PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

FILTRO ADAPTIVO LMS Y SU APLICACIÓN EN EL RECONOCIMIENTO DE PALABRAS AISLADAS PARA EL CONTROL DE UN EQUIPO DE SONIDO POR MEDIO DE LA VOZ

Tesis para Optar el Título de:

INGENIERO ELECTRÓNICO

Presentado por:

GIOVANI SAID SIMÓN BENDEZÚ

Lima - Perú

2004

RESUMEN

El intentar controlar un equipo de sonido por medio de órdenes verbales es un gran desafío, debido a que los algoritmos de reconocimiento de voz son muy frágiles y tienen una buena eficiencia sólo en un ambiente sin ruido, basta la presencia de un ligero sonido para que la voz no pueda ser reconocida fácilmente. Esta investigación intenta mejorar el porcentaje de aciertos de un sistema de reconocimiento ante la presencia de ruido, y por tal motivo se va a intentar controlar un equipo de sonido marca *Panasonic* y modelo SC-AK45 con control remoto modelo EUR644853, del cual se tomarán algunas de las funciones para el control mediante la voz.

Para alcanzar este objetivo, se emplea un filtro adaptivo LMS que es la interfase de entrada al sistema y que utiliza dos señales (la voz con ruido y la referencia) capturadas por dos micrófonos marca AKG D-230 implementado en la tarjeta de evaluación DSP56002EVM de Motorola. Este filtro cumple la función de reducir el nivel del sonido del equipo en la señal que contiene la voz para que el sistema de reconocimiento pueda tener buena eficiencia y por lo tanto un bajo índice de error.

La señal filtrada ingresa al sistema de reconocimiento implementado en la tarjeta de desarrollo TMS320C6711 que permite el control en tiempo real, es decir en el instante en que se pronuncie la orden, el sistema procesa la señal de voz y genera una respuesta enviando una señal infrarroja a través de un pequeño circuito IR que es la interfase entre la tarjeta y el equipo de sonido. Por último, esta investigación es la base para controlar cualquier sistema que posea audio, como el caso de un TV, un DVD, un VHS, etc., es decir para controlar todo un sistema audio visual mediante la voz.

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título	:	**F**iltro adaptivo LMS y su aplicación en el reconocimiento de palabras aisladas para el control de un equipo de sonido por medio de la voz
Área	:	Procesamiento Digital de Señales
Asesor	:	Ing. Andrés Flores Espinoza
Alumno	:	Giovani Said Simón Bendezu
Código	:	1991.1259.7.12
Fecha	:	10 de Setiembre del 2003

Descripción y Objetivos

El presente proyecto tiene como objetivo la implementación de un filtro adaptivo LMS (Least Mean Square) y un sistema de reconocimiento de palabras aisladas en tiempo real utilizando la tarjeta de desarrollo TMS320C6711 de Texas Instruments con la finalidad de controlar un equipo de sonido por medio de la voz.

Para controlar un equipo de sonido por medio de la voz, previamente se debe reducir de alguna manera el sonido que emite el equipo respecto de la orden ejecutada por el usuario del sistema para que el algoritmo de reconocimiento de palabras aisladas pueda funcionar correctamente, pues de lo contrario el sistema tendría una eficiencia baja dependiendo de la amplitud (volumen) del sonido que emita el equipo.

El filtro adaptivo es la interfase entre la voz de la persona y el sistema de reconocimiento de voz. Su función es eliminar ó reducir el ruido que se mezcla con la orden que da el usuario cuando esta es captada por el micrófono.

Para el reconocimiento de las palabras aisladas se utilizan los modelos escondidos de Markov (HMM) que es el método mayormente usado en este campo y se basan en un entrenamiento previo de las órdenes a utilizar que luego son comparadas con la orden que se pronuncie siendo el resultado la palabra más probable que será determinada por el algoritmo de Viterbi.

Por último se implementa un pequeño circuito de rayos infrarrojos que servirá de interfase entre la tarjeta DSP y el equipo de sonido.

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título : **F**iltro adaptivo LMS y su aplicación en el reconocimiento de palabras aisladas para el control de un equipo de sonido por medio de la voz

ÍNDICE

A mi madre Ena, quien siempre me apoyó en este largo camino.

A mi padre Ghattás que seguro estaría muy orgulloso.

A mi hermano Giorgio.

Al Ing. Andrés Flores que confió en mi capacidad para lograr este proyecto.

Al Dr. Felipe Uriarte quien me ayudó en la parte gramatical.

A todos mis amigos de la universidad, del barrio y a todos los que están en el extranjero.

Gracias, Giovani Simón Bendezu.

ÍNDICE

INTRODUCCIÓN

Día a día surgen nuevas ideas acerca de como lograr el control de algunos equipos o máquinas mediante la voz, pero también surgen nuevos inconvenientes, siendo uno de ellos dar órdenes a un sistema que emita algún tipo de sonido (por ejemplo voz, música, ruido en general). El caso escogido es controlar un equipo de sonido mediante órdenes verbales teniendo en cuenta que será una alternativa ó complemento frente al control remoto que es el aparato que se ha utilizado en los últimos años. El desafío es cómo dar órdenes así el volumen del equipo esté elevado y que las órdenes sean reconocidas como si el ruido no existiera, por ejemplo en los teléfonos celulares que poseen el sistema *'marcado por voz'* cuando se graba un nombre, si en ese momento hay un ruido como el claxon de un automóvil ó el sonido del motor de un bus, la voz va a estar superpuesta por este ruido y para que el nombre pueda ser identificado cuando se solicite, el bus debería pasar por ahí nuevamente, porque el algoritmo no va a reconocer lo que se dijo. De esta manera el micrófono por donde se capta la voz, recibe la señal de ruido (sonido del equipo) y para que no se mezcle con la orden se debe reducir de alguna manera.

El filtro adaptivo LMS tiene la capacidad de reducir sonidos o de separarlos si es que se tiene acceso a la señal de ruido (la referencia), y haciendo uso de ella es que se va a lograr la separación para que las órdenes que se dicten sean reconocidas sin mayor problema y con gran eficiencia como cuando no hay ruido. Además cabe destacar que el ruido debe ser focal, es decir que la señal se pueda obtener desde el emisor como en el caso del equipo de sonido que permanece en un determinado lugar. Generalmente los estudios realizados en la cancelación activa de ruido eliminan el ruido en el campo

acústico, es decir para silenciar ruidos, cosa que no se desea en este caso pues lo que se oiga por los parlantes se desea escuchar, es decir la eliminación se debe hacer en el campo eléctrico sin interferir con el normal funcionamiento del equipo.

Una vez filtrado el ruido sobre la voz, se procede a hacer un estudio estadístico de entre todo el grupo de órdenes de la base de datos generada después de hacer un entrenamiento previo del sistema utilizando un algoritmo basado en los modelos ocultos de Markov que luego con la intervención del algoritmo de Viterbi se selecciona y se compara la orden más probable del grupo respecto de la orden dicha.

Por último se envía una secuencia de bits que generan una señal infrarroja mediante un pequeño circuito de transmisión para poder actuar sobre el equipo de sonido haciendo cumplir la orden del usuario.

A continuación se hace una breve descripción de los capítulos de esta investigación.

El **capítulo 1** es un estudio del problema que presenta la eficiencia de un algoritmo de reconocimiento ante la presencia de ruido y a su posible solución, haciendo una prueba previa utilizando *Matlab*, hallando una gráfica (Eficiencia vs. Sonido del equipo).

El **capítulo 2** desarrolla los algoritmos y procesos a los cuales se somete la señal de voz para su reconocimiento ante la presencia de ruido (sonido del equipo) y las características de las señales infrarrojas de las funciones del control remoto.

En el **capítulo 3** se describen los procedimientos para la implementación del sistema de reconocimiento en la tarjeta de *Texas Instruments* y de las interfases de entrada y salida

En el **capítulo 4** se presentan los resultados de las pruebas en Matlab y en la tarjeta DSP y por último en los **capítulos 5** y **6** se redactan las conclusiones, observaciones y recomendaciones.

1. ANÁLISIS DEL PROBLEMA

1.1 FUNCIONAMIENTO DEL SISTEMA

El sistema de control del equipo de sonido *Panasonic SC-AK45* por medio de la voz debe funcionar de la siguiente manera:

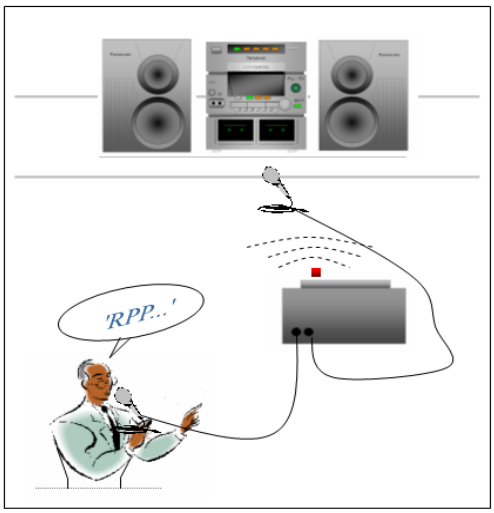


FIGURA 1.1 Funcionamiento del sistema de control del equipo por medio de la voz.

1.- El usuario debe pronunciar una palabra ó frase de la lista que contiene todas las posibles funciones que se han implementado.

2 - El sistema filtra el sonido del equipo (ruido) en la señal de voz.

3.- Se reconoce la orden emitida por el usuario de la lista que posee el sistema.

4.- Se envía la señal infrarroja que contiene la(s) trama(s) que corresponde(n) a la orden pronunciada por el usuario.

5.- Algunas de las funciones requieren nuevamente la presencia de voz para indicar el fin de su ejecución. Por ejemplo para el cambio sucesivo de emisoras de FM se debe pronunciar la orden *'buscar'*, con lo cual empezará a cambiar el dial cada 3 segundos, y para detenerlo se debe pronunciar cualquier otra expresión.

1.2 REQUERIMIENTOS DEL SISTEMA

El sistema debe cumplir los siguientes requerimientos:

- sistema independiente de la PC
- filtrado del ruido en tiempo real
- reconocimiento de la voz en tiempo real
- eficiencia del sistema mayor a 95% con la mitad del volumen y a 2.5m.

1.3 EFICIENCIA DE UN ALGORITMO DE RECONOCIMIENTO DE VOZ

En el estudio del reconocimiento de voz la variable más importante es sin duda la eficiencia, que es la cantidad porcentual que determina que tan bueno es el algoritmo empleado, es decir mide la cantidad de aciertos y de errores durante un ensayo de determinada cantidad de palabras. Matemáticamente, la eficiencia de un algoritmo es la cantidad de aciertos sobre la cantidad de veces que se evalúa el sistema y se muestra generalmente en porcentaje.

$$\text{Eficiencia} = \frac{\text{Número de palabras acertadas}}{\text{Número de palabras pronunciadas}} \times 100\%$$

Se dice que un algoritmo es de buena eficiencia cuando ésta es superior a 95%, es decir máximo 1 error en 20 pruebas, y se afirma que es de baja eficiencia cuando es menor a 85%, o sea más de 3 errores en 20 pruebas.

FIGURA 1.2 Niveles de eficiencia de los algoritmos

La eficiencia depende de muchas variables tales como: el tipo de algoritmo, cantidad de palabras a comparar, si hay ruido durante los ensayos, si se pronuncian las palabras muy lentas o muy rápidas, qué persona esté haciendo la prueba (varón, mujer o niño).

Algunos sistemas utilizan dos o más algoritmos a la vez para obtener una menor cantidad de errores, es decir si todos los algoritmos empleados obtienen el mismo resultado sin duda que esa es la respuesta, y cuando sean resultados diferentes, se considera el de mayor cantidad de aciertos que es el más probable. Cuando los resultados varían totalmente no se puede certificar qué palabra se dijo y además hay que tener en cuenta que el tiempo empleado en los cálculos es mucho mayor lo cual no es favorable. Para simplificar las cosas, en esta investigación sólo se emplea la voz de una persona y un solo algoritmo de reconocimiento, debido a que el objetivo es tener un modelo sencillo de reconocimiento de palabras aisladas que sirva para hacer comparaciones entre la eficiencia que se obtenga ante la presencia de ruido y aquella cuando no lo haya. Para esto, se ha elaborado un algoritmo de reconocimiento en *Matlab* usando el método HMM (‘modelos ocultos de Markov’) y se ha hecho un ensayo utilizando doce frases (estaciones de radio de FM) entrenadas con la voz de una sola persona y con 10 repeticiones por frase:

TABLA 1.1 Secuencia de la lista de 12 frases en 10 repeticiones

1	TELESTEREO	7	MIRAFLORES
2	RPP	8	DOBLE9
3	OK	9	STEREO100
4	STUDIO92	10	PANAMERICANA
5	RADIO AMERICA	11	RADIOMAR
6	ZETA	12	PLANETA

La figura 1.3 muestra los resultados del ensayo de aciertos del algoritmo de reconocimiento de palabras aisladas, empleado en el programa de prueba elaborado en *Matlab*. **Cada barra corresponde a cada una de las 12 frases dichas según la lista que se elaboró en la página 5, la altura representa la cantidad de veces que se dijo cada frase:**

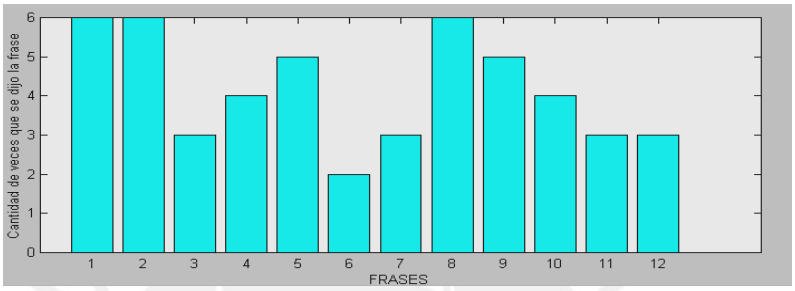


FIGURA 1.3 Resultados del algoritmo de reconocimiento.

Esta prueba se realizó sin presencia de ruido y a 5cm entre el micrófono y la persona. En las 50 repeticiones del algoritmo no ha habido error, esto quiere decir que el algoritmo tiene una eficiencia del 99.9% (por no decir 100%), cuando se comparan 12 palabras o frases y con un entrenamiento a una sola voz

1.4 ENERGÍA DE UNA SEÑAL DE AUDIO

La energía de una señal eléctrica es igual al cuadrado de su voltaje y como este es proporcional a la amplitud, se puede expresar lo siguiente:

$$\text{Energía} = k \times \sum_{i=1}^{t} \text{Amplitud}_i^{\,2}$$

La energía de la señal de voz capturada por un micrófono depende de la sensibilidad del transductor y la distancia que hay entre la persona y el micrófono, debido a que a mayor distancia, menor es la amplitud de la voz, lo cual es un inconveniente, porque la energía es cada vez menor y por lo tanto la eficiencia del algoritmo también lo será.

La figura 1.4 muestra dos gráficos, siendo el primero la señal de voz de la frase 'studio92' la cual fue grabada a una frecuencia de 8kHz durante 3 segundos y en 'silencio' (el mínimo ruido). El segundo gráfico es la energía que posee dicha señal y es proporcional al valor 0.0012 (se asume un valor referencial de k=1) que es el resultado de sumar los cuadrados de cada valor de la amplitud desde la muestra 1 hasta la muestra 24000, además se puede observar que la energía de la voz se concentra entre las muestras 1845 y 11300, donde los picos son las vocales de la frase, debido a que es ahí en donde se concentra la mayor energía cuando se habla.

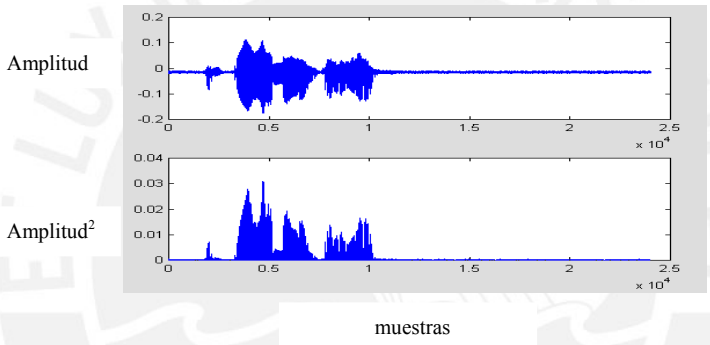


FIGURA 1.4 Señal de voz y de energía de la frase ('studio92')

Para tener una idea clara de cómo varía la energía de la señal del sonido de los parlantes que captura el micrófono por donde se dan las órdenes, se ha realizado una prueba en la cual se graban los primeros 30 segundos de una misma señal (canción 'Caraluna' en el disco compacto del grupo 'Bacilos') a 18 diferentes volúmenes y a una distancia de 2.5m entre los parlantes y el micrófono. En la figura 1.5 se muestra la amplitud elevada al cuadrado de cada señal a medida que se sube el volumen del equipo. Es importante resaltar que hay una variación en el inicio de cada señal debido al inicio del proceso de captura e inicio de reproducción de la música que produce un pequeño error que afecta poco al resultado final.

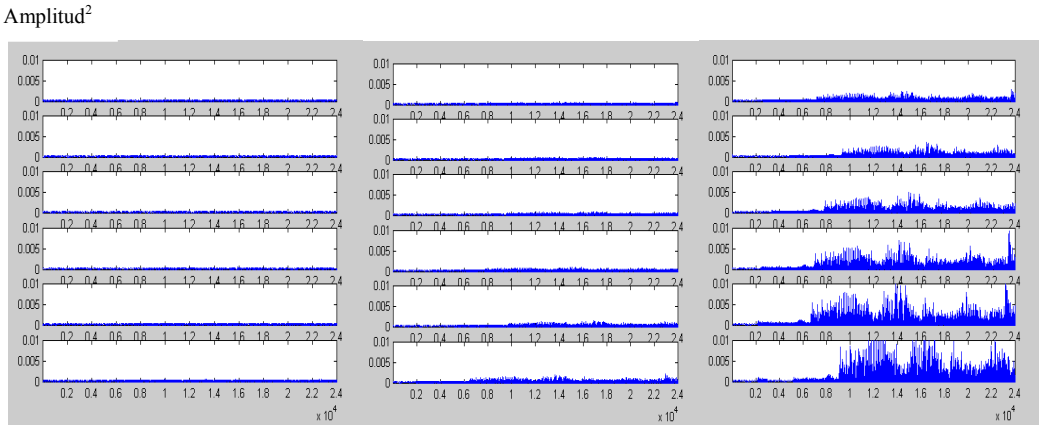


FIGURA 1.5 Amplitud al cuadrado de la señal de sonido a medida que se sube el volumen del equipo.

La tabla 1.2 muestra los valores de energía obtenidos con lo cual se ha elaborado el gráfico de la figura 1.6 en donde aparece una curva que resume lo que está sucediendo, es decir el aumento de la energía de la señal de sonido a medida que se sube el volumen en el equipo. La eficiencia caería de golpe apenas se empiece a distorsionar la orden.

TABLA 1.2 Relación de valores de la energía del sonido del equipo a diferentes volúmenes

	∑Amplitud²		∑Amplitud²		∑Amplitud²
1	2.2743	7	2.2974	13	3.0295
2	2.2693	8	2.3244	14	3.4869
3	2.2657	9	2.3696	15	4.1917
4	2.2712	10	2.4401	16	5.3550
5	2.2780	11	2.5694	17	7.0673
6	2.2908	12	2.7525	18	9.7014

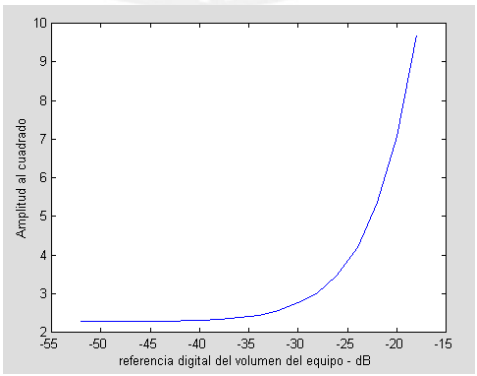


FIGURA. 1.6 Variación de la energía a diferentes volúmenes en el equipo.

Lo que se desea es reducir esta curva, es decir con la ayuda del filtro la energía de la señal del equipo (ruido) tendrá que ser menor. En el capítulo 4 se hace la misma prueba utilizando el filtro y se hace una comparación para determinar en cuánto se ha logrado reducir esta energía.

1.5 VARIACIÓN DE LA EFICIENCIA ANTE LA PRESENCIA DE RUIDO

Si dos personas hablan en la calle y durante el diálogo pasa un bus por allí, la persona que esté escuchando a la otra tiene que hacer un esfuerzo cada vez mayor a medida que el bus se acerca, porque la amplitud de este ruido es cada vez mayor, es decir a mayor ruido, menor el reconocimiento y por lo tanto menor eficiencia. Algo similar ocurre con el volumen del equipo de sonido: a mayor volumen menor eficiencia; esto se debe a que la señal de la voz a reconocer es superpuesta con la señal de ruido tal como se muestra en la figura 1.7, donde se aprecian cuatro señales: la primera es música emitida por los parlantes del equipo de sonido, la segunda es la voz de una persona pronunciando el nombre de una emisora de FM (‘Studio92’), la tercera es la voz superpuesta por la música con bajo volumen, y la cuarta con volumen alto

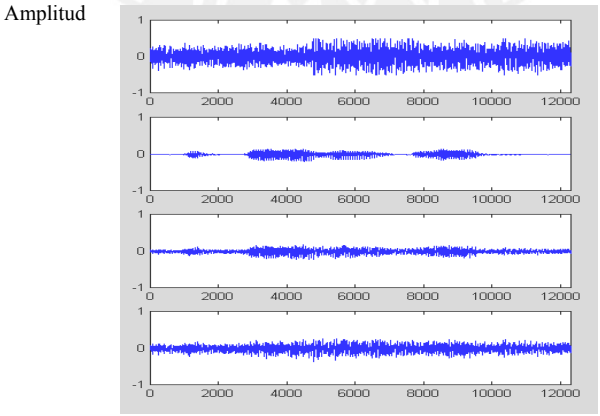


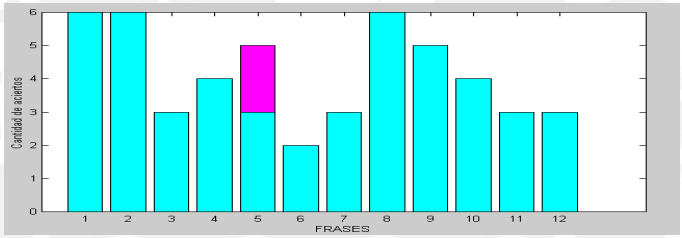
FIGURA 1.7. Superposición del sonido del equipo sobre la señal de voz ‘Studio92’

Este es el centro del problema de esta investigación, lograr una señal similar a la segunda partiendo de la primera y la tercera ó la primera y la cuarta.

