TECNOLÓGICO
DE MONTERREY

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Ciudad de México

División de Ingeniería y Arquitectura

Ingeniería Electrónica y Comunicaciones

Departamento de Ingeniería Eléctrica y Electrónica

"Compresión de voz"

Autores:

César Romero Núñez
David Mares Nava

Asesores:

M. en C. Alfredo Mantilla Caeiros
M. en C. Rogelio Bustamante Bello

Profesor:

M. en C. Flavio Lucio Pontecorvo Cassereau
Dr. José Ramón Álvarez Bada

México D.F. Mayo 2004

Prefacio

El presente trabajo se encuentra dividido en 10 capítulos, a lo largo de los cuales se exponen la teoría y la práctica de la compresión de voz.

En el primer capítulo se definen los objetivos y la justificación del proyecto, así como el calendario con el que se laboró durante los dos semestres que duraron los trabajos. Se incluye también un breve apartado en el que se da un paseo al lector a través de la historia de los codificadores de voz.

De los capítulos dos al cuatro se presentan las bases teóricas que apoyan el proyecto. En el capítulo "Estudio de la voz" se explica la naturaleza temporal y espectral de la voz, la forma en que ésta es producida y percibida, y los diferentes modelos matemáticos con los que el hombre ha intentado describirla.

En el capítulo tres "Recomendación G.711" se revisa ésta recomendación de codificación propuesta por la ITU-T (International Telecommunication Union)

El modelo de predicción lineal se introduce en el capítulo cuatro. Se explican los fundamentos teóricos y las condiciones necesarias para la aplicación de éste. Se analizan también las técnicas de obtención de los coeficientes de predicción lineal y demás parámetros que caracterizan al modelo.

Los capítulos 5, 6 y 7 cubren la teoría referente a los procesadores digitales de señales (DSP's). La adición de estos capítulos obedece a que fueron estos dispositivos en donde se realizó la mayor parte del trabajo.

En el capítulo cinco se hace una introducción general al mundo de los procesadores digitales. Se habla principalmente del integrado TMS320CV5416. El capitulo siete "El puerto serie" profundiza el estudio de ese periférico y circuitos anexos.

El capítulo seis "El codec: PCM3002" explica a detalle el funcionamiento y configuración de ese integrado. El dispositivo en cuestión se encuentra incluido en la tarjeta de evaluación utilizada y su intervención fue crucial en la realización del proyecto.

Los tres últimos capítulos presentan la metodología, los resultados y las conclusiones de cada una de las tres fases en las que fue dividido el proyecto. Cabe mencionar que ésta división siguió fines meramente administrativos.

El capítulo ocho muestra lo realizado durante el primer semestre de trabajo. En él se explica como es que se logró la obtención de la señal de voz a través del codec, la aplicación de la ley A para su codificación y la posterior recuperación analógica de la voz. Se presentan también las gráficas que comprueban los resultados obtenidos, así como las conclusiones pertinentes.

La información referente a la segunda fase del proyecto está contenida en el capítulo nueve "Comunicación entre DSP's". Se muestra la aplicación práctica de la teoría expuesta en el capítulo siete mediante los resultados y las conclusiones ahí presentadas.

Finalmente, el capítulo diez "Programación del modelo de predicción lineal" explica la metodología de programación seguida para resolver los problemas matemáticos referidos en el capítulo cuatro. Se revelan los resultados y las conclusiones obtenidas, tras el arduo trabajo computacional de éste apartado.

Se incluye un capítulo de Anexos con todos los programas que se realizaron y utilizaron en la elaboración del proyecto.

Índice

1. Introducción

1.1. Título del proyecto

Compresión de voz

1.2. Área de desarrollo

Procesamiento digital de señales

1.3. Descripción del proyecto

1.3.1. Resumen

Éste proyecto pretende estudiar y reproducir las bases en las que se cimientan las actuales tecnologías de compresión de voz utilizadas en la telefonía celular y el manejo de voz sobre redes IP, por citar algunos ejemplos. Para tal efecto, se debe estudiar a detalle el tratamiento digital de la voz, en donde existen tres partes esenciales:

Digitalización: tomar la voz analógica y transformarla en 1's y 0's para su posterior manejo

Compresión: disminuir la cantidad de 1's y 0's para ser transmitidos más eficientemente

Reconstrucción: recuperar la voz analógica utilizando la información comprimida.

1.3.2. Justificación

La voz es la principal forma de comunicación del ser humano. El incesante análisis de la misma ha arrojado diversos modelos que permiten la sintetización de la voz.

El campo de acción de este estudio cubre una amplia gama de aplicaciones entre las que resaltan: telefonía, reconocimiento automático de voz, voz sobre IP e incluso aplicaciones médicas.

1.3.3. Objetivos y metas

Desarrollar un sistema para la digitalización, compresión y posterior recuperación analógica de la voz utilizando un procesador digital de señales de Texas Instruments.

1.3.4. Alcance del proyecto

Con este trabajo se pretende establecer las bases de la compresión de voz, para su posterior uso en otros proyectos y aplicaciones que así lo requieran.

1.4. Calendarización

Agosto

- Definición del proyecto
- Introducción a DSP
- Arquitectura de DSP
- Set de instrucciones DSP

Septiembre

- Definición del proyecto
- Arquitectura de DSP
- Set de instrucciones DSP
- Pruebas de programación DSP
- Estudio de la voz
- Investigación: digitalización, compresión

Octubre

- Pruebas de programación DSP
- Estudio de la voz

- Investigación: digitalización, compresión

Noviembre

- Pruebas de programación DSP
- Investigación: digitalización, compresión
- Programación: digitalización
- *Presentación*

Diciembre

- Programación: digitalización
- Pruebas: digitalización

Enero

- Programación: digitalización, compresión
- Pruebas: digitalización, compresión

Febrero

- Programación: digitalización, compresión
- Pruebas: digitalización, compresión

Marzo

- Programación: digitalización, compresión, recuperación
- Pruebas: digitalización, compresión, recuperación

Abril

- Pruebas: digitalización, compresión, recuperación

Mayo

- Presentación final

1.5. Historia de la codificación de la voz

1.5.1. Un poco de historia

La codificación de voz consiste en convertir la señal de voz analógica en una señal digital con las menores pérdidas de calidad posibles; con el fin de optimizar el ancho de banda del canal de voz mediante la reducción del número de bits a transmitir. Permitiendo, incluso, transmitir varias comunicaciones por un solo canal o incorporar algoritmos de cifrado para establecer comunicaciones seguras.

El desarrollo de estos sistemas inicia en el año de 1605 cuando Bacon desarrolla un alfabeto de dos palabras para representar 24 letras usando 5 dígitos; a estas letras (*a* y *b*) se les llama palabras codificadas y el conjunto se conoce como código (code). Para 1641 estas ideas se extienden a los sistemas M-arios.

Hacia 1703 Leibniz describe el código binario usando solo 0's y 1's para representar enteros.

En 1791 Wolfgang Von Kempelen construye un rudimentario sintetizador de voz capaz de pronunciar frases cortas mediante un ingenio mecánico.

El telégrafo se considera el primer sistema de comunicaciones digitales. Inventado por Morse en 1837; usaba pulsos cortos y largos. Para 1875 Banfot desarrolla el sistema de telégrafo actual con 5 dígitos por palabra.

En 1924 Nyquist propone el teorema del muestreo, desarrollando así las bases matemáticas para el análisis de la señal de voz. Hacia 1928 se establece la máxima tasa de bits (bps) en un canal de ancho de banda determinado.

El año de 1937 resulta vital en la codificación de la voz, pues Reeves desarrolla uno de los sistemas más importantes: el PCM (Modulación por Pulsos Codificados). Planteando así, el problema de una codificación eficiente de la señal de voz que redujera al máximo el número de bits necesarios. Sistemas basados en PCM como DPCM (Delta PCM) y ADPCM (Adaptative DPCM) se utilizan como estándares por la CCIFF (International Consultative Committee for Telephone and Telegraph).

Ya en el año 1939 se presenta una versión electrónica del sintetizador de Kempelen. El sistema se denomina VODER, y consiste en un teclado que permite controlar la articulación y generación de sonidos vocálicos y consonánticos. Este sistema sembró las bases de los modernos sintetizadores digitales de voz, pues utilizan la misma teoría sobre el modelo de producción del habla.

En ese mismo año, Homer Dudley (laboratorios Bell) da a conocer al mundo el primer vocoder; sistema que basa su funcionamiento en analizar la voz para extraer una serie de características que son enviadas por el emisor, mientras que el receptor se encarga de reconstruir la señal de voz con base a esas características.

Los avances en la tecnología de integración de alta escala en los circuitos integrados permiten que durante las décadas de los 60's y los 70's se den nuevas soluciones al problema de codificación de voz.

A partir de los 80's, la investigación ha sido encaminada hacia conseguir codificadores que utilicen un ancho de banda cada vez menor. Entre los ejemplos más conocidos tenemos a: vocoders por predicción lineal, LPC-10, codificación RELP, codificación multiplulso (MPC), codificación CELP, codificación VSELP y codificación RPE-LTP.

1.5.2. Tipos de codificación

Los codificadores de voz se clasifican generalmente en tres grupos:

- Codificadores de forma de onda (Waveform coders).
- Vocoders.
- Codificadores híbridos.

Los codificadores de forma de onda reproducen la señal de voz original muestra a muestra, sin tomar en cuenta la naturaleza de la misma y pueden operar tanto en el dominio del tiempo, como en el dominio de la frecuencia. Este tipo de codificadores se caracteriza por proporcionar una alta calidad de voz pero con tasas de transmisión promedios de 32 k[bit/s]; lo cual los hace poco utilizados cuando se necesitan tasas bajas.

En cuanto a los vocoders, éstos extraen información acerca de la naturaleza de la voz y discrimina entre diferentes tipos de sonidos (sordo o sonoro) y su correspondiente frecuencia fundamental. Este tipo de sistemas puede llegar a disminuir de manera considerable la cantidad de información a transmitir, si se les compara con los codificadores de forma de onda. Suelen tener una calidad de voz aceptable, pero siempre se escucha demasiado sintética.

Por último, los codificadores híbridos combinan las técnicas de los dos tipos de codificaciones antes mencionados; tomando las ventajas de cada una de ellas y consiguiendo de ésta forma aproximarse al compromiso de una alta calidad de voz con tasas de transmisión bajas.

2. Estudio de la voz

2.1. Introducción

A pesar de que en la actualidad se cuenta con modernos medios de comunicación, la forma inicial de ésta, entre los hombres, es la voz. Su estudio se inició desde finales del siglo XIX y se continuó durante todo el siglo XX. Comenzando arduamente con la telefonía y extendiéndose, en la actualidad, con la transmisión de voz a través de medios compartidos de datos y video como lo es Internet.

Por esto, el estudio de la voz no ha pasado de moda y se presenta como un reto actual que requiere una amalgama de conocimientos en: procesamiento de señales, acústica, reconocimiento de patrones, teoría de la comunicación y la información, lingüística, fisiología, ciencias de la computación y psicología.

2.2. Producción y percepción de la voz

La generación de la voz se inicia cuando alguien tiene una idea que desea comunicar a otra persona. Una vez que la idea está clara (o talvez no tanto), el sujeto la codifica en un lenguaje determinado, lo cual da lugar a un sinnúmero de acciones neuromusculares, como la cantidad de aire que debe ser expelida de los pulmones y la tensión de las cuerdas vocales, por citar algunos ejemplos. El resultado final es la generación de la voz, la cual viaja a través del aire en forma de ondas de presión hasta alcanzar su destino.

Una vez que la voz llega al escucha, ésta se encarga de mover la membrana basilar que se encuentra en el oído interno. Un proceso neurotransductor (del cual aún no se sabe mucho) se encarga de convertir los movimientos de la membrana en señales que son transmitidas a través del nervio auditivo que las envía hasta el cerebro en donde finalmente se decodifica el mensaje transmitido.

La voz es producida por la secuencia de varios sonidos, los cuales han sido clasificados de acuerdo a la forma en que son creados. En general existen dos clasificaciones: sonidos sonoros y sonidos sordos.

Los sonidos sonoros son creados mediante la excitación del canal vocal por impulsos de aire casi-periódicos producidos por la vibración de las cuerdas vocales. Estos sonidos son de larga duración (comparados con los otros) y se encuentran espectralmente bien definidos.

Los sonidos sordos son producidos mediante la excitación del canal vocal por una corriente de aire constante que se vuelve turbulenta en algún punto del canal; éste punto depende del sonido que se esté creando.

2.3. Modelos de producción de la voz

El canal vocal inicia en la apertura de las cuerdas vocales o glotis, y termina en los labios. El canal vocal abarca la faringe y la boca o cavidad oral. En un hombre promedio, la longitud del canal vocal suele estar alrededor de los 17 cm., mientras que su área transversal puede llegar hasta los 20 cm^2.

El canal vocal puede modelarse como un tubo acústico excitado en uno de sus lados por la vibraciones de la glotis y por el otro por los labios que lo transforman de un tubo cerrado a uno abierto de acuerdo a la necesidad o el deseo de quien habla.

Cuando se tiene un tubo abierto por uno de sus extremos, los polos de la función de transferencia del tubo se encuentran sobre el círculo unitario distribuidos en las frecuencias dadas por:

$$f_n = (2n+1)c/4l, \qquad n = 0,1,2,... \qquad \text{ec. 2.1}$$

Para c = 344 m/s (velocidad del sonido en el aire) y l = 0.17 m (longitud del canal vocal), tenemos que f_n = 500, 1500, 2500 y 3500 Hz, lo cual implica que se cuenta con una frecuencia de resonancia por cada k[Hz].

Pero dado que el área transversal del canal vocal varia conforme se avanza de la glotis hacia los labios, una mejor aproximación al modelo consiste en utilizar pequeños tubos acústicos unidos entre sí, que cuenten con diferentes longitudes y áreas transversales entre ellos. Así, cada sección del canal vocal puede ser modelada lo más apegado posible a la realidad. Éste nuevo tubo cuenta con toda una variedad de resonancias que se espera sean lo más parecidas a las frecuencias que se presentan en el canal vocal.

Cientos de experimentos en percepción de la voz señalan la importancia de las frecuencias de resonancia (o mejor conocidos como *formants* en las disciplinas encargadas del análisis de la voz). Es por eso que se utiliza un solo modelo que sea capaz de representar un número de resonancias suficientes para representar la generación de voz.

Supóngase que cada *formant* puede ser representado por una función de transferencia de la forma:

$$H_i(z) = \frac{1}{1 - b_i z^{-1} - c_i z^{-2}} \qquad \text{ec. 2.2}$$

En donde el valor de c_i siempre debe ser menor a uno si se desea que el filtro sea estable. Si por ejemplo tomamos un ancho de banda de 5 kHz, serían necesarios cinco filtros para representar los cinco *formants* que se tendrían (uno por cada k[Hz]). Multiplicando todos los filtros necesarios, se llega a la forma final del filtro:

$$H(z) = \frac{1}{1 - \sum_{j=1}^{P} a_j z^{-j}} \qquad \text{ec. 2.3}$$

donde *P* es el número de secciones utilizadas multiplicado por dos, mientras que los coeficientes *a* son los resultantes del polinomio de orden *P*.

Este modelo de generación de la voz tiene como entrada el aire proveniente de los pulmones, mientras que la salida está constituida por la voz. Se ha encontrado que ésta función de transferencia se hallar estrechamente relacionada con la forma de la glotis, además de otros factores como la posición de la lengua y los labios. Cabe resaltar que se trata de un modelo basado en las propiedades espectrales de los sonidos sonoros.

2.4. Análisis de la señal de voz

De manera distinta a las señales determinísticas, que pueden ser caracterizadas con fórmulas analíticas, las señales de voz no. Las señales de voz se conocen como aleatorias y para su estudio se utilizan funciones estadísticas como las siguientes: la densidad espectral de potencia (DEP), la función de auto correlación (FAC), la función de distribución acumulada (FDA) y la función de densidad de probabilidad (FDP).

Los sonidos sordos tienen una potencia menor, lo que se refleja en su DEP. También se puede afirmar que el comportamiento de este tipo de señales es muy parecido al del ruido blanco.

Los sonidos vocales tienes características distintas de los anteriores, como que la señal es cuasi periódica y se repite, aproximadamente cada 80 muestras. Cada una de las muestras corresponde a 125μ[s], entonces cada uno de los picos se repiten cada 10m[s].

El intervalo se conoce como frecuencia fundamental (o *pitch*) y tiene una frecuencia de pico de 100[Hz]. Esta frecuencia puede oscilar dependiendo si se trata de un hombre o una mujer, teniendo para el hombre un rango de 40 a 120 [Hz] y para la mujer de 300-400 [Hz]. Después de cada uno de los picos hay un decaimiento de la señal que se debe a que la excitación en las cuerdas vocales se aminora.

La densidad espectral de potencia de la señal exhibe las siguientes características:

1) presenta componentes ricas en energía en frecuencias asociadas a *n*p*, siendo *n* un número entero y *p* la frecuencia de picos;

2) algunas veces podemos observar hasta 4 picos seguidos en las siguientes frecuencias: 500, 1500 y 2700 [Hz], para el ejemplo que observamos, estas son las manifestaciones de las resonancias vocales a estas frecuencias. En contraste las señales de sonidos sordos que no tienen una estructura como esta, más bien parece un filtro pasatodo, exhibiendo un pico alrededor de 2500 [Hz].

También es importante estudiar la función de autocorrelación (fig. 2.1 y 2.2) que arroja la siguiente información:

1) podemos observar un pico en 80 muestras, que indica el periodo de picos. Esta frecuencia puede ser usada para detectar y medir la periocidad de picos en muchas aplicaciones, como codificadores de voz y detectores de actividad de voz;

2) se observa que el primer pico en 20 muestras es igual de alto que el de ocho, por lo que un detector de picos debe ser capaz de cuantificar este tipo de similitudes. No se debe olvidar el hecho de que según el teorema de Wiener-Khitshin, la FAC es la transformada de Fourier de la DEP.

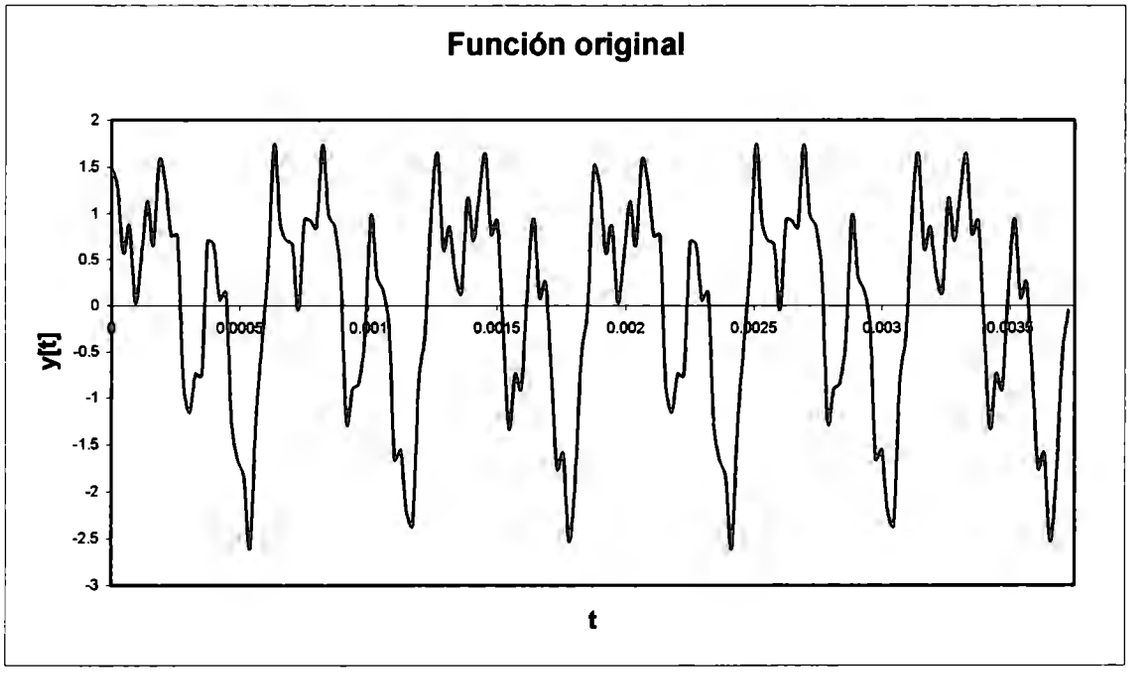


figura 2.1 Señal sonora

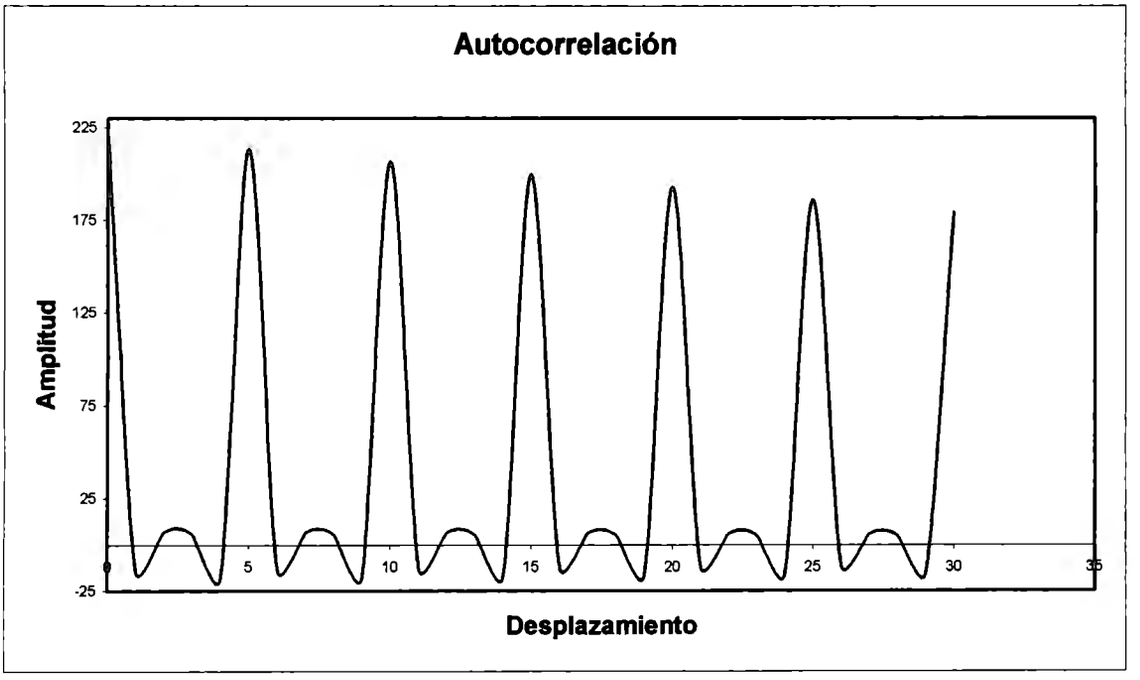


figura 2.2 Autocorrelación de una señal sonora

En contraste, la señal de sonidos sordos tiene un decaimiento mucho más rápido, que indica ninguna correlación entre muestras adyacentes, y ninguna periocidad

3. Recomendación G.711

A partir de los estudios estadísticos realizados acerca del espectro de la voz humana se llegó al consenso de que la mayor concentración de energía de la voz se encuentra entre los 300 y los 3600 [Hz]. Esto significa que un filtro que funcione entre estas frecuencias nos permitiría obtener la información relevante, utilizando así sólo el ancho de banda necesario para el posterior manejo de la voz.

