191

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

" IMPLEMENTACION DE UN MODELO DE COMPRESION DE VOZ Y SU APLICACION EN SISTEMAS DE TELECOMUNICACIONES "

FALLA DE ORIGEN

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

AREA ELECTRICA - ELECTRONICA

P R E S E N T A N :

LILIA ELENA DE LA VEGA SEGURA

ANTONIO ECHEVERRIA LOZANO

DIRECTOR DE TESIS: DR. BOHUMIL PSENICKA

MEXICO, D. F. 1995

Implementación de un modelo de compresión de voz y su aplicación en sistemas de telecomunicaciones

Lilia Elena de la Vega Segura
Antonio Echeverría Lozano

Mayo 1995

Agradecimientos

Al Dr. Bohumil Pšenička, por su apoyo y amistad sincera que fueron invaluables para realizar este trabajo.

A nuestra Universidad, la UNAM, a la cual debemos nuestra formación y los horizontes que como profesionistas podemos tener.

A nuestros maestros, por compartir con nosotros sus experiencias y conocimientos.

Dedicatorias

A Dios,
por ser la guía de mi vida.

A mis papas, Alejandro y Carmen,
por su infinito amor, apoyo
y lo valioso de sus consejos.

A mi familia, Ale, Isaac,
Dulce, Edy, Ascan y Aldo,
por su cariño incondicional.

A mi hermana Care,
por su apoyo y ser un
gran ejemplo a seguir.

A mis amigos,
por el gran cariño
que siento hacia ellos.

Lili.

Dedicatorias

A mis papás, Antonio y Guillermina,
por una vida con su apoyo y cariño, gracias.

A mis hermanas, Guille y Ale,
porque las quiero.

A mi familia, que siempre ha estado
conmigo y le da sentido a mis metas.

A todos mis amigos, con quienes comparto el
tesoro mas valioso que poseo: la amistad.

Antonio Echeverría L.

Contenido

Introducción

Debido a la evolución de los sistemas de comunicaciones hacia esquemas totalmente digitales, las tecnologías para el procesamiento digital de la voz han experimentado un creciente desarrollo desde hace varias décadas. La necesidad de arquitecturas de comunicaciones más eficientes ha acelerado este crecimiento en diversas áreas: depuración constante de los algoritmos de codificación; un rápido desarrollo en la electrónica que se ha especializado en este campo; asimismo las arquitecturas de redes de comunicaciones sobre las cuales se transmite este medio incrementan sus capacidades rápidamente.

En la presente tesis se pretende proporcionar un panorama que abarque los distintos aspectos que comprende la implantación de un modelo de codificación de voz. Dicho proceso inicia con un planteamiento teórico del modelo a implementarse, seguido de una evaluación de las variantes del modelo teórico elegido y de cómo dichas variantes han sido aplicadas en sistemas de comunicaciones existentes. A continuación, se evalúan los recursos tecnológicos disponibles para poder llevar a buen término la implementación de un modelo de codificación en un sistema específico. Se presenta también la implementación del modelo de codificación LPC[1] realizada en una computadora personal y las características obtenidas con esta aproximación. Como último objetivo de este trabajo se analiza de una forma más precisa la forma en que un codificador LPC puede ser implementado en un sistema de comunicaciones y las implicaciones que ello tiene en los distintos estratos que lo conforman.

Previo a una exposición detallada del modelo de codificación que se ha tratado aquí con mayor amplitud (Codificación Lineal Predictiva, LPC), es necesario considerar las bases a partir de las cuales se ha generado este modelo. En el primer capítulo, se efectúa una breve descripción de la mecánica del aparato vocal humano, así como del aparato auditivo que actúa como receptor. Esta primera parte del trabajo es de particular importancia para comprender posteriormente en que forma los métodos LPC parametrizan la señal original. Se reseñan también los fundamentos del análisis digital de la forma de onda, ya que son herramientas indispensables que se utilizan en la mayor parte de los codificadores digitales. En realidad, el procesamiento digital de voz es una ciencia muy extensa, que abarca no únicamente la codificación, sino también áreas como el reconocimiento y la síntesis. De hecho, algunas de las herramientas de procesamiento que se mencionan tienen también aplicación en las áreas antes mencionadas.

El desarrollo de los codificadores de voz se ha visto influenciado por diversos factores que han dado por resultado una gama muy extensa de esquemas y tecnologías. Estos factores han sido principalmente, la calidad deseada, la velocidad de transmisión requerida, la disponibilidad de electrónica capaz de ejecutar los algoritmos de codificación en el tiempo

[1] LPC: Linear Predictive Coding

requerido y el costo de cada tecnología. Es precisamente en el segundo capítulo donde se expone con cierta amplitud la gran variedad de tecnologías de codificación, sus aplicaciones y los protocolos en diversos sistemas de comunicaciones. Se hace un especial énfasis en aquellos codificadores derivados del modelo LPC y las distintas estrategias que se idearon para lograr el equilibrio entre los factores ancho de banda, calidad y costo.

Las bases teóricas de los algoritmos de codificación, en especial aquellas técnicas que involucran grandes cantidades de operaciones aritméticas, como lo son LPC y las técnicas de análisis en frecuencia, se hallaban disponibles y con la madurez suficiente para utilizarse como esquemas de codificación (tal es el caso del algoritmo de Levinson, inventado desde 1917). Sin embargo, es hasta décadas recientes cuando la electrónica ha permitido implementarlas en tiempo real a costos razonables. Los microprocesadores fabricados expresamente para poder ejecutar este tipo de operaciones de manera intensiva, conocidos como procesadores digitales de señales (DSP's), son el tema tratado en el capítulo tres. El estudio de los DSP's puede considerarse en sí mismo como una especialidad dentro del ramo de la electrónica digital. Los primeros dispositivos dedicados al cálculo intensivo, que comenzaron como simples arreglos de compuertas, han evolucionado en aspectos tales como la velocidad de ejecución, la flexibilidad en la arquitectura, herramientas y entornos de desarrollo, capacidad para desarrollar software de una manera rápida y eficiente, y la compatibilidad de estos de manera que puedan fácilmente ser implantados como subsistemas en esquemas más generales de transmisión. Se consideró el caso particular de un dispositivo (el TMS320C30 de Texas Instruments) que tiene capacidades sobresalientes en cuanto a su flexibilidad de programación y la facilidad con la que pueden ser probados en él algoritmos de procesamiento digital de señales. Se consideran los diversos entornos de desarrollo existentes así como el caso específico de una tarjeta evaluadora (EVM) utilizada para desarrollar aplicaciones con el microprocesador mencionado.

En el capítulo cuatro se detalla la implementación, en lenguaje C, de un codificador LPC con una tasa de codificación variable y con amplias capacidades para modificar el efecto de las diversas operaciones que constituyen en su conjunto un sistema de análisis/síntesis de voz. El programa que realiza estas operaciones, llamado ADFO (Análisis Digital de Formas de Onda) es extremadamente flexible en cuanto a los formatos que pueden ser utilizados como entradas -es posible tomar como entrada de los algoritmos formas de onda generadas internamente, exportar datos de formatos comerciales de voz, o bien digitalizar señales mediante una tarjeta -. Mediante el uso de un formato de archivo que permite el almacenamiento eficaz de la información, es posible reconstruirla y simular diversos efectos como son la suma de señales, atenuaciones y sucesivas etapas de *compresión* y *descompresión*. El programa también posee utilerías de graficación, impresión y generación de archivos en formatos compatibles con otros programas utilizados para el procesamiento digital de señales, como MATLAB.

A continuación, en el capítulo cinco se aborda la aplicación de un codificador LPC en diversos sistemas de comunicaciones. Se hace especial énfasis en las implicaciones de implantar un sistema de transmisión de voz en redes de conmutación de paquetes, así como un análisis comparativo contra otros medios como los datos y el video. Se hace un análisis de el uso de el codificador en otro tipo de redes, como pueden ser las redes satelitales. Se tratan también los aspectos relacionados con la inclusión del algoritmo en un protocolo de voz y la ubicación de este dentro de las distintas capas de protocolos con referencia al modelo OSI.

Se presenta como caso de estudio el algoritmo utilizado en la red de telefonía celular europea (GSM), presentándose algunas de sus características generales.

En el capítulo seis se presentan algunas pruebas realizadas sobre el algoritmo diseñado así como un análisis de ellas. Se hace una evaluación de las implicaciones para la implementación en tiempo real, particularizando en las capacidades del microprocesador TMS320C30. Por último, se incluye una conclusión sobre la investigación y el programa aquí presentados, se comentan asimismo las perspectivas hacia posteriores desarrollos y trabajos colaterales.

Capítulo 1

Introducción al procesamiento de la voz

En el presente capítulo se hace una recopilación de las técnicas y enfoques de uso más frecuente en el procesamiento digital de la voz. Se hace una breve semblanza de las bases fisiológicas de la producción y percepción humana del lenguaje hablado, y cómo dichas bases repercuten en los métodos de análisis comúnmente utilizados para procesar la señal eléctrica que se toma como entrada de un sistema de PDS. Se hace especial énfasis en aquellas técnicas que permitirán posteriormente procesar la dicha señal para los objetivos particulares de este trabajo. Cabe aclarar que los métodos y procedimientos de análisis pueden tener otras aplicaciones además de los descritos, por lo que su uso no se circunscribe solo al fin que se persigue.

1.1 Producción de la voz

1.1.1 Dinámica del aparato vocal

Con la finalidad de tener un mejor entendimiento de las técnicas utilizadas en el procesamiento digital de la voz, es necesario primero conocer la forma en que la voz se produce. Existen diferentes modelos para caracterizar la producción de sonidos artificialmente, y la mayoría de ellos se basan en una idealización del aparato vocal.

Los sonidos que percibimos como lenguaje hablado, llegan hasta nosotros como una onda de presión la cual toma su forma final en el aparato vocal.

Inicialmente estos sonidos son producidos al pasar el aire exhalado por los pulmones a través de las cuerdas vocales. Normalmente el 60% del ciclo de respiración se utiliza en la exhalación y el 40% restante en la inhalación, sin embargo durante los períodos de habla, el tiempo de exhalación se prolonga hasta un 90%. La respiración en reposo exige a los pulmones aproximadamente el 10% de su capacidad mientras que al hablar el esfuerzo se incrementa hasta en un 40%.

La siguiente etapa en la producción de sonidos es la laringe en la cual se ubican las cuerdas vocales, las cuales consisten en dos segmentos de materia muscular y membranosa llamados cartílagos tiroide y cricoide. Es en este punto donde se producen los llamados

sonidos vocales los cuales se caracterizan, como se verá mas adelante, por poseer un espectro donde destacan dos o tres frecuencias dominantes.

Al pasar el aire a través de la laringe la tensión en el interior varía de manera que se producen estrechamientos que se desplazan periódicamente a lo largo de ella. Esta oscilación del cartílago tiroide se traduce en la producción de una frecuencia vocal fundamental la cual depende principalmente de las dimensiones de las cuerdas y del tamaño de la laringe. Usualmente en los hombres el tamaño de cada cuerda varía entre los 17 y los 24 mm, mientras que en las mujeres está entre 13 y 17 mm. En promedio esta frecuencia fundamental es de 125 Hz en los hombres, de 200 Hz en las mujeres y alcanza los 300 Hz en los niños. A este proceso se le conoce como la teoría mioelástica-aerodinámica de la fonación.

El tracto vocal que recibe el sonido producido en la laringe, realiza el modelado espectral de la señal caracterizando de esta manera el tono de voz particular en cada persona. Es decir, en el tracto vocal algunas frecuencias obtienen mayor o menor ganancia que otras dependiendo de la morfología del aparato vocal.

Una aproximación que lo modela elegantemente es la siguiente, donde se supone inicialmente un tubo simple sin pérdidas a través del cual se desplaza una onda plana de manera axial. La variación de presión $p(x,t)$, función de la posición x y el tiempo t, y la variación de velocidad de flujo volumétrico $u(x,t)$, están dadas por:

$$-\frac{\partial p}{\partial x} = \rho\frac{\partial(u/A)}{\partial t} \qquad -\frac{\partial u}{\partial x} = \frac{1}{\rho c^2}\frac{\partial(pA)}{\partial t} + \frac{\partial A}{\partial t} \tag{1.1}$$

Donde además c es la velocidad del sonido (340 m/s) y $A(x,t)$ es el área seccional del tubo, la cual se supone uniforme. Las resonancias en este tubo se dan cuando su longitud es de 1/4 de la longitud de onda y múltiplos impares de este valor. Así por ejemplo el tracto vocal típico en un hombre es de 17 cm, de donde la primera frecuencia de resonancia resulta ser

$$f_1 = \frac{c}{\lambda} = \frac{340}{(4)(0.17)} = 500Hz \tag{1.2}$$

de la misma manera si se calculan las siguientes frecuencias de resonancia obtenemos para 3λ, $f_2 = 1500$ Hz y para 5λ, $f_3 = 2500$ Hz.

Las frecuencias obtenidas anteriormente resultan ser, notoriamente, las frecuencias que caracterizan el espectro de la llamada vocal neutra, que se emite al dejar salir el aire naturalmente a través del tracto sin efectuar ningún tipo de constricción en el mismo.

En la figura 1.1 se esquematizan los principales órganos que intervienen en la articulación del lenguaje.

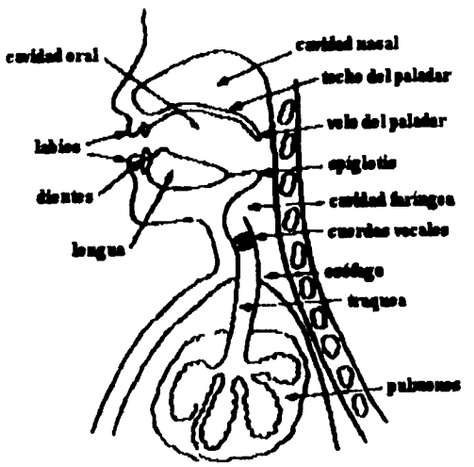

Figura 1.1: Organos vocales del cuerpo humano

En la producción del sonido que asciende por la cavidad faringea (fig. 1.1), interviene también la cavidad oral la cual se encuentra separada de la cavidad nasal mediante el velum del paladar. La cavidad nasal está siempre inmóvil y dependiendo de la posición del velum se acopla o no al tracto vocal para intervenir en algunos sonidos. La generación de los sonidos no vocales (nasales, fricativos, y consonantes en general), se realiza mediante la manipulación de la lengua, labios, dientes y mandíbula dentro de estas cavidades.

1.1.2 Producción de los sonidos

Se acostumbra hacer una distinción de los sonidos en función de la forma en que se producen. De esta manera se les clasifica en vocales, no vocales y plosivos. Los sonidos vocales se generan por la vibración de las cuerdas vocales, y en el dominio de la frecuencia se distinguen por presentar marcadas frecuencias dominantes llamadas formantes. Los sonidos no vocales como ya se mencionó, se producen en el tracto vocal mediante la fricción causada por la turbulencia del aire y constricciones en diferentes lugares de las cavidades. Los plosivos son en realidad una variante de los sonidos no vocales que se producen cuando el flujo de aire es detenido y la presión se libera repentinamente de manera similar a una explosión. Los silencios juegan también un papel importante, sin embargo pese a lo que pudiera pensarse son más significativos después de un sonido plosivo que entre palabras. Su duración está comúnmente entre los 30 y los 50 ms.

El elemento más pequeño en un lenguaje en el cual es posible diferenciar un significado se le llama *fonema*, y usualmente se les denota escribiéndolos entre diagonales. En cada idioma existe un conjunto específico de fonemas, sin embargo se ha logrado unificar en cierta medida algunos fonemas en un alfabeto fonético internacional (IPA). el cual se muestra en la siguiente tabla.

Símbolo Americano	*Símbolo IPA*	*Palabra Clave*	*Símbolo Americano*	*Símbolo IPA*	*Palabra Clave*
Diptongos			Semi-vocales		
/ei/	ei	aid	/w/	w	we
/u@/	$\upsilon\ \partial$	pure	/y/	j	you
/au/	au	cow	/r/	r	red
/ou/	∂ u	own	/l/	l	live
Plosivos			Nasales		
/p/	p	pie	/m/	m	me
/t/	t	ten	/n/	n	no
/k/	k	key	/ng/	η	sing
/b/	b	be	Africativos		
/d/	d	den	/tsh/	t $\int$	chair
/g/	g	go	/dzh/	d $\ni$	jar
Vocales			Fricativos		
/ee/	i	each	/f/	f	free
/i /	I	it	/th/	Θ	thin
/e/	ϵ	end	/s/	s	see
/ar/	α	hard	/sh/	$\int$	shall
/u/	υ	good	/v/	v	vine
/uu/	u	ooze	/dh/	δ	then
/er/	3	bird	/z/	z	zoo
/aa/	$\neq$	hat	/xh/	$\ni$	azure
/a/	Λ	bud	/h/	h	he
/aw/	$\supset$	hoard			
/@/	∂	allow			
/Q/	ρ	hot			

Se ha mostrado la representación norteamericana de los fonemas, así como ejemplos en el idioma Inglés, dado que la mayoría de la bibliografía accesible se encuentra en este idioma.

Vocales

Las resonancias en los tractos nasal y oral cambian dinámicamente debido al movimiento de las articulaciones vocales. Actúan como una caja de resonancia que da mayor o menor atenuación a las frecuencias básicas y sus armónicas provenientes de la laringe. La menor frecuencia de resonancia es llamada el primer formante, la segunda el segundo formante y así sucesivamente. El número de formantes puede estar entre dos y seis, dependiendo del individuo, teniendo considerables variaciones en magnitud y frecuencia, sin embargo es la relación entre las frecuencias formantes lo que hace a cada sonido reconocible. Las variaciones por lo general se centran alrededor de las frecuencias formantes de la vocal neutra, esto es a los 500, 1500 y 2500 Hz.

Por lo general sólo destacan las tres primeras formantes, y en la siguiente tabla pueden verse los valores típicos de las vocales producidas por un hombre en el idioma inglés.

Vocal	*Palabra Clave*	f_1	f_2	f_3
/ee/	beat	280	2620	3380
/i/	bit	360	2220	2960
/e/	bet	600	2060	2840
/er/	bird	560	1480	2520
/ar/	father	740	1110	2640
/a/	hut	760	1370	2500
/u/	hood	480	740	2620
/uu/	loot	320	920	2200

En una gráfica que muestre como abscisa y ordenada a la primera y segunda frecuencias formantes respectivamente puede observarse que las demás vocales se ubican alrededor de la vocal neutra formando un triángulo, llamado el *triángulo vocal*. En la fig. 1.2 se muestran los rangos típicos de los dos primeros formantes de las vocales producidos por diferentes personas.

Diptongos

La concatenación de dos vocales forman los diptongos y durante su producción se efectúan cambios en el tracto vocal para cambiar las resonancias. El sonido vocal es continuo y el tracto cambia gradualmente de la primera a la segunda posición de resonancia. Existen también algunos triptongos formados por tres vocales concatenadas.

Semi-vocales

Se les llama de esta manera debido a que también son producidos por resonancias en el tracto vocal al igual que las vocales, pero son usadas como consonantes. Se les clasifica en deslizantes y líquidas; a las primeras se les llama así por que los movimientos del tracto vocal provocan que las formantes se deslicen hacia abajo y hacia arriba como en la /w/ y en la

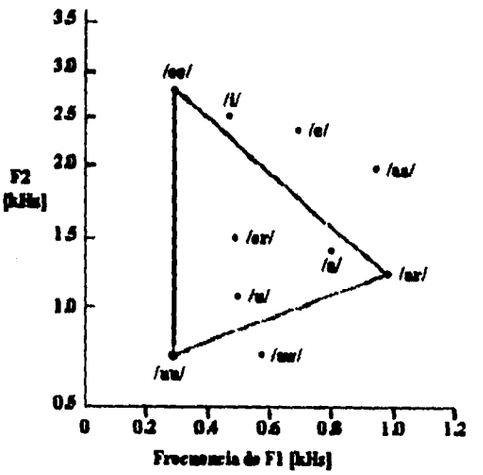

Figura 1.2: Posición de los formantes producidos por diferentes personas

/y/, mientras que las segundas se producen por el contacto de la lengua sobre el paladar y en los dientes superiores como en la /r/ y en la /l/.

Para los niños los líquidos son mas difíciles de producir por lo que comúnmente los sustituyen por deslizantes, como por ejemplo "quie/r/o" es sustituido por "quie/y/o".

Plosivos

También llamadas consonantes de paro, se caracterizan por una repentina liberación de aire al abrirse un punto de constricción en el tracto vocal como en /p/, /t/, ó /k/. La cantidad de aire liberada en un plosivo está afectada siempre por el contexto, liberándose mayor cantidad al inicio de una palabra que en medio de ella.

Nasales

Este tipo de sonidos se producen al abrirse el velum del paladar y cerrarse la boca, de manera que la cavidad nasal actúa como una caja de resonancia colateral emitiéndose los sonidos a través de las fosas nasales. Su intensidad es por lo general menor que la de los otros sonidos debido a que la cavidad nasal posee paredes de piel suave que absorben el sonido.

Fricativos

En esta clasificación se incluyen la mayoría de las consonantes, las cuales se producen por una constricción en algún punto del tracto vocal, de manera que el aire es forzado a pasar

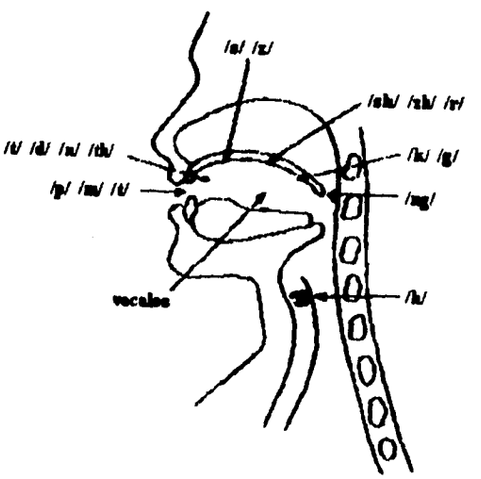

Figura 1.3: Localización de algunos sonidos en el aparato vocal

creando fricción, emitiendo vibraciones de ruido aleatorio. Por lo general, la constricción se realiza en algún punto del paladar, los dientes o los labios, sin embargo para el fonema /h/ esta constricción se lleva a cabo en la laringe.

Africativos

En el idioma inglés se definen solo dos de ellos. Se forman cuando los plosivos /t/ y /d/ se liberan gradualmente, dando por resultado un período más largo de fricción.

En la fig. 1.3 se esquematizan los sitios en el tracto vocal de importancia en la producción de varios sonidos

1.1.3 Variables que intervienen en la prosodia

Si se entiende como prosodia el acto de pronunciar y acentuar fonemas para articular el lenguaje hablado, podemos distinguir los diferentes factores que intervienen en su producción.

Tono de la voz. Depende de la frecuencia fundamental a la cual oscilan las cuerdas vocales, y comúnmente varía dependiendo del énfasis que se da a algunas sílabas. En subsecuentes capítulos este tono (*pitch*) será de gran relevancia al modelar el comportamiento del aparato vocal.

Volumen. Varía principalmente en función de las emociones que se desean expresar. Se incrementa también con el objeto de sobrepasar el ruido ambiental y permitir la conversación. Sus variaciones son producidas por la cantidad de aire expelida por los pulmones y por la acción de los músculos de la laringe, modulándose el flujo de aire a la salida.

Adaptación. Un fonema se adapta como resultado de la influencia de los sonidos vecinos que cambian suavemente la forma del tracto vocal y la posición de las articulaciones. Al hablar rápidamente la adaptación es mas notoria debido a que la lengua no alcanza posiciones tan marcadas como cuando se habla pausadamente.

Asimilación. En casos extremos de adaptación un sonido puede cambiar de manera que llega a tener algunas de las características de los sonidos vecinos, perdiendo a la vez algunas de sus características propias.

Co-articulación. Se produce cuando dos articulaciones se mueven al mismo tiempo para producir dos fonemas distintos. La adecuada combinación de adaptación y co-articulación da por resultado una producción más natural de los sonidos.

Es posible también encontrar diferencias en la producción de los mismos fonemas causadas por factores como la entonación, la duración o la acentuación.

Acentuación. Cuando se acentúa alguna sílaba en particular, se da también un incremento en el tono de la voz. Los segmentos acentuados hacen mayor uso de la articulación, mientras que las sílabas no acentuadas son mayormente afectadas por la adaptación y la co-articulación.

Entonación. Es el resultado de cambios en la frecuencia fundamental provocando que ciertas partes de una frase tengan mayor tono que otras, proporcionando un significado adicional.

Duración. La duración se ve afectada por la acentuación, pero también por los fonemas vecinos. Interviene también en lo que se denomina *unión*, que consiste en la formación de fonemas al unir palabras.

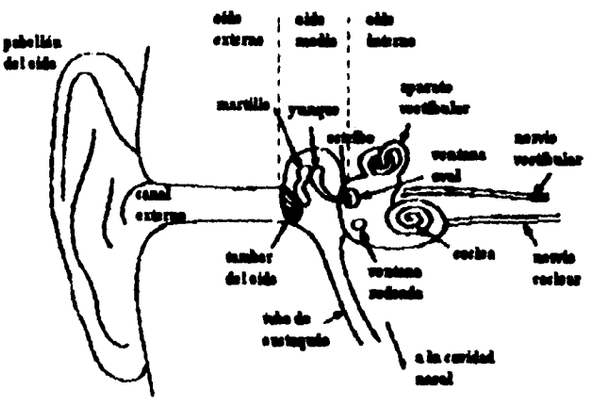

Figura 1.4: Organos del oído humano

1.2 Percepción de la voz

Como ya se ha visto, la producción del lenguaje hablado es un proceso complicado, sin embargo, este es aún la mitad de una conversación. En la percepción de la voz interviene un proceso fisiológico bastante complejo, el cual tiene sus bases en el funcionamiento del aparato auditivo, del cual se hará una breve reseña. En el reconocimiento de los sonidos, esto es, el proceso cognoscitivo mediante el cual se extrae un significado, intervienen además de la habilidad del cerebro para reconocer patrones, factores de tipo cultural y la experiencia adquirida. Es por esto que los intentos llevados a cabo en el campo del reconocimiento de la voz se caracterizan por utilizar algoritmos de elevada complejidad.

1.2.1 Fisiología del oído

El oído humano puede considerarse constituído por tres partes fundamentales - el oído externo está formado por una cubierta protectora, el oído medio encargado de ajustar los niveles de presión entre el oído externo y el interno, y el oído interno o *coclea* en el cual se encuentra el aparato sensitivo utilizado para convertir la energía extraída de la onda de presión en impulsos nerviosos que viajan al cerebro.

La oreja, la parte más externa del oído, capta y enfoca el sonido al interior del canal auditivo, el cual tiene un efecto de resonancia a la vez que incrementa la presión de sonido en un rango que va de los 2000 a los 5500 Hz, teniendo un pico de amplificación de 12 dB a los 4000 Hz.

Al terminar el canal auditivo se encuentra la membrana timpánica, a la que sigue el oído medio, el cual es el encargado de acoplar las impedancias aire-líquido, a la vez que un juego de huesecillos amplifican mecánicamente la onda de presión aproximadamente en 30 dB. El tubo de Eustaquio, que comunica el oído medio con la cavidad nasal, se encarga de igualar la presión a ambos lados de la membrana timpánica.

El oído interno, que contiene a la coclea, es el verdadero "transductor" que convierte la señal auditiva en impulsos nerviosos. Este pequeñísimo órgano en forma de caracol, posee en su interior tres cavidades que contienen un medio acuoso, el cual conduce las oscilaciones hasta la membrana basilar, que colinda con el órgano de Corti, formado por unas 15000 células vellosas.

A través de impulsos nerviosos, estas células se comunican con el cerebro por medio del nervio auditivo. Se presenta un fenómeno de "amarre de fase" entre estos impulsos nerviosos y las oscilaciones que se presentan en la membrana basilar, hasta frecuencias de 4 ó 5 kHz.

Bases fisiológicas de la percepción

La forma en que nuestro oído percibe el volumen tampoco es lineal en todo el rango de frecuencias. La mayor sensibilidad se presenta entre los 1000 y los 4000 Hz. Frecuencias fuera de este rango requieren de mayor intensidad para ser percibidas de igual manera. Existe también un umbral en el nivel de presión de sonido (SPL), a partir del cual se comienzan a distinguir los tonos. Este umbral varía en cada individuo hasta por 20 dB. En la fig. 1.5 se muestra un promedio del nivel de umbral para diferentes frecuencias.

Se grafica el SPL por encima del cero relativo (en dB) a $10^{-12}W/m^2$, distinguiéndose isocontornos a 60 y 40 phons [1]. El oído humano discrimina cambios en volumen entre los 0.5 y 2 dB, estando esta sensibilidad directamente relacionada con la velocidad a la que se presentan las ráfagas de impulsos en las neuronas auditivas.

En cuanto al tono, cuando se trata de una señal senoidal pura la frecuencia de dicha señal y la frecuencia del tono coinciden. En el caso de un tono periódico que contenga diferentes frecuencias, el tono que se distingue corresponde al de la frecuencia fundamental (la más baja). La explicación a nivel fisiológico de la percepción del tono se ha explicado por dos teorías distintas que a su vez se complementan. La teoría local identifica el tono por el lugar de máxima excitación en la membrana basilar, mientras que la teoría temporal identifica el tono por los patrones de tiempo de los impulsos neuronales.

El sistema auditivo humano posee una cualidad llamada detección residual del tono, mediante la cual le es posible distinguir un tono con solo escuchar múltiplos de la frecuencia fundamental, en ausencia de dicha frecuencia.

La selectividad de frecuencias que pueda realizar el oído depende principalmente de la sensibilidad de la membrana basilar, la cual actúa de manera similar a una serie de filtros pasobanda, donde cada filtro posee un ancho de banda denominado *banda crítica*.

Se presenta también un fenómeno llamado enmascaramiento, que consiste en el ocul-

[1] el phon es una unidad definida como unidades en dB por encima de los $10^{-12}W/m^2$ a 1 kHz

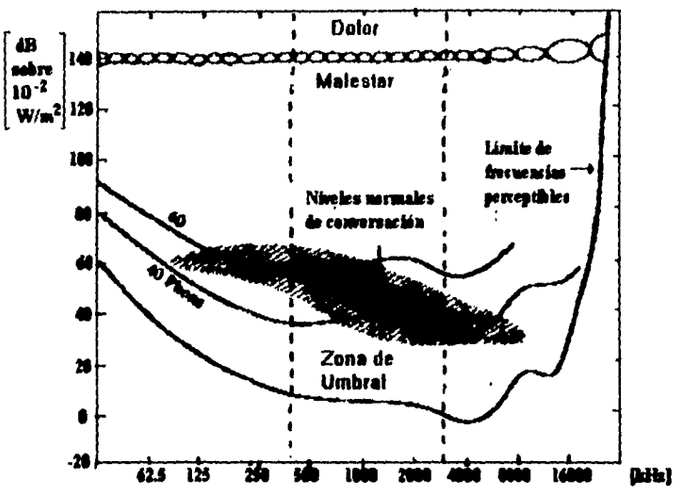

Figura 1.5: Niveles de Umbral e isocontornos de Volumen

tamiento de un sonido en presencia de otro. Su efecto es más notorio cuando los componentes de frecuencia del sonido máscara son cercanas a las del sonido mascarado. El mascaramiento se presenta también por algún sonido que se presente anterior o posteriormente al sonido ocultado, llamándose en este caso mascaramiento en adelanto o en atraso.

1.2.2 Modelos del proceso cognoscitivo de la percepción

Existen dos tendencias adoptadas al construir teorías que expliquen el proceso de la percepción - las teorías activas y las teorías pasivas.

Las teorías pasivas visualizan al receptor comparativamente pasivo en su percepción del lenguaje hablado. En dichos modelos el proceso de percepción se efectúa por el simple mapeo de las características de la señal escuchada en un conjunto de referencias fonéticas del lenguaje.

En las teorías activas se requiere que el individuo tenga un papel más interactivo en la percepción. En estos modelos se estima que el individuo receptor propone posibles construcciones fonéticas - extraídas de su representación interna y de sus propios mecanismos de construcción-, y las compara con la señal recibida. En la teoría motora propuesta por Liberman, se postula como unidad mínima de decodificación a la sílaba, incluyéndose dentro de ésta al fonema, el cual no tiene significado fuera de la sílaba dentro de la cual se encuentra. Esta teoría explica ventajosamente el por qué logran distinguirse fonemas distintos bajo el mismo concepto (como en el caso del plosivo /d/).

En el estado actual de conocimiento en este campo, no es posible concluir cuál de las dos teorías se encuentra más cercana a la verdad. Si las teorías activas aproximan mejor el verdadero mecanismo de la percepción, los intentos por encontrar características acústicas distintivas no cubren los requerimientos esperados para ser aplicados en algoritmos de decodificación eficientes.

1.3 Análisis del lenguaje hablado

La forma más primitiva de la que parte todo análisis es la de una señal continua en el tiempo y limitada en su ancho de banda. En cuanto a procesamiento digital se refiere se parte de archivos donde esta señal una vez digitalizada se ha almacenado. Los formatos más comunes para archivos de voz están formados por muestras de 12 ó 16 bits cada una con una tasa de muestreo entre los 8 y los 14 KHz. En muchas de las computadoras pequeñas es posible digitalizar la voz a 10 KHz con un convertidor A/D y grabar directamente la secuencia en disco. Para tener una idea de la capacidad de almacenamiento requerida, por ejemplo, para un minuto de voz digitalizada con las características mencionadas anteriormente se requieren 1.2 Mbytes de espacio en disco.

Usualmente se acostumbra accesar los archivos de voz en bloques de 512 ó 1024 palabras de datos, siendo el equivalente a 50 ms de señal, tiempo suficiente para acomodar la duración de un fonema.

Tradicionalmente el nivel más bajo de análisis es el nivel de forma de onda, donde se prepara la señal de manera que pueda ser procesada más fácilmente en el siguiente nivel, denominado nivel de segmento o de bloque. Es en estos dos niveles donde el procesamiento digital, especialmente en lo que a codificación se refiere es más significativo. El siguiente nivel llamado de sentencia en el cual se conjuntan ideas e información para conformar un lenguaje pertenece en mayor medida a otros campos de desarrollo. Asimismo se define un cuarto nivel -el nivel de aplicación-, el cual involucra la interacción entre los sistemas de lenguaje hablado y el ser humano.

1.3.1 Análisis a nivel de señal

El análisis a nivel de forma de onda de una secuencia de muestras se le conoce también como *análisis a nivel de señal*. En este análisis las características de la señal se estiman sobre un número de muestras utilizando un filtro promediador móvil. A continuación se enuncian algunas de las principales operaciones útiles en este nivel de análisis, así como sus aplicaciones más comunes.

Promedio de magnitud

Esta operación es tal vez la forma más sencilla de detectar la presencia de señal. Consiste simplemente en sumar los valores absolutos de un conjunto de muestras y dividir la suma entre el número de ellas:

$$M(n) = \frac{1}{N} \sum_{m=n-N+1}^{n} |x(m)| \tag{1.3}$$

Una alternativa para calcular el promedio es efectuar la convolución de la magnitud de las muestras con una función de ventaneo $w(n-m)$

$$M(n) = \frac{1}{N} \sum_{m=-\infty}^{\infty} |x(m)|w(n-m) \tag{1.4}$$

Con el objeto de minimizar los cálculos necesarios es posible obtener las muestras que se promediarán a una frecuencia más baja que la frecuencia de muestreo original. A este proceso se le llama comúnmente *down-sampling*.

El promedio de magnitud suele ser útil como parte de un control automático de ganancia (AGC) que asegure el acondicionamiento de la señal para etapas posteriores de procesamiento.

Función de densidad en amplitud

La operación enunciada anteriormente evalúa las características de la señal en un tiempo corto. Cuando se les requiere conocer en un tiempo más largo se utiliza la función de densidad en amplitud, la cual evalúa la distribución estadística de las muestras. El conocimiento de esta función permite, por ejemplo, escoger el rango correcto de voltaje a la entrada de un convertidor A/D. Esta función puede ser estimada determinando la proporción $p(x)$ de muestras x_i que están dentro del intervalo $x < x_i \leq x + dx$. Es obvio que cualquiera de estas muestras se encuentra dentro del intervalo $(-\infty, \infty)$.

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{1.5}$$

Una aproximación usada muy comúnmente es la función de densidad de Laplace $p(x)$, la cual se normaliza también para hacer que el valor r.m.s. σ_x sea igual a la unidad:

$$p(x) = \frac{1}{\sqrt{2}\sigma_x} \exp -(\frac{\sqrt{2}|x|}{\sigma_x}) \tag{1.6}$$

La función definida por la ec. 1.6 tiene la ventaja de ser sensible al tratamiento analítico, además de registrar asimetrías con respecto al cero en el eje del tiempo.

Varianza

La varianza, que se define como el cuadrado de la diferencia entre el valor de la muestra y el promedio, puede proporcionar una medida de la energía contenida en la señal. Para una señal promediada en cero, la energía en un promedio corto puede ser definida como el promedio de los cuadrados de las muestras.

$$E(n) = \frac{1}{N} \sum_{m=-\infty}^{\infty} [x(m)w(n-m)]^2 \tag{1.7}$$

Donde $w()$ es una función de ventana aplicable a las muestras. Si se define un filtro $h()$ el cual es teóricamente el cuadrado de la función de ventana, normalizándolo:

$$h(n) = \frac{1}{N} w^2() \tag{1.8}$$

la ecuación 1.7 se redefine como:

$$E_n = \sum_{m=-\infty}^{\infty} x^2(m)h(n-m) \tag{1.9}$$

Sin embargo, en la práctica es común usar una ventana normalizada al tamaño N de la misma. La energía promedio de un conjunto de muestras denota gran diferencia entre las secuencias de sonidos vocales y no vocales. Normalmente se le utiliza como detector de conversación pero en combinación con un detector de cruce por cero.

Tasa de cruce por cero

La frecuencia a la cual se presentan los cruces por cero es un indicador claro de que la señal es un sonido vocal o no vocal. Para los segmentos vocales dicha tasa es de aproximadamente 0.5 cruces/ms y para los segmentos no vocales se eleva hasta 3 cruces/ms. Usualmente se calcula mediante el algoritmo siguiente:

$$Z(n) = \frac{1}{2N} \sum_{m=-\infty}^{\infty} |sign(x(m)) - sign(x(m-1))|w(n-m) \tag{1.10}$$

donde la función $sign(x(m))$ se define como 1 si $x(m) \geq 0$ y como -1 en otro caso.

El cálculo de la función $Z(n)$ (ec. 1.10) se ve afectado por el ruido y por cualquier pequeño corrimiento de c.d. que pudiera presentarse. Este efecto indeseado puede corregirse fácilmente con la implementación de un filtro pasoaltas con frecuencia de corte cercana a los 70 Hz.

1.3.2 Análisis a nivel de segmento

Es en este nivel donde se realizan los procesos de mayor importancia utilizados por los *vocoders* [1]. Existen algunas características de la señal las cuales tienen mayor ponderación cuando el análisis se realiza a nivel de bloques. Tal es el caso de los algoritmos que obtienen la respuesta en frecuencia como la DFT y la FFT.

Como un paso previo al procesamiento de un bloque suele aplicarse a este una función de ventana, la cual tiene por objeto conformarlo de manera que la siguiente etapa de procesamiento se efectúe mas eficientemente. Una familia de ventanas cuya respuesta en frecuencia es especialmente apreciada es aquella que modela el bloque mediante una función senoidal. Casos especiales de esta familia son la ventana de Hamming y la ventana de Hann, expresadas como:

$$w(n) = a - (1 - a)\cos(\frac{2\pi n}{N-1}) \tag{1.11}$$

donde la función expresada anteriormente queda definida en el intervalo $0 \leq n \leq N-1$ y toma el valor de cero en otro caso. Para el caso de Hamming $a = 0.54$ y para Hann, $a = 0.5$.

Por lo general, se prefiere la ventana de Hamming debido a que presenta una transición mas abrupta en la magnitud a medida que se sopesan las muestras más alejadas del centro de la ventana.

Para asegurar que todas las muestras tengan el mismo peso en los cálculos, se acostumbra traslapar las ventanas adyacentes de manera que las muestras que tenían menor magnitud al evaluar un bloque sean mayormente consideradas en el siguiente.

El hecho de procesar digitalmente una señal que en su origen fue continua, acarrea necesariamente pérdidas de información, que se traducen en la aparición de lóbulos espectrales falsos o deformados, cuando se obtiene la respuesta de la señal a la frecuencia mediante algoritmos como la DFT. El uso de la ventana de Hamming disminuye en gran medida este efecto.

El multiplicar por cero todas aquellas muestras que se encuentran fuera de la ventana de longitud N, es un proceso que se conoce como *zero-padding* que se efectúa con objeto de insertar la ventana inicial dentro de otra ventana más grande de longitud N'. El uso de esta técnica tiene como consecuencia una disminución aún mayor en la magnitud de los lóbulos indeseables.

Una característica poco deseada en el espectro de la señal de la voz es la atenuación que crece a frecuencias altas. Para lograr un espectro uniforme se da una preamplificación a estas frecuencias utilizando un filtro de preénfasis, el cual tiene una función de transferencia H_{preenf} con un cero de baja frecuencia:

[1] La palabra vocoder proviene del anglicismo voice-coding, es decir, codificación de voz.

$$H_{preenf}(z) = 1 - az^{-1} \tag{1.12}$$

Una vez realizado el procesamiento requerido, la señal se convierte de nuevo a una señal auditiva canalizándola a través de un filtro de deénfasis que restaura el espectro original. El filtro de deénfasis tiene una función de transferencia dada por:

$$H_{deenf}(z) = \frac{1}{1 - az^{-1}} \tag{1.13}$$

usualmente el coeficiente a toma valores en el intervalo $0.95 < a < 0.98$.

Un procedimiento muy comúnmente usado para encontrar la periodicidad en segmentos vocales es el que hace uso de la autocovarianza y de la autocorrelación. La *autocovarianza* es función del retardo k (muestras), y consiste en el promedio de los productos de una muestra $s(n)$ con una muestra distante $s(n + k)$. La función se evalúa dentro de un período N de muestras capaz de contener un período completo de la señal. La autocovarianza se define como la función $C(k)$ tal que:

$$C(k) = \frac{1}{N} \sum_{n=0}^{N-1} s(n)s(n + k) \tag{1.14}$$

Para un retardo $k = 0$, la función alcanza un máximo como por lo que suele normalizarse dividiéndola entre dicho máximo $C(0)$, dando lugar a la función de *autocorrelación* $R(k) = C(k)/C(0)$.

En señales de tipo vocal la función de autocorrelación generalmente tiene máximos cercanos a $+1$, mientras que los sonidos no vocales usualmente no sobrepasan valores mayores a 0.2. La frecuencia fundamental de un sonido vocal oscila entre los 50 y los 500 Hz, por lo que son necesarias entre 16 y 160 muestras para que sea posible encontrar un pico de la función en este intervalo.

Como resultado del teorema de Wiener-Khintchine es posible conocer el espectro de potencia de la señal aplicando la transformada de Fourier a la función de autocorrelación, lo que nos da por resultado la *función espectral de potencia*.

1.4 Técnicas espectrales aplicadas al análisis de la voz

Como ya se ha discutido, la transformada de Fourier es una herramienta de gran utilidad y de uso generalizado para diversas aplicaciones. Sin embargo, para el caso del análisis automático de la voz existen técnicas de mayor eficiencia, como la Predicción Lineal y el Análisis Cepstral. Ambas técnicas se aplican cuando se analiza la señal vocal bajo el *modelo excitación/filtro* (figura 1.6), el cual se describe con detalle en el capítulo 4 de este trabajo. Bajo este esquema, los pulsos que caracterizan el movimiento de la laringe se modelan como un tren de pulsos a una frecuencia fundamental (en el caso de sonidos *vocales*). En el caso de los sonidos *no vocales* como es el caso de los fricativos /th/, /s/ y /f/, la señal se asemeja mucho al ruido blanco. En el caso de algunos fricativos producidos en el fondo de la garganta como /h/ y /sh/ su espectro suele semejarse al de un sonido vocal con frecuencias formantes bien definidas. Una vez obtenida la excitación (*filtro de análisis*) debe definirse también un filtro que modele espectralmente la excitación, de manera que puedan ser delineadas las frecuencias formantes para lograr que la señal obtenida sintéticamente tenga un espectro semejante a la señal original.

En este contexto, las tareas principales para realizar el modelado resultan ser

- Determinar si el segmento en proceso es vocal o no vocal.
- Si el segmento es vocal, determinar la frecuencia fundamental.
- Determinar los parámetros del filtro modelador.

Para obtener de la señal eléctrica la información anterior, existen dos corrientes de análisis, que se mencionan a continuación.