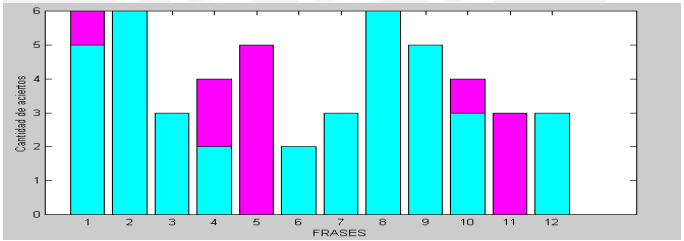
En resumen, si la energía de la señal de ruido es mayor, la voz es menos reconocible. Para apreciarlo mejor, se ha hecho la misma prueba de eficiencia, pero ahora con presencia de ruido a diferentes niveles que han sido medidos con un **sonómetro** *('Sound level meter'* de *'Radio Shack'* – especificaciones en el **Anexo K**):

La altura de cada barra es la cantidad de veces que se dice cada frase donde el color celeste representa aciertos y el color violeta representa errores.

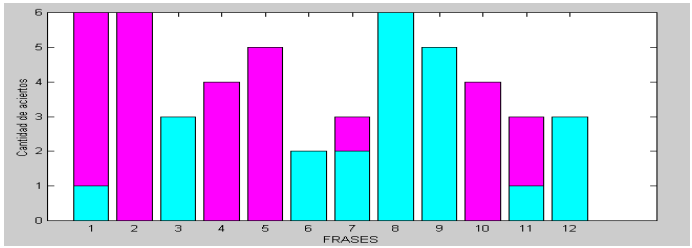
a) 57 – 63dBC en el sonómetro. Eficiencia: **96%.**



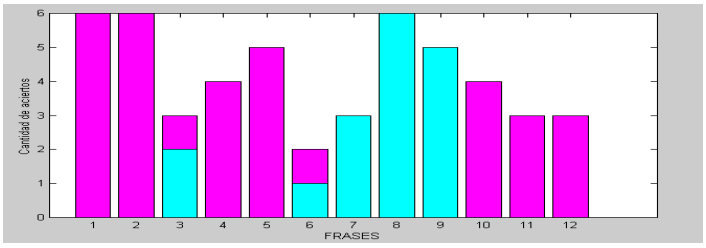
b) 64 – 68dBC en el sonómetro. Eficiencia: **76%.**



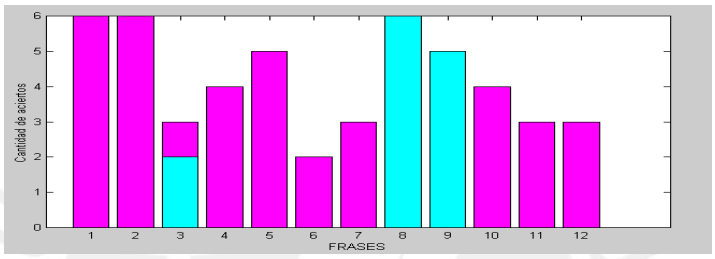
c) 65 – 74dBC en el sonómetro. Eficiencia: **46%.**



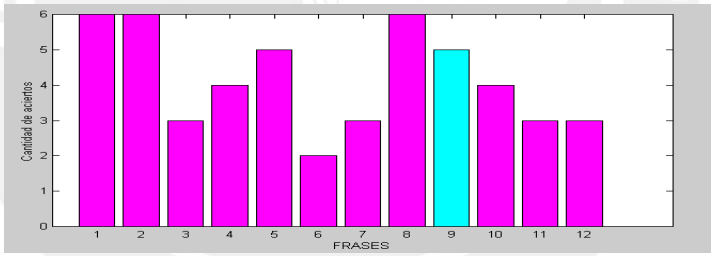
d) 77 – 81dBC en el sonómetro. Eficiencia: **34%.**



e) 82 – 87dBC en el sonómetro. Eficiencia: **26%.**



f) 87 – 91dBC en el sonómetro. Eficiencia: **10%.**



Se asume que con mayor volumen, la eficiencia alcanza cero porque siempre un algoritmo tiende a estancarse en ciertas frases (8 y 9) como se demuestra en la prueba.

El resultado de las pruebas se resume en el siguiente gráfico de la figura 1.8 que refleja la relación entre la eficiencia versus el sonido del equipo (en dBC).

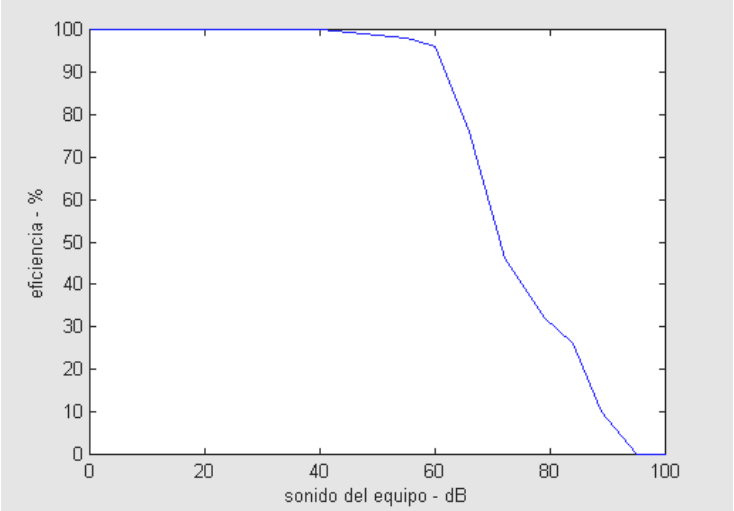


FIGURA. 1.8 Eficiencia vs. Sonido del equipo (dBC).

De acuerdo al resultado el algoritmo es bueno (>95%) solamente cuando el ruido es menor a 60dBC y es regular (>85%) solo hasta con 63.3dBC (ruido relativamente bajo).

1.6 USO DE FILTROS ADAPTIVOS PARA CANCELAR EL RUIDO

Para separar dos sonidos que tienen un ancho de banda grande y son variantes en el tiempo como es el caso de la voz y la música, no se pueden utilizar filtros comunes (coeficientes invariantes en el tiempo), lo que se debe hacer es corregir constantemente los coeficientes de acuerdo a cómo vayan cambiando las señales, es decir se deben 'adaptar'.

El filtro LMS (*Least Mean Square*) es adaptivo porque va cambiando con el tiempo. Este tipo de filtro es utilizado en diferentes aplicaciones tales como la cancelación activa de ruido que se basa en eliminar un sonido emitiendo su señal inversa a través de altavoces; también se aplica en la identificación de funciones de transferencia de una planta haciendo que sus coeficientes converjan a los de ella. En esta investigación se

utiliza para separar dos sonidos que están mezclados o superpuestos siendo uno de ellos el ruido y el otro, el sonido que se desea limpiar. Para lograr este objetivo se necesitan dos señales:

- La referencia que es el ruido en sí, en este caso el sonido del equipo.
- La señal contaminada que es la señal que está superpuesta por el ruido, que viene a ser la voz de la persona mezclada con el ruido. Cabe destacar que este ruido no es igual al ruido de referencia pues ha sufrido alteraciones debido a la distancia entre los sensores (micrófonos de referencia y primario).

Se ha elaborado un programa de prueba en *Matlab* **(anexo C)**, el cual sirve para constatar que realmente se pueden separar la voz de algún ruido siempre y cuando se tenga acceso a la referencia.

Los pasos son los siguientes:

- Frecuencia de muestreo de 8kHz y 16 bits de resolución.
- Luego se graba una frase 'Studio92' por 2 segundos (16000 muestras).
- A continuación se graba música proveniente del equipo de sonido el mismo tiempo (2 segundos).
- Esta música se altera usando una función de transferencia con coeficientes polinómicos 0.5, 0.4, 0.1 y con una atenuación a su tercera parte simulando la transformación que sufre el sonido desde los parlantes hacia el micrófono.
- Esta señal de ruido modificada se superpone a la señal de voz obteniendo la señal a filtrar y la referencia que es el ruido sin modificar.
- Se elabora el algoritmo LMS y se procede al filtrado.
- Por último se grafican los resultados.

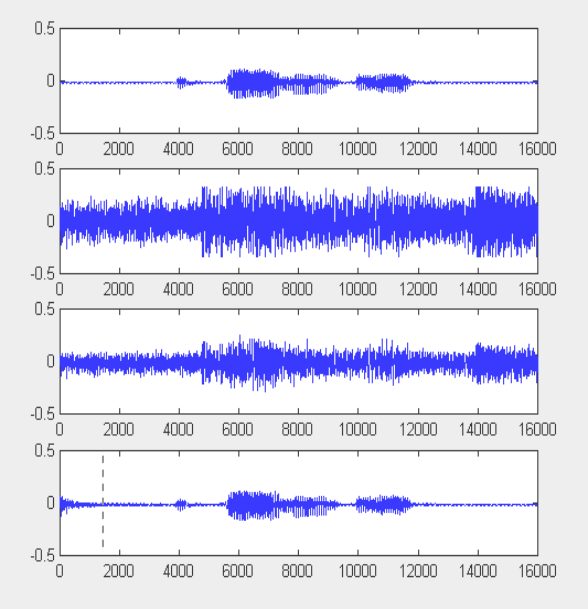


FIGURA 1.8 Prueba de ruido utilizando un filtro LMS elaborado en Matlab.

En la figura 1.8 la primera señal es la voz sin ruido ‘studio92’; la segunda es el ruido, es decir la referencia; la tercera es la voz superpuesta por el ruido luego de una función de transferencia ‘desconocida’ 0.5/3, 0.4/3, 0.1/3 y la cuarta es la voz filtrada, es decir sin ruido. Esta señal no es igual a la primera pero es similar y se debe resaltar que al principio hay un tiempo en que los coeficientes del filtro se van acomodando, es decir el tiempo de convergencia de los coeficientes, y por tal motivo es que al principio la señal filtrada es diferente a la señal que se desea (la primera). En la figura se ha colocado una marca para apreciar hasta dónde sucede la convergencia. En la sección 2.2.4 se muestra la variación de los coeficientes del filtro de esta demostración. Y aunque este caso es ideal, se demuestra que es posible reducir el ruido para poder utilizar la señal de voz en cualquier sistema que lo requiera.

1.7 IDENTIFICAR EL NIVEL DE RUIDO DONDE LA EFICIENCIA SEA MAYOR A 95% AL USAR EL FILTRO.

Se desea una eficiencia mayor a 95%, es decir que el sistema logre que el algoritmo alcance una buena calidad para poder aplicarlo en el control del equipo; pero todo tiene un límite tal como se demostró en sección 1.5 cuando se obtuvo una curva 'Eficiencia vs. Nivel de sonido del equipo' y de la cual se puede desprender que el nivel de ruido de 63dBC es donde la eficiencia empieza su descenso por debajo de 95%, entonces lo que se desea es que este nivel sea mayor con el uso del filtro, y se espera que este límite sea por lo menos de 75dBC que es un volumen de ruido regular para el oído humano.

2. DESCRIPCIÓN DEL SISTEMA

2.1 DIAGRAMA DE BLOQUES DEL SISTEMA

El sistema de control del equipo de sonido por medio de la voz se divide en tres etapas:

- Filtrado adaptivo del ruido (interfase de entrada).
- Reconocimiento de la orden (sistema).
- Transmisión de la señal infrarroja (interfase de salida).

La etapa de filtrado adaptivo es la interfase de entrada y ha sido desarrollada en la tarjeta de evaluación DSP56002EVM de *Motorola*, la cual posee un conversor análogo-digital estéreo que permite la captura y digitalización de dos señales de audio a la vez: la primera de ellas es la voz que está superpuesta por el ruido y la segunda el sonido del equipo, es decir el ruido de referencia. La salida del filtro es una señal similar a la voz que es la que ingresará a la segunda etapa en donde se reconoce la orden de la persona. Esta etapa es la principal y ha sido desarrollada en la tarjeta TMS320C6711DSK de *Texas Instruments* la cual contiene un conversor (‘*codec*’) mono que posee dos canales: uno de voz (mono) y otro de datos. El sistema reconoce la orden y envía el tren de pulsos por un puerto de salida al circuito IR, que es la interfase entre el sistema y el equipo de sonido.

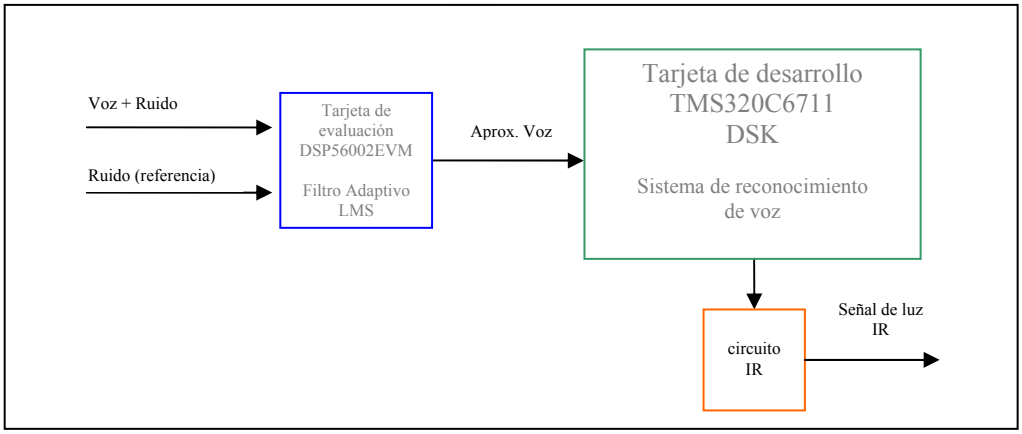


FIGURA 2.1 Diagrama de bloques del sistema

2.2 FILTRO ADAPTIVO LMS

El filtro adaptivo LMS (‘*least mean square’*) es un filtro tipo fir (respuesta impulsiva finita) y es utilizado para reducir la amplitud del sonido del equipo

2.2.1 FILTRO ADAPTIVO

Es un filtro que varía sus coeficientes a través del tiempo para cumplir con ciertos requerimientos. Por medio de este filtro se pretende separar la voz del usuario y el ruido producido por el equipo de sonido, teniendo como referencia el mismo ruido. Es decir se modifica la referencia intentando hacerla similar al ruido que está superpuesto a la voz, para poder restarlo y lograr una señal de salida muy similar a la voz del usuario.

La idea básica de todo filtro adaptivo es ajustar continuamente los parámetros bajo control usando un algoritmo adaptivo haciendo uso tanto de la información de entrada como la de salida, mediante la realimentación del error para poder así modificar los coeficientes con el propósito de converger en un punto donde el error sea mínimo, y siendo estos coeficientes los que van a modificar la información de entrada para obtener el resultado deseado.

Las partes básicas de un filtro adaptivo son las siguientes:

a) Filtro transversal, que son los coeficientes que modifican la señal de entrada.

b) Algoritmo adaptivo, que es utilizado para modificar los coeficientes del filtro.

c) El error en la salida.

La figura 2.2 muestra el diagrama de bloques del filtro en donde:

x(n): es la señal de entrada,
y(n): es la señal de salida,
d(n): es la señal deseada y
e(n): es el error.

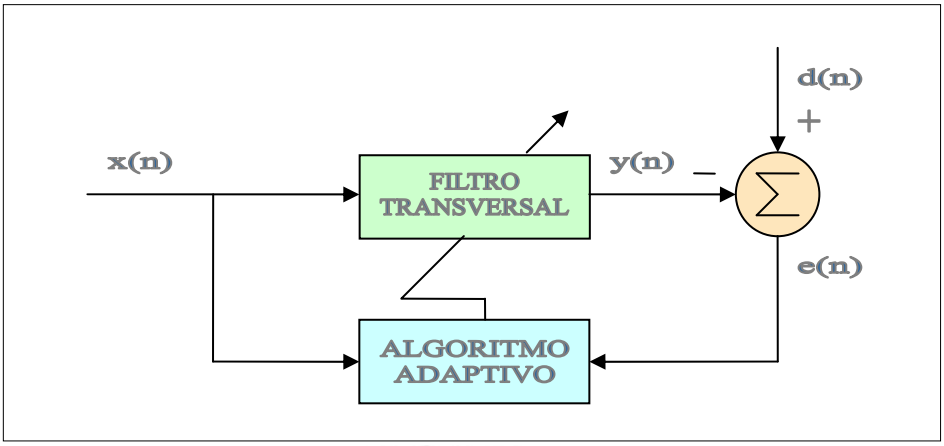


FIGURA 2.2 Diagrama de bloques de un filtro adaptivo [1].

En esta investigación se va a separar la voz del usuario y el sonido del equipo, entonces se redefine de la siguiente manera:

x(n): referencia ------ ruido en el secundario

y(n): referencia filtrada ------ aproximadamente ruido en el primario

d(n): señal + ruido ------ voz + ruido en el primario

e(n): señal‘ ------ aproximadamente voz (e = d - y)

Donde primario y secundario son las rutas de los dos micrófonos que se utilizan.

Primario: micrófono cerca de del usuario del sistema.

Secundario: micrófono cerca de los parlantes del equipo.

El filtro se encarga de producir el ruido primario teniendo como referencia el ruido secundario en la entrada y el error que es realimentado.

2.2.2 FILTRO TRANSVERSAL

Este tipo de filtro se denomina así porque tiene la característica de actuar sobre la señal de entrada con una serie de continuos retardos que provocan una modificación desde la

muestra actual hacia atrás de manera secuencial (respecto del número de orden del filtro). Los coeficientes del filtro modifican una a una las muestras y luego la salida viene a ser la suma de todas las multiplicaciones hechas durante la operación.

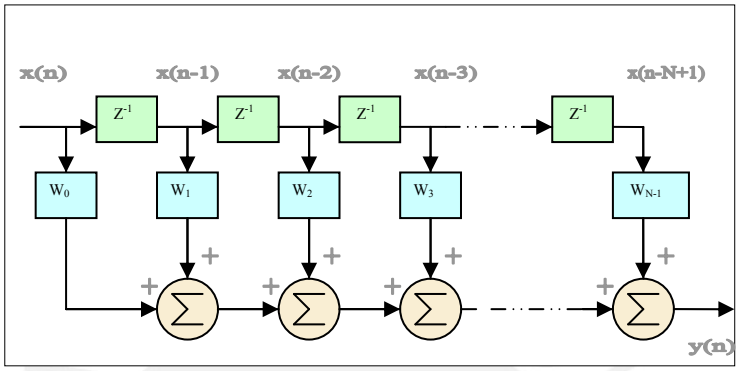


FIGURA 2.3 Filtro transversal

$$y(n) = \sum_{k=1}^{N-1} w_k \, . \, x(n-k)$$

x(k) : señal de entrada
w_k : coeficientes del filtro
y(k) : salida del filtro
n : muestra actual
N : número de orden del filtro

FIGURA 2.4 Representación matemática del filtro transversal

2.2.3 ALGORITMO ADAPTIVO LMS

El algoritmo adaptivo es la lógica del filtro, es la que se encarga de actualizar los valores de los coeficientes teniendo como base la señal de entrada y la realimentación de la señal de error. Existen muchos algoritmos adaptivos que se aplican en la cancelación activa de ruido, unos de tipo *fir* y otros del tipo *iir*. Los filtros *fir* (*finite impulse response*) son mayormente usados pues son filtros más sencillos cuyo tiempo de proceso es mucho menor que los filtro de tipo *iir*.

Hay muchos tipos de filtros adaptivos, pero todos se basan en el diagrama general, y el más simple de todos y del cual se derivan muchos de ellos es el filtro adaptivo LMS que significa el menor cuadrado promedio (‘*least mean square*’).

La fuerza del filtro LMS reside básicamente sobre su inherente simplicidad y cálculos matemáticos. El proceso de filtrado adaptivo se puede describir mediante la siguiente fórmula:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu \mathbf{u}(n) e^{*}(n)$$

FIGURA 2.5 Algoritmo adaptivo [1].

El siguiente programa elaborado en *Matlab* muestra la función **`lms.m`.**

```
function  error = lms (ruido, d, u, N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                             %%
%%   Filtro adaptivo LMS - Aplicación          %%
%%                                             %%
%%   ruido: sonido del equipo (referencia)     %%
%%       d: voz de la persona con ruido        %%
%%   error: señal de voz filtrada              %%
%%       u: paso de descenso hacia error mínimo %%
%%       N: número de coeficientes             %%
%%                                             %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

L = length(x);
W = zeros(N,1);
ruido = [zeros(N-1,1); ruido];

i = 1;
while (i<L+1)
x = ruido(i+N-1:-1:i);
error(i)= d(i) - W'*x;
W=W+u*x*e(i);
i=i+1;
end;
```

La señal ‘d’ es la voz que está superpuesta por parte del ruido, y el `ruido` es la señal de referencia (sonido del equipo) dando como resultado el ‘error’ que es aproximadamente la señal de voz. ‘N’ es el número de coeficientes del filtro transversal, y ‘u’ es el tamaño del paso (gradiente) que dan los coeficientes cada espacio de tiempo hasta converger en el error mínimo.

