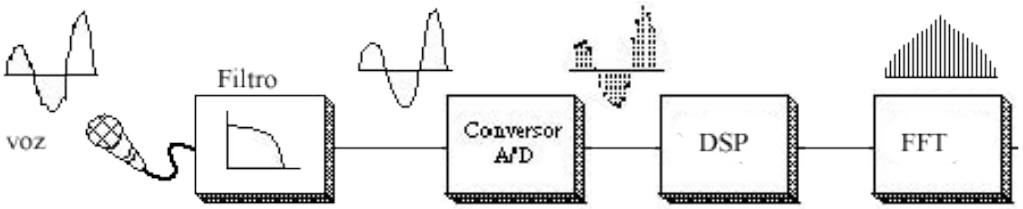

Figura 4.1: Diagrama a bloques de la implantación

Observe, que en esta etapa incluye un filtro, esto es debido a que se obtiene de un proceso real, aunque no hay que olvidar que la tarjeta de procesador de audio TMS320C6711 ya tiene incluido un filtro. Para esta segunda etapa, el algoritmo de la transformada rápida de Fourier es de base dos, ya está elaborado en una librería del Code Composer Studio, solo hay que desarrollar las secuencias que se pide; también se pide dar de alta las librerías necesarias.

4.2. Configuración de puertos

El algoritmo implantado contiene la siguiente secuencia de etapas, ver diagrama que se muestra en la Figura 4.2

- En la primera parte, se trata de dar arranque a los elementos encargados para el procesamiento de voz, esto se hace mediante la rutina DSS_init, después; mientras el sistema detecte alguna interrupción saltara a la rutina DSS_isr. Todo lo anterior se incluye en una rutina principal denominada *Main*.
- El proceso DSS_init, inicializa y configura los elementos implicados para el desarrollo del proceso, correspondiente a los modulo: codificador/decodificador de audio(CODEC), puerto serial multicanal(McBSP) y además, activa el servicio de interrupción para después regresar a Main y preguntar si existe alguna.
- Cada vez que el registro de recepción de datos (DRR), activa un servicio de interrupción desde el McBSP, en el ciclo *Main*, inicialización el vector de interrupciones, mapeo de la interrupción 11 del procesador utilizado. Para dar paso a las instrucciones contenidas en la función DSS_ISR .
- Por último, cuando el buffer a completa su valor, proveniente de aplicar los algoritmos, se gráfican los valores obtenidos de la rutina anterior. Todo esto se realiza en la rutina *processing* y después regresa a main, mientras haya interrupcones.

Las primeras muestra obtenidas del DSP se muestran en la Figura 4.3, en esta parte se aplico el algoritmo de ventaneo, explicado en el capitulo anterior, el código del programa se muestra en al Apéndice A.

4.3. Proceso de implantación

Una vez configurado los puertos, se van a implantar los algoritmos en tiempo real que se mostraron en el capítulo anterior. Los pasos son:

- Ventaneo tipo Hamming, con traslape de 30 muestras, en el segmento de señal adquirido (cuya apertura es de 128 datos), para reducir sensibilidad a los cambios presentados en extremos de ventana.
- Posteriormente se filtra la señal, se usa un FIR, con 44 coeficientes.

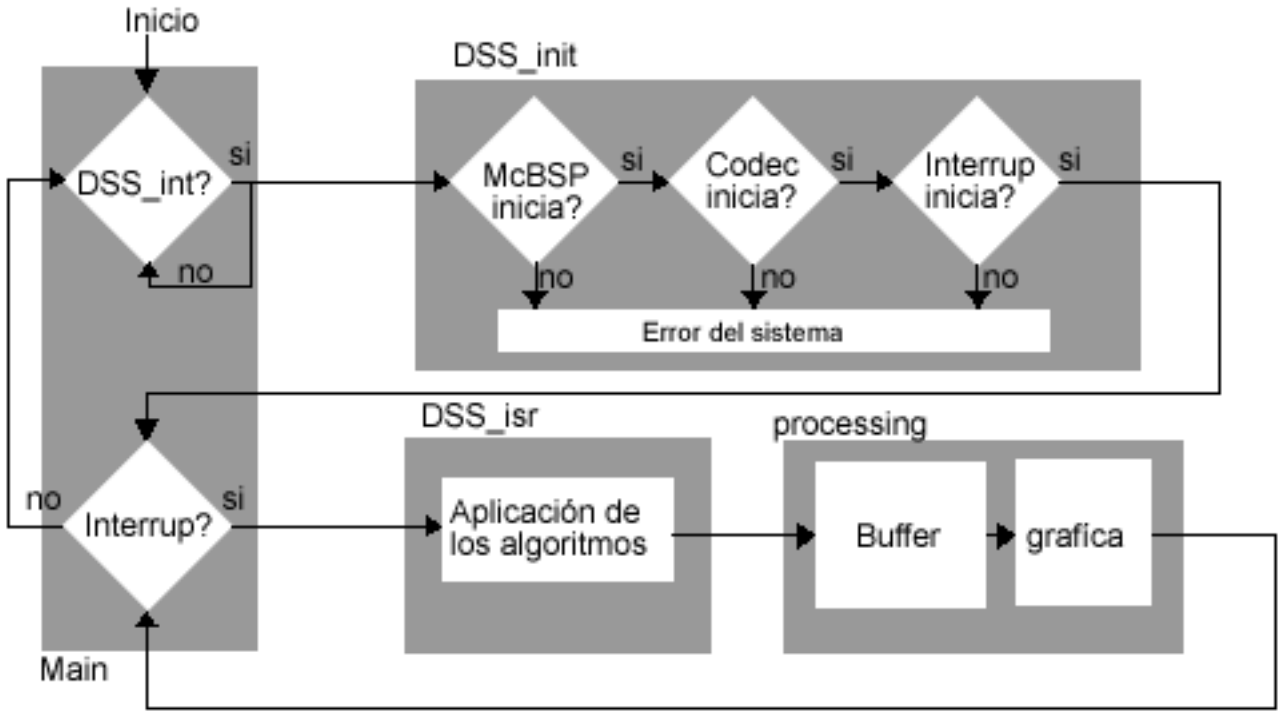

Figura 4.2: Diagrama de flujo para la configuración de puertos

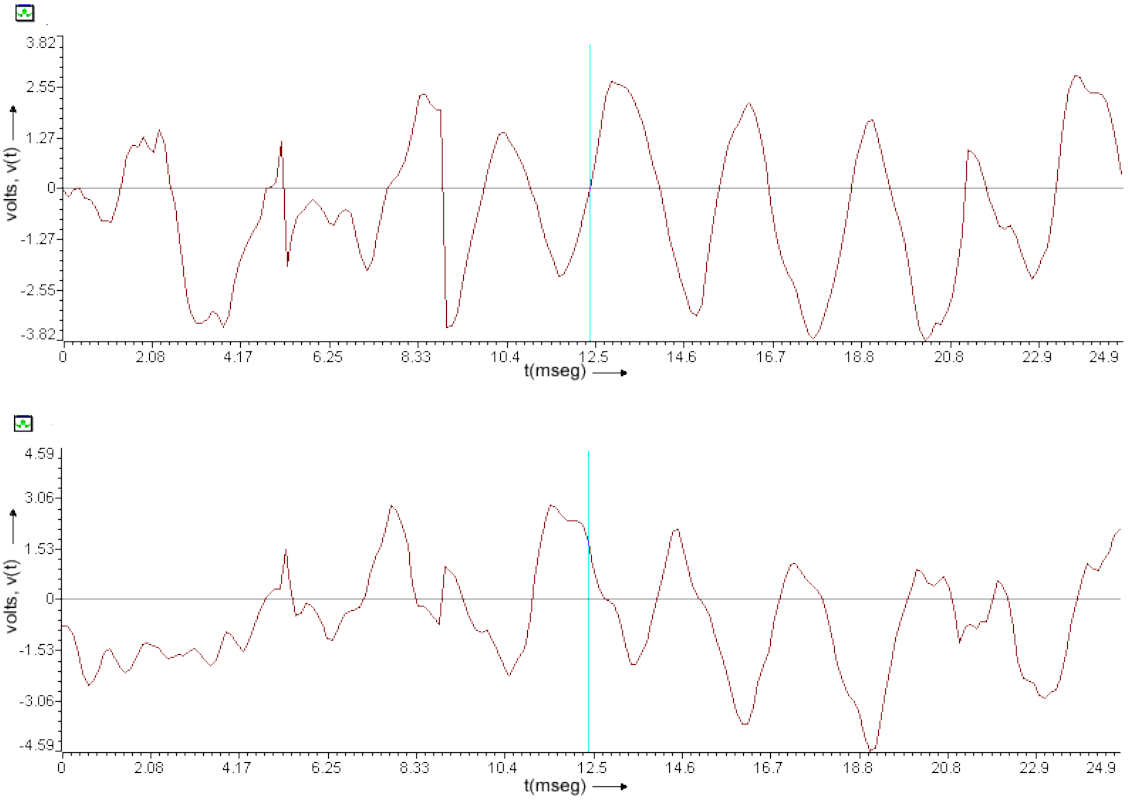

Figura 4.3: Muestras obtenidas tras el ventaneo, en a) muestras impares y b) muestras pares

- Se aplica la transformada rápida de Fourier (FFT), con resolución de 2048 puntos, sobre dicho segmento.
- Se normaliza.
- Se gráfica, en esta parte se utiliza un buffer y un reset para el vector.

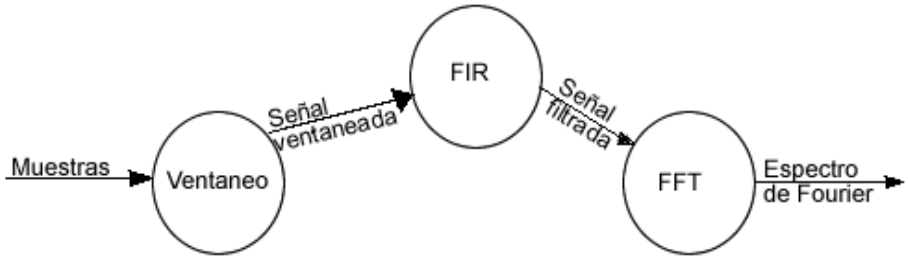

Figura 4.4: Diagrama de implantación de algoritmos.

4.4. Funciones definidas en el code composer studio(CCS)

Algunas de las funciones que se pueden encontrar en la librería del CCS son:

bitrev esta función regresa el número de bits a su posición original.

digitrev_index esta función indexa los datos para optimizar las operaciones.

radix2 con esta función se obtiene la transformada rápida de Fourier.

fltoq15 convierte un número de tipo float a short.

fir_gen realiza el algoritmo FIR por medio de la convolución.

4.4.1. Implantación de la FFT base-2

Antes de implantar dicho algoritmo, se explica a grandes rasgos el algoritmo que se maneja en las librerías del DSP.

Existen dos métodos para la realización de este algoritmo: diezmado en el tiempo y diezmado en la frecuencia. El algoritmo que se desarrolla en el DSP es la FFT base 2 diezmado en la frecuencia.

Algoritmo mariposa

Cada mariposa implica una multiplicación y dos sumas complejas. Para $N = 2^{\nu}$, se tiene $N/2$ mariposas por cada etapa del proceso de cálculo y $log_2 N$ etapas. Por lo tanto, como se indico anteriormente, el número total de multiplicaciones complejas es ($\frac{N}{2} log_2 N$ y de sumas complejas es $N log_2 N$.

Cada vez que se realiza una mariposa sobre un par de números complejos (a, b) para obtener (A, B) como se muestra en la Figura 4.5, no es necesario guardar el par de entrada (a, b). Por lo tanto se puede guardar el resultado (A, B) en las mismas posiciones que (a, b). En consecuencia, se necesita una cantidad fija de memoria, en concreto, $2N$ registros de almacenamiento, para guardar los resultados (N números complejos) de los cálculos correspondientes a cada etapa. Dado que durante todo el cálculo de la DFT de N puntos se usan las mismas $2N$ posiciones de memoria decimos que los cálculos se hacen *in situ*.

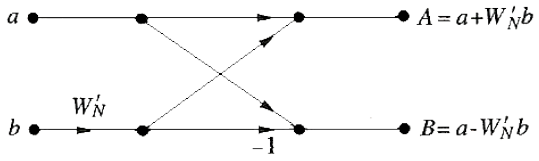

Figura 4.5: Mariposa básica del algoritmo para la FFT.

Algoritmo de base dos diezmado en frecuencia

Esta elección implica un almacenamiento por columnas de la secuencia de datos de entrada. Para deducir el algoritmo se empieza dividiendo la fórmula de la DFT en dos sumatorios, uno de los cuales contiene los primeros $\frac{N}{2}$ puntos de datos. Así se obtiene:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{kn}. \tag{4.1}$$

$$\tag{4.2}$$

Si la secuencia de entrada la separamos en dos mitades:

$$x(0), x(2), ..., x(\frac{N}{2} - 1)$$

$$x(\frac{N}{2}), x(\frac{N}{2}+1), ..., x(N-1)$$

la ecuación de la transformada discreta la podemos separar en dos puntos:

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{kn} + W_N^{\frac{Nk}{2}} \sum_{n=0}^{\frac{N}{2}-1} x(n+\frac{N}{2})W_N^{kn}. \tag{4.3}$$

Dado que $W_N^{\frac{kN}{2}} = (-1)^k$, la expresión (4.1) puede escribirse como:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + (-1)^k x(n+\frac{N}{2})]W_N^{kn}. \tag{4.4}$$

Se divide ahora(diezmamos) $X(k)$ en las muestras pares e impares. De esta manera, se obtiene:

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+\frac{N}{2})]W_{\frac{N}{2}}^{kn}, \tag{4.5}$$

y

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}} -1[x(n) + x(n+\frac{N}{2})]W_{\frac{N}{2}}^{kn}W_N^n, \tag{4.6}$$

donde hemos usado el hecho de que $W_N^2 = W^{\frac{N}{2}}$. Si se define las secuencias de $\frac{N}{2}$ puntos $g_1(n)$ y $g_2(n)$ como:

$$g_1(n) = x(n) + x(n+\frac{N}{2}), \tag{4.7}$$

y

$$g_2(n) = [x(n) - x(n+\frac{N}{2})]W_N^n \qquad n = 0, 1, 2, ..., \frac{N}{2}-1 \tag{4.8}$$

entonces:

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} g_1(n)W_{\frac{N}{2}}^{kn} \tag{4.9}$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} g_2(n) W_{\frac{N}{2}}^{kn} \tag{4.10}$$

El cálculo de las secuencia $g_1(n)$ y $g_2(n)$ según (4.8), y el uso de estas secuencias para el cálculo de las DFTs de $\frac{N}{2}$ puntos se muestra en la Figura 4.6.

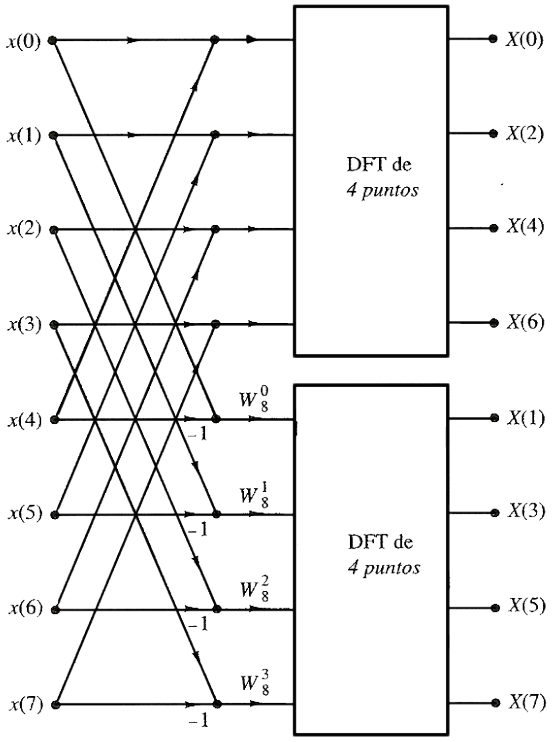

Figura 4.6: Primera etapa para la FFT de diezmado en frecuencia.