Las frecuencias femeninas podrían quedar afuera de este rango dada su naturaleza. Una mujer tiene un espectro de voz que ubica sus niveles energéticos fuera de los 4 k[Hz]; es por esto que, en el sistema telefónico actual una mujer que este gritando o expresándose de manera agitada seguramente no será bien escuchada.

En la práctica, se utilizan filtros con un rango de acción entre los 0 [Hz] y los 4 k[Hz] debido a la respuesta no ideal de los filtros analógicos. En la actualidad, y gracias al uso de técnicas basadas en el procesamiento digital de señales, los filtros se aproximan cada vez más a las condiciones ideales; pero la tradición suele imperar y el límite de los 4 k[Hz] parece haber llegado para quedarse.

El teorema de Nyquist indica que para poder recuperar una señal a partir de sus muestras es necesario que la frecuencia de muestreo sea de cuando menos dos veces el ancho de banda de la señal analógica. La G.711 recomienda que la frecuencia de muestreo para la voz sea de 8000 [Hz], es decir que se deben de tomar 8000 muestras de voz en el intervalo de un segundo, se permite una tolerancia de ±50 partes de millón (ppm). Para lograrlo es necesario tomar muestras de la voz cada 125 microsegundos.

Los procesos de cuantificación y codificación de las muestras vienen siempre de la mano. La cuantificación se refiere a escoger el digito más cercano para que este represente el valor de la muestra. La codificación se refiere al código en el que se representaran cada uno de los dígitos elegidos. La G.711 permite dos opciones para llevar a cabo estas tareas: la ley A y la ley μ. Ambas leyes de codificación son no lineales, es decir que no existe un espacio uniforme entre cada uno de los dígitos que las conforman.

La G.711 indica que para los circuitos internacionales se deben utilizar ocho dígitos binarios por muestra. Lo cual esta previsto por ambas leyes, ya que cuentan con 256 (2^8) valores de salida.

La ley μ es el estándar en la mayor parte de América, mientras que Europa optó por la ley A. Debido a nuestra cercanía geográfica y la aún mayor cercanía económica con los Estados Unidos, se analizará la ley μ (ecuación 3.1).

La ley μ está dada por

$$e_0 = \frac{\log(1+\mu e_1)}{\log(1+\mu)} \qquad \text{ec. 3.1}$$

donde e_1, e_0 están normalizadas a (0,1) y μ es una constante. En los sistemas telefónicos de los EUA se utiliza μ=255.

En cuanto a la transmisión en serie del código a través de un medio físico, el bit 1 (bit de polaridad) se envía en primer lugar y el bit 8 (el bit menos significativo) en último lugar.

- En una salida de frecuencias vocales cualquiera del multiplexor PCM debe haber una señal sinusoidal de 1kHZ con un nivel nominal de 0dBm0 al aplicarse a la entrada del decodificador la secuencia periódica de señales.

- El nivel de sobrecarga teórica resultante ($T_{máx}$) es de +3.17dBm0 para la ley μ.

Por razones históricas, México utiliza el estándar europeo, por lo que a continuación se presenta un breve estudio sobre la ley A (ecuación 3.2).

La ley A está dada por:

$$e_0 = \begin{cases} \dfrac{Ae_1}{1+\log(A)} & 0 \le e_1 < 1/A \\ \dfrac{1+\log(Ae_1)}{1+\log(A)} & 1/A \le e_1 < 1 \end{cases} \qquad \text{ec. 3.2}$$

En cuanto a la transmisión en serie del código a través de un medio físico, el bit 1 (bit de polaridad) se envía en primer lugar y el bit 8 (el bit menos significativo) en último lugar.

- En una salida de frecuencias vocales cualquiera del multiplexor PCM debe haber una señal sinusoidal de 1kHZ con un nivel nominal de 0dBm0 al aplicarse a la entrada del decodificador la secuencia periódica de señales.

- El nivel de sobrecarga teórica resultante ($T_{máx}$) es de +3.14dBm0 para la ley A.

La ley A se encarga de codificar los niveles de voz con base en su probabilidad de aparición en una señal aleatoria. Como se mencionó anteriormente, no es lineal, y reserva una gran cantidad de pasos de cuantización para muestras de voz con amplitud baja y viceversa. Con esto se logró tener una norma de codificación ajustada a las características de las señales de voz y que es bastante eficiente. Se muestra a continuación la norma de codificación de la ley A, para distintos valores de la señal.

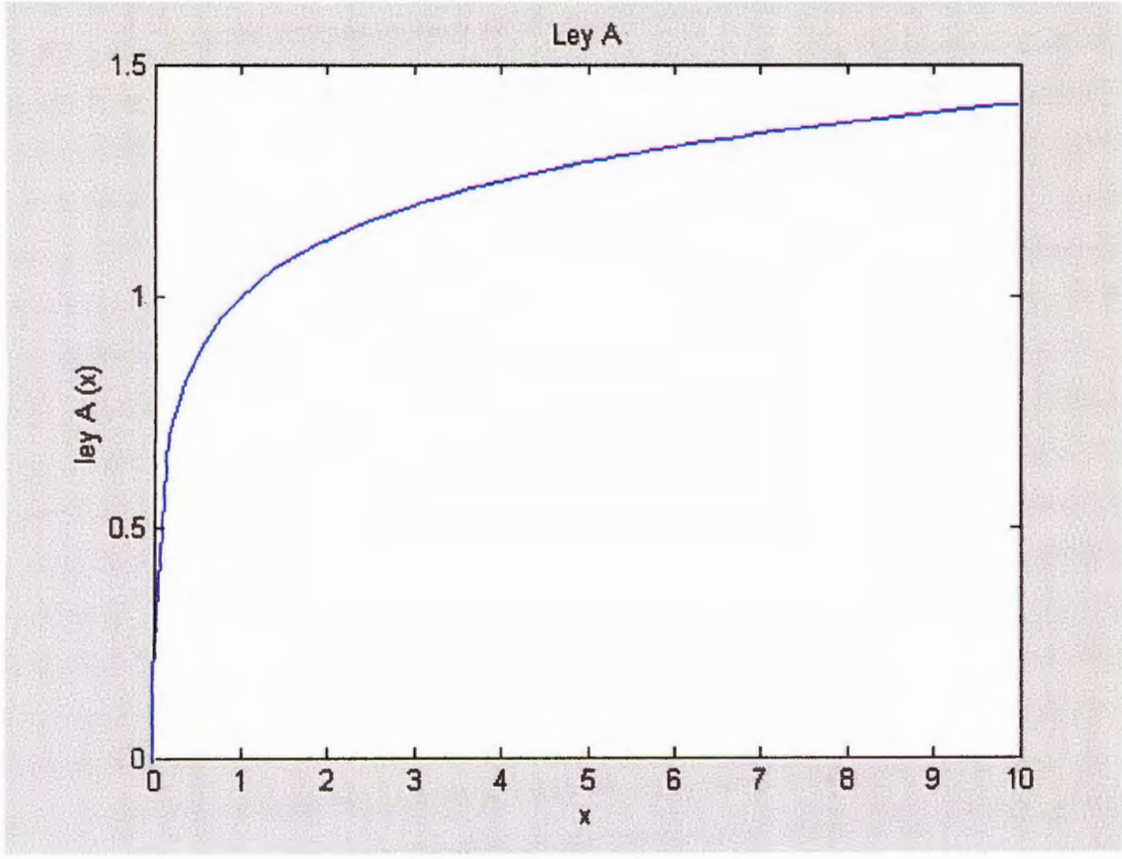


figura. 3.1 Gráfica de la ley A

Cabe señalar que ambas leyes de codificación fueron creadas para maximizar la relación señal a ruido de las señales que representan.

4. El modelo de predicción lineal para reconocimiento de voz

4.1. Introducción

Se puede pensar en predecir la siguiente muestra de una señal a partir de sus muestras anteriores. Es decir, conociendo las muestras actuales estimar cuál es la que viene. Las características de la voz y las técnicas de procesamiento digital permiten encontrar una aproximación de la siguiente muestra de voz contando solamente con un número determinado de muestras anteriores. Este es el fundamento de la predicción lineal.

El fundamento teórico tiene que ver con la redundancia de la voz y su carácter cuasiperiódico y cuasiestacionario. Es por esto que al analizar una muestra dada se encuentran similitudes entre distintos segmentos de voz permitiendo así que ésta se pueda predecir.

Esta técnica de cálculo de coeficientes de predicción lineal se usa en las tecnologías celulares actuales. Con los coeficientes es posible comprimir la voz y liberar los canales de transmisión de gran cantidad de carga informativa. Se ha logrado una disminución de tasas de transmisión de hasta los 4 k[bps].

4.2. Formulación del problema

Debido a la redundancia de la voz en pequeños espacios temporales (10-30 m[s]), cualquier muestra presente puede ser predicha como una combinación lineal de *p* muestras anteriores, como se muestra:

$$\tilde{s}(n) = \sum_{k=1}^{p} a_k s(n-k) \qquad \text{ec. 4.1}$$

donde *p* es el orden del predictor, a_k representa los coeficientes de predicción lineal del filtro y $\tilde{s}(n)$ las muestras estimadas.

El error de predicción está representado por

$$e(n) = s(n) - \tilde{s}(n) \qquad \text{ec. 4.2}$$

$$e(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \qquad \text{ec. 4.3}$$

$$e(n) = \sum_{k=0}^{p} a_k s(n-k) \qquad \text{ec. 4.4}$$

Calculando la transformada z de la última ecuación tenemos que:

$$E(z) = S(z)A(z) \qquad \text{ec. 4.5}$$

que muestra claramente la separación lineal del modelo. Se observa también que

$$A(z) = 1 - \sum_{k=1}^{p} \acute{a}_k z^{-k} = \sum_{k=0}^{p} \acute{a}_k z^{-k} \qquad \text{ec. 4.6}$$

$$a_0 = 1$$

lo cual indica que el polinomio sólo tiene ceros mas no polos. Se le conoce comúnmente como un filtro de puros ceros. Expresando la señal *S(z)* en términos de *E(z)* y *A(z)* se tiene:

$$S(z) = \frac{E(z)}{A(z)} = E(z)H(z) \qquad \text{ec. 4.7}$$

Esta forma sugiere que $H(z) = 1/A(z)$ puede modelar adecuadamente la señal de entrada *S(z)*. Siempre y cuando se encuentren los coeficientes a_k (LPC *Linear Prediction Coefficient*) que minimicen el error cuadrático de la función, el cual está definido como:

$$E\left[e^2(n)\right] = E\left\{\left[s(n) - \sum_{k=1}^{p} a_k s(n-k)\right]^2\right\} \qquad \text{ec. 4.8}$$

para encontrar los LPC que minimizan la función se deriva respecto a los LPC y se iguala a cero

$$\frac{\partial E\left[e^2(n)\right]}{\partial a_k} = -2 \cdot E\left\{\left[s(n) - \sum_{k=1}^{p} a_k s(n-k)\right] s(n-i)\right\} = 0 \qquad \text{ec. 4.9}$$

simplificando un poco se llega a

$$E\{s(n)s(n-i)\} = \sum_{k=1}^{p} a_k E\{s(n-k)s(n-i)\}, \qquad i = 1,\ldots,p \qquad \text{ec. 4.10}$$

en donde se puede observar que

$$C(i,k) = E\{s(n-i)s(n-k)\} \qquad \text{ec. 4.11}$$

son los coeficientes de covarianza de la señal de entrada. Lo cual permite redefinir el conjunto de *p* ecuaciones como:

$$\sum_{k=1}^{p} a_k C(i,k) = C(i,0), \qquad i = 1,...,p \qquad \text{ec. 4.12}$$

4.3. Cálculo de los coeficientes de predicción lineal.

Existe una gran variedad de formas para calcular los coeficientes de predicción lineal. Las más usadas son: el método de la autocorrelación y el método de la covarianza. Por cuestiones práctica, en éste trabajo se utilizará el método de la autocorrelación.

Se trabaja con un intervalo finito de muestras, $0 \le n \le L_a - 1$. Así, los coeficientes de covarianza $C(i,k)$ son calculados mediante la siguiente expresión:

$$C(i,k) = \sum_{n=0}^{La+p-1} s(n-i)s(n-k), \qquad \begin{matrix} i = 1,.....,p \\ k = 0,.....,p \end{matrix} \qquad \text{ec. 4.13}$$

Haciendo el cambio de variable $m = n - i$, la ecuación puede ser expresada como:

$$C(i,k) = \sum_{m=0}^{La-1-(i-k)} s(m)s(m+i-k) \qquad \text{ec. 4.14}$$

que nos dice que $C(i,k)$ es la autocorrelación de un intervalo corto de tiempo de la señal $s(m)$ evaluada en un desplazamiento de $(i-k)$, dando:

$$C(i,k) = R(i-k)$$

donde

$$R(j) = \sum_{n=0}^{La-1-j} s(n)s(n+j) = \sum_{n=j}^{La-1} s(n)s(n+j) \qquad \text{ec. 4.16}$$

representa los coeficientes de autocorrelación de la señal de voz. De aquí las *p* ecuaciones pueden ser representadas de la siguiente forma:

$$\sum_{k=1}^{p} a_k R(|i-k|) = R(i),, \qquad i = 1,.....,p \qquad \text{ec. 4.17}$$

Esta misma ecuación en su forma matricial se escribe de la siguiente forma:

$$\begin{pmatrix} R(0) & R(1) & R(2) & R(p-1) \\ R(1) & R(0) & R(1) & R(p-2) \\ R(2) & R(1) & R(0) & R(p-3) \\ \cdots & & \cdots & \cdots \\ R(p-1) & R(p-2) & R(p-3) & R(0) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{pmatrix} = \begin{pmatrix} R(1) \\ R(2) \\ R(3) \\ \cdots \\ R(p) \end{pmatrix} \qquad \text{ec. 4.18}$$

Esta matriz de coeficientes de autocorrelación de dimensión pxp tiene características estructurales Toeplitz en donde todos los elementos alrededor de cierta diagonal son idénticos. Existen decenas de algoritmos recursivos capaces de resolver eficientemente éste tipo de sistemas de ecuaciones. Entre los más eficientes de acuerdo a su rapidez y fácil programación se encuentra el método de Levinson-Durbin. A continuación se muestra el método de resolución de sistemas de ecuaciones de Levinson-Durbin.

$$E(0) = R(0) \qquad \text{ec. 4.19}$$

para $1 \le i \le p$ hacer

$$k_i = \frac{\left[R(i) - \sum_{j=1}^{i-1} a_j^{(i-1)} R(i-j) \right]}{E(i-1)} \qquad \text{ec. 4.20}$$

$$a_i^{(i)} = k_i \qquad \text{ec. 4.21}$$

para $1 \le j \le i-1$ hacer

$$a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)} \qquad \text{ec. 4.22}$$

$$E(i) = (1 - k_i^2) E(i-1) \qquad \text{ec. 4.23}$$

La solución final después de *p* iteraciones está dada por:

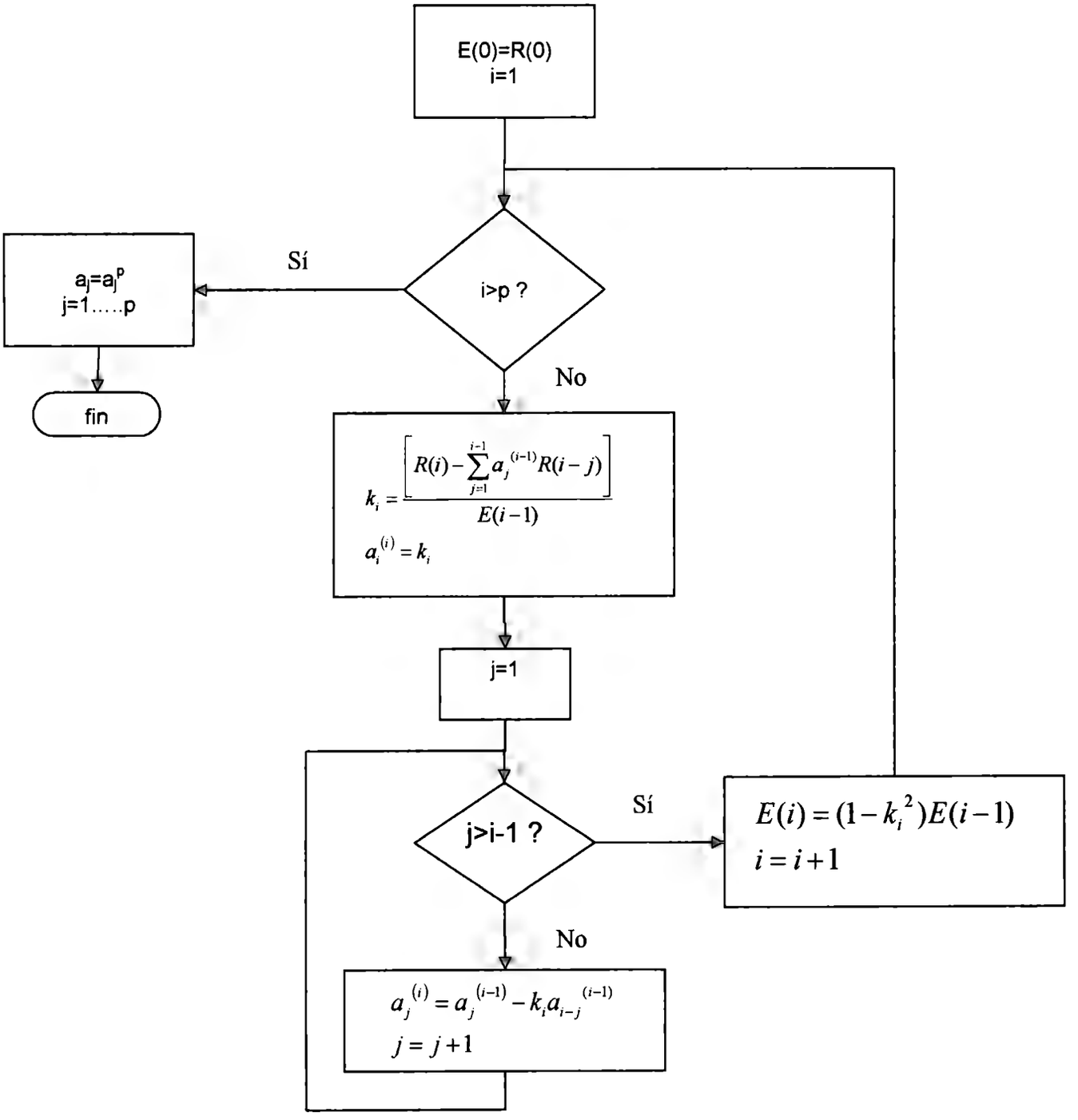
$$a_j = a_j^{(p)}, \qquad j = 1, \ldots\ldots, p \qquad \text{ec. 4.24}$$

Este método hace una aproximación para encontrar la ganancia de la señal.

$$G^2 = R(0) - \sum_{i=1}^{N} a_i R(i) = E_n \qquad \text{ec. 4.25}$$

en donde E_n es la última iteración del error cuadrático dada por el algoritmo Levinson-Durbin. Este dato es la información referente a la ganancia que se necesita.

El diagrama de flujo para el algoritmo es el siguiente:



4.4. Señal de excitación para el filtro de predicción lineal

La figura 4.1 muestra el diagrama de bloques que ejemplifica la forma en la que la voz es generada a través del filtro de predicción lineal.

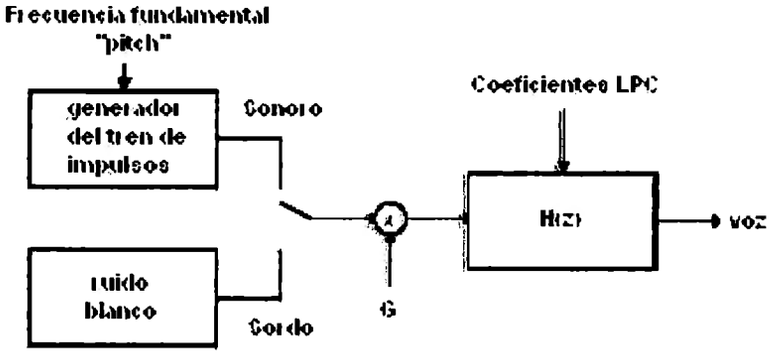


figura 4.1 Modelo de predicción lineal de la voz

Como puede observarse, existen dos fuentes con las que es factible alimentar al filtro $H(z)$: un tren de impulsos o ruido blanco. La decisión se basa en conocer si la trama que se desea reconstruir es sonora o sorda. Si la trama de la que se calcularon los coeficientes LPC es sonora se elige el tren de impulsos, si ésta fue sorda, se toma el ruido blanco como fuente de excitación.