2.2.4 CONVERGENCIA

Los coeficientes del filtro adaptivo se modifican constantemente hasta lograr que el error sea mínimo, es decir el algoritmo alcance la convergencia, dicho de otra manera que la entrada se aproxime a la respuesta deseada.

Para precisar lo que sucede, supongamos que el orden del filtro es N = 2, es decir los coeficientes son de la forma $w(n) = [\ w_1(n)\ \ w_2(n)\ ]$ y siendo ξ el error donde ξ_{min} es el mínimo.

Entonces el sistema se puede graficar en un espacio de tres dimensiones como se puede apreciar en el gráfico 2.6. Cuando empieza el algoritmo, el error es grande y a medida que el tiempo transcurre, los coeficientes van descendiendo por la curva a una velocidad relativa al valor de paso μ hasta llegar al error mínimo ξ_{min}. Pero se debe tener en cuenta que este proceso no sucede de una manera trunca, sino similar a una pelota deslizándose por la pared interna: cuando llega al fondo, pasa de largo y vuelve a subir pero con menor impulso y luego vuelve a bajar y así sucesivamente hasta detenerse en el fondo. El tiempo que se demore en detenerse en el fondo es el tiempo de convergencia y w_0 es el punto de convergencia.

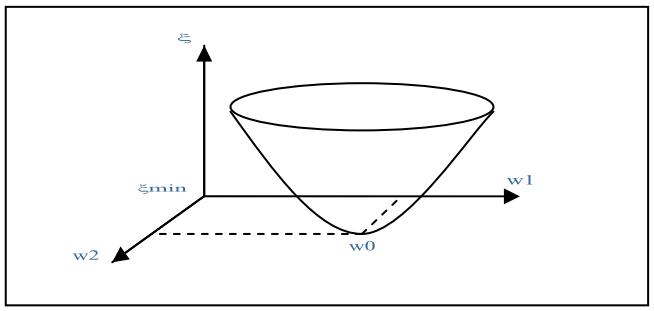


FIGURA 2.6 Convergencia y error mínimo [1].

La figura 2.7 muestra la convergencia de los coeficientes en el ejemplo de filtrado de la sección 1.5. Aquí se muestran 2 figuras: la primera es la señal filtrada y la segunda son los coeficientes del vector W que convergen a ciertos valores que vienen a ser los valores de la función de transferencia que modifica el ruido desde el punto de emisión hasta su captura 0.5, 0.4, 0.1 y atenuación a su tercera parte es decir 1.6667, 0.1333, 0.3333. Cuando los coeficientes mantienen estos valores, el error es mínimo, y si hay presencia de voz, esos valores varían pero con proximidad a la convergencia.. Dicho de otra manera, es como si de desplazaran en el fondo de la concavidad de la figura 2.6.

La línea punteada indica que el filtro ha demorado aproximadamente 1800 muestras para alcanzar la convergencia, es decir 0.225 segundos. Si la planta es invariable, se puede comenzar con valores iniciales en la convergencia luego de una prueba, con lo cual el tiempo de convergencia sería nulo. En el caso del micrófono y el parlante, la planta varía cuando uno de ellos se desplaza respecto del otro. Por ese motivo se debe colocar un valor inicial de cero para todos los coeficientes.

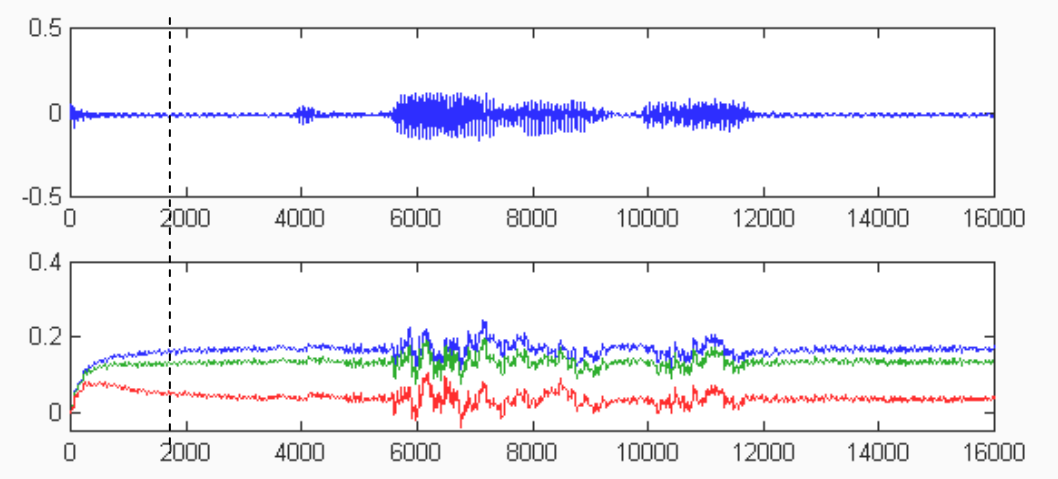


FIGURA 2.7 Convergencia de los coeficientes del filtro (W)

2.3 PRE-PROCESAMIENTO DE LA SEÑAL DE VOZ

La señal de voz debe someterse a un pre-procesamiento que permita obtener las características espectrales de la onda en forma vectorial para ser utilizadas en el proceso de reconocimiento. Este pre-procesamiento de la voz consta de cinco partes: ventanas de voz, filtro de pre-énfasis, ventana de hamming, transformada de fourier (FFT) y banco de filtros triangulares, y coeficientes cepstrales.

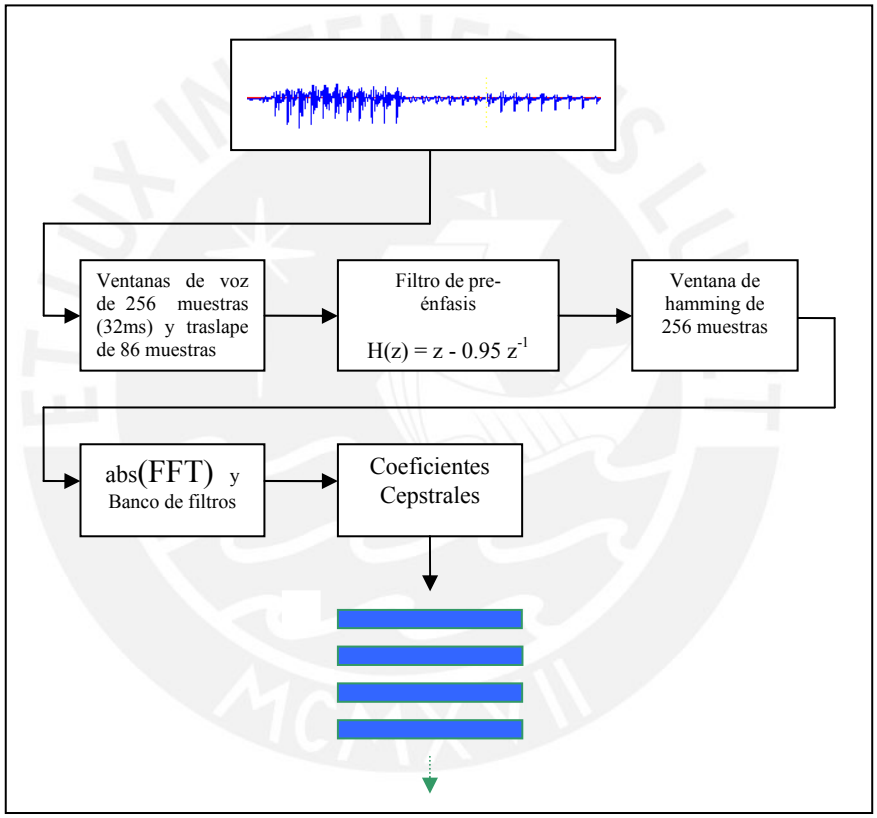


FIGURA 2.8 Pre-procesamiento de la señal de voz.

2.3.1 VENTANAS DE VOZ Y DESPLAZAMIENTO

La señal de voz no se puede analizar toda a la vez por lo que se debe seccionar en segmentos de cierto número de muestras y de un adecuado desplazamiento que permita concentrar la información de cada uno de ellos en el centro sin perder información. En

los procesos de voz se utiliza generalmente ventanas de 120 a 300 muestras, no hay un número específico, pero para el desplazamiento se debe tomar 2/3 del tamaño de la ventana para que de esta manera cada muestra en los extremos pertenezca a 2 ventanas contiguas para evitar la discontinuidad. En esta investigación se ha optado por ventanas de 256 muestras y desplazamiento de 170 es decir 32ms y 21.3ms en tiempo. Cada ventana es una unidad de información de la cual se extraerán sus características espectrales para ser utilizadas en el proceso de reconocimiento.

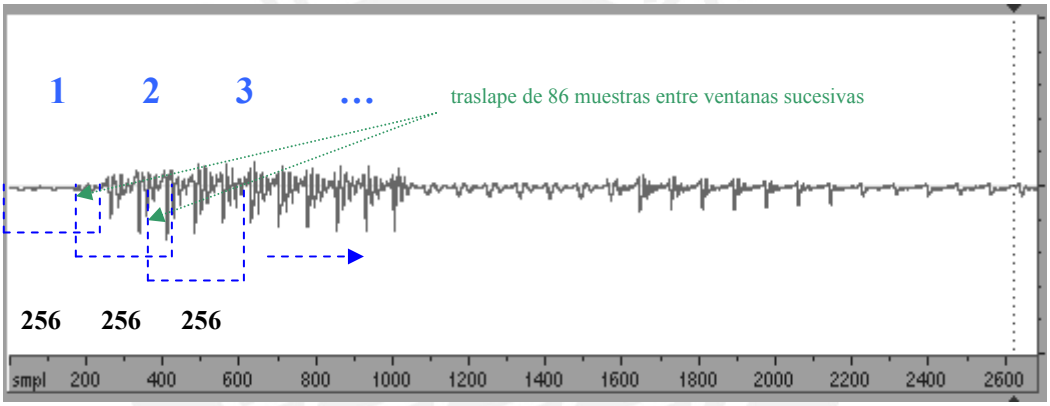


FIGURA 2.9 proceso de separación de la señal de voz en ventanas

2.3.2 FILTRO DE PRE-ÉNFASIS

En cada ventana se debe realzar las frecuencias altas debido a que la voz sufre atenuaciones durante el proceso del habla y para contrarrestar esta situación, se emplea un filtro de pre-énfasis según la siguiente expresión:

$$H(z)= 1 - 0.95\, z^{-1}$$

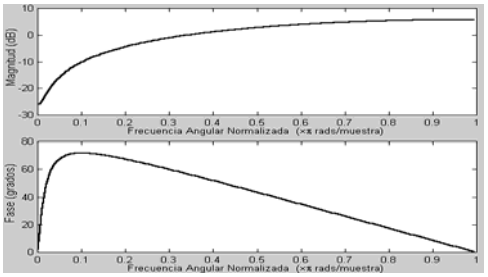


FIG 2.10. Filtro de pre-énfasis

La función **'preenfasis.m'** elaborada en *Matlab* muestra el algoritmo de pre-énfasis utilizado para modificar la señal de voz.

```
function vp = preenfasis(vent)

%  Filtro de pre-énfasis
%  vent : ventana de voz
%  vp   : ventana luego del filtro

vp(1) = vent(1);
for j = 2:length(vent)
  vp(j) = vent(j) - .95*vent(j-1);
end
```

2.3.3 VENTANA DE HAMMING

Emplear la ventana de hamming sirve para resaltar las muestras que están en el centro de la ventana, es decir su aplicación contrarresta el efecto de pérdida de continuidad que ocurre al separar la señal en ventanas. El método es sencillo, consiste en multiplicar cada muestra de la ventana de voz por la correspondiente de la ventana de hamming. En la figura 2.11 se muestra la gráfica de la ventana de hamming donde se puede ver que el centro mantiene la amplitud original y los extremos son atenuados. La función **'vent_hamming.m'** halla los valores de la ventana de hamming ingresando el tamaño de ventana deseado.

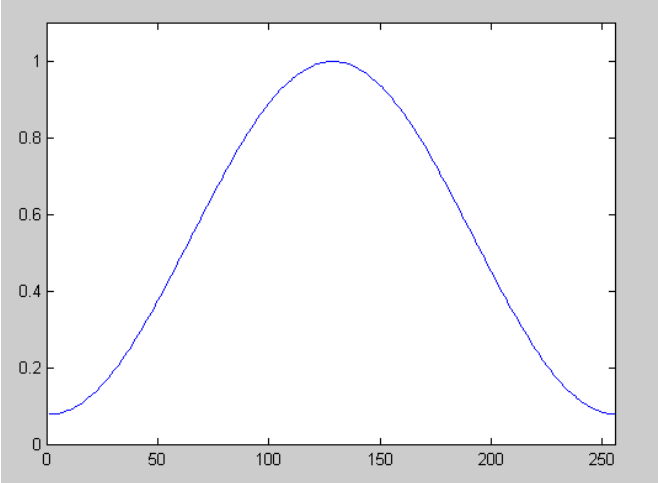


FIGURA. 2.11 Ventana de Hamming

```
function  vh = vent_hamming(N)

% Ventana de hamming
% N  : tamaño de la ventana
% vh : ventana de hamming

for j=0:N-1
   vh(j+1) = 0.54 - 0.46 * cos(2*pi*j/(N-1));
end
```

2.3.4 BANCO DE FILTROS Y COEFICIENTES CEPSTRALES

Una vez procesadas las ventanas de voz en el tiempo, el análisis pasa al campo de la frecuencia por lo que en cada ventana se debe aplicar la transformada de Fourier hallando su magnitud. Luego se utiliza un proceso de filtrado separando el espectro en bandas utilizando un banco de 20 filtros triangulares donde fi es la posición de la frecuencia inicial, fc es la posición de la central y fs la final:

TABLA 2.1 Posiciones de los filtros triangulares 'fi-fc-fs'

fi	1	3	5	7	9	11	13	15	17	19	23	27	31	35	40	46	53	61	70	81
fc	3	5	7	9	11	13	15	17	19	23	27	31	35	40	46	53	61	70	81	93
fs	5	7	9	11	13	15	17	19	23	27	31	35	40	46	53	61	70	81	93	108

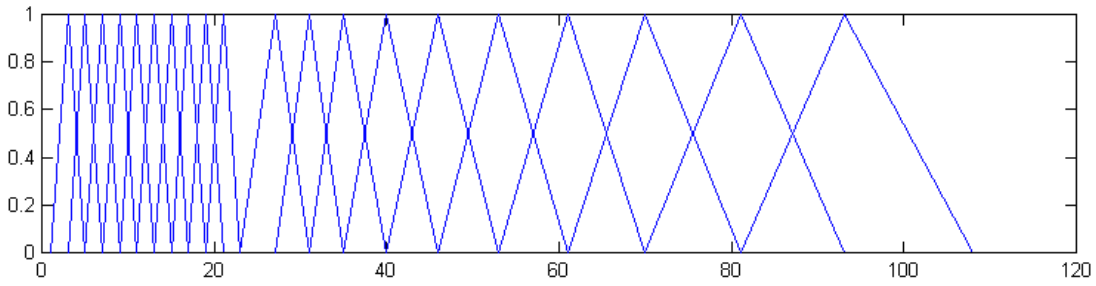


FIGURA 2.12 Filtros triangulares[2].

El número de muestras por ventana es de 256, por lo tanto la transformada de Fourier también lo será, pero el resultado es un espejo de 128 muestras a cada extremo en el campo de frecuencias, por lo que se ha escogido 20 filtros hasta la posición 108. ($108 \leq 256/2$ según Nyquist).

Cada filtro se aplica individualmente, multiplicando el valor de cada posición por su correspondiente en la transformada, luego estos valores se suman obteniendo un valor que representa la intensidad espectral de la señal en cada banda.

$$Y_i = \sum_{j=1}^{108} [\ \text{filtro triangular}_{i,j} \ \text{x} \ \text{FFT}(\text{ventana}_v)_j\ ] \quad , \quad i=1..20$$

Una vez hecho el análisis en frecuencia, se debe hallar las características del '*cepstrum*' que se define como el logaritmo natural del '*spectrum*' (frecuencia):

$$\textbf{\textit{Cepstrum = Ln (Spectrum)}}$$

$$Y_i = \text{Ln}\ (Y_{i\ \text{spectrum}})$$

Los coeficientes cepstrales se hallan utilizando un método parecido al de los filtros en el campo de la frecuencia, pero ahora se emplean 10 ventanas senoidales de diferentes longitudes de onda que se multiplicarán por los valores en el 'cepstrum'.

$$\text{Coef. Cepstrales} = \sum_{j=1}^{P} [\ \text{Ventana senoidal}_{j,i} \ x \ Y_i\ ] \quad , \quad i=1..20$$

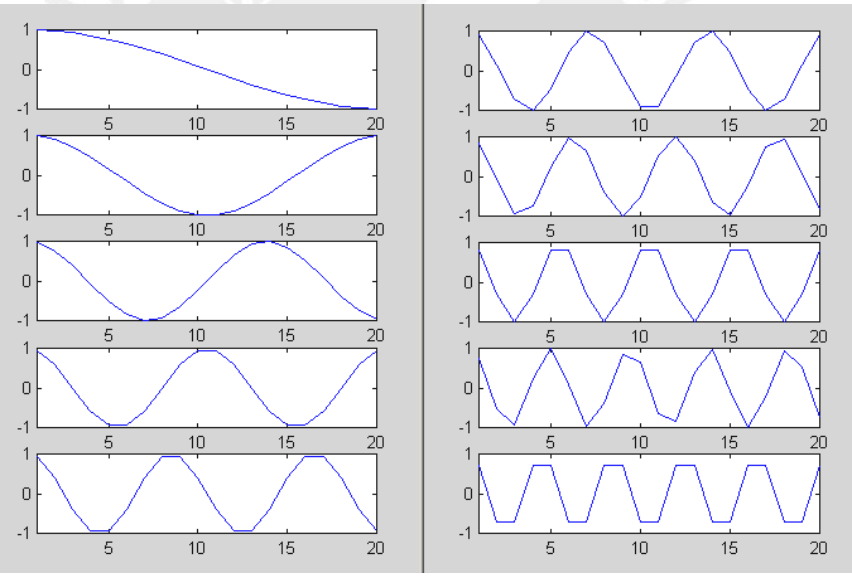


FIGURA 2.13 P ventanas senoidales utilizadas para hallar los coeficientes cepstrales
P es el orden del cepstrum, aquí P=10

2.3.5 DETECCIÓN DE LA SEÑAL DE VOZ

El proceso de detección se basa en un análisis de la energía que contiene un conjunto de muestras de voz, siguiendo un algoritmo sencillo que establece que hay voz si este valor es mayor a un valor umbral, que se determina experimentalmente. Si el resultado

es menor al valor umbral, no hay presencia de voz, y si es mayor el sistema asume que hay voz o una fuente de ruido.

El valor umbral depende de la ganancia del micrófono primario, debido a que la energía es función del voltaje, es decir la amplitud; entonces se deben hacer pruebas para determinar el valor final. En las pruebas en *matlab* se utiliza un valor umbral de 0.003.

2.4 MODELOS OCULTOS DE MARKOV (HMM)

2.4.1 MODELO A, B, π

Un modelo de Markov se basa en un sistema de N estados dependientes uno del otro mediante una función de probabilidad definida por una matriz de transición [2]:

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix}$$

donde a_{ij} es la probabilidad de pasar del estado i al estado j.

$q(t) = i \quad , \; i = 1, 2, 3, \ldots\ldots N$ es el estado en el tiempo t, siendo $t = 1, 2, 3, \ldots, T$

Para $t = 1$, la variable π determina que estado tiene mayor probabilidad de ser el primer estado. $P(q(1) = i) = \pi(i)$ probabilidad que el estado i sea el primer estado.

Una transición de un estado a otro se produce por cada unidad de tiempo y depende de una función de probabilidad. Cuando el sistema es de primer orden, la transición sólo depende del estado anterior.

$$P(q(t) = j \mid q(t-1) = i, q(t-2) = m, \ldots) = P(q(t) = j \mid q(t-1) = i) = a(i,j) \qquad [2]$$

Además la suma de las probabilidades de transición que actúan desde un estado hacia otro debe ser igual a 1:

$$\sum_{j=1}^{N} a_{ij} = 1 , \quad \text{para todo i} \qquad [2]$$

En procesos de voz, generalmente se utiliza el modelo de Markov con topología de izquierda a derecha que sólo permite la transición de un estado sobre sí mismo o a los dos estados siguientes, siendo estas probabilidades de igual valor, es decir 0.3333, porque cada estado tiene tres posibles caminos de transición excepto para las dos últimas debido a que sólo puede cambiar sobre si misma o a al siguiente donde la probabilidad es igual a 0.5 ó 1. Por ejemplo si el número de estados es N=5:

$$A = (a_{ij}) = \begin{bmatrix} 0.3333 & 0.3333 & 0.3333 & 0 & 0 \\ 0 & 0.3333 & 0.3333 & 0.3333 & 0 \\ 0 & 0 & 0.3333 & 0.3333 & 0.3333 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

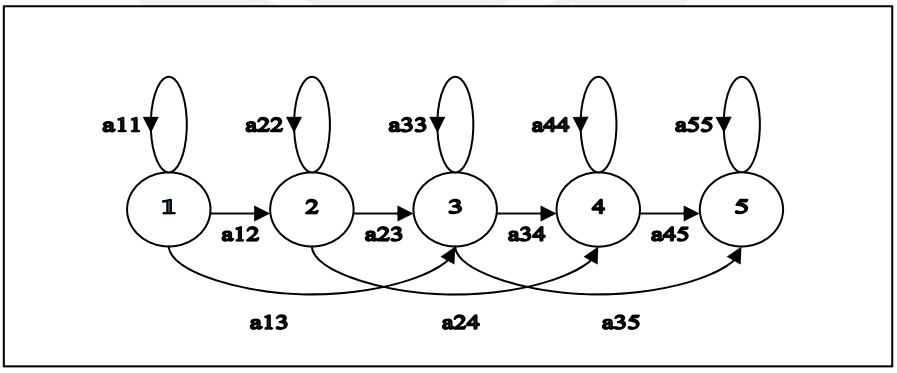


FIG 2.14 Modelo de Markov con topología 'Izquierda – Derecha'. [2]

Cuando un modelo de Markov se encuentra en un estado, éste genera una o más observaciones. En un "Modelo Oculto de Markov" (HMM) no se conoce la secuencia de estados, y durante la permanencia en cada estado, el sistema muestra un grupo de observaciones, pero no la información directa sobre el estado en el que se encuentra. En el caso del estudio de la voz, una observación es el vector característico de una ventana de voz.