Observé que el cálculo básico en esta figura es el mariposa mostrada en la Figura 4.5.

Este procedimiento computacional puede retirarse diezmando las DFTs de $\frac{N}{2}$ puntos, $X(2k)$ y $X(2k+1)$. El proceso conlleva $\nu = log_2 N$ etapas de diezmado, donde cada etapa implica $\frac{N}{2}$ mariposas como la de la Figura 4.5. Consecuentemente, el cálculo de la DFT de N puntos por medio del algoritmo para la FFT de diezmado en frecuencia, requiere $(\frac{N}{2})log_2 N$ multiplicaciones complejas y $Nlog_2 N$ sumas complejas, igual que el algoritmo de diezmado en el tiempo.

Un **segundo aspecto** importante tiene que ver con el orden de la secuencia de entrada después de haber sido diezmado $(\nu - 1)$ veces. Por ejemplo,

si se considera $N = 8$, sabemos que el primer diezmado nos da la secuencia $x(0), x(2), x(4), x(6), x(1), x(3), x(5), x(7)$, y que el segundo diezmado da lugar a la secuencia $x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)$. Este mezclado de la secuencia de entrada tiene un orden bien determinado, como puede observarse en la Figura 4.7 y tabla 4.1, que muestra el diezmado de una secuencia de ocho puntos. Si se expresa el índice n, de la secuencia diezmada leyendo la representación binaria de n al revés. Por lo tanto, el punto $x(3) \equiv x(011)$ se encontrará en la posición $m = 110$ ó $m = 6$ de la secuencia diezmada. Por tanto, se decide que la secuencia $x(n)$ después del diezmado se almacena en orden binario invertido.

Si la secuencia de datos se almacena en orden binario invertido, y las mariposas se realizan *in situ*, la DFT resultante, $X(k)$, se obtiene en orden natural (es decir, $k = 0, 1, ..., N - 1$). Por otra parte, debemos decir que es posible realizar el algoritmo para la FFT de manera que la entrada esté en orden natural y la salida en orden binario invertido. Además, se puede imponer la restricción de que tanto la entrada como la salida estén en orden natural, y deducir un algoritmo para la FFT en el que los cálculos no se haga *in situ*. Tal algoritmo requeriría almacenamiento adicional.

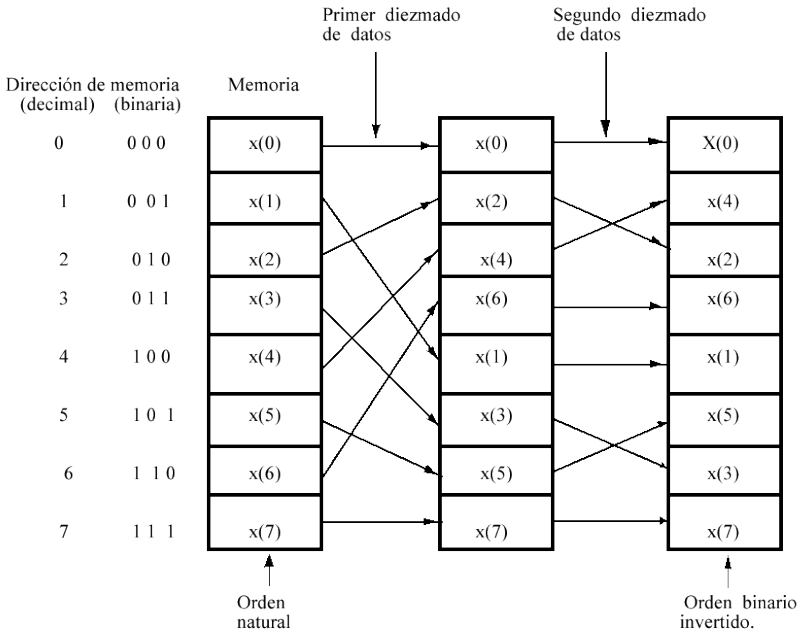

Figura 4.7: Mezclado de datos.

Cuadro 4.1: Inversión binaria.

$(\ n_2 n_1 n_0) \rightarrow (n_0 n_2 n_1) \rightarrow (n_0 n_1 n_2)$
$(\ 0\ 0\ 0\ ) \rightarrow (000) \rightarrow (000)$
$(\ 0\ 0\ 1\ ) \rightarrow (100) \rightarrow (100)$
$(\ 0\ 1\ 0\ ) \rightarrow (001) \rightarrow (010)$
$(\ 0\ 1\ 1\ ) \rightarrow (101) \rightarrow (110)$
$(\ 1\ 0\ 0\ ) \rightarrow (010) \rightarrow (001)$
$(\ 1\ 0\ 1\ ) \rightarrow (110) \rightarrow (101)$
$(\ 1\ 1\ 0\ ) \rightarrow (011) \rightarrow (011)$
$(\ 1\ 1\ 1\ ) \rightarrow (111) \rightarrow (111)$

Gráficas obtenidas tras la implantación en el Code Composer

ver Figuras 4.8, 4.9, 4.10 y 4.11.

4.5. Comentarios y referencias

En este capitulo se implantaron los diferentes algoritmos descritos en el capitulo 5 y se ocupo toda la teoría de los capítulos anteriores.

En el Code Composer, las herramientas virtuales en tiempo real son las gráficas tiempo-frecuencia, diagramas de ojo, transformada de Fourier magnitud y fase.

Para más detalles de aplicaciones sobre el DSP, consultar los manuales que vienen en la ayuda del Code Composer.

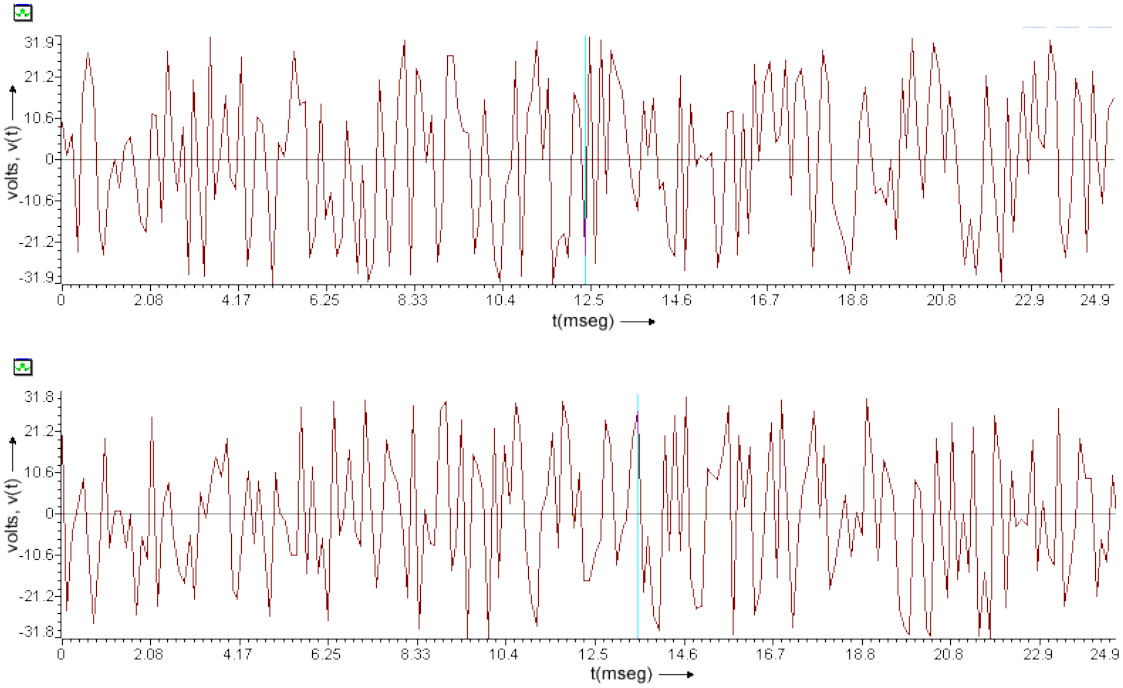

Figura 4.8: Multiplicación de los coeficientes Hamming en tiempo real.

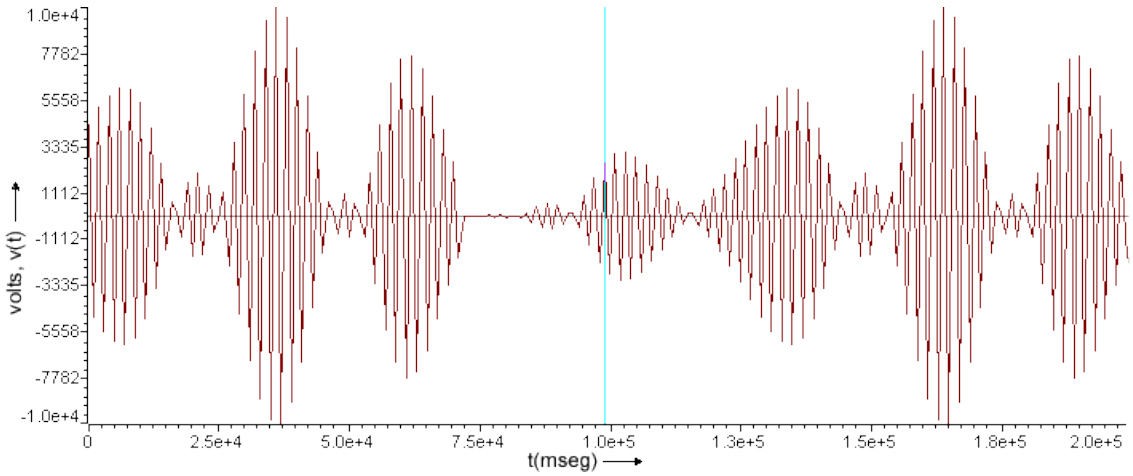

Figura 4.9: Ruido de alta frecuencia.

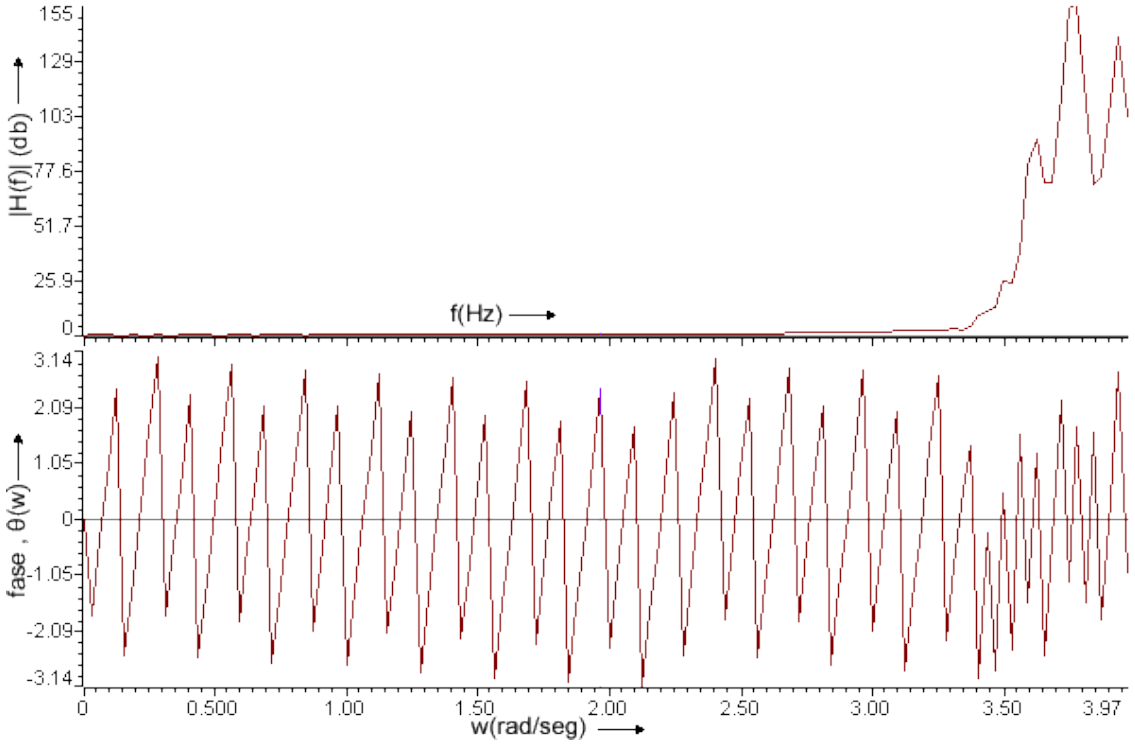

Figura 4.10: Espectro y fase del ruido de alta frecuencia .

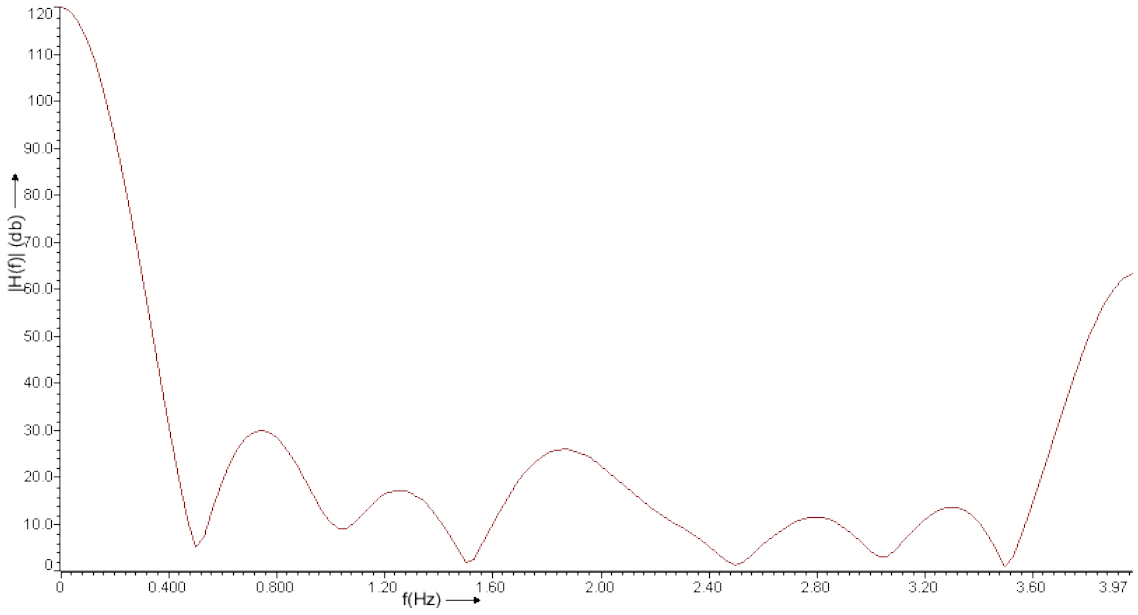

Figura 4.11: Espectro de Fourier en tiempo real de la señal de voz.

Para más detalles de aplicaciones sobre el DSP, consultar los manuales que vienen en la ayuda del Code Composer.