Si la trama resulta ser sonora, el generador del tren de impulsos necesita saber a que frecuencia debe trabajar. Para esto se tiene que calcular la frecuencia fundamental (*pitch*) de la trama original. Para tal efecto, existe al menos una decena de técnicas para determinar el *pitch* (y cada día se trabaja en nuevos caminos). Entre las más conocidas están la de Gold-Rabiner, la homomórfica y la de la autocorrelación. Se explica a detalle ésta última por ser el método elegido para ser implementado en éste proyecto.

La figura 2.2 muestra la función de autocorrelación de un segmento de voz sonoro, se observa que se presentan unos picos bien definidos; mientras que en un segmento de voz sordo, no se alcanza a apreciar ningún tipo de patrón. Por lo que se intuye que la función de autocorrelación también puede servir para calcular si la trama en cuestión es sorda o sonora.

De hecho, y en base a estudios estadísticos de la voz, el criterio de decisión entre un segmento de voz sordo y uno sonoro se basa en tomar el pico más significativo de aproximadamente las últimas dos terceras partes de la función de autocorrelación y compararlo con $0.3R(0)$. Si el pico es mayor, se trata de una trama sonora; si es menor, se toma como una trama sorda.

Como se observa en la figura 2.2, la función de autocorrelación contiene la información necesaria para determinar el *pitch*; ya que si una función es periódica, su autocorrelación también lo es; pero ésta función también muestra información extra que puede llegar a ser estorbosa.

Es por esto que para enfocar el análisis sólo en la detección de la frecuencia fundamental, se suele aplicar un recorte centrado a la señal de voz. Es decir, que se eliminan las partes de la señal original que se encuentren entre cierto rango de valores, y las que no se lleguen a eliminar iniciaran desde un nivel de cero.

Para llevar a cabo el recorte se necesita un umbral de decisión, el cual se define mediante el siguiente procedimiento:

- Encontrar el pico máximo de la señal en el primer tercio de la trama A_1, y el pico máximo en el último tercio de la trama A_2.
- Determinar el umbral en el nivel $C_L = K \min[A_1, A_2]$, $K = 0.6 - 0.8$. ec. 4.26

Una vez calculado el umbral se utiliza la ecuación 4.27 que define la función de recorte centrado para la señal de voz.

$$y(n) = \begin{cases} 1 & s(n) \geq C_L \\ -1 & s(n) \leq -C_L \\ 0 & otros \end{cases} \quad \text{ec. 4.27}$$

Con esto, la señal original de voz se transforma en una serie de valores más sencillos, con los que se calcula una función de autocorrelación (mediante la ecuación 4.28) que sólo presenta la información necesaria para determinar de manera rápida y eficiente la frecuencia fundamental de la trama. Pues ya sólo se tienen los picos de la función de autocorrelación de la señal original y basta con determinar la separación entre estos para encontrar el *pitch*. Así, ya es factible generar el tren de impulsos necesario para excitar al filtro de predicción lineal caracterizado por los coeficientes LPC y la ganancia.

$$R(k) = \sum_{n=0}^{N-1} s(n)s(n+k), \qquad k = 0,1,2,...,N \quad \text{ec. 4.28}$$

4.5. Reconstrucción de la voz

Una vez calculados los coeficientes de predicción lineal y la ganancia, se puede conformar el filtro $H(z)$ (ec. 4.29) en el dominio de la frecuencia o la ecuación de diferencias LPC (ec. 4.30), cuya salida representa la señal de voz recuperada en el dominio del tiempo. Además, ya identificados el tipo de trama y su frecuencia fundamental, se puede hacer uso de las ecuaciones 4.29 y 4.30 que definen la reconstrucción de la señal de voz $S(z)$ o $s(n)$ mediante una señal de excitación $X(z)$ o $x(n)$. Como ya se explicó, la señal de excitación puede ser un tren de pulsos o ruido blanco.

$$H(z) = \frac{G}{1 - \sum_{j=1}^{p} a_j z^{-1}} = \frac{S(z)}{X(z)} \quad \text{ec. 4.29}$$

$$s(n) = Gx(n) + \sum_{j=1}^{p} a_j s(n-j) \quad \text{ec. 4.30}$$

Lo importante es observar que en el dominio del tiempo, la salida $s(n)$ es el resultado de una combinación lineal de las salidas anteriores ponderadas mediante los coeficientes LPC más la señal de excitación actual multiplicada por la ganancia del sistema.

4.6. Algunas consideraciones

Antes de aplicar el modelo de predicción lineal a la voz, es necesario realizar ciertos procesos que llevan al modelo a un mejor nivel de eficiencia. Estos procesos son: el filtro de pre-énfasis y el ventaneo.

Con base a estudios espectrales de la voz, se sabe que cada vez que la frecuencia aumenta en un factor de dos, la amplitud de la señal de voz disminuye por un factor de dieciséis. Por lo que se suele aplicar un filtro de pre-énfasis a la señal de voz con el fin de obtener amplitudes similares a lo largo de todo el espectro. Así, el modelo de predicción lineal obtiene los parámetros necesarios para elaborar de mejor forma el análisis frecuencial de la voz.

Para implementar el pre-énfasis en un sistema digital, se utiliza un filtro pasa altas de primer orden con una frecuencia de corte que oscile entre los 100 [Hz] y los 1000 [Hz]. La ecuación 4.31 muestra el filtro en el dominio de la frecuencia, mientras que la ecuación 4.32 es la ecuación de diferencias que realiza la misma función de pre-énfasis. El valor típico de la constante a suele estar entre 0.9 y 1. En la práctica, ni el valor de la frecuencia de corte, ni el valor de la constante involucrada son críticos.

$$H_{pre}(z) = 1 - az^{-1} \qquad \text{ec. 4.31}$$

$$y[n] = x[n] - ax[n-1] \qquad \text{ec. 4.32}$$

En cuanto al ventaneo, éste elimina todo aquello que se encuentra fuera del área de interés, definiendo así la porción de voz que será procesada. Es importante señalar que en la práctica no existe la ventana ideal, sólo aproximaciones. La figura 4.2 muestra algunas de las técnicas más conocidas de ventaneo: rectangular (ec. 4.33), Hamming (ec 4.34), Hanning (ec. 4.35), Blackman y Kaiser (triángular).

$$w_{rect}(n) = \begin{cases} 1 & ;0 \le n \le N-1 \\ 0 & ;otros \end{cases} \qquad \text{ec. 4.33}$$

$$w_{Hamming}(n) = \begin{cases} 0.54 - 0.46\cos\left(2\pi \dfrac{n}{N-1}\right) & ;0 \le n \le N-1 \\ 0 & ;otros \end{cases} \qquad \text{ec. 4.34}$$

$$w_{Hanning}(n) = \begin{cases} 0.5 - 0.5\cos\left(2\pi \dfrac{n}{N-1}\right) & ;0 \le n \le N-1 \\ 0 & ;otros \end{cases} \qquad \text{ec. 4.35}$$

La ventana rectangular no suele ser utilizada en el análisis de la voz, ya que favorece la intromisión de las altas frecuencias, ocasionando que éste tipo de ventaneo incremente la cantidad de ruido a lo largo del proceso.

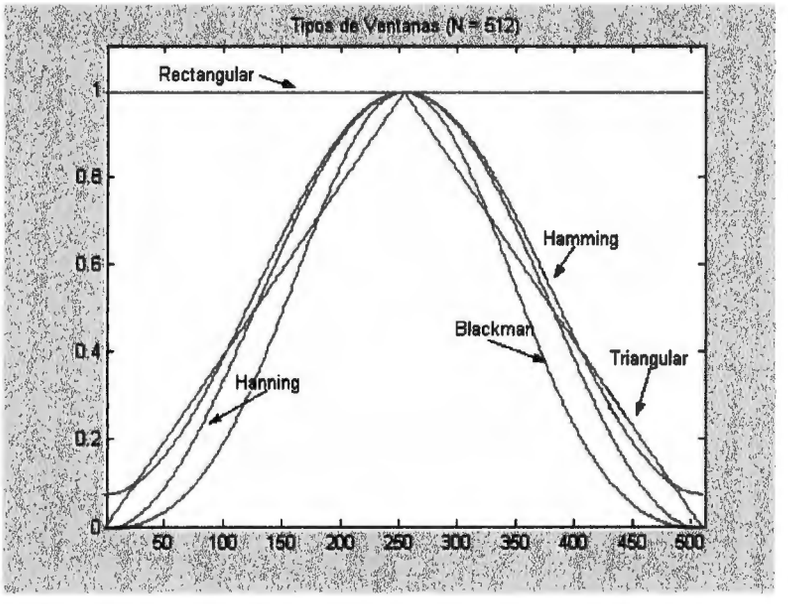


figura 4.2 Tipos de ventanas

5. Los Procesadores Digitales de Señales (DSP)

5.1. Introducción

Un DSP es un microprocesador de propósito específico diseñado para el procesamiento digital de señales. Con respecto a otro tipo de microprocesadores, el DSP cuenta con características que le facilitan llevar a cabo las operaciones propias del procesamiento digital.

Entre estas características se encuentran: el set de instrucciones, los periféricos y su velocidad.

El procesamiento digital de señales es una herramienta que facilita el tratado de las señales analógicas, como las del mundo real. Entre las operaciones que podemos realizarle a una señal digitalizada se encuentran las siguientes: filtrado, amplificación, compresión, atenuación, almacenado, análisis frecuencial, etc.

El proceso señal analógica de entrada – señal analógica de salida, con el cual la mayoría de dispositivos electrónicos digitales opera es el siguiente. Las variables físicas del mundo real se transforman en señales eléctricas mediante el uso de un transductor. Después, la señal es tratada para su digitalización en un convertidor analógico-digital. Con esto, tenemos lista la señal digital para trabajar con ella en el DSP. Una vez que realizamos las operaciones necesarias y obtenemos lo que queremos de ella, la señal sale del DSP directo a un convertidor digital-análogo que realiza la operación contraria. Se trata la señal una vez más para su salida al mundo real pasándose a través de un actuador que tiene la función de convertir la señal eléctrica en una variable física. El procesamiento se repite cuántas veces sea necesario.

5.2. Set de Instrucciones

En cuanto al conjunto de instrucciones podemos decir que entre ellas encontramos las operaciones aritméticas y lógicas necesarias para llevar a cabo ecuaciones en diferencias con gran rapidez y eficiencia por citar un ejemplo. Las ecuaciones en diferencias son la base

del procesamiento digital y es por ello que al poder resolver estas en forma eficiente, el trabajo con el DSP se verá optimizado.

Estos son los siguientes tipos de instrucciones:

- Aritméticas
- Lógicas
- Flujo de programa
- Lectura y escritura
- Repetición de instrucciones

Dentro de las instrucciones que se encuentran en el rubro de operaciones aritméticas encontramos las operaciones de suma/multiplicación dentro de una misma instrucción. Lo cual facilita el procesamiento digital de señales.

He aquí un ejemplo:

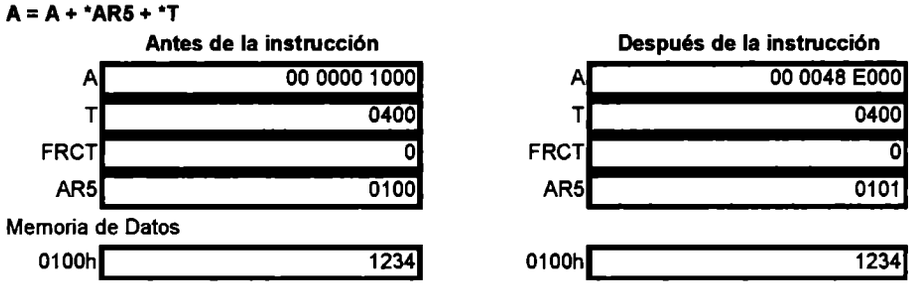


figura 5.1 Ejemplo de una instrucción en lenguaje ensamblador

5.3. Periféricos y Velocidad

Pasando a los periféricos es importante mencionar que de acuerdo con las aplicaciones del DSP, se cuenta con los periféricos necesarios para interactuar con el mundo real. Así, el TMS320C5416 cuenta con los periféricos y los convertidores (Analógico/Digital y Digital/Analógico) necesarios para aplicaciones que caen en el rango de las comunicaciones; mientras que los DSP's de la familia C2xx cuentan con las interfaces y periféricos necesarios para aplicaciones enfocadas al control.

El asunto de la velocidad podría parecer engañoso, ya que existen microprocesadores de propósito general de mayor velocidad que los DSP's. Sin embargo, los DSP's están especializados en ciertos aspectos puntuales (sumas y multiplicaciones), y es ahí en donde muestran su superioridad: mayor velocidad de procesamiento de un determinado conjunto de instrucciones necesarias para llevar a buen fin su función específica; el procesamiento digital en este caso.

Es por estas razones técnicas que se decidió utilizar un DSP dentro del proyecto. Ya que gracias a su uso se logrará un mejor manejo de la señal de voz.

5.4. Arquitectura Interna

En cuanto a la arquitectura interna del DSP, sus características son las siguientes: buses de datos de 16 bits, dos acumuladores de 40 bits cada uno, lo cual implica la necesidad de una ALU de 40 bits. Pero sin duda lo más interesante es el sumador/multiplicador que se encuentra cableado por separado de la ALU; con lo cual se comprueba que este microprocesador está optimizado para la resolución de operaciones aritméticas de suma y multiplicación y todo lo que este basado en ellas como las ecuaciones en diferencias y por ende el procesamiento digital de señales.

Hablando sobre la capacidad en memoria se pueden mencionar los siguientes datos técnicos:

- 16 kW en memoria ROM interna
- 128 kW en memoria RAM interna
- 8192 kW en memoria extendida (ROM y/o RAM)

El punto referente a la memoria extendida es el que necesita un mayor análisis. Por memoria extendida se entiende a toda la memoria de programa disponible, sin importar que esta sea ROM, RAM o cualquier otro tipo.

En un DSP se habla de memoria de programa, de memoria de datos y de memoria de entrada/salida. Esta clasificación de memorias está basada en el uso que se le da al espacio en memoria y no al tipo de memoria en si. Así, la memoria de programa se refiere al espacio utilizado para almacenar las instrucciones que debe seguir el DSP. La memoria de datos se encarga de las variables que el DSP necesite para la ejecución del programa. Mientras que la memoria de entrada/salida se refiere a los dispositivos de entrada/salida que se encuentran mapeados dentro del espacio de memoria.

Como se ha visto hasta ahora, el DSP es una poderosa herramienta en el procesamiento digital de señales. Y es por ello que resultó ser la opción a elegir para éste proyecto de ingeniería.

6. El Codec: PCM3002 del 5416 DSP Starter Kit

6.1. Introducción

El 5416 DSP Starter Kit (DSK de aquí en adelante) es una de las tantas tarjetas de evaluación que permiten verificar ciertas propiedades de los DSP C54X de Texas Instruments (TI). Entre las características más relevantes con que cuenta el DSK tenemos:

- DSP TMS320VC5416 a 160MHz
- Codec Estéreo PCM3002
- Cuatro DIP switches y cuatro leds accesibles por el usuario
- Chips de Flash y SRAM

Además, se cuenta con el Code Componer Studio (CCS), una herramienta creada por TI para poder interactuar con el DSK a través de los lenguajes C y ensamblador. El CCS se comunica con el DSK usando un emulador JTAG.

Éste capítulo se centra en el PCM3002 ya que es una de las secciones del DSK que más se explotará debido a la naturaleza del proyecto de ingeniería.

6.2. El codec PCM3002

Ya que una de las más típicas aplicaciones de los DSP es el procesamiento de señales de audio (como la voz en éste proyecto), el DSK incluye el codec PCM3002. Por codec se entiende un codificador/decodificador; por lo que el PCM3002 se encarga de codificar las entradas analógicas en un formato digital, que son el tipo de señales con las que un DSP puede trabajar; y después se encargará de decodificar los datos provenientes de DSP para generar una señal analógica. En términos coloquiales, podemos decir que el PCM3002 es la puerta entre el mundo real y el digital.

El codec necesita de dos canales seriales para comunicarse, uno para controlar los registros internos del codec y otro para el traslado de los datos. Los registros CODEC_H y CODEC_L se usan para generar el flujo de datos apropiados en el canal de control mientras que el McBSP2 (Multichannel Buffered Serial Port 2) se utiliza como canal de datos. En general, se pueden utilizar el McBSP0 o el McBSP1 para el canal de datos, pero el DSK utiliza los registros del CPLD (Complex Programmable Logic Device, figura 6.1) para tal propósito; dejando libres los McBSP 0 y 1. En cualquier caso, y en la mayoría de las ocasiones, el canal de control se utiliza sólo al principio del programa para configurar el codec.

6.3. Configuración del Codec

La configuración por default para el canal de datos está dada para una frecuencia de muestreo de 48 k[Hz] con datos de 16bits. El reloj del canal de datos esta referenciado a una señal de 12.288 M[Hz] provenientes del DSK. Esta referencia se divide entre 4 en el CPLD creando un reloj de 3.072 M[Hz] que utiliza el McBSP2 (que es canal de datos, para el que no lo recuerde). Como el codec siempre transmite tramas estéreo de 64bits, la frecuencia de muestreo se puede calcular al dividir 3.072 M[Hz] entre 64 para así obtener los 48 k[Hz]. El reloj puede ser dividido más allá usando el registro CPLD llamado CODEC_CLK; y así lograr frecuencias de muestreo de 8kHz como la que se necesita en el proyecto para implementar la recomendación G.711.

El codec puede manipular muestras de 16 o 20 bits de acuerdo al formato seleccionado en el campo FMT del registro 3 del codec. Los datos muestreados siempre se empaquetan en tramas de 64bits con los datos del canal izquierdo y derecho separados en 32 y 32 bits dentro de la trama. A pesar de que se transmiten 32 bits, el codec sólo alcanza a reconocer 16 o 20 bits de acuerdo al formato elegido. Las fases derecha - izquierda de los datos se sincronizan con ayuda de la entrada LCRIN del codec. El LCRIN está controlado por la trama de sincronización del McBSP2.

Para lograr un buen funcionamiento del codec, siempre será bueno realizar las siguientes recomendaciones:

- el McBSP2 esté configurado como el canal de datos
- el bit BSP2SEL del registro MISC del CPLD esté en cero
- la frecuencia de muestreo deseada esté configurada en el registro CODEC_CLK del CPLD
- Los registros del codec estén listos, escribiendo primero el CODEC_L, y después en el CODEC_H para iniciar la transferencia de datos. Verificar CODEC_RDY en el registro MISC para saber cuando haya terminado la transferencia.

Registros del CPLD

Dir. I/O	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	USER_REG	USR_SW3	USR_SW2	USR_SW1	USR_SW0	USR_LED3	USR_LED2	USR_LED1	USR_LED0
1	DC_REG	DC_DET	DC_IO_CTL	DC_STAT1	DC_STAT0	DC_RST	0	DC_CNTL1	DC_CNTL0
2	CODEC_L	CODEC_L_CMD(7:0)							
3	CODEC_H	CODEC_H_CMD(15:8)							
4	VERSION	CPLD_VER(3:0)				0	BOARD VERSION(2:0)		
5	DM_CNTL	DM_SEL	MEMTYPE_DS	MEMTYPE_PS	DM_PG4	DM_PG3	DM_PG2	DM_PG1	DM_PG0
6	MISC	CODEC_RDY	0	0	0	0	DC_WIDE	DC32_ODD	BSP2SEL
7	CODEC_CLK	0	0	0	0	DIV_SEL	CLK_STOP	CLK_DIV1	CLK_DIV0

figura 6.1 Registros del CPLD

El PCM3002 cuenta con 4 registros que sirven para controlar funciones como el volumen, formato de datos, filtrado y el modo loopback. Estos registros se configuran mediante los registros CODEC del CPLD, utilizando el canal de control del PCM3002. A continuación se muestran estos registros, así como una pequeña descripción de los mismos.

Registros del Codec PCM3002

	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
Registro 0	res	res	res	res	res	A1	A0	LDL	AL7	AL6	AL5	AL4	AL3	AL2	AL1	AL0
Registro 1	res	res	res	res	res	A1	A0	LDR	AR7	AR6	AR5	AR4	AR3	AR2	AR1	AR0
Registro 2	res	res	res	res	res	A1	A0	PDAD	BYPS	PDDA	ATC	IZD	OUT	DEM1	DEM0	MJT
Registro 3	res	res	res	res	res	A1	A0	res	res	res	LCP	res	FMT1	FMT0	LRP	res

figura 6.2 Registros del codec

Nombre del Registro	Nombre del Bit	Descripción
Registro 0	A (1:0)	Dirección del registro "00"
	res	Reservado
	LDL	Control de atenuación del canal izquierdo del DAC
	AL (7:0)	Atenuación del canal izquierdo
Registro 1	A (1:0)	Dirección del registro "01"
	res	Reservado
	LDR	Control de atenuación del canal derecho del DAC
	AR (7:0)	Atenuación del canal derecho
Registro 2	A (1:0)	Dirección del registro "10"
	res	Reservado
	PDAD	Control de apagado del ADC
	PDDA	Control de apagado del DAC
	BY+B4PS	Control del filtro pasa altas del ADC
	ATC	Control de atenuación del DAC
	IZD	Control de detección de cero del DAC
	OJT	Habilitación de salida del DAC
	DEM (1:0)	Control de De-énfasis del DAC
	MJT	Control de Mute
Registro 3	A (1:0)	Dirección del registro "11"
	res	Reservado
	LOP	Control del loop-back
	FMT (1:0)	Selección del formato de datos
	LRP	Selección de la polaridad del reloj

figura 6.3 Descripción de los registros del codec

6.4. Interfaz del codec

La interfaz de audio está formada por cuatro jacks estéreo estándares de 3.4mm (fig. 6.4); los cuales están dispuestos de la siguiente forma:

- Un jack para conectar un micrófono
- Un jack para conectar una señal de audio estéreo de entrada
- Un jack para conectar una señal de audio estéreo de salida (line-out)
- Un jack para conectar una bocina

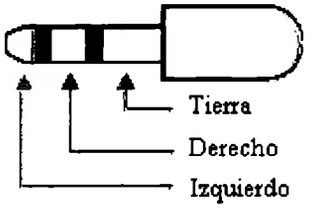


figura 6.4 Conector de audio estándar

La entrada de audio está acoplada en AC e incluye:

- Un mezclador para el micrófono monoaural con la línea de entrada estéreo
- Un potenciómetro para ajustar la ganancia del micrófono
- Un voltaje de polarización que soporta el uso de micrófonos que funcionen con baterías o que necesiten corriente. El máximo nivel proveniente del micrófono es de 1 Vrms.
- Filtrado pasivo entre los jacks de audio del DSK y el codec para incrementar la eficiencia

La salida de audio estéreo está disponible en dos formas; sin amplificar (line-out) y amplificada (bocina). Otras de sus características son:

- Un filtro de segundo orden en la salida estéreo (line-out)
- Un amplificador en la salida de la bocina para conectar bocinas de baja impedancia (la típica de 8 [ohms])
- Una salida directa (line-out) capaz de manejar cargas de alta impedancia
- La ganancia en las salidas de audio es programable vía software

7. El puerto serie: McBSP del TMS320VC5416

7.1. Introducción

El TMS320VC5416 cuenta con tres puertos seriales (mejor conocidos como McBSPs): de alta velocidad, transmisión full duplex, multicanalizados y con múltiples registros de almacenamiento de transmisión y recepción. A través de estos puertos, el DSP es capaz de comunicarse con: otros DSPs, codecs, y demás sistemas que requieran una conexión vía serie. Como puede recordarse, en el capítulo 6 se estudió la relación que existe entre el codec PCM3002 y el McBSP2. Debido a la naturaleza de este proyecto fue necesario realizar un estudio más profundo sobre el manejo de los McBSPs.