La probabilidad de tener una observación k en el estado j se representa a través de otra matriz:

$$B = \{ b_j(k) \} \text{ donde } b_j(k) = P[o_t = v_k \mid q_t = j]$$

En conclusión, el procedimiento de los modelos escondidos de Markov posee los siguientes elementos:

N, número de estados
A, matriz de transición (probabilidad de pasar de un estado a otro).
B, distribución de probabilidad de observación.
π, vector de probabilidad inicial.

El modelo en conjunto se representa por: **$\lambda = (A, B, \pi)$**.

2.4.2 TRES PROBLEMAS BÁSICOS EN UN HMM

Dado un modelo $\lambda = (A, B, \pi)$ y una secuencia de observación $O = [o(1)\ o(2)\ ...\ o(T)]$ se presentan los siguientes problemas [2]:

1) ¿Cómo hallar $P(O \mid \lambda)$ la probabilidad que la secuencia de observación O halla sido producida por el modelo de manera eficiente?
2) ¿Cómo escoger la secuencia de estados más probable?
3) ¿Cómo ajustar el modelo λ para maximizar $P(O \mid \lambda)$?

En el caso de esta investigación se va a utilizar HMM en el *'reconocimiento de palabras aisladas'*, es decir de una lista de 'W' cantidad de palabras o frases se va a escoger la más probable, por lo tanto para cada palabra se diseñará una secuencia de N-estados que tendrán una secuencia de entrenamiento con una determinada cantidad de repeticiones por cada palabra dicha por una o más personas (problema 3). El significado físico de las secuencias de entrenamiento de las palabras viene a ser la conversión en secuencias de estados para cada una de ellas (problema 1) que servirán para el estudio que determine que observaciones son más probables para cada estado. Finalmente, una vez que se halla diseñado y optimizado el modelo de las palabras, ya se puede reconocer, es decir dada una secuencia de observación (palabra a reconocer), se debe mostrar qué palabra posee el mejor modelo de toda la lista, escogiendo la más probable del grupo (problema 2).

2.4.3 ENTRENAMIENTO DE PALABRAS AISLADAS

2.4.3.1 PROCESO DE ENTRENAMIENTO

El entrenamiento de las palabras que conforman la lista de posibles órdenes se basa en hallar un modelo **A y B (mu,sigma)** para cada una de ellas, teniendo como punto de partida una serie de repeticiones de las palabras dichas por una o varias personas. Primero se genera un modelo inicial A, B y π para cada palabra utilizando todas las repeticiones grabadas previamente y luego se va optimizando por medio de continuas iteraciones que generan el modelo final que será utilizado para el reconocimiento mediante el algoritmo de viterbi.

Considerar lo siguiente:

- El entrenamiento se efectúa palabra por palabra (lista de w palabras)
- Se crea un modelo inicial A y B.
- Se emplea una serie de iteraciones para optimizar A y B (mu y sigma) empleando *'forward & backward algorithm'*.
- Se almacena cada modelo y se prosigue con la palabra siguiente.

2.4.3.2 VALORES INICIALES DE MU Y SIGMA

La función **'hmm_ini.m'** (anexo E) genera un vector 'mu' y un vector 'sigma' para cada palabra para luego poder ser utilizados en el proceso iterativo siendo los valores iniciales en el proceso de entrenamiento para cada una de ellas.

Las variables utilizadas son las siguientes:

- K es el número de repeticiones, es decir el número de secuencias de voz.
- X es la matriz que contiene los vectores.
- SEG es la normalización de las repeticiones debido a que cada una contiene diferente número de vectores porque el tiempo en que se pronuncia la misma palabra es diferente y por este motivo se segmenta cada valor de 'v' en partes iguales hasta tener un segmento para cada estado. Por ejemplo si v = 26 (st(1)=1 y st(2)=27) y el número de estado es 5 el valor de seg = [1 6 11 16 21 26] es decir 5 segmentos.
- CONT mide el desplazamiento, para el caso anterior seg = [5 5 5 5 5]
- MU es la suma de los contenidos de los vectores para cada segmento y para todas las repeticiones, es decir en el ejemplo se suman los vectores del 1 al 6 valor por valor para el estado 1, luego se hace lo mismo para todos los estados

restantes, se reinicia el ciclo y se prosigue con la segunda repetición de st(2) a st(3) y se vuelve a sumar a los valores ya obtenidos y se concluye con la última repetición. Al final se divide entre los valores del contador para volver a normalizar.

- SIGMA es lo mismo que 'mu' pero cada valor se eleva al cuadrado y al finalizar se divide entre el contador menos el valor de mu elevado al cuadrado.

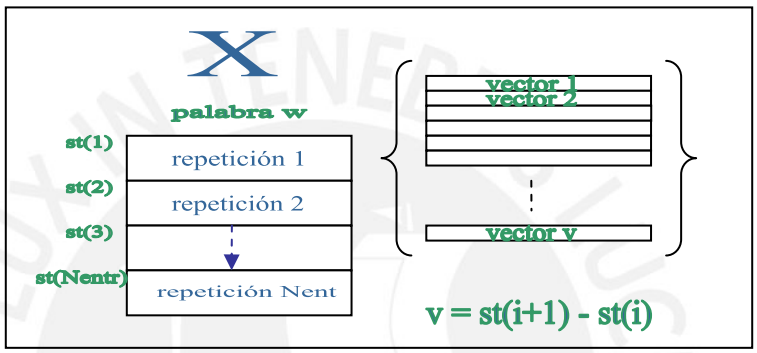


FIGURA. 2.15 Orden de los vectores en la matriz X para el entrenamiento

2.4.3.3 DISTRIBUCIÓN GAUSSIANA MULTI-VARIABLE

$$dens = \frac{1}{\sqrt{(2\pi)^P \, x \det(\mathrm{cov}_k)}} \, x \; e^{-0.5 \, x (mu_k - X)' \, x (mu_k - X) \, x (\mathrm{cov}_k^{-1})}$$

P : es el orden del cepstrum
X : es el vector extraído de la voz.
mu_k : es la media multi-variable de la k-ésima mixtura.
cov_k : es la matriz de covarianza diagonal

$$\mathrm{cov}_k = \begin{bmatrix} \sigma_{11}^{\,2} & 0 & 0 & 0 \\ 0 & \sigma_{22}^{\,2} & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \sigma_{nn}^{\,2} \end{bmatrix}$$

FIGURA. 2.16 Función de densidad gaussiana multi-variable

El algoritmo de la función **'gauseval.m'** (anexo F) elaborado en *Matlab*, halla el valor de la función de 'densidad gaussiana multi-variable' para un número T de vectores extraídos de la señal de voz., donde P es el número de orden de los vectores (coeficientes cepstrales):

2.4.3.4 FUNCIÓN DE PROBABILIDAD DE OBSERVACIÓN

Es la función que indica que tan probable es que un estado genere un vector y se determina por la siguiente fórmula:

$$b(q,x) = \sum (C_k \; x \; dens(X, mu_k, cov_k))$$

$\mathbf{C_k}$: coeficiente de k-ésima mixtura

2.4.3.5 REESTIMACIÓN DE PARÁMETROS

La reestimación evalúa el modelo de cada palabra por un ciclo de iteraciones donde los valores A y B son optimizados utilizando los algoritmos 'forward y backward' desarrollados en la función **'hmm_fb.m'** (anexo F), donde *alfa y beta* son las variables de cada algoritmo respectivamente. Esta función es requerida por la función principal **`hmm_reest.m'** que prepara las variables para su reestimación. Además la variable *dens* contiene los valores de densidad gaussiana para cada instante de tiempo para la estimación de la probabilidad de transición de estado. Y por último se emplea otra función **'mix_par.m'** que re-estima los parámetros (mu y sigma) de cada palabra a partir del parámetro gamma que es la variable obtenida durante el proceso de

recursividad '*forward*' y '*backward*'. Estas funciones se pueden ver en el **anexo F** donde está el programa completo de entrenamiento en *Matlab*.

2.4.4 RECONOCIMIENTO: ALGORITMO DE VITERBI

El reconocimiento de las palabras aisladas se basa en hacer una comparación ('*score*') entre el modelo de la palabra pronunciada y los modelos obtenidos durante el entrenamiento, produciendo un resultado que es determinado mediante el algoritmo de viterbi, el cual emplea inducción y cálculos logarítmicos para reducir el tiempo de procesamiento y proceder de manera eficiente. Este proceso es la solución al problema 2 visto en la sección 2.4.2 y sirve para determinar qué palabra o frase se dijo, utilizando su modelo A y B, siendo el resultado el modelo más probable de la lista de palabras.

$$\pi_i = \mathrm{Ln}(\pi_i) \qquad 1 \le i, j < \mathrm{Nest}$$

$$\mathrm{logdens}_i = \mathrm{Ln}\left(\frac{1}{\sqrt{(2\pi)^p \times \det(\mathrm{cov}_k)}} \right) \times e^{-0.5 \times (mu_1 - X)' \times (mu_1 - X) \times (\mathrm{cov}_1^{-1})}$$

$$A_{ij} = \mathrm{Ln}(A_{ij})$$

$$\mathrm{Plog}_1(i) = \pi_i + \mathrm{logdens}_1(i)$$

$$\mathrm{Plog}_t(j) = \mathrm{máx}\,[\,\mathrm{Plog}_{t-1}(i) + A_{ij}\,] + \mathrm{logdens}_t(j) \qquad 2 \le t \le T$$

$$\mathrm{logl} = \mathrm{máx}\,[\,\mathrm{Plog}_T(i)]$$

FIGURA 2.17 Algoritmo de viterbi

El algoritmo de viterbi ha sido elaborado en *Matlab* y para su utilización se debe recurrir a la función **'HMM_vit.m'.** Este algoritmo se halla en el disco compacto que

acompaña al texto, y además se ha elaborado un diagrama de flujo del proceso de reconocimiento que puede ser visto en el **anexo G**.

2.5 MICRÓFONOS AKG D230.

En el proyecto se utilizan dos micrófonos marca AKG modelo D230. Cada uno de ellos dispone de un transductor dinámico con característica direccional omni-direccional, de modo que la sensibilidad del micrófono es igual para todas las ondas sonoras que vengan de cualquier dirección. La bobina de compensación en el transductor evita interferencias producidas por campos magnéticos exteriores, los cuales surgen cerca de aparatos conectados a la red eléctrica.

TABLA 2.2 Características técnicas de los micrófonos

Modo de funcionamiento eléctrico	Micrófono dinámico
Modo de funcionamiento acústico	Micrófono de presión
Característica direccional	Omni-direccional
Campo de frecuencia	30 a 20000Hz
Sensibilidad a 1000Hz	2.5mV/Pa -52dB rel. Con 1V/Pa
Impedancia eléctrica	600Ω
Impedancia de carga recomendada	>2000 Ω
Dimensiones	Diámetro máx. 50mm
	Longitud 218mm
Peso neto	225g
Peso bruto	840g
Tipo de conector	Conector XLR de 3 polos
Modo de conexión del conector	Espigas 1,2,3: masa, audio en fase, audio

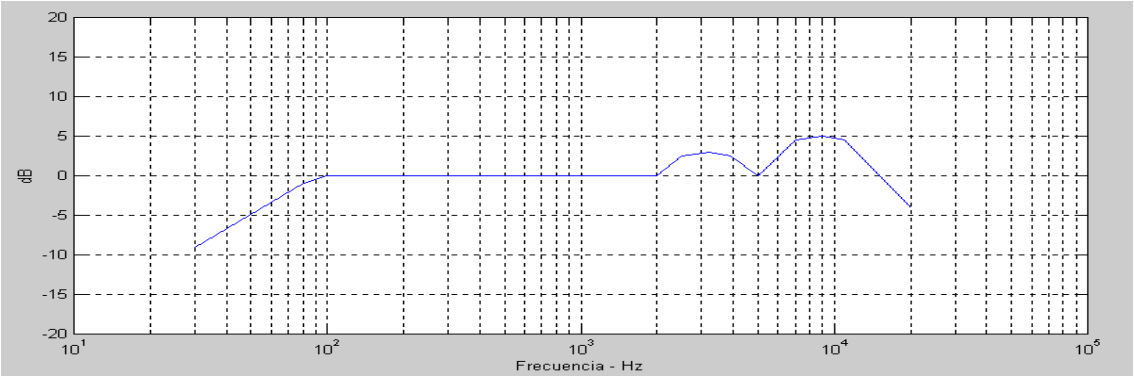


FIGURA 2.18 Respuesta en frecuencia

2.6 EQUIPO DE SONIDO (PANASONIC SC-AK45)

Equipo de sonido estéreo con tecnología *MASH (multi-stage noise shaping)* que reduce las frecuencias agudas y realza las frecuencias bajas a través de dos *sub-woofers* con una potencia de 80W(DIN) por parlante con impedancia de 6Ω. El equipo posee 4 modos de operación: amplificador auxiliar (AUX), casetera (TAPE), reproductor de discos compactos (CD), y receptor de radio AM, FM y WM (TUNER). En este proyecto sólo se va a controlar el selector, las funciones del amplificador, las emisoras de FM, y el reproductor de discos compactos.

Selector: consta de 4 funciones principales (*aux, tape, cd, tuner*)

Funciones del amplificador: son las que controlan el encendido, el ecualizador, el 'woofer' , el 'muting'y el volumen

Emisoras de FM: el control remoto controla sólo 12 del total de emisoras debido a que se deben pre-grabar en el sistema del equipo y esto es un inconveniente que no se puede superar, por tal motivo sólo se controlarán 12 de las emisoras a la vez.

Reproductor de discos compactos: el equipo posee un cartucho que puede contener hasta 5 discos. Se podrá controlar el número del disco, el número de pista y también las funciones para tocar la siguiente pista, tocar el anterior, repetir, 'stop'y 'play'.

2.7 SEÑALES INFRARROJAS DEL CONTROL REMOTO EUR644853

El control remoto es el instrumento que se utiliza para enviar las señales infrarrojas al receptor IR del equipo y consta de 32 funciones (teclas) de las cuales se copiarán algunas de ellas para el control por medio de la voz. La tabla 2.9 muestra todas las funciones implementadas por el fabricante del sistema.

Las tramas de las señales infrarrojas cumplen las reglas de un protocolo de control remoto mediante rayos infrarrojos para seguir un orden y evitar problemas entre un sistema y otro. De esta manera cada fabricante de artefactos tiene un código que es único para cada marca y para cada tipo de producto que utiliza el sistema.

El código de *Panasonic* consta de 48 bits: los primeros 25 sirven para identificar la marca y el tipo de sistema a controlar, y los 23 restantes para seleccionar la función. Para este caso el código de marca y modelo es **#080080A.**

La trama completa se transmite por medio de una señal portadora a una frecuencia de **32KHz**, además se hace uso de una cabecera y de una codificación para indicar si es '0' ó '1' el bit transmitido. El espacio entre transmisiones consecutivas debe ser mayor a 190T donde T es aproximadamente 420us.

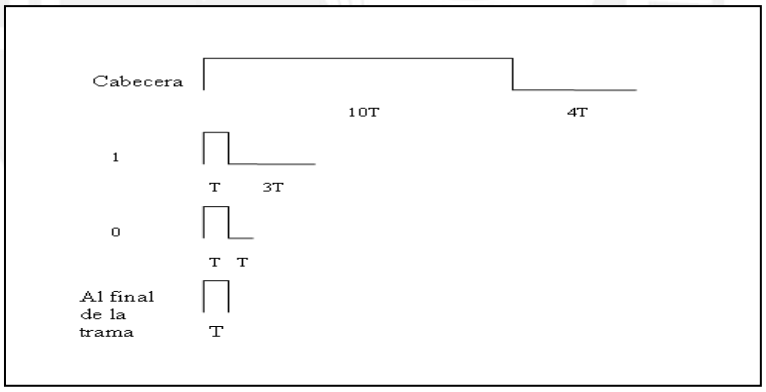
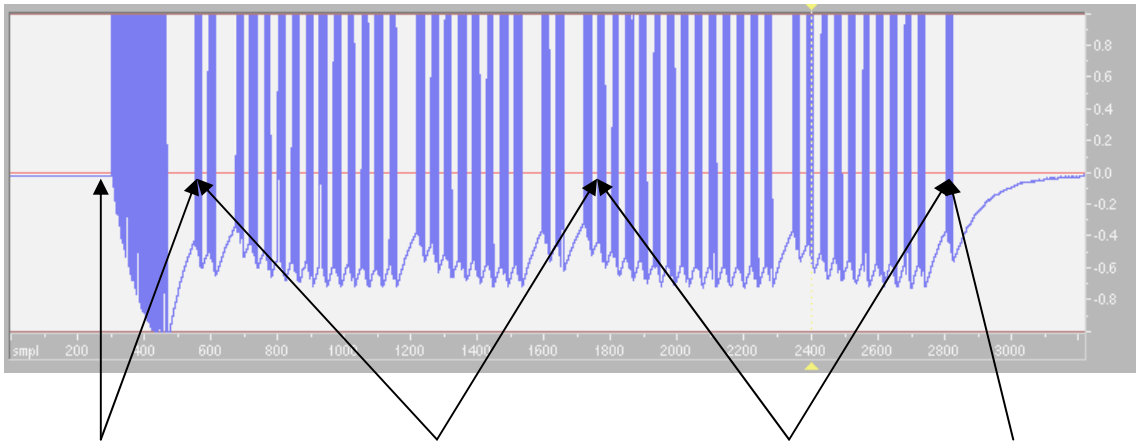


FIGURA 2.19 Codificación de la cabecera, valores lógicos 0 y 1 y pulso final de la trama.

Para obtener los códigos se tuvo que capturar la secuencia directamente del control remoto utilizando un aplicativo de audio 'cool-edit', ingresando la señal de la base del transistor del transmisor por la entrada del micrófono en la tarjeta de sonido de la PC.

La figura 2.13 muestra la señal (.WAV) de la secuencia de bits de la tecla 'volumen +'



Pre-data: 01000000000001000000001010 → 080080A
Función: 00000000000010000000001 → 000401

FIGURA 2.20 Señal IR (.WAV) de la tecla 'VOLUMEN +'.

TABLA 2.3 Funciones del control remoto EUR644853.

TECLA	FUNCIÓN	CÓDIGO
POWER	Encendido y apagado del equipo	#38BC81
SLEEP	Apagado automático en 30,60 ó 90 minutos	-
EQ	9 modos de ecualizado	#08C1CC
S.WOOFER	3 modos (apagado, medio y máximo)	#010346
1	Selección	#380835
2	Selección	#3888B5
3	Selección	#384875
4	Selección	#38C8F5
5	Selección	#382815
6	Selección	#38ª895
7	Selección	#386855
8	Selección	#38E8D5
9	Selección	#381825
0	Selección	#3898A5
>10	Para números mayores o igual a10. Se deben presionar dos teclas de números más. Por ejemplo el número 12 (>10,1,2)	#38211C
DISC	Selector de disco (disc + número(s))	#502570
PROGRAM	Programación de discos	-
CANCEL	Cancelar	-
REPEAT	Repetir pista	-
RANDOM	Orden aleatorio en la reproducción	-
AUX	Modo auxiliar	#00595C
TAPE	Modo reproductor de cintas	#10697C
CD	Modo reproductor de discos compactos	#00292C
TUNER	Modo de receptor de radioemisoras 3 opciones (FM, AM y MW)	#202500
|◄◄/◄◄	Empezar nuevamente (CD)	#38526F
|►►/►►	Siguiente (CD)	#3892AF
◄/ V	Anterior(TUNER)	#38605D
►/||/ Λ	Reproduce y Pausa (CD) y siguiente (TUNER)	#38506D
■ (STOP)	Detener (CD)	#38003D
MUTING	Silencio y no silencio	#004C49
VOLUME +	Sube el volumen	#000401
VOLUME -	Baja el volumen	#008481

3. IMPLEMENTACIÓN DEL SISTEMA EN LA TARJETA DE DESARROLLO TMS320C6711

3.1 CARACTERÍSTICAS DE LA TARJETA

La tarjeta de desarrollo TMDS320006711 está gobernada por el procesador digital de señales TMS30C6711 de *Texas Instruments*, el cual es capaz de efectuar hasta 1200 MIPS y 600 MFLOPs operando a 150MHz.