Capítulo 5

Conclusiones y trabajos futuros

Conclusiones

Este trabajo implanta a grandes rasgos algoritmos matemáticos para la caracterización de parámetros de la voz en tiempo real. Aunque a medida que pasa el tiempo surgen nuevas teorías con algoritmos que corrigen las desventajas de los anteriores, esto lleva al surgimiento de nueva tecnología más apta para las comodidades del ser humano.

La voz del ser humano es una señal difícil de caracterizar porque es variante, no lineal y no homogénea, pero, gracias al surgimiento de nuevas algoritmos es posible caracterizarla y formar patrones que sirvan como parámetros de la voz.

Se implanta la transformada rápida de Fourier en lugar de la transformada discreta de Fourier porque su costo computacional es menor.

Para la implantación del filtro FIR se tomaron los coeficientes hechos en la simulación; el DSP permite manejar de 2 a 600 coeficientes FIR.

La interfaz gráfica del Code Composer es flexible para la programación de algoritmos matemáticos, estos pueden ser escritos en lenguaje ANSI C y ensamblador.

Gracias a algunas técnicas de optimización descritas en [4] y a la medición de ciclos por medio de las herramientas virtuales del Code Composer, se permitió ejecutar algoritmos en menor tiempo aunque aún faltan de aplicar más técnicas.

MatLab es una excelente herramienta para simular algoritmos y comprobar modelos. Por medio de MatLab se obtuvieron los coeficientes necesarios del filtro FIR, coeficientes del ruido y los de la ventana Hamming.

Se concluye que es útil utilizar algún tipo de ventana, ya que transforma la serie de muestras infinitas a una serie finita.

De los resultados acerca del tamaño de la ventana, se concluye que si la ventana es muy grande puede existir pérdida de información. Y si la ventana es pequeña puede faltar información, esto se comprueba con una de la propiedades de Fourier.

Con respecto a la transformada de Fourier, se concluye que es la herramienta principal para conocer las componentes de frecuencia de la señal, tanto de magnitud como de fase.

Con respecto al tipo de ventana se llegó a la conclusión que es eficiente utilizar una ventana Hamming que una rectangular porque las muestras están más ordenadas, esto se debe a que la respuesta de frecuencia de la ventana Hamming tiene la forma de una distribución Gaussiana y amplifica los valores a un determinado valor.

Con respecto al filtro FIR, se concluye que este tipo de filtro se puede incrementar el número de coeficientes, al incrementarlos, mejora la respuesta del filtro de la banda de rechazo.

Al comparar la señal sin ruido con la de ruido se da cuenta que funciona el filtro porque atenúa la señal y además la defasa.

El code composer ya tiene una librería con algunas funciones, pero no está completa, si se desea implantar otro algoritmo es útil manejar la transformada z. Se utiliza la transformada z porque la señal es discreta.

Se normalizan los resultados obtenidos de los algoritmos para adecuar los valores a la escala.

Los cruces por ceros, los máximos, la energía y la media aritmética son unos de los algoritmos que sirven para comprobar la calidad de la señal o distinguir ciertos tonos de voz.

Trabajos futuros

Dentro de los trabajos futuros se pueden mencionar los siguientes

- Un banco de filtros para mejorar la respuesta del sistema.
- Algoritmos de celosia adaptativa y predicción lineal.
- Mejorar la configuración de puertos en el DSP con el fin de aumentar la velocidad de recepción de datos.

Bibliografía

[1] Shawn Dirksen. An audio example using dsp/bios. *Texas Instruments*, 31(6):851–862, 1998.

[2] GoldWave Inc. *GoldWave Help*. Canada, NF, 2001.

[3] Texas Instruments Inc. *TMS320C6000 DSP/BIOS Application Pragramming Interface(API) Reference Guide*. Dallas, TX, 1998.

[4] Texas Instruments Inc. *TMS320C6000 Optimizing compiler User's Guide*. Dallas, TX, 2001.

[5] B. P. Lathi. *Introducción a la teoria y sistemas de comunicación*. Balderas 95, México, D.F., 2004.

[6] Juan C. Campo Rodríguez Fco. Javier Ferrero Martín Gustavo J.Grillo Ortega Miguel A. Pérez García, Juan C. Álvarez Antón. *Instrumentación electrónica*. Madrid España, primera edition, 2004.

[7] A. V. Oppenheim and R. W. Schafer. *MATLAB Mathworks*. Junio 2002.

[8] Jesús Bodadilla Sancho Pedro Gómez Vilda, Jesús Bernal Bermúdez. *Reconocimiento de voz y Fonética Acústica*. Madrid, España, 2000.

[9] Dimitris G. Proakis, John G. Manolakis. *Tratamiento digital de señales-Principios, algoritmos, y aplicaciones*. Prentice Hall, Madrid, tercera edition, 1998.

[10] Inc Texas Instruments. *TMS320C6711, User's Guide*. Dallas, TX, 2001.

Glosario

Algoritmo. Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

CODEC. Es un dispositivo que codifica en una dirección de transmisión y decodifica en otra dirección de transmisión.

Control. Proceso mediante el cual un sistema es llevado a parámetros preestablecidos.

Escenario, es un concepto que se refiere a configuraciones predeterminadas que, con la realización de un único comando (la pulsación de un botón, el acceso a una habitación o según la hora), cambia la configuración de uno o varios sistemas simultáneamente. Por ejemplo, podemos programar nuestro sistema para que a las ocho de la mañana, en días laborales, el sistema automáticamente encienda la luz al cincuenta por ciento en el dormitorio, abra las persianas de toda la vivienda, etc. El número de escenarios es indefinido y debería adaptarse a cada situación específica.

Fidelidad. Calidad en la reproducción de un sonido.

Hardware. Se dice de cualquier componente físico relacionado con cierta tecnología.

Interfaz programable para la aplicación (API), se usa por programas de aplicación para interactuar con software de comunicaciones o para ajustarse a otros productos.

Interrupción, señal enviada por hadware o software para solicitar la atención del procesador. La señal le indica al procesador que suspenda su operación presente, guarde el estado de la tarea presenta, y ejecute un conjunto particular de instrucciones. Las interrupciones se comunican con el sistema operativo y establece un orden de prioridad de las tareas para ejecutarse.

MFLOPS. Million Floating Operations Persecond(millones de operaciones en coma flotante por segundo). Se refiere al número de operaciones en punto flotante (multiplicaciones, sumas, restas, etc.) que puede realizar en un segundo.

Modelo matemático. Es la representación por medio de ecuaciones de la dinámica de un sistema. Es el tipo de modelo más importantes para la ciencia y la tecnología.

Optimización. Procedimiento en el cual se determina la mejor (óptima) solución a un problema. Para que la optimización sea significativa debe existir una función objetivo y debe existir más de una solución factible (solución que no viola las restricciones impuestas al problema).

Ruido. Típicamente es una perturbación aleatoria. Señal indeseada presente en un sistema físico.

Sensor. Dispositivo que convierte un parámetro físico (como temperatura, presión, flujo, velocidad, posición) en una señal eléctrica. En algunos casos se le considera un sinónimo de transductor, pero un verdadero sensor contiene un sistema de acondicionamiento de la señal, de manera que es mucho más sencillo realizar una medición.

Sistema en tiempo real. Un sistema de tiempo real (STR) es un sistema informático en el que es significativo el tiempo en el que se producen las acciones. Las acciones deben realizarse dentro de un intervalo de tiempo determinado. Por ejemplo: un escenario habitual, el filtrado.

Sistema lineal. Se dice que un sistema es lineal sí cumple con los principios de homogeneidad y superposición.

Sistema no lineal. Se dice que un sistema es lineal sí no cumple con los principios de homogeneidad o superposición.

Software. Se dice de todos los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

Trama. En redes una trama es una unidad de envío de datos. Viene a ser sinónimo de paquete de datos o Paquete de red, aunque se aplica principalmente en los niveles OSI más bajos, especialmente en el Nivel de enlace de datos.

Apéndice A

Herramientas del procesador digital de señales

Un DSP(Digital Signal Processors, Procesador Digital de señales) es un procesador digital cuyo hardware y conjunto de instrucciones está optimizado para la implementación eficiente de aplicaciones de procesamiento numérico intensivo a alta velocidad. Este tipo de procesadores ejecutan, generalmente, los algoritmos típicos de procesamiento digital de señales(Digital Signal Processing) como el filtrado digital y el análisis espectral, entre otros. Este tipo de sistema se requiere cuando se procesa variables analógicas en tiempo real[6].

Las características de los dispositivos C62x / C67x, incluyen:

1. Un CPU avanzado VLIW (very long instruction word) con 8 unidades funcionales, que incluyen 2 multiplicadores y 6 ALU's (unidades lógico aritméticas).
   a) Ejecuta un máximo de 8 instrucciones por ciclo, 10 veces mas que los DSP's típicos.
   b) Permite rápido tiempo de desarrollo en diseños con código RISC altamente efectivos.
2. Empaquetado de instrucción.
   a) Obtiene el tamaño del código equivalente a las 8 instrucciones ejecutadas en serie o en paralelo.
   b) Reduce el tamaño del código, el consumo de energía y el *fetch* del programa.
3. Ejecución condicional de todas las instrucciones.

   a) Reduce los saltos costosos.
   b) Incrementa el paralelismo para mantener un alto desempeño.
4. Ejecuta el código programado, en unidades funcionales independientes.
5. Proporciona soporte eficiente de memoria para una variedad de aplicaciones de 8, 16 y 32 bits de datos.
6. Maneja operaciones aritméticas de 40 bits, adicionando precisión extra a vocoders y otras aplicaciones computacionalmente intensivas.
7. Proporciona soporte de normalización y saturación, en operaciones aritméticas claves.
8. Soporta operaciones comunes, halladas en aplicaciones de control y manipulación de datos, como: manipulación de campos, extracción de instrucción, activación, desactivación y conteo de bits.

Además, el C67x tiene las siguientes características:

- Un máximo de 1336 MIPS (millones de instrucciones por segundo), a 167 MHz.
- Un máximo de 1 G FLOPS (operaciones en punto flotante por segundo), a 167 MHz, para operaciones de precisión simple.
- Un máximo de 250 MFLOPS a 167 MHz, para operaciones de doble precisión.
- Un máximo de 688 MFLOPS a 167 MHz, para operaciones de multiplicación y acumulación.
- Soporte de hardware para operaciones de punto flotante, de simple y doble precisión (con formato IEEE).
- Multiplicación entera de 32 x 32 bits, con resultado de 32 o 64 bits.

Los dispositivos C62x / C67x tienen la siguiente variedad de opciones de memoria y periféricos:

- Amplia memoria RAM para ejecución rápida de algoritmos.
- Soporta interfases para memoria externa de 32 bits (SDRAM SBSRAM, SRAM y otras memorias asíncronas), para aumentar el rango de memoria externa y maximizar el desempeño del sistema.
- Acceso a la memoria y periféricos de los dispositivos C62x / C67x a través del puerto *host*.

- Controlador del multicanal DMA.
- Puerto(s) serie multicanal.
- Timer(s) de 32 bits.

A.1. Arquitectura de los dispositivos TMS320 C62x/C67x

Los dispositivos 'C6211, 'C6711, 'C6701, 'C6201 y 'C6202 operan a 150, 150, 167, 200 y 250 MHz respectivamente. Todos estos DSP's ejecutan un máximo de 8 instrucciones por ciclo. El DSP 'C6211 es de punto fijo mientras que el 'C6711 es de punto flotante.

Los procesadores 'C62x/C67x consisten de tres partes: el CPU, los periféricos y la memoria. Ocho unidades funcionales operan en paralelo (seis ALU's y dos multiplicadores), con dos conjuntos similares de cuatro unidades funcionales básicas. Las unidades se comunican usando un camino cruzado entre dos clasificaciones de registros, cada una de los cuales contiene 16 registros de 32 bits. La Figura A.4 muestra el diagrama de bloque de los dispositivos TMS320C62x/C67x.

A.2. Interrupciones

Los CPUs 'C62x/'C67x tienen 14 interrupciones. Estas son *reset*, la interrupción no mascarable (NMI) e interrupciones de la 4 a la 15. Estas interrupciones corresponden a las señales `RESET`, NMI e INT4-INT15 respectivamente, sobre los limites del CPU. En los mismos dispositivos 'C62x/'C67x, estas señales pueden estar ligadas directamente a los pines del dispositivo, conectando periféricos al chip, o pueden ser desactivadas permanentemente, cuando están ligadas e inactivas en el chip. Generalmente, `RESET` y NMI son conectadas directamente a los pines del dispositivo. Las características del servicio de interrupción incluyen:

- El pin IACK del CPU es usado para confirmar la recepción de una petición de interrupción.
- Los pines INUM0 INUM3 indican el vector de interrupción que esta siendo utilizado.
- Los vectores de interrupción son reubicables.

- Los vectores de interrupción consisten de un paquete *fetch*. Con los paquetes se proporciona un rápido servicio.

A.3. Herramientas de hadware

Un diagrama a bloques para cualquier interfaz a nivel Hadware se presenta en la Figura **??**.

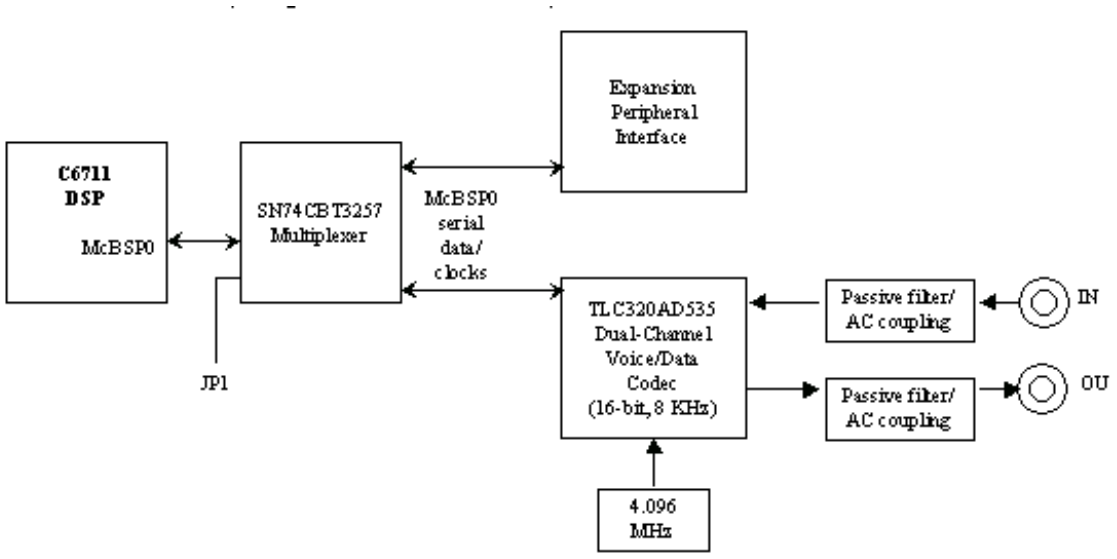

Figura A.1: Diagrama a bloques de una interfaz analógica.

A.3.1. El AD535

O mejor conocido como circuito de interfaz analógica(AIC), se muestra en la Figura A.2.