A continuación se enlistan algunas de las características más importantes de estos periféricos:

- Comunicación Full Duplex
- Transmisión con doble registro de almacenamiento; y recepción con triple registro de almacenamiento.
- Señales de reloj y de sincronía de trama de recepción y transmisión independientes.
- Multicanalización de hasta 128 canales.
- Manejo de palabras de 8, 12, 16, 20, 24 y 32 bits.
- Codificación y decodificación de las leyes A y μ.
- Reloj y generador de trama internos programables.

El McBSP cuanta con un canal de datos y un canal de control, los cuales son manejados a través de siete pines, como puede verse en la figura 7.1 los pines DR y DX forman el canal de datos, mientras que el resto se encarga del canal de control. En la tabla 7.1 se especifica la función de cada uno de los pines involucrados.

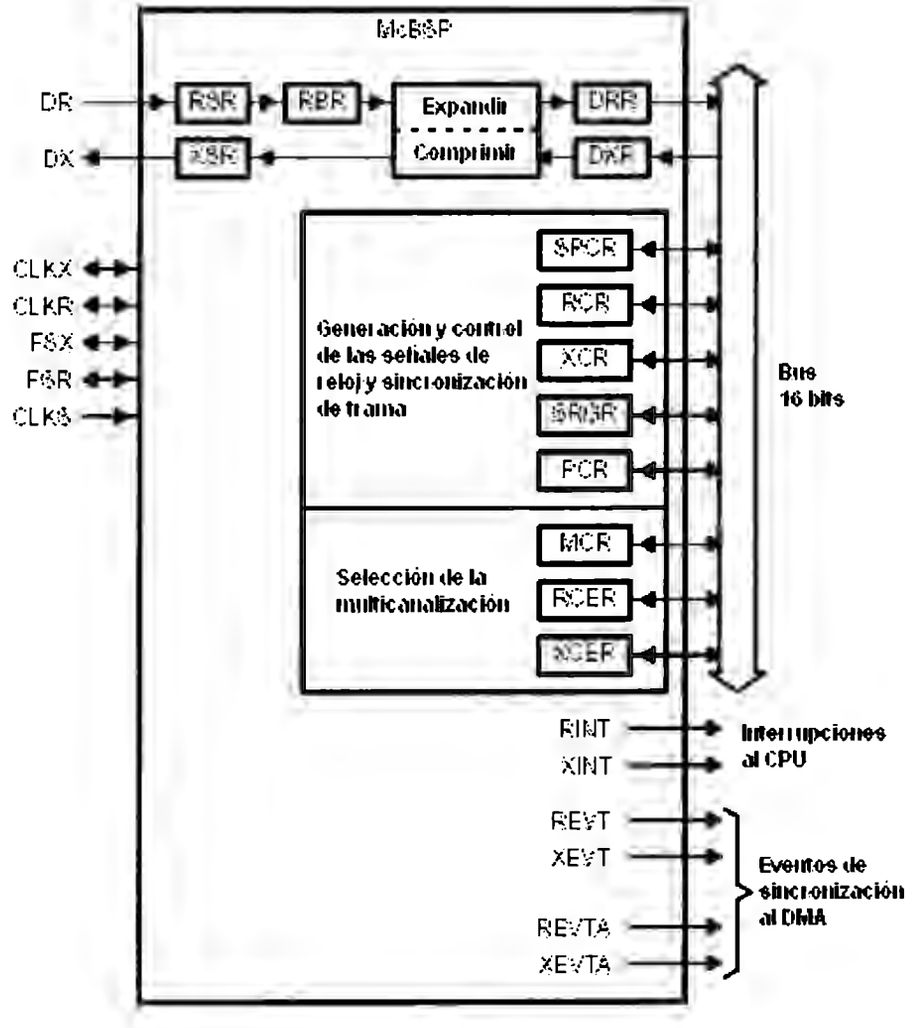


figura 7.1 Diagrama de bloques del puerto

Señales de la interfaz del McBSP

Pin	E/S/Z	Descripción
CLKR	E/S/Z	reloj de recepción
CLKX	E/S/Z	reloj de transmisión
CLKS	E	reloj externo
DR	E	dato de recepción
DX	S/Z	dato de transmisión
FSR	E/S/Z	sincronía de trama de recepción
FSX	E/S/Z	sincronía de trama de transmisión

tabla 7.1 Señales de la interfaz del puerto serie

7.2. Configuración del McBSP

El McBSP se configura a través de dos registros de control de 16 bits llamados (SPCR[1,2]) y del registro de control de pines (PCR). Ambos contienen información del estado actual del McBSP y pueden ser configurados de acuerdo a la ocasión.

A continuación se muestran las tablas 7.2, 7.3 y 7.4 en las que se desglosa la función de cada uno de los campos de estos registros.

SPCR1

Bit	Nombre	E/S	Descripción
15	DLB	E	Modo digital loopback
14-13	RJUST	E	Modo de extensión de signo y justificación para recepción
12-11	CLKSTP	E	Modo de paro de reloj
10-8	Reservado	-	Reservado
7	DXENA	E	Habilitador del DX
6	ABIS	E	Modo abis
5-4	RINTM	E	Modo de interrupcion para la recepción
3	RSYNCERR	E	Error de sincronización en la recepción
2	RFULL	S	Registro de almacenamiento en la recepción lleno
1	RRDY	S	Listo para recepción
0	RRST	E	Para resetear la recepción

tabla 7.2 Registro de configuración SPCR1

SPCR2

Bit	Nombre	E/S	Descripción
15-10	Reservado	-	Reservado
9	FREE	E	Modo de trabajo libre
8	SOFT	E	Modo soft
7	FRST	E	Reset del FSG
6	GRST	E	Reset del SRG
5-4	XINTM	E	Modo de interrupción para la transmisión
3	XSYNCERR	E	Error de sincronización en la transmisión
2	XEMPTY	S	Registro de almacenamiento en la transmisión vacío
1	XRDY	S	Listo para transmitir
0	XRST	E	Para resetear la transmisión

tabla 7.3 Registro de configuración SPCR2

PCR

Bit	Nombre	E/S	Descripción
15-14	Reservado	-	Reservado
13	XIOEN	E	Modo de E-S propósito general en transmisión
12	RIOEN	E	Modo de E-S propósito general en recepción
11	FSXM	E	Modo de sincronización de trama para la transmisión
10	FSRM	E	Modo de sincronización de trama para la recepción
9	CLKXM	E	Modo de reloj para la transmisión
8	CLKRM	E	Modo de reloj para la recepción
7	Reservado	-	Reservado
6	CLKS_STAT	S	Estado del CLKS
5	DX_STAT	S	Estado del DX
4	DR_STAT	S	Estado del DR
3	FSXP	E	Polaridad de la sincronización de trama para la transmisión
2	FSRP	E	Polaridad de la sincronización de trama para la recepción
1	CLKXP	E	Polaridad de reloj para la transmisión
0	CLKRP	E	Polaridad de reloj para la recepción

tabla 7.4 Registro de configuración PCR

Existen otros dos registros que se encargan de configurar la forma en la que se llevará a cabo la recepción y la transmisión. Estos registros son el RCR[1,2] y el XCR[1,2] para recepción y transmisión respectivamente.

A continuación se muestran las tablas 7.5, 7.6, 7.7 y 7.8 en donde se desglosan los campos que comprenden estos registros.

RCR1

Bit	Nombre	E/S	Descripción
15	Reservado	-	Reservado
14-8	RFRLEN1	E	Longitud de la trama de recepción 1
7-5	RWDLEN1	E	Longitud de la palabra de recepción 1
4-0	Reservado	-	Reservado

tabla 7.5 Registro de configuración RCR1

RCR2

Bit	Nombre	E/S	Descripción
15	RPHASE	E	Fases de recepción
14-8	RFRLEN2	E	Longitud de la trama de recepción 2
7-5	RWDLEN2	E	Longitud de la palabra de recepción 2
4-3	RCOMPAND	E	Modo de decompresión en la recepción
2	RFIJ	E	Ignorar trama en la recepción
1-0	RDATDLY	E	Retardo de datos en la recepción

tabla 7.6 Registro de configuración RCR2

XCR1			
Bit	**Nombre**	**E/S**	**Descripción**
15	Reservado	-	Reservado
14-8	XFRLEN1	E	Longitud de la trama de transmisión 1
7-5	XWDLEN1	E	Longitud de la palabra de transmisión 1
4-0	Reservado	-	Reservado

tabla 7.7 Registro de configuración XCR1

XCR2			
Bit	**Nombre**	**E/S**	**Descripción**
15	XPHASE	E	Fases de transmisión
14-8	XFRLEN2	E	Longitud de la trama de transmisión 2
7-5	XWDLEN2	E	Longitud de la palabra de transmisión 2
4-3	XCOMPAND	E	Modo de compresión en la transmisión
2	XFIJ	E	Ignorar trama en la transmisión
1-0	XDATDLY	E	Retardo de datos en la transmisión

tabla 7.8 Registro de configuración XCR2

7.3. Reloj y sincronización de trama

Las señales de reloj presentes en los pines CLKR y CLKX delimitan los bits de recepción y transmisión respectivamente. De manera análoga las señales presentes en FSR y en FSX indican el inicio de una nueva palabra. La figura 7.2 ejemplifica este hecho.

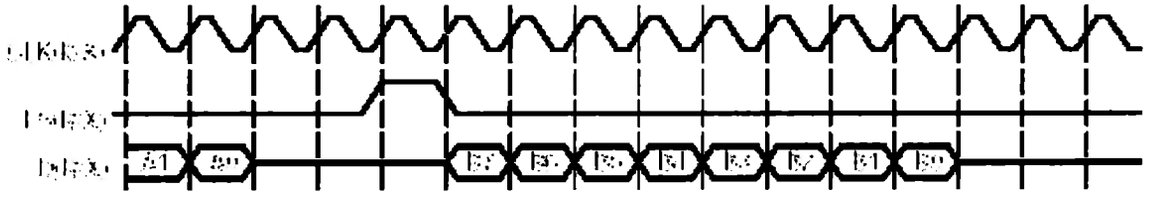


figura 7.2 Señales de reloj y sincronización en la transmisión de datos

Se pueden configurar diferentes parámetros que modifican la sincronización de las tramas de datos. Esta configuración se realiza tanto para la transmisión como para la recepción de forma independiente. A continuación se enuncian algunos de estos parámetros:

- Polaridad de las señales FSR, FSX, CLKX y CLKR
- Tramas de fase sencilla o doble
- Para cada fase, el número de palabras por trama y el número de bits por palabra
- Justificación de los datos recibidos

- Extensión de Signo o llenado con ceros

La señal de sincronización de la trama puede provenir de una fuente externa o del SRG (Sample Rate Generador, figura 7.3) para elegir entre estas dos opciones se utiliza el bit FS(R/X)M ubicado en el registro PCR. De igual manera las señales de reloj de transmisión y recepción pueden elegir su procedencia de acuerdo al bit CLK(R/X)M también presente en el registro PCR.

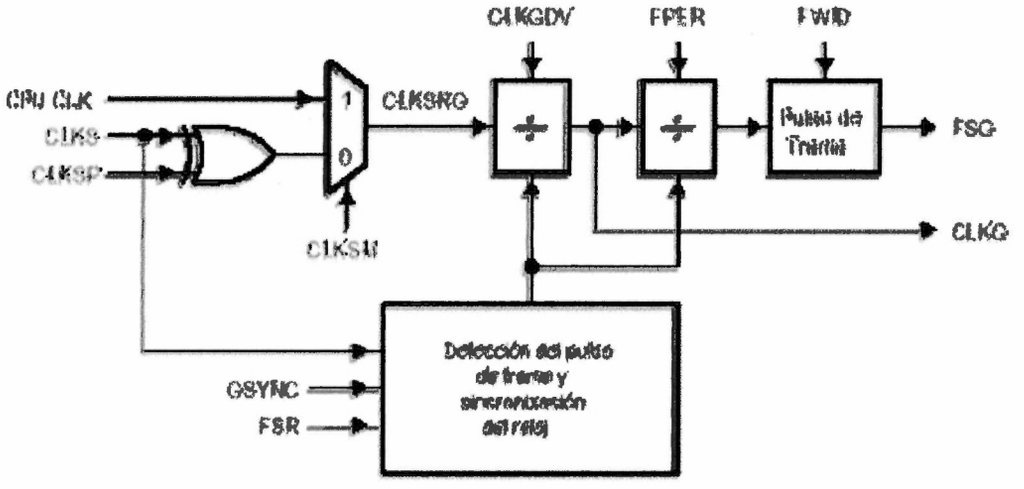


figura 7.3 Diagrama de bloques del SGR

El SRG cuenta con dos registros que controlan los parámetros con los que éste ha de funcionar. Los registros son los SRGR[1,2] y las tablas 7.9 y 7.10 desglosan los campos contenidos en estos.

SRGR1

Bit	Nombre	E/S	Descripción
15-8	FWID	E	Ancho de pulso de sincronía de trama
7-0	CLKGDV	E	Divisor de reloj del SRG

tabla 7.9 Registro de configuración SRGR1

SRGR2

Bit	Nombre	E/S	Descripción
15	GYSNC	E	Sincronización del reloj del SRG
14	CLKSP	E	Selecciona la polaridad del reloj CLKS
13	CLKSM	E	Modo de reloj del SRG
12	FSGM	E	Modo de sincronización de la trama del SRG para la transmisión
11-0	FPER	E	Período de la trama

tabla 7.10 Registro de configuración SRGR2

8. Primera fase: Codificación y Decodificación de la voz usando el 5416DSK.

8.1. Metodología

Siempre es preferible tomar un gran problema y dividirlo en varios pequeños para después ir resolviendo cada uno de ellos. Es por eso que en ésta primera etapa se trabajará sobre la digitalización de la voz y la aplicación de la ley A, cumpliendo así la recomendación G.711. Ya que el proceso inverso es muy similar, también se trabajará sobre la ley A inversa y la recuperación de la señal analógica. Los siguientes párrafos son un resumen de la primera parte del proyecto.

Una vez adquiridos los conocimientos necesarios, se puede iniciar con la implementación en el DSP. De las investigaciones referentes a la voz y el estándar G.711 se obtuvieron los parámetros y procedimientos a seguir a lo largo de ésta primera fase del proyecto. Mientras que el estudio del DSP (arquitectura, instrucciones, etc.) brinda las herramientas para llevar a cabo la implementación física del sistema. La figura 8.1 muestra el sistema propuesto para ésta fase del proyecto.

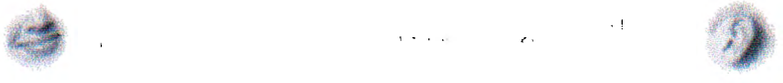


figura 8.1 Esquema de funcionamiento

Resumiendo su funcionamiento: el sistema digitalizará la voz por medio de un ADC, para que después el DSP se encargue de aplicar la ley A. Se comprobará tal hecho y el mismo DSP realizará el proceso inverso (aplicando la ley A en su forma inversa) para que al final un DAC reconstruya una señal analógica que simule a la voz presente al inicio del sistema.

El primer problema es digitalizar la señal, lo cual se logra a través de un ADC que sea capaz de muestrear la señal a 8000 [Hz] según la recomendación G.711. Dentro del TMS320C5416

DSK se cuenta con el codec PCM3002 el cual incluye a su vez un ADC con frecuencia de muestreo programable por software.

Para llevar a cabo éste tipo de acciones se hace uso de la "Board Supply Library (BSL)", que es una librería creada en lenguaje C para interactuar con dispositivos externos al DSP que se encuentran en la tarjeta de evaluación C5416 DSK. Entre estos dispositivos se puede mencionar al codec, los leds y los dip switches.

Ya que se utilizará el codec para la digitalización de la voz, es necesario configurarlo correctamente antes de poder utilizarlo. El codec cuenta con 4 registros, dos de los cuales se encargan de la atenuación en las salidas del DAC (también incluido). El resto de los registros se encarga de diversas funciones como modos de ahorro de energía entre otros. Las líneas de código necesarias para la adecuada configuración del codec son las siguientes:

```
DSK5416_PCM3002_Config setup =

{

   0x1FF,     // Set-Up Reg 0 - Left channel DAC attenuation

   0x1FF,     // Set-Up Reg 1 - Right channel DAC attenuation

   0x0,       // Set-Up Reg 2 - Various ctl e.g. power-down modes

   0x0        // Set-Up Reg 3 - Codec data format control

};

hCodec = DSK5416_PCM3002_openCodec(0, &setup);

DSK5416_PCM3002_CodecHandle hCodec;
```

El primer conjunto de líneas forman las palabras de control con que se configuraran los cuatro registros con que cuenta el codec. En este caso se utilizó una atenuación de 0db en ambos canales. Mientras que las opciones de ahorro de energía y loopback se encuentran inhabilitadas.

La segunda línea se encarga de la necesaria inicialización de una variable del tipo CodecHandle que lleva como nombre hCodec, y la cual se encargará de dar servicio a todas las instrucciones que interactúen con el codec. En esta variable se guarda el estatus del PCM3002 (activo o inactivo). El tipo de la variable es entero y se define en un archivo tipo cabecera llamado dsk5416_pcm3002.h.

La última línea se encarga de activar el codec. Es decir, de configurarlo de acuerdo a las palabras de control definidas por el programador y ponerlo en funcionamiento. Es importante señalar que desde este momento inicia la comunicación entre el PCM3002 y el DSP (vía McBSP2). Ahora corresponde al usuario elegir cuando recibir o transmitir los datos a través del codec.

Ya que hCodec ha sido inicializada, se procede a configurar la frecuencia de muestreo del codec de acuerdo al estándar G.711. La frecuencia por default es de 48 k[Hz], y para modificarla se usó la siguiente línea de código:

```
DSK5416_PCM3002_setFreq(hCodec, 8000);
```

Una vez configurado el codec según las especificaciones necesarias. Lo que resta es adquirir los datos a través del McBSP2 para después hacer con ellos lo que se desee. Para llevar a cabo tal lectura se cuenta con la siguiente instrucción, también disponible en la BSL:

```
DSK5416_PCM3002_read16(hCodec, &channel)
```

Esta instrucción se encarga de leer un dato de 16 bits y lo guarda en la variable que se le indique (channel en este caso). Y es así como se obtiene con una muestra de voz digitalizada en una localidad de memoria dentro del DSP.

Ya que el codec PCM3002 trabaja en estéreo, cada muestra tiene una longitud de 32 bits. Existe una instrucción muy parecida a la anterior con la que se puede obtener el dato en 32 bits. Pero como el proyecto no exige un sistema en estéreo, se puede eliminar uno de lo canales. Para llevar a cabo esto se tienen dos opciones: leer en 32 bits y eliminar la parte que no se necesite o leer dos datos de 16 bits y eliminar una de las lecturas. La última opción se presenta a continuación:

```
while(!DSK5416_PCM3002_read16(hCodec, &channel));

while(!DSK5416_PCM3002_read16(hCodec, &trash_channel));
```

Ahora existen dos instrucciones de lectura que almacenan las muestras en dos variables diferentes. Una de ellas se utilizará para el procesamiento de la señal de voz, la otra se dejara en la memoria sin usar.

Las instrucciones de lectura se ciclan para que el DSP espere a que el codec tenga el dato listo. En caso de que codec esté en proceso de la obtención del dato, el programa esperará hasta entonces. En el prototipo de la función se encuentra definido el retorno de un uno o un cero dependiendo si recibió el dato o se encuentra en espera de él.

Para verificar el correcto funcionamiento del codec, se realiza el proceso inverso: escribir desde una localidad de memoria al McBSP2 para que éste lo envíe al codec y sea reconstruida la señal a través del DAC.

Para tal efecto, existe una instrucción que mueve un dato desde una localidad de memoria hacia el codec, que se ocupará del resto del proceso.

```
while(!DSK5416_PCM3002_write16(hCodec, channel));

while(!DSK5416_PCM3002_write16(hCodec, channel));
```

Se utilizan dos instrucciones de escritura de 16 bits para cada uno de los canales, formando así el dato de 32 bits que el codec necesita para reconstruir la señal de la voz.