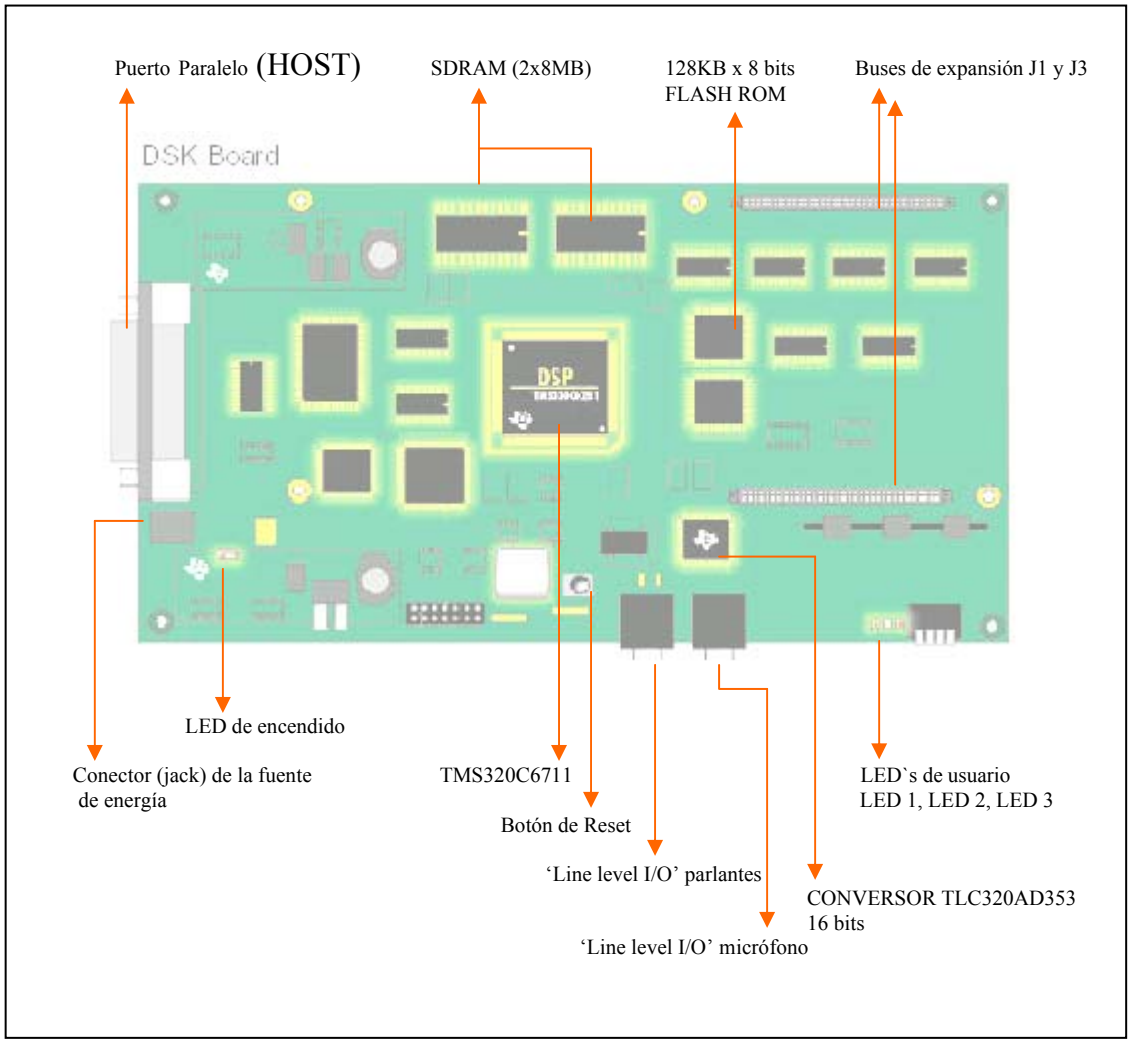


FIGURA 3.1 'TMDS320006711: tarjeta de desarrollo del procesador TMS320C6711' [16]

→ Procesador DSP TMS320C6711 de *Texas Instruments* con 16KB de memoria interna y *caché* de 2 niveles.

→ 2 circuitos integrados de memoria SDRAM de 8MB cada una.

→ 1 circuito integrado de memoria FLASH (ROM) de 128KB para operar sin conexión con la PC.

→ Reloj: 150MHz para el CPU y 100MHz para la interfase de memoria externa.

→ Conversor de audio AD535 de 16bits de resolución.

→ Buses de expansión para memoria externa y periféricos (J1 y J3).

El procesador 'c6711 está basado en una arquitectura '*Very Long Instruction Word*' (VLIW) con intrucciones de 32 bits con capacidad de proceso de hasta 8 de ellas por ciclo de reloj. Usa dos niveles de memoria caché independientes: una de programas (L1P) y otra de datos (L1D) que no están incluidas en el mapa de memoria. Además posee un nivel de operación 2 (L2) que permiten el acceso a la memoria interna por parte del CPU y el EDMA.

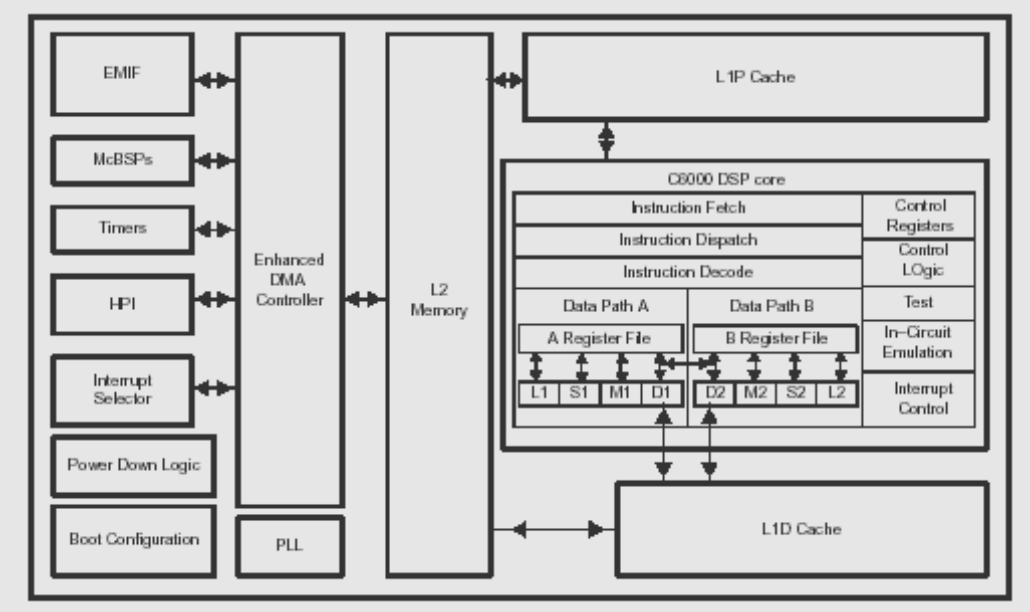


FIGURA 3.2 Diagrama de bloques del procesador 'C6711 [12].

Controlador EDMA: (Enhanced direct memory access) Transfiere datos entre direcciones del mapa de memoria sin intervención del CPU. El EDMA del procesador 'c6711 posee 16 canales programables.

HPI: Es un puerto paralelo a través del cual el host (procesador maestro, ej. PC) tiene acceso directo al espacio de memoria del CPU.

EMIF: (*Exchange memory interface*) Es la interface que controla el intercambio de datos entre los diferentes dispositivos en el sistema. El EMIF soporta muchos dispositivos incluyendo los siguientes:

- SRAM de ráfaga síncrona (SBSRAM).
- DRAM síncrona (SDRAM).
- Dispositivos asíncronos, incluyendo SRAM, ROM, y FIFOs.
- Dispositivo de memoria externa compartida.

Boot Configuration: El procesador 'C6711 provee una variedad de configuraciones en el arranque del sistema que determinan que acciones debe realizar el procesador luego de inicializar el sistema. Estos incluyen la carga del código desde un espacio ROM externo a través del EMIF y la carga desde un *host* externo a través del HPI.

McBSP: (*Multichannel buffered serial port*) Es el puerto de interfase serial que puede colocar muestras en la memoria automáticamente con la ayuda del controlador EDMA. Es compatible con las redes estándar T1, E1, SCSA, y MVIP y con sus predecesores:

- Comunicación *full-duplex*.
- Entramado independiente y sincronización para transmisión y recepción.
- Interfase directa para conversores de audio estándar, chips de interfase analógica. (AICs), y otros conversores analógico-digitales (A/D) y digital-analógicos (D/A) conectados serialmente.

Además, el McBSP tiene las siguientes características:

- Más de 128 canales de transmission y recepción.

- Selección de datos de 8, 12, 16, 20, 24 y 32-bits.
- Compresión ley-m y ley-A.
- Transferencia de 8-bit con LSB o MSB primero.
- Generación de trama y programación de reloj interno.

Timer: Dos temporizadores de 32 bits y son utilizados en las siguientes tareas:

- Eventos temporales.
- Contar eventos.
- Generador de pulsos.
- Interrumpir el CPU.
- Envío de eventos de sincronización al controlador EDMA.

GPIO: El GPIO provee una herramienta de propósito general para el ingreso y salida de señales al procesador. Estas señales pueden ser usadas para sincronización de eventos y del EDMA.

El mapa de la distribución de la memoria del sistema se muestra en el anexo H.

3.2 ENTORNO DE DESARROLLO: CODE COMPOSER STUDIO

El sistema se implementó utilizando la PC como sistema de arranque antes de ser independiente, a través del software de *Texas Instruments* que acompaña la tarjeta: '*Code Composer Studio'*.

Una de las ventajas de este entorno, es que se puede programar tanto en lenguaje ensamblador como en 'C' y se pueden combinar ambos lo cual es muy importante pues hay partes que resultan mejor hacerlas en un lenguaje que en otro. Si se desea se puede utilizar la herramienta *DSP-BIOS* para intercambiar data con el *host* y ejecutar el

programa a la vez, pero este no es el caso, pues el reconocimiento de las órdenes no requiere de acceso a la *PC* en medio del algoritmo.

Generalmente el programa principal, en donde se encuentra la aplicación y algunas funciones, se escriben en 'C', es decir con extensiones '.c y .h'; en ensamblador se escriben la rutina de inicialización si es que la hubiere, y también la tabla de vectores donde se indica cual es el '*entry point*' (inicio de la aplicación), el vector de '*reset*' y las interrupciones. Todos ellos con extensión '.asm'. Luego siguen las librerías que pueden ser '.h' o '.l' y son las que generalmente contienen información del tipo de procesador que se está programando u otras aplicaciones. Y para que todos estos objetos se enlacen y se pueda ejecutar la aplicación correctamente se crea un archivo con extensión '.cmd' que es el administrador y va a ubicar el programa y los datos en la memoria del sistema DSP. Al ensamblar el programa se genera el archivo principal, el ejecutable con extensión '.out'. Un archivo con extensión '.out' no necesariamente es ejecutable, puede ser un conjunto de datos de la memoria que se han copiado para salvar información o para grabarlos en una memoria *ROM*. Por último, para poder programar la memoria *Flash* se generan otros archivos del mismo tipo '.cmd' para ser utilizados por el ejecutable 'hex6x.exe' en la ventana de *comando de windows,* que convierte el código del archivo '.out' a códigos hexadecimales de 8 bits para poder ser grabados en la memoria flash a través de otro ejecutable llamado 'FLASH.exe'. Estos ejecutables son propiedad de *Texas Instruments* y vienen en el CD del software.

La ventana de la figura 3.3 muestra la aplicación desarrollada en el entrono del *Code Componer Studio*. La siguiente lista es la relación de archivos (objetos) elaborados para

el sistema de reconocimiento y la generación de las señales para la transmisión infrarroja:

- Control remoto de voz.c: Captura de la señal de audio proveniente de la interfase de entrada (el filtro), verifica la presencia de voz, ejecuta el algoritmo de viterbi y genera la secuencia infrarroja correspondiente.
- Boot.asm: Contiene el arranque y la configuración del sistema, copia el código de la aplicación, los modelos de Markov y los códigos IR de la Flash a la SDRAM y configura el EMIF para uso de los puertos de salida (LED's)
- Vectors.asm: Es la tabla de vectores, de reset y de interrupciones.
- Lnk.cmd: Es el enlazador del programa.

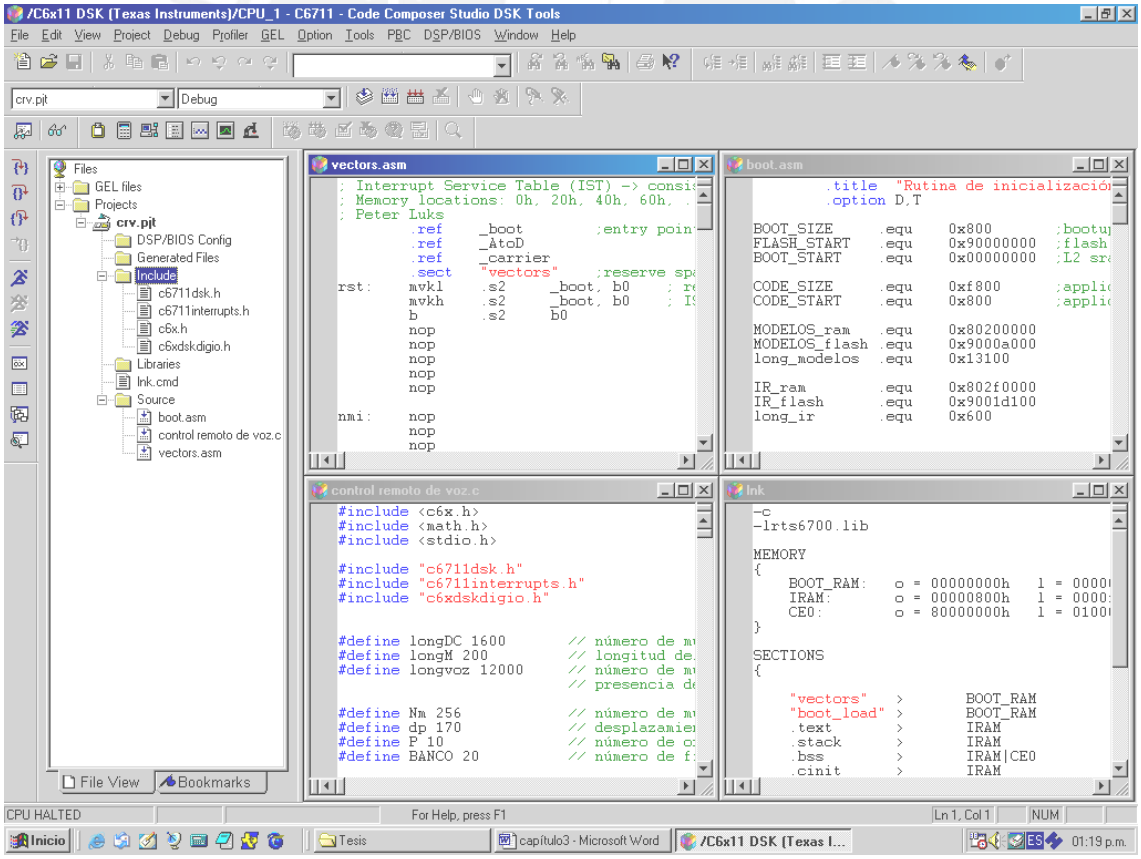


FIGURA 3.3 Entorno de programación: 'Code Componer Studio'.

3.3 ADQUISICIÓN DE LA SEÑAL DE VOZ

La señal de voz proviene de la interfase de entrada (la salida del filtro) e ingresa a la tarjeta de desarrollo utilizando un conversor de audio de 16bits modelo TLC320AD535C/I de doble canal (voz y datos) que permiten velocidades de muestreo independientes a más de 11.025 kHz con rango dinámico de 80 dB en cada canal y con amplificadores de ganancia programable tanto en la entrada como en la salida [14].

La comunicación con el procesador se logra utilizando uno de los dos puertos del McBSP que consisten de una ruta para datos y otra para el control, y se establece mediante vías separadas tanto en la transmisión como en la recepción. Para el control se utilizan otros 4 pines(reloj y trama de sincronización). El dispositivo se comunica con el McBSP mediante registros de control de 32-bits accesible mediante el bus de periféricos.

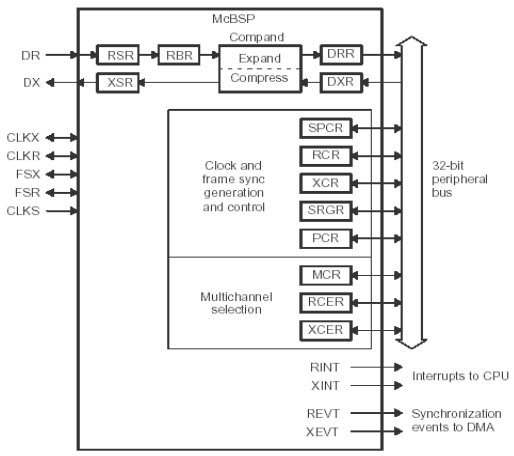


FIGURA 3.4 McBSP (Multichannel Buffered serial port) [12].

La señal es capturada a una frecuencia de muestreo de 8 kHz y con una ganancia de entrada de 20dB (pre-amp gain). Las tablas 3.1 y 3.3 muestran la inicialización del McBSP y del codificador AD535 respectivamente.

TABLA 3.1 Inicialización del McBSP_0

Registro de control del puerto serial	McBSP0_SPCR = 0	Inicializa el Puerto de control
Pin del Registro de control	McBSP0_PCR = 0	Activa pin de control
Registro de control de recepción	McBSP0_RCR = 0x10040	Recepción de 16 bits por trama
Registro de control de transmisión	McBSP0_XCR = 0x10040	Transmisión de 16 bits por trama
Registro de recepción de datos	McBSP0_DRR = 0	Inicializa el registro de tx datos
	McBSP0_SPCR = 0x12001	Activa el Puerto serial

Para poder leer y escribir en el McBSP se han creado 2 funciones para la recepción y transmisión de datos:

TABLA 3.2 Funciones de lectura y escritura sobre el McBSP_0.

Mcbsp0_read()	Espera hasta que el pin 1 del Mcbsp0_SPRC sea 0 y luego retorna el valor recibido en el registro de recepción DRR.
Mcbsp0_write(data)	Espera hasta que el pin 18 del Mcbsp0_SPRC sea 0 y luego asigna el valor de la data en el registro de transmisión DXR

En la inicialización del conversor AD535 se establece el tipo de entrada (mic) y la ganacia de entrada:

TABLA 3.3 Inicialización del codificador de audio AD535 [14].

Registro 3 – Control del canal de voz	0x0386	Inicializa el software del canal de voz
	0x0306	Selecciona 'mic-preamp' como entrada en el ADC
Registro 4 – Ganancia de entrada	0x0479	ganancia pre-amp de 20 dB

El proceso de captura de la señal de voz requiere de un aviso que indique que la siguiente muestra está lista para poder ser recibida, por tal motivo se recurre a la interrupción de recepción del McBSP0: 'RINT0' cuyo valor es '001101b'. La tabla 3.4 muestra la configuración del RINT0 en la interrupción 12.

TABLA 3.4 Configuración de la interrupción RINT0 en el vector 12.

config_Interrupt_Selector(12, RINT0)	Configura interrupción 12 para McBSP0
enableSpecificINT(12)	Habilita la interrupción 12
enableNMI()	Habilita la interrupción no enmascarable
enableGlobalINT()	Activa todas las interrupciones

El proceso de detección de voz se realiza dentro de la rutina de interrupción 12. Un *buffer* tipo cola que almacena valores de energía de 200 muestras (bufferE), sirve para

determinar si hay voz mediante la comparación con un valor umbral. Paralelamente otro *buffer* almacena los valores de las muestras analizadas (bufferM), y si hay voz se graban otras muestras hasta completar 1.5 segundos es decir 12000 muestras (voz), siendo las 200 primeras las analizadas al principio.

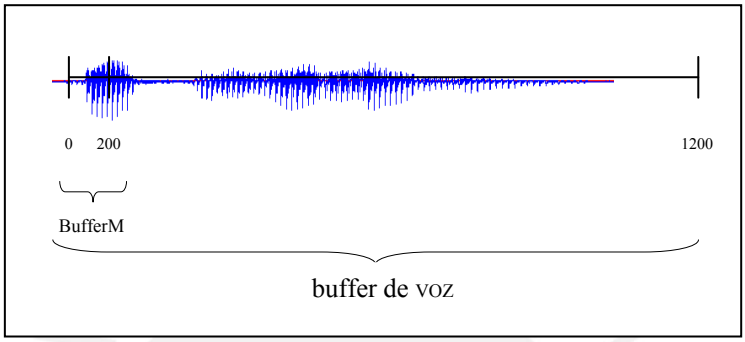


FIGURA 3.5 Buffer de voz

Una vez que hay presencia de voz, se procede a hallar la posición final de la orden dentro de las 12000 muestras. Para ello se ejecuta un proceso similar al de detección pero desde la muestra 12000 hacia atrás, hasta que sea mayor a otro valor umbral y como la posición inicial ya se ubicó, por lo tanto se tienen los límites de la voz, por lo que se puede pasar a la fase de pre-procesamiento.

3.4 PRE-PROCESAMIENTO DE LA SEÑAL DE VOZ

El pre-procesamiento es igual al desarrollado en *matlab* y no ha sido necesario ningún artificio para su implementación en la tarjeta. El cuadro 3.5 muestra las funciones implementadas para este pre-proceso.

TABLA 3.5 funciones de pre-procesamiento

Preenfasis(Nm)	Filtro de pre-énfasis (de la ventana con las muestras)
VentanaHamming (Nm, i_o_real, i_o_complejo)	Ventana de hamming (i_o_real es el resultado de pre-énfasis y i_o_complejo es cero)
fft(Nm, i_o_real, i_o_complejo)	Transformada de fourier (radix-2) (i_o_real es el resultado de la ventana Hamming y i_o_complejo es 0).
magnitud(Nm, i_o_real, i_o_complejo)	Valor absoluto de la FFT
CoeficientesCepstrales(fi, fs)	Coeficientes cepstrales (inicio y fin del filtro triangular)

3.5 ALGORITMO DE VITERBI Y RECONOCIMIENTO

Se han implementado 32 funciones de control mediante la voz, estas funciones se pueden ver en el **anexo E** que muestran las órdenes y su función. Hay dos tipos de funciones:

- Inmediatas, que son aquellas que se ejecutan una sola vez.
- Continuas, que se ejecutan varias veces hasta que haya presencia de voz nuevamente para indicar el fin de la función (woofer, volumen, baja, EQ, buscar y pista).

El algoritmo de viterbi decide cuál de las 32 órdenes es la que se ha pronunciado, si este valor se divide entre la longitud de dicha frase, se obtiene un valor relativo que puede ser utilizado para descartar frases o sonidos que no estén en la lista de posibles órdenes; también se utiliza la misma longitud para restringir. A mayor energía, el valor relativo será mayor pero siempre dentro de un rango. La figura 3.6 muestra los resultados obtenidos al pronunciar la orden 10 (0-10→11) 'Radiomar'. El valor relativo de esta orden varía entre -0.16 y -0.19 luego de un análisis mediante varias pruebas; entonces en el programa se restringe con un margen de seguridad al rango de -0.14 a -0.21 tal que si se dice cualquier cosa y resulta que el algoritmo de viterbi encuentra que la orden 10 es la más probable, esa restricción va a indicar que no se dijo esa frase con lo cual se descarta y el sistema no ejecuta ninguna orden. Este sistema restringe muchas 'órdenes fantasma', pero no es 100% eficaz.