1.4.1 Predicción Lineal

La predicción lineal se ha convertido en una de las técnicas más populares en el análisis del lenguaje hablado al enfocársele desde el modelo excitación/filtro. Existe una amplia variedad de aplicaciones de ella, su fácil entendimiento así como la eficiencia con la que se le puede implementar le hacen una herramienta muy atractiva. El modelo general de la predicción lineal (LP) tiene la forma:

$$s_n = \sum_{k=1}^{p} a_k s_{n-k} + G \sum_{l=0}^{q} b_l u_{n-l}, b_0 = 1 \tag{1.15}$$

donde s_n representa las salidas del sistema, u_n son las entradas del sistema, y los coeficentes de cada una de ellas son $a_k; K = 1, ..., p$ y $b_l; l = 1, ..., q$, además del factor de ganancia G. La idea entonces es que conociendo las últimas p salidas así como las últimas q entradas, adecuando los coeficientes a_k y b_l es posible predecir la siguiente salida del modelo mediante una combinación lineal de las salidas y las entradas. Haciendo un reacomodo de la ecuación 1.15 y transformando al dominio de z, la función de transferencia del sistema se define:

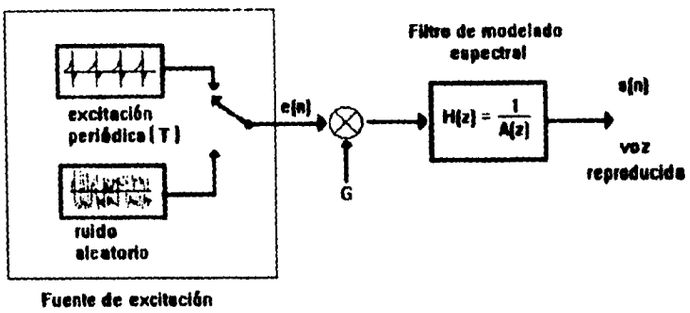

Figura 1.6: Modelo generalizado excitación/filtro

$$H(z) = \frac{S(z)}{U(z)} = G\frac{1 + \sum_{l=1}^{q} b_l z^{-1}}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{1.16}$$

o bien

$$H(z) = \frac{B(z)}{A(z)} \tag{1.17}$$

donde los polos del sistema son las raíces de $A(z)$ y los ceros las raíces de $B(z)$. Retomando la idea de la ecuación 1.15, en los sistemas de procesamiento de voz por lo general la entrada u_n no siempre se encuentra disponible, por lo que el algoritmo se limita entonces a calcular la posible salida en base a las previas salidas obtenidas, de donde la ecuación 1.15 se redefine como:

$$\tilde{s}_n = \sum_{k=1}^{p} a_k s_{n-k} \tag{1.18}$$

El error de predicción, o residuo, suele ser un parámetro útil en el análisis automático de

la voz, y se define como la diferencia que existe entre el valor predecido por el modelo y el valor real muestreado:

$$e_n = s_n - \sum_{k=1}^{p} a_k s_{n-k} \tag{1.19}$$

En una implantación eficente, se busca reducir el error cuadrático total, el cual está dado por la suma de los cuadrados de los últimos n errores:

$$E = \sum_{n} e_n^2 \tag{1.20}$$

Existen a partir de este punto dos tendencias encaminadas a optimizar los coeficientes a_k: el método de autocorrelación y el método de covarianza. No es el objeto de este capítulo profundizar en dichos métodos, baste decir que para un segmento de voz en el cual las características se consideran relativamente estacionarias, la secuencia de muestras de voz representadas por s_n puede ser sustituida por p coeficientes a_k de un filtro predictor y por la secuencia e_n llamada señal de residuo.

La señal de residuo en segmentos vocales es por lo general de magnitud pequeña excepto por discontinuidades periódicas que coinciden precisamente con el inicio de los pulsos de la laringe. Para los segmentos no vocales la señal de residuo (residuo LP) es semejante al ruido blanco con un nivel pequeño y uniforme. Los coeficientes del filtro predictor en un segmento de condiciones estacionarias cambian a un ritmo relativamente lento. De hecho esta característica de modelo excitación/filtro es aprovechada para economizar el número necesario de bits para codificar dicho segmento. La tasa de actualización de los coeficientes se le llama *tasa de segmentación* la cual se encuentra en el rango de 10 a 25 ms, que para voz muestreada a 8 kHz equivale a 80 a 200 muestras. Por lo general, el número de muestras utilizadas por el predictor lineal está entre 10 y 20.

1.4.2 Análisis Cepstral

Un método alternativo para separar la frecuencia fundamental emitida por la laringe de su envolvente espectral es el análisis Cepstral. Haciendo referencia al aparato vocal humano como un sistema con una función de transferencia en el tiempo denotada por $v(t)$ la cual modela el tracto vocal, con una entrada $e(t)$ la cual se identifica con la excitación, se tiene a la salida del sistema una señal $s(t)$ la cual representa la voz a la salida del tracto vocal. (Figura 1.7).

En el dominio del tiempo, la señal $s(t)$ está dada por la convolución de $e(t)$ con $v(t)$:

$$s(t) = e(t) * v(t) \tag{1.21}$$

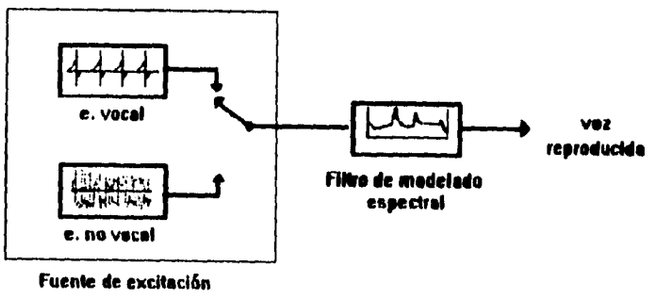

Figura 1.7: Modelo excitación/filtro en el dominio del tiempo

El análisis Cepstral es un *método de deconvolución* pues su objetivo es separar la excitación $e(t)$ de la envolvente $v(t)$. Para ello, se recurre a lo que se denomina *filtrado homomórfico* mediante el cual se aplica una serie de transformaciones para realizar a cabo dicha separación. En el dominio de la frecuencia, la salida $s(w)$ del sistema está dada por:

$$S(w) = E(w)V(w) \tag{1.22}$$

donde $S(w)$, $E(w)$ y $V(w)$ son las respectivas transformadas de Fourier de las señales ya conocidas. Ahora bien, ya en el dominio de la frecuencia es relativamente sencillo separar ambas señales aplicando la función logaritmo, de manera que:

$$log[S(w)] = log[E(w)] + log[V(w)] \tag{1.23}$$

Al graficar la función $log[S(w)]$ es posible identificar una serie de lóbulos igualmente espaciados, montados sobre una curva que delinea la envolvente espectral. El espaciamiento entre dicho lóbulos, dado en unidades de frecuencia, es el inverso de la componente fundamental que se busca (Figura 1.8a). El último paso en el procedimiento es regresar al dominio

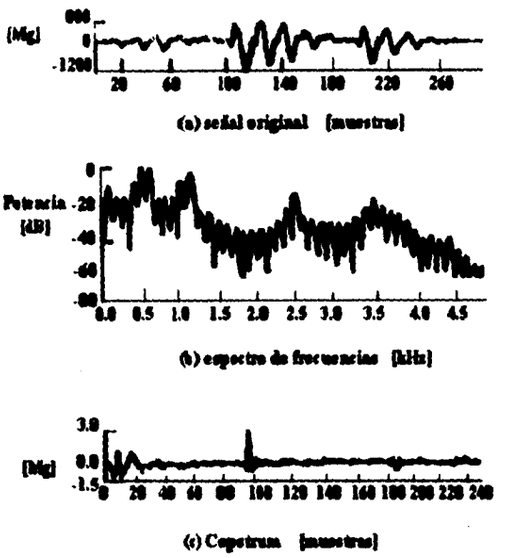

Figura 1.8: Aplicación de la función Cepstrum

del tiempo aplicando la transformada inversa de Fourier. En esta última etapa se realiza la separación completa de la excitación y su envolvente (Fig. 1.8b). La función obtenida tiene unidades inversas a la frecuencia, es decir segundos, por lo que a la variable independiente se le llama *qüefrencia* y la función en sí se le denomina *Cepstrum*. No es de sorprender entonces que al proceso de separación realizado se le llame en ocasiones "liftrado".

En una gráfica de Cepstrum para un segmento vocal, se destacan en su primera parte una serie de lóbulos que representan las componentes de baja frecuencia de la envolvente espectral, así como un pico angosto y pronunciado en la parte derecha de la gráfica que identifica la frecuencia fundamental o *pitch* (Figura 1.8b).

1.5 Herramientas de Análisis

El lenguaje hablado, al ser tratado como una señal eléctrica es susceptible de ser analizado mediante técnicas que han evolucionado de acuerdo a los avances en electrónica y los métodos computacionales.

1.5.1 Análisis por métodos analógicos

Previo a la aparición de las herramientas de procesamiento digital de señales, el análisis de la señal de habla se llevaba a cabo mediante técnicas puramente analógicas. En este sentido, la herramienta que continúa siendo uno de los pilares para el análisis de la señal eléctrica es el osciloscopio, el cual permite observar el desempeño de la forma de onda en el dominio del tiempo. Con un enfoque más especializado, han aparecido otras variantes en la instrumentación que dieron un gran impulso al desarrollo de la ciencia del análisis vocal.

Espectrógrafo

Aunque es una herramienta que en la actualidad ha sido francamente superada por las técnicas digitales, es conveniente hacer mención de ella como un antecedente importante. Su funcionamiento se basa en los mismos principios que utiliza un analizador de espectros analógico, pero su banda de frecuencias de análisis se restringe a la banda ocupada por el espectro de la voz (4 khz). Básicamente está constituido por filtros sintonizables dentro de la banda de frecuencias a ser exploradas. Un elemento de control asigna la operación a cada filtro de manera que sea "barrida" la banda deseada. La respuesta en magnitud de cada filtro es entonces traducida en una gráfica que muestra con bastante aproximación el espectro de frecuencias de la señal en análisis. Su uso forzaba al experimentador a repetir constantemente la reproducción de cada segmento de señal de manera que se obtuviesen gráficos más confiables.

Laringógrafo

El laringógrafo es un dispositivo que aporta información del movimiento de las cuerdas vocales. Mediante electrodos conectados a la garganta, se lleva un registro de los cambios en la conductancia a lo largo de la laringe. Dicha conductancia varía de acuerdo a las oscilaciones de las cuerdas vocales, incrementándose cuando se estrechan y decrementándose cuando se ensanchan. En la figura 1.9 se muestra la curva obtenida por un laringógrafo para un segmento vocal emitido por una mujer. En dicha figura puede apreciarse que el estrechamiento de la laringe se asocia con un rápido incremento en la conductancia, así como las aperturas de la laringe se identifican por un gradual decremento en magnitud. Se observa también una tercera gráfica donde aparece el residuo LP para este segmento, donde sobresalen incrementos en el error cuando los pulsos de la laringe tienen un máximo - es decir, cuando el estrechamiento es máximo también. Cabe hacer notar que la relación entre la gráfica obtenida con el laringógrafo no sigue con precisión la forma de onda de los pulsos glotales, sino que su relación se identifica únicamente en cuanto a su periodicidad. Lo anterior se debe a que con este instrumento se obtiene la curva de estrechamientos de la laringe, mientras que la forma de onda de los pulsos se modela por la dinámica del flujo de aire a través del tracto vocal.

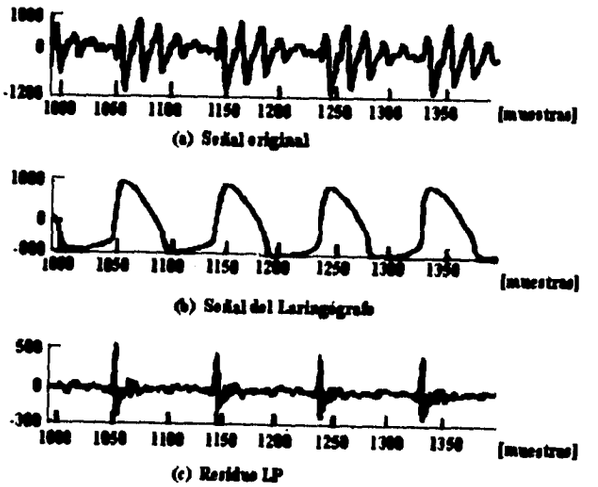

Figura 1.9: Formas de onda producidas por un laringógrafo

1.5.2 Análisis por métodos de PDS

La aparición de los métodos de análisis de las señales discretas, aplicados al análisis del lenguaje hablado, marcó un punto de despegue en el desarrollo de las ciencias del lenguaje. La aplicación de algoritmos de análisis en la frecuencia tales como DFT, FFT ó DCT se ha generalizado en la última década, al grado de formar parte integral de programas especializados en el análisis de señales como son MATLAB ó Mathematica. Una de las herramientas de análisis más populares son los espectrogramas, las cuales son la aplicación directa de métodos de respuesta en frecuencia como la FFT en el espacio del tiempo, generando gráficas frecuencia-tiempo que permiten observar la evolución espectral de la señal en un intervalo de tiempo dado. Comúnmente el tiempo (o el número de muestras) aparece como la abcisa, mientras que como ordenada se genera un patrón de color o de tonos de gris que representa las diferentes componentes espectrales (a un tono más elevado corresponde una magnitud mayor para una frecuencia dada). Un espectrograma puede también ser representado por una superficie, en tal caso el eje perpendicular al plano frecuencia-tiempo corresponde a la magnitud instantánea de una componente espectral. En este caso la gama de colores que se asignaba a las diferentes magnitudes corresponde a la altura del relieve en el espectrograma de superficie.

Espectrograma de banda ancha

En el contexto de los espectrogramas generados utilizando transformadas de Fourier, el término "banda ancha" implica que se utilizaron relativamente pocas muestras en la obten-

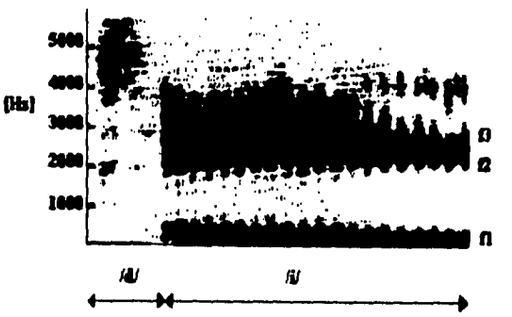

Figura 1.10: Ejemplo de espectrograma de banda ancha

ción de cada espectro instantáneo; ello acarrea una disminución en la resolución en el eje de la frecuencia, dando mayor preferencia a los detalles en el dominio del tiempo. Si se examina el ejemplo que se presenta en la figura 1.10, pueden apreciarse una serie de estrías verticales a través de los tonos oscuros de las frecuencias formantes. Estas estrías son características de los segmentos de sonidos vocales, y representan las excitaciones individuales del tracto vocal cuando éste se cierra. Midiendo la distancia entre estas estrías puede hacerse un cálculo aproximado del periodo fundamental del tracto vocal (*pitch*).

Espectrograma de banda angosta

Una alternativa para la representación espectral de la señal vocal puede ser generada aplicando la transformada de Fourier en un segmento más amplio de muestras. Los espectrogramas producidos de esta manera se denominan de "banda ancha" y contienen información más detallada en el dominio de la frecuencia a costa de un decremento en la resolución en el dominio del tiempo. En el ejemplo mostrado en la figura 1.11 se hace mucho mas evidente la presencia de las frecuencias formantes en los sonidos vocales. Así como en los espectrogramas de banda ancha era posible hacer un cálculo de la oscilación fundamental midiendo la distancia entre las estrías, en los espectrogramas de banda angosta es posible identificar las diferentes armónicas que conforman a una vocal y así hacer una aproximación de la frecuencia fundamental.

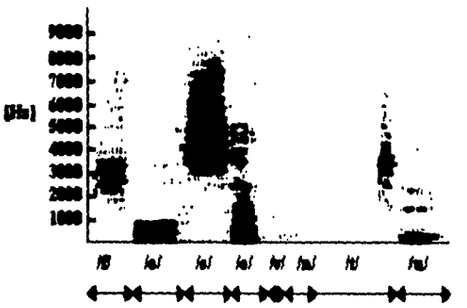

Figura 1.11: Ejemplo de espectrograma de banda angosta

Conclusión

Una vista panorámica del estado actual de las ciencias y técnicas que en la actualidad intentan explicar, modelar y reproducir el fenómeno del lenguaje hablado, aportan las bases necesarias para atacar un problema específico en este campo, como lo son las técnicas de compresión, las cuales fundamentan sus algoritmos en las distintas aproximaciones que se han argumentado.

Referencias

[1] Chris Rowden: *Speech Processing*, Essex series in telecommunications and information systems, cap. 1, 2 y 3, Mc Graw Hill, 1991.

[2] F.A. Westall, S.F.A. Ip: *Digital Signal Processing in Telecommunications*, cap. 1, 10 y 11, Gran Bretaña, Chapman & Hall, B.T. Telecommunications, 1993.

[3] Panos E. Papamichalis: *Practical Approaches to Speech Coding*, cap. 1 y 4, New Jersey, Prentice-Hall, 1987.

[4] Burrus,C.S. & Parks, T.W.: *DFT / FFT and Convolution*, E.U.A., John Wiley and Sons, 1985.

[5] Parks, T.W. & Burrus, C.S.: *Digital Filter Design*, E.U.A., John Wiley and Sons, 1987.

Capítulo 2

Alternativas en la codificación de la voz

Los grandes desarrollos en tecnologías de codificación han incrementado la demanda de los servicios de telecomunicaciones tales como las aplicaciones en aeronáutica, satélites móviles, larga distancia y telefonía digital, entre otros; y debido a su gran demanda, para poder tener disponibles los sistemas de comunicaciones en ciertos mercados, se tiene un gran crecimiento principalmente en tecnologías de codificación, como en el caso de aquellas que incorporan compresión de la voz.

Esta búsqueda es intensa ya que se busca una muy alta calidad en la voz a velocidades que hasta hace pocos años se consideraban casi imposibles, teniéndose una mayor aplicación en redes y servicios que introduzcan esta tecnología.

2.1 Codificación de la voz en telecomunicaciones

Los codificadores se utilizan para convertir una señal de voz analógica a una de tipo digital para que se tenga una transmisión eficiente; para un almacenamiento digital así como para diseñar una función complementaria de conversión de una señal digital recibida hacia su forma analógica.

La principal tarea en la codificación es reducir la velocidad de transmisión. Por ejemplo, la calidad de codificación en alta fidelidad (Hi-Fi) de una señal de audio de 20 KHz requiere de una tasa de muestreo de 40 muestras/s y una cuantización a 16 bits, lográndose una velocidad de transmisión de 640 kbit/s (40 x 16 kbit/s), mientras que para una señal telefónica de 3.4 kHz se requiere de una reducción a 96 kbit/s muestreando a 8 KHz y 12 bits de cuantización lineal, sin embargo es posible reducir la velocidad de transmisión lineal de 12 bits por una cuantización no lineal de 8 bits (esquema de ley μ o ley A). Para una señal telefónica, en la mayor parte de las redes públicas, la velocidad de transmisión estándar es de 64 kbit/s . Estos son algunos ejemplos donde se utiliza una velocidad mas baja de transmisión, lo cual se ha logrado con el uso de las nuevas tecnologías digitales.

El uso más importante de la codificación de la voz a baja velocidad se ha dado en la creación de los nuevos *servicios móviles* ya que el uso de los espectros es más eficiente; esta

área se está desarrollando en gran medida dado que tanto para las comunicaciones móviles como las inalámbricas se realizan accesos a servicios de comunicación externos contactándose instantáneamente sin tener que estar en alguna ubicación particular.

Como ejemplos se pueden citar los siguientes: El servicio de teléfonos aéreos instalados por Bell Telephone, en 1990, con una codificación de voz de 9.6 kbit/s; el nuevo sistema DMR [1] de telefonía celular el cual trabaja a 13 kbit/s así como las redes de comunicación personal (PCN) en el Reino Unido donde se trabajarán también 13 kbit/s. Una nueva generación de teléfonos inalámbricos (CT2) empleará 32 kbit/s el cual aumentará su capacidad en el sistema así como su calidad y seguridad al servicio que darán.

Una de los principales problemas que existen en los servicios de enlaces de radio digital son los errores de transmisión, por lo que se usa una baja velocidad en estos servicios y así optimizar el ancho de banda. La capacidad de los sistemas de telefonía celular digital está limitado por la interferencia de células vecinas que usan la misma frecuencia ocasionando errores digitales en la señal recibida. Pero la habilidad de los codificadores de la voz para dar una buena calidad se ve poco afectada por esos errores por lo que las radio frecuencias se podrán reutilizar mucho más eficientemente que en sistemas analógicos, por esta razón el incremento en la capacidad del sistema es tan importante como la reducción de la velocidad.

Para poder multiplexar varias conversaciones en un canal se utilizan los codificadores lográndose reducir el costo además de ser un factor importante para lograr una baja velocidad. Por ejemplo se puede reemplazar un canal PCM de 64 kbit/s por ocho canales con voz codificada a 8 kbit/s. El multiplexaje se usa comúnmente en circuitos de comunicación privada así como en redes de trabajo públicas que son muy costosas (como los enlaces por cable submarino).

En resumen, la codificación de la voz en telecomunicaciones es una tecnología que se está desarrollando ampliamente aplicada a sistemas tanto públicos como privados; y como ya se mencionó anteriormente la calidad de la voz juega un papel muy importante. A continuación se mencionarán los requerimientos así como los factores que intervienen para una codificación eficiente.

2.1.1 Calidad de la voz

Requerimientos

En los codificadores de voz utilizados en sistemas de telecomunicaciones públicos generalmente, se requiere de una buena calidad para tener la misma calidad de voz que en llamadas de larga distancia en redes telefónicas públicas, conocida también como *"toll quality"*. También se puede mencionar que los teléfonos inalámbricos, los sistemas de transmisión como los codificadores, entre otros utilizan una calidad alta.

[1] DMR:Digital Mobile Radio

Factores que intervienen en la calidad de la voz

- Errores en la transmisión digital
- Sensibilidad a diversos niveles de potencia
- Conexiones tandem
- Retardo

Errores en la transmisión digital

Los sistemas de transmisión que usan enlaces de radio pueden tener errores, pero la codificación de canales provee la detección y corrección de éstos. Sin embargo estos sistemas dejarán un error promedio de velocidad a la entrada del decodificador de la voz, típicamente entre 1 en 50 y 1 en 1000. Es por esto que el decodificador de voz se debe diseñar de manera robusta para mantener la calidad de la voz en donde la entrada del codificación fue alterada.

Sensibilidad a diversos niveles de potencia

Los codificadores de voz en servicios públicos deben operar en rangos de voz humana, teniendo desde los hombres adultos hasta los niños, con diferencias en los niveles de potencia de por lo menos 25 dB, con voz distorsionada, o con ruido el cual puede ser alto en algunas aplicaciones o cuando se encuentra más de una persona hablando a la vez.

Conexiones tandem

Estas conexiones de codificadores de voz se darán en llamadas de usuario a usuario para sistemas DMR. Para algunos codificadores de voz, en las conexiones tandem con otros codificadores (iguales o de otro tipo), se presenta un poco de distorsión que se da por el primer codificador que precede a un gran incremento en la distorsión introducida por el segundo codificador.

Retardo

Este puede ser medido al conectar a la salida del codificador directamente a la entrada del decodificador y midiendo el tiempo que tarda desde que entra la voz al codificador hasta que se tiene la versión reconstruida de la misma en el decodificador.

El retardo en algunos casos, como el que ocurre en el almacenamiento de voz, no es tan significativo; pero en aplicaciones de transmisión es importante por dos razones:

- El retardo excesivo dificulta la conversación -más de 400 ms- .
- Y si hay alguna fuente de eco, ya sea eléctrica o acústica, en la ruta de transmisión, entonces la propagación del eco puede ser muy grande.

Ambas consideraciones pueden limitar la codificación de retardo y este dependerá de otros retardos producidos en las conexiones involucradas, en alguna aplicación en particular. Así por ejemplo, si un codificador se conecta a un equipo de la red pública, entonces el retardo debe limitarse a 5 ms, a menos que se cuente con dispositivos de control de eco.

2.2 Técnicas de codificación de la voz

Estas se pueden clasificar en tres tipos:

1. Codificación por forma de onda
2. Codificación híbrida
3. Codificación paramétrica

La codificación por forma de onda más básica no hace uso de alguna teoría que modele el proceso de la producción de la voz en la codificación de la señal de entrada. En estos codificadores se reproduce la forma de onda original tan fielmente como sea posible, utilizándose principalmente en señales no vocales sin dificultad.

En los codificadores paramétricos, conocidos también como *Vocoders*, no se reproduce la forma de onda original pero se toma un conjunto de parámetros en la codificación la cual se puede usar para controlar un modelo de reproducción de la voz en la decodificación. Este conjunto es relativamente pequeño y puede ser cuantizado eficientemente para la transmisión operando a una tasa muy baja.

Los codificadores híbridos combinan las codificaciones antes mencionadas para generar esquemas de codificación más sofisticados y poder manejar una buena calidad así como una eficiente codificación. Lo que tienen en común con los vocoders, es que ambos usan un modelo de producción de la voz; sin embargo, no tiene la representación tan simple de la señal de excitación de los vocoders. Estos codificadores trabajan a una velocidad baja que está entre los codificacores de forma de onda y los vocoders.

2.2.1 Codificación de forma de onda

La técnica más simple y por lo tanto la más conocida es la PCM [2], la cual trabaja con una cuantización no uniforme de 8 bits (ley A ó ley μ) con un muestreo de 8 KHz, obteniéndose una muy buena calidad de voz a 64 kbit/s.

Una alternativa es la codificación diferencial en la cual se codifican las diferencias entre muestras adyacentes y no la magnitud de las muestras en sí, conocida como DPCM [3]. Así también se tiene el ADPCM [4], para adaptar el cuantizador del tamaño del escalón de un DPCM de acuerdo a la potencia promedio de la voz.

[2]PCM: Pulse Code Modulation
[3]DPCM: Differential Pulse Code Modulation
[4]ADPCM: Adaptive Differential Pulse Code Modulation

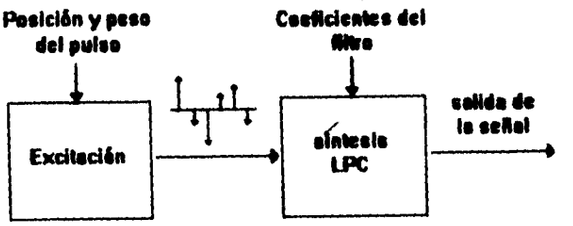

Figura 2.1: Modelo de producción de la voz

2.2.2 Codificación Paramétrica *Vocoders*

En la figura 2.1 se muestra la operación de los vocoders el cual se basa en un modelo de producción de la voz.

La sección del tracto vocal se puede representar en tres formas diferentes, por la amplitud del espectro a alguna frecuencia específica (vocoder de canal); por los picos en el espectro (vocoder de formantes); o bien, por los coeficientes de la codificación lineal predictiva (vocoder LPC).

El análisis que se efectúa en el codificador lineal predictivo puede ser usado para obtener los coeficientes de un filtro digital lineal variable en el tiempo. Para codificar la voz los parámetros del filtro se actualizan en intervalos que varían entre 20 ms -siendo el mas común- y 30 ms, este último se usa en codificadores que operan por debajo y máximo a los 4.8 kbit/s. El análisis del LPC es para extraer el conjunto de parámetros de la señal de voz la cual especifica la función de transferencia del filtro dando la mejor aproximación espectral para la voz codificada. Los filtros que sólo contienen polos de orden p entre 10 y 16, se usan para modelar el espectro del tracto vocal.

Los vocoder LPC usualmente operan alrededor de 2.4 kbit/s y dan una calidad que no se acepta en aplicaciones de redes públicas; aunque es aceptable para las comunicaciones militares.

La excitación se podría modelar en los vocoders como una serie de pulsos unipolares

espaciados al tono de frecuencia para los segmentos vocales y ruido para los no vocales. Pero esta excitación no es buena porque la voz no cae siempre dentro de las dos categorías vocales y no vocales y además porque el tono no es constante en la del tipo vocal debido a que está sujeto a pequeñas variaciones.

Los vocoders se diseñan para señales no vocales así como para sistemas donde no importe demasiado los niveles altos de ruido.

2.2.3 Codificación Híbrida

Esta codificación, como ya se mencionó, combina características de los dos tipos de codificación: por forma de onda y paramétrica, lo que da por resultado una buena calidad y una codificación eficiente.

El codificador híbrido más común utiliza un proceso de síntesis LPC en el instante mismo de la codificación (análisis por síntesis). Una forma de entender el funcionamiento de este codificador es obteniendo un filtro LPC inverso (sólo ceros). Cuando una señal no es vocal y se filtra a través del filtro LPC inverso se obtiene una señal que parece ruido conocida como señal de residuo. Si la señal es vocal entonces la de residuo tendrá una forma de onda que contendrá pulsos periódicos al tono de frecuencia y si es una señal no vocal tendrá una estructura casi irreconocible.

Si la señal de residuo no es cuantizada se filtra a través de un modelo de síntesis LPC (todos polos) entonces la señal original se reproduce a la salida del sintetizador (Ver figura 2.2).

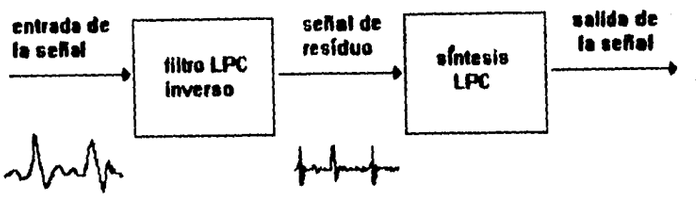

Figura 2.2: Formación de la señal de residuo

2.3 Codificadores híbridos

2.3.1 Codificador APC (Adaptive predictive coding) y Codificador RELP (Residual excited linear predictive coding)

La codificación conocida como APC (Adaptive Predictive Coding) estudia a la señal de residuo como una forma de onda cuantizándola directamente. Sin embargo, como la señal de residuo tiene ruido casi no existe correlación entre las muestras adyacentes así que ninguna codificación es efectiva y por esto se requiere de alta velocidad.

Una variante de la codificación APC es la RELP (Residual Excited Linear Predictive Coding). Para el RELP la señal de residuo se filtra por un paso-bajas y su banda límite es de aproximadamente 1 KHz; esta señal con banda limitada se vuelve a muestrear a una velocidad más baja. Como esta velocidad se redujo, habrá menos muestras para transmitir y se requieren menos bits para cuantizar la señal de residuo.

Para los APC y los RELP la velocidad se reduce hasta 8 kbit/s por lo que el número de bits disponible para representar la señal de residuo es insuficiente además de que la calidad se deteriora rápidamente. A una velocidad menor de 8 kbit/s se requiere de una representación de la excitación muy compleja para que la codificación de la voz sea eficiente y efectiva. Esta técnica es conocida como *sustitución de la señal de residuo*. Para reemplazar a la señal de residuo, se requiere que actúe como la excitación del filtro de síntesis LPC.

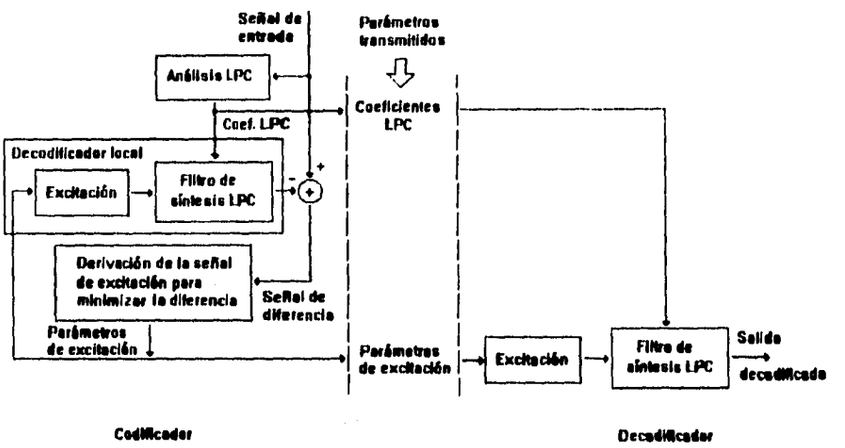

Figura 2.3: Esquema de la codificación híbrida "análisis por síntesis"

Esta señal que sustituirá a la señal de residuo debe cumplir con:

- Producir una salida del filtro de síntesis LPC similar a la que se obtiene usando una señal de residuo no cuantizada.
- Debe poder representarse con mucho menos bits que los requeridos por la señal de residuo.

La técnica que cumple con las especificaciones anteriores es la de *análisis por síntesis*. Para un modelo de síntesis que utiliza LPC, en un vocoder se usa sólo para la decodificación, sin embargo el que usa análisis por síntesis se emplea tanto para codificar como para decodificar.

Los coeficientes del modelo de síntesis LPC se obtienen directamente de la señal de entrada y la señal de excitación se obtiene de la técnica antes mencionada, la cual está representada en la figura 2.3.

En análisis por síntesis, como su nombre lo indica, utiliza una "síntesis" como parte integral del proceso de análisis. El objetivo de este proceso es obtener una señal de excitación, así que la diferencia entre la entrada y las señales de excitación será mínima con este criterio. Para poder diferenciar la señal que se va a obtener de la señal de síntesis tiene que generarse en el codificador LPC una combinación "síntesis-excitación". Los coeficientes del LPC y los

de la señal de excitación derivados de esta técnica son transmitidos como valores cuantizados de la decodificación.

2.3.2 Codificador MPLPC (Multipulse LPC)

El MPLPC (Multipulse LPC) es una técnica de codificación que utiliza una sustitución de la señal de excitación, conocida como excitación multipulsos. Para excitar el filtro se utiliza una serie de pulsos espaciados de manera no uniforme con diferentes amplitudes. No hay diferencia entre las señales vocales o no vocales por lo que se utiliza el mismo tipo de forma de onda de excitación para todos los segmentos de la voz. Para que se obtenga una buena calidad, varios pulsos se requieren por período de tono. Sin embargo, como todos los pulsos y amplitudes se deben transmitir, se tiene que decidir entre tener una mayor calidad de la señal a costa de una cantidad mayor de bits a ser transmitidos. Es indispensable para el diseño del codificador la derivación de las posiciones de los pulsos apropiados así como sus amplitudes.

Un codificador MPLPC está esquematizado en la figura 2.4, donde el método de derivación de la excitación multipulsos involucra un procedimiento de análisis por síntesis, por ejemplo, la entrada y la voz sintetizada son comparadas y la excitación para su procedimiento de análisis requiere de la partición de la señal de entrada en bloques más pequeños (20-80 muestras) y buscar, para las posiciones de los pulsos y amplitudes, el mínimo error, entre la entrada y la voz sintetizada.

Una solución para encontrar las posiciones óptimas y amplitudes es un problema no lineal, por lo que se vuelve muy complejo. Se puede pensar en dividir en dos partes y así analizar cada posible combinación de posiciones de pulsos. Dadas las posiciones de los pulsos, las amplitudes se minimizan y el error se puede encontrar fácilmente.

El número de combinaciones posible para unos pocos pulsos por bloque es alto y resulta una carga excesiva de cálculos. Algunos métodos sub-óptimos se han desarrollado para buscar las posiciones y amplitudes al mismo tiempo. Estos métodos reducen el proceso de minimización del error . Dadas la posición y la amplitud del pulso, en la función de croscorrelación se puede actualizar y se procede a la siguente búsqueda.

El codificador MPLPC puede predecir los términos de segmento largo por lo que se requerirán menos pulsos por período de tono para obtener la misma calidad de voz. Los parámetros para la predicción de estos términos puede ser cuantizada relativamente con menos bits. Estos codificadores pueden operar bien entre 8 y 16 kbit/s.

Aplicaciones

Los codificadores desarrollados en Bell Telephone (BT), por ejemplo operan a 9.6 kbit/s, donde 8.9 kbit/s se usan para la codificación de la voz y 0.7 kbit/s para la corrección-detección del error. Este codificador usa tamaños de trama de 20 ms para el análisis del LPC y cada 20 ms se generan nuevos coeficientes LPC; los parámetros de segmento largo se

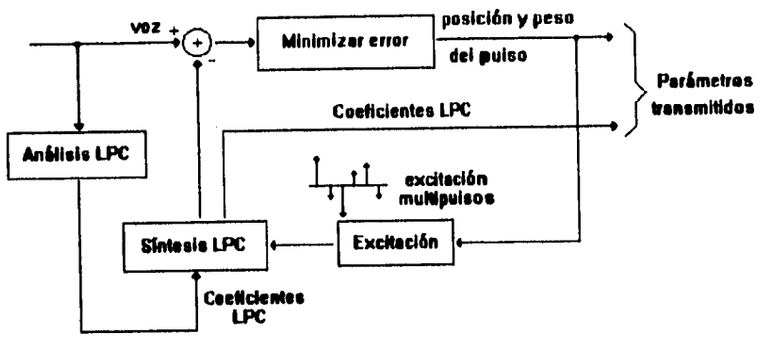

Figura 2.4: Decodificador multipulsos

actualizan también cada 20 ms. Basándose en el procedimiento de análisis por síntesis, el de la excitación usa 4 ms de sub-trama con tres pulsos de excitación calculados. Este último se seleccionó para aplicaciones aeronáuticas.

2.3.3 Codificador RPE (Regular Pulse Excited) LPC

Este codificador es una variante del MPLPC donde los pulsos de la señal de excitación están uniformemente espaciados, por lo general, cada 3 o 5 muestras con pulsos espaciados cada 4 posiciones. El proceso de codificación busca el mejor vector y amplitud del pulso apropiados y hay cuatro posibles vectores de excitación. Para ejemplificarlo se tiene la figura 2.5.

Aplicaciones

Los codificadores que trabajan a 13 kbit/s, adoptados por el servicio de telefonía digital de Europa, utilizan algoritmos RPE LPC que incluyen la predicción en segmentos largos. El codificador opera con tamaños de segmento de 20 ms (160 muestras) y trabaja con un LPC de orden ocho. Cada 5 ms se actualizan los valores de los parámetros de estos segmentos (ganancia y retardo), y los parámetros de excitación con un procedimiento de análisis por síntesis.

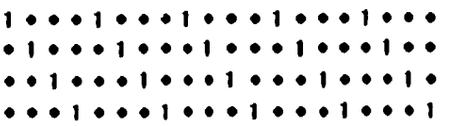

Figura 2.5: Cuatro posibles vectores para un codificador RPE

2.3.4 Codificador CELP (Code-Excited Linear Prediction)

También conocido como SELP (Stochastic-Excited Linear Prediction). Se basa en la técnica de análisis por síntesis.

La diferencia entre los codificadores MPLPC y CELP es la función de excitación -los pulsos del MPLPC se reemplazan por una secuencia de innovación.

En el codificador CELP, cada bloque de muestras de voz reconstruida es producida por los filtros de segmentos largos y por los filtros de tracto vocal LPC.

En el codificador hay una tabla de códigos donde se selecciona la secuencia de innovación óptima con la cual se tiene el mínimo error entre la señal de entrada y la señal sintetizada que se escogió. En el decodificador se tiene una copia de la tabla de códigos y se transmite un número indexado para indentificar la secuencia de innovación elegida con ganancia en términos, la cual selecciona la potencia de la señal de excitación.

Como en los codificadores MPLPC, tampoco existe diferencia entre las señales vocales y las no vocales, por lo que se utiliza el mismo método de análisis para determinar la forma de onda de excitación para todos los segmentos de la voz.

La secuencia de innovación es de 40 muestras y es más común el análisis adaptivo de la tabla de códigos, el cual también se da cada 40 muestras. La secuencia de innovación pueden ser vectores de números aleatorios, o vectores , donde sólo pocas muestras tienen valor

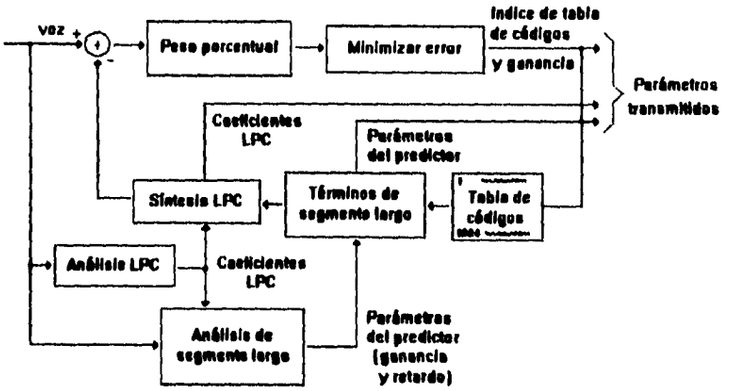

Figura 2.6: Codificador CELP (Code-excited LPC)

diferente de cero; o también vectores ternarios, donde todas las muestras tienen amplitud entre -1, 0 y 1; o bien alguna estructura de la tabla de códigos.

Para ejemplificar la utilidad de este codificador se tienen las dos siguientes figuras. La figura 2.7 muestra el espectro de la señal de voz original mientras que en la figura 2.8 ejemplifica el espectro de una señal de voz decodificada donde se utilizó un codificador CELP.

Aplicaciones

La codificación CELP es una técnica que se utiliza en los estándares de radio celulares. En los estándares especificados en el Departamento de Defensa de los E.U.A. [5] de 4.8 kbit/s, trabajan con un filtro de orden 10, el cual se actualiza cada 30 ms. Cada 7.5 ms se derivan los parámetros de la predicción de segmentos largos en la forma de la tabla de código adaptivo utilizando un procedimiento de análisis-por-síntesis. El codificador se diseñó para operar hasta 4.8 kbit/s, sin embargo esta tasa de codificación puede ser reducida al utilizar un subconjunto de la tabla de códigos.

La tabla de código está estructurada para que el codificador opere a diferentes velocidades y así poder establecer la comunicación y la calidad percibida será equivalente a la de los codificadores de baja velocidad. Los codificadores CELP especificados por los estándares

[5] US DOD: US Department of Defense

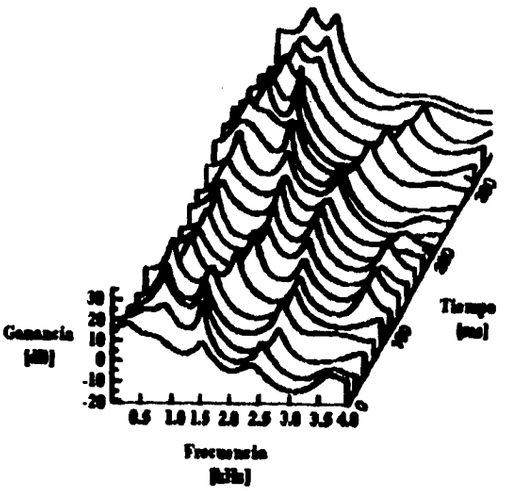

Figura 2.7: Espectro de la señal de voz original

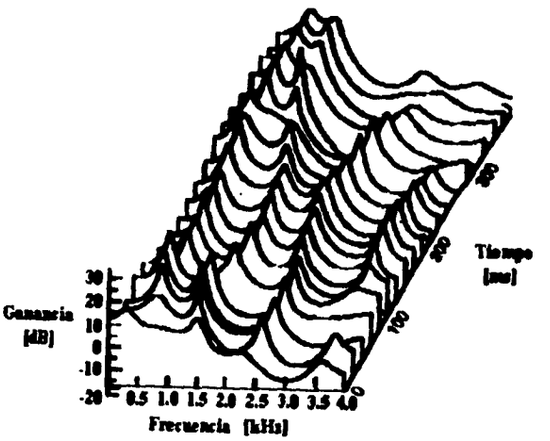

Figura 2.8: Espectro de la señal de voz decodificada, codificada con un CELP de 8 kbit/s

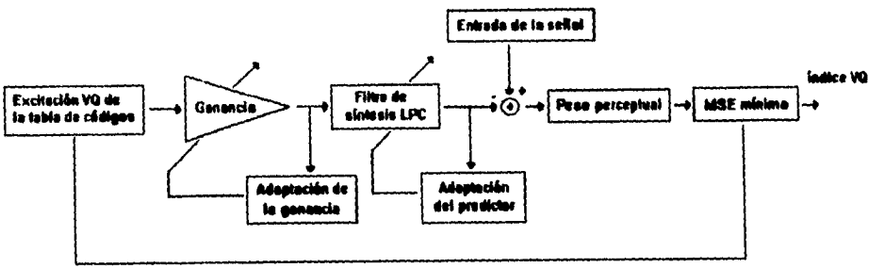

Figura 2.9: Codificador CELP de bajo retardo LD-CELP

de telefonía americanos y japoneses fueron desarrollados por Motorola y se conocen como VSELP (Vector-sum excited LPC). Los VSELP trabajan con la misma estructura de análisis LPC y de la tabla de código adaptivos que los CELP convencionales, la diferencia con éstos es la estructura de la tabla de código fija, dos tablas de código fijas se utilizan para los estándares de 8 kbit/s de sistemas de E.U. digitales y una tabla de código fija para los estándares de 6.7 kbit/s japoneses. Las tablas de código fijas de los VSELP son formados a partir de la combinación lineal de los vectores base y la estructura de cada tabla se usa para controlar el procedimiento de búsqueda.