La descripción funcional de cada bloque es:

Funcionamiento de codec. El dispositivo TLC320AD535, la parte del CODEC, se encarga de las funciones requeridas para los dos canales conversión digital-analógica, conversión analógica-digital, filtro pasabajas, control de ganancia a la entrada y salida, acopla el sobremuestreo interno con desimación interna e interpolación y los dos puertos serial de 16 bits del host

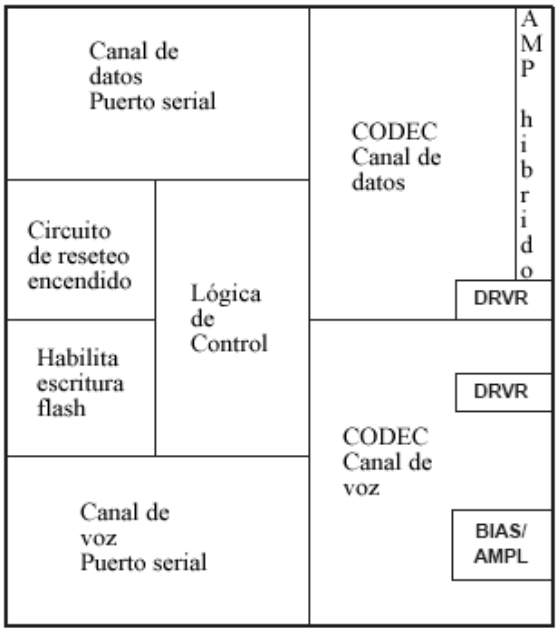

Figura A.2: Diagrama a bloques del convertidor

del procesador. Los dos puertos seriales operan independientemente y son capaces de operar a diferentes frecuencias de muestreo. La máxima frecuencia de muestreo para ambos canales es de 11.025 KHz.

Funcionamiento del circuito híbrido. El circuito híbrido del canal de datos incluye amplificadores integrados cuya ganancia y frecuencia del filtro es colocada por resistencias y capacitores externos. Lo que permite mayor flexibilidad para ajustar estándares internacionales y variantes de la tarjeta mientras lo demuestran algunas funciones. La ganancia del amplificador puede ser programada por medio del registro de control dos. El filtro del amplificador del canal de datos es seguido de un amplificador de ganancia, cuya retroalimentación es un altavoz de 8 Ohms: El control del altavoz puede ser programado para una ganancia de 0 dB o modo silencio del registro de control dos. La fuente de entrada para el control del altavoz puede ser la salida del amplificador DAC (Salida de datos PGA) o la señal de entrada de ADC proveniente del registro de control uno.

Canal analógico de voz. El circuito analógico del canal de voz incluye una interfaz para conectar un micrófono, cuya fuente maxima es de 5mA a 2.5V/1.5V, y preamplifica la señal del micrófono, puede ser seleccionada la ganancia del preamplificador de 0 db o 20db. También cuenta con una micro interfaz con amplificadores receptores y transmisores.

Circuito lógicos de control y otros circuitos. Las funciones lógicas incluyen la circuitería necesaria para implementar dos puertos seriales independientes y programar los registros de control para la comunicación secundaria de estos puertos seriales. Hay cinco registros de control que son programados durante la comunicación secundaria del canal de datos o el puerto serial del canal de voz. En estos registros de control se coloca la ganancia de los amplificadores, escogemos la entrada a los multiplexores, se selecciona las funciones de retroalimentación y leemos los flancos de escrituras del convertidor ADC. Este dispositivo también incluye un circuito de reseteo y de encendido. Además, cuenta con un circuito externo lógico para que sirve como memoria.La escritura de la memoria flash habilita el encendido.

Para transmitir la señal de datos del ADC o DAC, se usa una comunicación serial primaria. Una comunicación serial secundaria es leída o escrita como palabra a el registro de control

El propósito de una comunicación primaria y secundaria es para permitir la conversión y control de datos para ser transmitidas por el mismo puerto serial. Una transferencia primaria siempre esta dedicada a la conversion de datos. Una transferencia secundaria es usada para leer los registros de control. Los registros de control 1 y 2 únicamente puede ser escrita o leída de un canal de datos del puerto serial. Los registros de control 3 y 6 únicamente pueden ser escrito o leídos de un canal de voz del puerto serial.

Comunicación primaria serial. La comunicación primaria serial transmite y recibe la conversion de la señal de datos. El DAC tiene el formato de palabra que es de 15 bits y el último bit de la comunicación serial primaria el 16 bit es usada para el control de una comunicación serial. Para todas las comunicaciones, el bit mas significativo es el primer bit de transferencia. Para el modo de ADC es de longitud de una palabra, 16 bit, D15 es el más significativo bit, y D0 es el bit menos significativo.

A.3.2. McBSP

El puerto serial multicanal con buffer (McBSP) se basa en las interfaces estándar del puerto serie, encontrada en los dispositivos con plataformas TMS320C2000 y C5000. Resumiendo, el puerto puede almacenar muestras seriales en un buffer de memoria automáticamente, con la ayuda del controlador EDMA. Este también tiene capacidad multicanal, compatible con los estándares de conexión de redes T1, E1, SCSA y MVIP. Las características del McBSP son:

- Comunicación Full-Duplex.

- Registros de datos de doble buffer para flujo continuo de datos.
- Tramado independiente y temporización para dispositivos y transmisión.
- Interfaz directa a códecs estándar, chips de interfaz analógica (AIC) y otros dispositivos A/D y D/A conectados serialmente.

Tiene las siguientes capacidades:

- Interfaz directa a:
  1. Tramas T1/E1.
  2. Dispositivos conforme a ST-BUS.
  3. Dispositivos conforme a IOM-2.
  4. Dispositivos conforme a AC97.
  5. Dispositivos conforme a IIS.
  6. Dispositivos conforme a SPI.
- Transmisión y recepción multicanal de 128 canales.
- Un selector de ancho del tamaño del dato, que incluye 8, 12, 16, 20, 24 y 32 bits.
- Ley μ y ley A de comprensión/expansión
- Transferencia inicial de 8 bits con LSB o MSB.
- Polaridad programable para ambas tramas de sincronización y relojes de datos.
- Reloj interno altamente programable y generación de trama.

El McBSP consta de una vía de datos y una de control, como se muestra en la Figura 6.2, que se conectan a dispositivos externos. Los datos se comunican a los dispositivos externos a través de terminales separadas para transmisión y recepción. El control de información es a través de otras cuatro terminales. El dispositivo se comunica con el McBSP por medio de registros de control de 32 bits por conducto del bus periférico.

Los datos se comunican con el McBSP a través de la terminal datos de transmisión (DX) y de la terminal datos de recepción (DR). El control de la información de temporización y de la trama de sincronización es a través de CLKX, CLKR, FXR y FSR. Los dispositivos periféricos se comunican con el McBSP a través de los registros de control accesible del bus interno de periféricos. El CPU o el controlador del EDMA leen los datos recibidos del registro DRR y escriben los datos para transmitirlos, en el registro DXR. Los datos escritos en DXR se desplazan a DX a través del registro XSR. De igual forma, los datos

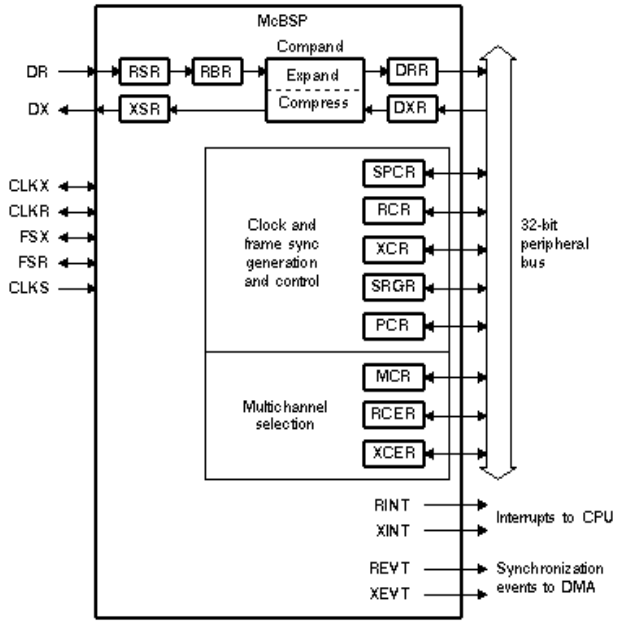

Figura A.3: Diagrama a bloques del McBSP.

recibidos en la terminal DR se desplazan por medio del registro RSR y se copian en el registro de buffer de recepción (RBR). RBR se copia en DRR, que puede ser leído por el CPU o el controlador del EDMAÈsto permite el movimiento interno de los datos y la comunicación externa simultáneamente. Los registros restantes tienen acceso a la configuración del CPU del mecanismo de control del McBSP. Estos registros se listan en la Tabla 6.1. El bloque de control se compone de un generador de reloj interno, generador de las señales de la trama de sincronización, control de ambos, y selección del multicanal. Este bloque de control envía la notificación de eventos importantes al CPU y al controlador EDMA,

A.3.3. EDMA

El controlador de acceso directo mejorado a la memoria (Enhanced Direct Memory Access), EDMA, maneja todas las transferencias entre la memoria caché del nivel L2 y los periféricos, como lo muestra la Figura 6.3. Estas transferencias de datos incluyen el servicio de caché, acceso a la memoria no caché, transferencia de datos programados por el usuario y acceso al host.

El EDMA del DSP C6711 incluye 16 canales, con prioridad programable, y la capacidad de enlazar la transferencia de datos. EDMA permite el movimiento

de los datos de o hacia los espacios direccionables de la memoria, incluyendo a la memoria interna (SRAM L2), a los periféricos, y a la memoria externa.

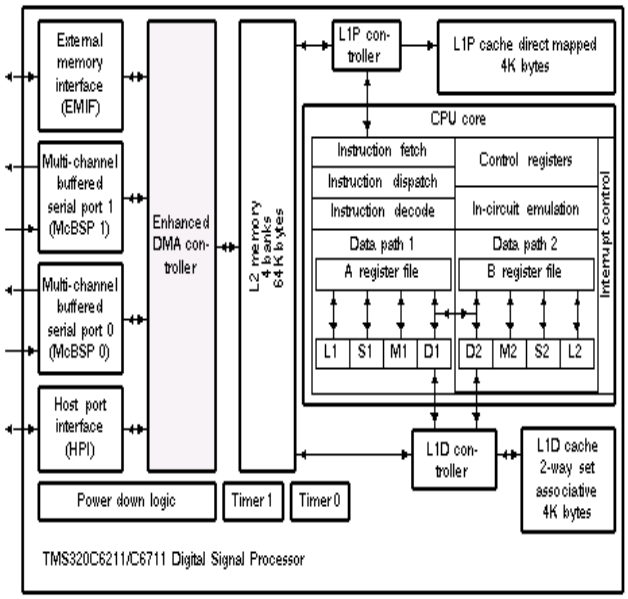

Figura A.4: Diagrama a bloques del EDMA.

El EDMA consta de dos componentes primarios:

1. El controlador de transferencia (EDMATC), manipula toda la transferencia de datos entre el controlador de la memoria caché y los periféricos como se muestra en la Figura 6.3. En la transferencia de datos se involucra la transferencia al control del canal del EDMA, el acceso a o hacia la EMIF, acceso a la memoria no caché y acceso al periférico maestro.
2. El controlador del canal (EDMACC) es la parte programable del EDMA que soporta un flexible y poderoso conjunto de transferencias, incluyendo las transferencias 1D y 2D; transferencias flexibles de disparo incluyendo las de disparo de eventos, disparo de cadena y disparo del CPU; modos de direccionamiento flexibles que soportan los buffers ping-pong, para almacenamiento circular en buffers, extracción de trama y ordenación.

A.4. Herramientas de software

El kit para el procesador TMS320c6711, ofrece un programa para la simulación de algoritmos matemáticos, esta interfaz de programación es conocida como

Code Composer Studio, trae varias herramientas virtuales, ver Figura A.5.

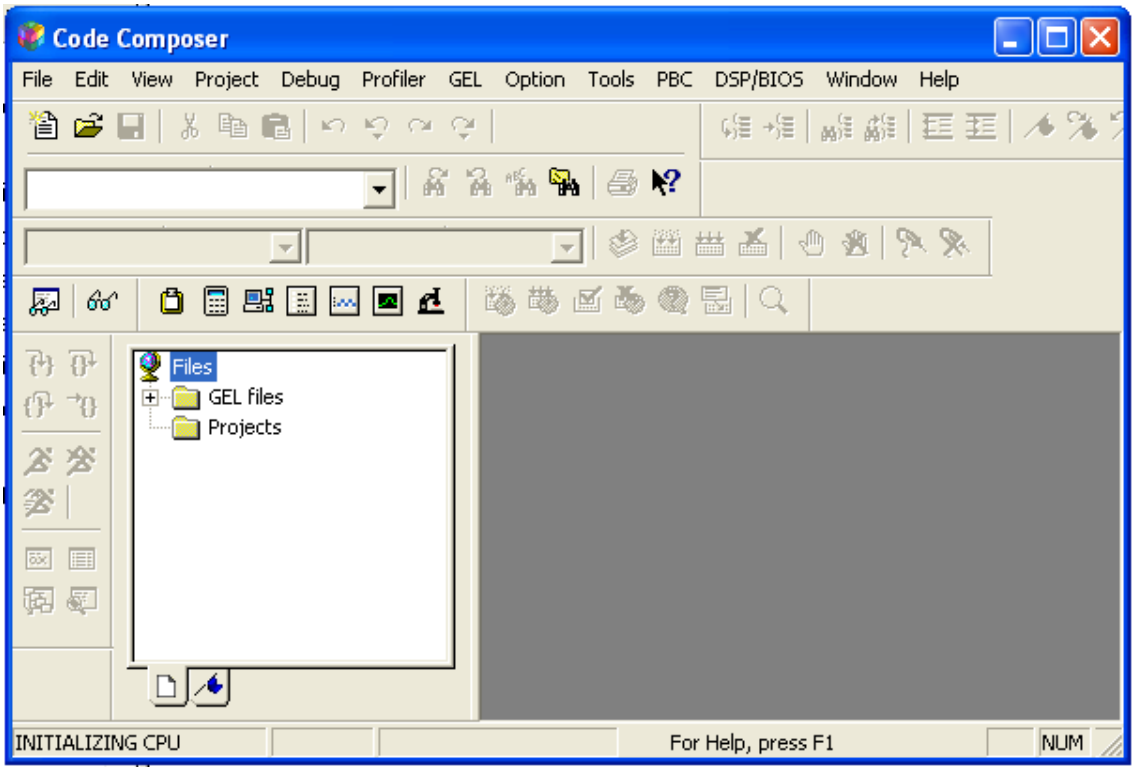

Figura A.5: Pantalla de inicio del Code Composer.

El lenguaje de programación para Code Composer Studio es ensamblador y lenguaje ANSI C. El lenguaje que más se ocupa en esta tesis es Lenguaje C++.

Para la implantación de un programa en tiempo real, debe tomarse en cuenta las herramientas con las que cuenta el DSP [3], monitoreo de variables y la optimización del programa. El DSP ofrece seis herramientas de configuración:

- Horarios.
- Instrumentos virtuales.
- Sincronización.
- Entradas - Salidas.
- Librerías de Soporte para el Chip(CSL).
- Configuración global del sistema.

A.4.1. Entorno de desarrollo Integrado de Code Composer Studio (IDE)

El IDE del Code Composer Studio esta diseñado para permitir editar, construir y depurar el programa del DSP[10].

A.4.1.1. Características del editor de código de programa

El Code Composer Studio permite editar código en C y en ensamblador. Puede verse el código fuente C con las instrucciones correspondientes en ensamblador, mostrando las sentencias de C después (utilizando de menú View > Mixed Source/ASM)

El editor integrado proporciona soporte a las siguientes actividades:

- Resalte palabras clave, comentarios y cadenas en color.
- Marcar bloques de C en paréntesis y corchetes, encontrando el par o próximo paréntesis o corchete.
- Niveles de sangrado.
- Búsqueda y reemplazo en uno o más archivos.
- Deshaciendo y rehaciendo múltiples acciones.
- Obtención de ayuda sensible al contexto.