Es importante recalcar que todo el entrenamiento se hizo utilizando *matlab* y que el algoritmo de viterbi elaborado en la tarjeta funciona bien; es decir se ha logrado compatibilidad entre dos entornos de programación diferentes.

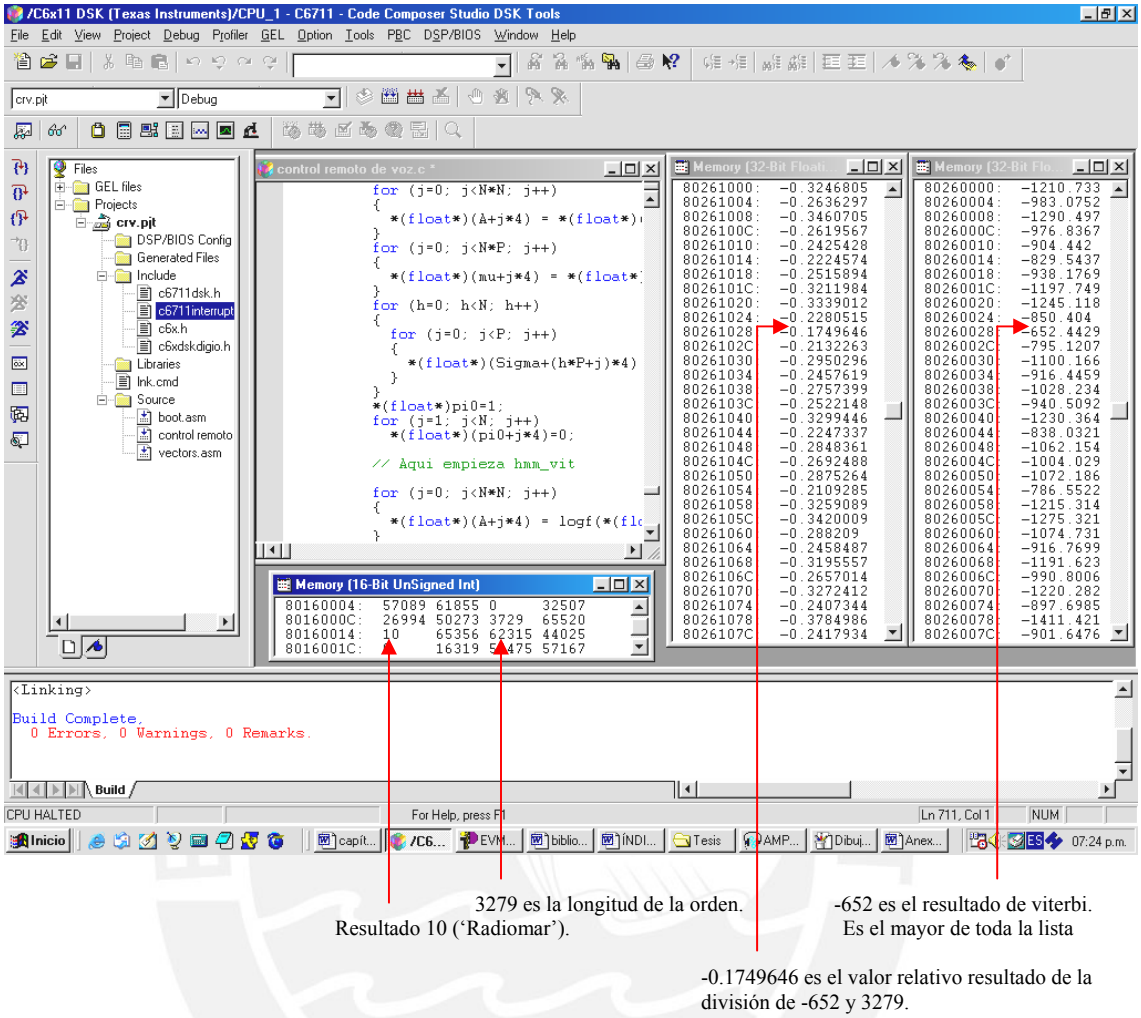


FIGURA 3.6 Resultados del algoritmo de viterbi usando el 'Code Componer Studio'.

3.6 GENERACIÓN DE LAS TRAMAS PARA LA TRANSMISIÓN IR

En la sección 2.7 se detalló que el protocolo de comunicación infrarroja se debe transmitir mediante una señal portadora de 32kHz para evitar que otras fuentes IR interfieran con el sistema. Por tal motivo se utiliza uno de los temporizadores que tiene el procesador para indicar en que momento la portadora debe cambiar de estado de nivel bajo a alto y viceversa mediante el uso de una nueva interrupción producida por el mismo temporizador (TINT1). La rutina de interrupción consiste en cambiar de estado

lógico una variable (de 0 a 1 ó de 1 a 0) cada vez que se ejecute a una frecuencia de 32kHz, es decir cada 31.25us.

Cada temporizador tiene tres registros, los cuales se describen en la tabla 3.6

Tabla 3.6 Registros de los temporizadores [12].

Registro de control (CTL)	Determina el modo de operación, monitorea el estado del temporizador y controla la función del pin TOUT.
Registro de período (PRD)	Contiene el número de ciclos de reloj a contar. Este número controla la frecuencia de la señal TSTAT.
Registro contador (CNT)	Contiene el valor actual del contador que se incrementa a una frecuencia de f-clock/4, es decir 150MHz / 4.

Estos registros se configuran de la siguiente manera (tabla 3.7) para poder obtener la frecuencia deseada:

Tabla 3.7 Configuración del temporizador 1 para la frecuencia portadora.

TIMER1_CTRL	Se desactivan los pines 6 y 7 (HLD - estado de espera)
TIMER1_CTRL	Se activa el pin 0 (TOUT es el pin de salida del temporizador)
TIMER1_CTRL	Se activa el pin 9 indicando que el reloj es interno (150MHz / 4)
TIMER1_CTRL	Se activa pin 8 indicando modo de operación 'clock' 50% DutyCycle
TIMER1_PRD	Igual a 586 (586 * 2 = 1172 → 1172 * 4 / 150MHz = 31.25333us → 1/ 31.25333us = 31.996 KHz
TIMER1_CTRL	Se activan pines 6 y 7 (HLD=1 termina la espera y GO=1 empieza a contar)

Una vez que empiece la cuenta, cada 4/150MHz el contador (TIMER1_CNT) incrementa su valor y cuando llegue a ser igual a 586 (valor en TIMER1_PRD) el contador vuelve a 0 y el pin TINT1 se activará indicando que el temporizador solicita una interrupción en donde se cambiará de estado a la portadora.

La rutina se ha implementado en la interrupción 10 con el nombre '*carrier( )*' y consiste en cambiar de nivel lógico a la variable 'luz', entonces cada vez que se solicite enviar una señal infrarroja basta con hacer la codificación y recoger el valor de la portadora en esta variable para encender y apagar el LED infrarrojo a esa frecuencia.

Para la codificación se utiliza el temporizador '0' para enviar la portadora por un determinado tiempo o para lograr un simple retardo por medio de una función denominada '*retardo( )*'.

Tabla 3.8 Función retardo(int delay, int valor)

Retardo(T,0)	Simple retardo por un tiempo T (transmite nivel bajo, es decir led IR apagado)
Retardo(T,1)	Transmite la señal portadora por un tiempo T (led IR se enciende y se apaga a aprox 32KHz)

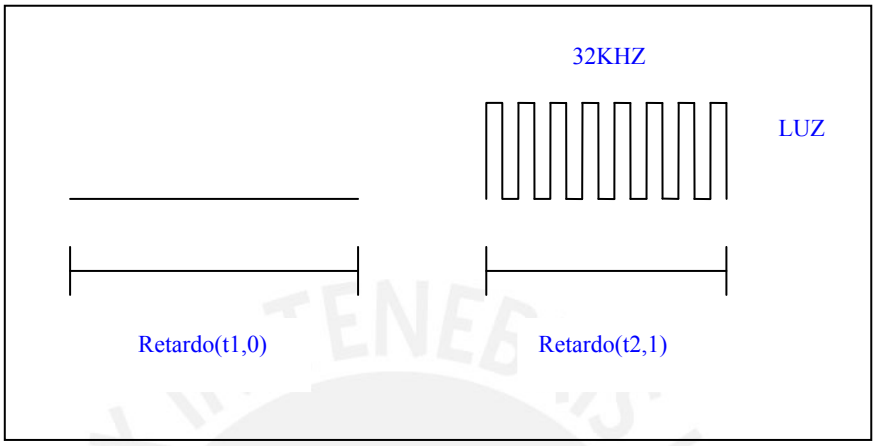


FIGURA 3.7 Encendido y apagado del led infrarrojo mediante la función 'retardo'.

En el protocolo un '1' es codificado como '1-0-0-0' y un '0' como '1-0', es decir cada '1' se implementa con la secuencia *'un retardo(T,1) y tres veces retardo(T,0)'* y cada '0' con *'un retardo(T,1) y un retardo(T,0)'*. Utilizando la tabla de códigos IR previamente almacenada en la memoria y con la función creada, ya se puede enviar la secuencia lógica al circuito de transmisión mediante el pin CNTL0 del puerto I/O cuya extensión DB_CNTL0 se halla en el bus de expansión para periféricos J3.

3.7 INDEPENDENCIA DE LA PC: MEMORIA FLASH

La tarjeta de desarrollo tiene una memoria FLASH que sirve para operar en modo 'stand alone' es decir sin uso de la PC para el proceso de arranque. Esta memoria es de 128KB x 8 bits y para ser programada requiere que el código sea también de 8 bits, por lo que se utiliza la aplicación 'hex6x.exe'. Para transferir todo el sistema a 8 bits se han elaborado tres archivos de comando (.cmd): uno para el programa, otro para los

modelos de Markov y el último para los códigos infrarrojos. Los códigos a transferir son los de extensión (.out) que se obtienen cuando se enlaza el programa o para el caso de los datos cuando se transfiere parte de la memoria a un archivo de este tipo.

Luego se utiliza la aplicación 'Flash.exe' para grabar en la memoria. La tabla 3.9 muestra las posiciones de memoria utilizadas:

Tabla 3.9 Distribución del código y datos en la memoria Flash

Programa	0x90000000
Modelos de Markov	0x9000a000
Códigos IR	0x9001d100

Cuando se enciende el sistema un bloque de 1KB se copia automáticamente de la memoria FLASH a la RAM interna, por lo que en ese sector debe estar la rutina de inicio (*boot*). Esta rutina, primero coloca al procesador en modo L2 (unificado), luego configura el EMIF para poder transferir 8 bits desde la ROM. La siguiente fase es copiar todo el código del programa a partir de la dirección 0x00000800 hasta la dirección 0x00009fff. Los datos son copiados a sus posiciones en la SDRAM (CE0): Markov a partir de la dirección 0x80200000 y los códigos IR en la dirección 0x802f0000. Una vez culminada esta labor, se procede a configurar el EMIF para que el espacio de memoria CE1 que ocupan la FLASH y el puerto I/O vuelvan a ser de 32 bits. Por último el contador del programa se transfiere al 'entry_point', es decir el inicio de la aplicación ('_c_int00').

3.8 INTERFASE DE ENTRADA: FILTRO LMS

El filtro LMS ha sido implementado en la tarjeta DSP56002EVM de Motorola que utiliza el procesador DSP56002 de 24 bits que opera a 40MHz con más de 120 MFLOP's, con memoria interna de programas de 512x24bits y con dos memorias de

datos de 256x24bits. Posee un conversor de audio estéreo (*Crystal Semiconductor's CS4215*) que permite el ingreso de dos señales de audio a la vez. El conversor tiene la capacidad de muestrear señales a velocidades de 48, 32, 16, 9.6, ó 8kHz. Formato de datos lineal para 8 y 16 bits. 8 bits lay A y mu. Opción para otras velocidades de muestreo tales como 44.1 kHz (discos compactos). Además la tarjeta consta de un pre-amplificador de 22.5dB MC33078.

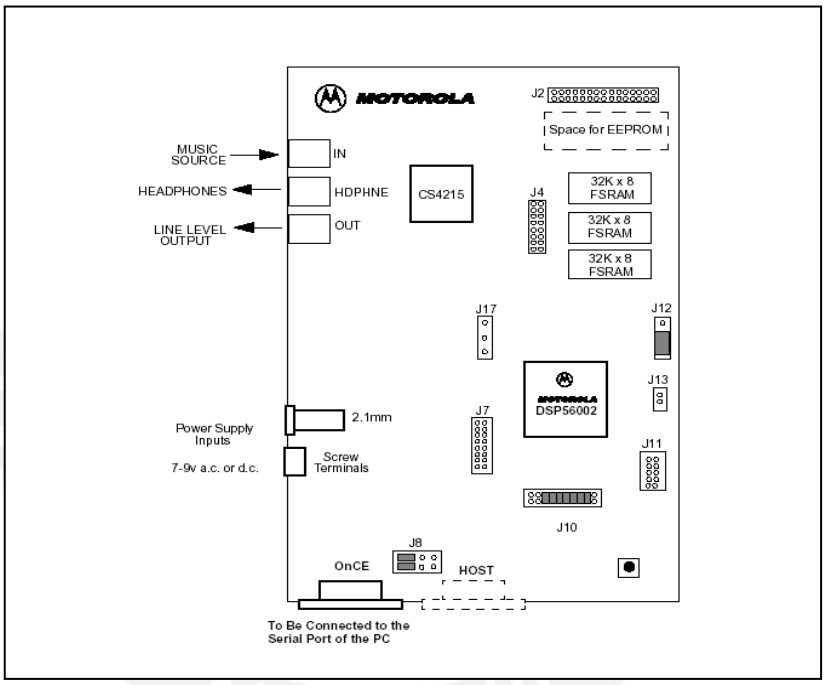


FIG 3.8 Tarjeta de evaluación DSP56002EVM [18]

La figura 3.8 muestra la distribución de la memoria para 2 posibles configuraciones: de 16K y 32K. Estas configuraciones se pueden controlar mediante el puente (*jumper*) J-12 tal como se ve en la figura 3.8 en donde se ha colocado en la posición de 16k.

Estas distribuciones dependen de tres tipos de memoria:

- Memoria interna, o sea dentro del procesador (on-chip).
- SRAM externa de 32K x 24-bits con cero estados de espera para memoria de expansión.

- Opción para programación con independencia de la PC utilizando memoria flash EEPROM de 32K x 8 bits.

La memoria 'P' es de programación y las memorias 'X' e 'Y' son de datos.

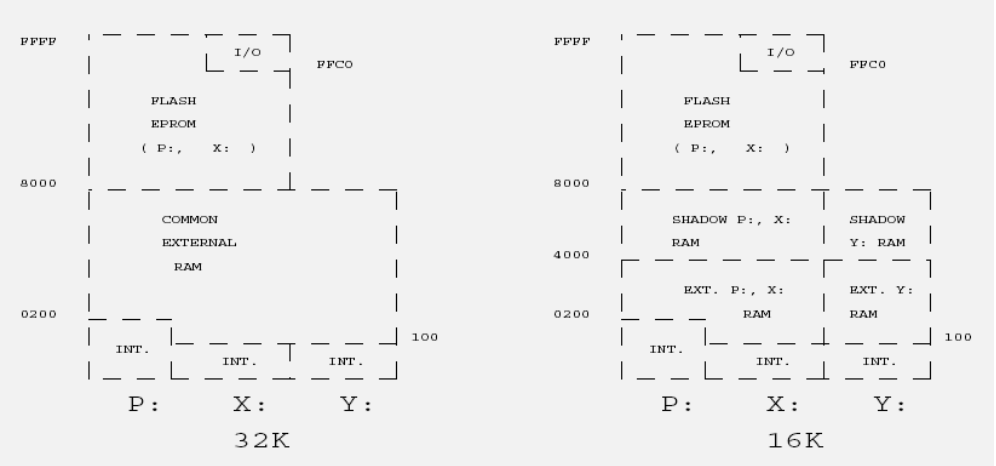


FIGURA 3.9 Distribución de la memoria [20].

En el proyecto se utiliza la configuración de 16K.

Para la adquisición de señales analógicas, Motorola ofrece dos rutinas: **'ada_init.asm'** que se utiliza en la inicialización del conversor CS-4215 y **'txrx_isr.asm'.**

1. Configurar el puerto SSI y las lineas GPIO utilizadas (D/C ~ y Reset)
2. Seleccionar modo de control y resetear el conversor (50ms @ 40MHz)
3. Inicializar el buffer de transmisión de datos con palabras de control (CLB=0)
4. Habilitar SSI
5. Esperar hasta que CLB=0 en el buffer de recepción
6. Colocar el CLB=1 en el buffer de transmisión
7. Enviar 4 tramas de datos
8. Deshabilitar el puerto SSI
9. Programar el SSI para la recepción de la trama de sincronización y el reloj.
10. Seleccionar modo de datos (D/C~ = 1)
11. Habilitar puerto SSI.

CUADRO 3.1 Inicialización del conversor

Al muestrear una señal analógica se necesita una rutina de interrupción que se ejecute apenas la frecuencia de muestreo indique que ya se debe digitalizar. Sucede lo mismo para convertir una señal digital en analógica; por lo tanto, la rutina de interrupción posee dos partes:

- **SSI Receive ISR:**

  Se lee el puntero del *buffer* de recepción y si no hay trama de sincronización, se recibe el dato a través del SSI, de lo contrario se inicializa nuevamente el puntero. Luego de recibir el dato, se envía al *buffer* y se actualiza el puntero

- **SSI Transmit ISR:**

  Se lee el puntero del *buffer* de transmisión y si no hay trama de sincronización, se transmite el dato de este *buffer* a través del SSI, de lo contrario se inicializa nuevamente el puntero. Luego de transmitir el dato se actualiza el puntero.

El vector de estas rutinas debe ser declaradas al inicio del programa principal en la dirección **P:$000C** para que el procesador sea forzado a ingresar a la rutina de interrupción cuando sea necesario.

Tabla 3.10 Vectores de interrupción de la adquisición de datos

P:$000C	0 - 2	SSI Receive Data
P:$000E	0 - 2	SSI Receive Data With Exception Status
P:$0010	0 - 2	SSI Transmit Data
P:$0012	0 - 2	SSI Transmit Data with Exception Status

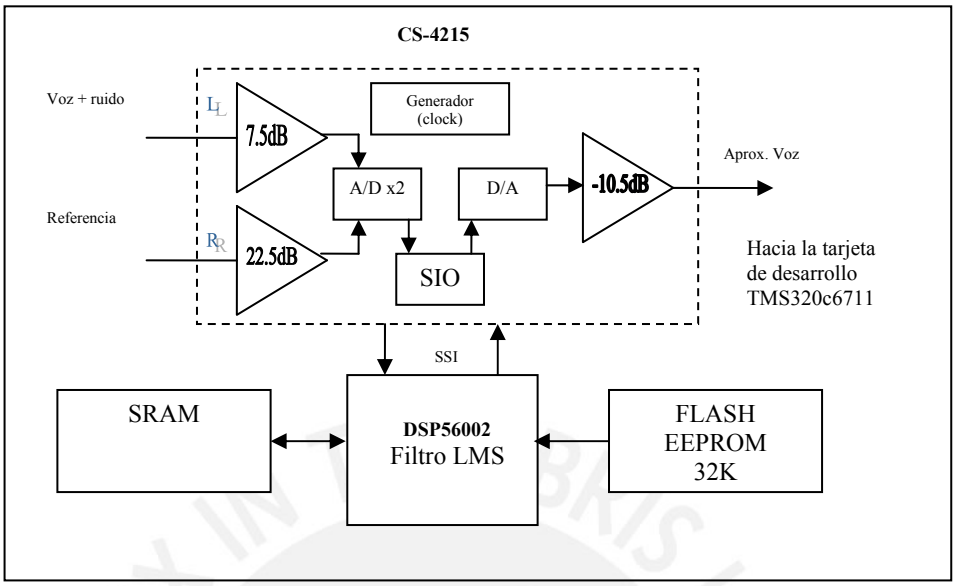


FIG 3.10 Diagrama de bloques del sistema de filtrado en la tarjeta DSP56002EVM.

Las variables que se utilizan en la implementación del filtro son las siguientes:

N, número de orden del filtro.

u, tamaño de paso.

r, vector que contiene las N-últimas muestras del 'ruido' en forma de cola.

W, vector que contiene los coeficientes del filtro.

d, señal de voz contaminada.

e, señal de error, o sea la voz depues del proceso de filtrado.

ruido, señal de referencia.

El programa **'LMS.asm'** se muestra en el **anexo D** el diagrama de flujo en el **anexo B.**

3.9 INTERFASE DE SALIDA: CIRCUITO DE TRANSMISIÓN DE SEÑALES INFRARROJAS

La tarjeta de desarrollo de *Texas Instruments* tiene dos buses de expansión para memorias externas y para periféricos (J1 y J3). El bus que se usa es el J3 pues aquí están los pines del puerto de salida CNTL0 (transmisor IR) y CNTL1 (led verde de presencia

de voz) en los pines 64 y 63 respectivamente. Además hay pines de voltaje y tierra: 3.3v (pines 19 ó 20) y GND (pines 3, 4, 7, 8, 25, 26, 31, 32, etc).

El circuito de transmisión consiste en un led transmisor de rayos infrarrojos que es activado por un transistor pnp 2N3906 que opera en la región de corte y saturación, además se utiliza un *buffer inversor* 74LS06 para evitar dañar la tarjeta en caso de un cortocircuito. El circuito del led de presencia de voz es aún más sencillo debido a que solo utiliza una resistencia y otro buffer de protección. La figura 3.11 muestra el diagrama esquemático y diagrama de pistas del circuito.