2.3.5 Codificador LD-CELP (Low Delay CELP)

En la compañía Bell Telephone (BT), se desarrollaron los CELP de bajo retardo (LD-CELP). La característica más importante es que su retardo es menor a 2 ms, obteniéndose con un análisis LPC que tiene tiempos de espera en bloques de muestras de voz de hasta 20 ms antes de comenzarlo.

Un análisis LPC hacia atrás se utiliza para reconstruir la señal; y como ésta se tiene tanto en el codificador como en el decodificador, el análisis LPC se repite en este último y por lo tanto no hay necesidad de transmitir los coeficientes del LPC. Las longitudes de los vectores de excitación son de 5 muestras, mientras que una tradicional son de 40.

Una desventaja es su gran complejidad de diseño porque trabaja con un LPC de orden 50.

2.4 Codificadores en el dominio de la frecuencia

2.4.1 Codificadores SBC y Codificadores ATC

En el dominio de la frecuencia se tienen dos técnicas de codificación que son:

1. Codificador sub-banda SBC.
2. Codificador de transformación adaptivo ATC.

Estos codificadores se usan para obtener una alta calidad con un ancho de banda de aproximadamente 7 KHz en señales de audio. Se tienen, por ejemplo, los codificadores de señales telefónicas que aseguran la compatibilidad entre diferentes implementaciones. La codificación de algoritmos de señales de audio de 7 kHz a 64 kbit/s se basa en dividir la señal de entrada en dos frecuencias, la primera de 0 a 4 kHz y la segunda de 4 a 7 kHz. La codificación puede operar a 64 kbit/s, 56 kbit/s y 48 kbit/s e incrementará la calidad de la señal codificada conforme aumente la velocidad de transmisión.

2.4.2 Codificador de excitación multibanda (MBE)

Una técnica nueva es la codificación de excitación multibanda, la cual se basa en tener paquetes de voz de 20 ms donde el número de bandas usadas por algún paquete de voz en particular depende de la frecuencia fundamental. El ancho de banda de las frecuencias de banda se analiza con la transformada discreta de Fourier (DFT).

Haciendo referencia en la figura 2.1, en el modelo MBE, la clasificación de vocal o no-vocal se derivan de cada banda de frecuencia para cada paquete de voz. Así, por ejemplo:

- Si se tuviera a todas las bandas de frecuencia como vocales, entonces la excitación de esas bandas de frecuencia serán de la forma de excitación vocal (senoidal);
- Si se consideran no-vocales, entonces la banda de frecuencia serán del tipo no-vocal (como ruido);
- Pero si algunas son vocales y otras no-vocales, entonces la señal de excitación será una combinación de formas de excitación vocal y no-vocal.

Dentro de una banda de frecuencias se tienen tres armónicas de frecuencia fundamental y sus amplitudes las cuales definen la respuesta en frecuencia del modelo MBE dentro de la banda de frecuencia que se esté trabajando. En el decodificador, en cada banda de frecuencia vocal se genera una excitación donde se combinan tres señales senoidales, y sus amplitudes dependen de la amplitud de los espectros transmitidos y las frecuencias son tres armónicas de frecuencia fundamental que están en la banda de frecuencias seleccionada.

En una banda de frecuencias no-vocal, la señal se deriva de un generador de ruido blanco con amplitudes de espectros que controlan la energía de la señal de excitación.

Aplicaciones

Un codificador IMBE (codificador de excitación multibanda optimizado) se utiliza para sistemas de comunicaciones satelitales. La velocidad de codificación del canal es de 6.4 kbit/s y la codificación de la voz a 4.5 kbit/s. Pero si se requiere una velocidad de codificación de canal de 2.25 kbit/s, se obtendrá combinando los códigos Hamming y Golay.

2.5 Estándares para sistemas de codificación

Para la optimización de los diferentes sistemas de comunicación se han creado diferentes estándares. En décadas pasadas los estándares de telefonía, por ejemplo en los E.U.A. eran desarrollados de acuerdo al Sistema Bell, sin embargo los cambios en esta industria han generado una gran proliferación de ellos.

De esta manera, dentro de los organismos que han desarrollado más estándares en sistemas de codificación se encuentran:

1. Unión Internacional de Telecomunicaciones ITU.

2. Comité Consultivo Internacional de Radio CCIR.

3. Comité Consultivo Internacional de Telefonía y Telegrafía CCITT.

La CCITT [6] es el organismo encargado de marcar todos los estándares para transmisión de red pública. Sin embargo existen servicios donde no sería costeable si se rigieran por los criterios de calidad de CCITT, así por ejemplo los sistemas de celulares están por debajo de esos estándares, así como la telefonía marítima y aeronáutica; sin embargo, como ya se mencionó anteriormente, para teléfonos inalámbricos si es necesario tener una calidad "toll quality".

Una vez creado este organismo, se han desarrollado muchas de las llamadas "Recomendaciones" que es donde se fijan los estándares; en la tabla siguiente se mencionan algunas de estas:

Recomendación	*Codificación*	*Velocidad kbit/s*
G.711	PCM	64
G.726 G.727	ADPCM	40,32,24 y 16
G.722	ADPCM (AB de 7 kHz)	56-64
G.728	LD-CELP	17

[6] CCITT: International Telegraph and Telephone Consultative Committee

Debido a que la codificación ADPCM es de amplio uso se tienen varios estándares, donde por ejemplo, la CCITT ADPCM marca la Recomendación G.721, la cual define los 32 Kbit/s y ha sido utilizada en la CAI [7] para las estaciones base que se usaron para la segunda generación de teléfonos inalámbricos (CT2). También se puede mencionar el uso en los estándares de la tecnología digital inalámbrica europea (DECT).

La Recomendación G.726 es una extensión de la G.721, ya que incluye la operación del algoritmo a velocidades de 16, 24, 32 y 40 kbit/s; ésta se usa en equipo de circuitos digitales de multiplicación (DCME), el cual es empleado para incrementar la capacidad de conexiones intercontinentales usando una velocidad de codificación de 32 kbit/s, la codificación a 40 kbit/s se usa en transmisión de datos en canales de voz.

La Recomendación G.727 es una variante de la G.726, definiendo un algoritmo que usa una estructura añadida para también codificar a las cuatro velocidades antes mencionadas en la Recomendación G.726. El equipo que se rige con esta Recomendación es el que utiliza en redes de conmutación de paquetes.

La codificación CELP utiliza los estándares de radio celulares, CCITT de 16 kbit/s; utilizándose estos estándares en el Departamento de Defensa de los E.U.A.

Los LD-CELP se estandarizan con la Recomendación G.728, que es para algoritmos de 16 kbit/s en los cuales pueden usar aritmética de punto flotante.

Para los codificadores en el dominio de la frecuencia SBC y ATC, se tiene la Recomendación G.722 la cual especifica la codificación de algoritmos de señales de audio de 7 kHz a 64 kbit/s.

4. Instituto Nacional Americano de Estándares ANSI.

   El comité T1 en telecomunicaciones a través del subcomité técnico T1A1.6 (formalmente T1Y1.2) se dedica a los estándares de procesamiento de señales, el cual incluye la codificación de voz para aplicaciones de redes. Este comité ha tomado Recomendaciones CCITT y las ha modificado de acuerdo a las redes de E.U.A.

5. Àsociación de la Industria de Telecomunicaciones ANSI/TIA.

   El subcomité TR.45 estandariza la tecnología de codificación de la voz para aplicaciones en radio digital móvil. Recientemente se han desarrollado los estándares IS-54 para los VSELP a 79 kbit/s utilizados en redes digitales de E.U.A.

6. Sistema Nacional de Communicaciones NCS.

[7] CAI:Common Air Interface

El programa de estándares federales de telecomunicaciones a través de la NCS coordina la estandarización de codificación de la voz para el gobierno de los E.U.A. Los estándares federales 1015 (FS 1015) usados para los LPC-10 a 2.4 kbit/s y los FS 1016 en codificación CELP a 4.8 kbit/s se utilizan para los sistemas de seguridad de las comunicaciones del gobierno norteamericano.

7. Sistema Global para Telecomunicaciones Móviles GSM.

   Estos han estandarizado los RPE-LTP usados en los sistemas de radio digital móvil. Ahora trabajan en los estándares de la siguiente generación que emplea una velocidad de 13 kbit/s.

8. Inmarsat

   Un codificador IMBE (Codificador de excitación multibanda optimizado) se usa en Inmarsat para sistemas de comunicaciones satelitales. La velocidad de codificación del canal es de 6.4 kbit/s y la codificación de la voz a 4.5 kbit/s.

Otras organizaciones que se ocupan de la emisión de estándares son:

Instituto Nacional Europeo de Estándares de Telecomunicaciones ETSI, JDC, CITEL.

Cada una de las organizaciones se ha dedicado a estandarizar de acuerdo a las diferentes aplicaciones de redes que existen, así por ejemplo la CCITT estandariza la Redes Públicas Telefónicas Conmutadas PSTN; la ANSI el equipo de seguridad, las terminales digitales celulares y las terminales end/user; la ANSI-TIA se ha enfocado también a la tecnología de terminales end-user.

2.6 Diferencias entre las técnicas de codificación de la voz

La calidad de la voz se mide de acuerdo a la complejidad y una manera es por el número de instrucciones por segundo MIPS y por el número de operaciones por segundo MOPS. Otra forma de medir la calidad de la voz es por el consumo de potencia en la implementación, es decir, cantidad de memoria requerida. También se tienen medidas de algoritmos de codificación de voz que combinan las dos anteriores, conocidas como *ad hoc* (MIPS/MOPS y requerimiento de memoria).

La comparación que se establece entre las diferentes técnicas depende de tres categorías de calidad:

1. Calidad "Toll" (calidad de conexiones de red pública de larga distancia).
2. Calidad "Comunicaciones" (mantener la identidad del que habla).
3. Calidad "Sintética" (sonidos de la voz sintéticos).

La tabla siguiente muestra las diferentes técnicas de codificación, clasificándolas dentro de algunas de las categorías arriba mencionadas:

Técnica de codificación de la voz	*Estandarización*	*Velocidad (kbit/s)*	*Calidad de la voz*	*Complejidad*
PCM	G.711	64	Toll	1
ADPCM	G.721	32	Toll	10
CELP	G.728	16	Toll	450
RPE-LPC	GSM	13	Comunicaciones	100
VSELP	Celular EUA	8	Comunicaciones	250
CELP	DOD EUA	4.8	Sintética/ Comunicaciones	400
IMBE	Inmarsat estándar-M	4.15	Sintética/ Comunicaciones	150
LPC10	Estándar militar	2.4	Sintética	100

Conclusión

Debido a los grandes avances hechos dentro de este campo, en el futuro se pretende desarrollar estas técnicas y poder lograr que:

1. Los estándares CCITT en algoritmos de bajo retardo de 8 kbit/s de codificación de la voz ofrezcan una calidad equivalente a los estándares de 16 kbit/s de la CCITT.
2. Un codificador de voz de 4 kbit/s tenga una calidad igual a la del ADPCM a 32 kbit/s.
3. Un codificador de voz de 2.4 kbit/s alcance la calidad de los algoritmos de 4.8 kbit/s.

Todas las técnicas de codificación antes mencionadas han sido implementadas a la fecha gracias a la evolución que se ha dado en los microprocesadores los cuales hacen posible la operación en tiempo real de los algoritmos mencionados. Sin embargo, dichas técnicas continuan en desarrollo para poder lograr calidades más altas y tasa de codificación más bajas. El siguiente capítulo profundiza en los medios sobre los cuales se aplican las diferentes técnicas de codificación: los Procesadores Digitales de Señales - DSP's.

Referencias

[1] F.A. Westall, S.F.A. Ip: *Digital Signal Processing in Telecommunications*, cap. 11, Gran Bretaña, Chapman & Hall, B.T. Telecommunications, 1993.

[2] Spiros Dimolitsas: *Standardazing Speech-Coding Technology for Network Applications*, IEEE Communications Magazine, pp. 26-33, E.U.A, IEEE Inc., Noviembre 1993.

[3] Jutta Degener: *Digital Speech Compression*, Dr. Dobb's Journal, pp. 30-34, Dec. 1994.

[4] Panos E. Papamichalis: *Practical Approaches to Speech Coding*, cap. 3 y 8, New Jersey, Prentice-Hall Inc., 1987.

[5] Panos Papamichalis, Ph.D.: *Digital Signal Processing Applications with the TMS320 family*, vol.3, (Theory, Algorithms and implementations), E.U.A., Texas Instruments Inc., marzo 1990.

Capítulo 3

Papel de los DSP's en el procesamiento de la voz

3.1 Introducción

Debido a su amplia utilización en los sistemas de telecomunicaciones actuales tales como los filtros digitales, procesamiento de voz e imágenes, transformada rápida de Fourier, entre otros; en las últimas décadas se ha acelerado el desarrollo de los procesadores digitales de señales DSP [1].

El término DSP es un sinónimo de microcomputador rápido de procesamiento de señales en tiempo real; aunque también podría ser un procesador matemático que acopla la señal de transformación que se desea. Y los dos significados son válidos ya que el desarrollo de la tecnología ha permitido que se den ambos; además se ha logrado una disminución en los costos de fabricación y con esto se ha beneficiado la implementación de mas diseños con este tipo de microprocesadores.

3.2 Esquema de funcionamiento de un DSP

Un DSP toma una señal de entrada de diferentes formas, como pueden ser las de un micrófono; línea telefónica cuando son datos los que se van a modular; de una señal de video para ser codificada o bien para para almacenar datos en una computadora. Esta señal de banda limitada se ha muestreado previamente a través de un convertidor analógico-digital, y para que no se tenga pérdida en la información es necesario tener una frecuencia de muestreo f_s que debe ser dos veces mayor a la banda de frecuencia de la señal de entrada, esto es: $f_s > 2f_m$ para evitar el efecto de aliasing.

De esta manera se tiene que el DSP actúa sobre una señal digital, muestra por muestra y modifica el dato de alguna manera, esto quiere decir que la información será procesada. Por ejemplo se puede tomar una secuencia de operaciones de multiplicación y sumas, es decir una instrucción MAC [2] y de esta manera poder procesar la información con una mayor velocidad. Este proceso es clave ya que así se puede establecer la diferencia entre los DSP's

[1] DSP: Digital Signal Processor
[2] MAC: Multiply and Acummulate

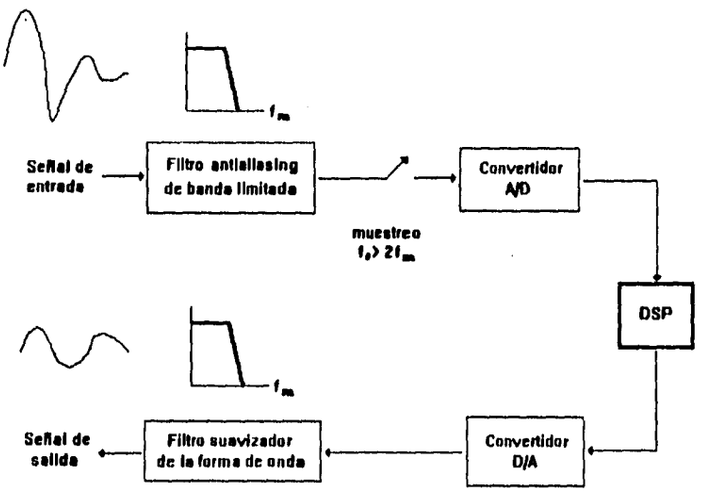

Figura 3.1: Esquema de funcionamiento de un DSP

y los microprocesadores comunes. Un DSP hace posible la ejecución del sistema en tiempo real.

Finalmente, después del procesamiento, las muestras se convierten a valores analógicos nuevamente, pasando por un filtro regenerador (usualmente un filtro paso bajas) obteniendo así la salida de la señal.

La figura 3.1 presenta el esquema del funcionamiento de un DSP como se ha mencionado anteriormente.

3.3 Características generales de los DSP

Cualquier tipo de dispositivo que se vaya a considerar como un DSP debe cumplir con las siguientes características:

- *Algoritmos que manejen procesos matemáticos intensos.*

  Dentro de las aplicaciones que se mencionaron anteriormente, por ejemplo se encuentra el diseño de filtros digitales (FIR, IIR), los cuales requieren llevar a cabo una serie N de multiplicaciones y sumas o bien sumas de productos. Estos cálculos son intensos y de hecho tanto los filtros FIR como la convolución son indispensables en el procesamiento digital de señales. Estas operaciones como ya se mencionó anteriormente, se conocen como *MAC*.

- *Operación en tiempo real.*

  Los DSP pueden desempeñar los algoritmos en tiempo real, esto quiere decir que el proceso que realice el dispositivo no incrementa el retardo notablemente desde el punto de vista del usuario. Por ejemplo en los filtros FIR las operaciones se deben calcular en el orden de cientos de microsegundos antes de que la siguiente muestra entre al sistema; otro ejemplo sería en el reconocimiento de voz, donde el retardo sería notable en el momento que se tenga la palabra hablada y se vaya a realizar el reconocimiento de la misma, entonces sería inaceptable sino se realizara en tiempo real.

- *Sistema flexible.*

  El diseño de un sistema en procesamiento digital de señales debe ser flexible, esto es que el algoritmo del DSP se pueda estar actualizando. Esto último es posible desde la aparición de los DSPs programables. En el reconocimiento de voz se tienen que estar modificando continuamente los algoritmos ya que se requieren de nuevas etapas de desarrollo.

- *Operaciónes sobre vectores de datos.*

  La aplicación debe tener la capacidad de manejar los datos muestreados del sistema en el orden en que van a ser ejecutados por el procesador digital. Por ejemplo, en un filtro FIR la señal de entrada está siendo muestreada a intervalos periódicos (a la frecuencia de muestreo) multiplicados por un factor $a(i)$ y sumados todos dan por resultado la salida $y(n)$. Ejemplos típicos de frecuencia de muestreo se tienen en la siguiente tabla:

Aplicación	*Frecuencia de muestreo nominal*
Control	1 kHz
Telecomunicaciones	8 kHz
Procesamiento de voz	8-10 kHz
Procesamiento de audio	40-48 kHz

3.4 Ventajas y desventajas de los DSP

Las ventajas que se pueden mencionar son:

- Una ventaja sobresaliente ha sido la reducción del costo, como se dijo al inicio del capítulo, en su tamaño y consumo de potencia comparados con los dispositivos analógicos equivalentes.
- Los sistemas DSP son compatibles con otras tecnologías digitales tales como la transmisión, redes y computadoras. La interface en esos sistemas es mucho más simple que en la tecnología analógica que funcione de manera similar.
- La programabilidad, ya que ofrece flexibilidad en el sistema y puede ser modificado en cualquier momento que se requiera dependiendo de la aplicación.
- Una buena precisión con una excelente definición, esto extiende el rango del procesamiento de la señal que se va a implementar reduciendo así los costos de prueba y por tanto de manufactura.
- Estabilidad en el tiempo, significa que el DSP no cambia con el tiempo sus características.

Por otro lado se tienen ciertas desventajas sobre los sistemas analógicos. Por ejemplo si se requieren tareas simples de procesamiento digital, como es el caso de la interface de las líneas analógicas, el costo del uso de un DSP sería innecesario. Además de que se pueden tener problemas asociados con la velocidad del reloj del DSP el cual puede provocar distorsión en la comunicación debido a la interferencia electromagnética.

Las tendencias actuales para minimizar estas desventajas, se centran alrededor de los siguientes puntos:

- Amplio desarrollo de herramientas de software tales como los depuradores, que ayudan al diseñador a implementar sistemas cada vez mas complejos y sofisticados de acuerdo a los requerimientos del entorno donde se implantarán.
- Debido a que los DSP's se apoyan primordialmente en los rápidos procesos matemáticos existe la necesidad de desarrollar procesadores cada vez mas complejos y así poder implementar algoritmos con mayor facilidad.

3.5 Evolución de los DSP's

Debido al avance que ha habido de los DSP's a partir de los últimos años de la década de los 70's, una serie de familias se han desarrollado en diferentes compañías. Los dispositivos programables acaparan de hecho los mayores esfuerzos de desarrollo, debido a su versatilidad. La tabla siguiente muestra estos dispositivos así como sus características.

Compañía	*Serie*	*Año de introducción*	*Tiempo del MAC (nseg)*	*Número de bits de punto fijo*	*Número de bits de punto flotante*
AMI	S2811	1978	300	12/16	
NEC	μPD7720	1980	250	16/32	
	μPD77230	1985	150		32
Texas Instruments	TMS32010	1982	390	16/32	
	TMS32020	1987	200	16/32	
	TMS320C25	1989	100	16/32	
	TMS320C30	1989	60	24/32	32/40
	TMS320C40	1992	40	32	40
	TMS320C50	1990	35	16	
Motorola	MC56001	1986	75	24	
	MC 96002	1990	50	32/64	32/44
	MC56002	1991	50	24/48	
AT&T	DSP32C	1988	80	16 o 24	32/40
	DSP16A	1988	25	16/36	
	DSP3210	1992	60	24	32/40
Analog Devices	ADSP-2101	1990	60	16	
	ADSP-21020	1991	40	32	32/40

El DSP S2811 de AMI se considera el primero, aunque se disponía del 2920 de Intel el cual tenía un convertidor analógico/digital, sin embargo no trascendió porque no efectuaba las operaciones de multiplicación en un solo ciclo de máquina. En 1980, NEC lanzó comercialemte el μPD7720 teniendo la característica de realizar multiplicaciones en un solo ciclo. Texas Instruments (TI) desde 1982 diseñó el TMS32010 y apartir de este se ha desarrollado toda una familia, mejorando su arquitectura, como por ejemplo la ejecución de las instrucciones en menos ciclos de máquina:

Dispositivo	*Ciclos de máquina (ns)*
TMS320C10	160-200
TMS32020	160-200
TMS320C25	100-125
TMS320C30	60-75
TMS320C40	40

El MC56001 de Motorola manejaba operaciones de punto fijo mientras que el MC96002 las puede manejar tanto con punto fijo como con punto flotante. Los DSP's que trabajan con punto flotante incrementan su rango dinámico pero también es mucho mas costosa su implementación.

De esta manera se ha logrado:

- tener una mayor capacidad de memoria;
- combinar las operaciones se suma y multiplación en una sola instrucción (MAC) que han reducido de 400 ns (TMS32010) hasta 40 ns (TMS320C40);
- expander las funciones de entrada/salida teniendo un mayor desempeño con los dispositivos externos para atender más rápido a las interrupciones;
- para ello el número de pines que inicialmente era de 64 se ha incrementado a mas de 200 teniendo como resultado una mayor flexibilidad;
- la zona del componente multiplicador ocupa ahora el 5 % del área total (tamaño del *chip*) mientras que en 1980 era del 40 %;
- la memoria RAM del DSP aumentó de 2.5 k (tecnología NMOS) a 32 kbytes (tecnología CMOS) con dos bloques con buses de datos independiente;
- acceso simultáneo de instrucciones y datos, operaciones de multiplicación en paralelo.

Con todos estos cambios en la arquitectura se ha tenido una reducción a una tercera parte en costo, tamaño, peso y consumo de potencia.

Dispositivo	*Fujitsu MB8764*	*NEC 7720*	*Texas Instruments TMS32010*
Procesador	CMOS	CMOS	NMOS
Año	1983	1982	1980
Superficie (mm^2)	91	28	45
Potencia (W)	0.29	1.0	0.9
Precisión (*bits*)	16	16	16
Velocidad (ns)	100	250	200
Multiplicador	16*16:=26 entero	16*16:=32 entero	16*16:=32 entero
Memoria de programa	1k*24 ROM/EXT	512*23 EPROM/ROM	4k*16 ROM/EXT
Datos en RAM	256*16	128*16	144*16
Datos en ROM	en programa	512*13	en programa

Aplicación de los DSP's para el diseño de filtros FIR

Dentro de las aplicaciones mas poderosas que se tienen con los dispositivos DSP's están por ejemplo, la transmisión de datos, reducción de ruido, cancelación de eco e interferencia en telefonía; y en la reducción de velocidad de transmisión en la codificación de voz e imagenes, entre otras. En los filtros adaptivos se requiere del cálculo de los coeficientes, y algunas señales rara vez requieren más de 16 bits de precisión pero en otras esta debe de ser mucho mayor. En estas últimas la doble precisión del software implica una mayor velocidad y los DSP's con tales características actualmente son muy costosos. Pero en la práctica , una precisión de 24 bits es aceptable utilizando un microprocesador de punto fijo, el cual como ya se dijo anteriomente es mucho mas económico comparado con uno de punto flotante.

Para implementar un filtro adaptivo eficiente es necesario efectuar operaciones en paralelo, es decir se pueden realizar en menor número de ciclos de máquina el fetch de instrucciones y datos, las multiplicaciones, los direccionamientos, el almacenamiento de datos. Por ejemplo el TMS320C25 requiere de 7 ciclos de máquina para poder calcular un coeficientes del filtro FIR, mientras que el TMS320C30 y solo requiere de 3 ciclos de máquina por coeficiente, haciéndolo doblemente eficiente.

Un filtro LMS FIR convencional con un arreglo simple de coeficientes reales, se podría

implementar con un DSP56001 (de Motorola), utilizando 3 ciclos de máquina por coeficiente. Y precisamente una de las modificaciones que se han hecho en las arquitecturas es poder calcular el dato, sacarlo e introducirlo en el acumulador en un solo ciclo. Si los coeficientes se procesaran en parejas se requerirían solo dos ciclos por coeficiente, se tiene el código Kernel (ver tabla), lográndose tener únicamente un retardo por muestra en el proceso adaptivo.

Código de operación	*Operandos*	*Dato a mover*	Dato a *mover*	*Ejecución*
mac	x0,y0,a	y0,b	b,y:(r5)+	a=$c_0(j) * x(n), b = c_0(j)$
macr	x1,y1,b	x:(r0)+,x0	y:(r4)+,y0	b=$c_0 + e * x(n-1) -> c_0(j+1)$
mac	x1,y0,a	y0,b	b,y:(r5)+	a=a+$c_1(j) * x(n-1), b = c_1(j)$
macr	x0,y1,b	x:(r0)+,x1	y:(r4)+,y0	b=$c_1 + e * x(n-2) -> c_1(j+1)$

El tamaño de la RAM del DSP56001 limita el número de coeficientes a 120, pero si se extiende la memoria externa se requerirán de 3 ciclos por coeficiente, lo que siginificaría que efectuando las operaciones, como se menciona en la tabla, se tendría que este dispositivo a 33 MHz y con una RAM con un tiempo de acceso lo suficientemente corto podría procesar 650 coeficientes en un período de muestreo de 8 KHz.

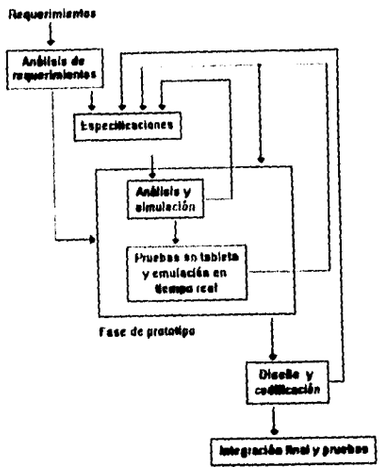

Figura 3.2: Proceso de diseño para un sistema DSP

3.6 Secuencia de desarrollo de un sistema DSP

No existe una metodología única que pueda tomarse como modelo en el desarrollo de un sistema de este tipo. Por lo general se parte de una base teórica que expresa analíticamente las capacidades de los algoritmos que se desean implantar. Es común actualmente basarse en implantaciones del algoritmo en algún lenguaje de alto nivel en el cual se han probado las bondades de dicho algoritmo. Para llegar a una implementación en tiempo real se establecen de inicio una serie de requerimientos y especificaciones las cuales deben ser cumplidas para que el proyecto llegue a buen término. En la figura 3.2 se esquematiza un proceso típico de diseño para un sistema DSP.

Posteriormente al establecimiento de las especificaciones se efectúa una simulación donde de ser posible, se toman en cuenta las limitaciones propias del hardware, como pueden ser la precisión propia de este y su capacidad de cálculo. La simulación puede realizarse utilizando también un lenguaje de alto nivel de propósito general como lo es C, o bien utilizar algún paquete de software diseñado para ello, como los son Hypersignal o SPW. La depuración en la etapa de simulación es importante debido a que un cambio en las especificaciones o en el algoritmo en esta etapa es fácil de llevar a cabo sin mayores complicaciones. Un cambio de este tipo en la etapa de implantación física requiere mucho mayores inversiones en tiempo y dinero.

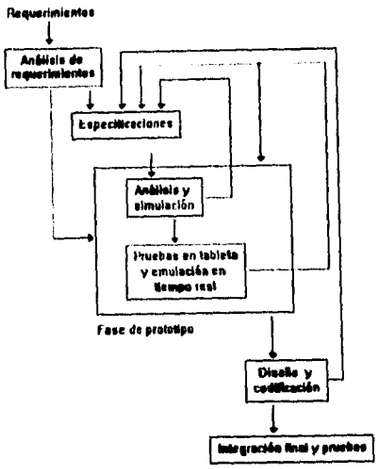

Figura 3.2: Proceso de diseño para un sistema DSP

3.6 Secuencia de desarrollo de un sistema DSP

No existe una metodología única que pueda tomarse como modelo en el desarrollo de un sistema de este tipo. Por lo general se parte de una base teórica que expresa analíticamente las capacidades de los algoritmos que se desean implantar. Es común actualmente basarse en implantaciones del algoritmo en algún lenguaje de alto nivel en el cual se han probado las bondades de dicho algoritmo. Para llegar a una implementación en tiempo real se establecen de inicio una serie de requerimientos y especificaciones las cuales deben ser cumplidas para que el proyecto llegue a buen término. En la figura 3.2 se esquematiza un proceso típico de diseño para un sistema DSP.

Posteriormente al establecimiento de las especificaciones se efectúa una simulación donde de ser posible, se toman en cuenta las limitaciones propias del hardware, como pueden ser la precisión propia de este y su capacidad de cálculo. La simulación puede realizarse utilizando también un lenguaje de alto nivel de propósito general como lo es C, o bien utilizar algún paquete de software diseñado para ello, como los son Hypersignal o SPW. La depuración en la etapa de simulación es importante debido a que un cambio en las especificaciones o en el algoritmo en esta etapa es fácil de llevar a cabo sin mayores complicaciones. Un cambio de este tipo en la etapa de implantación física requiere mucho mayores inversiones en tiempo y dinero.

3.6.1 Alternativas en hardware

Desarrollo de dispositivos LSI de propósito específico

En un principio el desarrollo de un sistema de procesamiento de señales comenzaba con la implementación de prototipos basados en electrónica de baja escala de integración como lo son las familias TTL y CMOS. Implementar una aplicación con este tipo de tecnología tenía muchas desventajas desde el punto de vista de la depuración del sistema, la cual era difícil y costosa; mientras que el costo del sistema se elevaba en gran medida cuando la complejidad de este crecía.

Con el desarrollo de mejores metodologías de diseño de circuitos integrados de propósito específico como son los ASIC [3], ha sido más fácil el diseñar aplicaciones de procesamiento digital de señales utilizándolos. De hecho, las primeras implementaciones sobresalientes en este campo se dieron mediante el uso de dispositivos de este tipo en aplicaciones como modems telefónicos, detectores MTDF y filtros digitales de pocas etapas. A la vez que se daba la posibilidad de implementar dispositivos mas complejos el alcance de las aplicaciones creció también, de manera que en la actualidad existen dispositivos LSI que realizan cancelación de eco, modulación digital e inclusive vocoders completos, como por ejemplo los vocoders que comercializa la empresa Qualcomm los cuales pueden ser insertados en un sistema de transmisión de voz y codificarla a una tasa variable a elección del usuario.

La popularidad de estos dispositivos ha disminuido con la aparición de opciones con capacidades de programación y depuración del sistema muy superiores; sin embargo su utilización es aún muy extensa debido a que una vez que el dispositivo se ha depurado sus costos de producción son muy bajos y pueden competir con éxito en el mercado. Su uso es amplio sobre todo en la industria de electrónica doméstica y para aplicaciones de alto desempeño como son las militares.

Existen opciones que representan un punto intermedio entre los dispositivos de diseño específico como los mencionados anteriormente y los dispositivos programables como la familia TMS. Dichos dispositivos conocidos genéricamente como gate-arrays ofrecen cierta flexibilidad al diseñador, a la vez que los tiempos de desarrollo y los costos de producción disminuyen. De hecho el avance en este sentido se debe al desarrollo de mejores herramientas CAE para el diseño de los circuitos, donde se ofrece al diseñador bloques cada vez mas complejos para cubrir sus necesidades específicas. Con las herramientas de sofware mas actuales el diseño se reduce a especificar un diagrama de flujo donde se indiquen las operaciones a realizarse sobre una señal de entrada. Existe una tendencia para definir lenguajes descriptores de hardware como VHDL [4] el cual es ahora un estándar apoyado por la IEEE y por un creciente número de simuladores, depuradores y herramientas gráficas.

[3] ASIC: Application specific integrated circuit
[4] VHDL: VHSIC Hardware description language

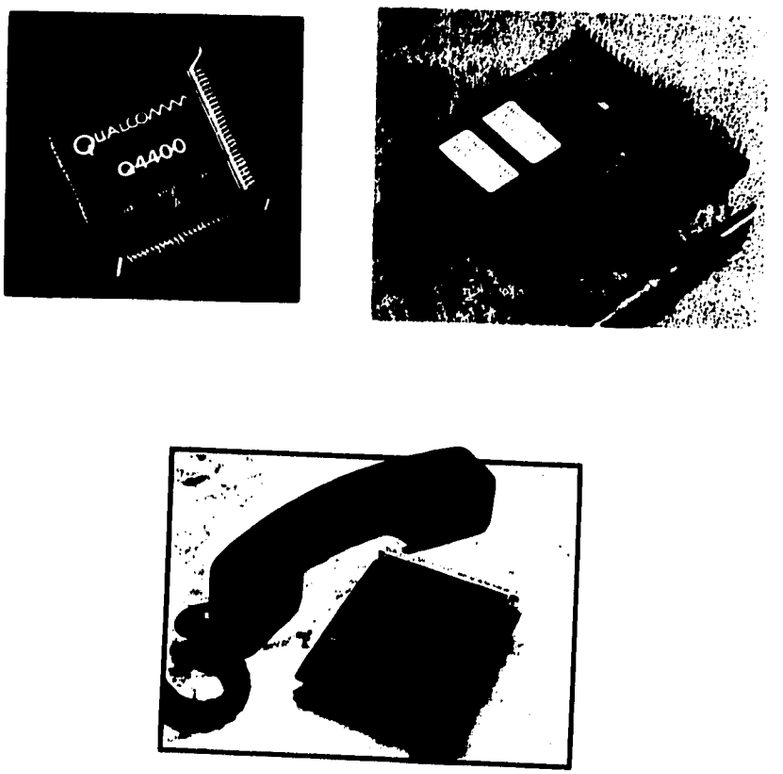

Figura 3.3: Vocoders comerciales de tecnología VLSI

3.6.2 Ventajas de los dispositivos programables

La aparición de dispositivos DSP programables ocasionó una revolución en la industria con consecuencias notables. El hecho de que un DSP tenga capacidades de programación pone al alcance de un número mucho mayor de usuarios los beneficios de esta tecnología, puesto que la inversión necesaria para desarrollar un sistema es mucho menor a la vez que el tiempo requerido para depurarlo disminuye también. Por lo general un algoritmo de procesamiento digital de señales surge primeramente como un proceso numérico probado únicamente de manera teórica mediante un programa en un lenguaje de alto nivel como son Fortran o C.

Los primeros DSPs programables eran muy similares a un microprocesador convencional pero introducían superiores capacidades de cálculo numérico, como la primera generación de la familia TMS, el 320C10. En este primer *chip* se introdujo la instrucción MAC [5]. Posteriormente las mejoras se han dado principalmente en dos sentidos:

1. *Mejoras en el hardware.* En posteriores generaciones de la familia TMS se introdujo la capacidad de manejo de números de punto flotante, que incrementó considerablemente el rango dinámico de las operaciones y minimizó el efecto de los errores por redondeo. Aparecieron también características de paralelización de las operaciones utilizando estrategias de duplicación de buses internos y unidades aritméticas. La implementación de esquemas de *pipeline* permite que se efectúen de manera simultánea las operaciones de búsqueda, de codificación, almacenamiento y ejecución de las instrucciones reduciendo drásticamente el número de ciclos necesarios para efectuarlas.

2. *Mejoras en el software.* Inicialmente la programación de estos dispositivos se realizaba únicamente en lenguaje ensamblador, lo que exigía del programador un conocimiento profundo de los recursos del sistema y una constante verificación de las variables internas utilizadas por el microprocesador. En recientes versiones la capacidad de programar en lenguajes de alto nivel permite al programador concentrar su atención en la programación del algoritmo que desea implantar sin distraer su atención en administrar los recursos del sistema donde lo está implantando. Otra ventaja obtenida al programar en lenguajes de alto nivel consiste en el corto tiempo utilizado para transportar un algoritmo probado numéricamente a una aplicación en tiempo real.

[5] MAC: Multiply and Accumulate

3.7 Herramientas de desarrollo

Como ya se ha dicho, el éxito al desarrollar un sistema consiste en gran medida de las facilidades que se tengan para depurarlo y así reducir el tiempo crítico en el cual tenga que finalizarse el proyecto. Para ello los principales fabricantes de DSP's proveen de una amplia gama de herramientas de desarrollo.

Genéricamente hablando, los entornos de desarrollo están formados por elementos como los siguientes:

1. Compiladores para lenguajes de alto nivel
2. Optimizadores de código
3. Tarjetas de evaluación (hardware de desarrollo)
4. Simuladores
5. Depuradores integrados al hardware de desarrollo
6. Librerías de funciones para PDS

En las siguientes líneas se refieren dos de los principales entornos de desarrollo: el paquete SPADE[6], utilizado en los laboratorios BT, y el entorno de desarrollo para el TMS320C30, proporcionado por Texas Instruments.

3.7.1 El sistema SPADE

El sistema SPADE está pensado para el desarrollo y diseño de técnicas y herramientas de integración que describan, simulen e implementen las aplicaciones de los DSP's. Las características principales son:

- generar software con características de un lenguaje de alto nivel;
- sistemas de adquisición de datos que trabajen en tiempo real para las simulaciones;
- dar soporte a los dispositivos para mejorar su arquitectura;
- poder realizar un análisis gráfico de datos;

[6]SPADE: Signal Processing Application Development Environment

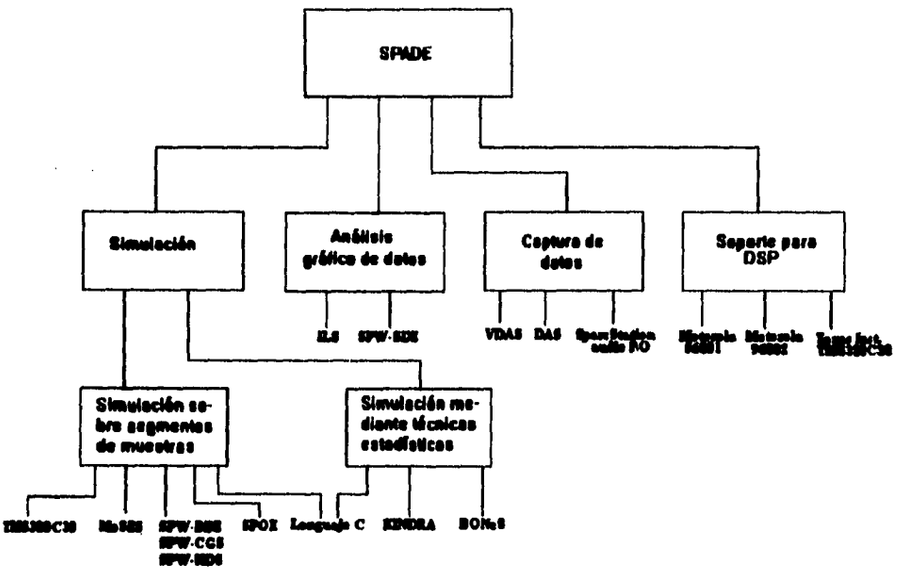

Figura 3.4: Diagrama de bloques de los componentes de SPADE

3.7.2 El entorno de desarrollo del TMS320C30

En el caso específico del TMS320C30 pueden mencionarse las siguientes herramientas de desarrollo:

1. Un compilador para ANSI C y un ligador asociado (CL30 y LNK30).
2. Un archivador que permite generar librerías de macros (AR30).
3. Un optimizador que minimiza el código resultante de una compilación.
4. Un conjunto de librerías de funciones para el C30 (SPOX de Spectrum Microsystems).
5. Una tarjeta de evaluación (EVM) que permite ejecutar en tiempo real los algoritmos a la vez que es posible monitorear su desempeño a través de un puerto de emulación.
6. Un depurador para la tarjeta de evaluación que permite monitorear paso por paso la ejecución de un programa (C Source Debugger).
7. Una tarjeta emuladora que permite monitorear el desempeño de un sistema objetivo mediante la adecuada conexión de un puerto de monitoreo a dicha tarjeta emuladora (XDS1000).
8. Utilerías varias que permiten la relocalización de código así como generación de código para ser grabado en diversos dispositivos de memoria.

3.7.3 Otras herramientas de desarrollo

Algunas de las *herramientas de desarrollo* en cuanto a software integrado de uso extendido son:

1. SPW [7]: Sistema de trabajo para el procesamiento de señales, desarrollada por *Comdisco Systems*. Es de utilidad para la simulaci'on de sistemas gráficamente y todas las funciones que se ejecutan en sincronía con el reloj de simulación son del tipo fuente a destino. Esta herramienta cuenta con los componentes siguientes:

   (a) SPW-BDE, es un editor de diagramas de bloques gráfico, de manera que se tiene una descripción del algoritmo del DSP además de proveer el código del programa ya que representa la jerarquía de los subsistemas ofreciendo también que el sistema se haga mas accesible para poder manipular mejor la información.

   (b) SPW-SDE, es un conjunto de rutinas que se utiliza para desplegar, procesar y analizar las señales que se generaron en el SPW-BDE, y estas se pueden utilizar después para simulaciones subsecuentes.

   (c) SPW-CGS, que es un sistema que genera el código y está pensado para producir un código ejecutable que pueda correrse en tres códigos diferentes del lenguje C, del TMS320C30 o bien del 96002.

   (d) SPW-HDS, es un sistema para diseñar hardware y se requiere para poder tener una precisión fija en el procesamiento que se va a aplicar en las simulaciones que se hubieran desarrollado previamente en el SPW-BDE, pudiendo representar el efecto de error de redondeo y truncamiento en la implementación del DSP de punto fijo o bien en los ASIC.

   (e) SPW-FDS, se utiliza para el diseño y especificación de diferentes tipos de filtros y únicamente se pueden usar en los editores SPW-BDE y SPW-SDE.

2. SPOX [8]: es un paquete comercial desarrollado por *SPECTRON MicroSystems Inc.* se implementaron un conjunto de librerías con funciones que se pueden llamar desde programas en *C*, como es el caso específico para el TMS320C30 de Texas Instruments y MC56001 de Motorola. Consta de tres elementos:

   (a) SPOX-LINK, tiene funciones para habilitar la comunicación entre DSP's, pudiéndose dar ya sea entre diferentes dispositivos o bien entre el host y el DSP.

   (b) SPOX-DSP, tiene funciones para un buen manejo de memoria, optimización de las funciones matemáticas y la entrada/salida de datos.

   (c) SPOX-RTK, consiste en funciones para crear y correr programas en una tarjeta evaluadora.

3. MoSES [9]; fue desarrollado por BT para que se pudiera tener una interface mas amigable con un programa de simulación de un modem, ya que provee un menú para inicializar variables. Este sistema de evaluación da una salida gráfica además de un análisis si así

[7] SPW:Signal Processing Worksystem
[8] SPOX:Signal Processing Operating eXecutive
[9] MoSES: Modem Simulation Evaluation System

se requiere. Una aplicación importante se tuvo durante la modificación de un modem receptor V.32 para probar la secuencia en tiempo real realizándose en una VAX y en una cinta de audio digital que tenía un sistema para adquirir datos. Las modificaciones a este programa se dan de acuerdo a la evolución del hardware.

4. BONeS [10]: es una herramienta comercial desarrollada por *Comdisco Systems*. Se usa en sistemas asíncronos como en redes de comunicación y la simulación de protocolos ofreciéndolo en un ambiente gráfico.