Es de ésta forma como se logra digitalizar una señal de voz y reconstruirla posteriormente. Como pudo observarse, las instrucciones proporcionadas por la BSL facilitan en gran medida la interacción con el PCM3002.

El siguiente problema a resolver es la aplicación de la ley A en las muestras que se obtengan por medio del codec.

Los puertos serie del DSP TMS320C5416 cuentan con un módulo que puede llevar a cabo tal ley. La figura 8.2 presenta el flujo de expansión-comprensión del puerto serie.

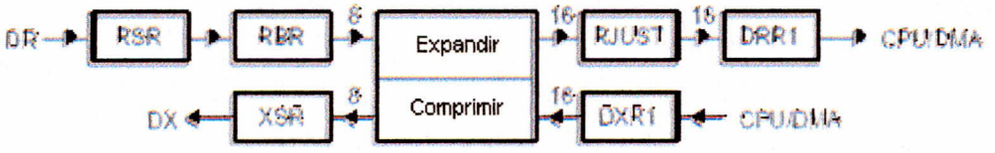


figura 8.2 Diagrama del flujo de datos en el puerto serie

El problema con este módulo es que sólo puede aplicar la ley A en los datos que se envían, y sólo se aplican la ley A inversa en los datos que llegan. Así es que no se pueden transformar de manera directa los datos provenientes del codec. El caso inverso tampoco es factible.

Para resolver tal cuestión se suele utilizar la siguiente técnica (figura 8.3):

- Activar el modo DLB (Digital Loopback) en el puerto serie
- Si se desea aplicar la ley A activar XCOMPAND
- Si se desea aplicar la ley A inversa activar RCOMPAND

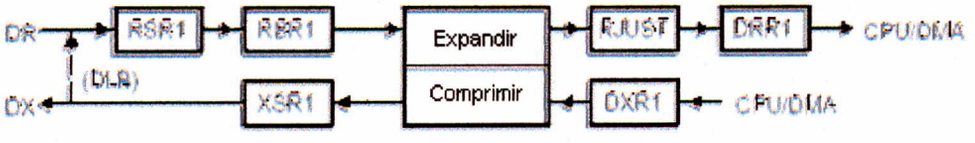


figura 8.3 Diagrama del flujo de datos (DLB) en el puerto

Estos cambios de configuración se llevan a cabo en los registros de control del puerto serie. Mientras que el dato ya modificado se toma del registro de recepción (DRR). Las funciones que realizan estas modificaciones dentro del programa (ver Anexo) son:

A_law_McBSP0_config ()

inverse_ A_law_McBSP1_config ()

Mientras que las leyes A y A inversa se efectúan con las líneas:

A_law (channel);

Inverse_A_law (channel_A);

Ya que el manejo de los puertos serie se facilita a través del lenguaje ensamblador, ésta parte del programa está escrita en ése lenguaje. El programa principal se encuentra en lenguaje C con llamadas a las funciones creadas en ensamblador.

8.2. Resultados

Para comprobar la calidad de la señal recuperada después de haber sido tratada con la ley A y su correspondiente ley inversa, se introdujo una señal senoidal con una amplitud apropiada para no dañar al codec, y con frecuencias que iban variando para medir la respuesta del sistema.

Se sabe que la voz oscila entre 500 y 4000 [Hz] y es en este rango donde debe trabajar el codec. Esta prueba se hizo para evitar juicios subjetivos en cuanto a la calidad de la señal recuperada, comparando la señal original con la señal recuperada.

A continuación se muestran las imágenes proporcionadas por el osciloscopio a 500, 1500, 2500, 3500, y 4500 [Hz]. Además de tres gráficas en la que se muestran el cambio proporcional de la amplitud de la señal recuperada con respecto a la señal original. Una última gráfica muestra los espectros de ambas señales. La señal original (arriba, azul oscuro) fue capturada en el canal 1 y proviene de un generador de funciones. La señal recuperada (abajo, azul claro) corresponde a la leída en el canal 2 del osciloscopio.

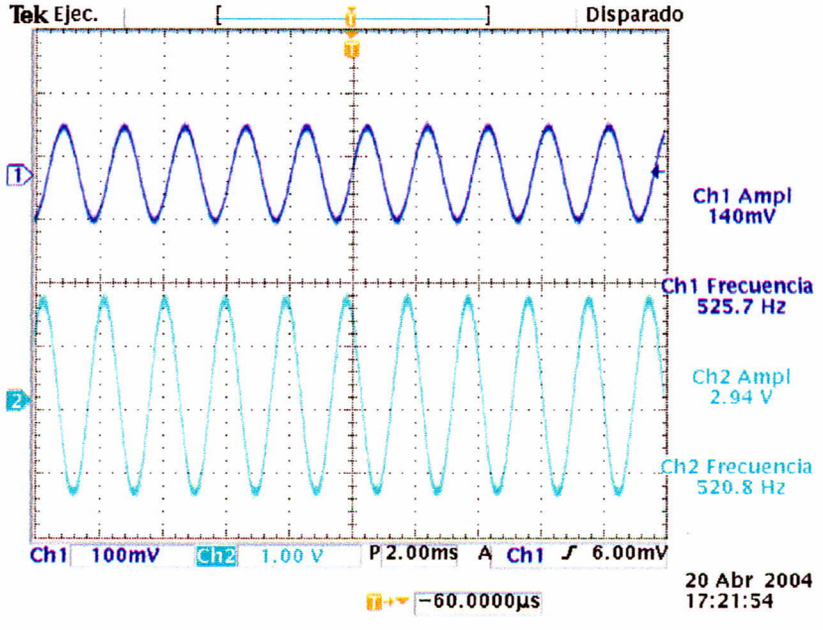


figura 8.4 Señales original y recuperada a 500[Hz]

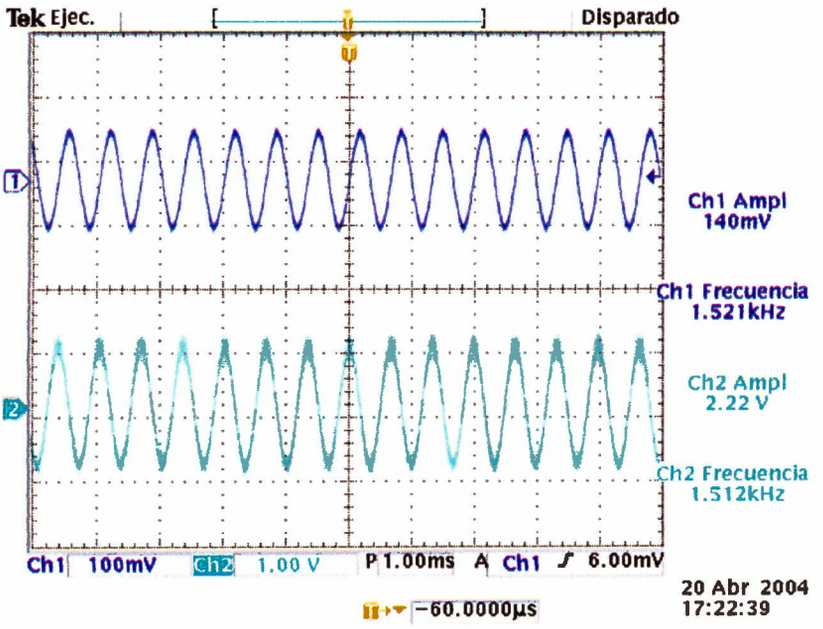


figura 8.5 Señales original y recuperada a 1500[Hz]

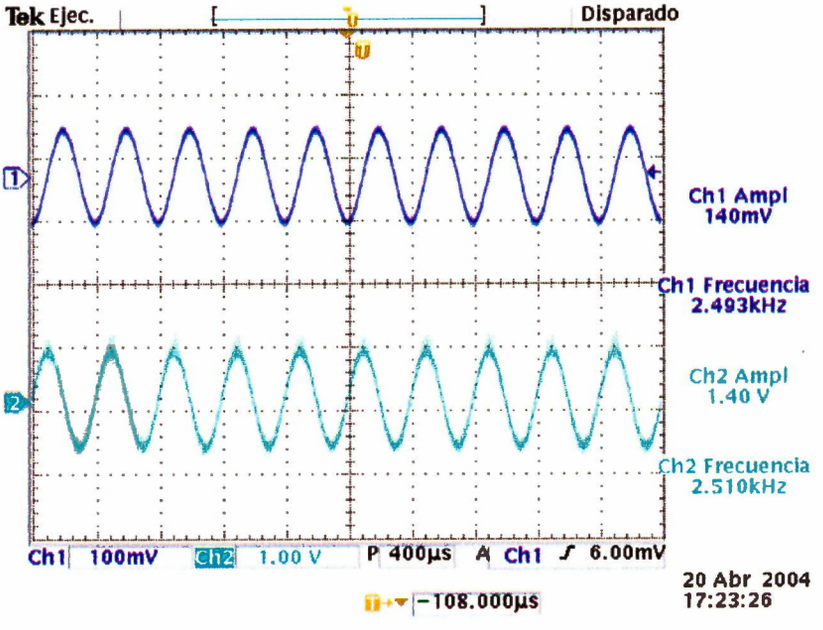


figura 8.6 Señales original y recuperada a 2500[Hz]

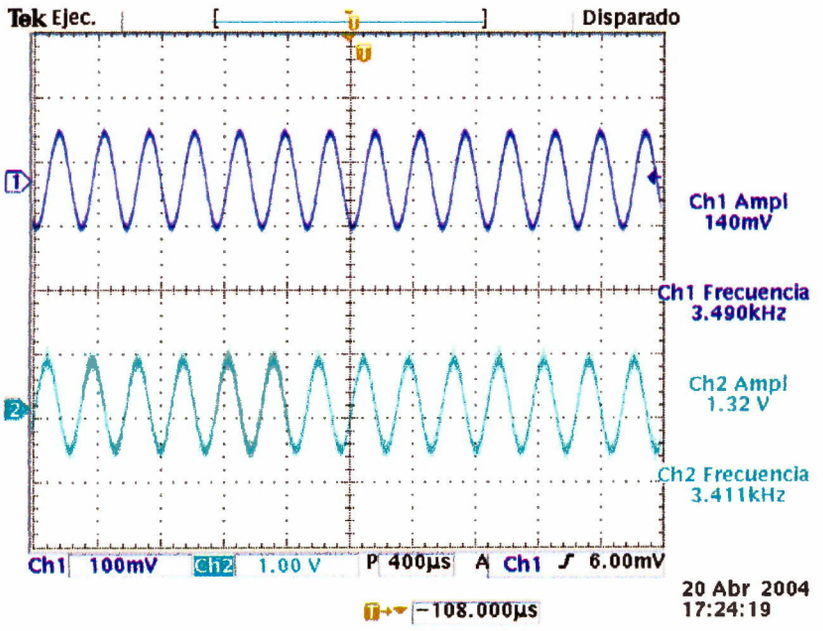


figura 8.7 Señales original y recuperada a 3500[Hz]

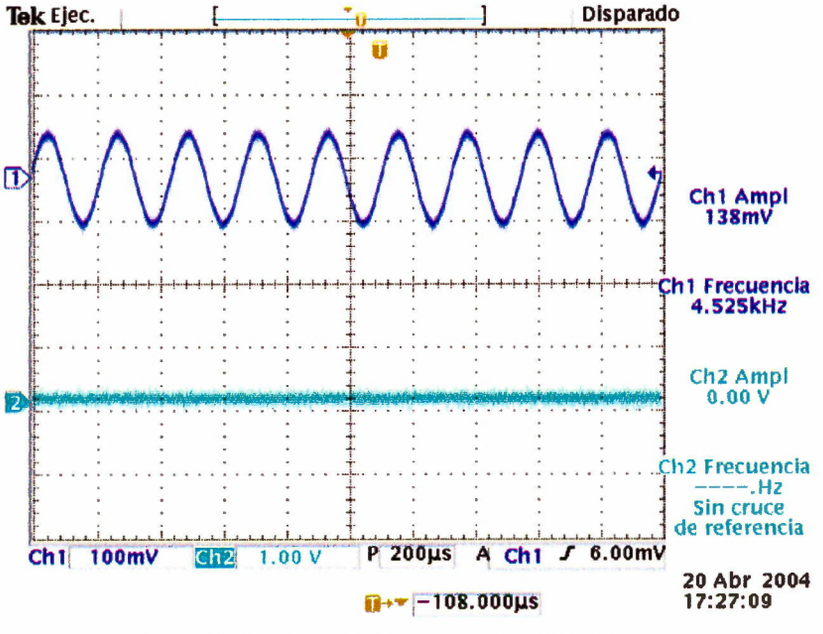


figura 8.8 Señales original y recuperada a 4500[Hz]

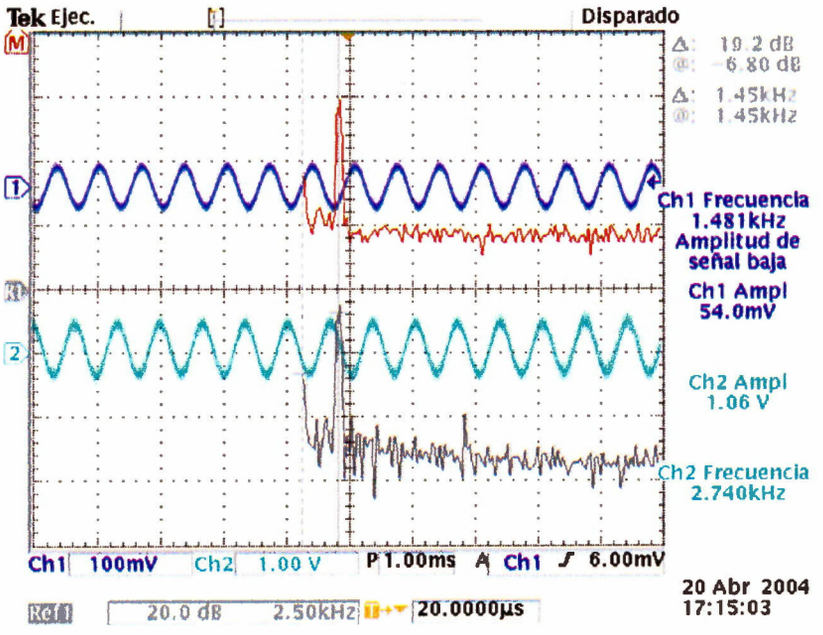


figura 8.9 Señales original y recuperada de amplitud reducida

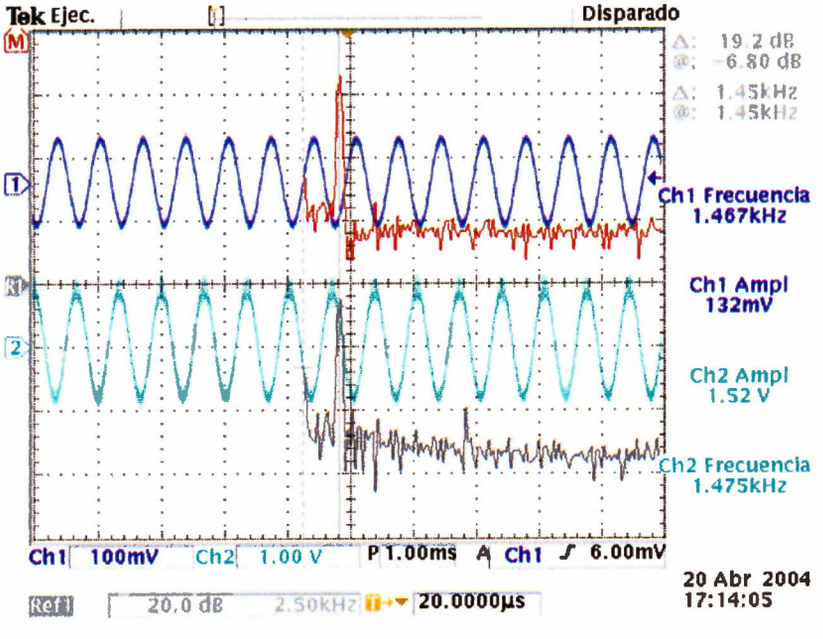


figura 8.10 Señales original y recuperada de amplitud media

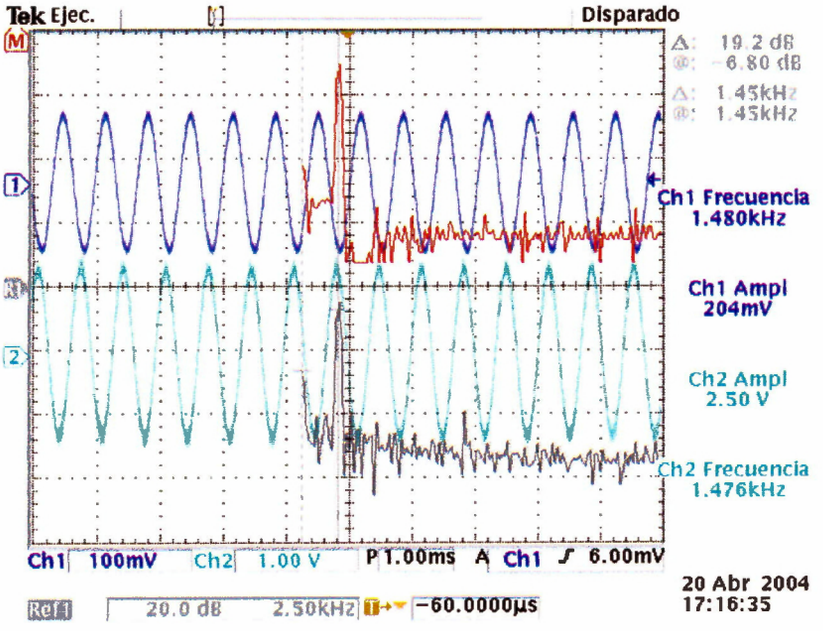


figura 8.11 Señales original y recuperada de amplitud aumentada

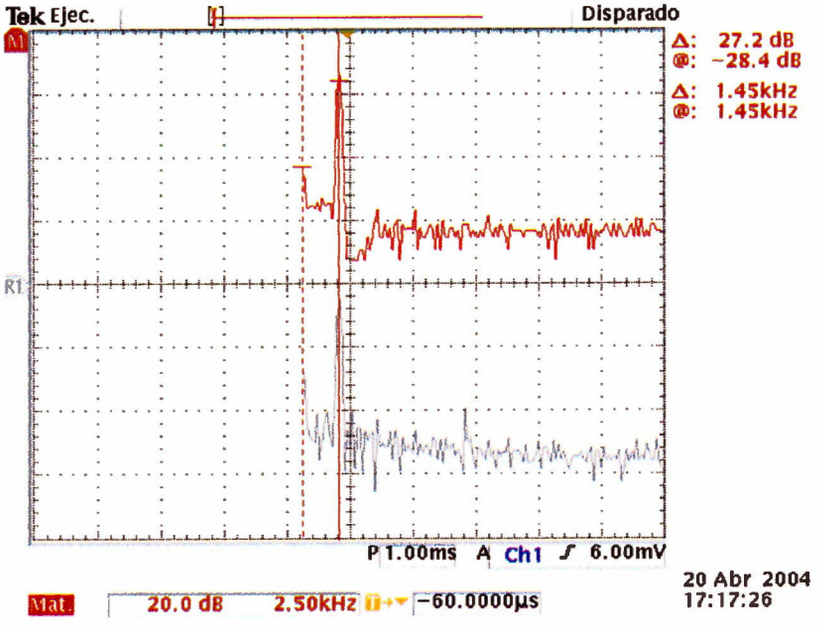


figura 8.12 Espectros de las señales original (arriba) y recuperada (abajo)

8.3. Conclusiones

En la banda de frecuencias en la que oscila la voz (figuras 8.4 a 8.8), el comportamiento temporal de la señal recuperada es muy parecido al de la señal original.

En todas las gráficas se muestra un retardo en el tiempo, el cual se atribuye al tiempo utilizado por el DSP para llevar a cabo el proceso.

En la figura 8.8 la señal recuperada desaparece, esto debido a que estamos trabajando con voz y se utiliza una frecuencia de muestro de 8 k[Hz] (según el teorema de Nyquist), ya que no nos interesa recuperar las señales con frecuencia arriba de los 4 k[Hz].

Las figuras 8.9 a 8.11 muestran diferentes amplitudes de la señal de entrada y el correspondiente valor proporcional de la señal recuperada. Esto sirve para comprobar que al variar la amplitud en la entrada, se obtiene una variación en la amplitud de la señal recuperada. Como se sabe, nosotros los humanos no hablamos siempre con el mismo volumen de voz, y es por eso que fue importante haber realizado ésta prueba.

En la última figura (8.12) se muestran los espectros de ambas señales. Como era de esperarse, ambos espectros son similares. Los dos presentan un armónico en la frecuencia de la señal original (1.45 k[Hz]), mientras que el espectro de la señal recuperada muestra algunos armónicos extras, que indican una pérdida en la fidelidad de la señal original debido a todo el proceso al que fue sometida.

Es necesario señalar que en los sistemas de comunicaciones que tratan con voz, ni el parecido temporal ni el frecuencial son cruciales. Lo que realmente importa es que la señal recuperada o reconstruida sea inteligible.

9. Segunda fase: Comunicación entre DSP's

9.1. Metodología

Para simular un sistema de comunicaciónes, se utilizaron dos DSP's, uno funcionando como DCE y otro como DTE (figura 9.1). Éste paso fue necesario para verificar una correcta transmisión de los datos a través de un medio real. Para tal propósito se utilizaron los puertos serie incluidos en los DSP's; se optó por el McBSP1 en ambos casos. A lo largo de este capítulo se explica como se logró programar los puertos serie para funcionar uno como transmisor y el otro como receptor. El objetivo de esta sección es sincronizar dos sistemas de comunicación a través de un medio de transmisión para que puedan comunicarse y compartir datos.