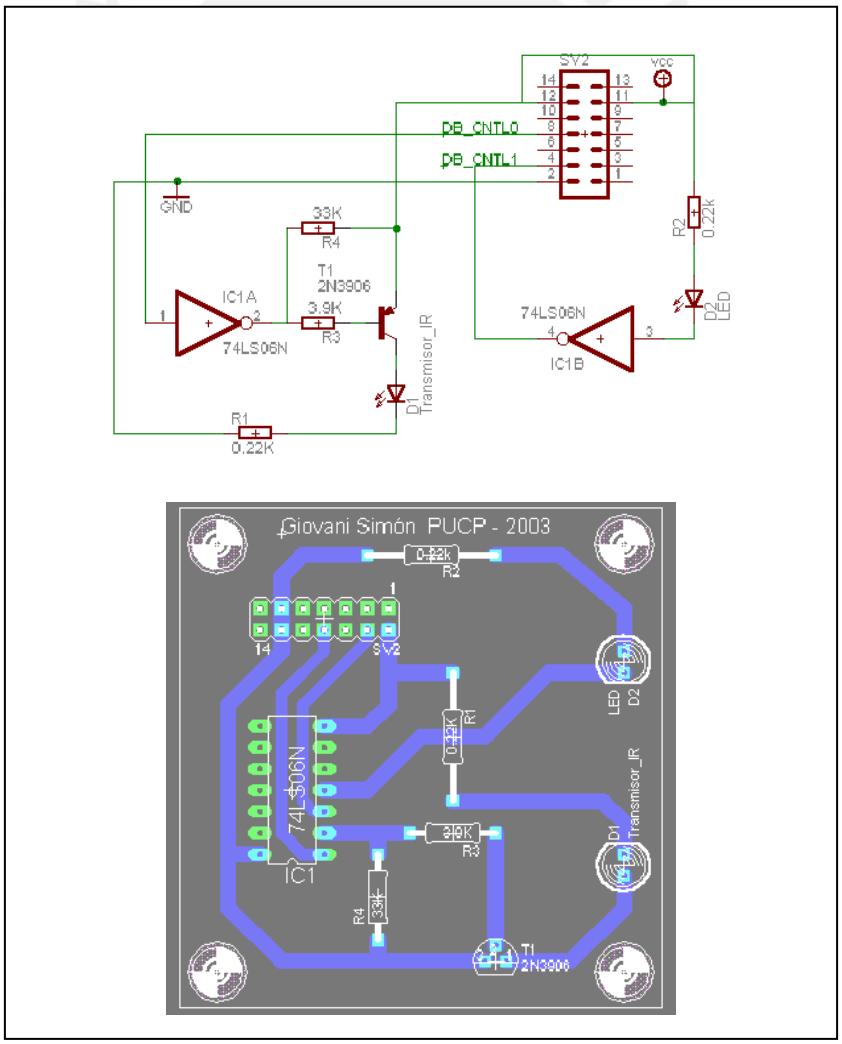


FIGURA 3.11 Diagrama esquemático y circuito impreso del transmisor IR.

4. PRUEBAS Y RESULTADOS

4.1 EFICIENCIA DEL ALGORITMO UTILIZANDO MATLAB Y EL FILTRO IMPLEMENTADO EN LA TARJETA

4.1.1 CÁLCULO DEL MEJOR NÚMERO DE ORDEN PARA EL FILTRO

El siguiente ensayo determinará el mejor número de orden y tamaño de paso que serán usados en el filtro. Para esto, se ha utilizado solamente ruido de referencia, una melodía de 20 segundos para todas las combinaciones de **N** y **u**, mientras que la señal de voz es nula. Mediante diferentes combinaciones de estos elementos, el error se puede minimizar, teniendo en cuenta que a mayor número de orden el error va a ser menor, pero en un determinado momento el procesador no puede hacer los cálculos requeridos en el tiempo necesario, siendo una desventaja que limita la versatilidad del filtro.

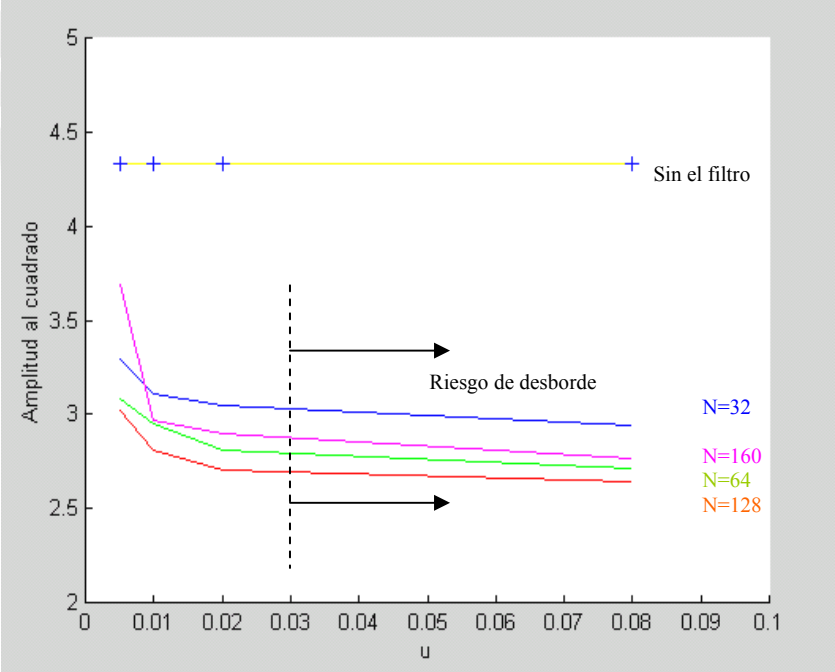


FIGURA 4.1 Cuadrado promedio para diferentes valores de N y u

Debido a que el vector de los coeficientes (W) del filtro depende del valor del error directamente y este a la vez de la señal de voz, el filtro corre riesgo de desborde es decir que la amplitud del error se eleve sin control. Cuando se utiliza un tamaño de paso grande (u>0.03) y el error en un determinado momento se eleva, el valor de W se torna inestable, por lo cual se debe elegir un valor menor.

En la figura 4.1 se puede apreciar como la energía (el cuadrado promedio) se va reduciendo desde N=32 hasta N=128, pero luego vuelve a incrementarse como se dijo anteriormente. Entonces se puede concluir que el mejor valor de N bordea el valor 128.

El error varía poco para un tamaño de paso entre 0.02 y 0.08, además debe ser menor a 0.03 para evitar el desborde, entonces se ha elegido el valor 0.02 que asegura la estabilidad del filtro y mantiene el error a un nivel aceptable.

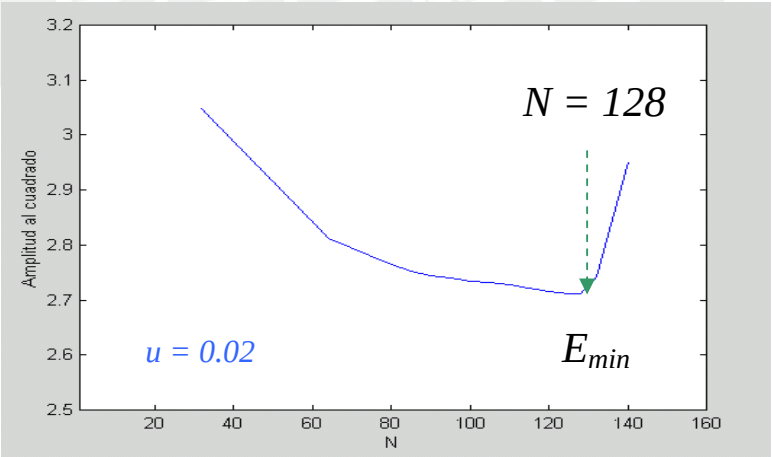


FIGURA 4.2 Valor del número de orden N donde el error es mínimo para u=0.02

4.1.2 ENERGÍA DE LA SEÑAL UTILIZANDO EL FILTRO

En la sección 1.3 se hizo una prueba para determinar la energía a medida que se subía el volumen del equipo dando como resultado la curva del gráfico 1.6 la cual será comparada con el resultado utilizando el filtro adaptivo LMS.

El micrófono primario (voz + sonido del equipo) se mantuvo en la misma posición que en la prueba anterior (a 2m de los parlantes) y el micrófono de referencia que utiliza el filtro se colocó a 50cm de los parlantes. Lo demás es similar, es decir 18 diferentes volúmenes. A continuación se presenta la tabla con los valores obtenidos:

TABLA 4.1 Valores de la amplitud^2 del sonido del equipo utilizando el filtro adaptivo LMS

	∑amplitud^2		∑amplitud^2		∑Amplitud^2
1	2.2575	7	2.2689	13	2.3164
2	2.2678	8	2.2637	14	2.3594
3	2.2555	9	2.2602	15	2.3958
4	2.2731	10	2.2642	16	2.4562
5	2.2759	11	2.2821	17	2.5252
6	2.2724	12	2.2966	18	2.5988

La figura 4.3 muestra dos curvas, donde la de color verde es la obtenida sin usar el filtro, y allí se puede ver como la curva crece rápidamente a medida que se sube el volumen, y la de color azul es la curva que se obtuvo al usar el filtro. Claramente se ve la diferencia: al principio la energía del sonido es similar debido a que la amplitud aún es pequeña y a partir de cierto punto la curva presenta sólo un crecimiento suave. Este es el resultado que se esperaba, es decir que la energía del ruido sea pequeña haciendo que la relación señal a ruido sea grande. Por último esta prueba confirma el éxito del filtro ante la presencia de ruido.

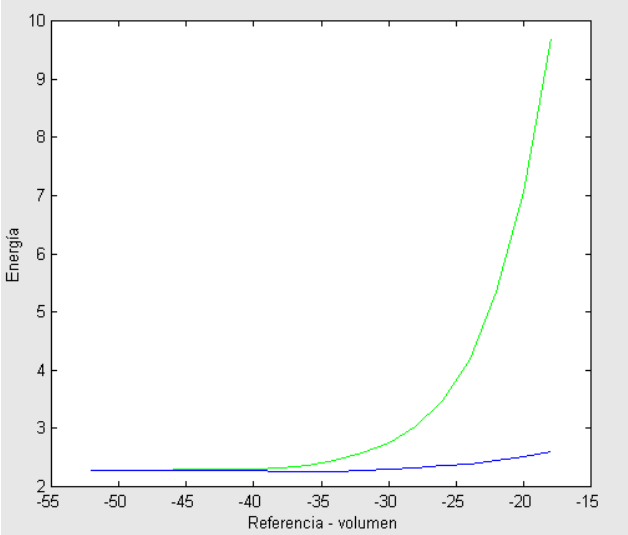


FIGURA 4.3 Comparación de la energía al usar y no el filtro LMS.

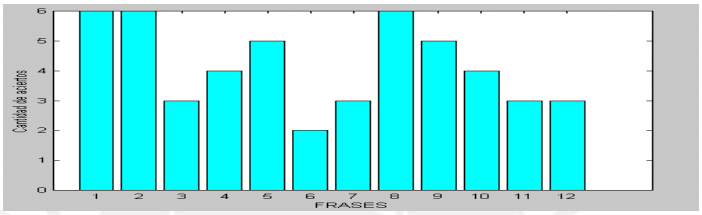
Es importante resaltar que en esta prueba no es necesario usar el sonómetro porque sólo se desea saber si el filtro cumple con la reducción de la energía del ruido a una distancia determinada entre los micrófonos y el equipo, ya que es el primer paso para el futuro reconocimiento.

4.1.3 CURVA EFICIENCIA VS. SONIDO DEL EQUIPO

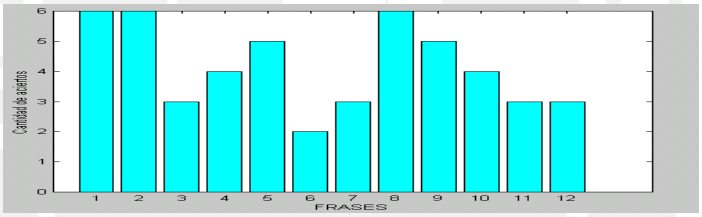
La primera prueba que se hizo con el filtro implementado fue la más importante del proyecto, esta prueba es la base del problema. Primero se probó el filtro utilizando el entrenamiento del capítulo 1 y el resultado fue malo, la eficiencia cayó por debajo del resultado cuando no se usó el filtro, lo que sucedía era que el filtro en sí es una función de transferencia (F.T.) entonces las voces grabadas en la primera prueba no tomaban en cuenta esa F.T. por lo que la voz que salía del filtro estaba modificada y por lo tanto era diferente a las voces en el entrenamiento. Entonces se procedió a elaborar un nuevo entrenamiento, similar pero utilizando el filtro, y el resultado fue el esperado, la

eficiencia mejoró al punto que se obtuvo 100% hasta con 80dBC de ruido usando 12 frases. A continuación se presentan los resultados obtenidos en la prueba de eficiencia usando el filtro. Cada barra corresponde a cada una de las 12 frases dichas según la lista que se elaboró en la página 5, la altura es la cantidad de veces que se dice cada frase donde el color celeste representa aciertos y el color violeta representa errores.

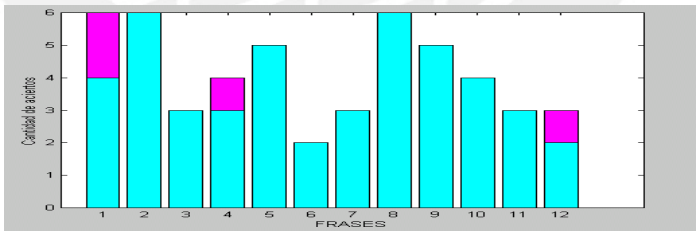
a) 77 – 81dBC en el sonómetro. Eficiencia: **100%.**



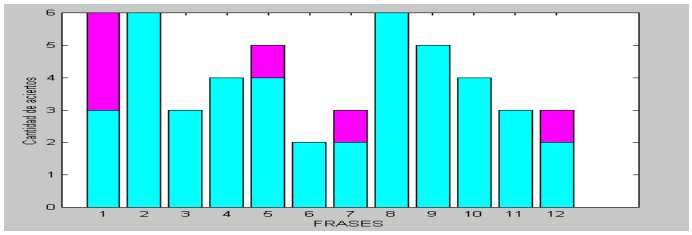
b) 79 – 82dBC en el sonómetro. Eficiencia: **100%.**



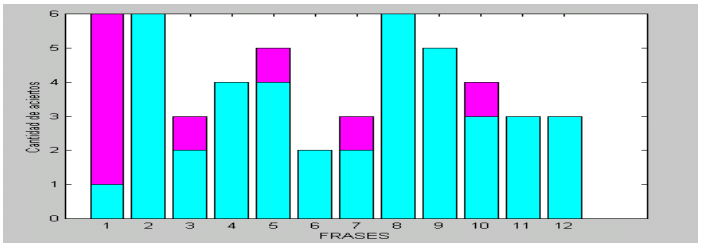
c) 80 – 84dBC en el sonómetro. Eficiencia: **92%.**



d) 82 – 87dBC en el sonómetro. Eficiencia: **88%.**



e) 84 – 88dBC en el sonómetro. Eficiencia: **80%.**



Para mayor volumen se asume que la curva cae rápidamente.

La gráfica 4.4 muestra la curva eficiencia (%) vs. sonido del equipo (dBC) obtenida en esta prueba en color azul y la curva que se obtuvo sin el filtro en color verde.

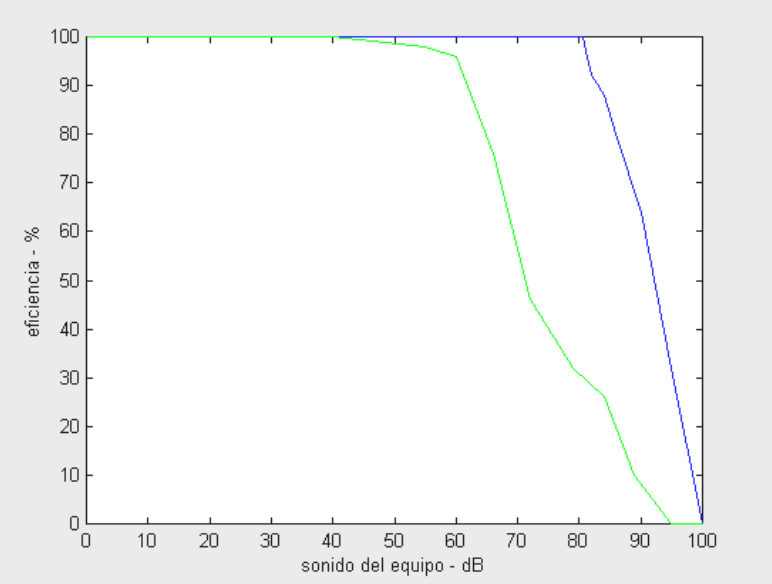


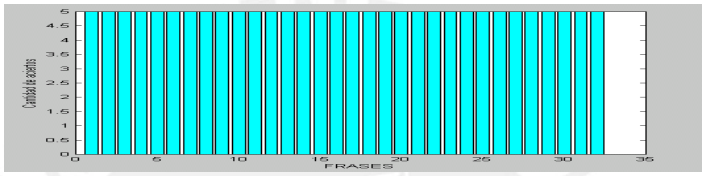
FIG 4.4 Eficiencia vs. Sonido del equipo usando el filtro

El filtro ha logrado que el algoritmo sea de 'buena' calidad (95% de aciertos) hasta con 81dBC de ruido es decir claramente se puede apreciar el grado en que se mejora la eficiencia, confirmando que el algoritmo 'no fallaría' (99.9% de aciertos) hasta con 80dBC como límite y 77dBC con un margen de seguridad.

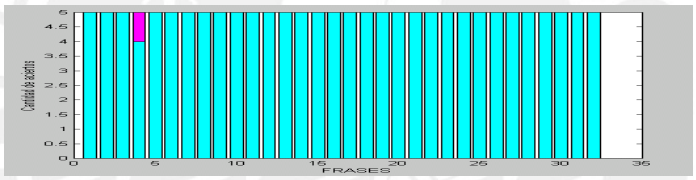
4.2 PRUEBAS DE RECONOCIMIENTO EN LA TARJETA DE DESARROLLO TMS320C6711

En las pruebas de reconocimiento en *matlab* se utilizaron sólo las 12 primeras órdenes de la lista de 32 por simplicidad. En esta sección se realizan para todas las órdenes (lista del **anexo E**) con 5 pruebas por cada una para cada volumen en el equipo. **Las barras celestes son la cantidad de aciertos y las violetas son la cantidad de errores:**

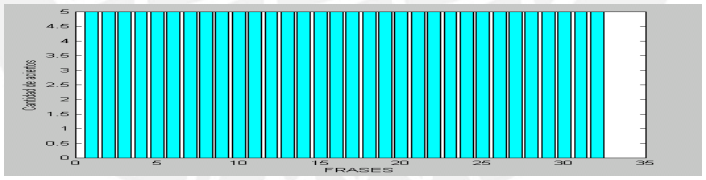
a) 63-72dBC en el sonómetro y -36dB de referencia en el equipo. **Eficiencia:** 100%



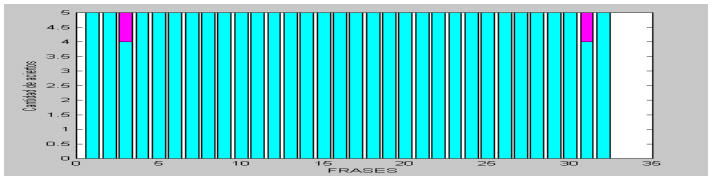
b) 65-74dBC en el sonómetro y -34dB de referencia en el equipo. **Eficiencia:** 99.37%



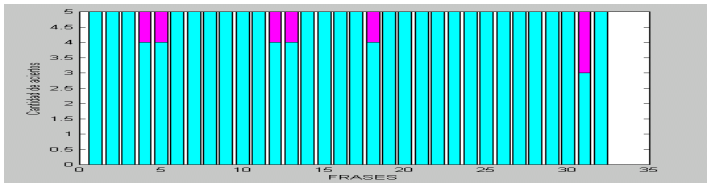
c) 69-78dBC en el sonómetro y -32dB de referencia en el equipo. . **Eficiencia:** 100%



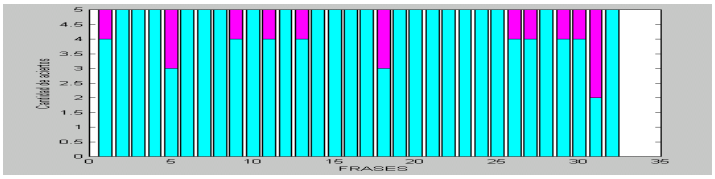
d) 73-80dBC en el sonómetro y -30dB de referencia en el equipo. . **Eficiencia:** 98.75%



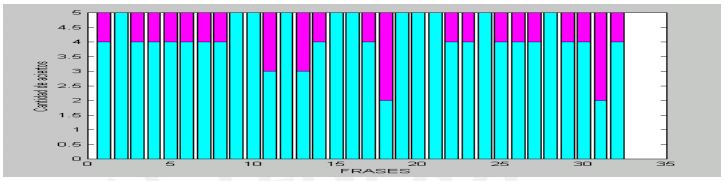
e) 77-81dBC en el sonómetro y -28dB de referencia en el equipo. . **Eficiencia:** 95.62%



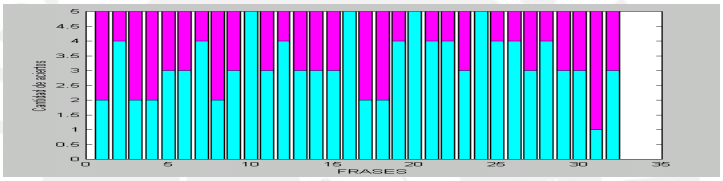
f) 79-82dBC en el sonómetro y -26dB de referencia en el equipo. . **Eficiencia:** 91.87%



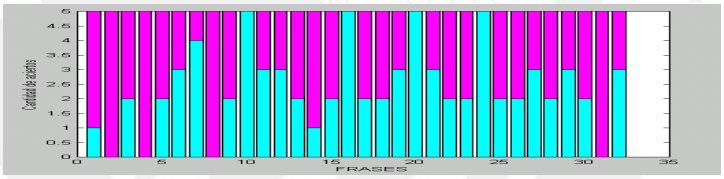
g) 80-84dBC en el sonómetro y -24dB de referencia en el equipo. **Eficiencia:** 83.12%



h) 82-87dBC en el sonómetro y -22dB de referencia en el equipo. **Eficiencia:** 66.87%



i) 83-88dBC en el sonómetro y -20dB de referencia en el equipo. **Eficiencia:** 47.50%



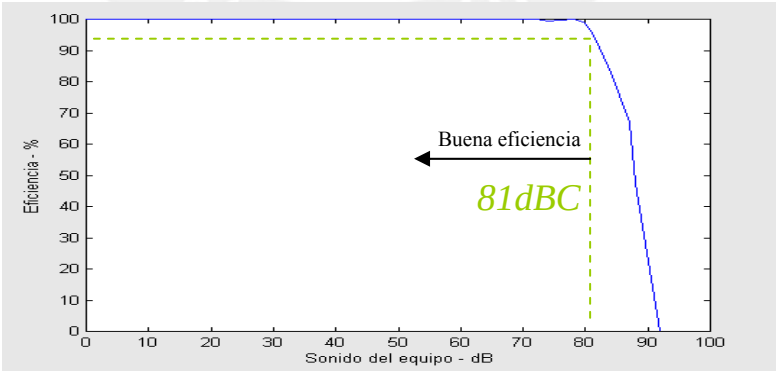


FIGURA 4.5 Curva Eficiencia vs. Sonido del equipo

Se ha obtenido una buena eficiencia hasta con **81dBC** de ruido (medido en el sonómetro) y volumen de referencia de -28dB a 2.5m de distancia de los parlantes.