   Existen diversos componentes que son de gran ayuda para el desarrollo:

   (a) BONeS-BDE, es un editor de diagramas de bloques que se utiliza para crear, editár, documentar y almacenar diagramas de redes. Igual que en SPW se pueden crear bloques codificados usando *C* para implementar una función que no existe en las librerías estándar. Cuando un diagrama de red se salva en BONeS-BDE se genera y compila autómaticamente el código correspondiente en *C*.

   (b) BONeS-DSE, editor para datos estructurados desempeñando funciones similares a BDE, pero con datos. Estas estructuras pueden tener un número aleatorio de campos definiéndolos jerárquicamente.

   (c) BONeS-SIM, genera un programa principal para la simulación y se especifican los parámetros que se requieren en tiempo real y pueden extraerse datos desde la simulación para ser usados en el post-procesamiento. Todos los resultados de la simulación se almacenan en una base de datos para ser utilizados en análisis posteriores.

   (d) BONeS-PP, se utliza para el análisis de los resultados de la simulación. La información estadística se puede calcular desde la simulación de los datos para obtener, bajo diferentes condiciones, varias ejecuciones.

   BONeS se utilizó para modelar características importantes de la Recomendación G.764 CCITT, de los protocolos de voz empaquetada, con un codificador G.727 ADPCM siendo modelado en SPW. Gracias a estos dos sistemas de simulación fue posible obtener los efectos de una red congestionada en ADPCM transmitiéndose voz.

5. También puede mencionarse el paquete Kindra, desarrollado por BT.

Adicionalmente otras compañías han desarrollado principalmente software complementario para ayudar en la etapa de diseño y depuración del algoritmo. Ejemplos de estas últimas son Mathworks con su programa Matlab, Dadisp y otras.

[10] BONeS: Block-Oriented Network Simulator

Conclusión

La tendencia mas fuerte en la actualidad es generar herramientas cada vez mas poderosas partiendo de la programación en lenguaje de alto nivel a la vez que se aprovechan las características de hardware del dispositivo que se este utilizando, como son la paralelización y las operaciones en pipeline. En el estado actual de la tecnología en este campo, la mejor solución parece ser generar el cuerpo principal del programa en un lenguaje de alto nivel como C e implementar las rutinas mas críticas en lenguaje ensamblador para optimizar la ejecución del programa. Sin embargo, la decisión que se tome sobre cual de todas las opciones que existen dependerá de las necesidades en cada caso, de manera que se equilibren los factores de calidad, costo, tiempo de desarrollo y facilidad de actualización.

Referencias

[1] F.A. Westall, S.F.A. Ip: *Digital Signal Processing in Telecommunications*, cap. 1, 5 y 12, Gran Bretaña, Chapman & Hall, B.T. Telecommunications, 1993.

[2] J. H. Snyder et. al.: *Tools for Real-time Signal-Processing Research*, IEEE Communications Magazine, pp 64-74 ,E.U.A, IEEE pub., Noviembre 1993.

[3] Panos Papamichalis, Ph.D.: *Digital Signal Processing Applications with the TMS320 family*, vol.3 (Theory, Algorithms and implementations),E.U.A., Texas Instruments Inc., marzo 1990.

[4] Texas Instruments: *TMS320C3x, User's Guide*, Texas Instruments Inc., E.U.A., 1990.

Capítulo 4

Implementación del modelo de compresión de voz

4.1 Elección de un modelo de codificación: LPC

En base al análisis realizado en el capítulo 2, en el cual se hizo una revisión de algunas de las múltiples derivaciones que tienen los métodos basados en la codificación lineal predictiva (LPC), se tomo la decisión de que este es el método más viable para poder ser implementado para los propósitos de esta tesis. Específicamente, el presente capítulo detalla la implementación de un *Vocoder*, esto es, un codificador/decodificador LPC que modela la excitación (*pitch*) como una secuencia normalizada de pulsos espaciados entre sí el número de muestras determinadas por el sintonizador del pitch (*pitch tracker*).

4.1.1 Bases para el desarrollo del algoritmo

Dentro de las características en el algoritmo implementado, están:

- Manejo de aritmética de punto flotante de 32 bits.
- El algoritmo acepta como entrada voz muestreada a 8 kHz en muestras de números enteros de 16 bits.
- Los tipos de datos utilizados y mencionados anteriormente, son aceptados por DSP's de punto flotante comerciales, como es el caso del TMS320C30.
- Se eligió una tasa de segmentación de 160 muestras (20 ms) la cual es la más comunmente utilizada en codificadores de este tipo.
- La tasa de compresión es variable dependiendo del orden del filtro Lattice seleccionado.

4.2 El programa ADFO: *Análisis Digital de Formas de Onda*

4.2.1 Ambiente de desarrollo del algoritmo

Se llevó a cabo la programación del algoritmo en lenguaje C, utilizando el compilador de Microsoft C versión 5.1, cuidando de asegurar la compatibilidad de código con el ANSI C, especialmente en las rutinas correspondientes a las operaciones ligadas al proceso de compresión y descompresión.

El programa se llevó a cabo con objeto de probar distintas rutinas que en su conjunto conforman un modelo de compresión de voz con características muy flexibles.

Haciendo referencia al esquema mostrado en el capítulo anterior en el cual se presentan las distintas etapas necesarias para llevar un modelo de codificación hasta la fase final en la cual sea aplicado, el programa ADFO corresponde a la etapa de análisis y simulación.

Dado que el programa en sí es un simulador, en el cual se han contemplado como se dijo anteriormente, características que hacen factible transportar el código a dispositivos programables (específicamente el TMS320C30); el programa está en disponibilidad de aceptar diferentes tipos de formatos de forma de onda y asimismo generar distintos tipos de formatos como salidas que puedan ser utilizadas a su vez por otro tipo de programas que realicen análisis posteriores -en especial se contempló la generación de archivos de datos tipo texto los cuales pueden ser aceptados por MATLAB.

Organización de datos

El programa inicia su ejecución al ser invocado el programa `ADFO.EXE`, el cual a su vez debe encontrarse junto con algunos otros archivos de datos en un subdirectorio dedicado. Como subdirectorios dependientes del primero, deben encontrarse los siguientes:

- **WAVES**. En este subdirectorio se almacenan los archivos en formato `.WAV` que se importarán a su vez hacia formato `.C30`.
- **C30**. Como es de suponerse, en este subdirectorio se almacenan todos aquellos archivos en formato `.C30` generados de diversas maneras. Se almacenan inclusive aquellos archivos generados de forma paramétrica que se consideran un subconjunto del mismo formato `.C30`.
- **TEXT**. Al ser exportado un archivo `.C30` hacia formato texto, el archivo de salida (con extensión `.txt`) se almacenará en este subdirectorio.

4.2.2 Definición de un formato de archivo

El algoritmo de compresión acepta como entrada archivos de enteros de 16 bits en formato binario, los cuales junto con un encabezado que se definió específicamente para los propósitos de este programa, constituyen un formato de archivo que se bautizó como formato `.C30`, con lo cual se pretende hacer referencia a la compatibilidad de los datos para su uso con la tarjeta evaluadora utilizada (EVM30).

Datos contenidos en el encabezado de archivo

A continuación se presenta el código en ANSI C que representa a la estructura en la cual se almacenan los datos que contiene el encabezado de un archivo .C30:

```
struct el_header{
char head[21];
char coding[5];
char uso[5];
int f_muestreo;
int l_datos;
char reservado[32];
}header;
```

Los campos de dicho encabezado, se utilizan como sigue:

- head. En este campo se escribe una cadena de identificación.
- coding. Aquí se especifica la forma de codificación, tiene únicamente dos variantes: "onda", que corresponde a archivos de forma de onda; "parm", que corresponde a archivos almacenados en forma paramétrica.
- uso. Este campo es utilizado para especificar el uso o procedencia de los datos contenidos en el archivo, como sigue: "sgen" se utiliza cuando el archivo fue producido por el generador de señales que contiene el programa; "mixd" se usa cuando el archivo procede de la función de mezclado; se escribe la cadena "grab" cuando el archivo fue grabado a través de la tarjeta EVM30; cuando se ha exportado desde un archivo .WAV de 8 bits se escribe la cadena "xprt"; si el archivo procede de un proceso de descompresión se identifica con la cadena "dcmp". Cuando el archivo contenga información en forma paramétrica, se le identifica con la cadena "LPC ".
- f-muestreo. Este dato de tipo entero codifica la frecuencia de muestreo, cuando sea aplicable.
- l-datos. En este campo se almacena el número de muestras que contiene el archivo. En el caso de un archivo almacenado en forma paramétrica, contiene el número de segmentos almacenados.
- reservado. Este espacio se deja en blanco cuando el archivo contiene una forma de onda. Cuando se trata de un archivo que contiene parámetros LPC, los dos primeros bytes de este campo contienen un entero corto que debe interpretarse como el número de coeficientes de reflexión con los que fué parametrizado el archivo. Los siguientes 30 bytes se continuan dejando en blanco para aplicaciones futuras.

4.2.3 Entrada de datos

Existen cuatro formas a través de las cuales se pueden obtener o generar archivos en formato .C30 que se utilicen como entradas a las funciones de compresión y descompresión:

1. Grabación de archivos utilizando la tarjeta EVM30. Dicha grabación puede realizarse a tres distintas frecuencias de muestreo: 8, 11.25 y 18 kHz, de manera que se asegure cierta compatibilidad con otro tipo de formatos (por lo general los archivos comerciales generados en formato `.WAV` han sido muestrados a 11.25 kHz o múltiplos de esta frecuencia). La ventana del convertidor A/D de la tarjeta puede especificarse a 3 ó 6 Vpp.

2. Exportación a partir de archivos en formato `.WAV` de 8 bits.

3. Generación de formas de onda senoidales, cuadradas o rampas mediante un generador de señales integrado en el programa.

4. Mezcla de cualesquiera dos archivos previamente generados en formato `.C30`.

4.2.4 Interfaz con la tarjeta EVM30

La tarjeta EVM30 se utilizó en este programa como medio de adquisición de datos y como plataforma de prueba para algunas rutinas en tiempo real. A continuación se describen las funciones que realizan intercambio de información con dicha tarjeta. En el apéndice B de este trabajo se describen con detalle los procedimientos para programar una interfaz desde un programa anfitrión.

Configuración de la tarjeta

Al iniciar el programa se carga en la tarjeta un programa con el cual interactúa el programa ADFO, a la vez que una función detecta la presencia de la tarjeta y avisa de ello al usuario. Se puede ejecutar un RESET por software a la tarjeta para reinicializarla.

Dado que la tarjeta puede ubicarse en cuatro regiones distintas del mapa de memoria de la computadora, existe una opción para cambiar la dirección base a partir de la cual se mapean los registros de comunicación. En la mayoría de los casos deberá utilizarse la dirección 0x240, sin embargo puede desplazarse hacia 0x280, 0x320 ó 0x340.

Con objeto de monitorear el estado de los registros de configuración, se incluyó la opción de observar su contenido.

Función de Osciloscopio

Con el objeto de facilitar la calibración de la entrada que se desee muestrear utilizando la tarjeta EVM30, se implementó una función de osciloscopio en la cual es posible modificar las opciones de frecuencia de muestreo y ventana del convertidor A/D. Un factor de normalización puede ser seleccionado para facilitar la observación de la forma de onda.

Carga de programas en la tarjeta

Para poder cargar distintos programas en la tarjeta sin tener necesidad de abandonar el programa, se implementó la función EVMLOAD que invoca a su vez al programa `EVMLOAD.EXE` que debe estar presente en el mismo subdirectorio que el programa ADFO.

4.2.5 Salida de datos

Como ya se ha visto, el programa genera básicamente dos tipos de archivos en formato .C30: archivos de forma de onda y archivos en forma paramétrica. Además de este formato, es posible generar a partir de un archivo de forma de onda en formato binario un archivo con las mismas muestras pero en formato de texto, que es aceptado por otros programas, específicamente por MATLAB.

Es posible también graficar los archivos de forma de onda con diferentes escalas para poder normalizar la salida. Dichas gráficas pueden tener salida en una impresora de matriz de puntos con ocho diferentes modalidades de impresión, combinando la dirección de impresión con la resolución elegida.

4.2.6 Análisis y Síntesis de los datos

La parte más importante del programa, es aquella en la cual se llevan a cabo las operaciones de compresión y descompresión o dicho de una manera mas formal: de análisis y síntesis.

Análisis

El análisis consiste básicamente en la obtención de los parámetros que definen a cada segmento analizado. En esta parte del programa se ejecutan rutinas que obtienen un valor para el *pitch*, la ganancia asociada y los correspondientes coeficientes de reflexión; asimismo se decide si el segmento es vocal o no vocal. Todas estas operaciones contienen opciones en las cuales es posible especificar variaciones en su ejecución, de manera que sea posible observar los cambios en el proceso de una manera gráfica a la vez que se pueden monitorear los resultados generados por las rutinas como son los coeficientes de autocorrelación, los candidatos a pitch y los coeficientes de reflexión.

El proceso de análisis conlleva en si varias etapas las cuales pueden ser monitoreadas de manera gráfica, como son la salida de un filtro pasobanda cuyas características se precisan mas adelante; así como el efecto de la aplicación de un filtro de preénfasis y una función de ventana.

Opciones modificables en el análisis

Dado que en muchos archivos de voz existen largos períodos de silencio se implementó la capacidad de poder adelantar un número variable de muestras y poder comenzar el análisis a partir de un segmento determinado; asimismo se puede determinar el número de segmentos a analizar. Sin embargo es posible interrumpir el proceso oprimiendo la tecla ESC al terminar el análisis de cualquier segmento. Es posible también imprimir la ventana gráfica oprimiendo la tecla F1. Con el objeto de hacer la salida gráfica mas comprensible puede modificarse un factor que alterará únicamente la graficación.

Los parámetros que es posible alterar son el número de etapas del filtro Lattice y el algoritmo utilizado para calcularlos, que pueden ser el algoritmo de Durbin o Schur, así como la acción de un detector de silencios el cual es útil para evitar la operación incorrecta del filtro Lattice.

Síntesis

En esta etapa existen muchos mas parámetros modificables que alteran el resultado final, y que son importantes para elevar la calidad de la reproducción de la señal. El procesamiento posterior de los coeficientes de reflexión, de la ganancia y del período de pitch obtenidos en el análisis puede ser modificado u omitido para observar el efecto de ello.

Opciones modificables en la síntesis

En cuanto al período de pitch obtenido en el análisis, su correcto postprocesamiento influye en gran medida para obtener a la salida una señal mas depurada y con menos cambios abruptos. Como parte inicial del postprocesamiento se puede omitir la correción de errores de decisión. Se pueden anular también tanto la acción de los filtros mediador y supresor de picos que tienen la función de uniformizar la secuencia de decisiones de pitch que reliza el sintonizador.

Una correcta interpolación de los coeficientes de reflexión y de la ganancia ayudan en gran medida a obtener a la salida una señal sin cambios demasiado rápidos en su contenido armónico y en el volumen. Estas opciones pueden no utilizarse para observar su resultado.

Como excitación pueden utilizarse tanto la secuencia D.O.D. [1], como una función impulso las cuales se aplican como entrada del filtro Lattice a la frecuencia especificada por el pitch obtenido.

Al inicio del proceso se especifican todas las opciones mencionadas las cuales serán válidas a lo largo del proceso de análisis/síntesis en los segmentos especificados.

Asimismo puede elegirse observar gráficamente los resultados intermedios de los procesos de análisis y síntesis, como son la salida del filtro paso-banda, la aplicación de la función de ventana o bién la salida de la forma de onda del decodificador.

4.3 Operación del codificador LPC

4.3.1 Segmentación del flujo continuo de muestras

El análisis LPC describe las características de la señal de voz a nivel de segmento, esto es, a lo largo de un conjunto corto de muestras, precisa las características de la señal cada 160 muestras, las que a continuación se referirán como *trama*. Es necesario también almacenar las últimas 80 muestras del segmento anterior dado que la función de ventana se aplica sobre un período de 240 muestras.

4.3.2 Preprocesamiento de la señal de entrada

El preprocesamiento de la señal consiste básicamente en dos acciones que se ejecutan sobre la señal de entrada.

[1] D.O.D.- Department Of Defense

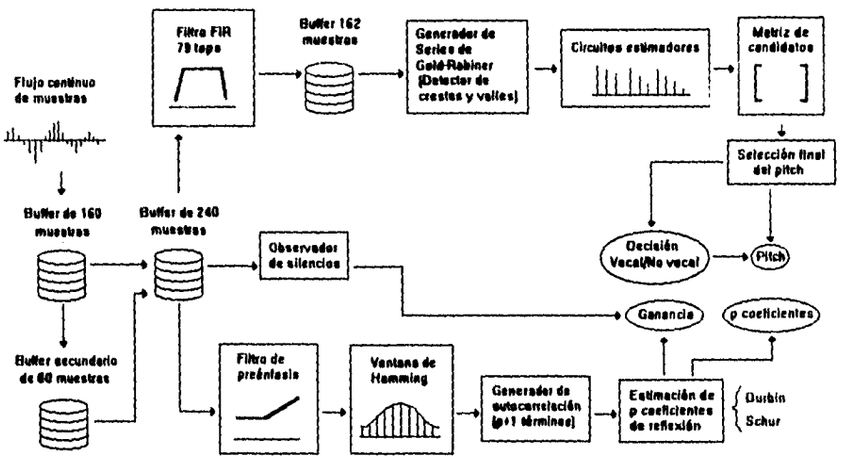

Figura 4.1: Estructura del codificador LPC

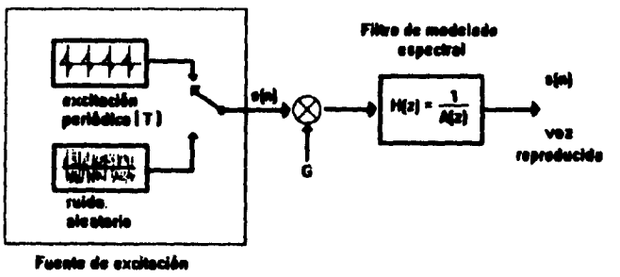

Figura 4.2: Modelo excitación/filtro de producción de la voz

a) Aplicación de un filtro de preénfasis. Por lo general la señal eléctrica de la voz posee un espectro en el cual contienen mayor energía las frecuencias bajas, por lo que es necesario elevar artificialmente la energía de las frecuencias altas. El encargado de llevar a cabo esta función es un filtro de preénfasis, cuya función de transferencia tiene un solo retardo y fue ya referida en el capítulo 1.

b) Aplicación de una función de ventana. Como en muchas otras aplicaciones de procesamiento de voz se aplicó la ventana de Hamming que sopesa las muestras dando mayor peso a la información que se encuentra al centro de la ventana, la cual tiene una longitud de 240 muestras.

4.3.3 Opciones para estimar los parámetros LPC

El modelo básico de producción de la voz que utilizan todos los métodos LPC es el modelo fuente/excitación que se presenta en la figura 4.2, donde la señal de excitación pasa a través del filtro cuya función de transferencia es $H(z)$ y que modela la envolvente espectral del tracto vocal. Dicho filtro contiene solo polos en el polinomio $A(z)$, esto es, un filtro predictor el cual obtiene el siguiente valor de la salida en base solamente a los valores previos de la señal de entrada (Ver capítulo 1).

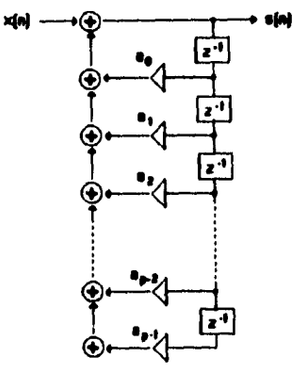

Figura 4.3: Implementación de un filtro IIR

Parámetros del filtro predictor $A(z)$

En un modelo clásico LPC son los coeficientes del polinomio $A(z)$ los parámetros necesarios para poder reproducir la señal a través de un filtro de síntesis ($H(z)$). Sin embargo cuando un modelo de predicción lineal se lleva a la práctica utilizando hardware especializado (DSP's), el manejo adecuado de estos coeficientes puede ser una tarea difícil pues no se encuentran normalizados y en ocasiones pueden exceder el rango aceptado en una plataforma en particular, en especial los dispositivos de punto fijo como son la primera y segunda generaciones de la familia TMS320.

Es por lo anterior que existen múltiples representaciones de los parámetros LPC, de manera que sea posible almacenarlos y manipularlos adecuadamente así como minimizar el efecto de los errores de cuantización.

Un filtro IIR convencional se ha mostrado en la figura 4.3, donde son los coeficientes $a_0 \ldots a_{p-1}$ los que definen al filtro y que se obtuvieron directamente del polinomio $A(z)$. Sin embargo un esquema mucho más conocido del filtro de síntesis es utilizando una implementación de forma cruzada o Lattice como se puede ver en la figura 4.4, donde el filtro se define mediante los coeficientes de reflexión, mejor conocidos como PARCOR (PARtial CORrelation), los cuales tienen la ventaja de estar siempre limitados en el rango de -1 a +1, lo cual es una característica especialmente atractiva para implementar este filtro en dispositivos de punto fijo.

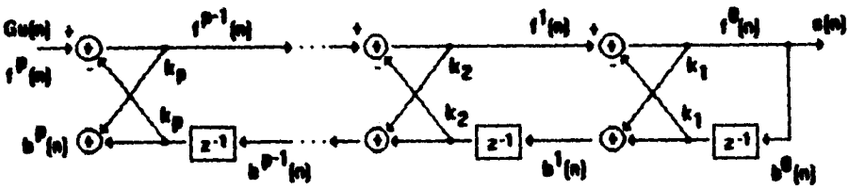

Figura 4.4: Implementación de un filtro Lattice

El filtro Lattice es la implementación más comúnmente usada del filtro de síntesis por lo que este suele describirse mediante los coeficientes de reflexión k_i. Existen varias formas para calcular dichos coeficientes, en base siempre a los coeficientes de autocorrelación $r(i)$.

Si se considera que un filtro predictor estima una muestra $\hat{s}(n)$ como una combinación lineal de las muestras pasadas, donde p es el orden del filtro (ec. 4.1).

$$\hat{s}(n) = -a_1 s(n-1) - \cdots - a_p s(n-p) \tag{4.1}$$

Siempre existirá un error entre dicha predicción y el valor real de la muestra $s(n)$. Para tratar de minimizar el error mínimo cuadrático (MSE) dado por la ecuación 4.2.

$$\sum_n (s(n) - \hat{s}(n))^2 \tag{4.2}$$

Se plantea el conjunto de ecuaciones lineales siguientes, para poder calcular los valores de los coeficientes $a_0 \ldots a_{p-1}$.

$$
\begin{aligned}
a_1 r(0) + a_2 r(1) + \cdots + a_p r(p-1) &= -r(1) \\
a_1 r(1) + a_2 r(0) + \cdots + a_p r(p-2) &= -r(2) \\
&\vdots \\
a_1 r(p-1) + a_2 r(p-2) + \cdots + a_p r(0) &= -r(p)
\end{aligned}
$$

Lo que puede ser expresado también en forma matricial

$$\overline{R} \cdot \overline{a} = -\overline{r} \tag{4.3}$$

donde

$$r^T = [r(1) r(2) \ldots r(p)] \tag{4.4}$$

$$a^T = [a_1 a_2 \ldots a_p] \tag{4.5}$$

y $\overline{R}$ es la matriz de Toeplitz

$$
\begin{bmatrix}
r(0) & r(1) & r(2) & \ldots & r(p-1) \\
r(1) & r(0) & r(1) & \ldots & r(p-2) \\
r(2) & r(1) & r(0) & \ldots & r(p-3) \\
\vdots & \vdots & \vdots & & \vdots \\
r(p-1) & r(p-2) & r(p-3) & \ldots & r(0)
\end{bmatrix}
$$

Y donde cada coeficiente de autocorrelación $r(i)$ está dado por

$$r(i) = r(-i) = \sum_{n=0}^{N-i-1} s(n) s(n+i) \tag{4.6}$$

Gráficamente esta operación se ha representado en la figura 4.5.

A partir de la ecuación 4.3 y dado que la matriz de Toeplitz siempre es inversible es posible calcular los coeficientes de $A(z)$ mediante la ecuación 4.7, esta se puede resolver a través de métodos numéricos como sería el de Gauss-Jordan.

$$\overline{a} = -\overline{R}^{-1} \overline{r} \tag{4.7}$$

Una forma de evitar la inversión de la matriz de Toeplitz y obtener tanto los coeficientes a_i como k_i los cuales corresponden a los filtros IIR y Lattice respectivamente, la aporta el método recursivo de Durbin.

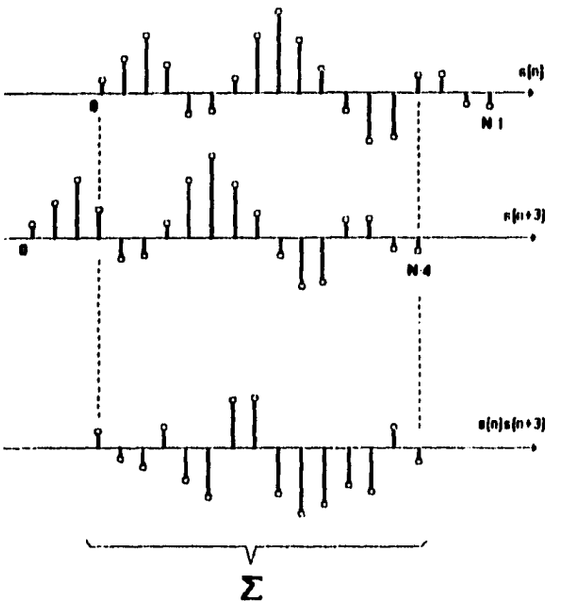

Figura 4.5: Representación del cálculo de los coeficientes $r(i)$

Método de Durbin

Este método utiliza una variable auxiliar $E(i)$ para calcular al mismo tiempo los coeficientes k_i y a_i. el término $E(p)$ es útil también pues a partir de él se obtiene la ganancia del segmento en análisis mediante $G^2 = E(p)$.

A continuación se muestra la secuencia iterativa que calcula los coeficientes deseados.

$$E(0) = r(0) \tag{4.8}$$

$$k_i = -\frac{r(i) + a_1^{i-1} r(i-1) + \cdots + a_{i-1}^{i-1} r(1)}{E(i-1)} \tag{4.9}$$

para $i = 1, \ldots, p$.

$$a_i^{(i)} = k_i \tag{4.10}$$

$$a_j^i = a_j^{(i-1)} + k_i a_{i-j}^{(i-1)} \tag{4.11}$$

para $j = 1, \ldots, (i-1)$

$$E(i) = (1 - k_i^2)E(i-1) \tag{4.12}$$

En el filtro Lattice de la figura 4.4 podemos ver que las ecuaciones que lo definen en todo momento están dadas por:

$$f^{(i-1)}(n) = f^{(i)}(n) - k_i b^{(i-1)}(n-1) \tag{4.13}$$

$$b^{(i)}(n) = k_i f^{(i-1)}(n) + b^{(i-1)}(n-1) \tag{4.14}$$

con

$$f^{(p)}(n) = Gu(n) \tag{4.15}$$

donde $u(n)$ es la excitación, $f^{(i)}(n)$ es la i-ésima reflexión hacia adelante y $b^{(i)}(n)$ es la i-ésima reflexión hacia atrás, siendo n el índice en el tiempo. El filtro Lattice aproxima el modelo del tracto vocal como una serie de ductos de diferentes diámetros a través de los cuales se desplaza la onda de presión proveniente de la laringe, como se refirió en el Capítulo 1 (figura 4.6).

Las ecuaciones 4.13, 4.14 pueden reescribirse para definir el filtro Lattice inverso de la siguiente manera

$$f^{(i)}(n) = f^{(i-1)}(n) + k_i b^{(i-1)}(n-1) \tag{4.16}$$

$$b^{(i)}(n) = b^{(i-1)}(n-1) + k_i f^{(i-1)}(n) \tag{4.17}$$

Así como el filtro Lattice convencional recibe como entrada la secuencia de pulsos provenientes de la laringe para obtener como salida la voz, el filtro Lattice inverso puede generar una señal de error a la salida teniendo como entrada la secuencia de muestras de la voz original. Esta señal de error puede ser utilizada como la excitación del filtro Lattice convencional de manera que la reproducción de la voz sea lo más fiel posible.

En el apéndice C se muestran los listados correspondientes a la programación de este método. En la parte de análisis del programa se muestran los coeficientes calculados tanto para el filtro IIR como para el filtro Lattice.

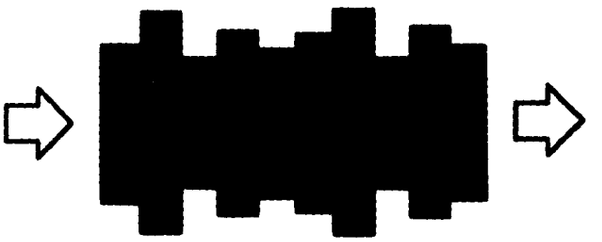

Figura 4.6: Aproximación del tracto vocal como ductos sin pérdidas

Método de Schur

Como una variante del método de Durbin, se hace uso del método de Schur. Dicho método se concibe principalmente para operar en arquitecturas paralelas. La ventaja del método de Schur es que el tiempo necesario para calcular los coeficientes de reflexión es proporcional a p, mientras que el método de Durbin requiere un tiempo proporcional a $p * \log(p)$. Este método evita el cálculo de variables intermedias generando directamente los coeficientes a partir de una matriz generadora de 2 renglones por p columnas. De hecho este método se utiliza en varias implementaciones comerciales como es el caso del estándar GSM.

En el apéndice C se muestra también el listado en ANSI C de este método.

4.3.4 Cálculo de la frecuencia fundamental (*Pitch Tracker*)

Si nos referimos al modelo inicial excitación/filtro, encontramos que para poder reproducir un segmento de voz son necesarios dos conjuntos de parámetros:

- Coeficientes del filtro de síntesis.
- Parámetros que definan a la excitación.

La obtención de los primeros se ha discutido ya con la suficiente amplitud. Determinar las características de la excitación requiere de un mayor cuidado debido a que está más sujeta a errores. Se codifican dos parámetros que definen a la excitación:

- El período de la secuencia de pulsos especificado en muestras.
- La ganancia asociada a dicho tren de pulsos.

El método utilizado en este caso es el algoritmo de Gold-Rabiner el cual aplica un complejo proceso estadístico para determinar el período fundamental que se busca, no sin estar aún libre de todo error.

Generación de series de Gold-Rabiner

Este método consta de seis series a través de las cuales se aplican diversos criterios con el fin de generar un conjunto de candidatos de entre los que se elegirá un valor adecuado para el período, o bien se decidirá si el segmento en proceso es vocal o no vocal, en cuyo último caso se utilizará una secuencia de números aleatorios como excitación del filtro.

Prefiltrado de la señal

La determinación inicial del período se realiza mediante un sencillo algoritmo de detección de crestas y valles en la forma de onda. Es necesario entonces que dicha forma de onda esté libre de armónicas en frecuencias más altas de los que puede ser la frecuencia fundamental. Es por esto que se antepone un filtro pasobanda con el objeto de suavizar la forma de onda que será analizada. Este filtro se fija con una banda de paso de 100-900 Hz y comúnmente se implementa como un filtro FIR o bien puede realizarse esta etapa de manera analógica.

En este caso se realizó el diseño de este filtro FIR con ayuda del programa MATLAB. El filtro consta de 79 taps y el listado que genera sus coeficientes, así como sus gráficas de respuesta en frecuencia se incluyen en el apéndice C.

El filtro puede probarse fácilmente generando señales senoidales con diferentes frecuencias y aplicándolas al filtro como entrada.

Este filtro fue probado en tiempo real utilizando la tarjeta EVM30. Dentro del programa ADFO puede ser cargado ejecutando la opción "Demo FPB". Puede verificarse su operación introduciendo en la entrada analógica de la tarjeta la señal proveniente de un generador de funciones y verificando con un osciloscopio en la salida analógica de la tarjeta.

Cálculo de las crestas y valles de la señal

En esta etapa se realiza la primera aproximación de los que serán candidatos a ser designados como período. Se utilizan seis criterios distintos para definir seis diferentes secuencias $m_1(n), \ldots, m_6(n)$ de números entre las cuales se decidirá el *pitch* correcto.

- $m_1(n)$ es la magnitud del pico.
- $m_2(n)$ es la distancia del pico al valle previo.
- $m_3(n)$ es la distancia del pico previo al actual. Si el pico anterior es menor al actual, entonces $m_3(n) = 0$.

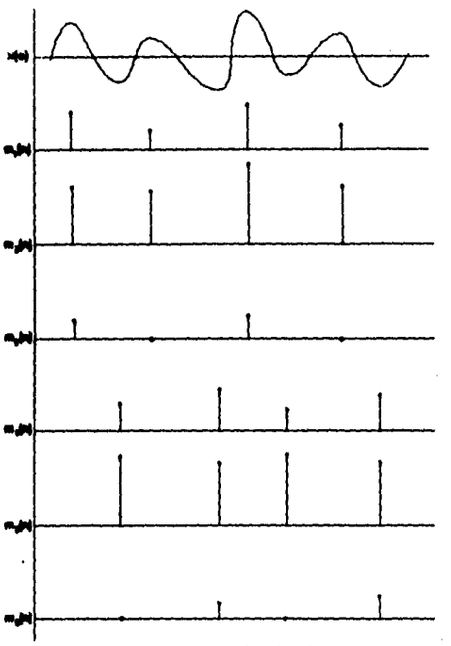

Figura 4.7: Ejemplo de medición de $m_1(n)$ a $m_6(n)$

- $m_4(n)$ es la magnitud del valle.
- $m_5(n$ es la distancia del valle al pico previo.
- $m_6(n)$ es la distancia del valle previo al actual. De nuevo, si el valle anterior es menor al actual, entonces $m_6(n) = 0$.

Depuración de los valores encontrados

El procedimiento es el siguiente:

Cada una de las secuencias $m_1(n)$ a $m_6(n)$ son procesadas dentro de un estimador que detecta el primer pulso encontrado, imponiendo a continuación un período t de "blanqueo" durante el cual ignora cualquier otro pulso que se presente. Seguido de dicho período t se genera una envolvente que decrece exponencialmente desde el último valor detectado del pulso. Si un nuevo pulso se presenta mientras la envolvente decrece y es mayor en magnitud a dicha envolvente, entonces dicho pulso reinicializa el proceso y se impone un nuevo período de "blanqueo". El intervalo en tiempo entre cada reinicialización del proceso se toma como el valor actualizado del *pitch*.

El período t así como el parámetro b de la exponencial se actualizan junto con el valor del pitch. Se calcula un período promedio P_{prom} a partir del valor previo y el valor actual

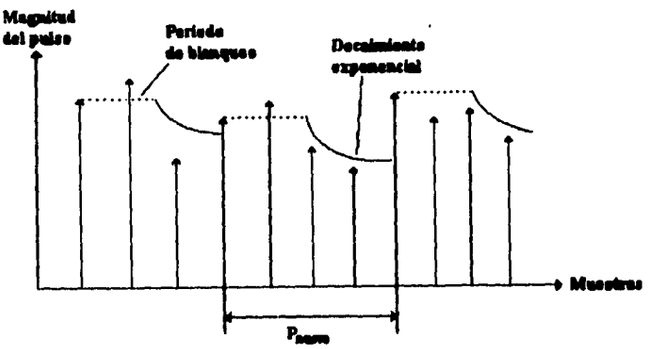

Figura 4.8: Circuito estimador del *pitch*

del *pitch*. El algoritmo no debe permitir que el valor de P_{prom} exceda el rango de 4 a 10 ms.

$$P_{prom}(n) = \frac{P_{prom}(n-1) + P_{actual}}{2} \tag{4.18}$$

Una vez obtenido el valor de P_{prom}, se recalculan t y b como sigue

$$t = 0.4 P_{prom} \tag{4.19}$$

$$b = \frac{Pprom}{0.695} \tag{4.20}$$

A la salida de cada uno de los seis estimadores se tiene ahora una secuencia depurada que será procesada de la siguiente manera.

Conformación de la matriz de candidatos

De cada una de las salidas de los seis estimadores se toman ahora los últimos tres valores calculados, en base a los cuales se generan otros tres valores para así formar un vector de seis

valores para cada uno de los estimadores. Esto se hace tomando en cuenta que la estimación original puede deberse a la presencia de segundas o terceras armónicas.

Para el i-ésimo estimador con P_{i1}, P_{i2} y P_{i3} se calculan

$$P_{i4} = P_{i1} + P_{i2} \tag{4.21}$$

$$P_{i5} = P_{i2} + P_{i3} \tag{4.22}$$

$$P_{i6} = P_{i1} + P_{i2} + P_{i3} \tag{4.23}$$

Con los seis vectores de seis elementos cada uno se genera una matriz de 6 x 6. En esta matriz cada uno de los elementos de la primera columna será comparado con los demás para determinar cual de ellos es lo suficientemente parecido a los otros como para ser escogido como el período válido. El cálculo de esta matriz y la determinación del período se repite para cada segmento.

Criterios de selección

Dependiendo del valor de cada uno de los elementos de la matriz, se establecen cuatro rangos dentro de los cuales puede estar dicho número. Cada uno de estos cuatro rangos tiene asociadas cuatro tolerancias como se muestra en la tabla siguiente

	-1	-2	-5	-7
1.6 - 3.1 ms	0.1	0.2	0.3	0.4
3.1 - 6.3 ms	0.2	0.4	0.6	0.8
6.3 - 12.7 ms	0.4	0.8	1.2	1.6
12.7 - 25.5 ms	0.8	1.6	2.4	3.2

Para cada uno de los 6 elementos de la primera columna de la matriz se realizan cuatro comparaciones contra los 35 elementos restantes de la siguiente manera:

Se llevan cuatro contadores de coincidencias, uno para cada una de las cuatro tolerancias que se establecen en cada rango. Entonces para cada elemento de la primera columna se cuenta el número de coincidencias dentro de cada uno de los cuatro rangos de tolerancia. Se tienen entonces asociados a cada elemento de la primera columna cuatro contadores, esto es, 24 contadores.

Dependiendo de la implantación, se toma un criterio para sopesar cada uno de estos contadores para compensar la tolerancia con la que se obtuvieron. Por ejemplo, para el contador con la menor tolerancia se le resta una unidad, al siguiente dos unidades, al tercero cinco y al cuarto siete unidades.

Una vez que se tengan los 24 contadores ya disminuidos, se busca el mayor de todos ellos y el elemento de la matriz asociado a él, será escogido como el valor válido del *pitch* en el segmento que se analiza.

Decisión vocal/no vocal

Una vez que se realizaron los cálculos anteriores, falta aún determinar si el segmento en análisis es vocal o no vocal.

Esto se determina comparando los cuatro contadores disminuidos asociados al número elegido contra un umbral experimental. Si el mayor de los contadores es mas grande que dicho umbral el segmento se declara vocal, de lo contrario se considera que el segmento es no vocal y como valor de *pitch* se codifica un cero.

Detector de silencios

Con objeto de incrementar el desempeño del codificador se implementó un detector de silencios que basa su operación en el cálculo de una diferencia mínima entre una muestra y la muestra adyacente. Si la diferencia no rebasa un umbral predeterminado, el detector declara el segmento como un silencio codificando una ganancia de cero. Su uso es necesario dado que el filtro Lattice presenta un comportamiento errático en condiciones como las anteriores.

4.4 Operación del decodificador LPC

En el decodificador se llevan a cabo otras operaciones además de la síntesis, que consiste simplemente en excitar al filtro Lattice. Las operaciones de postprocesamiento del pitch son importantes para elevar la calidad y continuidad de la forma de onda a la salida. En muchos casos la correcta aplicación de estas operaciones de depuración es mas importante aún que la obtención de los parámetros.

4.4.1 Parámetros de entrada del decodificador

El decodificador toma como entrada tres resultados que el codificador produce:

1. Un número entero que representa el período de pitch.
2. Un número de punto flotante que representa la ganancia asociada al segmento.
3. Un conjunto de p número de punto flotante que representan los coeficientes de reflexión del filtro Lattice.

4.4.2 Memoria de estados pasados e interpolación

Las funciones de interpolación, corrección de errores y aplicación de filtros requieren del conocimiento de los valores anteriores de los parámetros antes mencionados. Es por esto que se destinan variables para almacenar hasta los últimos tres valores anteriores de estos parámetros. De hecho, *el decodificador genera un retardo de un segmento* con el objeto de

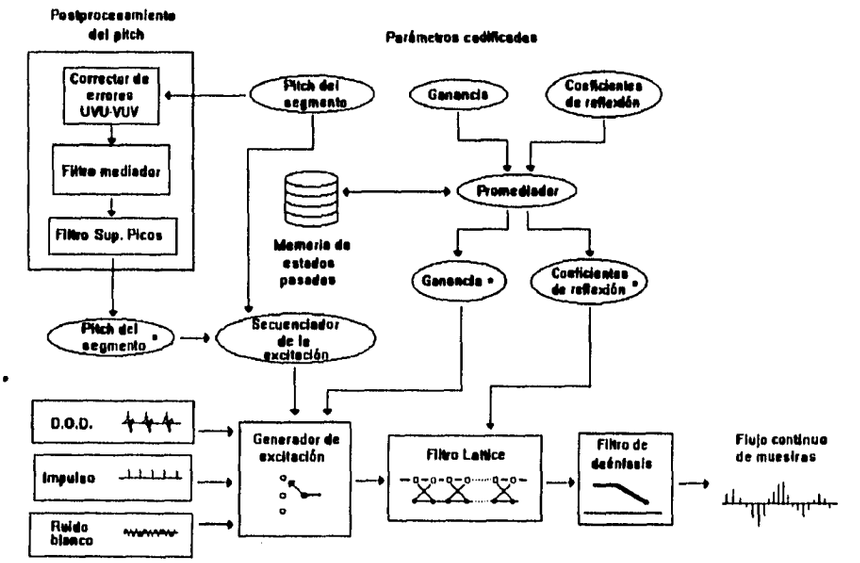

Figura 4.9: Estructura del decodificador LPC

poder interpolar los parámetros del segmento que analiza con respecto a los parámetros de los segmentos adyacentes. Los parámetros a interpolar son los coeficientes de reflexión y la ganancia. La interpolación se realiza para cada muestra que se vaya a obtener como salida del filtro Lattice, de manera que solo en la parte central de cada segmento se utilizan los coeficientes y ganancia tal cual fueron calculados por el codificador, y a medida que se calculan las muestras a los extremos del segmento se promedian los parámetros con los de los segmentos adyacentes.

4.4.3 Corrección de errores

Por lo general cuando se presenta un segmento vocal, este se encuentra dentro de una serie de segmentos vocales. Es muy poco probable que se presente un segmento no vocal entre dos segmentos vocales (VUV). Si lo anterior se presenta regularmente se identifica como un error. De manera similar si se presenta un segmento vocal entre dos segmentos no vocales también se identificará como un error que debe ser corregido (UVU).

Para el primer caso se realiza una interpolación entre los dos valores extremos de *pitch*. En el segundo caso basta con insertar un cero intermedio.

4.4.4 Aplicación del filtro mediador

No importando que tan robusto sea un sintonizador de pitch, la decisión última de pitch siempre está sujeta a errores y a variaciones demasiado grandes. Una forma de suavizar la curva de decisiones de pitch es aplicar un filtro mediador.

Su operación es muy sencilla, consiste únicamente en ordenar un número N de últimas decisiones de pitch de menor a mayor, y escoger entre ellas la que tiene el valor medio. Los valores típicos de N son entre 3 y 5. En esta implementación en particular se utilizó un filtro con N=3.

4.4.5 Aplicación del filtro supresor de picos

Cuando se presenta un valor de período fundamental muy disparado con respecto a los valores vecinos, es necesario aplicar un filtro FIR que uniformice la secuencia, como el siguiente:

$$H(z) = 0.25z + 0.5 + 0.25z^{-1} \tag{4.24}$$

Al igual que en el caso del filtro anterior este filtro sólo es aplicable cuando se presenta una secuencia de segmentos VVV (tres segmentos vocales consecutivos). La función del filtro supresor de picos (*smoothing filter*) es suavizar la curva de decisiones para hacer menos perceptibles los errores de cálculo de pitch.

En el programa es posible seleccionar la aplicación de uno de los dos filtros, deshabilitarlos a ambos, o bien hacerlos operar en cascada; esto es, el filtro supresor de picos toma como entrada la salida del filtro mediador.

4.4.6 Generador de excitación

Esta parte del programa cumple la función de generar un flujo continuo de pulsos que están destinados a actuar como la excitación del filtro Lattice. El generador de excitación debe:

- Secuenciar adecuadamente dicho flujo continuo de pulsos de manera que exista continuidad entre un segmento y otro.
- Utilizar los valores actualizados de ganancia y período de pitch sin perder la sincronía entre segmentos.
- Realizar las adecuadas interpolaciones de los valores de ganancia y coeficientes de reflexión para evitar cambios abruptos en la salida.