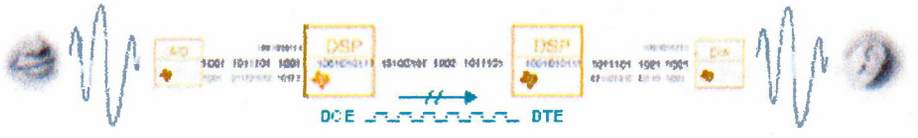


figura 9.1 Esquema de funcionamiento

Cada McBSP cuenta con 15 registros que configuran su modo de trabajo. Debido a las características de éste proyecto se trabajo sólo con nueve de estos registros, al resto no fue necesario realizarle ningún cambio. Los registros que se modificaron se encargan sobre todo de la configuración general del puerto, los modos de transmisión y recepción, el modo de trabajo del FSG (Frame-sync generator) y del SRG (Sample Rate Generador).

Los nueve registros utilizados son: SPCR[1,2], PCR, RCR[1,2], XCR[1,2] y SRGR[1,2]. Los tres primeros se utilizaron tanto en el transmisor como en el receptor, ya que estos se encargan de la configuración general del puerto. El RCR[1,2] y el XCR[1,2] se usaron en el receptor y el transmisor respectivamente. Finalmente, el SRGR[1,2] se configuró sólo en el transmisor que es el encargado de generar la señal de reloj y de sincronización para llevar a buen término la comunicación.

A continuación se explica que bits fueron modificados en todos y cada uno de los registros arriba mencionados. Se inicia el análisis con el transmisor, ya que éste fue el primero en ser

programado. Para un mayor detalle en cuanto a las funciones de cada uno de los registros usados, favor de pasar al capitulo 7, correspondiente al puerto serie del TMS320vc5416.

En el transmisor, el registro SPCR[1,2] fue el primero en ser programado, y los bits FRST, GRST y XRST los elegidos para ser modificados (figuras 9.1 y 9.2). Los bits FRST y GRST se encargan de habilitar las señales de reloj y sincronización provenientes del SRG (Sample Rate Generator), mientras que el bit XRST activa el pin DX para la transmisión de datos.

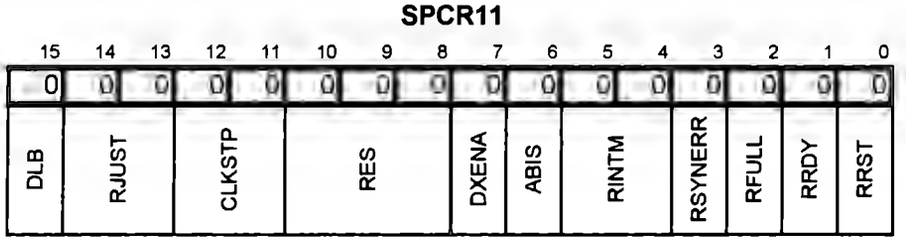


figura 9.2 Configuración del registro SPCR11

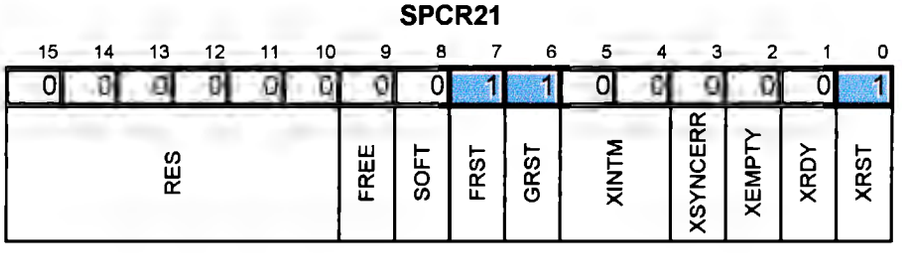


figura 9.3 Configuración del registro SPCR12

En el registro PCR se modificaron los bits FSXM y CLKXM. Estos bits habilitan los pines FSX y CLKX del puerto para que trabajen con la señal interna del DSP proveniente del SRG.

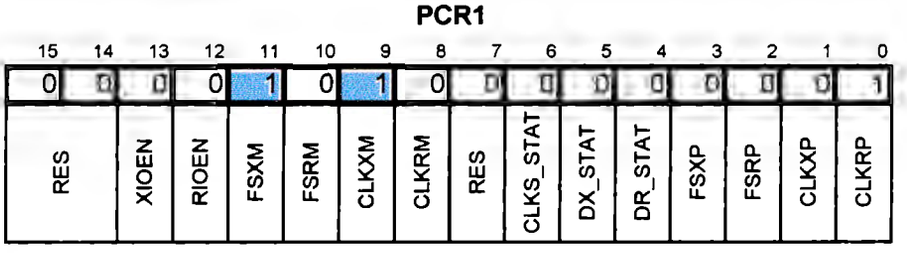


figura 9.4 Configuración del registro PCR1

El registro XCR[1,2] no sufrió cambios, ya que su configuración inicial (default) cumplió con los requerimientos necesarios para éste proyecto: número de tramas (una trama por transmisión), número de bits por trama (ocho bits por trama), número de fases (una fase por transmisión), retardos (sin retardos) y tipo de compresión (sin compresión).

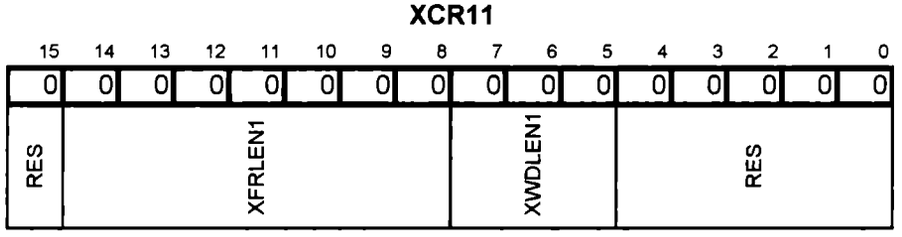


figura 9.5 Configuración del registro XCR11

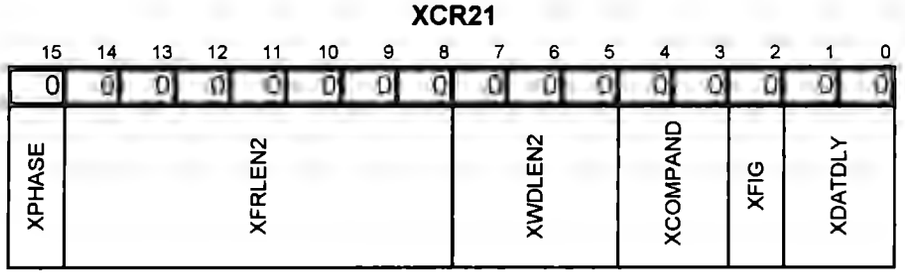


figura 9.6 Configuración del registro XCR21

Ya que el transmisor es el encargado de proporcionar la señal de reloj (de acuerdo al esquema DCE-DTE), fue necesario modificar el registro SRGR[1,2] (figuras 9.7 y 9.8) que configura el modo de trabajo del SRG.

El valor del campo FWID (bits 15-8 del SRGR1) más uno determina la duración del pulso de sincronía; en este caso se uso un valor de 10, lo cual indica que el pulso de sincronía se mantendrá en alto por once pulsos del reloj de transmisión.

El campo CLKDV (bits 7-0 del SRGR1) se encarga de tomar la señal fuente de SRG (la señal de reloj del DSP en este caso) y dividirla a petición del usuario para crear la señal del reloj de transmisión CLKG; se utilizo un valor de 255 en este campo, para alcanzar una frecuencia de 625 k[Hz], de acuerdo a la expresión:

$$CLKG = \frac{CLK\_CPU}{CLKGDV + 1} = \frac{160M[Hz]}{256} = 625k[Hz] \qquad \text{ec. 9.1}$$

Los bits CLKSM y FSGM (ambos en el SRGR2) también fueron cambiados, el primero de ellos se encarga de elegir la señal de reloj fuente del SRG; como ya se mencionó, en éste caso se utilizó la señal de reloj proveniente del DSP (160 M[Hz]). El FSGM decide en que momento será enviado el pulso de sincronía; lo cual puede depender del flujo de datos a transmitir o del FSG; por practicidad se optó por la segunda opción.

El valor del campo FPER (bits 11-0 del SRGR2) más uno divide la frecuencia de la señal CLKG para obtener la frecuencia de la señal de sincronía. Se utilizó un valor de 77 para tener un pulso de sincronía cada 125μs.

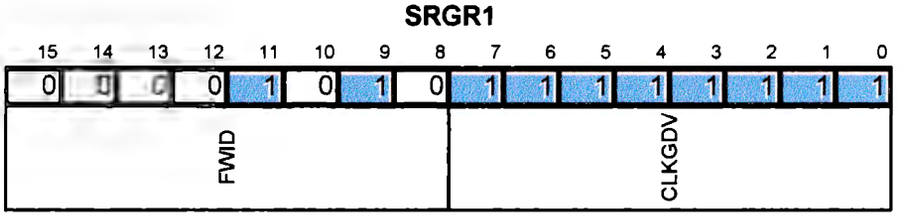


figura 9.7 Configuración del registro SRGR1

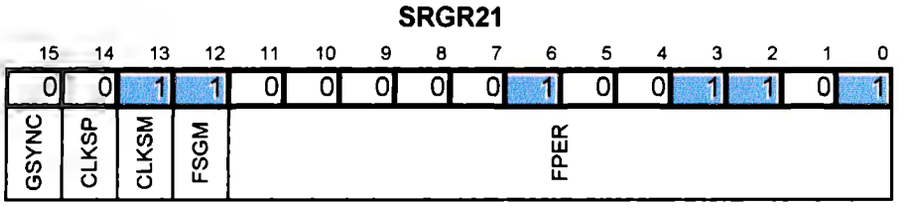


figura 9.8 Configuración del registro SRGR21

En el receptor, los registros SPCR[1,2], PCR y RCR[1,2] no sufrieron modificación alguna, ya que su configuración inicial (default) cumplió con los requerimientos necesarios para éste proyecto.

Se tomo ventaja de la tarjeta de evaluación (DSK), y más específicamente de su puerto de expansión P2, Interfaz de Periféricos Externos (External Peripheral Interface). Ya que a través de éste, se tiene acceso a los pines que componen los puertos serie McBSP 1 y 0. De ésta forma, se logró verificar el correcto funcionamiento de los puertos serie durante la comunicación, además de ser la puerta de enlace entre ambos dispositivos. Los pines utilizados para tal fin son:

Interfaz de Periféricos Externos	
Pin #	Nombre de la señal
33	CLKX1
35	FSX1
36	DX1
39	CLKR1
41	FSR1
42	DR1

figura 9.10 El puerto serie en la interfaz de periféricos externos

No está de más mencionar que las señales CLKX1, FSX1 y DX1 están presentes en el dispositivo transmisor, mientras que CLKR1, FSR1 y DR1 forman parte del receptor

9.2. Resultados

En éste apartado se muestran y comentan algunas gráficas obtenidas durante la programación de los puertos serie.

Se podrán observar las pruebas que se realizaron a lo largo de todo el proceso de configuración de los McBSP.

La figura 9.11 muestra la señal del reloj de transmisión CLKG, como puede observarse, su frecuencia es la esperada según los cálculos realizados (625 k[Hz]).

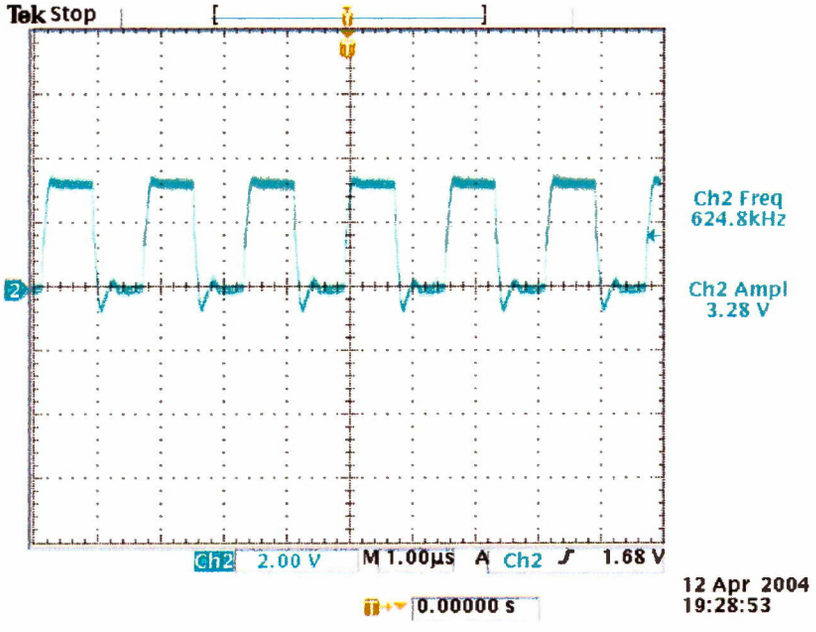


figura 9.11 Señal del reloj de transmisión

La figura 9.12 muestra el período de la sincronía de trama. La configuración anteriormente mencionada permite acercar la frecuencia a 8 k[Hz]. Esto es para asegurar que los datos serán transmitidos adecuadamente y en tiempo.

Cabe notar que el procesador es muy preciso, y el tamaño del pulso del reloj es de 1.6 µ[s], hecho que permite acomodar 77 bits dentro de cada trama. El requerimiento del sistema es de 8 bits por cada período de trama, por lo que el sistema trabaja sin contratiempos.

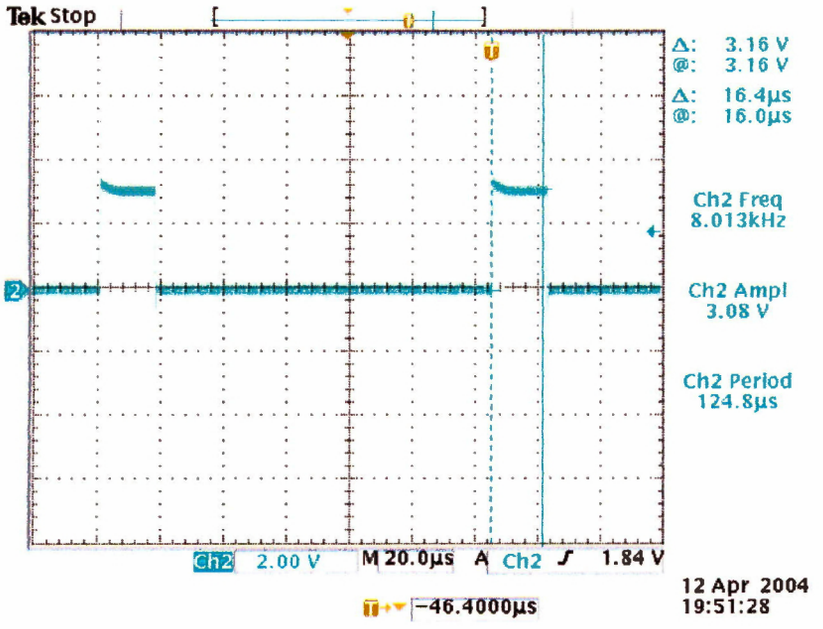


figura 9.12 Señal de sincronización del puerto serie

La última prueba (fig. 9.13) sirvió para verificar que la comunicación es correcta. Se muestra una transmisión de un número de 8 bits (A1h).

Para facilitar su lectura se acomodaron las barras verticales punteadas que limitan el número leído. Se observa que el número que se transmite es 10100001b = A1h. La transmisión se realiza de manera correcta.

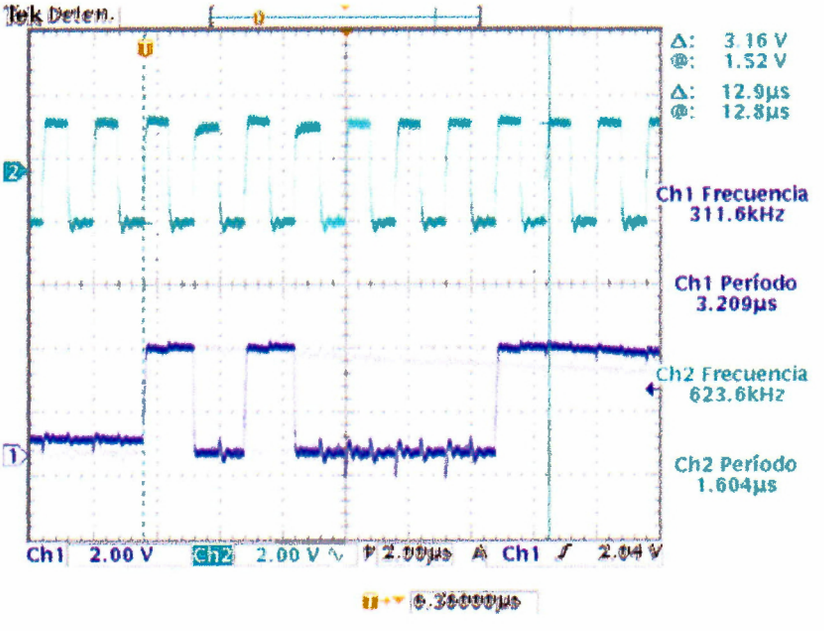


figura 9.13 Señal de reloj (arriba) y dato enviado (abajo)

Se observa que el último bit del dato enviado se mantiene fijo al final de la transmisión. Es decir que si se termina con un "uno", el DX permanece en estado alto. Lo propio sucede cuando el dato termina con un "cero".

9.3. Conclusiones

Todas las pruebas realizadas al sistema indican que la comunicación en serie se llevó a cabo de manera adecuada. Se verificó que todas las señales enviadas fueran recibidas correctamente. Desde la señal de reloj que sincroniza la transmisión, hasta el dato enviado; pasando por la señal de sincronización de trama.

Las frecuencias de las señales fueron las esperadas, por lo que se concluye que los puertos seriales del DSP's fueron correctamente configurados. Además se encontró la forma de acceder al los pines existentes en el 5416DSK, por lo que se concluye también que estos fueron identificados de manera correcta.

Para fijar la frecuencia de transmisión se tenía la opción de modificar el reloj del CPU, pero esto hubiese ocasionado una afectación en el número de MIPS del DSP. Por lo que esta opción fue descartada.

Finalmente se realizó una prueba de con voz (codificando con ley A), consiguiendo un resultado satisfactorio; de hecho, se escuchó tan bien como si todo el proceso lo hubiese realizado un solo DSP.

10. Tercera fase: Programación del modelo de predicción lineal

10.1. Metodología y resultados

El modelo de predicción lineal para el reconocimiento de la voz se programó siguiendo los pasos que se presentan en la figura 10.1.

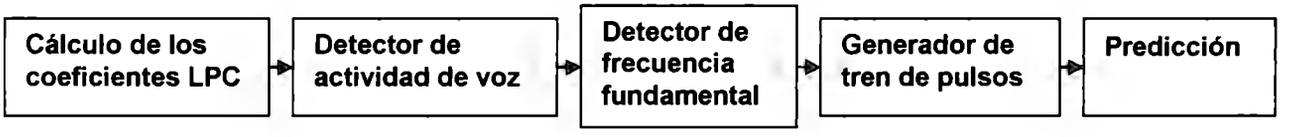


figura 10.1 Esquema de programación

Para programar el cálculo de los coeficientes de predicción lineal se usó una plataforma de programación en C independiente del procesador con el objeto de comprobar los resultados antes de implementar el código en el DSP. Con el programa MATLAB se corroboraron algunos de los valores obtenidos.

El algoritmo programado para la resolución de las ecuaciones de Yule-Walker y el consiguiente cálculo de los coeficientes LPC fue Levinson-Durbin (ver Anexos). El aporte de esté método al proyecto es el eficiente uso computacional de la memoria del procesador, hecho que permite realizar los cálculos de manera rápida y eficaz, obteniendo así, un porcentaje de error pequeño.

A continuación se muestra un ejemplo del cálculo de los coeficientes para la función:

$$y(x) = sen(x) + \cos(3x) + 0.5sen(2x) + 0.2\cos(5x) + 0.3sen(10x) \qquad \text{ec. 10.1}$$

La frecuencia que se fijó para ésta señal fue de 1600 [Hz] y un período de muestreo de 8000 [Hz]. La gráfica de la función propuesta, en el dominio del tiempo, se muestra en la figura 10.2. Se revela su carácter estacionario y periódico, condiciones necesarias para aplicar el

modelo de predicción lineal. Se muestra un segmento de 3.750 m[s], lo que corresponde a 30 muestras de voz una vez utilizada la frecuencia de muestreo (fig 10.3).