A.4.1.2. Características de construcción de aplicaciones

Con el Code Composer Studio se puede crear un proyecto de trabajo, que es usado para construir la aplicación. Los archivos en el proyecto pueden ser archivos del código fuente en C, archivos en ensamblador, archivos objeto, librerías, archivos de comando del ligador y archivos de declaración (*include*) [3].

A.4.1.3. Características de depuración de aplicaciones

El Code Composer Studio provee soporte para las siguientes actividades de depuración:

- Establecer puntos de ruptura.
- Actualizar automáticamente las ventanas en los puntos de ruptura.

- Visualizar el valor de las variables.
- Ver y editar registros y memoria.
- Ver el *stack* de las llamadas a funciones.
- Usar herramientas punto de prueba, para flujo de datos de y a la tarjeta.
- Graficar las señales de la tarjeta.
- Generar estadísticas de ejecución.
- Ver instrucciones desensambladas e instrucciones en C ejecutándose sobre la tarjeta.
- Proporciona un lenguaje GEL. Este lenguaje permite añadir funciones al menú para optimizar las tareas comunes.

A.4.2. DSP/ BIOS plugs-ins

Los plug-ins de Code Composer Studio proporcionan con el DSP/BIOS, soporte para el análisis en tiempo real. Se pueden usar para visualmente: probar, señalar y monitorizar una aplicación DSP con el mínimo impacto en el performance. Las APIs DSP/BIOS proporcionan las siguientes capacidades en tiempo real:

- Mensajes del programa (program tracing): Despliega los eventos escritos en re-gistros designados y refleja dinámicamente el control de flujo durante la ejecución del programa.
- Monitoreo del performance (*performance monitoring*): Rastrea las estadísticas que reflejan el uso de los recursos, como la carga del procesador y los tiempos de procesos.
- Archivos de flujo (flow file): Liga archivos de datos en la PC, a objetos de Entrada/ Salida en el programa del DSP.

A.4.2.1. Configuración del DSP/BIOS

Se pueden crear archivos de configuración utilizando el entorno del Code Composer Studio. Con ellos se definen objetos que son usados para las APIs del DSP/BIOSĖstos archivos también simplifican el mapeo de memoria y el mapeo de los vectores, en las rutinas de atención de interrupción.

Cuando se abre un archivo de configuración del DSP/BIOS, el Code Composer Studio muestra un editor visual, que permite crear y establecer propiedades para objetos de tiempo real. Estos objetos son usados en las llamadas a las APIŠs DSP/BIOS. También incluyen interrupciones por software, tuberías de I/O, mensajes de eventos (logs), etc. Figura A.6 muestra el editor visual DSP/BIOS.

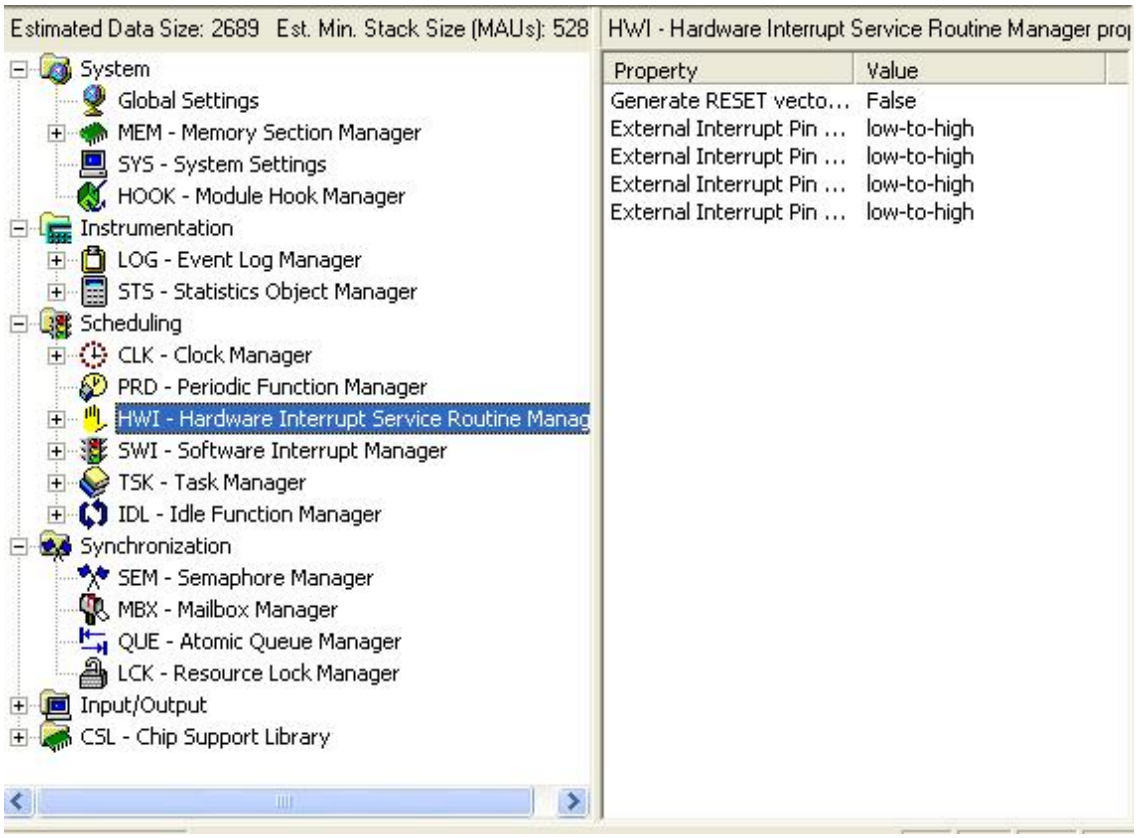

Figura A.6: Ventana de configuración DSP/BIOS.

A.4.2.2. Módulos del DSP/BIOS

Las APIs del DSP/BIOS están divididas en los siguientes módulos:

- ATM: Funciones atómicas que pueden ser usadas para manipular datos compartidos.
- C62: Este módulo proporciona funciones específicas del DSP, para manejo interrupciones.
- CLK: Este módulo controla el timer interno del DSP y proporciona un reloj lógico de 32 bits en tiempo real con alta resolución.
- DEV: Este módulo permite crear y usar sus propios controladores de dispositivos.
- HST: El módulo host, manejas este tipo de objetos de canal que permiten a una aplicación transmitir flujo de datos entre al DSP y un host. Los canales host son configurados estáticamente para entrada o salida.
- HWI: El módulo de interrupciones por hardware proporciona soporte para rutinas que atienden las interrupciones.
- IDL: Este módulo maneja funciones *idle*, que corren cuando no existe ninguna función de mayor prioridad ejecutando.
- LCK: El módulo *lock* maneja recursos globales compartidos y es usado para controlar el acceso a estos recursos, entre varias tareas que los disputen.
- LOG: Este módulo maneja objetos tipo LOG, que capturan eventos en tiempo real mientras el programa objeto se ejecuta. Se puede usar logs de sistema o se puede definir logs propios. Con el Code Composer Studio se puede visualizar estos mensajes logs mientras se ejecuta el programa.
- MBX: El módulo mailbox maneja objetos que pasan mensajes de una tarea a otra.
- MEM: El módulo de memoria permite especificar los segmentos de memoria requeridos.
- PIP: Este módulo maneja tuberías de datos (pipe), que son usadas como flujos de buffers para entrada y salida de datos. Estas tuberías de datos proporcionan una estructura de datos consistente, para manejar entradas/salidas entre el DSP y otros dispositivos periféricos, en tiempo real.
- PRD: El módulo de manejo de funciones periódicas administra objetos de este tipo. Permite activar ejecuciones cíclicas de una función. La velocidad de ejecución para estos objetos puede ser controlada, por la frecuencia de reloj mantenida por el módulo CLK o por llamadas regulares a la función PRD tick.

- QUE: Este módulo maneja estructuras de colas de datos.
- RTDX: Permite el intercambio de datos en tiempo real entre la PC y el DSP, y además analizar y desplegar los datos en la PC usando una automatización OLE cliente (esta puede estar programada en Visual C++, Visual Basic, Excel, Matlab, LabView, etc).
- SEM: El módulo de semáforos permite sincronizar tareas y realizar exclusión mutua.
- SIO: El módulo stream maneja objetos que proveen eficiencia en tiempo real de dispositivos de I/O.
- STS: Módulo de estadísticas, administra acumulación de estadísticas clave en tiempo real, mientras el programa se ejecuta.
- SWI: Este módulo administra las interrupciones por software. Estas interrupciones son procesos que tiene menor prioridad que las interrupciones por hardware y mayor prioridad que el módulo de tareas. Cuando un función anuncia a un objeto SWI, con una llamada de API, el módulo SWI programa para ejecución la función correspondiente.
- SYS: Módulo de dispositivos de sistema proporcionan funciones de propósito general.
- TRC: El módulo *trace* envía mensajes a la ventana de depuración en tiempo real.
- TSK: Módulo de tareas (las tareas procesos con prioridad más que las interrupciones por software.)

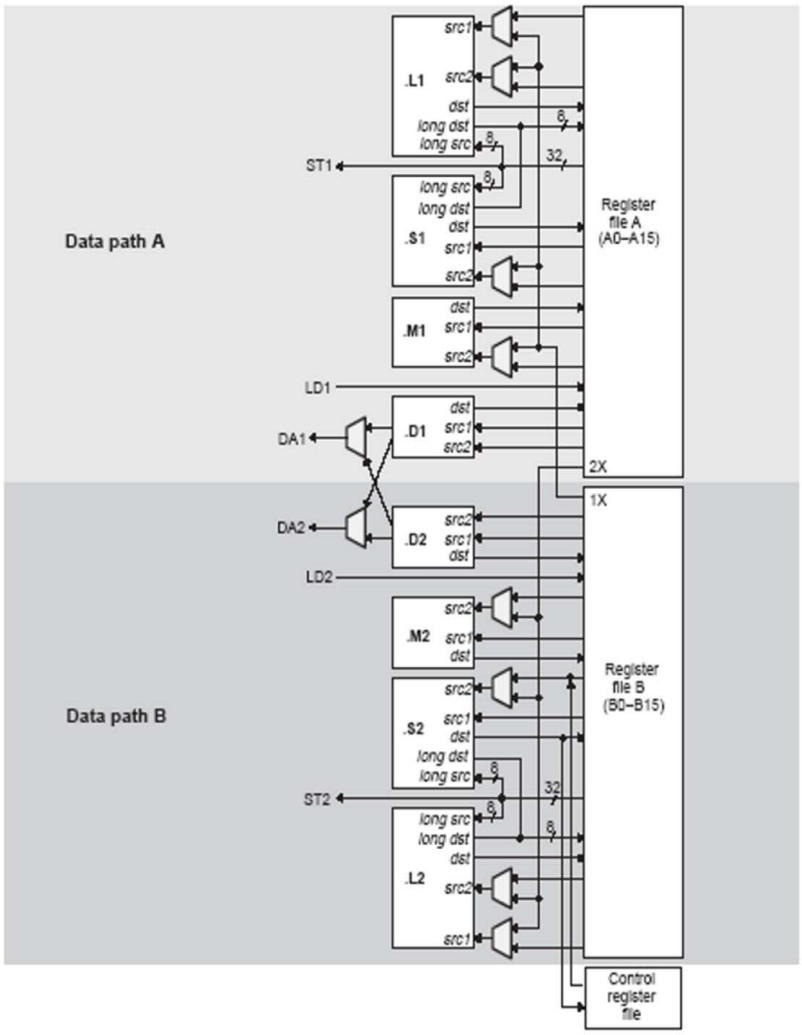

Figura A.7: Camino de datos del TMS320C67x.

Apéndice B

Wave

GoldWave es un editor de audio digital profesional. Contiene varias características:

Características generales

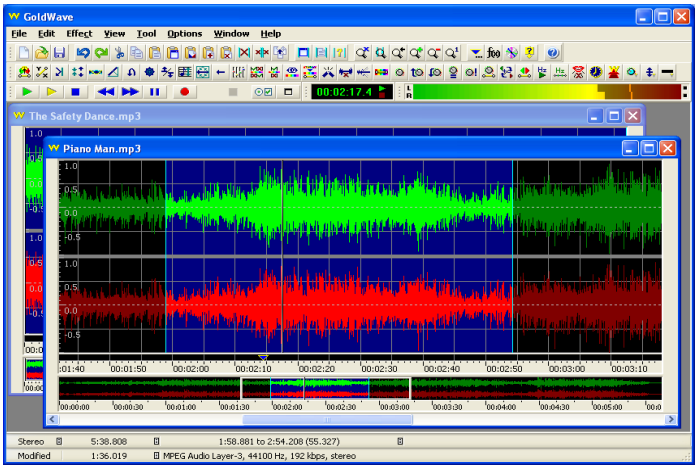

Figura B.1: Interfaz de trabajo de GoldWave.

- Reproducir, editar, mezclar, y de analizar de audio
- Record de casetes de audio, discos de vinilo, la radio, etc; a través de su ordenador en línea.

- Record de dictado a través de un micrófono o de dictado jugar de nuevo en una velocidad inferior para la transcripción (con pedal de control).
- Grabar y editar audio para sistemas de telefonía o el podcasting
- Aplicar efectos especiales, como desvanecimiento, ecualizador, Doppler, mecanizar, eco, invertir, flanger, y más.
- Restaurar y remasterizar digitalmente viejas grabaciones con la reducción de ruido y el pop / clic filtros.
- Haga copias digitales perfectas de pistas de CD de audio usando el CD Reader herramienta y guardarlas en el wav, del wma, mp3 o ogg.
- Convertir archivos a / desde diferentes formatos, como los del wav, del wma, de mp3, ogg, el AIFF, au, e incluso del vox binario de datos.

GoldWave es el más fiable y completo editor de audio disponibles en su gama de precios. Incluye todos los comandos de edición de audio común y los efectos, más potente incorporado en herramientas como un procesador por lotes / convertidor, una fusión de archivos, un lector de CD, y los filtros de audio de la restauración son caros añadidos en otros programas. Completa, bien diseñados, y fácil y divertido de usar, GoldWave ofrece el mejor valor en el software de edición de audio. Con más de 14 años de desarrollo y de uso generalizado, que tiene una excelente historia y inigualable. GoldWave pantalla principal, ver Figura

Control de ventanas

La ventana Control incluye controles para jugar, avance rápido, rebobinado, pausa y parada de lectura. Hay controles separados para iniciar, detener y pausar la grabación. Los tres faders ajustar el volumen de reproducción, el equilibrio, y la velocidad. Cuatro visuales en tiempo real se muestran. Más de una docena de diferentes visuales son incluidos. A visuales interfaz plug-in permite a otros desarrolladores crear fácilmente nuevas visuales para GoldWave, Figura B.2.