El generador de excitación puede excitar al filtro Lattice con tres distintos tipos de secuencias:

1. Secuencia de excitación D.O.D.

   Esta secuencia es un estándar generado para cumplir con la norma norteamericana que caracteriza al estándar LPC-10. Es una secuencia de 40 muestras determinada experimentalmente que caracteriza suficientemente bien el comportamiento de la laringe humana. En el apéndice C pueden consultarse sus valores característicos.

2. El impulso como excitación

   Como una idealización de la secuencia de pulsos provenientes de la laringe puede utilizarse la función impulso. Sin embargo su contenido espectral caracteriza de una manera menos precisa el funcionamiento del aparato vocal humano. Se incluyó esta opción en el programa de manera que existiera un parámetro de comparación con la secuencia D.O.D.

3. Ruido blanco

   Este se genera como una secuencia de números de magnitud unitaria y de signo aleatorio, multiplicada por la ganancia asociada al segmento.

Los primeros dos casos caracterizan a la excitación en los segmentos vocales, mientras que para los segmentos no vocales el ruido aleatorio simula la acción del flujo turbulento de aire en el tracto vocal (ver capítulo 1).

4.4.7 Aplicación del filtro de deénfasis

Posteriormente a la aplicación del filtro Lattice es necesario realizar el deénfasis de su salida para restaurar el espectro que fue artificialmente elevado en las frecuencias altas. Como se vio en el capítulo 1, el filtro de preénfasis posee sólo un retardo y puede ser aplicado casi simultáneamente con el filtro Lattice.

4.5 Tasas de codificación resultantes

Las tasas de codificación están directamente ligadas a la longitud asignada a cada tipo de datos en un compilador determinado. A la vez que cada compilador ha sido escrito para un microprocesador en particular.

En este caso se utilizó el compilador de Microsoft C versión 5.1, donde los enteros cortos tienen una longitud de 2 bytes y los números de punto flotante de precisión sencilla ocupan 4 bytes, se obtiene una tasa de codificación que puede ser calculada con el siguiente segmento de instrucciones en ANSI C:

```
BitRate = (sizeof(short int) * (1) + sizeof(float) * (p+1)) * 50 * 8;
```

donde es necesario codificar un entero (el período de pitch); p coeficientes de reflexión y una ganancia asociada los cuales son números de punto flotante. Considerando además que se utiliza una tasa de segmentación 20 ms, es decir 50 segmentos por segundo.

Calculando las tasas mínima y máxima obtenidas se obtiene:

- Tasa mínima de codificación. Cuando el filtro Lattice tiene un mínimo de 8 estados, 15.2 kbps.
- Tasa máxima de codificación. Generada con un máximo de 18 estados en el filtro Lattice, 31.2 kbps.

La diferencia fundamental entre ambos extremos es el contenido espectral que puede ser almacenado en los coeficientes. El número de armónicas que pueden ser reproducidas es mucho mayor a medida que se incrementa el orden del filtro.

La tasa de codificación máxima obtenida es aún un poco menor a otro estándares de codificación. Por ejemplo, ADPCM tiene una tasa de codificación de 32 kbps.

Si se utilizan tablas de búsqueda como en el caso de LPC-10, la tasa de codificación puede reducirse aún más, dado que solo sería necesario codificar el índice de una tabla en lugar de un número de punto flotante.

Conclusión

En el presente capítulo, se abordaron tanto las características particulares de la plataforma de desarrollo del algoritmo, así como la conformación final del sistema de codificación/decodificación obtenido. En el capítulo siguiente, se amplía el estudio de las implicaciones que tiene la implantación de un sistema como este en diversos esquemas de comunicaciones. En el capítulo 6 se presentan las conclusiones generales del trabajo aquí detallado así como algunas pruebas que se realizaron sobre él. Aquí se ha presentado un caso particular de un codificador LPC, sin embargo existe una gran cantidad de alternativas para modificar éste diseño, las cuales asimismo se discuten en el capítulo 6.

Referencias

[1] Panos E. Papamichalis: *Practical Approaches to Speech Coding*, cap. 4, 5 y 6, New Jersey, Prentice-Hall, Inc., 1987.

[2] Hamish Hubbard: *Inside the RIFF Specification*, Dr. Dobb's Journal, pp 38-45, Sept. 1994.

[3] Texas Instruments: *TMS320C3x, User's Guide*, Texas Instruments Inc., E.U.A., 1990.

[4] Texas Instruments: *TMS320C30 Evaluation Module, Technical Reference*, Texas Instruments Inc., E.U.A., 1991.

[5] Jutta Degener: *Digital Speech Compression*, Dr. Dobb's Journal, pp 30-34, Dec. 1994.

[6] Herbert Schildt: *C, Manual de Referencia*, Mc Graw-Hill, México, 2a. Edición, 1990.

[7] Kernighan, Ritchie: *El lenguaje de programación C*, Prentice Hall, 2a. Edición, México, 1988.

[8] Burrus,C.S. & Parks, T.W.: *DFT / FFT and Convolution*, E.U.A., John Wiley and Sons, 1985.

[9] Parks, T.W. & Burrus, C.S.: *Digital Filter Design*, E.U.A., John Wiley and Sons, 1987.

Capítulo 5

Aplicación del modelo en medios de transmisión digital

5.1 Implantación de un algoritmo de codificación/ compresión en un medio de transmisión digital

En capítulos previos, se ha estudiado con detalle las características y particularidades de los codificadores de voz, ya desde el punto de vista puramente matemático, ya desde la perspectiva del *hardware* utilizado para realizar dicha codificación. Sin embargo, la utilización de un algoritmo que haya demostrado su eficiencia en su fase de desarrollo y optimización, es tan solo una etapa de un largo proceso que culmina con su implantación exitosa en un sistema de comunicaciones.

Como ya es claro en este punto, se han referido las características de un modelo de codificación de voz, el cual al segmentar la secuencia de muestras extrae de cada segmento de dicha secuencia un conjunto de parámetros con los cuales podrá ser reconstruida en el extremo receptor. Hasta este punto, lo que se tiene es un *vocoder* que codifica la voz con calidad sintética, es decir, existe pérdida. Sin embargo, la deficiencia en la calidad de la voz se compensa con el bajo ancho de banda requerido en la transmisión de los parámetros de cada segmento, a la vez que se tiene mayor tolerancia a retardos propios del medio de transmisión.

5.1.1 Transmisión digital de voz

Tradicionalmente, al establecerse comunicación entre dos terminales y entablarse una conversación entre los usuarios que hacen uso de ellas, se establece un circuito bidireccional el cual no puede ser interrumpido mientras dure la conversación. Este es el enfoque tradicional *punto a punto*, sin embargo, con el desarrollo de la tecnología digital han aparecido otro tipo de servicios como la multiconferencia, en la cual un número considerable de personas pueden mantener una conversación simultánea. Otro servicio que se encuentra ya en operación en la mayoría de los conmutadores de uso privado (PABX) es el *paging* o servicio de localización de personas, mediante un *broadcast*[1] efectuado desde una terminal telefónica hacia un grupo

[1] *broadcast* es un término generalmente aceptado mediante el cual se hace referencia a una transmisión unidireccional punto-multipunto desde un emisor "central" hacia varias estaciones receptoras.

de terminales agrupadas en un área específica de un edificio.

El hecho de que se favorezca mediante la compresión la transmisión digital de la señal de voz, posibilita que los esquemas anteriores puedan ser aplicados en plataformas utilizadas típicamente para la transmisión de datos, como lo podría ser el caso de *Internet*. En párrafos posteriores se discutirá como caso de estudio la implantación de un teléfono a través de dicha infraestructura utilizando el algoritmo GSM 06.10 RPE-LTP como esquema de codificación. En las redes de datos actuales, basadas en su mayoría en tecnologías de acceso a un medio común como lo es el protocolo CSMA/CD[2], la calidad en el acceso a dicho medio común se halla supeditado al número de usuarios que intenten hacer uso de él en un momento dado, decrementándose dicha calidad a medida que el tráfico crece. Dado que los requerimientos para que se establezca un conversación telefónica incluyen una asignación estable de ancho de banda mientras dure la conversación, se hace indispensable el desarrollar tecnologías que permitan el establecimiento de conexiones telefónicas en redes con dichas características. En el futuro, se vislumbran mejores condiciones para la implantación de servicios sensibles a los retardos y a cambios en el ancho de banda, como son el audio y el video. La tecnología ATM[3] propone un sistema de transporte basado en la conmutación de circuitos, la cual, entre otras cosas, asegura un ancho de banda constante que favorece la transmisión de audio y video. Sin embargo, la migración de los esquemas de transporte actual hacia las tecnologías ATM no parecen apresurarse demasiado debido principalmente a la extensa infraestructura instalada y en funcionamiento. Al momento, parece mas factible desarrollar estrategias para librar de la mejor manera las limitaciones tecnológicas de la infraestructura actual, en vez de esperar a las nuevas tecnologías, aún en etapas de estandarización.

Obviamente, la calidad de la voz en una red de datos no podrá superar a la de la transmisión a través de una línea dedicada exclusivamente a ello, sin embargo, el desarrollar estrategias para transmitir voz junto con datos en una red propia para datos es una tendencia que se ve favorecida con el perfeccionamiento de los esquemas de codificación que incorporen compresión (esto es, a velocidades menores a los 64 kbps).

5.2 Transmisión en redes de conmutación de paquetes

La paquetización de la información digital ha resultado ser un medio eficaz para transmitirla en redes de datos. La tecnología utilizada para ello ha evolucionado dramáticamente en los últimos 20 años. Las mejoras se han dado en muchos aspectos: miniaturización de la electrónica hasta llegar a compactas soluciones VLSI[4], mejores medios de transmisión que se traducen en mayores anchos de banda y menores errores debidos al medio (uso extendido de la fibra óptica), mejora contínua en los protocolos y novedosos esquemas de switcheo (advenimiento de las tecnologias ATM).

Particularizando, en el caso de la transmisión de voz en una red basada en el empaquetamiento digital de la información -sin definir aún las particularidades de tal red-, son

[2]CSMA/CD : Carrier Sensing Multiple Access/ Carrier Detection
[3]ATM : Asyncronous Transfer Mode
[4]VLSI : Electrónica de muy alta escala de integración

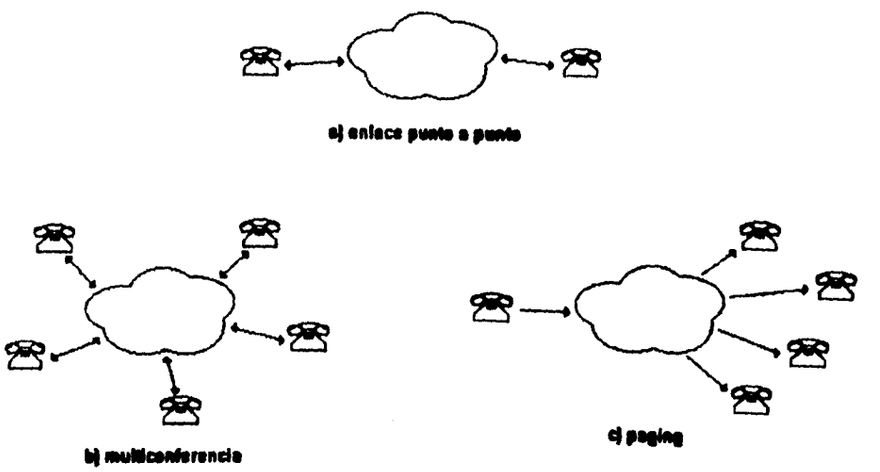

Figura 5.1: Diferentes servicios en transmisión de voz

necesarias las siguientes etapas, a manera de bloques, para que se pueda establecer un circuito de conversación entre dos usuarios de la red (figura 5.1):

Extremo transmisor:

- Elemento transductor (micrófono)
- Etapa de conversión A/D
- Etapa de codificación/compresión (vocoder)
- Etapa de paquetización
- Almacenamiento de los paquetes en un *buffer*

A continuación, el transmisor intenta enviar los paquetes a través del medio de transmisión entendiéndose para ello con los protocolos de acceso, los cuales dependen específicamente de la red en la cual se implementa el sistema.

En el extremo receptor, se conciben:

- Buffer almacenador de los paquetes
- Desempaquetamiento, resecuenciación e inserción de silencios
- Decodificación (vocoder)

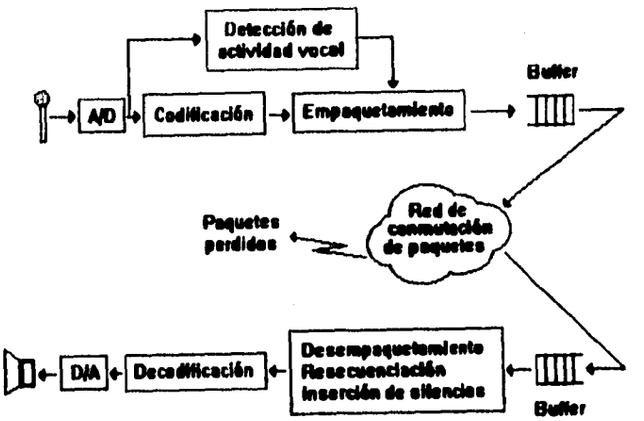

Figura 5.2: Manejo de una conversación por paquetes en una red

- Etapa de conversión D/A
- Transductor de salida (bocina)

Para transmitir los paquetes de información a través de la red, es necesario tomar cierto número de consideraciones que se reseñarán en secciones posteriores de este capítulo. Lo que debe quedar claro en este punto, es que la información es tratada básicamente como datos desde el momento en que esta abandona el buffer de salida hasta que es almacenada en el buffer del receptor. Esta etapa, que corresponde a los niveles más bajos del modelo de referencia OSI[5] consiste básicamente en electrónica que accesa el medio y traduce la información de señales eléctricas a paquetes de información (*transceivers* y puertos seriales).

Las etapas subsecuentes - empaquetamiento/desempaquetamiento, resecuenciación, decodificación - pueden ser realizadas por el vocoder mismo, mediante electrónica adicional (ASIC, por ejemplo), o bien un poco de este trabajo puede ser delegado al procesador del sistema anfitrión (por ejemplo, la etapa de resecuenciación). Para realizar la interfaz con el usuario, el sistema anfitrión debe efectuar algun tipo de interacción tanto con la electrónica que intercambia información con la red como con el vocoder. Dicha interacción está normada por un protocolo que puede ser manejado por un lenguaje de alto nivel (*drivers* para lenguaje C o Pascal).

[5]El modelo OSI (Open Systems Interconnection) establece las etapas del flujo de información en una red desde que esta es emitida por un usuario terminal, describiendo las transformaciones que sufre dicha información para viajar a través de la red y ser de nuevo recibida por otro usuario

5.2.1 Requerimientos para el tráfico de voz en una red

Se ha dicho que la voz se empaqueta para ser enviada a través del canal de transmisión y dentro de este recibe un trato similar al de los datos. Sin embargo, por la naturaleza propia de la señal existen otro tipo de exigencias para que la transmisión sea eficaz. En la tabla siguiente, se refiere una comparación de los requerimientos de diferentes medios para que estos puedan ser transmitidos, como lo son ancho de banda, tolerancia a los errores y tolerancia a los retardos.

Tipo	*Ancho de Banda*	*Naturaleza*	*Tolerancia a errores*	*Tolerancia a Retardos*
Datos (e-mail p.ej.)	0-10 Mbit/s	Ráfagas	Ninguna	Sí
Voz	64/32 kbit/s (s/c)	Periódica	Poca	No
Facsímil	0-64 kbit/s	Ráfagas	Ninguna	Sí
Video alta calidad	140 Mbit/s	Periódica	Alguna	No
Video calidad variable	0.5 - 30 Mbit/s	Ráfagas	Poca	No

En la actualidad, la mayor parte de las redes que manejan paquetes son dedicadas al tráfico de datos principalmente. Para ello, realizamos una comparación de los requerimientos de voz y datos para que su tráfico sea eficaz dentro de una red:

Tráfico de voz

- Se requiere de un bajo retardo a lo largo de la ruta recorrida por los paquetes, este requisito es vital para la correcta reconstrución de la señal.
- Se toleran rangos de error inaceptables para otras aplicaciones (BER[6]= 10^{-5} en ráfagas de hasta 2 ms de duración).
- Es absolutamente indispensable una asignación constante de ancho de banda dentro del canal de transmisión, de otra manera una conversación sería constantemente retardada o interrumpida - excepciones pueden ser hechas si se instrumenta detección de silencios, lo que lleva consigo ahorros momentáneos de ancho de banda-.
- El tráfico de voz solo es asequible y su implementación viable si todas o por lo menos la mayor parte de las terminales conectadas a la red tienen acceso a un adecuado ancho de banda en el momento que se requiere iniciar una conversación.

Tráfico de datos

- El retardo es por lo general poco importante (excepción hecha de las aplicaciones de control de la red misma).
- Cualquier tipo de errores imposibles de corregir son inaceptables.
- El tráfico se realiza mediante ráfagas, durante las cuales se presentan períodos donde el ancho de banda requerido se incrementa de manera poco predecible.

[6]BER : Bit Error Rate. La tasa de bits erróneos es un parámetro comúnmente evaluado en la tranmisión digital de información

- Existe poca restricción en cuanto al ancho de banda que deba tener disponible cada terminal, pues este puede ir desde el ancho de banda para líneas conmutadas (9.6 kbit/s) hasta varios Mbit/s.

- Sobre todo en redes de medio compartido (por ejemplo, Ethernet) durante los períodos de gran intensidad de tráfico la transferencia de información en las terminales se vuelve muy lento.

De hecho, el problema más significativo al intentar generar un esquema de transmisión de paquetes de voz es la presencia de retardos poco predecibles a lo largo de la ruta recorrida por dichos paquetes.

5.2.2 Fuentes de Retardo en una red

A continuación se reseñan las principales fuentes de retardo que afectan la transmisión de paquetes de voz en una red:

- El tiempo de transmisión requerido para la transmisión de cada paquete.

- Retardo de propagación a través del medio, el cual tiene también un valor fijo. Debe mantenerse menor a 24 ms para conexiones nacionales y menor a 300 ms para enlaces satelitales.

- Tiempo de procesamiento requerido por los codificadores, DSPs, etc. (i.e. *transcoding delay*).

- Retardo por conmutación de paquetes, el cual suele ser impredecible dependiendo del número de enrutadores, gateways y otros dispositivos de control de flujo que intervengan en la transmisión del paquete.

- Retardo por empaquetamiento, el cual está determinado por el tiempo necesario para llenar un paquete y depende del tamaño del paquete.

- Retardo por resincronización. Dada la aleatoriedad con que se presentan los retardos, es necesario que en el extremo receptor del paquete se lleve a cabo una resincronización de los paquetes para que estos puedan ser decodificados y reproducidos de manera síncrona.

En una LAN[1] típica, en la cual se desee transmitir voz no comprimida y se disponga del ancho de banda necesario, la suma de los retardos en un solo sentido debe mantenerse por debajo de 5 ms. En el caso de voz comprimida este límite se eleva hasta cerca de 20 ms, sin embargo la pérdida de un paquete tiene mayor significado pues contiene información de un período mayor de la señal (de hecho, este parámetro está directamente relacionado con la *tasa de segmentación* de la cual se habló en el capítulo 1).

El problema del retardo suele agudizarse cuando la conexión se efectúa entre terminales que no se encuentran dentro de la misma LAN, esto es, en una MAN[2] o una WAN[3].

[1] LAN: Local Area Network
[2] MAN: Metropolitan Area Network
[3] WAN: Wide Area Network

5.2.3 Influencia de los errores en la calidad de la transmisión

Por lo general, cuando no se utilizan complicadas técnicas de corrección de errores, un bit equivocado en un paquete significa la pérdida de este. En muchos casos, cuando una falla aparece en un paquete la red entrega dicho paquete marcado como erróneo. Cuando se descartan o se pierden paquetes dentro de una ráfaga o período de habla, aparece una discontinuidad en la señal reconstruida. Según estudios al respecto [Gruber and Le, 1983] se considera que la pérdida de paquetes es imperceptible si dicha pérdida representa menos del 1% de la ráfaga y tiene una duración máxima de 5 ms. Otros estudios consideran que una pérdida del 2 al 5% de los paquetes dentro de una ráfaga son aceptables si se utilizan técnicas de interpolación para recalcular las muestras perdidas.

Sin embargo, cuando se utilizan técnicas de codificación a menores tasas de codificación - el caso de los codificadores basados en la predicción lineal-, la pérdida de un paquete tiene efectos mucho más considerables al intentar reconstruir la señal. Es por esto que las técnicas de codificación que incorporan compresión exigen del medio de transmisión una tasa de errores (BER) mucho menor que la acostumbrada en canales de voz no comprimida (a 64 o 32 kbit/s).

5.2.4 Estrategias utilizadas para incorporar servicios de voz en redes de conmutación de paquetes

Se ha visto ya que la principal ventaja que aportan las técnicas de compresión de voz es una reducción en el ancho de banda requerido para entablar un enlace. Sin embargo se han referido también los requisitos que impone este medio para ser transmitido dentro de una red. Es necesario entonces implementar un "protocolo de voz" que brinde las condiciones necesarias para que dicha transmisión se efectúe exitosamente. Este protocolo de voz debe estar ubicado como un protocolo que medie el acceso del codec de voz con los protocolos propios de la red. Dicho protocolo deberá atender las siguientes áreas:

Empaquetamiento. Para realizar un empaquetamiento óptimo deben tomarse en cuenta diversos factores. Se prefiere un tamaño de paquetes por lo general pequeño para minimizar el efecto de los retardos y de la pérdida de paquetes. Pero es también cierto que un tamaño pequeño de paquetes contribuye a sobrecargar la red. En algunos casos es deseable poder variar la tasa de codificación para poder transmitir en períodos de alta congestión. Otra alternativa es variar el número de bits por paquete, aunque esto causa serios problemas de sincronización en el receptor. Por lo general el tamaño de los paquetes es fijo y se transmiten en períodos de 10 a 20 ms.

Asignación de prioridad. En el caso de tráfico de señales de voz, es necesario asegurar que se le asigne una prioridad correcta en comparación a otros medios como datos o video de manera que, por ejemplo, una ráfaga de datos se detenga momentáneamente en los instantes en que un paquete de voz tenga prioridad.

Resincronización. Esta operación es absolutamente indispensable que se realice en el receptor, pues los retardos generados en la transmisión de cada uno de los paquetes es variable y se almacenan en el buffer del receptor de una manera poco periódica. La resincronización consiste en generar un espaciamiento uniforme en el tiempo al reconstruir la señal en base a

los paquetes recibidos.

Sustitución de paquetes perdidos. De hecho, existen varias técnicas para sustituir información perdida: la más simple de ellas es insertar silencios, puede sustituirse por ruido blanco de baja intensidad, repetir el paquete previo o bien realizar una interpolación en base a la frecuencia fundamental (*pitch*).

5.2.5 Alternativas de simulación

Es claro ahora que la implantación de un algoritmo de codificación, y más aún, de un protocolo para la transmisión de voz en una red basada en la conmutación de paquetes es un proceso complicado que requiere herramientas que abarquen la gran cantidad de factores que intervienen en el diseño de un protocolo para la transmisión de voz.

Existe software CAE[7] comercial y disponible para simulaciones de este tipo, con dos principales tendencias en cuanto al enfoque de la simulación. Por un lado está el enfoque tradicional basado en eventos, el cual da una visión macroscópica de la red utilizando técnicas estadísticas. Su principal virtud radica en que al observar la red de esta manera disminuye la carga de cálculos y es posible modelar redes de mayor envergadura. El enfoque contrario atiende el desarrollo de eventos en la red tomando en cuenta la secuencia de muestras dentro de cada canal, lo que incrementa en gran medida la carga computacional para llevar a cabo una simulación. Este enfoque por lo general tiene necesidad de hardware más potente que acelere la ejecución de los cálculos, por lo que el uso de DSPs es común para llevar a cabo esta tarea.

Como ejemplo de la primera aproximación pueden mencionarse "BONeS Designer" de Comdisco Systems Inc., "COMNET III" de CACI Products Company y "OPNET" de MIL 3 Inc. Todos ellos proveen interfaces gráficas amigables y operan tanto en estaciones de trabajo como en computadoras personales. Por otro lado, un paquete que aborda el problema por la segunda alternativa es SPW.

Una simulación mixta utilizando ambas aproximaciones puede aportar soluciones más verídicas. En el capítulo 17 de [1] se reseñan los resultados de una simulación mixta, en la cual se simuló la transmisión de voz codificada con ADPCM sobre una red basada en Frame Relay[8]. Los resultados mostrados son interesantes en cuanto a que fue posible simular el efecto de la sobrecarga en la red y de la pérdida de paquetes sobre la señal reconstruida en un extremo receptor.

Existe de hecho un borrador de recomendación de CCITT para el diseño de protocolos de voz en redes de este tipo: la recomendación G.PVP[9] define un protocolo para transmisión de voz en redes de banda ancha. La voz es transmitida en paquetes de 128 muestras, a la vez que cada paquete se divide en bloques, conteniendo bits de la misma significancia. Según dicha recomendación, en los momentos de mayor congestión en la red pueden descartarse

[7] CAE : Computer Aided Engineering, es un término utilizado genéricamente para referirse a software de ayuda en procesos de ingeniería.

[8] *Frame Relay* es una tecnología de conmutación de paquetes considerada como la base para la implementación de redes bajo el esquema ATM

[9] PVP : Packetized Voice Protocol, reporte COM XV-R 21, Enero 1990.

hasta 3 bloques con los bits menos significativos de cada uno de estos paquetes. Esta es una solución a las sobrecargas momentáneas de la red a expensas de una reducción en la calidad de la señal. En esta recomendación no se define aún una estrategia para sustitución de paquetes perdidos.

5.3 Ubicación de los protocolos de voz en una arquitectura de red

Para que una red pueda funcionar adecuadamente de manera que todos sus usuarios tengan acceso eficiente a ella, es necesario definir una serie de reglas que deben ser seguidas por todos ellos. A dichas reglas se les llama *protocolos*, y para reducir la complejidad de dicho sistema de reglas, se les divide en siete *capas* siguiendo la recomendación ISO-OSI[10]. A tal conjunto de capas se les llama usualmente *arquitectura de red*, y con el objeto de dar cabida a diferentes tipos de servicios, en algunas de ellas se deben tomar ciertas consideraciones especiales, como en aquellos servicios que son sensibles a los retardos como es el caso de la voz y el video. La configuración y funciones de dichas capas pueden variar dependiendo de la implementación, sin embargo se hace mención de ellas para referir posteriormente los requerimientos de un protocolo de voz adecuado para la transmisión de la misma.

5.3.1 Protocolos de red según el modelo ISO-OSI

No. 1: Capa física. El protocolo de más bajo nivel se halla en la capa física, su función es asegurar que una conexión se haga posible a través del canal de transmisión. Se hace cargo también de que dicha condición se realice a una velocidad óptima tal que el rango de errores sea aceptable. Esto es, en la capa física se lleva a cabo la *señalización* entre el usuario y la red.

No. 2: Capa de enlace de datos. Los protocolos ubicados en este nivel se encargan de llevar a cabo el intercambio de mensajes entre lo que la recomendación de ISO llama IMP[11]. Un IMP transmisor toma los datos enviados por un DCE[12], los empaqueta y los transmite a un IMP receptor. El IMP receptor recibe el paquete, lo desempaqueta y lo pasa al DCE que espera la información. En esta capa usualmente se lleva a cabo un control de errores mediante la repetición de los paquetes erróneos, a través de un intercambio de mensajes entre los IMPs implicados.

No. 3: Capa de red. En la capa de red se lleva a cabo el *ruteo* de los paquetes, que tiene por objeto controlar la congestión tanto en la red como en los IMPs receptores. La congestión en la red se presenta principalmente cuando demasiados usuarios accesan el medio de transmisión y el ancho de banda disponible se reduce. La congestión en un IMP se debe a que dicho dispositivo tiene un buffer de tamaño limitado cuya capacidad puede excederse si se almacenan en él demasiados paquetes antes de que este pueda entregarlos a su correspondiente DCE. Un ejemplo de un protocolo que abarca las tres primeras capas de este modelo es el estándar X.25 de CCITT.

[10]ISO-OSI : International Standards Organization - Open Systems Interconnection
[11]IMP : Interface Message Processor
[12]DCE : Data Communications Equipment

No. 4: Capa de transporte. Los protocolos que proporcionan el servicio de transporte se ejecutan *en el interior de una computadora*, es decir, en un *Host*. Los protocolos de transporte se aseguran que los paquetes de información sean entregados en orden, sin pérdida o duplicación. Además, la capa de transporte funciona como un multiplexor, enviando a cada aplicación que ejecuta en el host los paquetes que le correspondan. Los programas que efectúan las funciones de estos protocolos tienen acceso a los recursos del sistema como son archivos, puertos, etc.

No. 5: Capa de sesión. La capa de sesión (también llamada capa de conexión) se usa para permitir a los usuarios identificarse cuando desean accesar a la red. En muchos casos prácticos la capa de sesión ha sido absorbida por la capa de transporte.

No. 6: Capa de presentación. La capa de presentación ejecuta dos funciones principales: encriptamiento y compresión. El encriptamiento suele usarse cuando la información a ser transmitida no debe ser entendida más que por el receptor, por lo tanto este último debe tener la capacidad de realizar un desencriptameinto para entender el mensaje. La compresión de datos se lleva a cabo debido a que por lo general el uso del medio de comunicación (línea telefónica, línea privada, etc.) resulta ser costoso. En general, cualquier flujo de datos puede ser comprimido en el momento de ser enviado (*run lenght coding*) mediante técnicas que identifican patrones repetitivos y los reemplazan por patrones mas cortos.

No. 7: Capa de aplicación. En la capa de aplicación, como su nombre lo indica, se encuentran los programas que ejecutan alguna aplicación específica, como son el correo electrónico, transferencia de archivos, etc.

Como puede apreciarse, los requisitos de un protocolo de voz intervienen en la mayoría de los protocolos definidos según el modelo anterior. En las tres primeras capas de dicho modelo, los datos, cualesquiera que sea su naturaleza son tratados de igual manera, y la prioridad en ancho de banda que pueda tener un servicio sobre los demás depende principalmente de la tecnología utilizada para transportar los paquetes. En este sentido, notablemente ATM provee mejores condiciones para el transporte de dichos paquetes, dado que el establecimiento de canales virtuales asegura que el ancho de banda permanecerá asegurado sin importar el tráfico en la red.

A partir de la cuarta capa del modelo OSI, los protocolos se manejan en el interior de un host que ejecute una aplicación de transmisión de voz. En este sentido es importante hacer notar que la eficiencia de un protocolo de voz dependerá en gran medida de las caraterísticas y arquitectura interna de *la plataforma* sobre la cual se desarrolle una aplicación de este tipo. En una sección posterior se discutirá la implementación sobre las dos plataformas más populares: Unix y Windows. Siguiendo los lineamientos de este modelo, un algoritmo de compresión de voz, tema de esta tésis, estaría ubicado en la sexta y séptima capas de protocolos, mientras que la prioridad necesaria de un servicio de voz sobre otros servicios tales como datos, debería manejarse en las capas cuarta y quinta. Sin embargo, en TCP/IP, el conjunto de protocolos más popularizado mundialmente, y que sería la base para el establecimiento de un estándar de protocolo de voz, las siete capas del modelo OSI se han simplificado y aglomerado de una maner más práctica. A continuación se refiere dicha *suite de protocolos* de una manera general.

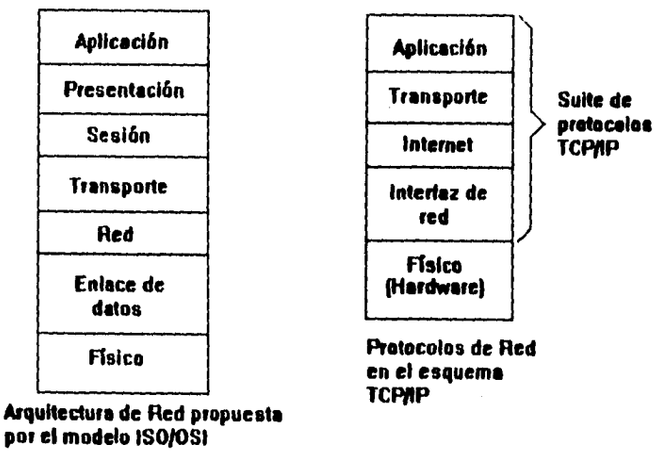

Figura 5.3: Arquitecturas de Red: modelo ISO/OSI y TCP/IP

5.3.2 Protocolos sobre TCP/IP

TCP/IP[13] es, de hecho, el protocolo utilizado para realizar la interconexión de una cantidad inmensa de redes universitarias, de investigación y con muy diversos propósitos en la red de datos más grande que existe: Internet. La suite de protocolos TCP/IP funciona sobre una base simplificada de cuatro capas (figura 5.3) : Aplicación, Transporte, Internet e Interfaz de red. En dicha suite de protocolos no se considera el hardware, esto es, el medio físico a través del cual se haga el intercambio de paquetes. Esto permite que la tecnología de dicho medio físico pueda evolucionar sin que sea necesario realizar modificaciones de fondo en los protocolos. En la capa más baja de la suite TCP/IP -Interfaz de red-, se realiza el empaquetamiento de datos en una *trama*, que se entrega al hardware encargado de la transmisión, como pueden ser módems, hubs, etc. El protocolo IP, ubicado en la capa Internet, se hace cargo del direccionamiento o *ruteo* de los paquetes. Esto es, lo ejecutan los *ruteadores*, los cuales añaden a los paquetes una *dirección IP* que sirve para que el paquete sea reconocido por los puertos de los ruteadores y enviado a su destino final. En sí, la estructura de Internet está formada por un sinúmero de ruteadores que interconectan entre sí a diferentes *segmentos* en todo el mundo. El protocolo TCP, que opera en la capa de transporte, proporciona una forma robusta de establecer comunicación entre aplicaciones. Este protocolo incluye en los paquetes un número de puerto asociado a cada aplicación, de manera que distribuye (o multiplexa) los paquetes en diferentes aplicaciones ejecutándose al mismo tiempo. Protocolos de este tipo se ven favorecidos por plataformas que puedan

[13]TCP/IP : Transmission Control Protocol/ Internet Protocol

administrar múltiples hilos de ejecución, como es el caso de Unix. En el nivel de aplicación, han conseguido estandarizarse varios protocolos que manejan los servicios básicos en Internet. Como ejemplos pueden citarse SMNP[14] el cual se utiliza con el fin de conocer los parámetros de desempeño de la red y monitorearla; y SMTP[15] el cual define las reglas que permiten el intercambio de correo electrónico (e-mail).

Si se desea desarrollar una aplicación de voz para Internet, el principal obstáculo a vencer es precisamente el precisar un protocolo de voz consistente que opere entre las capas de aplicación y transporte. *Hasta el momento no existe un protocolo de este tipo que sea usado de manera generalizada* como son utilizados, por ejemplo, los protocolos de correo electrónico. Esto se debe en parte a que en un principio, el transmitir voz o datos a través de Internet era poco factible por el poco ancho de banda disponible. En el estado actual de uso de los protocolos sobre TCP/IP, lo más factible es desarrollar aplicaciones que cumplan con ellos de la forma más estándar posible. Un programador que desee interactuar con dichos protocolos se encontrará bajo la siguiente situación: es necesario interactuar con un programa residente que habilite las funciones de la capa de transporte. A dichos programas se les conoce con el nombre genérico de *sockets* y su función es realizar el enlace entre un programa de aplicación el cual accesa a los diferentes puertos (asociados a dicha aplicación en particular), comunicándose con el socket mediante mensajes. La eficiencia en la utilización de los sockets se halla ligada al entorno de programación utilizado así como a la plataforma (la eficiencia es mucho mayor cuando dicha plataforma proporciona un real ambiente multitareas). En la siguiente sección se analizan las implicaciones al desarrollar aplicaciones sobre las dos plataformas de uso más extendido: Windows y Unix. Existen otras plataformas que también reúnen las condiciones para aplicaciones de este tipo, como es el caso de OS/2, sin embargo, dado que su uso es poco generalizado no existen el mismo número de aplicaciones escritas para ellos y el acceso a ellas es más restringido.

5.3.3 Aplicación de un protocolo de voz

Como es sabido, la red de datos más grande del mundo, Internet, está constituida sobre una infraestructura heterogénea con anchos de banda en extremo variables, lo que exige de las aplicaciones que sobre ella ejecuten un manejo cuidadoso y económico de dicho recurso. Muchos de los usuarios asiduos de Internet tienen acceso a ella a través de módems telefónicos a velocidades de 9600 y 14400 bps vía SLIP[16]. Es por esto que un servicio de Transmisión de voz a través de Internet deberá incorporar algún algoritmo de compresión para hacer extensivo este servicio a la mayor cantidad de usuarios posible.

Para estaciones de trabajo bajo ambiente Unix existe una amplia gama de aplicaciones de voz las cuales interactúan de una manera natural con los *sockets de Berkeley*. Todas ellas siguen arquitecturas de diseño similares en cuanto a programación se refiere. En seguida se resume el pseudocódigo de dichas aplicaciones operando bajo un esquema cliente-servidor:

[14]SMNP : Simple Management Network Protocol
[15]SMTP : Simple Mail Transfer Protocol
[16]SLIP : Serial Line Internet Protocol

```
CLIENTE
     while(!fin_de_conversacion)
      {
      graba_buffer   /* adquiere segmento de muestras  */
     /* de la tarjeta digitalizadora   */
      comprime_buffer /* aplica algoritmo de compresion */
      conecta_sockettcp_servidor
      establece_parametros_de_protocolo
      envia_buffer
      desconecta_sockettcp_servidor
      }

 SERVIDOR
     while(1)
      {
      espera_conexion
      if (conexion)
       {
       establece_parametros_de_protocolo
       recibe_buffer_comprimido
       descomprime_buffer
       reproduce_buffer
       }
      }
```

Para establecer un protocolo de voz que sea útil a la mayor cantidad de usuarios posible, incluyendo la base ya instalada de aplicaciones en plataforma Unix, debe diseñarse una aplicación que sea compatible con dichas aplicaciones además de extender sus beneficios a la gran cantidad de usuarios de computadoras personales IBM-PC. Esto lleva forzosamente a desarrollar una aplicación en ambiente Windows y que interactúe con su interfaz hacia TCP/IP: los *Windows Sockets*.

El desarrollo de aplicaciones que manejen eventos en tiempo real, como lo son el audio y el video, se ven favorecidas en gran medida porque Unix es un ambiente multitarea que soporta múltiples hilos de ejecución. En las aplicaciones de red en esta plataforma un programa que actúa como "administrador vigilante" en la computadora servidor, se encuentra en espera de que se inicien conexiones con una serie de clientes previamente conocidos. Al establecerse una de estas conexiones el sistema asigna a un socket que trabajará individualmente con el cliente en turno. Esta elegante y simple arquitectura de programación no tiene paralelo en el ambiente Windows. En este otro caso la ejecución del programa se basa en respuestas a eventos o mensajes específicos provenientes del sistema operativo. En realidad, Windows no es un ambiente multitareas, debido a que la atención a dichos mensajes detiene por completo la sesión del sistema operativo. Se puede hacer una analogía en la programación en ensamblador cuando se realiza la atención de una interrupción por hardware mientras se han deshabilitado las demás interrupciones. Para interactuar con TCP/IP, en Windows se utilizan un conjunto de rutinas de alto nivel conocidas como VBX que realizan la interfaz

con el *Windows Socket 1.1*. Asimismo, es necesaria también una interfaz de alto nivel con el software digitador. Para este último caso es común el uso de las rutinas MCI[17].

Las primeras aproximaciones en programación en esta dirección han demostrado poca eficiencia, debido a los inevitables retardos al atender los mensajes del sistema. Así también la naturaleza *half-duplex* de los manejadores de hardware digitador imponen la condición de "solo hablar" o "solo escuchar" a la aplicación.

En el programa IGP (ver parte final de este capítulo), los problemas anteriormente expuestos se solucionaron de la manera siguiente:

- Los algoritmos de adminstración del protocolo y de compresión se ejecutaban de una manera segmentada para permitir la atención de los mensajes del sistema en los intervalos entre cada ejecución segmentada.

- Se llevó a cabo una programación de bajo nivel tanto de los Windows Sockets como de los controladores de audio, con objeto de obtener mayor flexibilidad en cuanto a retardos.

- La programación de la aplicación se realizó en Visual C++, el cual, a diferencia de otras alternativas como Visual Basic, permite al programador mayor control sobre las funciones básicas del sistema operativo.

Un programa de aplicación ejecutando en un ambiente cuya naturaleza no sea multitareas, necesariamente deberá realizar un "switcheo" entre dos tareas: la atención de los mensajes del sistema operativo; y la adminstración del protocolo de red y la compresión de los segmentos de voz. Es claro que mientras exista, como se ha señalado, una amplia base instalada de máquinas ejecutando bajo DOS y Windows, se deberán seguir buscando estrategias de programación como esta de manera que una aplicación realmente sea de uso generalizado.

5.4 Aplicación de la compresión de voz en enlaces satelitales

En esta sección se tratan las aplicaciones de la compresión de voz en redes que utilizan el satélite como medio de transmisión. Particularmente, se estudia aquí el caso de las redes privadas constituidas por un número de terminales remotas o *VSAT's*[3], las cuales, bajo el esquema tradicional que las concibe, son coordinadas a través de una estación centralizada o *hub*.

Existen básicamente tres tecnologías de acceso al satélite que son utilizadas en redes VSAT: TDM/TDMA, TDMA y SCPC. En la actualidad, y debido al desarrollo de tecnologías de monitoreo y optimización de redes, ha surgido otro término en este campo: las redes *VSAT-Plus*, las cuales incorporan dichas tecnologías para realizar una asignación más dinámica de los recursos satelitales.

En las redes VSAT tradicionales el uso de compresores de voz es ya un hecho conocido. Sin embargo bajo los nuevos esquemas de administración de red que se manejan en las redes

[17]MCI : Media Control Interface

[3]VSAT : Very Small Aperture Terminal. Terminales satelitales con muy pequeña apertura, esto en relación al tamaño de la antena requerido y la potencia necesaria.

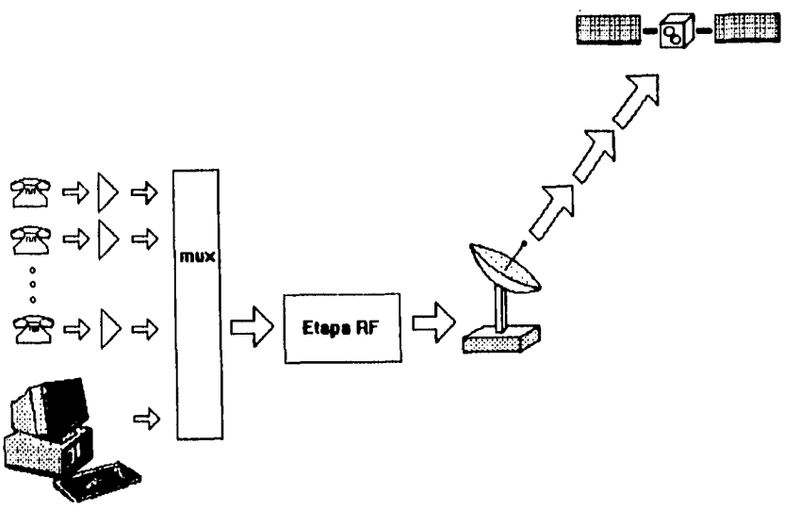

Figura 5.4: Inserción de compresores de voz en una estación VSAT remota

VSAT-Plus, los compresores de voz tienen una participación mucho más dinámica en la optimización del recurso satelital. Bajo la supervisión de un software especializado denominado genéricamente NMS [1] la tasa de codificación puede variar en función del tráfico instantáneo en toda la red. Asimismo el software de administración establece canales virtuales cuando se entabla una conversación y libera el ancho de banda correspondiente cuando dicha conversación termina. En la figura 5.5 se muestra un ejemplo de cómo se reconfigura la asignación de ancho de banda en función del tráfico a distintas horas del día. De manera similar el uso de compresores de voz permite optimizar recursos y así poder introducir otro tipo de servicios como es la videoconferencia bidireccional, la cual se hallaba limitada en un principio a enlaces terrestres.