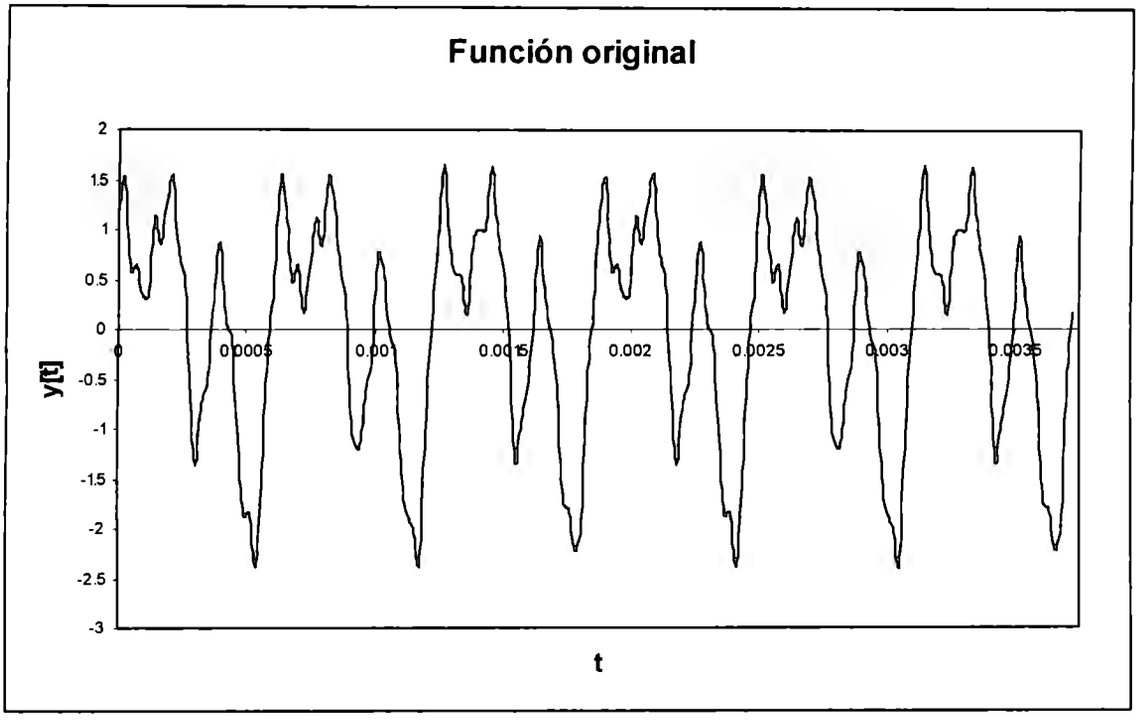


figura 10.2 Señal de prueba original

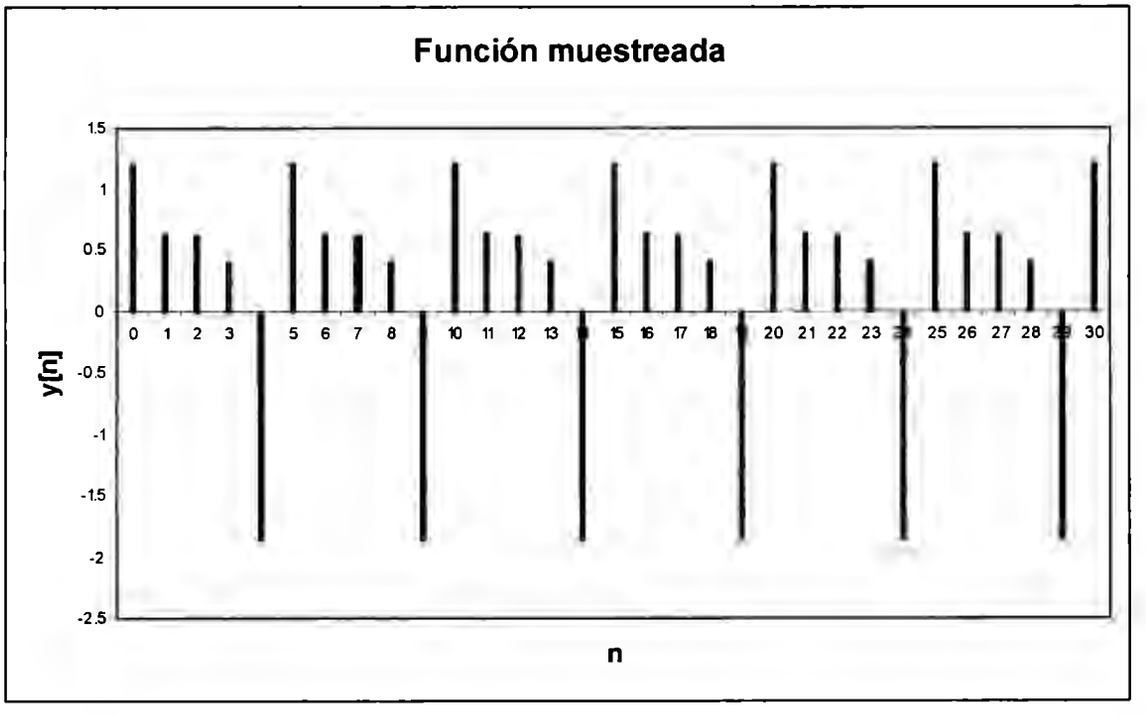


figura 10.3 Señal de prueba original muestreada

Al ejecutar el código programado en C se obtienen los resultados de la figura 10.4. En donde se muestran los coeficientes de autocorrelación necesarios para el algoritmo de Levinson-Durbin, así como los resultados de éste: los coeficientes LPC y el error cuadrático medio.

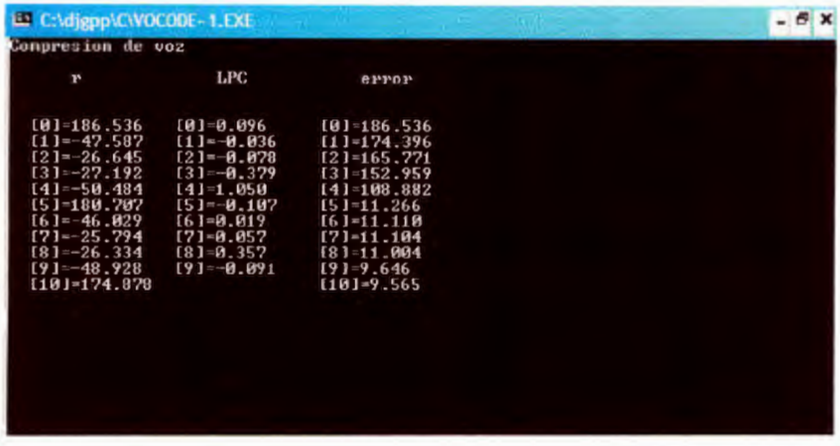


figura 10.4 Coeficientes LPC obtenidos con el programa

La figura 10.5 muestra el cálculo de los coeficientes usando MATLAB.

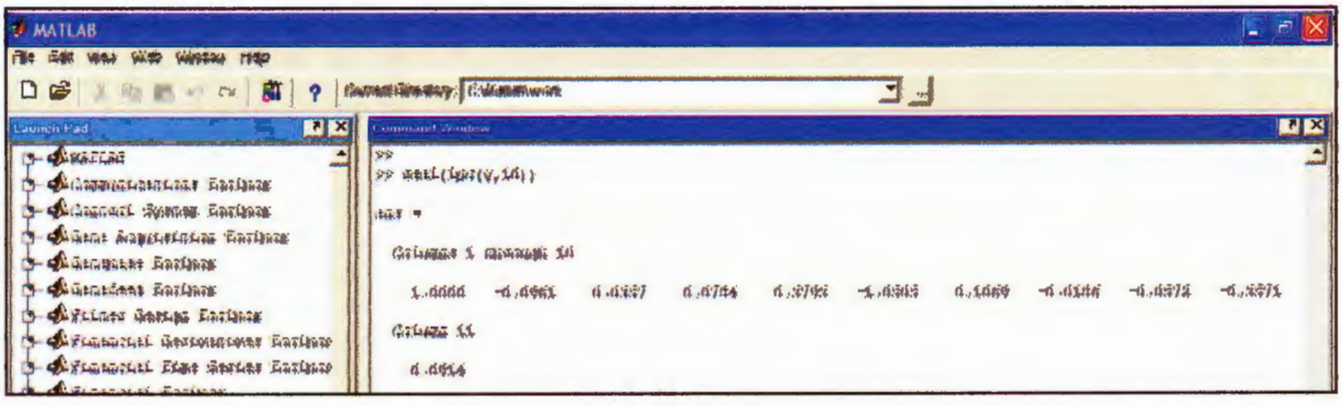


figura. 10.5 Coeficientes LPC obtenidos con MATLAB

Se observa que los resultados son idénticos en magnitud. El cambio de signo en los coeficientes LPC y la adición del uno al inicio de la serie presentada por MATLAB se deben a la forma del filtro que se enunció en la ecuación 4.6.

Para el detector de actividad de voz (trama sonora o sorda) se usó el criterio del umbral en la potencia de la señal. Según la teoría se establece un valor de umbral de $0.3R(0)$, es decir, la tercera parte de la potencia de la trama. Luego, se buscan los picos más altos de la función de autocorrelación a partir de la muestra 32 (aproximadamente los últimos cuartos quintos de trama). Para finalmente comparar el umbral con el pico más alto encontrado. Si el pico se halla por debajo del umbral la señal se cataloga como sorda, y si está por encima como sonora.

Para lograr lo anterior, se creó una función que calcula el pico máximo de un segmento de la trama (ver Anexos función "maximo"), con ayuda de la cual se programó el criterio de comparación, que decide entre un sonido sordo o sonoro. Si el sonido es sordo no es necesario calcular la frecuencia fundamental (*pitch*).

Para el cálculo de la frecuencia fundamental y la creación del tren de impulsos se usaron las funciones "centerclipping" y "pitchdetector" (ver Anexos) con los criterios de comparación descritos a detalle en el capítulo 4 (El modelo de predicción lineal para el reconocimiento de voz). El valor fijado para recortar las señales fue $K = 0.7$ (ver ecuación 4.26). La figura 10.6 muestra la señal recortada, mientras que la figura 10.7 presenta el resultado de aplicar la función de autocorrelación a dicha señal.

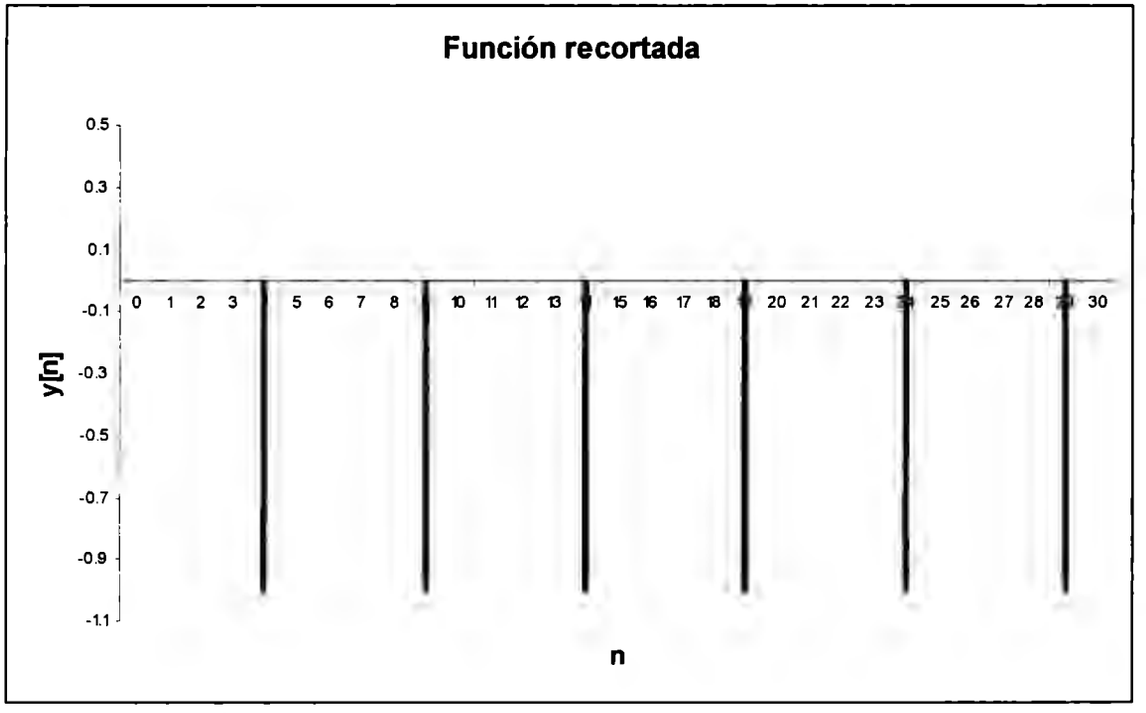


figura 10.6 Señal de prueba original muestreada y recortada

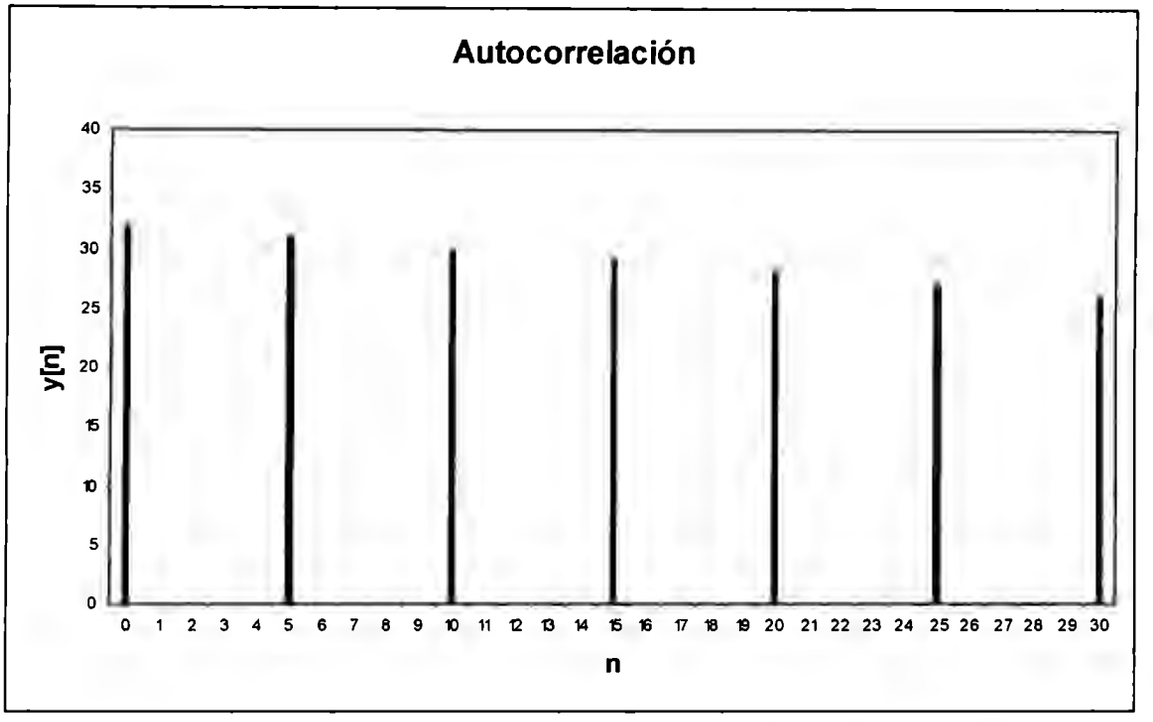


figura 10.7 Función de autocorrelación de la señal recortada

Para reconstruir la señal se integraron las funciones: "levinson_durbin", "centerclipping" y "pitchdetector" en una sola de nombre "speech" con base en la ecuación 4.30 (ver Anexos). La figura 10.8 muestra la señal recuperada junto con la original.

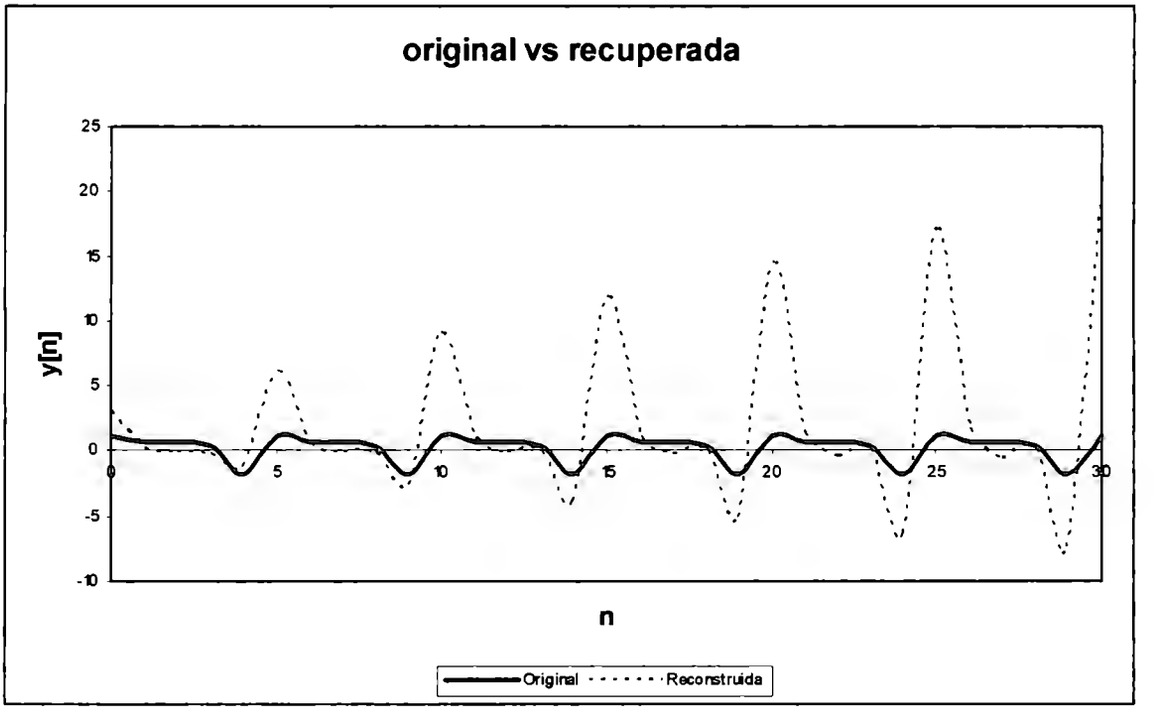


figura 10.8 Señales original y reconstruída

10.2. Conclusiones

El algoritmo de Levinson_Durbin, utilizado en el cálculo de los coeficientes de predicción lineal fue comparado con el método de Kramer; y se encontró que Levinson converge aproximadamente 10 veces más rápido. Por lo que el algoritmo usado en éste proyecto permitió el eficiente uso computacional del procesador y la memoria.

La función de autocorrelación nos muestra la similitud que tiene con ella misma recorrida en el tiempo. La información contenida en ésta función fue vital para obtener los coeficientes de predicción, la frecuencia fundamental y la ganancia.

Para el cálculo de la frecuencia fundamental los criterios de comparación enunciados en el capítulo 4 son aceptablemente buenos. Se necesitó realizar un recorte de la señal para obtener la información de los picos y procesarlos con la autocorrelación. La función de autocorrelación de la señal recortada muestra el periodo fundamental de la trama. Este método conocido como el de autocorrelación arrojó resultados precisos.

Cuando tomamos muestras de la señal perdemos información de la misma que no es relevante para la reconstrucción de la misma, siempre y cuando se tenga en cuenta el criterio de Nyquist.

11. Bibliografía

[1] Texas Instruments." TMS320C54x DSP Reference Set Vol. 2: Mnemonic Instruction Set ". spru172c.pdf. www.ti.com

[2] Texas Instruments. "TMS320C54x Reference Set Vol. 5: Enhanced Peripherals". spru302.pdf. www.ti.com

[3] Texas Instruments. "TMS320C54x Fixed Point Digital Signal Processor Vol. 1: CPU and Peripherals". tms320vc5416.pdf. www.ti.com

[4] Romero, Mares. "Apuntes de la clase de introducción a DSP". Profesor: Alfredo Mantilla. ITESM-CCM

[5] Hanzos Lajos, Somerville Clae, Woodard Jason. "Voice compression and communications". IEE Series on Digital and Mobile Communication, New Jersey: 2001.

[6] ITU-T. "Fascículo III.4 – Recomendación G.711". Libro azul.

[7] Texas Instruments. "PCM 3002,3003". PCM3002.pdf. www.burr-brown.com

[8] Texas Instruments. "TMS320VC5416 DSK Technical Reference". 5416_dsk_techref.pdf.

[9] Texas Instruments. TMS320C54x Code Composer Studio Help.

[10] Spectrum Digital. "TMS320VC5416 DSK Technical Reference". 5416_dsk_techref.pdf. www.spectrumdigital.com

[11] Rabiner, Juang. "Fundamentals of speech recognition". Prentice Hall, New Jersey: 1993.

[12] Haykin. "Adaptive Filter Theory". Prentice Hall, New Jersey:1996.

[13] Séller, Proakis, Hansen. "Discrete-Time Processing of Speech Signals". Mc Millan, New York: 1993.

[14] Box, Jenkins, Reinsel. "Time Series Analysis". Prentice Hall, New Jersey: 1994.

[15] Gold, Morgan. "Speech and audio signal processing". John Wiley, New York: 2000.