B.1. Control de propiedades

La ventana Propiedades de Control configura muchas características de la ventana Control. Puede seleccionar un dispositivo de audio, cambiar la reproducción bucles y regiones, fijar una fecha y hora para el registro, permitir la entrada de

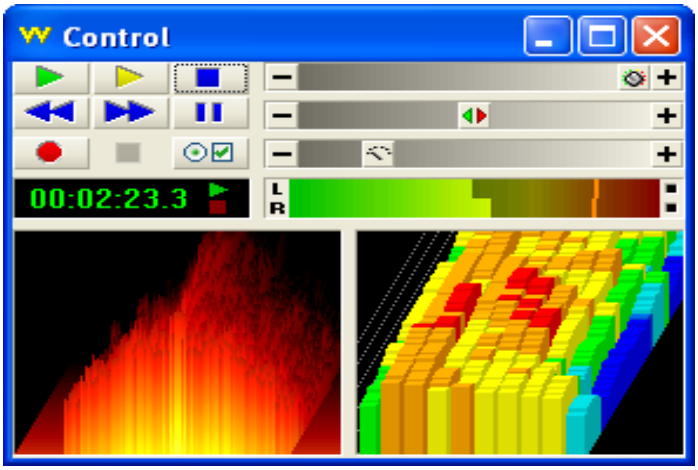

Figura B.2: Modos gráficos de GoldWave.

vigilancia, configurar los visuales, o establecer dispositivos de reproducción y grabación, Figura B.3.

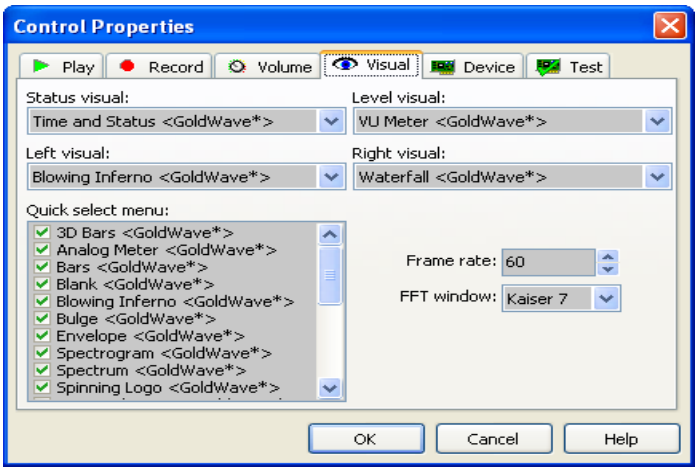

Figura B.3: Control de propiedades.

Ecualizador

El ecualizador efecto es similar al ecualizador faders encontrarse en sistemas estéreo. Utilícelo para modificar agudos / bajo nivel. La mayoría de los efectos incluyen previa de los botones (1) y preestablecidos controles (3). Los niveles

de volumen se da en dB (2), sino que también se muestra en el por ciento de los usuarios, Figura B.4.

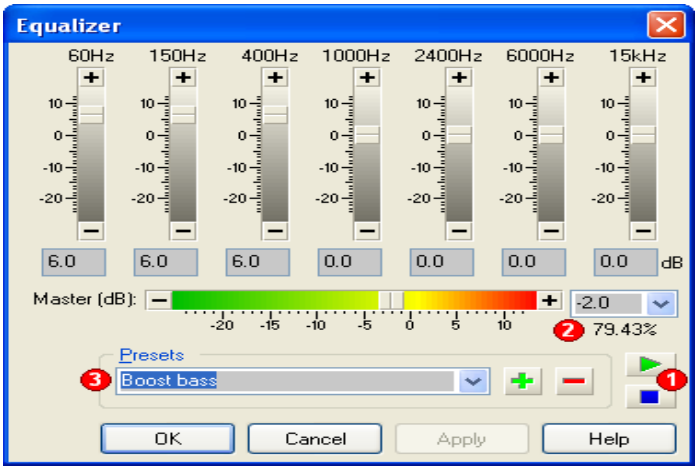

Figura B.4: Ecualizador.

CD reader

El CD Reader herramienta le permite guardar un conjunto de pistas de audio a los archivos separados en mp3, ogg, el del wma, u otros formatos. Pista de títulos y de la etiqueta ID3 de información que puede obtenerse en el www.freedb.org base de datos y se almacenan en los archivos automáticamente, Figura B.5.

Efecto plug-ins

El efecto interfaz plug-in permite a otros desarrolladores para añadir efectos a GoldWave. Un plug-in de DirectX está incluida en la envoltura, Figura B.6.

Evaluador de la expresión

El Evaluador de la expresión permite sonido que se genera desde casi cualquier ecuación. Por ejemplo, para generar una simple onda sinusoidal, las siguientes se pueden introducir:

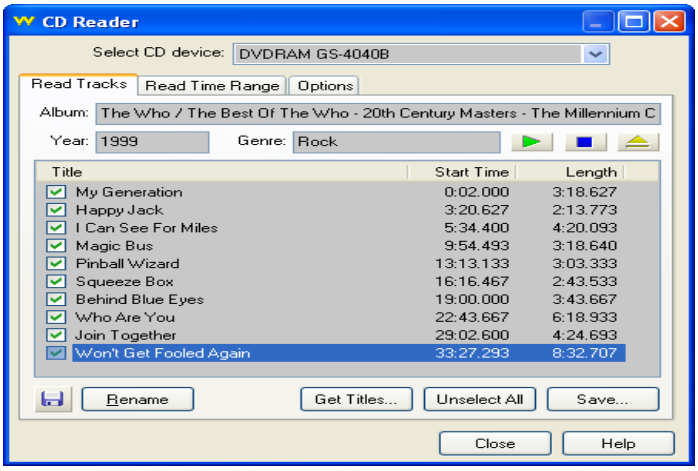

Figura B.5: Lector de disco.

Figura B.6: Plug-ins.

$$Pecado(2 * pi * *yftoneladas)$$

General expresiones de condición, triángulo, cuadrado y olas ya están previstos, además de las expresiones para marcar los tonos, efectos y ruidos, Figura B.7.

Reducción de ruido

Use el filtro de reducción de ruido para eliminar ruidos continuos, como un silbido o un zumbido de un sonido. Reducción de la dotación se pueden crear manualmente o automáticamente calculará desde cualquier punto del sonido o del sujetapapeles de audio, Figura B.8.

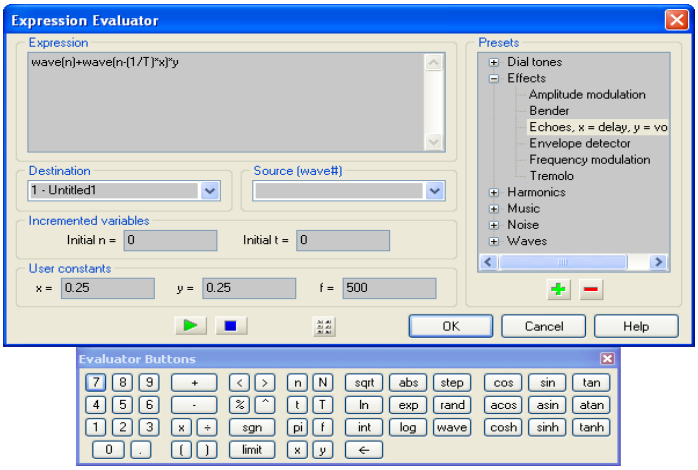

Figura B.7: Valuador de expresión.

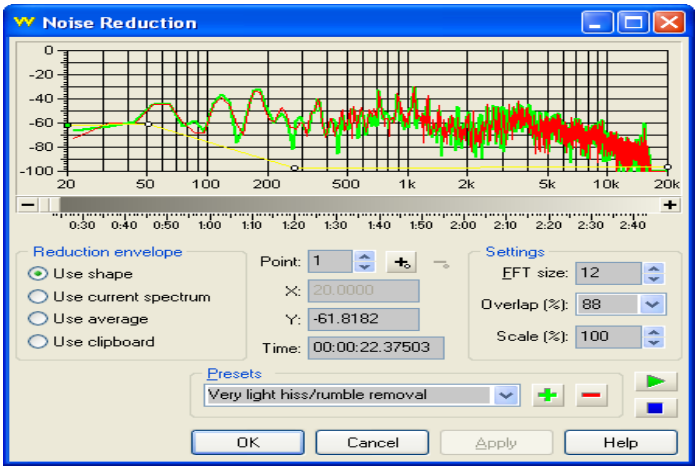

Figura B.8: Filtro.

Apéndice C

Filtros

Un filtro es un sistema que dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Los filtros digitales tienen como entrada una señal analógica o digital y a su salida tienen otra señal analógica o digital, pudiendo haber cambiado en amplitud, frecuencia o fase dependiendo de las características del filtro. El filtrado digital es parte del procesamiento digital de señales. Se le da la denominación de digital más por su funcionamiento interno que por su dependencia del tipo de señal a filtrar, así un filtro digital puede ser un filtro que realiza el procesado de señales digitales u otro que lo haga de señales analógicas. El filtrado digital consiste en la realización interna de un procesado de datos de entrada.

C.1. Tipos de filtro

Hay varios tipos de filtros, así como distintas clasificaciones para éstos. De acuerdo con la parte del espectro que dejan pasar y que atenúan existen:

- Filtro pasa alto.
- Filtro pasa bajo.
- Filtro pasa banda.
- Filtro de banda eliminada.
- Filtro multibanda.
- Filtro pasa todo.
- Filtro resonador.
- Filtro oscilador.

- Filtro peine *(comb filter)*.
- Filtro ranura *(notch filter)*.

De acuerdo con su orden:

- Filtro de primer orden.
- Filtro de segundo orden.
- $\cdots$ etc.

De acuerdo con el tipo de respuesta ante entrada unitaria:

- Filtro FIR *(Finite Impulse Response)*.
- Filtro IIR *(Infinite Impulse Response)*.
- Filtro TIIR *(Truncated Infinite Impulse Response)*.

De acuerdo con la estructura con que se implementa:

- Filtro Laticce.
- Arreglo de filtros en cascada.
- Arreglo de filtros en paralelo.
- $\cdots$ etc.

Hay muchas formas de representar un filtro. Por ejemplo, en función de w *(frecuencia digital)*, en función de z y en función de n *(número de muestra)*. Todas son equivalentes, pero su utilización depende de la aplicación en particular. Como regla general se suele dejar el término $a_0 = 0$ (para tener filtros causales, es decir físicamente realizables). Si se expresa en función de z y en forma de fracción:

$$H(z) = \frac{\sum_{k=0}^{M} b_k \cdot z^{-k}}{\sum_{k=0}^{N} a_k \cdot z^{-k}}.$$

Y en dominio de n:

$$y(n) = \sum_{k=0}^{N} b_k \cdot x(n-k) - \sum_{k=0}^{m} a_k \cdot y(n-k).$$

Los coeficientes a y b son los que definen el filtro, por lo tanto el diseño consiste en calcularlos.

C.2. Filtros FIR

Los filtros FIR o de respuesta finita al impulso *(Finite Impulse Response)*, corres-ponden a un grupo de filtros digitales en el que, como su nombre indica, si la entrada es una señal impulso, la salida tendrá un número finito de términos no nulos.

Para obtener la salida sólo se basan en entradas actuales y anteriores. Su expresión en el dominio n es:

$$y(n) = \sum_{k=0}^{N-1} b_k \cdot x(n-k).$$

En la expresión anterior, N es el orden del filtro, que también coincide con el número de términos no nulos y con el número de coeficientes del filtro. Los coeficientes son b_k. La salida también puede expresarse como la *convolución* de la señal de entrada $x(n)$ con la respuesta impulsional $h(n)$:

$$y(n) = \sum_{k=0}^{N-1} h_k \cdot x_{n-k}.$$

Aplicando la transformada **Z** a la expresión anterior, resulta la función de transferencia:

$$H(z) = \frac{y(z)}{x(z)} = \sum_{k=0}^{N-1} h_k \, z^{-k} = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-(N-1)}.$$

La estructura básica de un FIR se muestra en la Figura C.1. En la figura los términos h son los coeficientes y los T son retardos. Pueden hacerse multitud de variaciones de esta estructura. Hacerlo como varios filtros en serie, en cascada, etc.

Estos filtros tienen todos los polos en el origen, por lo que son estables. Los ceros se presentan en pares de recíprocos si el filtro se diseña para tener fase lineal.

Hay tres método básicos para diseñar este tipo de filtros:

1. Método de las ventanas.
   - Ventana rectangular.
   - Ventana de Bartlett.

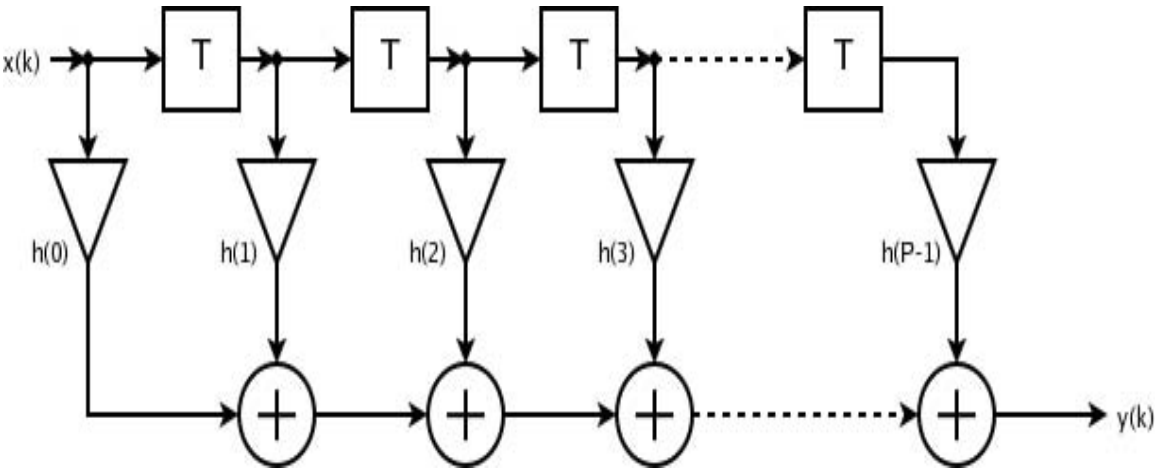

Figura C.1: Estructura básica de un FIR.

- Ventana de Hanning.
- Ventana de Hamming.
- Ventana de Blackman.
- Ventana de Kaiser.

2. Muestreo en frecuencia.
3. Rizado constante *(Aproximación de Tchebyshev y algoritmo de intercambio de Remez)*.

Los filtros FIR tienen la gran ventaja de que pueden diseñarse para ser de fase lineal, lo cual hace que presenten ciertas propiedades en la simetría de los coeficientes. Este tipo de filtros tiene especial interés en aplicaciones de audio. Además son siempre estables. Por contra, también tienen la desventaja de necesitar un orden mayor respecto a los filtros IIR para cumplir las mismas características. Esto se traduce en un mayor gasto computacional.

Diseño de filtros FIR

Para diseñar los filtros es necesario conocer algunas características. Los filtros ideales son no causales, y por tanto, físicamente irrealizables para aplicaciones de procesado de señal en tiempo real. La causalidad implica que la característica de respuesta en frecuencia $H(\omega)$ del filtro no puede ser cero excepto en un conjunto finito de puntos en el rango de frecuencias. Además, $H(\omega)$ no puede tener un corte infinitamente abrupto desde la banda de paso a la banda de rechazo, es

decir, $H(\omega)$ no puede caer desde uno hasta cero abruptamente. Aunque las características de respuesta en frecuencia que poseen los filtros ideales deben ser deseables, no son absolutamente necesarias en la mayoría de las aplicaciones prácticas.

Los siguientes métodos para el diseño de filtros se centran en la importante clase de filtros FIR de fase lineal.