5.5 Caso de estudio: GSM, la Telefonía Celular en Europa

Se conoce con el nombre genérico de GSM a la *suite de protocolos* definidos para normar el sistema de telecomunicaciones móviles (*Global System for Mobile Telecommunications*) que está siendo implantado en la Comunidad Económica Europea. Dentro de esta suite de protocolos se definió desde principios de la década de los noventas el que sería el algoritmo de

[1] NMS: Network Management System

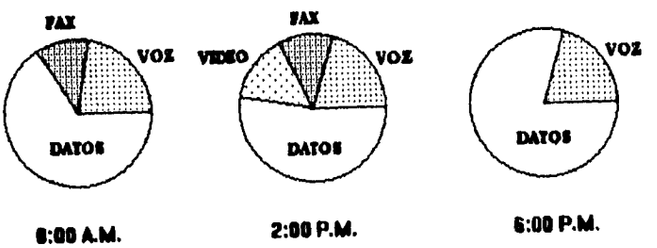

Figura 5.5: Reconfiguración de una red VSAT debido al tráfico

codificación utilizado en la telefonía digital, y que se denomina formalmente como *GSM 06.10 RPE-LTP* (Regular Pulse Excitation Long-Term Predictor). Definido por la ETSI[1] tiene una tasa de codificación de 13 kbits/s con una calidad de voz aceptable para telefonía celular comercial. El algoritmo GSM es más rápido que el CELP, por ejemplo, porque a diferencia de este último no necesita realizar comparaciones al buscar en tablas. Sin embargo, para su implementación en tiempo real se necesita por lo menos una estación de trabajo de mediano tamaño.

5.5.1 Particularidades del algoritmo

El algoritmo GSM 06.10 RPE-LTP utilizado es una variante de los métodos de codificación paramétrica basados en la codificación lineal predictiva de la voz. La entrada al algoritmo es voz digitalizada a 8 kHz con una resolución de 13 bits con signo codificada linealmente (es decir, sin utilizar técnicas de compansión). El algoritmo parametriza dicho flujo de muestras con una tasa de segmentación de 20 ms lo que equivale a 160 muestras, y codifica dichos segmentos en paquetes de 260 bits. Esto es, un segundo de voz comprimida se codifica en 1625 bytes, sin incluir la señalización necesaria en el sistema en el cual se implante.

Se utiliza un filtro Lattice de 8 retardos. Los coeficientes de reflexión son calculados mediante el algoritmo de Levinson-Durbin. En el codificador se utiliza la técnica de análisis por síntesis, por lo que puede catalogarse como un codificador híbrido.

[1]ETSI : European Telecommunications Standard Institute

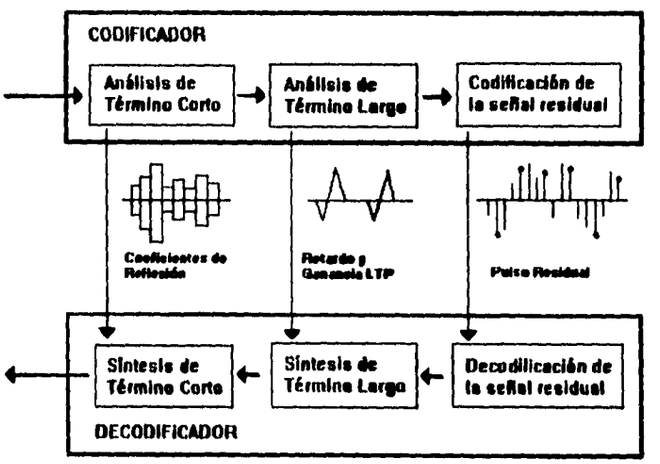

Figura 5.6: Esquema general de la arquitectura del algoritmo GSM

5.5.2 Cobertura actual en la red celular digital en Europa

El algoritmo GSM fue aceptado como un estándar desde 1987, y en la actualidad se utiliza uniformemente para el servicio de telefonía móvil internacional, operando en la banda de los 900 MHz. La cobertura abarca prácticamente la totalidad del territorio alemán, aproximadamente un 64 % del territorio italiano y un porcentaje similar en Francia e Inglaterra, donde la instalación del sitema avanza rápidamente. El sistema se halla en desarrollo también en Suiza, Luxemburgo, Noruega, Suecia, Finlandia, Dinamarca, Portugal y Grecia. En cada país se aplica una política de uso y tarificación propio, la cual es determinada por el gobierno local y la compañía concesionada para proporcionar el servicio. en Italia, por ejemplo, se aplica el estándar TACS, el cual asigna una tarifa preferencial a los servicios de negocios, ofreciendo servicios similares a los que se obtienen en un PBX, como aviso y desvío de llamadas, contestador telefónico, llamadas diferenciadas, números de seguridad y otros.

5.5.3 Reuso del algoritmo GSM: un teléfono a través de Internet

El algoritmo implementado en el sistema GSM ha sido liberado y sus lineamientos generales son de dominio público. De hecho se encuentra disponible una versión de él escrita en ANSI C por Jutta Degener de la Universidad Tecnológica de Berlín. Aunque esta implementación fue escrita originalmente para ejecutar en una estación de trabajo UNIX, puede con mínimas modificaciones utilizarse en otras plataformas.

Existen ya algunos intentos para implementar compresión de voz a través de Internet.

Figura 5.7: GSM, área de cobertura europea

Notablemente se puede mencionar el programa IGP [1] escrito por Sing Li [5] en Visual C++ 1.5, el cual ejecuta bajo windows 3.1 utilizando tarjetas digitalizadoras comerciales como Sound Blaster. Dicho programa es una primera aproximación a los servicios de voz sobre Internet, sin embargo dado que Internet adolece de las limitaciones que ya se han expuesto (anchos de banda variables y cuellos de botella) la estandarización de una propuesta como esta no es posible aún en un futuro cercano. Su funcionamiento es eficiente siempre y cuando los usuarios se encuentren, por ejemplo, dentro de la misma red local lo que minimizaría las estrecheces de anchos de banda y los retardos.

Conclusión

Complementando lo que inicialmente se expuso en el capítulo dos, se han presentado aquí de manera general los factores a considerar para implementar un algoritmo de codificación en un sistema de comunicaciones. Estos abarcan desde consideraciones especiales en los protocolos de enrutamiento de los paquetes hasta requisitos indispensables en las estaciones de trabajo que manejen un servicio de transmisión de voz. La implementación exitosa de tal servicio debe coordinar eficientemente todas estas consideraciones para ofrecer finalmente al usuario una operación confiable. En cada caso particular siempre existirán algunos factores que no se hayan mencionado aquí y que dependen por ejemplo, de las características del medio de transmisión, factores geográficos e inclusive factores económicos.

[1] IGP : Internet Global Phone

Referencias

[1] F.A. Westall & S.F.A. Ip: *Digital Signal Processing in Telecommunications*, Gran Bretaña, Chapman & Hall, B.T. Telecommunications, 1993.

[2] Spiros Dimolitsas: *Standardazing Speech-Coding Technology for Network Applications*, IEEE Communications Magazine, pp 26-33,E.U.A, IEEE Inc., Noviembre 1993.

[3] Ethevaldo Siqueira: *La Revolución de los sistemas VSAT*, Telepress Latinoamérica,pp 26-31, Brasil, Telepress Editora Ltda., Mayo-Junio 1993.

[4] Jutta Degener: *Digital Speech Compression*, Dr. Dobb's Journal, pp 30-34, Dec. 1994.

[5] Sing Li: *Building an Internet Global Phone*, Dr. Dobb's Journal Surcebook, pp 46-51, Winter 1994.

[6] Chris Rowden: *Speech Processing*, Essex series in telecomunnications and information systems, Mc Graw Hill, 1991.

[7] Hebert Taub & Donald L. Schilling: *Principles of Communications Systems*, Singapur, Mc Graw Hill Telecommunicatios Series, 1986.

Capítulo 6

Resultados y Conclusiones

6.1 Pruebas sobre el algoritmo

Con objeto de evaluar el desempeño del vocoder LPC implementado, se presentan a continuación un conjunto de resultados gráficos obtenidos tanto con el programa ADFO como utilizando facilidades que presta el programa MATLAB. Se presentan comparaciones de segmentos de voz digitalizada antes y después de la acción del compresor. Se analizan los resultados tanto en el dominio del tiempo como en el de la frecuencia, y se comentan los mismos.

6.1.1 Pruebas en el dominio del tiempo

En la gráficas siguientes se presenta el resultado de la acción del compresor sobre un segmento vocal típico. En la figura 6.1 se muestra la forma de onda original, que corresponde a un segmento vocal de 150 ms con una armónica principal muy bien definida.

Se realizaron pruebas con todas las opciones de depuración de la señal habilitadas (18 coeficientes LPC generados con el algoritmo de Schur, detector de silencios, corrección y filtrado del pitch e interpolación de parámetros); esto es en el mejor de los casos. En la figura 6.2 se graficó el mismo segmento tras una etapa de compresión/descompresión con las características mencionadas. Se observa que en el inicio de la gráfica, que corresponde todavía a un sonido no vocal se conserva la forma de onda original con bastante precisión; mientras que en la segunda parte de la gráfica, que corresponde a un sonido vocal, se conserva también la forma de onda y se observa un efecto de aumento de magnitud. En esta y otras pruebas se observó este efecto que es más notorio cuando la forma de onda original tiende a ser una senoidal pura. Si se utiliza como entrada al algoritmo una senoidal de 400 Hz por ejemplo, se observará un incremento en la magnitud de hasta un 100 %.

Al procesar por segunda vez el segmento mostrado en la figura 6.2 se observa un aumento en la ganancia de la parte vocal de dicho segmento. Si una señal continúa sometiéndose a sucesivas etapas de compresión/descompresión se observará un proceso de degradación sobre todo en los sonidos vocales, lo que provoca una distorsión en el tono de la voz y deforma el sentido de la palabra. Es por esta razón que no se recomienda el uso en cascada de este tipo de codificadores, dado que en cada etapa se genera irremediablemente degradación en la señal (Figura 6.3).

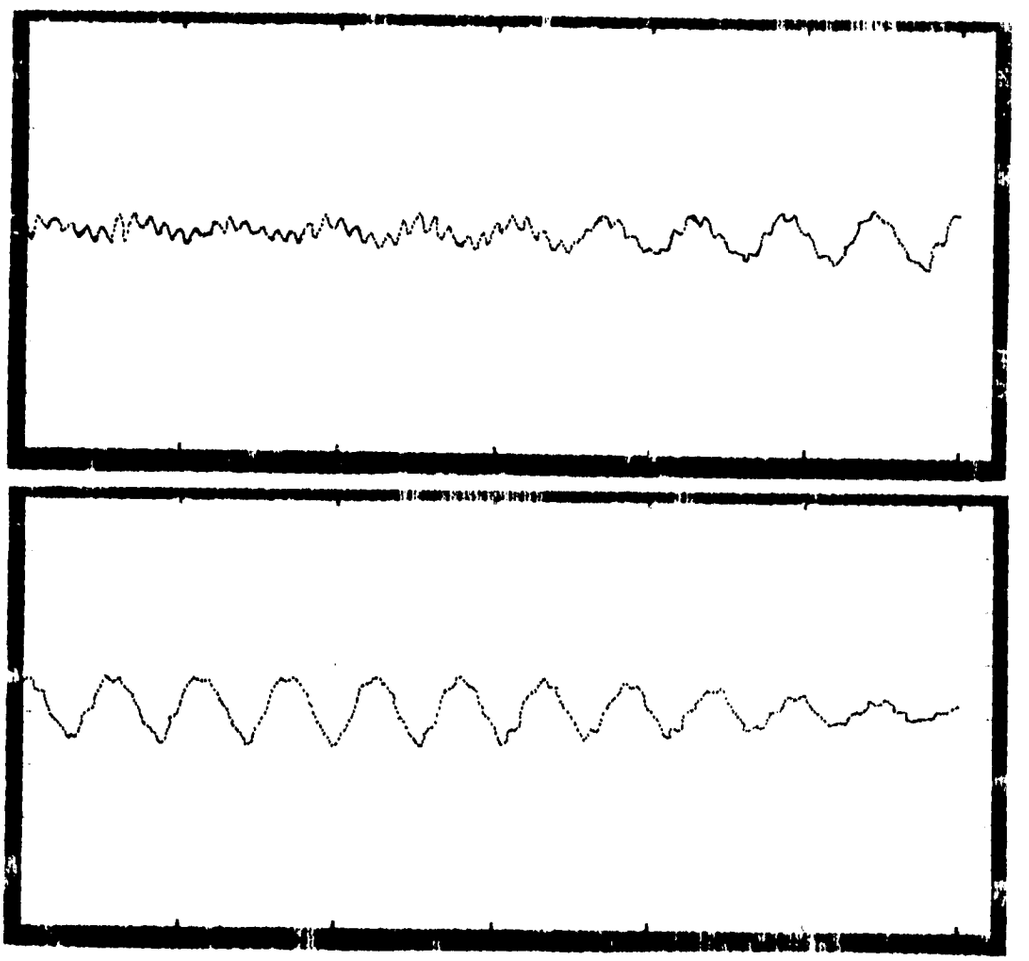

Figura 6.1: Segmento original de 1200 muestras

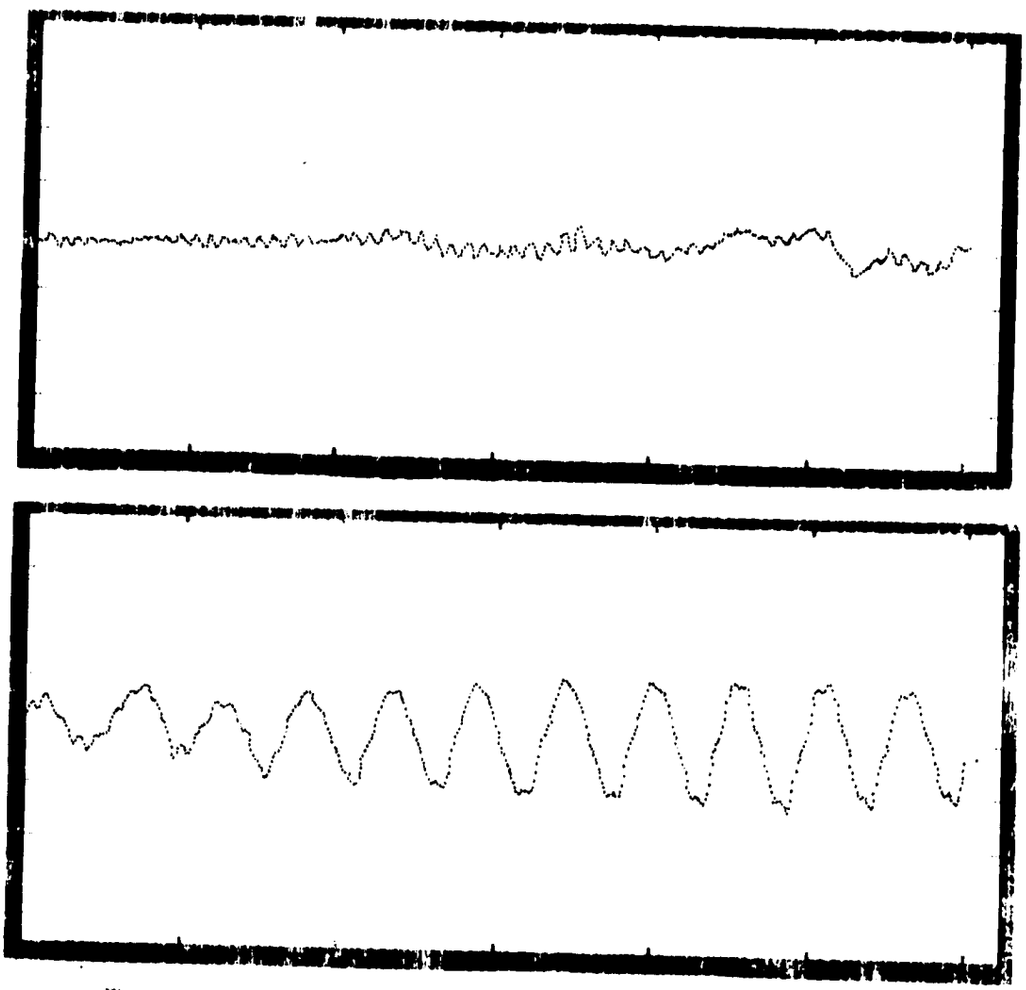

Figura 6.2: Segmento de 1200 muestras después de una etapa de compresión/descompresión

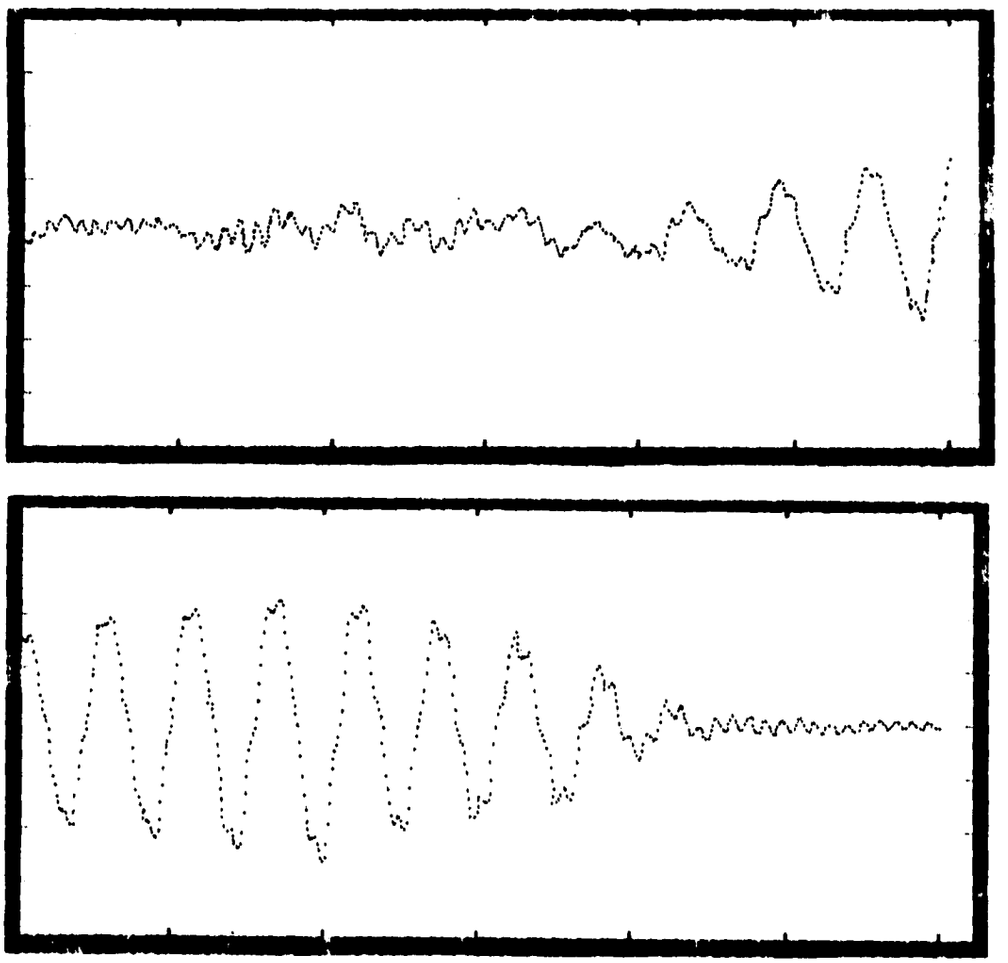

Figura 6.3: Segmento de 1200 muestras después de dos etapas de compresión/descompresión

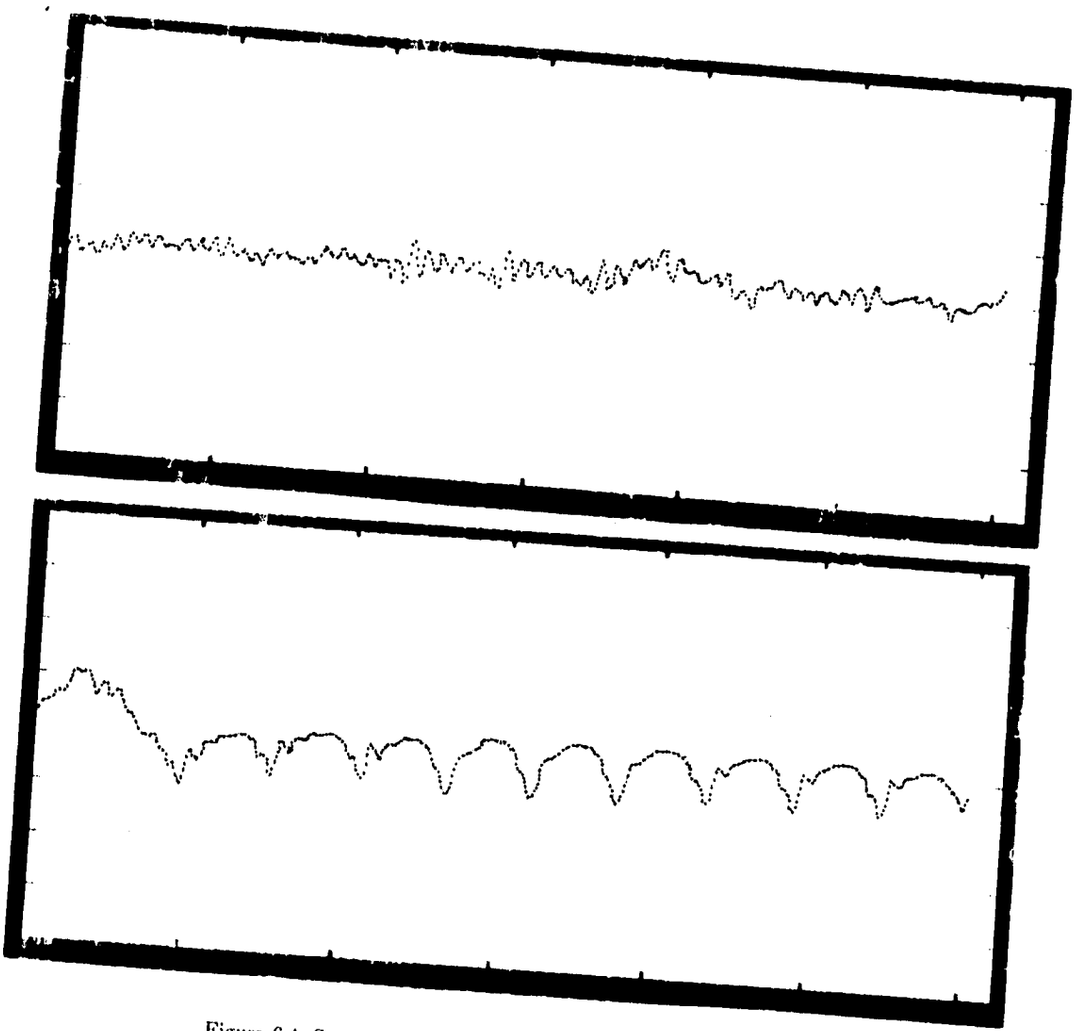

Figura 6.4: Segmento de 1200 muestras sintetizado en el peor de los casos

6.1.2 Pruebas en el dominio de la frecuencia

Con ayuda del paquete MATLAB fue posbile generar espectrogramas de banda angosta para diferentes secuencias de prueba. Si se observa por ejemplo el espectrograma de una señal no vocal (correspondiente por ejemplo, a un sonido fricativo) son notorios dos intervalos de ruido blanco a los 600 y 1800 ms (figura 6.5).

Después de una etapa de compresión/descompresión se observaron varios efectos:

- En general la forma del espectro se conservó, sin embargo puede notarse una disminución en la magnitud de las frecuencias más altas y un efecto de afirmación de las frecuencias más bajas. En la figura 6.7 se confirma el efecto de crecimiento de las armónicas de más baja frecuencia observado en las gráficas 6.2 y 6.3.
- Aparecen estrías en los segmentos no vocales, esto es, el ruido blanco tiende a segmentarse en varias componentes diferenciadas. Este efecto puede asociarse a la limitada cantidad de armónicas que el filtro Lattice puede reproducir.
- Haciendo referencia a los espectrogramas de superficie 6.6 y 6.8 es notoria una atenuación general en la señal, destacándose sin embargo los picos que inicialmente se observaban en la señal original.

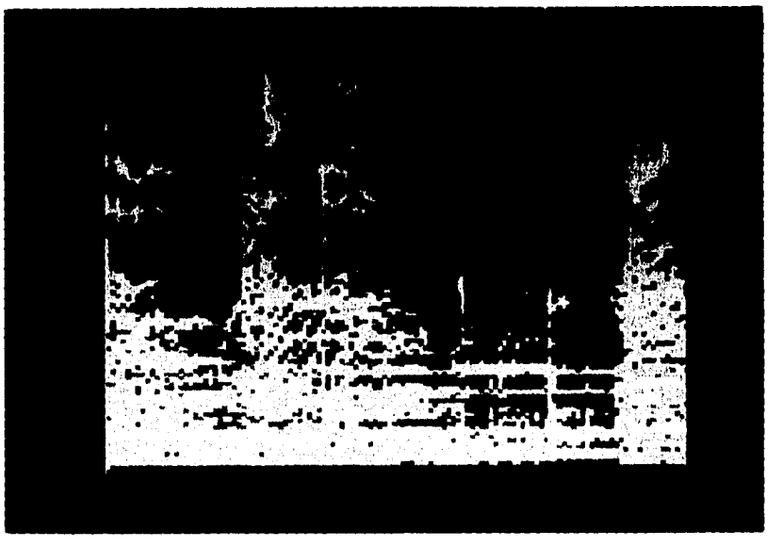

Figura 6.5: Espectrograma de banda angosta de un segmento de 2 segundos

6.1.3 Efectos observados al depurar la señal

Número de coeficientes

Se observó que al incrementarse el número de coeficientes LPC considerados al sintetizar, se conservaba de mejor manera el contenido armónico de la forma de onda. Es notoria la diferencia que existe cuando se sintetiza la señal en el peor de los casos (8 coeficientes) comparándolo con el mejor de los casos (18 coeficientes). En la figura 6.4 puede observarse que el contenido armónico del segmento vocal es tan pobre que incluso la forma de onda tiene poca semejanza con la original.

Detector de silencios

Pese a que el detector de silencios es un algoritmo sencillo, incrementa la calidad de la señal sintetizada notablemente. Su uso es necesario dado que el filtro Lattice tiende a producir una salida que oscila alrededor del nivel de tierra lo cual genera ruido de baja frecuencia que puede ser evitado con este recurso.

Depuración de decisiones de pitch

Aunque se obtiene un valor de período de pitch para cada segmento (esto es, cada 20 ms), por lo general la variación de este valor se presenta en intervalos mucho más grandes (intervalos de 200 ms o más). Es por esto que una correcta depuración de la secuencia de decisiones

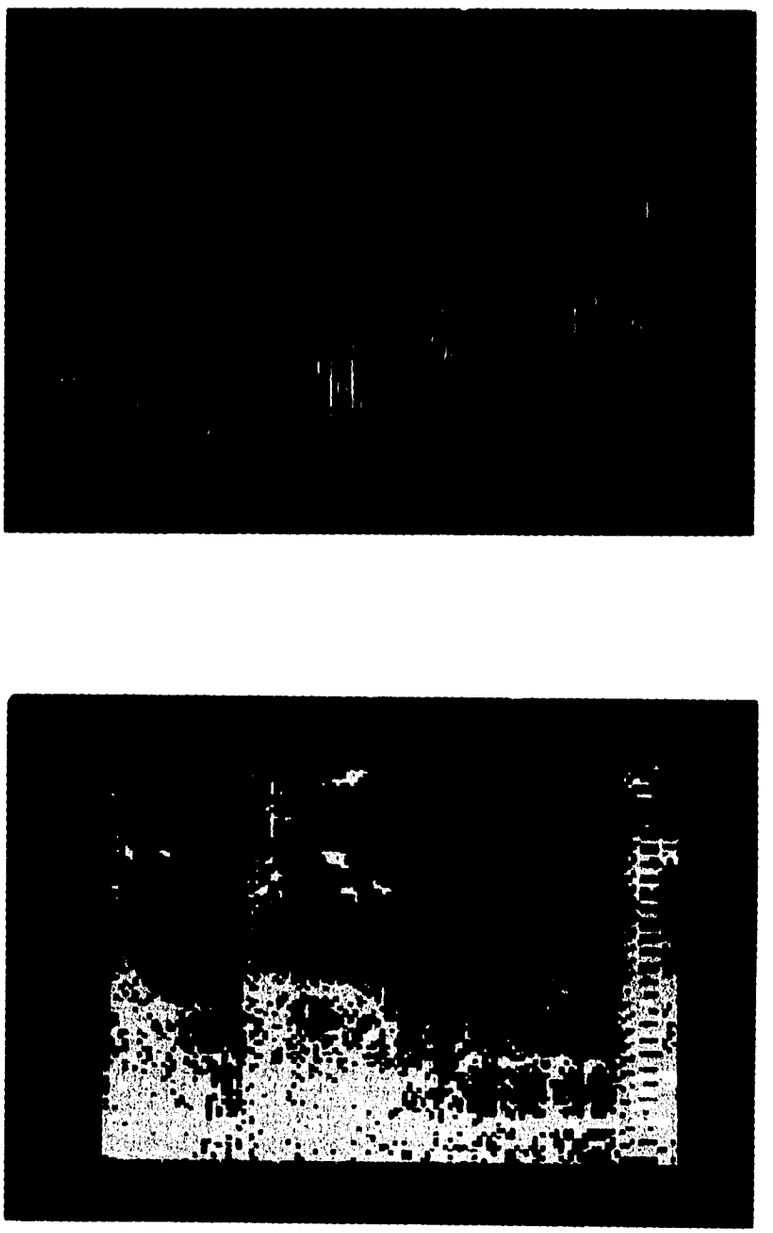

Figura 6.7: Espectrograma de banda angosta del segmento sintetizado

Figura 6.8: Espectrograma de superficie del segmento sintetizado

llevada a cabo por el corrector de errores VUV y UVU, así como por los filtros mediador y supresor de picos dan continuidad al tono de la voz.

Interpolación de valores de ganancia y coeficientes LPC

Esta interpolación, que se realiza para cada muestra sintetizada, incrementa notablemente la calidad de la señal. En la figura 6.4, en la cual no se aplicó la interpolación, se observa que aproximadamente cada tres períodos de la señal, tanto la amplitud como el contenido armónico tienen variaciones bruscas que no suelen presentarse en la realidad.

6.2 Consideraciones para la implementación en tiempo real

6.2.1 Compatibilidad de formatos

En el presente caso, el algoritmo implementado maneja como entrada números enteros signados de 16 bits, los cuales son convertidos a números de punto flotante de 32 bits. Al implementar en tiempo real las rutinas debe considerarse este factor ya que por ejemplo, un convertidor A/D de menos de 8 bits tal vez resulte insuficiente para conservar la calidad de la señal.

Al transportar los algoritmos programados a un microprocesador como el TMS320C30. la precisión en las operaciones inclusive se incrementará, dado que dicho microprocesador extiende la precisión en los cálculos de punto flotante hasta 40 bits internamente.

6.2.2 Transportabilidad del código

Las rutinas que intervienen directamente en el análisis y síntesis son completamente compatibles con el ANSI C, lo que las hace susceptibles de ser transportadas al compilador de C del TMS320C30. Inclusive se hace un extenso manejo de apuntadores a arreglos de números de punto flotante, característica que optimiza el manejo de memoria y es soportada por el mencionado compilador de manera amplia.

6.2.3 Capacidad del procesador

El microprocesador TMS320C30 destaca por su elevada capacidad de realizar cálculos intensivos; sin embargo dicha capacidad tiene un límite el cual debe cuidarse de no ser rebasado para lograr una operación en tiempo real. En las rutinas programadas existen algunas que no representan problema alguno al ser transportadas a tiempo real, como son las inicializaciones de arreglos, o los filtros de preénfasis y deénfasis, que contienen sólo un retardo. Sin embargo otras operaciones, notoriamente el filtro FIR utilizado en el análisis, las operaciones de selección de pitch y la generación de coeficientes de autocorrelación consumen un alto porcentaje de la capacidad del procesador. Dependiendo entonces del grado de depuración que tenga el código del programa será posible implementarle en tiempo real. En el caso de que el número de operaciones que puede realizar como máximo el microprocesador sean insuficientes se deberá pensar entonces en cambiar el diseño del programa con objeto de reducir el número de ciclos totales. Por ejemplo, en el caso del filtro FIR de 79 taps utilizado en el análisis, este podría ser sustituido por un filtro IIR con características equivalentes.

6.3 Perspectivas de desarrollo y trabajos colaterales

Una de las características del programa desarrollado es que tiene una gran flexibilidad en cuanto a las opciones de depuración se refiere, pudiéndose seleccionar la aplicación de los diferentes módulos del programa y observar el efecto de ello. Sin embargo existen mucho más caminos para realizar las diferentes operaciones que se realizan en el programa. A continuación se mencionan algunas alternativas:

- Puede pensarse en variar la tasa de segmentación del programa y observar el efecto de retardos mayores.
- Es posible utilizar otros sintonizadores de pitch de eficiencia comprobada, como por ejemplo el sintonizador AMDF [1] [1].
- Existen otras alternativas para calcular los coeficientes LPC, como por ejemplo el algoritmo de Leroux-Gueguen [1].
- Con objeto de incrementar la precisión en la señal de excitación puede implementarse un codificador de análisis por síntesis que obtenga una señal de residuo única para cada segmento con la cual se excite al filtro Lattice. Esta mejora elevaría la calidad de la señal y convertiría al codificador en un codificador híbrido.
- Los métodos utilizados para la corrección de decisiones de pitch pueden modificarse y mejorarse. Por ejemplo, puede pensarse en elevar el orden de los filtros mediador y supresor de picos y observar su efecto.
- Con objeto de reducir significativamente la tasa de codificación es aconsejable generar tablas de búsqueda para los coeficientes LPC de manera que en lugar de codificar p números de punto flotante se codifiquen p números enteros que sean apuntadores a dichas tablas.

[1] AMDF: Average Magnitude Difference Function

Referencias

[1] Panos E. Papamichalis: *Practical Approaches to Speech Coding*, New Jersey, Prentice-Hall, Inc., 1987.

[2] Texas Instruments: *TMS320C3x C Source Debugger, User's Guide*, Texas Instruments Inc., E.U.A., 1991.

[3] Texas Instruments: *TMS320 Floating-Point DSP Assembly Language Tools, User's Guide*, Texas Instruments Inc., E.U.A., 1991.

[4] Texas Instruments: *TMS320C3x, User's Guide*, Texas Instruments Inc., E.U.A., 1990.

Apéndice A

El microprocesador TMS320C30

A.1 Introducción

El microprocesador TMS320C30 forma parte de la tercera generación de la familia TMS320 de procesadores digitales de señales (DSPs) implementados en tecnología 1μm CMOS, capaces de manejar operaciones con punto flotante de 32 bits. Se fabrica en un encapsulado de tipo PGA (pin grid array) de 181 pins.

Sus principales características son : El ciclo de bus es de 60 ns, palabra de datos y programa de 32 bits, con bus de direcciones de 24 bits, lo que resulta en un espacio de memoria de 2^{24} = 16.7 Megapalabras. Ejecutando instrucciones a una tasa de 33.3 MFLOPS [1], y 16.7 MIPS [2]. Funciona con un reloj externo de 66 MHz.

La principal característica de un DSP con respecto a los demás microprocesadores es la multiplicación en paralelo y carga al acumulador , todo esto en un sólo ciclo (instrucción tipo MAC "Multiply and Accumulate").

Posee un conjunto de registros, algunos de propósito general denominados archivo de registros, utilizados por el CPU para llevar el control de sus operaciones. Tiene un caché de 64 palabras de longitud, utilizado por la memoria de programa. Para realizar operaciones aritméticas, hace uso de dos unidades aritméticas trabajando en paralelo, cada una de las cuales posee un registro auxiliar asociado (ARAU0 y ARAU1).

Los bloques de memoria internos son de acceso dual (pueden accesar dos operandos en un ciclo de máquina). Existe un canal dedicado de DMA (acceso directo a memoria) que disminuye la carga al CPU, soportándose operaciones de Entrada/Salida concurrente.

En cuanto a los periféricos implementados dentro del circuito integrado se contemplan dos temporizadores y dos puertos seriales independientes. Debido a su arquitectura basada en registros se facilita la implementación de compiladores de alto nivel como es el caso del lenguaje C. El bloque de memoria ROM es de 4k palabras y existen además dos bloques de memoria RAM de 1k. La unidad aritmética Lógica y el multiplicador son de 40 bits para punto flotante y de 32 bits para números enteros. Tiene un registro de corrimiento de hasta

[1] MFLOPS = millones de operaciones de punto flotante por segundo
[2] MIPS = millones de instrucciones por segundo

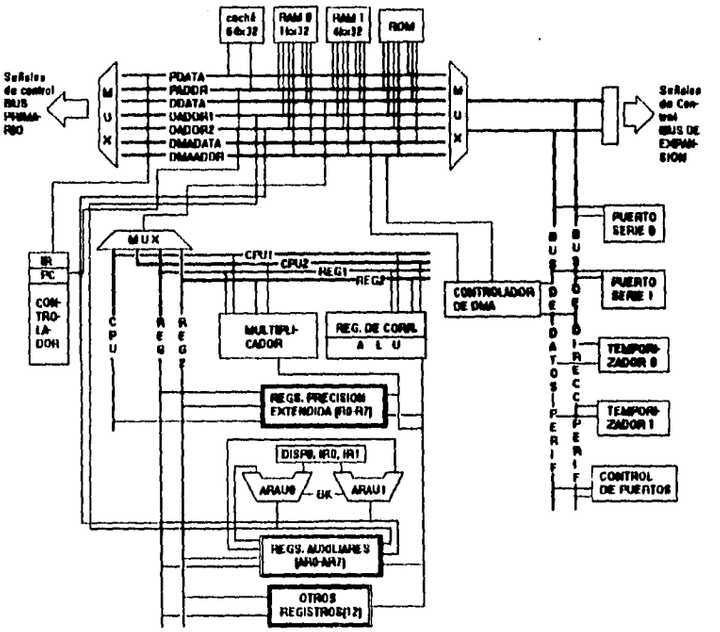

Figura A.1: Arquitectura del microprocesador TMS320C30

32 bits, ocho registros o acumuladores de precisión extendida así como dos generadores de direcciones con ocho registros auxiliares y dos registros auxiliares de las unidades aritméticas.

Puede ejecutar instrucciones de dos o tres operandos. Posee dos banderas externas de propósito general y cuatro interrupciones externas. El controlador de DMA se utiliza para leer o escribir en la memoria sin que se interrumpa al CPU.

A.2 Arquitectura

La arquitectura interna del TMS320C30 se estructuró con la finalidad de poder ejecutar grandes volúmenes de operaciones aritméticas utilizando tanto Hardware como Software. Posee una extensa memoria interna y alto grado de paralelismo en las operaciones lográndose así una reducción significativa en el tiempo de ejecución de un bloque de instrucciones.

A.3 Unidad Central de Proceso CPU

La administración de los recursos del CPU se realiza a través de registros agrupados en un *archivo de registros* conteniendo este los siguientes componentes:

1. Multiplicador de enteros y números de punto flotante.
2. Unidad aritmética lógica (ALU), la cual ejecuta operaciones de punto flotante, de enteros y operaciones lógicas.
3. Registro de corrimiento de hasta 32 bits.
4. Buses internos (CPU1/CPU2 y REG1/REG2).
5. Unidades Aritméticas con registros auxiliares asociados (ARAU).
6. Conjunto de 28 registros con diversos propósitos (*register file*).

A.3.1 Multiplicador para enteros y punto flotante

Ejecuta instrucciones de un ciclo de máquina de duración de la siguiente manera: 1. Multiplicación entera de 24 bits con resultado de 32 bits. 2. Multiplicación de punto flotante de 32 bits con resultado de 40 bits.

A.3.2 Unidad Aritmética Lógica ALU

Aquí se ejecutan operaciones de duración de un ciclo de máquina para datos enteros y lógicos de 32 bits, y datos de punto flotante de 40 bits. La extensión de la palabra de datos resultado tiene la misma extensión que los operandos.

A.3.3 Registro de corrimiento de 32 bits

Es utilizado para realizar desplazamientos de hasta 32 bits hacia la izquierda o hacia la derecha en un sólo ciclo.

A.3.4 Buses Internos CPU1/CPU2 y REG1/REG2

La concepción separada de estos buses permite extraer dos operandos de memoria y dos operandos desde los registros para que se realicen multiplicaciones paralelas así como sumas y restas de cuatro operandos en un sólo ciclo de máquina.

A.3.5 Unidades Aritméticas con Registros Auxiliares Asociados ARAU0 y ARAU1

Es posible generar dos direcciones en un sólo ciclo operando en paralelo con el multiplicador y la ALU, esto permite direccionamientos con desplazamientos usando modo de direccionamiento indexado circular o de inversión de bit.

A.3.6 Archivo de Registros del CPU

Estos se pueden utilizar como registros de propósito general, aunque tambien cada uno de ellos tiene una función específica; por ejemplo, se tienen ocho registros de precisión extendida, ocho registros auxiliares, entre otros que se verán a continuación.

Nombre del Registro	*Función*
R0	Registro de precisión extendida 0
R1	Registro de precisión extendida 1
R2	Registro de precisión extendida 2
R3	Registro de precisión extendida 3
R4	Registro de precisión extendida 4
R5	Registro de precisión extendida 5
R6	Registro de precisión extendida 6
R7	Registro de precisión extendida 7
AR0	Registro Auxiliar 0
AR1	Registro Auxiliar 1
AR2	Registro Auxiliar 2
AR3	Registro Auxiliar 3
AR4	Registro Auxiliar 4
AR5	Registro Auxiliar 5
AR6	Registro Auxiliar 6
AR7	Registro Auxiliar 7
DP	Apuntador a Página de Datos
IR0	Registro Indice 0
IR1	Registro Indice 1
BK	Registro de Tamaño de Bloque
SP	Apuntador al Stack
ST	Registro de Status
IE	Registro de Habilitación de Int. CPU/DMA
IF	Registro de Banderas de Int. del CPU
IOF	Registro de Banderas de E/S
RS	Registro de dir. inicial de Repetición
RE	Registro de dir. final de Repeticón
RC	Contador de Repetición
PC	Contador de Programa

Registros de Precisión extendida R0-R7

Se utilizan para almacenar operandos y realizar operaciones con enteros de 32 bits y números de punto flotante de 40 bits. Se asume que para operaciones con enteros se usan los 32 bits menos significativos y en operaciones de punto flotante se usan los 40 bits.

Registros Auxiliares AR0-AR7

Se accesan por el CPU y por los ARAUs. Su función principal es la generación de direcciones de 24 bits, pero se les puede usar como contadores de cuenta cerrada o como registros de propósito general.

Apuntador a Página de datos DP

Es un registro de 32 bits, donde los ocho menos significativos se utilizan para direccionamiento directo como un apuntador a la página de datos direccionada. Las páginas son de 64 k-palabras teniéndose un total de 256 páginas.

Registros Indice IR0,IR1

Estos registros contienen el valor usado por la correspondiente ARAU para calcular una dirección indexada.

Registros de tamaño de bloque BK

Es utilizado por la ARAU para especificar el tamaño del bloque en el direccionamiento circular.

Apuntador al Stack del Sistema SP

Es un registro de 32 bits que contiene la dirección guardada en la localidad más alta del stack. Este registro siempre apunta al último elemento que entra. El stack es intervenido por interrupciones, llamadas y retornos de subrutinas, y por las instrucciones PUSH y POP.

Registro de Status ST

Contiene información global sobre el estado del CPU, es decir, contiene las banderas que indican si el resultado de una operación fue cero, o negativo, entre otras. Es posible mediante software salvar y restaurar este registro.

Registro de Habilitación de Interrupciones de CPU/DMA IE

Es un registro de 32 bits donde un "1" en la posición adecuada habilita la interrupción corre spondiente. Las interrupciones para DMA están en los bits 26 a 16 de este registro. Un 0 deshabilita la interrupción.

Registro de Banderas de interrupciones del CPU IF

Es un registro de 32 bits en el cual de la misma manera que el anterior con un "1" se habilita y con un "0" se deshabilita la interrupción correspondiente.

Registro de Banderas de E/S IOF

Controla los pines externos XF0 y XF1 los cuales pueden ser configurados como entradas y/o salidas.

Registro de dirección inicial de Repetición RS

Cuando el procesador se encuentra operando en el modo de repetición este registro contiene la dirección inicial del bloque de instrucciones a ser repetidas.

Registro de direccionamiento final de Repetición RE

Este registro contiene la última dirección del bloque de programa a ser repetido.

Contador de Repetición RC

Este registro, de 32 bits, es utilizado para especificar el número de veces que un bloque de código se va a repetir.

Contador de Programa PC

Este registro, de 32 bits, contiene la dirección de la siguiente instrucción que va a ser accesada por el CPU. Aunque el PC no es parte de los registros del CPU, si es un registro que puede ser modificado por instrucciones que alteran el flujo del programa.