12. Anexos

- Código del programa (el código se comentó en español). Se presentan tres programas:
    - Vocoder
    - Transmisor
    - Receptor

```
//-------------------------------//
//          ITESM                //
// Proyectos de Ingeniería II    //
//   "Compresión de voz"         //
//                               //
// César Romero Núñez            //
// David Mares Nava              //
//-------------------------------//


//-------------------------------//
//Este programa calcula:         //
//          1 - Los LPC          //
//          2 - El tren de impulsos //
//              con la frec fundamental //
//          3 - Reconstruye la señal //
//-------------------------------//


// Llamado a las librerías necesarias

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

// Declaración de las variables globales
// Se define una ventana de 160 muestras máximo

float xclip[160];
float rcom[160];
float rcomx[160];
float r[11];
float x[160];
float a[10]={0,0,0,0,0,0,0,0,0,0};
float a1[11]={0,0,0,0,0,0,0,0,0,0,0};
float a2[11]={0,0,0,0,0,0,0,0,0,0,0};
float a3[11]={0,0,0,0,0,0,0,0,0,0,0};
float a4[11]={0,0,0,0,0,0,0,0,0,0,0};
float a5[11]={0,0,0,0,0,0,0,0,0,0,0};
float a6[11]={0,0,0,0,0,0,0,0,0,0,0};
float a7[11]={0,0,0,0,0,0,0,0,0,0,0};
float a8[11]={0,0,0,0,0,0,0,0,0,0,0};
float a9[11]={0,0,0,0,0,0,0,0,0,0,0};
float a10[11]={0,0,0,0,0,0,0,0,0,0,0};
float e[11]={0,0,0,0,0,0,0,0,0,0,0};
float pp[160];
float voz[160];
float pred[10];
char bandera;
int muestras=160;
int pantallax=3;
int frecuencia=1600;
int orden=10;

// Se define el valor de pi

#define pi 3.1416
```

```
//----------Funciones----------//

/* Recibe el numero de muestras, necesariamente menor
que
160 y calcula los mismos coeficientes de autocorrelacion
de la señal x y los guarda en r */

void autocompleta(int muestras, float *x, float *r)
{
 int k,n;

 for(k=0;k<=muestras;k++)
   {
    r[k]=0;
    for(n=0;n<muestras;n++)
      {
       if((n+k)<muestras)
          r[k]=r[k]+x[n]*x[n+k];
                        }
   }
}

/* Genera la señal de prueba, la frecuencia está dada en
Hz*/

void senoidal(int frecuencia, int nmuestras, float *seno)
{
 int i;
 float w;

 w=2*pi*frecuencia;

 for(i=0;i<nmuestras;i++)

seno[i]=sin(w*i/8000)+cos(3*w*i/8000)+.5*sin(2*w*i/8000)+.2
*cos(5*w*i/8000)+.3*sin(10*w*i/8000)+.3*cos(10*w*i/8000);
   //seno[i]=sin(w*i/8000);
   }

/* Imprime en pantalla el arreglo que recibe */

void imprime(int longitud, float *array)
{
 int i;

 clrscr();

 for(i=0;i<longitud;i++)
   {
    printf("%f |",array[i]);
    }
}

/* Regresa el valor absoluto de una variable */

float absoluto(float valor)
{
 if(valor<0)
   valor=-1*valor;
 return valor;
}
```

```
/* Regresa el valor máximo de un arreglo,
recibe el arreglo y el punto de inicio de barrido
tomar en cuenta que el nmuestras es el tamaño de
frame menos el inicio de cuenta */


float maximo(float *x, int nmuestras, int inicio)
{
 int i;
 float A1,actual;

 A1=absoluto(x[inicio]);
 for(i=1;i<nmuestras;i++)
    {
     actual=absoluto(x[i+inicio]);
     if(actual>A1)
       A1=actual;
      }
 return A1;
}

/* Recibe la señal para recortar y el arreglo donde
guardara la señal recortada */

void centerclipping(float *x,float *xclip, int nmuestras)
{
 int i;
 float A1,A2;
 float min;

 A1=maximo(x,(nmuestras/3),0);
 A2=maximo(x,(nmuestras/3),2*(nmuestras/3));

 if(A1<A2)
  min=A1;
  else
   min=A2;

 min=min*.8;

 for(i=0;i<nmuestras;i++)
   {
    if(x[i]>=min)
     xclip[i]=1;
     else if(x[i]<=(-1*min))
      xclip[i]=-1;
      else
       xclip[i]=0;
       }
}
```

```
/* Recibe el arreglo original de muestras, la señal
recortada, la correlación completa [160 muestras]
y el número de muestras */

void pitchdetector(float *x, float *xclip, float *r,float *pp, int
nmuestras)
{
 int i=0;
 centerclipping(x,xclip,nmuestras);
 imprime(nmuestras,xclip);
 getch();
 clrscr();
 autocompleta(nmuestras,xclip,r);
 autocompleta(nmuestras,x,rcomx);
 imprime(nmuestras,r);
 getch();
 for(i=0;i<nmuestras;i++)
   {
    if(r[i]>0)
      pp[i]=1;
      else
        pp[i]=0;
    }

   imprime(nmuestras,pp);
   getch();
}

/* Calcula la autocorrelación para un arreglo de maximo
160
muestras y regresa #(orden) coeficientes de
autocorrelación
Recibe en x y regresa en r*/

void autocorrelacion(int muestras, int orden, float *x, float
*r)
{
 int j,m;

 for(j=0;j<=orden;j++)
   {
    r[j]=0;
    for(m=0;m<=(muestras-j-1);m++)
      r[j]=r[j]+x[m]*x[m+j];
    }
}

/* Imprime un vector agregandole su nombre en pantalla*/

void imprime_vector(int tamano, float *vector, char
*nombre)
{
 int i;
 gotoxy(pantallax+4,3);
 puts(nombre);
 for(i=0;i<tamano;i++)
   {
    gotoxy(pantallax,i+6);
    printf("[%i]=%.3f\n ",i,vector[i]);
    }
   pantallax=pantallax+14;
}
```

```
/* Función para condiciones iniciales del algoritmo de
Levinson*/

void inicializar(void)
{
 e[0]=r[0];
 }

/* Regresa la dirección del arreglo cuyo número esta dado.
Son diez arreglos máximo y el número regresa la dirección
del
arreglo seleccionado. Ej regresa_a(3). Con esto obtienes la
dirección del arreglo a3 */

float* regresa_a(int numero)
{
 float *b;
 switch(numero)
  {
   case 1: b=a1 ; break;
   case 2: b=a2 ; break;
   case 3: b=a3 ; break;
   case 4: b=a4 ; break;
   case 5: b=a5 ; break;
   case 6: b=a6 ; break;
   case 7: b=a7 ; break;
   case 8: b=a8 ; break;
   case 9: b=a9 ; break;
   case 10: b=a10 ; break;
  }
  return b;
 }

/* Cálcula el error según la ecuación
de Levinson*/

void error(int i)
{
 float *a;
 a=regresa_a(i);
 e[i]=(1-a[i]*a[i])*e[i-1];
 }

/* Cálculo los índices PARCOR del algoritmo
de Levinson durbin */

void aidei(int i)
{
 float *a,*b;
 int j;
 float suma=0;

 a=regresa_a(i);

 for(j=1;j<=(i-1);j++)
          {
   b=regresa_a(j);
   suma=suma+b[i-1]*r[i-j];
   }
  if(i==1)
          a[i]=(r[i]-suma)/(e[i-1]);
  else
   {
    error(i-1);
    a[i]=(r[i]-suma)/(e[i-1]);
   }
 }


/* Sirve para el calculo del coeficiente
aij para el algoritmo de Levinson*/

void a(int i,int j)
{
 float *a,*b,*c;
 a=regresa_a(j);
 b=regresa_a(i);
 c=regresa_a(i-j);
 aidei(j);
 a[i]=a[i-1]-b[i]*c[i-1];
 }

/* Almacena la última iteración de los
coeficientes LPC en el arreglo af*/

void almacena_a(int orden)
{
 float *b;
 int i;
 for(i=1;i<=orden;i++)
   {
    b=regresa_a(i);
    af[i-1]=b[orden];
    }
 }

/* El cartel del entrada*/

void cartel(void)
{
 gotoxy(25,5);
 printf("Tecnologico de Monterrey");
 gotoxy(15,7);
 printf("C lculo de los coeficientes de predicci¢n lineal");
 gotoxy(25,11);
 printf("Proyectos de Ingenieria II");
 gotoxy(27,15);
 printf("Compresion de voz");
 getch();
 clrscr();
}
```

```
/* Función que calcula los coeficientes LPC */

void levinson_durbin(int orden)
{
 int i,j;
 //char basura;
 inicializar();
 for(i=1;i<=orden;i++)
   {
   aidei(i);
    for(j=1;j<=(i-1);j++)
      {
       a(i,j);
      }
    error(i);
   }
 almacena_a(orden);
 gotoxy(20,10);
 printf("Listo para mostrar resultados");
 gotoxy(22,12);
 printf("Presiona cualquier tecla");
 getch();
 clrscr();
 printf("Compresion de voz");
 imprime_vector(orden+1,r,"r");
 imprime_vector(orden,af,"LPC");
 imprime_vector(orden+1,e,"error");
 getch();
}


void imprime_tren(float *x,int len)
{
 int i;

 for(i=0;i<len;i++)
   {
    printf("%.0f ",x[i]);
   }
 clrscr();
 printf("\n");
 getch();
  }

/* Con esta función se regresa la muestra completa de voz
con solamente 10 coeficientes lpc [af] el tren de impulsos
[pp]
el numero del orden del filtro predictor y el numero de
muestras */

void speech(float *pp,float *voz, float *af, float g, int
orden,int muestras)
{
 int i,j;

 for(i=0;i<muestras;i++)
   {
    voz[i]=0;
   for(j=0;j<orden;j++)
     {
      if((i-j-1)>=0)
       voz[i]=voz[i]+af[j]*voz[i-j-1];
      }
  voz[i]=voz[i]+g*pp[i];
      }
}

/* Imprime la señal de entrada vs reconstruida */

void final(float *original, float *reconstruida, int muestras)
{
 int i,j,x;
 x=0;
 for(i=0;i<4;i++)
   {
    for(j=0;j<40;j++)
      {
       //if((j+x)<muestras)
       printf("[%i]
%f\t\t%f\n",j,original[j+x],reconstruida[j+x]);
      }
    x=40*(i+1);
    getch();
   }
}



/* Programa principal */

int main ()
{
 int i;

 /* Archivos de salida para graficar en excel */

 FILE *original,*autoc,*clipp,*tren,*recons,*rcompx;


 cartel();

 /* Generación de la señal de voz */

 senoidal(frecuencia, muestras, x);
 imprime(muestras,x);

 getch();

 /* Detector de la frecuencia fundamental */

 pitchdetector(x,xclip,rcom,pp,muestras);

 /* LPC */

 autocorrelacion(muestras,orden,x,r);
 levinson_durbin(orden);

 /* Reconstrucción de la voz */

 speech(pp,voz,af,1.2,orden,muestras);

 clrscr();

 final(x,voz,muestras);
 getch();
```

```
/* Generación de archivos, éstos se
encuentran en la carpeta donde está el
principal */

original=fopen("original.lpc","w");
autoc=fopen("autoc.lpc","w");
clipp=fopen("clipp.lpc","w");
tren=fopen("tren.lpc","w");
recons=fopen("recons.lpc","w");
rcompx=fopen("comx.lpc","w");


for(i=0;i<muestras;i++)
  {
   fprintf(original,"%f\n",x[i]);
   fprintf(autoc,"%f\n",rcom[i]);
   fprintf(clipp,"%f\n",xclip[i]);
   fprintf(tren,"%f\n",pp[i]);
   fprintf(recons,"%f\n",voz[i]);
   fprintf(rcompx,"%g\n",rcomx[i]);
   }
fclose(original);
fclose(autoc);
fclose(clipp);
fclose(tren);
fclose(recons);


return 0;
}
```

```
//-------------------------------------//
//            ITESM                    //
//  Proyectos de Ingeniería II         //
//     "Compresión de voz"             //
//                                     //
// César Romero Núñez                  //
// David Mares Nava                    //
//-------------------------------------//

//-------------------------------------//
//Este programa:                       //
//1 - Adquiere datos por el McBSP2     //
//2 - Los digitaliza usando la ley A   McBSP0 //
//3 - Los envia en formato digital por el puerto //
//                                     //
//-------------------------------------//
#include "tonecfg.h"
#include "dsk5416.h"
#include "dsk5416_pcm3002.h"
#include <math.h>
#include <stdio.h>

#define PI          ((double)3.1415927)

/* Las funciones declaradas globlamente, realizadas
   en ensamblador */

extern void A_law_McBSP0_config(void);
extern void inverse_A_law_McBSP1_config(void);
extern void enable_serial(void);
extern void A_law(Int16 A);
extern void A_law1(Int16 A);
extern void A_law2(Int16 A);
extern void enable_interrupts(void);
extern void inverse_A_law(Int16 A);
extern void initial(void);

/* Los prototipos de funciones locales*/

extern Int array[40];

/* Las variables globales declaradas en ensamblador
   y usadas en C*/

/* Las variables locales declaras en c y usada en
   ensamblador*/

volatile Int16 channel_A, exit_channel;
Int16 channel,trash_channel;

/* La configuración del registro del codec*/

DSK5416_PCM3002_Config setup = {

    0x1FF,     // Set-Up Reg 0 - Atenuación del canal
izquierdo
    0x1FF,     // Set-Up Reg 1 - Atenuación del canal derecho
    0x0,       // Set-Up Reg 2 - Otros
    0x0        // Set-Up Reg 3 - Otros
};
```

```
/*
 * El principal
 */

void UserTask()
{
   DSK5416_PCM3002_CodecHandle hCodec;
          int i,j;

   // Inicializar el Codec

   hCodec = DSK5416_PCM3002_openCodec(0, &setup);

   //Frecuencia de muestreo = 8kHz

   DSK5416_PCM3002_setFreq(hCodec, 8000);

   //Inicializar la ley A

   A_law_McBSP0_config();
   inverse_A_law_McBSP1_config();

   // Inicializar el puerto serie para transmisión

   initial();
   enable_serial();


   for(i=0;i<8000000;i++)
    {
                  for(j=0;j<8000;j++)
                  {


                  //Leyendo del Codec


          while(!DSK5416_PCM3002_read16(hCodec,
&channel));

                  // Esta otra será descartada


          while(!DSK5416_PCM3002_read16(hCodec,
&trash_channel));

                  //Codificación Ley A

                  A_law(channel);

                  }
          }


   // Cerrar el Codec

   DSK5416_PCM3002_closeCodec(hCodec);
}
```

```
.mmregs
 .global _enable_serial
  .data
SPSA1 .set 0048h
SPCR11 .set 0049h
SPCR21 .set 0049h
PCR1 .set 0049h
RCR11 .set 0049h
RCR21 .set 0049h
XCR11 .set 0049h
XCR21 .set 0049h
SRGR11 .set 0049h
SRGR21 .set 0049h
 .text
_enable_serial:
;Las funciones para la transmision de datos
;por el McBSP 0

        ;-------------------
        stm #0000h,SPSA1
        ldm SPCR11,A
        or #0000000000000000b,A
        stlm A,SPCR11
        stm #0001h,SPSA1
        ldm SPCR21,A
        or #0000001011000001b,A
        stlm A,SPCR21
        ;-------------------
        stm #000Eh,SPSA1
        ldm PCR1,A
        or #0000101000000000b,A
        stlm A,PCR1
        ;-------------------
        stm #0002h,SPSA1
        ldm RCR11,A
        or #0000000000000000b,A
        stlm A,RCR11
        stm #0003h,SPSA1
        ldm RCR21,A
        or #0000000000000000b,A
        stlm A,RCR21
        ;-------------------
        stm #0004h,SPSA1
        ldm XCR11,A
        or #0000000000000000b,A
        stlm A,XCR11
        stm #0005h,SPSA1
        ldm XCR21,A
        or #0000000000000000b,A
        stlm A,XCR21
        ;-------------------
        stm #0006h,SPSA1
        ldm SRGR11,A
        or #0000101011111111b,A
        stlm A,SRGR11
        stm #0007h,SPSA1
        ldm SRGR21,A
        or #0011000001001101b,A
        stlm A,SRGR21
        ;-------------------

        fret

.end
```

```
.mmregs

 .global _A_law_McBSP0_config

 .data

SPSA0 .set 0038h
SPCR10 .set 0039h           ;sub-address 0x0000
XCR20 .set 0039h  ;sub-address 0x0005

 .text

_A_law_McBSP0_config:

;La funcion para activar la ley A por el McBSP0

        stm #0000h,SPSA0        ;sub-address
SPCR10
        ldm SPCR10,A
        or #8000h,A
        stlm A,SPCR10           ;activando DLB

        stm #0005h,SPSA0        ;sub-address XCR20
        ldm XCR20,A
        or #0018h,A
        stlm A,XCR20            ;XCOMPAND=11 para
ley A

        fret

.end
```

```
.mmregs

.global _A_law

.global _channel_A

.global _receiving_A_law

.data

DRR10 .set 0021h
DXR10 .set 0023h
DXR11 .set 0043h

; Esta función recibe el valor leido en el McBSP 2(DRR22) y
lo manda al serial DxR10
; recordar que RR especifica que el registro es de
recepción el siguiente digito
; indica que registro está en uso y el último el puerto serial
utilizado.


.text

_A_law:

        stlm A,DXR10        ;enviar al McBSP0 para ley A
                            ;el dato en formato lineal
                            ;está en el acumulador A

  NOP
  NOP
  NOP
  NOP
  NOP

        ;fret               ;regresa a C

;_receiving_A_law:          ;viene de vector rint0

        ldm DRR10,A                 ;leer del McBSP0 la
ley A
        stl A,*(_channel_A)         ;y enviar a channel_A
        nop
        nop
        nop
        ;ld *(_channel_A),8,A
        nop
        nop
        nop
        ;ld #101010101110b,A
        nop
        nop
        nop
        stlm A,DXR11
        ;stl A,*(_array)
        nop
        nop
        nop


        fret                        ;fin de la interrupción

.end
```

```
//-------------------------------------//
//            ITESM                    //
// Proyectos de Ingeniería II          //
//    "Compresión de voz"              //
//                                     //
// César Romero Núñez                  //
// David Mares Nava                    //
//-------------------------------------//


//-------------------------------------//
//Este programa:                       //
//1 - Adquiere datos por el puerto serie//
//2 - Los digitaliza usando la ley A en el McBSP0 //
//3 - Los envia en formato analógico por el McBSP2 //
//                                     //
//-------------------------------------//

#include "tonecfg.h"
#include "dsk5416.h"
#include "dsk5416_pcm3002.h"
#include <math.h>
#include <stdio.h>

#define PI              ((double)3.1415927)

/* Las funciones declaradas globlamente, realizadas
   en ensamblador */

extern void A_law_McBSP0_config(void);
extern void inverse_A_law_McBSP1_config(void);
extern void enable_serial(void);
extern void A_law(Int16 A);
extern void A_law1(Int16 A);
extern void A_law2(Int16 A);
extern void enable_interrupts(void);
extern void enable_serial(void);
extern void inverse_A_law(Int16 A);
extern void initial(void);

/* Los prototipos de funciones locales*/

extern Int array[40];

/* Las variables globales declaradas en ensamblador
   y usadas en C*/

/* Las variables locales declaras en c y usada en
   ensamblador*/

volatile Int16 channel_A, exit_channel;
Int16 channel,trash_channel;

/* La configuración del registro del codec*/

DSK5416_PCM3002_Config setup = {

    0x1FF,     // Set-Up Reg 0 - Atenuación del canal
izquierdo
    0x1FF,     // Set-Up Reg 1 - Atenuación del canal derecho
    0x0,       // Set-Up Reg 2 - Otros
    0x0        // Set-Up Reg 3 - Otros
};
```

```
/*
 * UserTask() - El principal
 */

void UserTask()
{
   DSK5416_PCM3002_CodecHandle hCodec;
            int i,j;

   // Iniciar el Codec
   hCodec = DSK5416_PCM3002_openCodec(0, &setup);

   //Frecuencia de muestreo = 8kHz
   DSK5416_PCM3002_setFreq(hCodec, 8000);

   //Inicializar el McBSP0 para ley a inversa

   inverse_A_law_McBSP1_config();

   // Habilitar el puerto serie para recepción

   enable_serial();

   for(i=0;i<800000;i++)
    {
                    for(j=0;j<8000;j++)
                    {

                    // La ley A inversa

                    inverse_A_law(channel_A);


                    //Escribiendo al codec en stereo


            while(!DSK5416_PCM3002_write16(hCodec,
exit_channel));

            while(!DSK5416_PCM3002_write16(hCodec,
exit_channel));
                    }
            }
   // Cerrando el Codec
   DSK5416_PCM3002_closeCodec(hCodec);
}

void main()
{
   // Inicializando la librería de soporte

   DSK5416_init();
}
```

```
.mmregs
 .global _enable_serial
 .data
SPSA1 .set 0048h
SPCR11 .set 0049h
SPCR21 .set 0049h
PCR1 .set 0049h
RCR11 .set 0049h
RCR21 .set 0049h
XCR11 .set 0049h
XCR21 .set 0049h
SRGR11 .set 0049h
SRGR21 .set 0049h

 .text
;Para recibir por el Puerto serie 0

_enable_serial:
        ;-------------------
        stm #0000h,SPSA1
        ldm SPCR11,A
        or #0000000000000001b,A
        stlm A,SPCR11
        stm #0001h,SPSA1
        ldm SPCR21,A
        or #0000000000000000b,A
        stlm A,SPCR21
        ;-------------------
        stm #000Eh,SPSA1
        ldm PCR1,A
        or #0000000000000000b,A
        stlm A,PCR1
        ;-------------------
        stm #0002h,SPSA1
        ldm RCR11,A
        or #0000000000000000b,A
        stlm A,RCR11
        stm #0003h,SPSA1
        ldm RCR21,A
        or #0000000000000000b,A
        stlm A,RCR21
        ;-------------------
        stm #0004h,SPSA1
        ldm XCR11,A
        or #0000000000000000b,A
        stlm A,XCR11
        stm #0005h,SPSA1
        ldm XCR21,A
        or #0000000000000000b,A
        stlm A,XCR21
        ;-------------------
        stm #0006h,SPSA1
        ldm SRGR11,A
        or #0000000000000000b,A
        stlm A,SRGR11
        stm #0007h,SPSA1
        ldm SRGR21,A
        or #0000000000000000b,A
        stlm A,SRGR21
        ;-------------------

        fret

.end
```

```
.mmregs

 .global _inverse_A_law_McBSP1_config

 .data

;Preparada para la ley A inversa


SPSA1 .set 0048h
SPCR11 .set 0049h          ;sub-address 0x0000
RCR21 .set 0049h           ;sub-address 0x0003

SPSA0 .set 0038h
SPCR10 .set 0039h          ;sub-address 0x0000
XCR20 .set 0039h           ;sub-address 0x0005
RCR20 .set 0039h


 .text

_inverse_A_law_McBSP1_config:

        stm #0000h,SPSA0        ;sub-address
SPCR1x
        ldm SPCR10,A
        or #8000h,A
        stlm A,SPCR10           ;activando DLB

        stm #0003h,SPSA0        ;sub-address RCR2x
        ldm RCR20,A
        or #0018h,A
        stlm A,RCR20            ;RCOMPAND=11 para
ley A inversa

        fret

 .end
```

```
.mmregs

 .global _inverse_A_law

 .global _exit_channel

 ;.ref receiving_inverse_A_law

 .data

DRR11 .set 0041h
DXR11 .set 0043h

DRR10 .set 0021h
DXR10 .set 0023h

 .text

_inverse_A_law:

        ldm DRR11,A
        nop
        nop
        nop

        stlm A,DXR10                    ;enviar al
McBSP1 para ley A
                                        ;inversa el
dato en formato de
                                        ;ley A está
en el acumulador A


        ;fret                           ;regresa a C
        NOP
        NOP
        NOP
        NOP

;receiving_inverse_A_law:  ;viene de rint1 en vectors

        ldm DRR10,A             ;leer del McBSP1 la
ley A inversa
        stl A,*(_exit_channel)  ;y enviar a
exit_channel

        ;rete                   ;fin de la interrupción
        fret

 .end
```