Diseño de filtros FIR de fase lineal usando ventanas

Para empezar, es necesario especificar la respuesta en frecuencia deseada $H(\omega)$ y determinar la correspondiente respuesta impulsional $h(n)$, la cual se relaciona con $H(\omega)$ mediante la transformada discreta de Fourier

$$H(\omega) = \sum_{n=0}^{\infty} h(n)\, \mathrm{e}^{-j\omega n}, \tag{C.1}$$

donde

$$h(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega)\, \mathrm{e}^{j\omega n} d\omega.$$

Asi, dado $H(\omega)$, se puede determinar la respuesta impulsional $h(n)$ evaluando la integral en (C.1).

En general, la respuesta impulsional $h(n)$, obtenida de (C.1) es infinita en duración y debe ser truncada en algún punto, digamos que en $n = M - 1$, para producir un filtro FIR de longitud M. El truncamiento de $h(n)$ a una longitud M es equivalente a multiplicar $h(n)$ por una ventana rectangular definida como

$$\omega(n) = \begin{cases} 1, & n = 0, 1, \ldots, M-1. \\ 0, & \text{en otro caso.} \end{cases}$$

La función ventana tiene una respuesta en magnitud

$$|W(\omega)| = \frac{|\operatorname{sen}(\omega M/2)|}{\operatorname{sen}(\omega/2)}, \qquad \pi \leq \omega \leq \pi,$$

y una fase lineal a tramos

$$\Theta(\omega) = \begin{cases} -\omega(\frac{M-1}{2}), & \textit{cuando} \quad \operatorname{sen}(\omega M/2) \geq 0. \\ -\omega(\frac{M-1}{2}) + \pi, & \textit{cuando} \quad \operatorname{sen}(\omega M/2) < 0. \end{cases}$$

A medida que M crece, el ancho del lóbulo principal (medido al primer cero de $W(\omega)$) se hace más estrecho, sin embargo, los lóbulos laterales de $|W(\omega)|$ son relativamente altos y permanecen inalterados con incrementos en M. De hecho, incluso aunque el ancho de cada lóbulo lateral decrece con el incremento en M, la altura de cada lóbulo lateral se incrementa con un incremento en M.

La siguiente tabla lista varias funciones ventana que poseen características de respuesta en frecuencia deseable.

Nombre de la ventana	Secuencia en el dominio temporal, $h(n), 0 \leq n \leq M-1$
Bartlett (triangular)	$1 - \frac{2\left\lvert n - \frac{M-1}{2}\right\rvert}{M-1}$
Blackman	$0.42 - 0.5\cos\frac{2\pi n}{M-1} + 0.08\cos\frac{4\pi n}{M-1}$
Hamming	$0.54 - 0.46\cos\frac{2\pi n}{M-1}$
Hanning	$\frac{1}{2}(1 - \cos\frac{2\pi n}{M-1})$
Kaiser	$\frac{I_0\left[\alpha\sqrt{\left(\frac{M-1}{2}\right)^2 - \left(n-\frac{M-1}{2}\right)^2}\right]}{I_0\left[\alpha\left(\frac{M-1}{2}\right)\right]}, \quad I_o > 0$
Lanczos	$\left[\frac{\text{sen}\left[2\pi\left(n-\frac{M-1}{2}\right)\Big/\left(M-1\right)\right]}{2\pi\left(n-\frac{M-1}{2}\right)\Big/\left(\frac{M-1}{2}\right)}\right]^L, \qquad L > 0$

Todas estas funciones ventana tienen lóbulos laterales significativamente más bajos, comparados con la ventana rectangular. Sin embargo, para el mismo valor de M, el ancho del lóbulo principal es también más amplio para estas ventanas comparado con la ventana rectangular.

Diseño de filtros FIR de fase lineal mediante el método de muestreo en frecuencia

El método de muestreo en frecuencia para el diseño de filtros FIR, se específica la respuesta en frecuencia deseada $H(\omega)$ en un conjunto de frecuencia equies-

paciadas, a saber,

$$\omega_k = \frac{2\pi}{M}(k+\alpha), \tag{C.2}$$

$$k = 0, 1, \ldots, \frac{M-1}{2} \qquad M \quad \text{impar}, \tag{C.3}$$

$$k = 0, 1, \ldots, \frac{M}{2} - 1 \qquad M \quad \text{par}, \tag{C.4}$$

con

$$\alpha = 0 \ \text{ó} \ \frac{1}{2},$$

y se calcula la respuesta impulsional $h(n)$ del filtro FIR a partir de estas especificaciones en frecuencia equiespaciadas. Para reducir los lóbulos laterales, es deseable optimizar la especificación de frecuencia en la banda de transición del filtro. Esta optimización se puede llevar a cabo numéricamente en un procesador digital por medio de técnicas de programación lineal.

Aunque el método de muestreo en frecuencia proporciona otros medios para di-señar filtros FIR de fase lineal, su ventaja reside en la estructura eficiente de muestreo en frecuencia que se puede obtener cuando la mayoría de las muestras de frecuencia son cero.

Diseño de filtros óptimos FIR de fase lineal y rizado constante

El método de ventana y muestreo en frecuencia son técnicas relativamente simples para diseñar filtros FIR de fase lineal. Sin embargo, poseen algunas pequeñas desventajas que pueden hacerlos indeseables para algunas aplicaciones. El problema más grande es la carencia de control preciso de las frecuencias críticas, tales como ω_p y ω_s.

Este método de diseño se formúla como un problema de aproximación de Chebyshev. Se ve como un criterio de diseño óptimo en el sentido que el error de aproximación ponderado entre la respuesta en frecuencia deseada y la respuesta en frecuencia actual se distribuye equitativamente a lo largo de la banda de paso y equitativamente a lo largo de la banda de rechazo del filtro que minimiza el error máximo. Los diseños de filtro resultantes tienen rizados tanto en la banda de paso como en la banda de rechazo.

Para describir el procedimiento, se va a considerar el diseño de un filtro paso bajo con frecuencia de corte de la banda de paso ω_p y frecuencia de corte de la banda de rechazo ω_s. A partir de estas especificaciones generales, dadas en la

Figura C.2, en la banda de paso, la respuesta en frecuencia del filtro satisface la condición

$$1 - \delta_1 \leq H(\omega) \leq 1 + \delta_1, \qquad |\omega| \leq \omega_p.$$

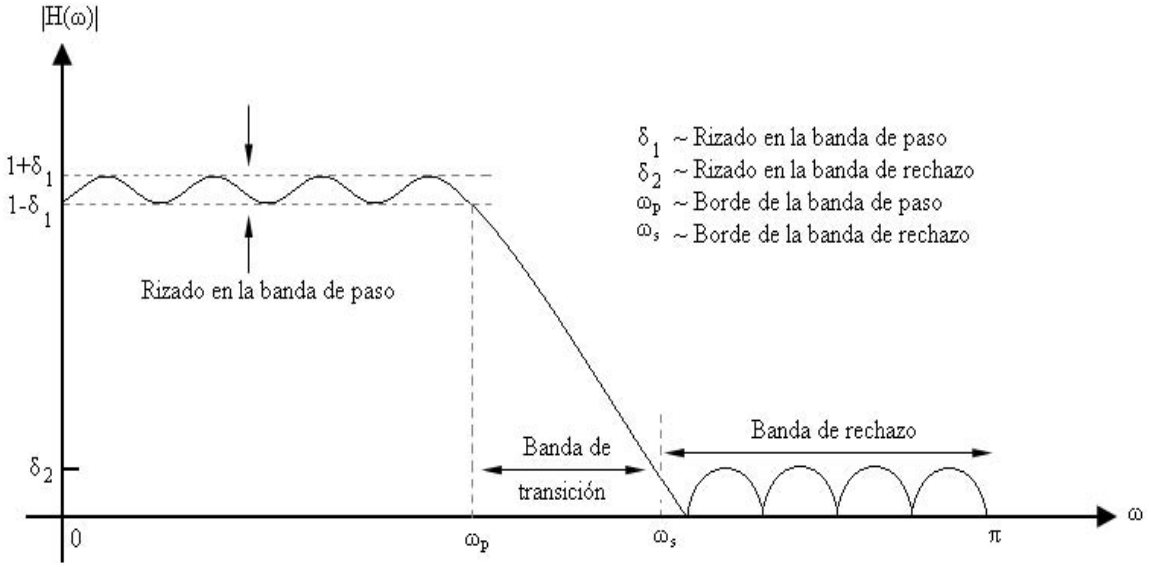

Figura C.2: Características de magnitud de filtros físicamente realizables.

Similarmente, en la banda de rechazo la respuesta en frecuencia del filtro se especifica para que esté entre los límites $\pm\delta_2$, es decir,

$$-\delta_2 \leq H(\omega) \leq \delta_2, \qquad |\omega| > \omega_s.$$

Así, δ_1 representa el rizado en la banda de paso y δ_2 representa la atenuación o rizado en la banda de rechazo. El parámetro del filtro que queda es M, la longitud del filtro o el número de coeficientes del filtro.

Hay cuatro casos principales que resultan de un filtro FIR de fase lineal:

- *Caso 1:* Respuesta impulsional simétrica $h(n) = h(M-1-n)$ y M impar.
- *Caso 2:* Respuesta impulsional simétrica $h(n) = h(M-1-n)$ y M par.
- *Caso 3:* Respuesta impulsional antisimétrica $h(n) = -h(M-1-n)$ y M impar.
- *Caso 4:* Respuesta impulsional antisimétrica $h(n) = -h(M-1-n)$ y M par.

Por conveniencia matemática se define una función de ponderación modificada $\hat{W}(\omega)$ y una respuesta en frecuencia deseada modificada $\hat{H}_{dr}(\omega)$ como

$$\hat{W}(\omega) = W(\omega)Q(\omega), \tag{C.5}$$

$$\hat{H}_{dr}(\omega) = \frac{H_{dr}(\omega)}{Q(\omega)}. \tag{C.6}$$

Entonces el error de aproximación ponderado se puede expresar como

$$E(\omega) = \hat{W}(\omega)[\hat{H}_{dr}(\omega) - H(\omega)],$$

para los cuatro diferentes tipos de filtros FIR de fase lineal. Dado el error de la función $E(\omega)$, el problema de aproximación de Chebyshev consiste básicamente en determinar los parámetros del filtro $\{\alpha(k)\}$ que minimizan el máximo valor absoluto de $E(\omega)$ sobre las bandas de frecuencia en las que se realiza la aproximación.

C.3. Filtros IIR

Los filtros IIR o de respuesta infinita al impulso *(Infinite Impulse Response)* en los que si la entrada es una señal impulso, la salida tendrá un número infinito de términos no nulos, es decir, nunca vuelve al reposo.

La salida de los filtros IIR depende de las entradas actuales y pasadas, y además de las salidas en instantes anteriores. Esto se consigue mediante el uso de realimentación de la salida.

Su expresión matemática es:

$$y_n = b_0 x_n + b_1 x_{n-1} + \cdots + b_N x_{n-N} + a_1 y_{n-1} + a_2 y_{n-2} + \cdots + a_M y_{n-M}$$

Donde a y b son los coeficientes del filtro. El orden es el máximo entre los valores de M y N. Aplicando la transformada **Z** a la expresión anterior, resulta:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k\, z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}}.$$

Hay numerosas formas de implementar los filtros IIR. En la Figura C.3 se muestra una estructura de un filtro IIR. La estructura afecta a las características

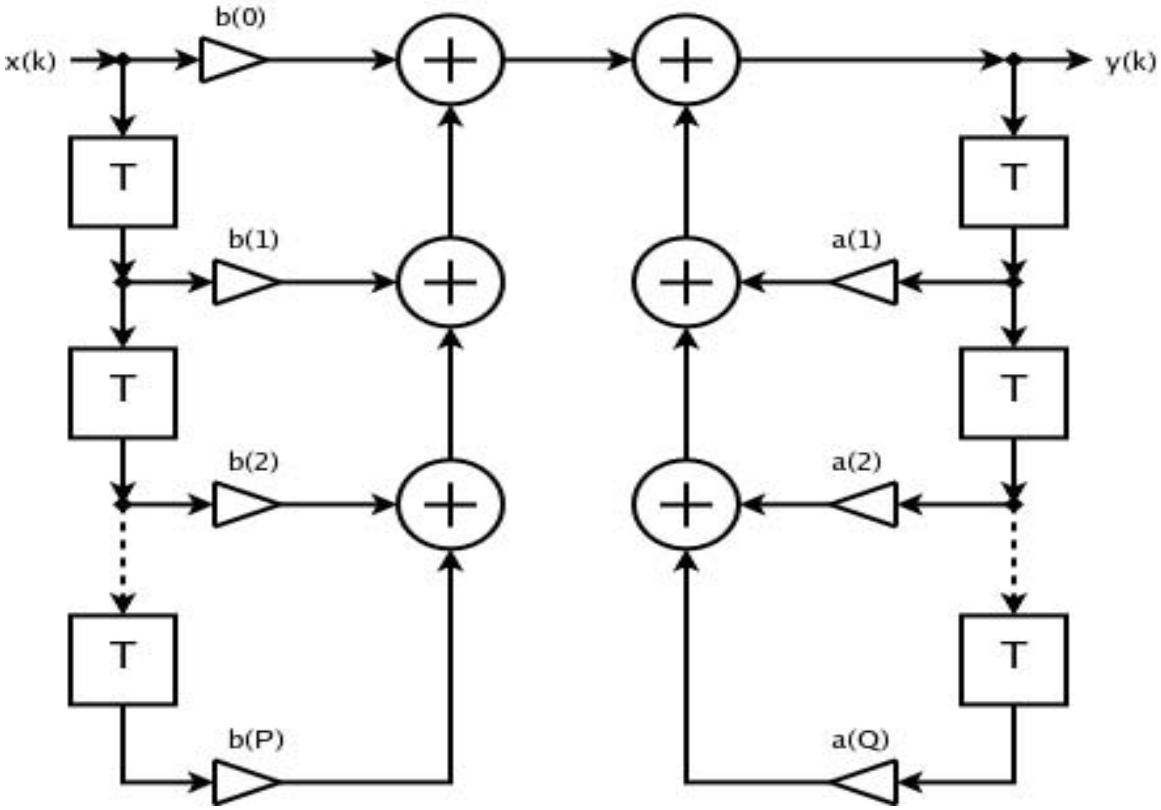

Figura C.3: Estructura básica de un IIR.

finales que presentará el filtro como la estabilidad. Otros parámetros a tener en cuenta a la hora de elegir una estructura es el gasto computacional que presenta.

Este tipo de filtros presenta polos y ceros que determinan la estabilidad y la causalidad del sistema. Cuando todos los ceros están en el interior de la circunferencia unidad se dice que es fase mínima. Si todos están en el exterior es fase máxima. Si algún polo está fuera de la circunferencia unidad el sistema es inestable.

Las formas habituales de diseñar este tipo de filtros son:

1. Indirecta (a partir de prototipos analógicos):
   - Impulso invariante.
   - Aproximación de derivadas.
   - Transformación bilineal.
2. Directa.

- Aproximación de Padé.
- Aproximación de mínimos cuadrados.

Las principales diferencias respecto a los filtros FIR es que los IIR pueden cumplir las mismas exigencias que los anteriores pero con menos orden de filtro. Esto es importante a la hora de implementar el filtro, pues presenta una menor carga computacional. Este tipo de filtros puede ser inestable, aún cuando se diseñan para ser estables. En principio no pueden diseñarse para tener fase lineal pero se pueden aplicar algunas técnicas como el filtrado bidireccional para lograrlo.

C.3.1. Diseño de filtros IIR a partir de filtros analógicos

Al igual que en el diseño de filtros FIR, hay varios métodos que se pueden utilizar para diseñar filtros digitales que tengan una respuesta impulsional de duración infinita. Las siguientes técnicas están basadas en convertir un filtro analógico en un filtro digital.