A.4 Organización de la memoria

El espacio total de memoria del procesador, definido por un bus de direcciones de 24 bits, da por resultado 16 M-palabras. El programa, los da tos y el espacio de E/S están contenidos dentro de este espacio de 16 M-palabras, permitiendo que tablas, coeficientes, código de programa o datos puedan estar guardados ya sea en RAM o en ROM.

A.4.1 Memorias RAM, ROM y Caché

El procesador tiene dos bloques de memoria RAM (Bloque 0 y Bloque 1), cada uno de los cuales es de 1 K-palabras; y un bloque de ROM de 4 k-palabras. Cada bloque permite dos accesos en un solo ciclo de máquina.

Dado que los buses de programa (PADDR,PDATA), de datos (DADDR1,DADDR2) y de DMA (DMAADDR,DMADATA), están separados, es permitido que se realicen paralelamente lecturas y escrituras de datos, acceso al código del programa y operaciones de DMA. Existe un caché de instrucciones de 64 palabras utilizado para almacenar las secciones de código más utiliza das reduciendo así el número de accesos necesario.

A.4.2 Mapas de Memoria

La configuración del mapa de memoria depende del estado del pin $MC/\overline{MP}$.

Modo de microcomputadora ($MC/\overline{MP} = 1$), el espacio en ROM de 4 k está mapeado de las localidades 0h a la 0FFFh; las primeras 192 localidades se destinan a vectores de interrupción, vectores de "trampa", y a un espacio reservado. Las localidades de la 1000h a la 7FFFFFh se destinan a memoria externa, siendo activo el pin $\overline{STRB}$.

En el modo de microprocesador ($MC/\overline{MP} = 0$), no existe el bloque de 4 k de ROM. Las primeras 192 localidades están destinadas a interrupciones, trampas y algunas localidades reservadas. En este espacio se accesa a memoria externa siendo activa la terminal $\overline{STRB}$. A partir de la localidad C0h, es decir, el espacio de memoria restante, corresponde a memoria externa.

Para ambos modos de operación: Las localidades 800000h a la 801FFFh están mapeadas al bus de expansión activándose la terminal $\overline{MSTRB}$. Los registros de control de los periféricos están entre las localidades 808000h a la 8097FFh para ambos modos de operación. El bloque 0 de RAM está mapeado de la 809800h a la 8098FFh y el bloque 1 de la 809C00h a la 809FFFh. A partir de la localidad 80A000h a la 0FFFFFFh está destinado a memoria externa.

A.4.3 Modos de Direccionamiento

En este microprocesador se soportan seis tipos de direccionamiento:

Direccionamiento por registro El operando es siempre un registro del CPU, pudiendo ser un registro de precisión extendida.

Direccionamiento corto-inmediato El operando es un valor inmediato de 16 bits.

Direccionamiento largo-inmediato El operando es un valor inmediato de 24 bits.

Direccionamiento directo El operando es el contenido de una dirección de 24 bits.

Direccionamiento indirecto Un registro auxiliar indica la dirección del operando.

Direccionamiento relativo al PC Al PC se le suma un desplazamiento signado de 16 bits.

A.5 Operación del bus interno

El contador de programa (PC) se encuentra conectado al bus de direcciones de programa y el registro de instrucción (IR) está conectado al bus de datos de programa, ambos buses pueden accesar sólo una palabra cada ciclo de máquina. En cuanto al bus de direciones de datos, este permite dos accesos en un ciclo de máquina.

El controlador de DMA posee sus propios buses de direcciones (DMAADDR) y de datos (DMADATA), lo cual permite al controlador hacer accesos a memoria en paralelo con los accesos a memoria de los buses de programa y datos.

A.6 Operación del bus externo

Esta constituido por un bus primario de direcciones de 24 bits y un bus de expansión de 13 bits. Este bus se utiliza para accesar memoria de programa, datos, y espacio de E/S. Se utiliza una señal externa $\overline{RDY}$ para generar estados de espera.

A.6.1 Interrupciones externas

Se soportan cuatro interrupciones externas ($\overline{INT3} - \overline{INT0}$). Se manejan también interrupciones internas y una señal externa de $\overline{RESET}$ no mascarable. Estas señales se utilizan para interrumpir ya sea al CPU o al controlador de DMA. Cuando el CPU responde a la interrupción, el pin $\overline{IACK}$ puede utilizarse como una contraseña externa a la interrupción.

A.6.2 Señalización para ejecución de instrucciones sincronizadas

Se utilizan dos banderas externas XF0 y XF1 configurables mediante software como entradas o salidas. Se utilizan por las operaciones de sincronización del procesador las cuales soportan comunicación entre microprocesadores, así como también es posible implementar tiempos de espera, entre otras aplicaciones.

A.7 Control de periféricos

Los periféricos incluídos en el circuito integrados son dos puertos seriales y dos temporizadores. el control de ellos se realiza a través de registros mapeados en memoria utilizando para ello un bus periférico dedicado. dichos registros de control se encuentran en las localidades 808000h hasta la 8097FFh.

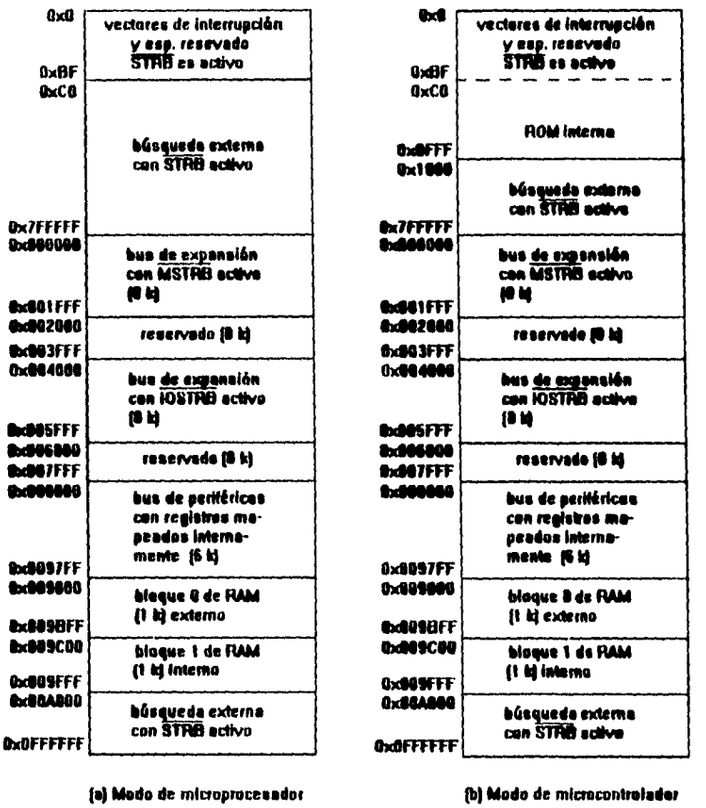

Figura A.2: Mapa de memoria

A.7.1 Temporizadores

Los dos módulos de temporización de 32 bits pueden funcionar como temporizadores o bien como contadores de eventos con dos modos de señalización y utilizando señal de reloj generada ya sea interna o externamente. Asociado a cada módulo hay un pin que puede ser configurado como entrada de reloj para el temporizador o como una salida controlada por el mismo.

A.7.2 Puertos Seriales

Ambos puertos son completamente independientes uno del otro, pero idénticos en su configuración de registros y también son configurables para operar con palabras de datos de 8,16,24 o 32 bits.

La señal de reloj para cada puerto serial puede ser generada en forma interna o externa, contándose con un divisor de frecuencia. Pueden funcionar también como temporizadores.

A.8 Acceso directo a memoria (DMA)

El controlador de DMA como ya se mencionó, puede leer o escribir a memoria sin interferir con la operación del CPU. Esto es de gran utilidad cuando se interactúa con dispositivos externos de baja velocidad, evitando disminuir el desempeño del CPU. El controlador de DMA contiene sus propios generadores de direcciones, registro fuente y destino y contadores de transferencia. Dado que el controlador posee sus propios buses de datos y direcciones se minimizan los conflictos con el CPU. Una operación de DMA consiste en la transferencia de una palabra o un bloque de datos desde o bien hacia la memoria.

A.9 Operaciones con Pipeline

El microprocesador, al accesar una instrucción de memoria y proceder a ejecutarla, realiza básicamente cuatro operaciones, las cuales son ejecutadas en el modo *Pipeline* como sigue:

Unidad de Búsqueda (FETCH) (F):

Obtiene las palabras de instrucción de memoria y actualiza el contador de programa (PC).

Unidad de Decodificación (D):

Realiza la decodificación de la instrucción y genera una dirección, también controla las modificaciones realizadas a los registros auxiliares y al apuntador al stack.

Unidad de Lectura (R):

En caso de ser necesario, realiza lectura de operandos.

Unidad de Ejecución (E):

De requerirse, lee los operandos del archivo de registros, ejecuta la operación necesaria y actualiza el archivo de registros. En ocasiones también escribe resultados previos a memoria.

Cuando estas cuatro operaciones se superponen perfectamente en el mismo ciclo de pipeline, operando en paralelo, la facilidad del pipeline se utiliza plenamente (Figura A.1). Para el programador, la ejecución de instrucciones en Pipeline es transparente, sin embargo este puede buscar que el acomodo de las instrucciones favorezcan que se realice óptimamente.

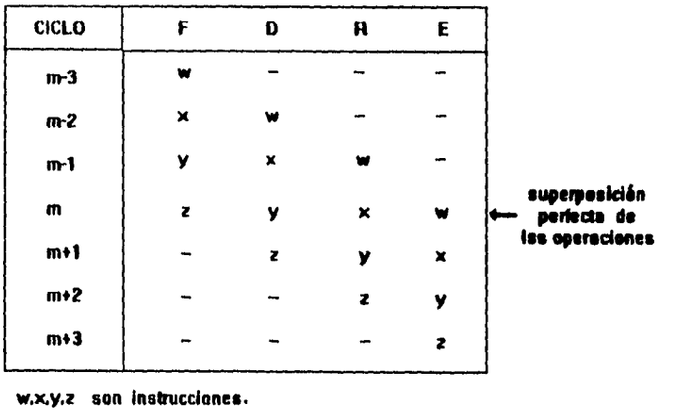

Figura A.3: Estructura de las operaciones en Pipeline

Durante la ejecución de operaciones en Pipeline, pueden generarse conflictos debidos a los diferentes accesos que realiza una instrucción en particular, o bién al uso del stack que pueda realizar, entre otras causas.

Prioridad de las unidades de Pipeline. Se asigna una prioridad a las operaciones, esto es, el orden en que se llevan a cabo:

- Ejecución (E)
- Lectura (R)
- Decodificación (D)
- Búsqueda (F)
- Canal de DMA (DMA) (en caso de presentarse)

Cuando una instrucción está lista para entrar al siguiente nivel de Pipeline, pero ese nivel no está listo para aceptar a dicha instrucción se presenta un conflicto. En este caso la unidad de más baja prioridad espera hasta que la siguiente unidad en prioridad complete su ejecución.

Los conflictos entre las operaciones de DMA y la estructura de Pipeline se minimizan debido a que el controlador de DMA posee sus propios buses de datos y direcciones.

Conflictos en Pipeline Se les puede agrupar en tres categorías principales:

1. Conflictos de saltos. Se generan cuando alguna instrucción lee o modifica el contador de programa (PC).
2. Conflictos con Registros. Involucra retardos generados al leer o escribir a registros que sean usados para generar direcciones.
3. Conflictos con memoria. Ocurren cuando las diferente unidades del microprocesador compiten por los recursos de memoria.

A.10 Uso de los recursos del sistema

El microprocesador soporta aritmética entera y de punto flotante permitiendo al programador concentrarse en el algoritmo que debe programar preocupándose lo menos posible de problemas como escalamiento, rango dinámico o sobreflujos.

A.10.1 Inicialización del Procesador

Al efectuarse un RESET, el microprocesador finaliza la ejecución del programa en curso y coloca el valor del vector de reset en el contador de programa (PC). El reset por Hardware (pin externo) también inicializa varios registros y bits de estado. Después del reset deben

inicializarse modos de operación, apuntadores de memoria, interrupciones y especificaciones propias de la aplicación. Dentro de la configuración inicial debe incluirse la asignación de valores iniciales a los registros mapeados en memoria y en la estructura de interrupciones.

A.10.2 Control de Flujo dentro de un Programa

El modelo de programación de este microprocesador posee diversas estructuras que facilitan la programación de algoritmos de procesamiento digital de señales, como son:

- Llamadas a subrutinas (llamadas, trampas y retornos),
- Uso del stack por software,
- Saltos retardados,
- Operaciones en modo multiprocesador,
- Interrupciones y
- Modo de repetición.

Llamadas a Subrutinas (Llamadas, Trampas y Retornos)

Una vez diseñada la subrutina correspondiente, se ejecuta mediante las instrucciones CALL, CALLcond y TRAPcond. A través de las instrucciones RETScond y RETIcond se regresa de la subrutina invocada, restaurando el stack automáticamente.

Uso del Stack por Software

El microprocesador tiene un stack controlado por software cuya localización se determina por el contenido del apuntador al stack (SP). Dicho stack se accesa no solamente por las instrucciones CALL y RETS, sino también como una forma de almacenamiento temporal con las instrucciones PUSH y POP para números enteros y PUSHF y POPF para números de punto flotante. El apuntador al stack no se inicializa por hardware durante el reset, es entonces importante asignarle un valor con una dirección de memoria predeterminada.

Saltos Retardados

Este tipo de saltos funcionan como los saltos normales, sin embargo no omiten el uso de la estructura de pipeline, por lo que es conveniente procurar implementarlos. Cuando un salto retardado es encontrado dentro de un programa las tres instrucciones siguientes dentro de un programa son ejecutadas. Las restricciones en este caso son que ninguna de estas tres instrucciones sea: un salto, llamada a subrutina, retorno de subrutina o de interrupción, instrucción de repetición, instrucción trampa o instrucción de espera.

Operación en modo multiprocesador.

Estas operaciones, auxiliadas mediante señalización externa, garantizan la integridad de la comunicación e incrementan la velocidad de ejecución.

Para mantener el acceso a una memoria global y compartir datos de una manera coherente es necesario un protocolo para arbitrar este tipo de procesamiento. Este es el propósito de un pequeño conjunto de instrucciones que ayudan a la sincronización (LDFI, LDII, SIGI,STFI, STII).

Interrupciones.

Se encuentran en forma vectorizada y existe un orden de prioridad entre ellas. Cuando una interrupción ocurre se activa el correspondiente bit en el registro de banderas de interrupción (IF). Si el correspondiente bit en el registro de habilitación de interrupciones (IE) se encuentra activo, y las interruciones a su vez se han habilitado por el bit GIE en el registro de estado, la correspondiente rutina de interrupción se ejecuta. Existe también la posibilidad de escribir en el IE y de esta manera forzar la interrupción por software, o bién deshabilitarla para que no se lleve a cabo.

Exceptuando rutinas de interrupción muy simples, es importante asegurar que el contexto del procesador (es decir, el archivo de registros) se ha respaldado previamente a dar el salto a la subrutina. Este conjunto de registros debe almacenarse en el stack previamente al brinco de subrutina y recuperarse de manera inversa (el stack puede verse como una memoria LIFO), conservándose entonces el contexto original.

Modo de Repetición.

Se contemplan dos modos de repetición: repetición de un bloque de código (RPTB), y repetición de una sola instrucción (RPTS). El control del número de repeticiones, así como

las direcciones inicial y final del bloque a repetirse se realiza manipulando los registros RS, RE y RC.

A.11 Conjunto de Instrucciones

Con un total de 113 instrucciones, el microprocesador TMS320C30 provee un versátil juego de instrucciones tanto de propósito general como las que están especialmente diseñadas para cálculo intensivo.

A su vez el conjunto se subdivide en grupos de instrucciones especializadas en carga-almacenamiento, operaciones lógicas o aritméticas de dos o tres operandos, operaciones paralelas - justificando plenamente la existencia de dos ALUs -, operaciones de control de programa - donde sobresalen las instrucciones para repetición de instrucciones o bloques de ellas-, así como cinco instrucciones previstas especialmente para ejecución compartida con otros microprocesadores C30. Todas las instrucciones ocupan al codificarse una palabra completa de 32 bits, y algunas de ellas se ejecutan en un solo ciclo. La eficiencia en la ejecución depende también de la optimización del código que favorezca las operaciones en pipeline.

A.11.1 Instrucciones de Carga y Almacenamiento

Se contemplan 12 instrucciones de este tipo, las cuales consisten ya sea de cargas de memoria hacia un registro, cargas de un registro hacia memoria, o manipulación de datos en el stack del sistema. Dos de estas instrucciones pueden realizar cargas condicionales.

Instrucción	*Descripción*	*Ciclos*
LDE	Carga exponente de punto flotante	1
LDF	Carga valor de punto flotante	1
LDFcond	Carga v. de p. f. condicionalmente	1
LDI	Carga un entero	1
LDIcond	Carga un entero condicionalmente	1
LDM	Carga mantisa de punto flotante	1
POP	Extrae un entero del stack	1
POPF	Extrae un valor de p. f. del stack	1
PUSH	Deposita un entero en el stack	1
PUSHF	Deposita un valor p. f. al stack	1
STF	Almacena un valor de p.f.	1
STI	Almacena un entero	1

A.11.2 Instrucciones de dos operandos

Son 35 instrucciones de este tipo, donde el operando fuente puede ser una localidad de memoria, un registro, o parte de la palabra de instrucción. El operando destino es siempre un registro del CPU. Se comprenden operaciones con enteros, punto flotante, lógicas y de aritmética de precisión múltiple.

Instrucción	*Descripción*	*Ciclos*
ABSF	Valor absoluto de un valor de p.f.	1
ABSI	Valor absoluto de un entero	1
ADDC	Suma de enteros con acarreo	1
ADDF	Suma de valores de p.f.	1
ADDI	Suma de enteros	1
AND	Operación AND a nivel de bits	1
ANDN	AND a nivel de bits con complemento	1
	de uno de los operandos	1
ASH	Corrimento aritmético	1
CMPF	Compara valores de p. f.	1
CMPI	Compara enteros	1
FIX	Convierte valor de p.f. a entero	1
FLOAT	Convierte entero a p.f.	1
LSH	Corrimiento lógico	1
MPYF	Multiplica valores de p.f.	1
MPYI	Multiplica enteros	1
NEGB	Negación entera con préstamo	1
NEGF	Negación de p.f.	1
NEGI	Negación entera	1
NORM	Normalización de un valor de p.f.	1
NOT	Complemento lógico a nivel de bits	1
OR	Operación OR a nivel de bits	1
RND	Redondeo de un valor p.f.	1
ROL	Rotación a la izquierda de un bit	1
ROLC	Rotación a la izquierda de un bit,	1
	incluyendo al bit de acarreo	1
ROR	Rotación a la derecha de un bit	1
RORC	Rotación a la derecha de un bit,	1
	incluyendo al bit de acarreo	1
SUBB	Sustracción de enteros con préstamo	1
SUBC	Sustracción condicional de enteros	1
SUBF	Sustracción de valores de p.f.	1
SUBI	Sustracción de enteros	1
SUBRB	Sustracción inversa de enteros con	1
	préstamo	1
SUBRF	Sustracción inversa de valores	1
	de p.f.	1
SUBRI	sustracción inversa de enteros	1
TSTB	AND a nivel de bits entre dos ente-	1
	ros, pero sin sustitución de regs.	1
XOR	Operación OR exclusiva	1

A.11.3 Operaciones con tres operandos

Estas operaciones son un beneficio directo de la duplicación de los buses internos en el CPU (ver operación del bus interno, en este mismo apéndice). En este caso el CPU toma dos

operandos fuente, ya sea de registros o memoria, o bién un operando y un contador, y deposita el resultado de la operación en un registro. Las 17 instrucciones de tres operandos se distinguen por terminar siempre con un número 3.

Instrucción	*Descripción*	*Ciclos*
ADDC3	Suma con acarreo	1
ADDF3	Suma de valores de p.f.	1
ADDI3	Suma de enteros	1
CMPF3	Compara valores de p. flotante	1
CMPI3	Compara enteros	1
LSH3	Corrimiento lógico	1
MPYF3	Multiplicación de valores p.f.	1
MPYI3	Multiplicación de enteros	1
OR3	Operación OR a nivel de bits	1
SUBB3	Sustracción de enteros con préstamo	1 1
SUBF3	Sustracción de valores de p.f.	1
SUBI3	Sustracción de enteros	1
TSTB3	Operación AND a nivel de bits, en versión de 3 operandos.	1 1
XOR3	Operación OR exclusiva, en versión de 3 operandos.	1 1

A.11.4 Instrucciones para control de flujo

Este grupo comprende a 16 instrucciones utilizadas para realizar saltos, llamar a subrutinas, ejecutar repeticiones de segmentos de código, efectuar ciclos. Cabe hacer notar la existencia de saltos retardados, en los cuales las siguientes 3 instrucciones después del salto son cargadas sin incrementar el PC, dando por resultado un salto en un solo ciclo, lo que a su vez agiliza la ejecución en pipeline.

Instrucción	*Descripción*	*Ciclos*
Bcond	Salto condicional (estándar)	4
BcondD	Salto condicional (retardado)	1
BR	Salto incondicional (estándar)	4
BRD	Salto incondicional (retardado)	1
CALL	Llamada a subrutina	4
CALLcond	Llamada condicional a subrutina	5
DBcond	Decremento y salto condicional (estándar)	4
DBcondD	Decremento y salto condicional (retardado)	1
IDLE	Desocupado hasta interrupción	1
NOP	No realiza ninguna operación	1
RETIcond	Regreso de interrupción condicionalmente	4
RETScond	Regreso de subrutina condicionalmente	4
RPTB	Repite bloque de instrucciones	4
RPTS	Repite una sola instrucción	4
SWI	Ejecuta una interrupción del emulador (instr. reservada)	4

A.11.5 Instrucciones para operar sincronizadamente

El microprocesador TMS320C30 posee cinco instrucciones diseñadas especialmente para permitir comunicación entre procesadores y el manejo de señales externas de sincronización.

Instrucción	*Descripción*	*Ciclos*
LDFI	Carga de valor de p.f., sincronizadamente	1*
LDII	Carga de entero, sincronizadamente	1*
SIGI	Señalización externa a través de los pines XF0 y XF1	1 1
STFI	Almacena valor de p.f., sincronizadamente	1
STII	Almacena un entero, sincronizadamente	1

* la duración de estas instrucciones es de un ciclo siempre y cuando la bandera XF = 0.

A.11.6 Instrucciones para operaciones en paralelo

Como consecuencia de la duplicación interna de buses, cierto número de operaciones de carga, aritméticas y lógicas están permitidas para su ejecución de manera paralela. Al ser escrito el código fuente se indica la operación paralela con una doble barra (||) entre ambas operaciones. A continuación se listan los pares de instrucciones permitidos. Es claro que se debe buscar el paralelismo durante la programación, puesto que todas las operaciones paralelas permitidas se ejecutan en un solo ciclo.

ABSF ‖ STF	NEGI ‖ STI
ABSI ‖ STI	NOT3 ‖ STI
ADDF3 ‖ STF	OR3 ‖ STI
ADDI3 ‖ STI	STF ‖ STF
FIX ‖ STI	SUB3 ‖ STI
FLOAT ‖ STF	XOR3 ‖ STI
LDF ‖ STF	LDF ‖ LDF
LDI ‖ STI	LDI ‖ LDI
LSH3 ‖ STI	MPYF3 ‖ ADDF3
MPYF3 ‖ STF	MPYF3 ‖ SUBF3
MPYI3 ‖ STI	MPYI3 ‖ ADDI3
NEGF ‖ STF	MPYI3 ‖ SUBI3

Referencias

[1] Texas Instruments: *TMS320C3x, User's Guide*, Texas Instruments Inc., E.U.A., 1990.

[2] Panos Papamichalis, Ph.D.: *Digital Signal Processing. Applications with TMS320 family.*, vol.3 (Theory, Algorithms and implementations), E.U.A.Texas Instruments Inc., marzo 1990.

Apéndice B

Herramientas de desarrollo, la EVM

B.1 Introducción

En este apéndice se hace una breve descripción de la herramienta de desarrollo utilizada para implementar algunos de los algoritmos propuestos. El módulo de evaluación (EVM) es una tarjeta diseñada para utilizarse con el software depurador de código (*C Source Debugger*), el cual ofrece al usuario la posibilidad de realizar una emulación en pantalla, a la vez que los algoritmos se ejecutan en la tarjeta. La intervención del Controlador de Bus de Prueba (TBC) permite la emulación directa en la tarjeta, permitiendo parar la ejecución del programa, a la vez que se visualizan registros y memoria. El uso de un puerto de emulación MPSD mediante el TBC permite cargar a través de dicho puerto el programa en la EVM.

Se define un protocolo de comunicación entre el host (una computadora IBM PC/AT compatible) y la tarjeta objetivo, es decir la EVM. Dicho protocolo, el cual se describe en detalle, basa su operación en el intercambio de información entre registros mapeados en memoria, lo que permite desarrollar software de propósito específico para una aplicación dada utilizando un lenguaje de alto nivel que permita el acceso a memoria, como lo es el lenguaje C.

El programa debe ser previamente formateado según la especificación COFF (common object file format) mediante el cual es posible la generación de código objeto a partir de segmentos de programa escritos tanto en ensamblador como en ANSI C. A partir de dicho código objeto se genera código ejecutable en un archivo `.out` el cual debe ser cargado en la tarjeta mediante el programa **`evmload.exe`** o bién mediante el ambiente del depurador.

B.2 Características generales

B.2.1 Interfaz con el bus anfitrión

La tarjeta emuladora sigue las especificaciones para el bus IBM PC/AT de tarjetas de 8 bits, para insertarse en una ranura libre de este bus. En la figura B.1 se esquematiza la tarjeta EVM y sus componentes principales.

La diferencia más notoria sobre otros emuladores existentes consiste en lo que sus diseñadores denominan emulación en el sistema mismo (*embedded in-system emulation*). Esto significa que la emulación se reliza dentro del sistema objetivo (la EVM) realizándose la

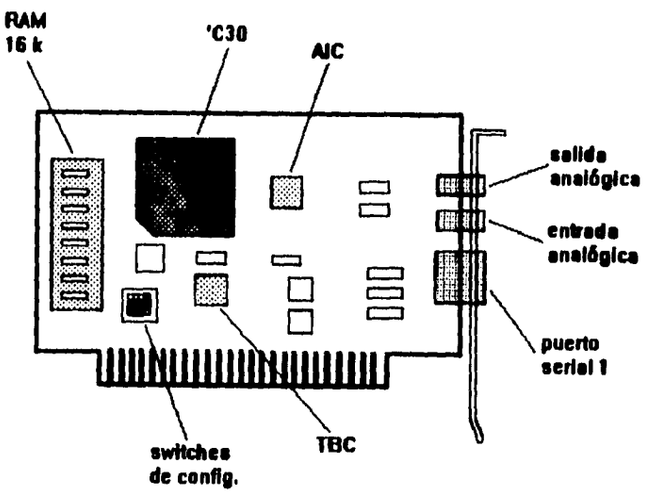

Figura B.1: Módulo de Evaluación (EVM) para el TMS320C30

interfaz con el host mediante un circuito dedicado a ello: el 74ACT8990 *Test Bus Controller* (TBC). Dicho circuito controla el intercambio de información entre el TMS320C30 y el bus de la computadora anfitriona.

Un programa ejecutando desde la memoria de la computadora personal, se comunicará en realidad con el TBC mediante el *poleo* de seis direcciones de 16 bits . El TBC interactuará con el TMS320C30 mediante interrupciones por Hardware, a través de los pines INT0, INT1 e INT2. Esto quiere decir que la comunicación desde un programa anfitrión en la computadora opera mediante poleo, mientras que el TMS320C30 se comunica con el TBC a través de interrupciones de Hardware, lo que le proporciona a este último total autonomía del anfitrión mientras no se realice una requisición de intercambio de información.

La configuración del sistema como se ha referido, provee una tasa moderada de intercambio de información (alrededor de 200 kbytes por segundo), sin embargo garantiza la sincronización de dicha transferencia, dado que para que un dato sea escrito o leído por parte del host (la PC) es necesario realizar un poleo y un clareo de banderas en los registros mapeados en memoria. Bajo ciertas condiciones, es posible evitar dicho poleo, lo que eleva el *throughput*[1] hasta 400 kbytes por segundo. Como hardware adicional en la generación de señales de protocolo (*glue logic*) se incluyen un par de PAL (dispositivos lógicos programables de uso generalizado), y dos *tranceivers* 74ALS652. El formato de interfaz entre el TBC y

[1]Throughput es un término generalmente aceptado para referir la cantidad de información digital que puede obtenerse como salida de un sistema, se utiliza como un parámetro de desempeño.

el puerto de emulación del TMS320C30 sigue las especificaciones MPSD (*modular port scan device*) de Texas Instruments.

B.2.2 Memoria externa

El módulo de evaluación posee un banco de memoria estática SRAM de 16K palabras con cero estados de espera - se halla directamente conectado al espacio de direcciones del bus primario. Las memorias utilizadas son de la marca Cypress modelo CY7C164-35VC con un tiempo de acceso de 35 ns, lo que requiere un reloj de 30 MHz como base de tiempo en el 'C30 para satisfacer los requerimientos de la SRAM. Este banco de memoria dota al módulo de cierta autonomía en cuanto a capacidad de almacenamiento se refiere. Es posible almacenar aproximadamente dos segundos de una señal muestreada a 8 kHz sin ser necesario el intercambio de información con el host.

B.2.3 Interfaz analógica

De los dos puertos seriales de los que dispone el TMS320C30, el primero de ellos (puerto 0) se ha utilizado para comunicarse con un controlador de interfaz analógica: el TLC32044 *analog interface controller* (AIC). Dicha interfaz necesita de una base de tiempo, por lo que se utilizó para ello el primer temporizador, además del pin XF0 como $\overline{RESET}$ del AIC. Este controlador provee de una interfaz A/D y D/A con 14 bits de resolución, tasa de muestreo y filtrado programables, que cubre satisfactoriamente los requerimientos de una arquitectura para procesar señales de voz.

La entrada analógica acepta voltajes en un rango de +/- 1.5 V y de +/- 3 V. Se incluye también una etapa de acondicionamiento de la señal proveniente del exterior consistente en un seguidor y un amplificador con ganancia 2, basados en amplificadores TL072 con entrada JFET. La salida analógica acepta una carga mínima de salida de 300Ω, sin embargo, para poder ser manejada por un equipo amplificador comercial, es necesario acoplarle para manejar cargas de hasta 4Ω. Con una salida típica de 1.5 Vpp, el AIC se conecta a una etapa de atenuación con una ganancia de x0.2, para posteriormente ser amplificado por un factor de x20 en el amplificador de potencia LM386, de manera que la ganancia de voltaje resultante es de x4. Por ejemplo, para una carga de 8Ω con una salida inicial de 1.5 Vpp el voltaje resultante a la salida del LM386 será de 6 Vpp.

B.2.4 Puerto serial externo

El puerto serial 1, completamente independiente de aquel que se usa como interfaz A/D y D/A, se ha previsto como un medio de comunicación al exterior de la EVM. Se incluye además en el conector de salida de 10 pins la señal XF1. Los pins disponibles en dicho conector son:

Pin	Señal	Pin	Señal
1	GND	2	FSR1
3	CLKX1	4	DR1
5	DX1	6	TCLK1
7	FSX1	8	XF1
9	CLKR1	10	GND

Los usos que pueden darse a dicho conector son diversos: conexión con otras EVM para ejecutar procesamiento distribuido, conexión a otro dispositivo serial para recibir información, como puede ser el backplane de un PBX, etc.

B.3 Configuración y generación de código

B.3.1 Configuración del mapa de memoria de la tarjeta

Es posible configurar algunas características de la tarjeta mediante microinterruptores. Con los interruptores marcados SW1, SW2 (figura B.1) se selecciona la dirección base de los registros utilizados por el programa de aplicación en el host para la comunicación hacia la EVM. La primera dirección base debe ser la opción por default, a menos que exista otro periférico (por ejemplo, un *mouse*) en ese espacio de memoria.

SW1	SW2	Dirección Base
ON	ON	0x0240-0x025F
ON	OFF	0x0280-0x029F
OFF	ON	0x0320-0x033F
OFF	OFF	0x0340-0x035F

Los interruptores SW3 y SW4 por lo general se mantienen en ON para permitir modo de microprocesador y la salida SBM en alta impedancia, respectivamente.

B.3.2 Configuración interna del TMS320C30

Al inicio de cualquier programa que sea cargado en la memoria de la tarjeta, es absolutamente necesario realizar una inicialización de algunas variables y banderas internas. A continuación se muestra un segmento de programa que efectúa dicha inicialización, habilitando solo las interrupciones que son necesarias, como aquellas utilizadas para comunicarse con el AIC. Esta secuencia de instrucciones puede variar un poco dependiendo principalmente de los periféricos que se utilicen y de las interrupciones que se manejen.

```
stack    .usect  "stack",100h
   .sect  "vectors"
PARMS:
reset    .word   sysinit ; al arrancar ejecuta sysinit
int0     .word   interr0
int1     .word   interr1
```

```
int2        .word   interr2
int3        .word   interr3
xint0       .word   transmit0
rint0       .word   receive0 ; RINT0 es causada por el AIC al mandar un dato,
xint1       .word   transmit1 ; y es atendida por receive0
rint1       .word   recieve1
tint0       .word   timer0
tint1       .word   timer1
dint0       .word   dmadone

   .sect   "comdata"
stack_addr .word   elstack      ; dirección del stack
dma_ctl     .word   000808000h  ; registro de control global del dma
mcntlr0     .word   000808064h  ; registro de control del bus primario
mcntlr1     .word   000808060h  ; registro de control del bus secundario
t0_ctladdr .word   000808020h  ; reg. control global timer 0
t1_ctladdr .word   000808030h  ; reg. control global timer 1
p0_addr     .word   000808040h  ; reg. control global puerto serial 0
enbl_sp0_r .word   000000020h  ; palabra para habilitar int. por Rx en p.s. 0
t0_ctlinit .word   0C00002C1h  ; habilita timer 0 como salida de reloj
; a (H)1/2 (reloj interno), el timer
; continua funcionando aun cuando el CPU este
; en emulación
p0_global  .word   00e970300h   ; palabra para configurar el p.s. 0
CACHE              .set    1800h ; palabra para habilitar cache
ENBL_GIE           .set    2000h ; palabra para habilitar interrupciones
ENBL_XINT0         .set    0010h ; habilita la interrupción de Tx del p.s. 0
ENBL_RINT0         .set    0020h ; habilita la interrupción de Rx del p.s. 0

   .text
sysinit:
   xor     ie,ie ; borra IE y deshabilita todas las ints.
   xor     if,if ; también borra todas las banderas de int. en IF
   ldp     PARMS ; carga DATA PAGE POINTER con los MSBs de PARMS
 ; para poder usar direccionamiento directo
   ldi     @stack_addr,sp ; carga un 100h en el STACK POINTER
   ldi     CACHE,st ; habilita el bit de cache en el STATUS REGISTER
   ldi     0,r0     ; carga 0 en R0
   ldi     @mcntlr0,ar0 ; carga AR0 con la direccion del PRIMARY BUS
; CONTROL REGISTER
   sti     r0,*ar0      ; pone el bus I/O en READY
   ldi     @mcntlr1,ar0 ; carga AR0 con la dirección del EXPANSION BUS
; CONTROL REGISTER
   sti     r0,*ar0      ; tambien lo pone en READY
   OR      80h,ST       ; habilita overflow mode en STATUS REGISTER
```

```
   or      ENBL_GIE,st ; habilita bit GLOBAL INT, que es el 13 de ST

interr0:   reti
interr1:   reti
interr2:   reti
interr3:   reti
transmit0: reti
transmit1: reti
recieve1:  reti
timer0:    reti
timer1:    reti
dmadone:   reti

*** como ejemplo, se propone una rutina de atención de interrupción, ******
*** que atienda aquella que se genera cuando se recibe un dato en p.s. 0 **

receive0:  push    st                ; salva en el stack el valor de
   push    r0                  ; algunos registros
   push    R3
   push    ar0
   push    dp

   ldp     PARMS

**** instrucciones propias de la rutina ****

   pop     dp                  ; saca del stack el valor de los
   pop     ar0                 ; registros salvados
   pop     R3
   pop     r0
   pop     st
   reti
  .end
```

B.3.3 Generación de código

Una vez que se ha COMPILADO o ENSAMBLADO cualquier programa para el TMS320C30, se genera un archivo `.obj` el cual contiene ya el código de máquina, en formato COFF (Common Object File Format) que es posible ejecutar en un sistema basado en este microprocesador.

Ensamblado

Si se trata de un programa escrito exclusivamente en lenguaje ensamblador, se utiliza el programa `asm30.exe` como sigue:

```
asm30 [ archivo_de_entrada [ archivo_objeto [archivo_de_lista ]]] [-opciones]
```

las opciones más comunes son: **-v** especifica la versión, pude ser -v30 para el TMS320C30 o -v40 para el TMS320C40.

-l produce un archivo de listado (.LST).

-i especifica un directorio para búsqueda de archivos especificados en las directivas `.copy`, `.include` o `.mlib`.

-c realiza la compilación con sensibilidad a la mayúsculas.

-x produce una tabla de referencias cruzadas al final del archivo de listado.

Una alternativa es invocar al programa sin ningun argumento, lo que ocasiona que el programa pida entonces los argumentos al usuario.

Ligado

Sin embargo, es necesaria una última etapa de LIGADO en la cual se asignan las direcciones definitivas para que el código ejecutable sea localizado dentro del mapa de memoria del sistema objetivo. En este caso dicho sistema objetivo es la EVM, y es necesario realizar esta etapa de ligado con información adicional sobre la localización de los datos, del código de programa, memorias RAM y ROM, etc. Tal acción se realiza con el programa `lnk30.exe`, cuya sintaxis es la siguiente:

```
lnk30 [-opciones] archivo_1 ... archivo_n
```

Las opciones mas usuales son:

-m produce un archivo de mapeo (.MAP)

-o especifica a continuación el archivo de salida (por default A.OUT)

-q al momento de ligar no aparece el rótulo de TI

-x fuerza a que relean las librerias y encuentre referencias no encontradas en la primera pasada

-a produce código con direcciones absolutas

-r produce código con direcciones relocalizables

Una opción común de ligado es la siguiente:

```
lnk30 archivo  archivo_de_comandos -o archivo_de_salida
```

El ligador también tiene la opcion de ser invocado sin argumentos lo que ocasiona que dicho programa pida los datos necesarios al usuario.

Archivos de comandos

Para que el ligador conozca la localización de la memoria dentro de un sistema objetivo, es necesario especificarle tal información en un archivo de texto con extensión `.cmd` (archivo de comandos). En dicho archivo de comandos, se especifica la naturaleza de los bloques de memoria usados (lectura y escritura, solo lectura) así como su localización y extensión dentro del mapa de memoria. A cada bloque se le da un nombre y a su vez se localizan en ellos las diferentes *secciones* de código. La división en secciones del código al momento de ligar obedece a que tal código puede tener diferentes características, siendo en algunas ocasiones código de programa, una tabla de datos fija, o una tabla de datos variable.

En el formato .coff se definen tres secciones por default:

sección .text contiene código ejecutable.
sección .data contiene datos inicializados e invariables.
sección .bss es un espacio reservado para variables no inicializadas (por ejemplo los estados internos de un filtro).

Además, el ligador permite que el usuario pueda crear otras secciones con características similares, y básicamente las contempla en dos categorías:

secciones inicializadas contienen datos o código ejecutable, las seccions .text y .data son secciones inicializadas. El usuario puede crear nuevas secciones inicializadas con las directivas .sect y .asect.
seciones no inicializadas Se utilizan para reservar espacio en memoria para datos no inicializados. La sección .bss es de este tipo y el usuario puede definir nuevas secciones con la directiva .usect.

En la sección anterior pueden encontrarse varios ejemplos de secciones creadas por el usuario, con los nombres "stack", "vectors" y "comdata".

A continuación se presenta el contenido de un archivo de comandos utilizado para ligar el código de inicialización listado en la sección anterior.

```
MEMORY
{
INT_V : org = 0x000000, len = 0x40
SRAM  : org = 0x000040, len = 0x1FC0
a     : org = 0x002000, len = 0x2000
RAM0  : org = 0x809800, len = 0x400
RAM1  : org = 0x809C00, len = 0x400
}

SECTIONS
{
 vectors: {} > INT_V
 comdata: {} > SRAM
.text   : {} > SRAM
 buffer : {} > a
 stack  : {} > RAM1
}
```

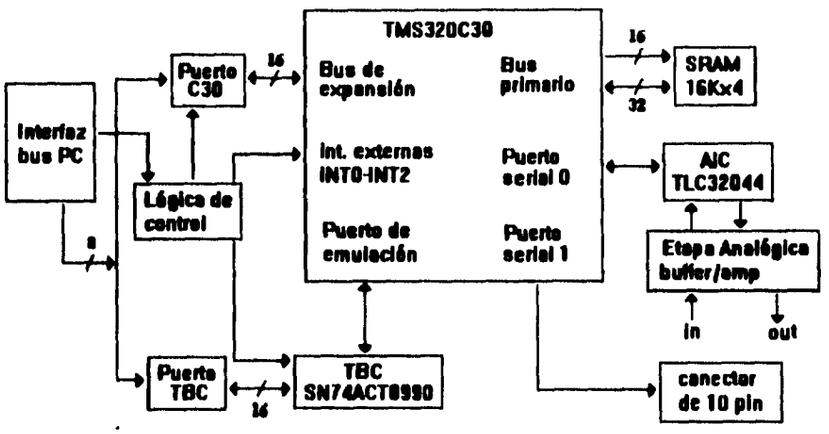

Figura B.2: Interconexión entre el bus anfitrión, el TBC y el TMS320C30

B.4 Comunicación entre el host y la EVM

La microcomputadora anfitriona, es decir el *host*, tiene comunicación con la EVM bajo un esquema maestro-esclavo donde obviamente el host ejecuta el papel de maestro. Como ya se ha mencionado, la interfaz entre ambos la realiza un circuito denominado Test Bus Controller (TBC), el cual es direccionado desde el bus AT mediante PAL's que actúan como decodificadores de memoria, y que también se utilizan entre el TBC y el 'C30 para sincronizar adecuadamente la producción de las interrupciones con la base de tiempo propia del microprocesador 'C30. Un esquema de la interconexión del bus PC/AT (host), el TBC y el TMS320C30 se muestra a continuación.

B.4.1 Operaciones de Lectura/Escritura desde el host hacia el TBC

Se direccionan 6 registros desde el bus PC/AT, los cuales se localizan física y lógicamente en el TBC, y a través de los cuales un programa de aplicación interactua con el TMS320C30. Aunque la mayor parte de ellos son de 16 bits, uno de ellos en realidad no existe mas que como una dirección y al realizarse una escritura en él se ejecuta un reset por software (registro *SOFTRESET*). A continuación se muestran los desplazamientos a partir de una dirección base (ver configuración) de los registros mencionados, así como su mnemónico asociado y su tamaño en bits.

Registro	*Desplazamiento*	*tamaño (bits)*	*Acceso*
RESERVADO	0x0000 - 0x0008	-	-
CONTROL5	0x000A	16	L/E
RESERVADO	0x000C - 0x0012	-	-
MINOR CMD	0x0014	16	L/E
RESERVADO	0x0016 - 0x001E	-	-
STATUS 0	0x0400	16	L
RESERVADO	0x0402 - 0x041F	-	-
COM CMD	0x0800	8	L/E
COM DATA	0x0808	16	L/E
SOFT RESET	0x818	0	E

B.4.2 Manejo de eventos a través del TBC

El TBC contempla cuatro eventos provenientes del bus PC/AT, los cuales a su vez disparan las correspondientes interrupciones al TMS320C30. Se les enumera EVT0 a EVT3 y su utilización es la siguiente:

EVT0. Embedded Simulation Support Evento no modificable, se utiliza como parte del protocolo entre en TBC y el puerto de emulación del TMS320C30.

EVT1. Host Read Acknowledge Se activa (EVT1=1) cuando el TMS320C30 escribe al registro de comunicaciones. Entonces el host puede leer el dato de dicho registro y poner en cero esta bandera para que una futura escritura del C30 pueda ser detectada.

EVT2. Host Write Acknowledge Se activa (EVT2=1) cuando el TMS320C30 efectúa una lectura del registro de comunicaciones. Esto quiere decir que el C30 ha recogido el dato. Acto seguido el host debe poner de nuevo en cero a esta bandera y puede escribir de nuevo.

EVT3. TMS320C30 Reset Source Mientras esta bandera sea mantenida en 1 en C30 se mantiene en estado de reset. existe un registro cuyo direccionamiento provoca dicho estado.