C.3.1.1. Diseño de filtros IIR mediante la aproximación de derivadas

Uno de los métodos más simples para convertir un filtro analógico en un filtro digital es aproximar la ecuación diferencial transferencia de un filtro analógico

$$\sum_{k=0}^{N} \alpha_k \frac{d^k(y(t))}{dt^k} = \sum_{k=0}^{M} \beta_k \frac{d^k(x(t))}{dt^k}, \tag{C.7}$$

mediante una ecuación en diferencias equivalentes. Esta aproximación se usa a menudo para resolver una ecuación diferencial lineal con coeficientes constantes de forma numérica en un procesador digital.

La función de transferencia para el filtro IIR digital obtenida como resultado de la aproximación de las derivadas mediante diferencias finitas es

$$H(z) = H_a(s)|_{s=(1-z^{-1})/T}$$

donde $H_a(s)$ es la función de transferencia del filtro analógico caracterizada por la ecuación diferencial dada en (C.7) es evaluada en $s = \frac{1-z^{-1}}{T}$ o equivalentemente,

$$z = \frac{1}{1 - sT} \tag{C.8}$$

si se sustituye $s = j\Omega$ en (C.8), resulta

$$z = \frac{1}{1 - j\Omega T} = \frac{1}{1 + \Omega^2 T^2} + j\frac{\Omega T}{1 + \Omega^2 T^2}$$

Como Ω varía desde $-\infty$ hasta ∞, el correspondiente lugar de puntos en el plano z es un círculo de radio $\frac{1}{2}$ y con centro en $z = \frac{1}{2}$.

Se demuestra fácilmente que la correspondencia en (C.8) lleva puntos del semiplano izquierdo del plano s a puntos correspondientes dentro de ese círculo en el plano z y puntos del semiplano derecho del plano s se corresponden con puntos fuera de este círculo. Consecuentemente, esta correspondencia tiene la propiedad deseable que un filtro analógico estable se transforma en un filtro digital estable.

C.3.1.2. Diseño de filtros IIR mediante invarianza impulsional

En el método de varianza impulsional, el objetivo es diseñar un filtro IIR con una respuesta impulsional $h(n)$ que sea la versión muestreada de la respuesta impulsional del filtro analógico. Es decir,

$$H(n) \equiv h(nT), \qquad n = 0, 1, 2, \ldots$$

donde T es el periodo de muestreo.

El filtro digital con respuesta en frecuencia $H(\omega)$ tiene las características de res-puesta en frecuencia del correspondiente filtro analógico si el periodo de muestreo T se selecciona suficientemente pequeño para evitar completamente o al menos minimizar los efectos del *aliasing*. El método de varianza impulsional es inapropiado para diseñar filtros paso bajo, debido al efecto en el espectro que resulta del proceso de muestreo.

La función de transferencia del filtro analógico se expresa en forma de fracciones simples de acuerdo con el efecto del método de diseño de varianza impulsional en las características del filtro resultante. Suponiendo que los polos del filtro analógico son distintos

$$H_a(s) = \sum_{k=1}^{N} \frac{c_k}{s - p_k},$$

donde $\{p_k\}$ son los polos del filtro analógico y $\{c_k\}$ son los coeficientes en la expansión en fracciones simples. Consecuentemente,

$$h_a(t) = \sum_{k=1}^{N} c_k \, \mathrm{e}^{p_k t}, \qquad t \geq 0.$$

La función de transferencia del filtro digital es

$$H(z) = \sum_{k=1}^{N} \frac{c_k}{1 - \mathrm{e}^{p_k t} z^{-1}}.$$

Entonces, el filtro digital tiene polos en

$$z_k = \mathrm{e}^{p_k t}, \qquad k = 1, 2, \ldots, N.$$

Aunque los polos del plano s corresponden con los del plano z, debe resaltarse que los ceros en los dos dominios no satisfacen la misma relación.

C.3.1.3. Diseño de filtros IIR mediante la transformación bilineal

La técnicas de diseño de filtros IIR descritas anteriormente tienen una limitación severa, que sólo son apropiadas para filtros paso bajo y una clase limitada de filtros paso banda.

La transformación bilineal es una correspondencia del plano s al plano z que soluciona la limitación de los otros dos métodos.

La transformación bilineal es una correspondencia conformadora que transforma el eje $j\Omega$ en la circunferencia unidad del plano z sólo una vez, evitando el solapamiento de componentes de frecuencia. Además, todos los puntos en el semiplano izquierdo de s se corresponden con el interior de la circunferencia unidad en el plano z y todos los puntos en el semiplano derecho de s se corresponden con puntos fuera de la circunferencia unidad del plano z.

Apéndice D

Algoritmos de programación

load.asm

Este archivo contiene una rutina de cargado, un simple ciclo(loop) que es invocado desde lenguaje c con un argumento.

```
;  All rightsreserved. Property of Texas Instruments Incorporated

        .ref _load

        .text

N       .set    1000

;   ======== _load ========

;  This function simulates a load on the DSP by executing N *
loopCount

;  instructions, where loopCount is the input parameter to load().

; _load:

        mv a4, b0               ; use b0 as loop counter
  [!b0] b lend
        mvk N,b1
        mpy b1,b0,b0
```

```
        nop
        shru b0,3,b0            ; (loop counter)= (# loops)/8

loop:
        sub b0,1,b0
        nop
   [b0] b loop
        nop 5

lend:   b b3
        nop 5                   ; return

        .end
```

vector.asm

```
;  All rights reserved. Property of Texas Instruments Incorporated.

        .global unused:id:
unused:id:
        b unused:id:    ; nested branches to block interrupts
        nop 4
        b unused:id:
        nop
        nop
        nop
        nop
        nop

        .endm

        .sect ".vectors"

        .ref _c_int00           ; C entry point

        .align  32*8*4          ; must be aligned on 256 word boundary

RESET:                          ; reset vector
```

```
        mvkl _c_int00,b0        ; load destination function address to b0
        mvkh _c_int00,b0
        b b0                    ; start branch to destination function
        mvc PCE1,b0             ; address of interrupt vectors
        mvc b0,ISTP             ; set table to point here
        nop 3                   ; fill delay slot
        nop
        nop

        ;
        ;  plug unused interrupts with infinite loops to
        ;  catch stray interrupts
        ;
        unused 1
        unused 2
        unused 3
        unused 4
        unused 5
        unused 6
        unused 7
        unused 8
        unused 9
        unused 10
        unused 11
        unused 12
        unused 13
        unused 14
        unused 15
```

dss_dsk6711.c

```
/*         Programa, configura:
            Codec
            puerto serial */

/*===dss_dsk6711.c===*/
```

```
#include <std.h>

#include <log.h>

#include <c6x.h>

#include "c6711dsk.h"

#include "oooacfg.h"

#define MCSP_RXINT_BIT 0x0800

/* define McBSP interrupt */

extern far LOG_Obj trace;


/* function prototypes ... */

static Void codec_init(Void);

static Void codecError(Int id);

static Void mcbsp0_init(void);

Uns mcbsp0_read(void);

Void mcbsp0_write(Uns out_data);

Void DSS_init(void) {


    mcbsp0_init();
    codec_init();
    /* Enable McBSP interrupt */
    IER |= MCSP_RXINT_BIT;
    /* Reset McBsp then enable Transmit and Receive bits */


}
```

```
/*
* ======== mcbsp0_init ========
*/

Void mcbsp0_init(void) {
    /* set up McBSP0 */
    *(volatile Uns *)McBSP0_SPCR = 0x0; /* reset serial port */
    *(volatile Uns *)McBSP0_PCR = 0x0; /* set pin control reg. */
    /* set RX and TX control registers to 16 bit data/frame */
    *(volatile Uns *)McBSP0_RCR = 0x10040;
    *(volatile Uns *)McBSP0_XCR = 0x10040;
    /* setup SP control register */
    *(volatile Uns *)McBSP0_SPCR = 0x00010001;
}


/*
 *  ======== codecInit ========
 */
Void codec_init(void) {
    volatile Uns temp;
    /* set up control register 3 for S/W reset */
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x0386);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    /* set up control register 3 for mic input */
    mcbsp0_write(0);
    mcbsp0_read();
```

```
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0306);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();

/* read control register 3 to verify mic input */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2330);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xff) != 0x06) {
    codecError(3);
}

/* set up control register 4 to select Voice Channel Input 0db gain */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0400);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();

/* read and verify control register 4 */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
```

```
mcbsp0_read();
mcbsp0_write(0x2430);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xff) != 0x00) {
    codecError(4);
}

/* set up control register 5 to select Voice Channel Output 0db gain */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0502);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();

/* read and verify control register 5 */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2530);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xfe) != 0x2) {
    codecError(5);
}

/*
* ======== codec_error ========
```

```
    */
    Void codec_error(id) {
        LOG_error("Error del sistema %d", id);
        for (;;) {
        /* loop forever */
        }
    }
}
```

principal.c

```
 /*         Programa para  ventaneo
       utiliza  ventana rectangular
           utiliza la formula tv=Sv + Iv

     Donde:
        tv= Tamaño de ventana , muestras,
        sv= Solapamiento de ventana
        iv= incremento de ventana

*/

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#include "c6711dsk.h"
#include "oooecfg.h"

#define MCSP_RXINT_BIT 0x0800 /* Define la interrupción McBSP  */
#define iv 98
#define sv 30
```

```
/* Funciones prototipo ... */

extern Void DSS_init(void); /*Inicializa CODEC y puerto serial */
extern Void DSS_isr(void);
extern Void codec_init(void);

extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);

extern Void mcbsp0_write(Uns out_data);

Void mtca(short a[],short b[],short c[],short nx);

extern void load(unsigned int loadValue);

static int processing(short *output, short *aux);


short tv;
short x;
short i=iv+sv;
short j=-1;
int m=iv+sv;

short sai[iv+sv];
short sbi[iv+sv];
short fai[iv+sv];

short fbi[iv+sv];
short impar[iv+sv];
short sap[iv+sv];

short sbp[iv+sv];
short fap[iv+sv];
short fbp[iv+sv];

short par[iv+sv]; /*Coeficientes ventana hamming */ short
in[tv];
srev[tv];
```

```
floa vha[ ]={0.080000,0.080926,0.083701,0.088313,0.094744,0.102967,
       0.112951,0.124654,0.138029,0.153023,0.169576,0.187620,
       0.207082,0.227886,0.249946,0.273174,0.297476,0.322755,
       0.348909,0.375832,0.403417,0.431551,0.460122,0.489014,
       0.518112,0.547298,0.576455,0.605465,0.634211,0.662578,
       0.690451,0.717719,0.744271,0.770000,0.794803,0.818580,
       0.841236,0.862678,0.882822,0.901584,0.918891,0.934672,
       0.948864,0.961410,0.972259,0.981367,0.988698,0.994222,
       0.997917,0.999768,0.999768,0.997917,0.994222,0.988698,
       0.981367,0.972259,0.961410,0.948864,0.934672,0.918891,
       0.901584,0.882822,0.862678,0.841236,0.818580,0.794803,
       0.770000,0.744271,0.717719,0.690451,0.662578,0.634211,
       0.605465,0.576455,0.547298,0.518112,0.489014,0.460122,
       0.431551,0.403417,0.375832,0.348909,0.322755,0.297476,
       0.273174,0.249946,0.227886,0.207082,0.187620,0.169576,
       0.153023,0.138029,0.124654,0.112951,0.102967,0.094744,
       0.088313,0.083701,0.080926,0.080000};

/*Coeficientes FIR*/

float cfl[ ]={0.000656, -0.000687, -0.001713, 0.000989, -0.000504,
-0.004064, 0.001595, 0.000787, -0.008723, 0.001439, 0.004955,
-0.015534, -0.001440, 0.014076, -0.023528, -0.010387, 0.031450,
-0.031124, -0.033495, 0.068697, -0.036586, -0.121574, 0.283799,
0.627440, 0.283799, -0.121574, -0.036586, 0.068697, -0.033495,
-0.031124, 0.031450, -0.010387, -0.023528, 0.014076, -0.001440,
-0.015534, 0.004955, 0.001439, -0.008723, 0.000787, 0.001595,
-0.004064, -0.000504, 0.000989, -0.001713, -0.000687, 0.000656};

float revoloteo[]={0.605465,0.576455,0.547298,0.518112 -0.121574,
-0.036586, 0.068697,-0.033495,0.227886,0.207082,0.187620,0.169576,
0.690451,0.717719,0.744271,-0.001440, 0.014076, -0.023528,
-0.010387,-0.004064, 0.001595, 0.000787, -0.008723,0.403417,
0.375832,0.348909};


int out_buffer[BUFSIZE];
unsigned int processingLoad = BASELOAD;

struct PARMS str = {
```

```
    2934,
    1432,
    213,
    1432,
    &str
};


/*
* ======== main ========
*/
Void main() {
   DSS_init();   /* Esta función se definió en el archivo */
                 /* dss_dsk6211.c descrito previamente */
    LOG_printf(&trace,"Configuración realizada");
    return;
}


interrupt Void DSS_isr(Void)    /*Bloque de interrupciones */ {

    short *output = &out_buffer[0];
    tv=iv+sv;
    x=mcbsp0_read();

    if((i==sv)&&(sv!=0))         /* Solapamiento  par*/
      {
        j=tv;
      }


     if((i<=tv)&&(i>=1))           /* Ventanas impar*/
     {
          i--;
          sai[i]=0;
          sbi[i]=0;
          fai[i]=0;
          fbi[i]=0;
          impar[i]=0;
          impar[i]=x;
```

```
      if(i==0)
      {
            m--;
            me[m]=0;
            fltoq15(vha,sai,tv);
            mtca(impar,sai,sbi,tv);
            fltoq15(cfl,fai,tv);
            fir_gen(sbi,fai,fbi,47,100);
            fltoq15(revoloteo,srev,256);
            radix2(tv,fbi,srev);
            digitrev_index(in,128,2)
            bitrev_cplx(srev,128,in)
            processing(output, srev);
      }
}



if((j<=tv)&&(j>=1))           /*Ventanas par*/
{
   j--;
   sap[j]=0;
   sbp[j]=0;
   fap[j]=0;
   fbp[j]=0;
   par[j]=0;
   par[j]=x;
   if(j==0)
   {
       m--;
       me[m]=0;
       fltoq15(vha,sap,tv);
       mtca(par,sap,sbp,tv);
       fltoq15(cfl,fap,tv);
       fir_gen(sbp,fap,fbp,47,100);
       fltoq15(revoloteo,srev,256);
       radix2(tv,fai,srev);
       digitrev_index(in,128,2)
       bitrev_cplx(srev,128,in)
       processing(output, srev);
```

```
        }
      }

      if((j==sv)&&(sv!=0))          /*Solapamiento impar*/
      {
         i=tv;
      }


      if((j==0)&&(sv==0))           /*retroalimentación impar*/
      {
         i=tv;
         j--;
      }

      if((i==0)&&(sv==0))           /*retroalimentación par*/
      {
         j=tv;
         i--;
      }

      if(m==0)                        /* Valores media */
      {
        impriflt(me,tv);
        m=tv;
      }


}

Void mtca(short a[],short b[],short c[],short nx) {
  short k;
  for(k=(nx-1);k!=-1;k--)
       c[k]=a[k]*b[k];
}

static int processing(short *output, short *aux) {
    int size = BUFSIZE;
    while(size--){
        *output++ = (*aux++);
```

```
        }
        load(processingLoad);
        return(TRUE);
    }
```