B.4.3 Escritura de datos

Para realizar la escritura de un dato localizado en una variable en la memoria del host hacia el TMS320C30, se lleva a cabo una serie de pasos, los cuales se ejemplifican en el siguiente segmento de un programa en Microsoft C:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK1       0x0004
#define MASK2       0x6044

/* la variable 16_bit_data contiene el valor a ser escrito */
```

```
unsigned short 16_bit_data;

do {

/* 1. borra la bandera EVT2 localizada en el reg. MINOR_CMD */

outpw(IOBASE + MINOR_CMD, MASK1);

/* 2. escribe el dato en el registro de comunicaciones */

outpw(IOBASE + COM_DATA, 16_bit_data);

/* 3. actualiza el registro de estado del TBC */

outpw(IOBASE + MINOR_CMD, MASK2);

/* 4. verifica que el dato ha sido leído del lado del TMS320C30 */

}while( inpw(IOBASE + STATUS0) & MASK1);

/* actualiza el registro de estado */

outpw(IOBASE + MINOR_CMD, MASK2);

/* 5. una vez abandonado el ciclo anterior se puede empezar de
nuevo desde 1 */
```

B.4.4 Lectura de Datos

Para recoger un dato del registro de comunicaciones, se debe polear la bandera EVT2, hasta que se indique la presencia de un dato en el registro de comunicaciones, como se muestra en el siguiente segmento de código:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK0       0x0002
#define MASK2       0x6044

unsigned short 16_bit_data;

do {

/* actualiza el registro de estado del TBC */
```

```
outpw(IOBASE + MINOR_CMD, MASK2);

/* si la bandera EVT1 es activa, se borra, en otro caso
regresa a actualizar el registro de estado */

if (inpw(IOBASE + STATUS0) & MASK0)
outpw(IOBASE + MINOR_CMD, MASK0);

}while(!(inpw(IOBASE + STATUS0) & MASK0));

/* procede a leer el dato */

16_bit_data = inpw(IOBASE + COM_DATA);
```

B.4.5 Reinicialización por software

La inicialización del TMS320C30 se dá de manera automática cuando se le aplica energía por primera vez. Sin embargo, el TBC provee la manera de aplicarle un reset por software, a través del registro CONTROL5, de la siguiente manera:

```
#define IOBASE      0x0240
#define CONTROL5    0x000A
#define RESET_ON    0x0808
#define RESET_OFF   0x0800

outpw(IOBASE + CONTROL5, RESET_ON);

outpw(IOBASE + CONTROL5, RESET_OFF);
```

Existe una tercera manera de reinicializar al TMS320C30 sin la intervención del TBC. Como se había mencionado, el registro *SOFTRESET* no existe físicamente, pero su direccionamiento provoca que una señal activa en bajo se genere en el PAL UA5 el cual aplica el reset al microprocesador.

```
#define IOBASE      0x0240
#define SOFT_RESET  0x0818

outpw(IOBASE + SOFT_RESET, 0);
```

B.5 Manejo de información desde el TMS320C30

En el caso del TMS320C30, el manejo del protocolo de comunicaciones se realiza a través de interrupciones, a través de las cuales dicho dispositivo da servicio a los procesos de lectura/escritura por parte del host. Cuando el TMS320C30 desea efectuar una lectura o escritura, la lógica de direccionamiento implementada en las PAL's accesa los registros del TBC. El registro $COMDATA$, visto desde el espacio de direcciones del 'C30, se encuentra en todo el bus de expansión como un solo registro de 16 bits habilitado para lectura y escritura. Cualquier dirección contenida dentro del espacio $0x804000 - 0x805FFF$ corresponde a este registro.

B.5.1 Mapa de Memoria de la Tarjeta

A continuación se muestra el mapa de memoria del módulo de evaluación, donde se incluyen todos aquellos registros mapeados en memoria para el control de periféricos, así como el espacio estandar de datos y programa. La memoria puede utilizarse para almacenar programa, datos así como los vectores de reset, vectores de interrupción y vectores de trampa.

Dirección	*Función*
bus primario	
0x000000 - 0x003FFF	16K palabras de SRAM
bus de expansión	
0x804000	registro COM DATA hacia el host
0x808000	control global del DMA
0x808004	dirección fuente de DMA
0x808006	dirección destino de DMA
0x808008	contador de transferencia de DMA
0x808020	control global del timer 0
0x808024	contador del timer 0
0x808028	periodo del timer 0
0x808030	control global del timer 1
0x808034	contador del timer 1
0x808038	periodo del timer 1
0x808040	control global del puerto serie 0
0x808042	puerto control Tx p. serie 0
0x808043	puerto control Rx p. serie 0
0x808044	control de timer Rx/Tx p. serie 0
0x808045	contador de timer Rx/Tx p. serie 0
0x808046	periodo de timer Tx/Rx p. serie 0
0x808048	dato de Tx puerto serie 0
0x80804C	dato de Rx puerto serie 0
0x808050	control global del puerto serie 1
0x808052	puerto control Tx p. serie 1
0x808053	puerto control Rx p. serie 1
0x808054	control de timer Rx/Tx p. serie 1
0x808055	contador de timer Rx/Tx p. serie 1
0x808056	periodo de timer Tx/Rx p. serie 1
0x808058	dato de Tx puerto serie 1
0x80805C	dato de Rx puerto serie 1
0x808060	control del bus de expansión
0x808064	control del bus primario
0x809800 - 809FFF	2K palabras de RAM interna, acceso dual

B.5.2 Manejo de Interrupciones en el TMS320C30

Como se ha visto, las interrupciones por hardware son el medio mediante el cual el TBC avisa al microprocesador que se realizan operaciones de lectura/escritura. En este caso, las interrupciones se detectan por nivel durante el flanco de bajada de la señal H1. el 'C30 puede aceptar dos interrupciones de la misma fuente cada dos ciclos de reloj de H1.

Los PAL's encargados de generar las fuentes de interrupción para el 'C30 se implementaron como máquinas de 4 estados (con 2 flip-flops) con entrada asíncrona. dichos PAL's generan las señales $\overline{CNTLINT}$, $\overline{WRINT}$ y $\overline{RDINT}$ que controlan las correspondientes

entradas INT0, INT1 e INT2 en el TMS320C30. La cuarta entrada disponible para interrupción, se deshabilita mediante una resistencia de pull-up a +5V. Para asegurar que solo sea generada una interrupción en cada operación de lectura o escritura por parte del host, después de una interrupción el generador de interrupciones espera a que se genere un señal de lectura o escritura y que esta sea inactiva, para después pasar la correspondiente interrupción al TMS320C30 a trvés de los pines INT0 a INT2.

Cada una de estas entradas tiene una función en el protocolo de comunicaciones, a saber:

INT0. command interrupt. Su estado activo indica que el host depositó un comando en el registro de comandos. Se genera cuando el host escribe un dato de 8 bits, pero no cuando el host hace una lectura, lo que le posibilita a este último polear el registro de estado del 'C30 sin ocasionar cambios en algún registro.
INT1. data write interrupt. Su estado activo indica que el host realizó una escritura de 16 bits en *COMDATA*.
INT2. data read interrupt. su estado activo indica que el host realizó una lectura de 16 bits de *COMDATA*.

El uso de interrupciones por hardware posibilita al controlador de DMA el dar servicio a la entrada y salida de datos.

B.6 Manejo del Controlador de Interfaz Analógica

El controlador de interfaz analógica (AIC) es el dispositivo que hace posible que los algoritmos diseñados puedan ser implementados en tiempo real con las siguientes limitaciones:

1. El rango de voltajes que acepta es cuando mas de +- 3 Volts.

2. La frecuencia de muestreo máxima configurable es de 19200 Hz.

Asimismo, son programables las características anteriores mas:

1. Un filtro corrector $sin(x)/x$ en la salida del convertidor D/A.

2. Un filtro paso bajas antialiasing de entrada de frecuencia programable.

El AIC debe ser configurado mediante un protocolo serial, posee un pin de reset y necesita también una base de tiempo. Estas cuestiones fueron resueltas en la arquitectura de la tarjeta como sigue:

1. Se utilizó el puerto serial 1 para comunicarse con el AIC.

2. El timer 0 se utilizó como base de tiempo.

3. El pin XF0 funciona como señal de reset para el AIC.

Existen varias consideraciones que se toman para realizar una adecuada comunicación entre en AIC y el TMS320C30: el AIC es capaz de realizar el intercambio de información con otro dispositivo mediante un protocolo serial asíncrono (es decir, a una tasa variable de transmisión) y con una longitud de palabra de datos de 16 bits; además, el AIC maneja

datos de tipo entero, mientras que el TMS320C30 es un dispositivo de punto flotante, por lo que debe realizarse la conversión adecuada de la información.

El temporizador usado como base de tiempo se configura a una frecuencia de f_{MCLK} 7.5 MHz siendo la mitad de la señal interna $H1$ (es decir: 15MHz/2 = 7.5 MHz). El pin $XF0$ es programable modificando su estado en el registro **IOF**.

El AIC soporta dos tipos de transmisión:

1. Transmisión primaria. Cuando el dispositivo simplemente intercambia información desde el convertidor A/D ó hacia el convertidor D/A, los datos, de 14 bits, deben estar acomodados en los bits más significativos de la palabra de transmisión, *asignando dos ceros a los bits menos significativos.*

2. Transmisión secundaria. Este tipo de transmisión *se inicia cuando se envían al AIC palabras con los dos bits menos significativos con valores de uno.* A continuacion del inicio de la transmisión secundaria, se envían palabras que configuran tanto la frecuencia de muestreo de entrada como la de salida, la frecuencia de corte de filtro antialiasing así como la ventana de voltaje de los convertidores.

Tanto para Tx como para Rx deben configurarse dos parámetros, denominados A y B, que establecen las frecuencias de muestreo como sigue:

$$f_{SCF} = \frac{f_{MCLK}}{2A} \tag{B.1}$$

$$f_c = \frac{f_{MCLK}}{2AB} \tag{B.2}$$

Donde f_{SCF} es la frecuencia del filtro de capacitores conmutados de entrada (filtro pasobajas), y f_c es la frecuencia de muestreo. Para ello deben calcularse los números TA,TB para la transmisión, RA y RB para la recepción, que serán enviados posteriormente al inicio de una transmisión secundaria, como puede apreciarse en la figura B.3. Los números TA y RA se utilizan para realizar un ajuste fino de las frecuencias de muestreo, y su valor se resta al contenido original de TA y RA. El parámetro A tiene un rango de valores entre 1 y 31, mientras que B, con un bit más de resolución, puede variar entre 1 y 63. Además, existe un registro de control, accesible al colocar unos en los dos bits menos significativos de la palabra transmistida posteriormente al inicio de transmisión secundaria. En dicha palabra se encuentran los bits $d2$ a $d9$ con los siguientes significados:

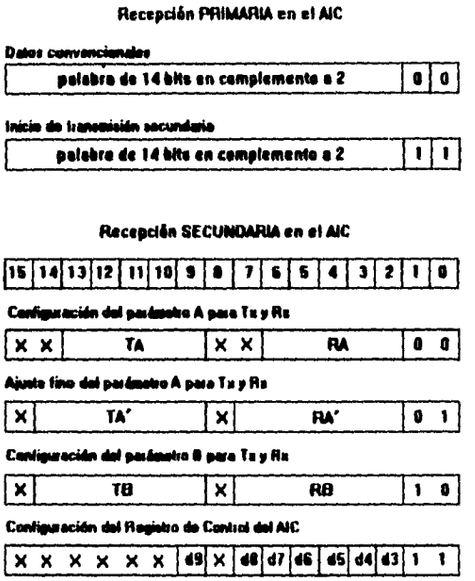

Figura B.3: Formatos de transmisión de datos hacia el AIC

Parámetros del registro de control del TLC32044

Bit	*Función*
d2	Habilita filtro paso-altas en A/D
d3	habilita "loopback"
d4	selecciona entrada analógica primaria(0) o secundaria (1)
d5	Tx y Rx síncrona
d7,d6	selecciona entrada +-1.5V (1,0) o entrada +- 3.0V (1,1)
d9	habilita filtro corrector $(sinx)/x$ en D/A

A continuación se muestra la secuencia de inicialización del AIC para configurarlo a una tasa de muestreo de 8 kHz:

```
aicreset:
   ldi     2,iof              ; configura XF0 para que sea salida con 0,
   ; con lo que se pone en reset al AIC
   ldi     @t0_ctladdr,ar0    ; carga AR0 con la dirección del TIMER 0
   ; GLOBAL CONTROL REG
   ldi     1,r1               ; con este 1, tclk0 será (H)1/2
   sti     r1,*+ar0(8)        ; pone ese 1 en el TIMER 0 PERIOD
```

```
   ldi     @t0_ctlinit,r1      ; carga R1 con la configuración del TIMER 0
   sti     r1,*ar0             ; pone la configuración en TIMER GLOBAL
   ; CONTROL REG
   ldi     @p0_addr,ar0        ; carga en AR0 la dir. del GLOBAL CONTROL
   ; REG del puerto serial 0
   ldi     111h,r1             ; carga en R1 la configuración del p.s. 0
   sti     r1,*+ar0(2)         ; carga la configuración en S.P. 0 Tx PORT
   ; CONTROL
   sti     r1,*+ar0(3)         ; carga la config. en S.P. 0 Rx PORT CONTROL
   ldi     @p0_global,r1       ; carga R1 con la config. para p.s. 0
   sti     r1,*ar0             ; escribe configuración en el GLOBAL CONTROL
   ; REG del p.s. 0
   xor     r1,r1               ; borra el contenido de R1
   sti     r1,*+ar0(8)         ; pone un 0 en el reg. de Tx del p.s. 0
   rpts    99                  ; espera a que hayan ocurrido 50 ciclos
   nop                         ; del timer 8 para que se estabilice ?
   ldi     6,iof               ; pone a XF0 en 1, saca del reset al AIC
   ; y comienza a enviarle su configuración
   call    wait_transmit_0     ; polea a ver si se ha transmitido algo
   ldi     3,r1                ; guarda un 3 en R1
   sti     r1,*+ar0(8)         ; escribe el 3 en el S.P 0 Tx DATA
   call    wait_transmit_0     ; espera a que se haya transmitido
   ldi     1a34h,r1            ; carga la palabra de config. del reg de
   ; control del AIC
   sti     r1,*+ar0(8)         ; envia dicha palabra al AIC
   ldi     *+ar0(12),r1        ; lee lo que haya en el S.P. 0 Rx DATA
   call    wait_transmit_0     ; espera a que esté listo el puerto
   ldi     3,r1                ; ahora, siguiendo el mismo protocolo
   sti     r1,*+ar0(8)         ; vuelve a mandar un 3
   call    wait_transmit_0     ; espera a que se vacie el puerto
   ldi     2a7h,r1             ; carga la config. de la frec. de muestreo
   sti     r1,*+ar0(8)         ; la envía
   ldi     *+ar0(12),r1        ; lee lo que haya en la recepción
   xor     if,if               ; borra todas las banderas de interrupción
   or      @enbl_sp0_r,ie      ; habilita el P.S. 0
   rets
wait_transmit_0:
   xor     if,if               ; borra todas las banderas de interrupción
wloop:    tstb    10h,if          ; checa la bandera de Tx efectuada
   bz      wloop
   rets
```

B.7 Software asociado: El depurador de código

El depurador de código es una poderosa herramienta de software que aprovecha la característica de emulación que posee la tarjeta. Mediante él es posible realizar una *emulación* en la cual, a diferencia de una *simulación* se realiza la ejecución de las instrucciones *realmente* en el interior del microprocesador, a la vez que se obtienen las ventajas típicas de los simuladores, como son:

1. poder realizar la ejecución paso a paso, o utilizando *breakpoints*
2. observar los cambios en diferentes variables a la vez
3. poder alterar aleatoriamente el contenido de las variables durante la ejecución
4. utilizar archivos como entrada y/o salida de datos
5. definir "macros" con los comandos mas usados
6. poder realizar el rastreo de variables en un sistema de múltiples ventanas

Al igual que el microprocesador al cual apoya, este programa está orientado hacia la programación en lenguaje C, por lo que sus instrucciones suelen tener similitudes principalmente en la sintaxis con la forma de expresar operaciones y direccionamientos en dicho lenguaje. El depurador se invoca ejecutando el comando `evm30.exe` y por default se inicializa con la información contenida en `evminit.cmd`, si se desea se puede editar este archivo o bien invocar el programa con otro archivo de comandos, mediante la sentencia

```
evm30 -t archivo.cmd
```

B.7.1 Comandos del depurador de código

A continuación se hace una reseña de las instrucciones mas usuales dentro del ambiente del depurador de código.

Instrucciones para cambiar de modo de programación

Instrucción	*Función*	*Ejemplo*
asm	rastrear programa en ensamblador	asm
c	rastreo en c (modo por default)	c
mix	modo mixto	mix

Teclas de función y otras

Instrucción	*Función*
< *F2* >	último comando
< *F3* >	extiende DISSASEMBLY y CPU ocultando FILE y CALLS
< *F4* >	cerrar la ventana actual de una estructura
< *F5* >	equivale a RUN
< *F6* >	brinca entre ventanas
< *F8* >	ejecuta paso a paso
< *F9* >	entrar a una subestructura (ver manipular datos)
< *F10* >	siguiente instrucción
< *TAB* >	recorre el buffer de comandos
< *ESC* >	salir

Instrucciones para manejo de ventanas

Instrucción	*Función*	*Ejemplo*
zoom	maximiza la ventana actual	zoom
move	mover la ventana actual	move
win	cambia de ventana	win CPU, win FIL, win CAL win MEM, win COM, win DIS
size	cambia dimensiones de la ventana	size

Instrucciones para ejecutar tareas del sistema

Instrucción	*Función*	*Ejemplo*
system	llamadas al DOS en modo SHELL	system, system "dir/w" [,0][,1] con 1 espera un ENTER con 0 regresa de inmediato
exit	salir del modo SHELL	exit
take	toma un archivo de comandos (.cmd)	take arch[.cmd][,0] el 0 inhibe el eco de los comandos
cd/chdir	cambia el directorio de trabajo	cd user
cls	borra la ventana de la ventana COM	cls
alias	definir un macro	alias macro1,"comando1;comando2"
	ejecutar el macro	macro1
	mostrar macros existentes	alias
unalias	cancelar el macro	unalias macro1
dir	listar el directorio actual	dir
use	nombrar directorios de fuentes adicionales	use otrodir
reset	resetear el sistema (la EVM)	reset
restart	continuar ejecución del programa	restart
quit	salir del depurador	quit

Instrucciones para desplegar y manipular datos y variables

Instrucción	*Función*	*Ejemplo*
wa	abrir ventanas para rastrear variables	wa variable[,etiqueta] (aparece la ventana watch)
	observar el contenido de una localidad	wa *0x100300
	observar una variable	wa variable [,etiqueta][o,x,i,etc.]
	observar como entero	wa i
	observar como float	wa f
	observar como double	wa d
	observar con etiqueta	wa i,NEW i
wd	borra una variable de la ventana	wd N (N es el renglon)
wr	borrar toda la ventana WATCH	wr
setf	cambiar la forma de desplegar tipo de datos	setf int, x (los enteros se verán como hexa)
	reestablecer el despliegue normal	setf*
	listar los tipos de datos y su modo de despliegue actual	setf
disp	mostrar en la ventana un dato:	
	mostrar una estructura	disp estructura
	mostrar en detalle un elemento de la estructura	hacer ¡click¿ en el elemento
	cerrar la ventana actual	presionar $< F4 >$
	entrar a una "subestructura"	presionar $< F9 >$
	conmutar entre "subestructuras" "hijas"	$< CTRL >< PgUp >$ $< CTRL >< PgDn >$
?	evaluar y desplegar el resultado de una expresión	
	evaluar una loc. de memoria	?*0x100200
	evaluar loc. de memoria forzando el despliegue como un tipo de datos	disp *(float *)0x100200
	evaluar el contenido de una variable	? variable[c,f,o,etc.]
mem	desplegar a partir de una localidad en la ventana MEMORY	mem 0x100200, pc [*,c,d,e,f,x,o,p,u]
	saber el valor de un símbolo	mem &simbolo
eval	evaluar una expresión	eval -R3
whatis	averiguar el tipo de una variable o una función	whatis variable, whatis funcion

Instrucciones para cargar y desplegar programas

Instrucción	*Función*	*Ejemplo*
load	cargar archivo ejecutable (.out)	load c:file[.out]
file	cargar un archivo ASCII en FILE	file myfile.ext
func	mueve el apuntador de la ventana FILE a una función específica	func funcion
dasm	desplegar código a partir de una función específica	dasm funcion
	desplegar código a partir de una función con fuente en C o ASM	dasm 0x100200
addr	desplegar código a partir de una función con fuente en C	addr[funcion][direcc]
reload	cargar archivo .out sin su tabla de símbolos	reload file1[.out]
sload	cargar solo la tabla de símbolos de un archivo .out	sload file1[.out]
	modificar una instrucción en lenguaje ensamblador	patch 0x100200[,instrucción]
calls	reabrir la ventana CALLS	calls

Instrucciones para utilizar puntos de prueba (breakpoints)

Instrucción	*Función*	*Ejemplo*
ba	marcar un punto de prueba	ba [dirección,función o etiqueta
br	desmarcar todos los puntos	br
bd	desmarcar un punto	bd 0x100200
bl	listar los puntos marcados	bl
clk	examinar el contenido de CLK, que cuenta ciclos de reloj entre breakpoints	? clk

Instrucciones para personalizar la pantalla

Instrucción	*Función*	*Ejemplo*
sconfig	cambiar archivo de configuración de la pantalla	sconfig archivo[.clr]
prompt	cambiar el prompt de la línea de comandos	prompt miprompt
ssave	salvar en archivo la configuración	ssave archivo.ext
scolor	modificar atributos de ventana	scolor menu-bar, yellow

Instrucciones para manejo de memoria

Instrucción	*Función*	*Ejemplo*
ma	dar de alta un bloque de memoria	ma 0x100200,0x0400,RAM/ROM
mc	conectar el bloque a un archivo de entrada	mc 0x100200,filein.ext,READ
mc	conectar el bloque a un archivo de salida	mc 0x100200,fileout.ext,WRITE
md	dar de baja un bloque de memoria	md 0x100300
mi	desconectar un puerto de memoria	mi 0x100200,READ
LDI	leer de un puerto en ensamblador	LDI@0x100200,R0
ml	despliega configuracion actual de memoria	ml
map	habilita/deshabilita el mapeo en memoria	map ON/OFF
mr	reinicializa tod el mapa de memoria	mr
fill	llena un bloque de memoria con un valor	fill 0x100200,0x100,0xffffffff
ms	salvar un segmento de código como un programa .coff	ms dir-de-inicio, longitud, archivo.ext

* solo se permite asociar a un puerto un archivo de entrada y otro de salida.
* un archivo puede tener asociados múltiples puertos.

Instrucciones para depuración de programas

Instrucción	*Función*	*Ejemplo*
step	recorrer N sentencias en ensamblador	step N
cstep	recorrer N sentencias en lenguaje C	cstep N
next	recorrer N sentencias en ensamblador pero sin entrar a las subrutinas	next N
cnext	recorrer N sentencias en C pero sin entrar a las subrutinas	cnext N
run	ejecutar un programa	run
runf	ejecutar un programa sin la intervención del puerto emulador (la EVM ejecuta libremente)	runf
halt	"conecta" de nuevo el puerto	halt
	de emulación a la EVM	
go	ejecutar una subrutina	go subrutina
runb	ejecutar hasta el siguiente breakpoint contando ciclos de CPU	runb (benchmarking)
ret	ejecuta el código de la función actual en C y regresa cuando termina	ret

Referencias

[1] Texas Instruments: *TMS320C3x C Source Debugger, User's Guide*, Texas Instruments Inc., E.U.A., 1991.

[2] Texas Instruments: *TMS320 Floating-Point DSP Assembly Language Tools, User's Guide*, Texas Instruments Inc., E.U.A., 1991.

[3] Texas Instruments: *TMS320C3x, User's Guide*, Texas Instruments Inc., E.U.A., 1990.

[4] Texas Instruments: *DSP Applications with the TMS320C30 Evaluation Module, App. Notes* , Texas Instruments Inc., E.U.A., 1991.

[5] Texas Instruments: *TMS320C30 Evaluation Module, Technical Reference*, Texas Instruments Inc., E.U.A., 1991.

Apéndice C

Listados y Rutinas

En este apéndice se listan algunas de las rutinas que se utilizaron en el programa desarrollado. Un compendio completo de rutinas se encuentra en el disco anexo a este trabajo, junto con archivos de prueba para el programa ADFO. Se presentan aquí solo las rutinas más significativas a manera de referencia.

C.1 Rutinas en lenguaje C

A continuación se incluyen algunas de las rutinas que implementan los algoritmos de análisis, síntesis y manejo de información en el programa ADFO. En el caso de las rutinas de análisis y síntesis existe compatibilidad con el ANSI C, mientras que para las demás rutinas se utilizan algunas librerías que pueden no estar soportadas en otros compiladores distintos a Microsoft C.

C.1.1 Rutinas de Análisis

```
#define WSIZE 240
#define SEGSIZE 160
float hamming_weights[WSIZE];
float candidato[6][6];

/*********************************************************************/
/* GENERACION DE LA VENTANA DE hamming Y CLAREO DE LA matriz
de candidatos */
/*********************************************************************/
void build_arrays()
{
 int i,j;
 for(i=0;i<WSIZE;i++)
   hamming_weights[i] = 0.54 - 0.46*cos(2*PI*i/(float) WSIZE);
 for(i=0;i<6;i++) for(j=0;j<6;j++) candidato[i][j]=0.0;
 for(i=0;i<20;i++) reflex_0[i]=reflex_1[i]=reflex_2[i]=0;
}
```

```
/*****************************************************************/
/* FUNCION GENERICA PARA APLICAR FILTROS fir */
/*****************************************************************/
void fir(float *input, float *output, float *fircoefs,
 int ntaps, int nouts)
{
  int i,j;
  float suma;
  input += ntaps-1;  /* se posiciona en el ntaps-esimo elemento */

  for(i=0; i<nouts; i++)
   {
suma = 0.0;
for(j=0; j<ntaps; j++) suma += fircoefs[j] * input[i-j];
output[i] = suma;
   }

}
/*****************************************************************/
/* FUNCION DE ventana_y_preenfasis() SOBRE *input, LA SECUENCIA
RESULTANTE LA DEPOSITA EN *output */
/*****************************************************************/
void ventana_y_preenfasis(float *input, float *output)
{
int i;
 for (i = WSIZE -1; i > 0; i--)
output[i] = (input[i] - 0.9375 * input[i-1]) * hamming_weights[i];
output[0] *= hamming_weights[0];
}

/*****************************************************************/
/* LA FUNCION DE autocorrelacion() DEBE OBTENER p + 1 TERMINOS
DE AUTOCORRELACION, DONDE p ES LE ORDEN DEL FILTRO LATTICE,
LA FUNCION DEPOSITA p + 1 TERMINOS EN autocor[0] ... autocor[P] */
/*****************************************************************/


void autoc(float *samp_array, float *autocor, int p, int nsamples)
{
 int i,j;
 float sum;
 for(i=0;i<=p;i++)
  {
   sum=0.0;
   for(j=0; j < nsamples-i;j++) sum += samp_array[j]*samp_array[i+j];
   autocor[i] = sum;
```

```
  }
}

/*****************************************************************/
/* ALGORITMO DE durbin(), EN *a ESCRIBE LOS COEFICIENTES DEL FILTRO iir,
EN *k LOS COEFICIENTES DE REFLEXION, Y DEVUELVE LA GANANCIA DEL SEGMENTO,
ojo: LOS COEFICIENTES SE DEPOSITAN DESDE EL INDICE 1 HASTA P DE AMBOS
ARREGLOS, EL PRIMER ELEMENTO NO SE USA */
/*****************************************************************/
float durbin(float *r, float *a, float *k, int p)
{
 int i,j,kk,l,jj;
 float suma;
 float a_temp[20];
 float e[20];

 for (l=0;l<p+1;l++) {
  a_temp[l] = 0.0;
  e[l] = 0.0;
  }

  e[0] = r[0];

  for ( i=1; i<=p; i++)
  {
   suma = 0.0;
   if (i>1)
 for(kk=i-1,j=1; j<=(i-1) ;j++,kk--)
 suma += a_temp[j]*r[kk];

k[i] = -( r[i] + suma ) / e[i-1];

a[i] = k[i];

if (i>1)
 for(jj=1; jj <= (i-1); jj++)
 a[jj] = a_temp[jj] + k[i] * a_temp[i-jj];

e[i] = (1 - k[i]*k[i]) * e[i-1];

for(l=1;l<=i;l++) a_temp[l] = a[l];
  }
  if (e[p]<0.0) e[p] *= -1.0;
  return( sqrt(e[p]) );


}
```

```
/*****************************************************************/
/* ALGORITMO DE schur() PARA OBTENER COEFICIENTES DE REFLEXION
Y GANANCIA DEL SEGMENTO, EN *ref COLOCA LOS COEFICIENTES DE
REFLEXION Y DEVUELVE EL VALOR DE GANANCIA
ojo: LOS COEFICIENTES SE DEPOSITAN DESDE EL INDICE 1 HASTA P DE AMBOS
ARREGLOS, EL PRIMER ELEMENTO NO SE USA */
/*****************************************************************/

float schur(float *ac, float *ref, int p)
{
 int i,m;
 float r,error = ac[0], G[2][20];

 if (ac[0] == 0.0) {
for (i=0; i<p; i++) ref[i] = 0.0;
return(0);
}

  for (i=0; i<p; i++) G[0][i] = G[1][i] = ac[i+1];

  for (i=0;;) {
ref[i] = r = -G[1][0] / error;
error += G[1][0] * r;

if (++i >= p) {
if (error<0.0) error *= -1.0;
return(sqrt(error));
}

for (m=0; m < p-i; m++) {
  G[1][m] = G[1][m+1] + r * G[0][m];
  G[0][m] = G[1][m+1] * r + G[0][m];
 }
   }

}
```

C.1.2 Funciones de Síntesis

```
/* arreglos de entrada y salida */

float disk_read_240[WSIZE];
float disk_write_162[SEGSIZE+2];
```

```
/* arreglos de SEGMENTOS de entrada al algoritmo */

float seg_in[WSIZE]; /* arreglo usado para acomodar la ventana de entrada */
float seg_a_out[SEGSIZE+2]; /* aqui se guarda la salida del filtro PBj */
float seg_b_out[WSIZE]; /* aqui se guarda la salida de hamming */

float autocorr_coefs[20];
float los_coefs_calculados[20];

float *disk_in = disk_read_240;
float *disk_out = disk_write_162;
float *algor_in = seg_in;
float *filter_out = seg_a_out;
float *hamm_out = seg_b_out;

float *hamm = hamming_weights;
float *autocoef = autocorr_coefs;
float *reflex_actual = los_coefs_calculados;

/* banderas y variable del detector de silencios */

#define SILENCE 44
#define SOUND 33
int segtype = SOUND;

float dod_sequence[40] =
{
     .249, -.262,  .363, -.362,  .100,  .367,  .079,  .078,  .010, -.277,
    -.082,  .376,  .288, -.065, -.020,  .138, -.062, -.315, -.247, -.078,
    -.082, -.123, -.039,  .065,  .064,  .019,  .016,  .032,  .018, -.015,
    -.029, -.021, -.018, -.027, -.031, -.022, -.012, -.010, -.010, -.004
};

float fir79[79] =
{
  0.0004,  0.0006,  0.0005, 0.0000, -0.0007, -0.0015, -0.0019, -0.0018,
 -0.0010,  0.0000,  0.0006, 0.0000, -0.0020, -0.0051, -0.0081, -0.0095,
 -0.0085, -0.0054, -0.0017, 0.0000, -0.0022, -0.0088, -0.0177, -0.0253,
 -0.0276, -0.0226, -0.0117, 0.0000,  0.0057,  0.0000, -0.0179, -0.0425,
 -0.0633, -0.0677, -0.0470, 0.0000,  0.0649,  0.1318,  0.1818,  0.2004,
  0.1818,  0.1318,  0.0649, 0.0000, -0.0470, -0.0677, -0.0633, -0.0425,
 -0.0179,  0.0000,  0.0057, 0.0000, -0.0117, -0.0226, -0.0276, -0.0253,
 -0.0177, -0.0088, -0.0022, 0.0000, -0.0017, -0.0054, -0.0085, -0.0095,
 -0.0081, -0.0051, -0.0020, 0.0000,  0.0006,  0.0000, -0.0010, -0.0018,
 -0.0019, -0.0015, -0.0007, 0.0000,  0.0005,  0.0006,  0.0004
};
```

```
/*********************************************************************/
/* FILTRO supresor de picos (SMOOTHING FILTER) */
/*********************************************************************/

int pitch_smoothing(int el_de_enmedio)
{
/* si V V V suaviza el valor del pitch */
if (pitch_n_0 && pitch_n_1 && pitch_n_2)
  return( (int) (0.25*pitch_n_0 + 0.5*el_de_enmedio + 0.25*pitch_n_2) );
else return(el_de_enmedio);
}

/*********************************************************************/
/* EL detector de silencios DECLARA UN SEGMENTO COMO SILENCIO
CUANDO SE CUMPLEN DOS CONDICIONES: que no haya cambio de signo
Y que la maxima oscilacion no sobrepase a "umbral" */
/*********************************************************************/

int silence_detect(float *inparray,float umbral,int nsamples)
{
int i;
int cambio_signo = 0;
float bigger_delta=0.0,delta;

for(i=1;i<nsamples-1;i++)
  { if ( (delta=inparray[i]-inparray[0]) < 0) delta *= -1.0;
if (delta > bigger_delta) bigger_delta = delta;
    if ( (inparray[i]*inparray[0]) < 0) cambio_signo = 1;
   }

 if ((bigger_delta < umbral) && (!cambio_signo)) return(1);
 else return(0);

}

/*********************************************************************/
/* LA FUNCION lattice() ACEPTA <una> MUESTRA DE ENTRADA AL SER
INVOCADA,NECESITA EL ARREGLO *k DE COEFS. DE REFLEXION, EL
ARREGLO *b DE SUS ESTADOS INTERNOS Y EL ORDEN p, DEVUELVE <una>
MUESTRA AL REGRESAR */
/*********************************************************************/
float lattice(float *b, float *k, float sample, int p)
{
  int i;
  sample -= b[p-1] * k[p-1];
  for (i=p-2; i>=0; i--)
   {
```

```
 sample -= b[i] * k[i];
 b[i+1] = b[i] + k[i]*sample;
}
   return b[0] = sample;

}
```

C.1.3 Rutinas de manejo de archivos

```
/* VARIABLES Y CONSTANTES UTILIZADAS */
/* HEADER PARA ARCHIVOS .WAV */
struct riff_header {
 char rId[4];   /* campo para "RIFF" */
 long int rLen;
 char wID[4];   /* campo para "WAVE" */
 char fId[4];   /* campo para "fmt " */
 long int fLen;
 short int wFormatTag;
 short int nChannels;
 short int nSamplesPerSec;
 short int nAvgBytePerSec;
 short int nBlockAlign;
 short int FormatSpecific;
 char OtherField[4];
 char dId[4];   /* campo para "data" */
 unsigned long int dLen;
 } data;

/* HEADERS PARA ARCHIVOS .C30 */
struct el_header{
char head[21];
char coding[5];
char uso[5];
int f_muestreo;
int l_datos;
char reservado[32];
}header;

struct el_header_LPC{
char head[21];
char coding[5];
char uso[5];
int f_muestreo;
int l_datos;
short int n_coefs;
```

```
char reserv[30];
}header_LPC;

/*=========================================================================*/
/* OBTIENE DATOS MUESTREADOS EN LA EVM Y LOS ESCRIBE A DISCO               */
/*=========================================================================*/
int record_and_store(char *filename,int nsamples,int sampfrec,int gain)
{
 FILE *fpw;
 short int i,j,coderror;
 char path[80];
 short int sample16;

 getcwd(path,80);strcat(path,"\\c30\\");strcat(path,filename);

 if ( (fpw = fopen(path,"wb")) == NULL)
   { vPopMsg(-1,-1,"Error Grabando Archivo .C30");
   return(0); }

strcpy(header.head,"TsisUNAMLDVSAEL C30 ");
strcpy(header.coding,"onda");
strcpy(header.uso,"grab");
header.f_muestreo = sampfrec;
header.l_datos = nsamples;
strcpy(header.reservado,"");

 fwrite(&header,sizeof(struct el_header),1,fpw);

   while(READ_CMD);WRITE_CMD(RECORDING);
   while(READ_CMD != RECORDING);
   CLR_READ_ACK;READ_DATA;

for(i = 0; i <= nsamples; i+=512)
{
   for (j=0; j<512; j++)
   {
CLR_READ_ACK;
do
{UPDATE_STATUS0;}
while(IS_READ_ACK);

 sample16 = READ_DATA;

 /* escribe la muestra a disco  */
 fwrite(&sample16,sizeof(short int), 1, fpw);

  if ( (coderror=ferror(fpw)) != 0 ) /* checa errores */
```

```
{fclose(fpw);
 vPopMsg(-1,-1,"Error al grabar archivo");
 return(0); break;}
}
WRITE_CMD(NONE);
WRITE_CMD(RECORDING);
while(READ_CMD != RECORDING);
READ_DATA;
}
WRITE_CMD(NONE);
fclose(fpw);
return(1);
}

/*************************************************************************/
/*      LA SIGUIENTE FUNCION TOMA UN ARCHIVO .WAV CUYO NOMBRE SE EXTRAE
DE LA VARIABLE edname Y LO CONVIERTE A FORMATO .C30 CON EL NOMBRE
ESPECIFICADO POR target, APLICA UN FACTOR DE AMPLIFICACION ESPECIFICADO
POR factor, LA FUENTE LA BUSCA EN \waves Y EL DESTINO LO PONE EN \c30 */
/*************************************************************************/

int export_wav_to_c30(char target[], int factor)
{
 FILE *fpsource, *fptarget;
 char paths[80],patht[80];
 char muestra8;
 short int sample,coderror;
 unsigned long int i,k,j,l;
 short int muestra16;

 /* CHECA QUE factor SEA VALIDO */

 if ( (factor>256) || (factor<(-256)) ) {
  vPopMsg(-1,-1,"Factor de Ganancia Erróneo"); return(0); }

/* no es necesario construir la ruta para "selected_file" */

 /* ABRE ARCHIVO WAV EN MODO BINARIO */
 if ( (fpsource = fopen(selected_file,"rb")) == NULL)
   { vPopMsg(-1,-1,"Error al abrir Archivo .WAV");return(0); }

 getcwd(patht,80);strcat(patht,"\\C30\\");strcat(patht,target);

 /* ABRE ARCHIVO C30 EN MODO BINARIO */

 if ( (fptarget = fopen(patht,"wb")) == NULL)
   { vPopMsg(-1,-1,"Error al abrir Archivo .C30");return(0); }
```

```
  /* LEE ENCABEZADO DE ARCHIVO WAV */
fread(&data,sizeof(struct riff_header),1,fpsource);

  /* PREPARA ENCABEZADO PAR ARCHIVO C30 */
strcpy(header.head,"TsisUNAMLDVSAEL C30 ");
strcpy(header.coding,"onda");
strcpy(header.uso,"xprt");
header.f_muestreo = data.nSamplesPerSec;
header.l_datos = data.dLen;
strcpy(header.reservado,"");
   /* ESCRIBE ENCABEZADO EN ARCHIVO C30 */
fwrite(&header,sizeof(struct el_header),1,fptarget);

for (;;)
{

fread(&muestra8,sizeof(char),1,fpsource);
if ( (coderror=feof(fpsource)) != 0 ) {fclose(fpsource); break;}
if ( (coderror=ferror(fpsource)) != 0 ) {fclose(fpsource); break;}

  muestra16=((int)(muestra8));
  if(muestra16>=0) muestra16=muestra16-128;
  else muestra16=muestra16+128;
  muestra16*=factor;
  fwrite(&muestra16,sizeof(short int),1,fptarget);
  if ( (coderror=ferror(fptarget)) != 0 ) /* checa errores */
{fclose(fptarget);
 vPopMsg(-1,-1,"Error al exportar archivo");
 fcloseall(); return(0); break;}

  } /* FIN DEL FOR infinito */

fcloseall();


 strcpy(selected_filec30,patht); arch=C30; return(1);

}
/*=========================================================================*/
/* LA FUNCION export_c30_to_txt SE IMPLEMENTO CON LA FINALIDAD DE PODER
EXPORTAR A ARCHIVOS CON FORMATO TIPO TEXTO, EL CUAL ES ACEPTADO, POR
EJEMPLO, POR MATLAB */
/*=========================================================================*/
int export_c30_to_txt(void)
{
FILE *fptar,*fpsour;
```

```
short int muestrax;
int coder,i;
unsigned long int cuantas;
char aux1[8],aux2[8];
char targt[20],patho[80];

 strcpy(aux1,"0"); strcpy(aux2,"1000");
 strcpy(targt,"arch1.txt");
 if (vPopGetStr(5,-1,"Adelantar cuantas muestras",aux1,7) == ESC) return(0);
 if (vPopGetStr(5,-1,"Exportar cuantas muestras",aux2,7) == ESC) return(0);
 if (vPopGetStr(5,-1,"Nombre del archivo de texto",targt,19) == ESC)
 return(0);

 getcwd(patho,80);strcat(patho,"\\TEXT\\");strcat(patho,targt);

 if ( (fpsour = fopen(selected_filec30,"rb")) == NULL)
   { vPopMsg(-1,-1,"Error Leyendo Archivo .C30");return(0); }

 if ( (fptar = fopen(patho,"w")) == NULL)
   { vPopMsg(-1,-1,"Error al abrir Archivo .TXT");return(0); }

 fread(&header,sizeof(struct el_header),1,fpsour);

 cuantas= abs(atoi(aux1));
  while(cuantas>0) {
   fread(&muestrax,sizeof(short int),1,fpsour);
if ( (coder=feof(fpsour)) != 0 ) {fclose(fpsour); break;}
if ( (coder=ferror(fpsour)) != 0 ) {fclose(fpsour); break;}
cuantas--;
   }

 cuantas=atoi(aux2);
for(i=0; i<cuantas; i++)
  {
   fread(&muestrax,sizeof(short int),1,fpsour);
if ( (coder=feof(fpsour)) != 0 ) {fclose(fpsour); break;}
if ( (coder=ferror(fpsour)) != 0 ) {fclose(fpsour); break;}

   fprintf(fptar,"%d \n",muestrax);
   }

fclose(fpsour);fclose(fptar);
vPopMsg(-1,-1,"Archivo exportado exitosamente");
}
```

C.2 Rutinas para MATLAB

Se incluyen los listados de los archivos *Script* para MATLAB que se utilizaron para simular el filtro paso banda de 100-900 Hz, así como aquellos que generan los espectrogramas de archivos de texto generados por el programa ADFO.

```
clc
echo on
%*****************************************************************************
%********* Diseño de un filtro pasobanda para deteccion de pitch *********
%************ FI UNAM - Antonio Echeverria y Lilia de la Vega   **********
%******************** Trabajo de Tesis, Enero de 1995 ********************
%************* Presione CTRL-C para detener la ejecucion *****************
%*****************************************************************************
echo off
% asigna en el vector B los coeficientes del filtro
N = input('Numero de taps del filtro: ');
echo on
B = fir1(N,[0.025 0.225], Hamming(N+1))
echo off
% El vector A es el denominador del filtro
A = [1];
% genera 256 puntos para el espectro en frecuencia
[H,W] = freqz(B,A,256);
% genera el vector f a partir del vector W
f= W*4000/pi;
% limpia la pantalla, genera los ejes
clg;
semilogy(f,abs(H));
% grafica la respuesta mag vs frec
plot(f,abs(H));
T=sprintf('Filtro FIR de 100-900 Hz, %g taps, Magnitud vs Frecuencia',N);
title(T);
pause
% ahora grafica fase vs frec
clg;
semilogy(f,angle(H));
plot(f,angle(H)*180/pi);
T=sprintf('Filtro FIR de 100-900 Hz, %g taps, Fase vs Frecuencia',N);
title(T);
pause
echo on
break
end
```

Archivo *Script* para generar los correspondientes espectrogramas de un archivo llamado **voice3.txt**.

```
clear
load voice3.txt;
b = voice3;
specgram(b,[],8000);
colormap(gray);
xlabel('Tiempo [s]');
ylabel('Frec [Hz]');
title('Espectrograma de banda angosta voice3');
pause;
[v,f,t] = specgram(b,[],8000);
surf(t,f,abs(v));
colormap(gray);
xlabel('Tiempo [s]');
ylabel('Frec [Hz]');
zlabel('Magnitud');
title('Espectrograma de superficie voice3');
```

Referencias

[1] Herbert Schildt: *C, Manual de Referencia*,México, Segunda edición, Osborne/Mc Graw-Hill, 1990.

[2] Kernighan & Ritchie: *El lenguaje de programación C*, Prentice Hall, 2ª *Edición*, *México*, 1988.