5. OBSERVACIONES Y CONCLUSIONES.

5.1 CONCLUSIONES.

1.- Con ayuda del filtro el sistema de reconocimiento es de buena eficiencia hasta con 81dBC de ruido y es óptimo (>99%) hasta 77dBC para una sola voz y con entrenamiento de 10 repeticiones por frase.

2.- Se ha conseguido 100% de eficiencia con el mínimo ruido (ambiental).

3.- Una orden es reconocible solo hasta cierto de nivel de ruido, y este límite es diferente para cada una de ellas, dependiendo del tipo de ruido que haya en el ambiente.

4.- La curva de eficiencia presenta una mayor pendiente con el uso del filtro que sin usarlo porque en cierto momento el filtro LMS se desestabiliza debido a que la salida depende de la entrada (el ruido de referencia).

5.- La máxima reducción de ruido se produce cuando los dos micrófonos están juntos uno al lado del otro (función de transferencia aproximadas), pero en este caso no es posible porque se desea ingresar una señal diferente (voz) por uno de ellos y por lo tanto se atenuaría también.

6.- Una distancia adecuada entre el micrófono de referencia y los parlantes es entre 50 y 70cms. No se deben acercar demasiado pues los coeficientes del filtro se pueden elevar sin control produciendo desestabilización y por lo tanto un incorrecto funcionamiento.

7.- Durante la simulación en *Matlab* se determinó que a mayor número de órdenes disminuye la eficiencia y aumenta el tiempo de procesamiento.

8.- La señal infrarroja se envía luego de aproximadamente 1.8 segundos; tiempo en el cual se procesa la voz y se ejecuta el algoritmo de viterbi.

5.2 OBSERVACIONES.

1.- Es necesario hacer un nuevo entrenamiento al usar el filtro, debido a que la señal de voz se transforma en el camino por la nueva función de transferencia que representa.

2.- Los modelos de markov se crearon en el entrenamiento usando *Matlab* y fueron almacenados en archivos con extensión '.mat'. Luego se crearon tres programas (para A, mu y Sigma) que transformaban estos archivos en listados del cual se tomaban los valores manualmente para crear archivos con extensión (.dat) utilizando el *block de notas* de Windows; estos archivos tienen un encabezado indicando la dirección de inicio, la longitud del bloque y el número de *bytes* del tipo de dato. Los archivos se cargaron en la memoria usando el *Code Composer Studio* para luego generar un solo archivo con extensión (.out)

3.- Se hizo una prueba con un micrófono inalámbrico, sin embargo la eficiencia cayó rápidamente porque se introducen nuevas interferencias, retardos en la transmisión y una función de transferencia no considerada durante el entrenamiento.

4.- Las ganancias de los conversores A/D se deben regular para lograr un resultado similar a la ganancia en el entrenamiento, porque son diferentes sistemas (tarjeta de sonido de la PC y tarjeta DSK).

5.- Es importante indicar que el sistema es de prueba y no es un producto final.

6. RECOMENDACIONES.

1.- Recordar que en cualquier sistema de reconocimiento de voz o de locutor, se debe hacer el entrenamiento usando el filtro y además no olvidar que se deben regular las ganancias para lograr una amplitud similar.

2.- Al dar las órdenes se debe hablar a una distancia no mayor de 5cm porque el sistema detecta que hay voz utilizando un valor umbral constante. Si se habla a mayor distancia la orden no ingresará al sistema de reconocimiento completa y además con poca energía lo que supondría una menor eficiencia.

3.- Subir el volumen a un nivel de ruido elevado, puede resultar incómodo porque tal vez no se reconozca la orden para bajarlo nuevamente.

4.- La memoria flash de la tarjeta DSK ha sido utilizada casi en su totalidad por lo tanto, para proyectos futuros y más extensos, se debe utilizar una memoria ROM externa.

5.- Es importante lograr una eficiencia de 100% sin ruido, para este objetivo se debe hacer un entrenamiento de 10 repeticiones mínimo por cada persona y utilizar el mayor número de orden haciendo un balance con el tiempo de procesamiento.

6.- Es necesario utilizar micrófonos similares para lograr la máxima reducción posible del ruido.

7.- Si se va a desarrollar un sistema basado en procesamiento de señales, se recomienda el uso de *Matlab*, porque es muy versatil y permite un manejo adecuado de los datos de punto flotante en forma de matrices y además en forma gráfica, siendo una muy buena herramienta para probar un sistema antes de su implementación en una tarjeta DSP. De la misma manera el uso de *Cool Edit* para grabar voces y para su análisis en frecuencia.

8.- Para proyectos futuros se debe considerar el uso de una memoria ROM de mayor capacidad para que el sistema pueda ser programable por los usuarios, o incluso para que sea independiente del locutor. Además se podrìa crear un control remoto de voz universal que sirva para cualquier equipo audiovisual.

9.- Debido a que las tarjetas utilizadas en este proyecto son de desarrollo y de evaluación, un tema de tésis podría ser la integración de las tarjetas para lograr un producto final, utilizando solo los componentes necesarios y de esta manera reducir los costos que incluso permitirían su comercialización.

BIBLIOGRAFÍA

[1] Sen M. Kuo, Dennis R. Morgan: “Active noise control systems”, Wiley 1996.

[2] L Rabiner: "Fundamentals of Speech Recognition", Prentice Hall. 1993

[3] J.Deller, J.Hansen, J. Proakis: "Discrete-Time Processing of Speech Signals" , IEEE Press. 2000

[4] Olivier Cappé: “H2M: functions for the EM estimation of mixtures and hidden Markov models”, August 24, 2001. 46 rue Barrault, 75634 Paris cedex 13, France. (http://tsi.enst.fr/˜cappe/mfiles/h2m.tgz).

[5] Yuri Ivanov: “Hidden Markov Models Toolbox”, MIT Media Lab 1999 (yivanov@media.mit.edu)

[6] Tapas Kanungo: “Hidden Markov Models” (.pdf), Center for automation research, University of Maryland. (www.cfar.umd.edu/˜kanungo).

[7] Ondracka J., Oravec R., Kadlec J., cocherová E., “Simulation of RLS and LMS algorithms for adaptive noise cancellation in Matlab”. Department of radioelectronics, FEI STU Bratislava, Slovac Republic.

[8] “TMS320C6000 Assembly Language Tools User's Guide”, SPRU186 de Texas Instruments.

[9] “TMS320C6000 Programmer's Guide”, SPRU198 de Texas Instruments.

[10] “TMS320C62x DSP Library Programmer's Reference”, SPRU402 de Texas Instruments.

[11] “TMS320C6000 CPU and Instruction Set Reference Guide”, SPRU189 de Texas Instruments.

[12] “TMS320C6000 Peripherals Reference Guide”, SPRU190 de Texas Instruments.

[13] “TMS320C6000 Tools: Vector Table and Boot ROM Creation”, SPRA544 de Texas Instruments.

[14] “TCL320AD535C/I Data manual”, SLAS202B de Texas Instruments.

[15] “Code Composer Studio User’s Guide”, SPRU328 de Texas Instruments.

[16] "c6211/c6711 dsk hardware installation quick start", SPRU436 de Texas Instruments.

[17] "TMS320C6000 Code Composer Studio Help", Texas Instruments.

[18] "DSP56002 24-bit Digital Signal Processor user's manual", 56002UM1 de Motorola.

[19] "DSP56002 Semiconductor technical data", 56002DS3 de Motorola.

[20] "DSP56002EVM board –schematics", 002EVMSC de Motorola.

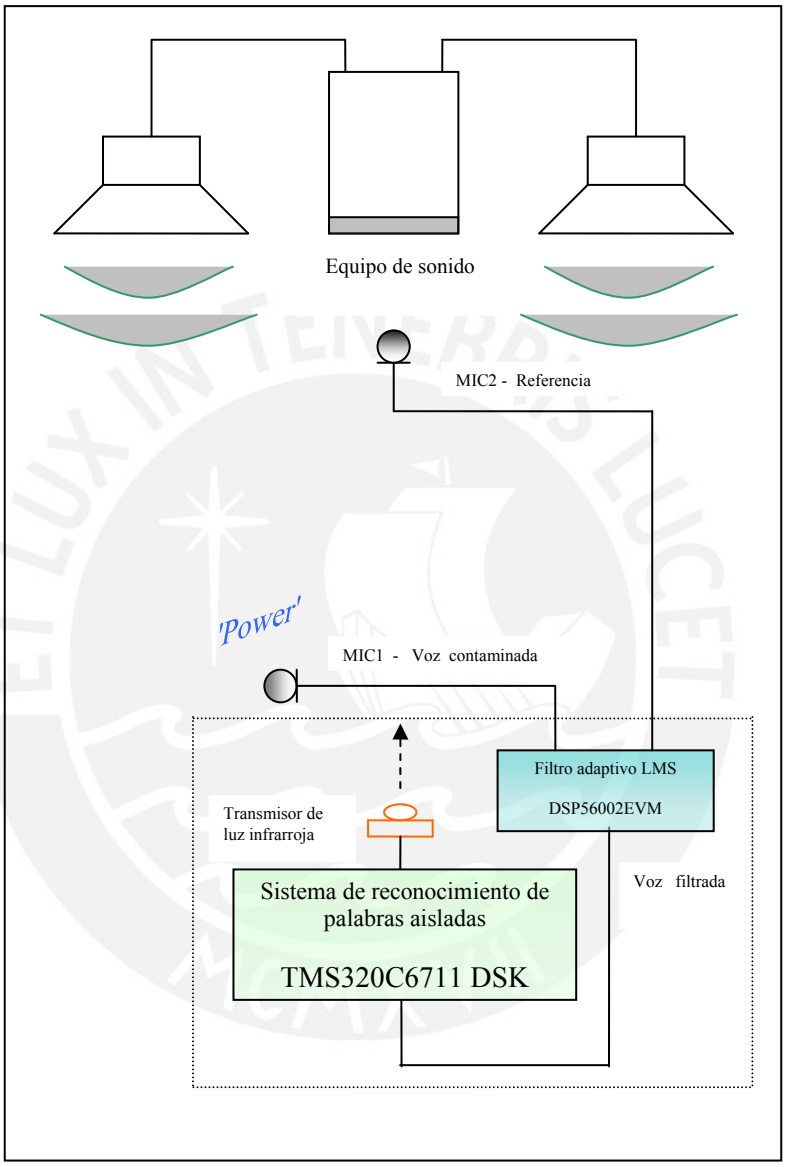
[21] Cool Edit 96 (demonstration version), Syntrillium software corporation. PO box 62255, Phoenix-USA.

[22] Jorge Lam Wong, Abel Luis Mellado Ochoa, "Diseño e implementación de un sistema de reconocimiento de locutor independiente del texto usando modelos ocultos de Harkov y cuantificación vectorial de ACW". PUCP-2001 Lima-Perú.

[23] C. San Román, "Sistema verificador de locutor de texto variable basado en concadenación de modelos fonéticos. PUCP-2003 Lima-Perú.

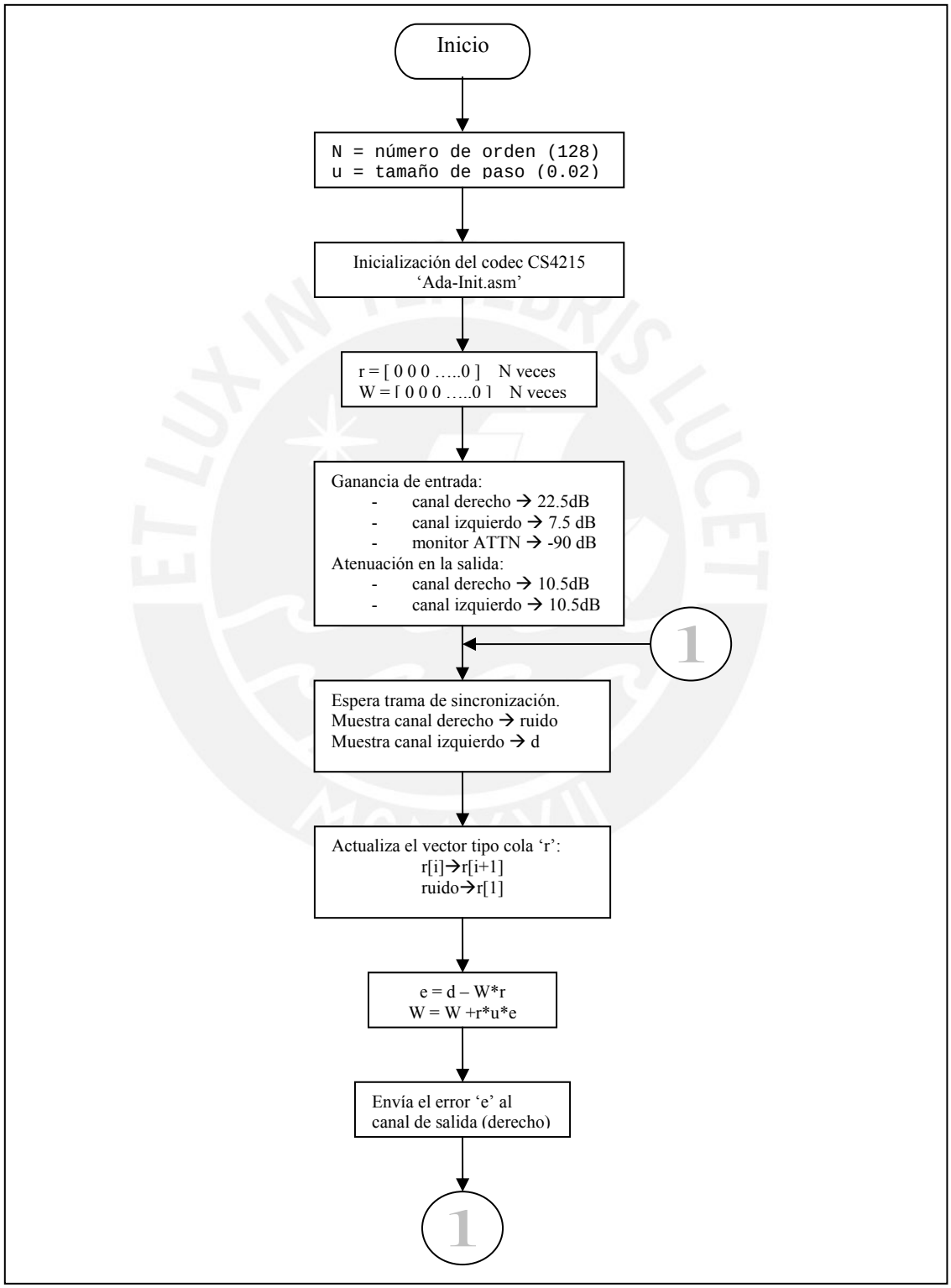
Anexo A

DIAGRAMA DE CONEXIONES DEL SISTEMA



Anexo B

DIAGRAMA DE FLUJO DEL FILTRO ADAPTIVO LMS



Anexo C

PROGRAMA ELABORADO EN MATLAB DEL FILTRO LMS PARA LA PRUEBA DE FILTRADO DE RUIDO

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                       %%
%%   Filtro adaptivo LMS - Aplicación    %%
%%                                       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
L=16000;
[voz,Fs]=wavread('studio92');   % archivo de voz a 8000Hz a 16 bits
frase(1:L)=voz(1:L);            % 16000 muestras
frase=frase(:);

musica=wavread('musica_radio'); % archivo de música a 8000Hz a 16 bits
ruido(1:L)=musica(1:L)/2;       % 16000 muestras
ruido=ruido(:);

rm=ruido/3;                     % creación de la voz contaminada
planta=[0.5 0.4 0.1];           %
rm=filter(planta,1,rm);         %
d=frase+rm;                     %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%   LMS   %%%%%%%%%%

N=length(planta);               % Número de orden del filtro
u=0.2;                          % tamaño de paso

W = zeros(N,1);                 % coeficientes del filtro
ruido = [zeros(N-1,1); ruido];  % ventana de la referencia (música)
i = 1;
while (i<L+1)
   x = ruido(i+N-1:-1:i);
   e(i)= d(i) - W'*x;
   W=W+u*x*e(i);
   i=i+1;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1)
y=0.5;
subplot(4,1,1);plot(frase,'b');axis([0 L -y y])  % gráfico de la voz pura
subplot(4,1,2);plot(ruido,'b');axis([0 L -y y])  % gráfico del ruido
subplot(4,1,3);plot(d,'b');axis([0 L -y y])      % gráfico de la voz
contaminada
subplot(4,1,4);plot(e,'b');axis([0 L -y y])      % gráfico de la voz filtrada
```

Anexo D

PROGRAMA DEL FILTRO ADAPTIVO LMS IMPLEMENTADO EN LA TARJETA DSP56002EVM

LMS.asm (programa principal)

```
page    132,60
;*********************************************************
;*                FILTRO DIGITAL ADAPTIVO LMS
;*
;* Entradas en LINE IN
;* canal izquierdo (voz+ruido)
;* canal derecho (ruido)
;*
;* Salida en LINE OUT
;* canal izquierdo (voz filtrada)
;* canal derecho (voz filtrada)
;*
;* Nota: se hace uso de las siguientes rutinas de MOTOROLA
;*       'ada_init.asm'     inicializacion del codec CS4215
;*       'txrx_isr.asm'     Rutina de interrupcion de TX y RX de datos
;*
;*              ALGORITMO:
;*                        e = d - W'*r
;*                        W = W + u*e*r
;*
;*              GIOVANI SAID SIMON BENDEZU   PUCP-2003
;*********************************************************
N           equ    128        ;  Numero de orden
u           equ    0.02       ;  tamaño de paso de descenso

r           equ    $1000      ;  r              Nx1        ventana de referencia con desplazamiento de 1
W           equ    $1100      ;  W              Nx1        vector de coeficientes del filtro
rue         equ    $1200      ;  u*e*r          Nx1
d           equ    $500       ;  d              real       voz contaminada
e           equ    $510       ;  e              real       error, voz filtrada
ruido       equ    $520       ;  ruido          real       referencia
W1r         equ    $530       ;  W'r            real
ue          equ    $540       ;  u*e            real

START   equ    $40

            org    p:0                ; Vector de Reset
            jmp    INICIO

            org    p:$000C
            jsr    ssi_rx_isr         ;SSI RX
            jsr    ssi_rx_isr         ;SSI RX  con excepción
            jsr    ssi_tx_isr         ;SSI TX
            jsr    ssi_tx_isr         ;SSI TX  con excepción
;*************************************************************************
;     inicialización del codificador CS-4215

            org    p:INICIO

      move   #r,r0              ; inicializa r=[0 0..0]  N veces
      move   #0,x0
      do     #N,_fin1a
      move          x0,x:(r0)+
_fin1a
;---------------------------------------------------------------------
      move   #W,r0              ; inicializa W=[0 0..0]  N veces
      move   #0,x0
```

```
 do       #N,_fin1b
 move             x0,x:(r0)+
_fin1b
;---------------------------------------------------------------------------
              movep  #$261009,x:PLL        ;indica que PLL es de 10x para MPY
              movep  #$0000,x:BCR          ; sin estados de espera para todas las memorias externas
              ori    #3,mr                 ; desabilita interrupciones
              movec  #0,sp                 ; limpia el puntero de la pila
              move   #0,omr                ;modo 0: habilita interrupciones. P:RAM, rst=0000
              move   #$40,r6               ; inicializa puntero de pila
              move   #-1,m6                ; direccionamiento lineal

      include 'ada_init.asm'

; habilitar micrófonos, 22.5dB de ganacia en el canal derecho (ruido), ganancia de 7.5dB en el canal
; izquierdo (voz contaminada), habilita audífono y line out con atenuación de salida de 10.5dB

TONE_OUTPUT    EQU    HEADPHONE_EN+LINEOUT_EN+(7*RIGHT_ATTN)+(7*LEFT_ATTN)
TONE_INPUT     EQU    MIC_IN_SELECT+(15*MONITOR_ATTN)+(15*RIGHT_GAIN)+(5*LEFT_GAIN)

            move   #TONE_OUTPUT,y0       .
              move         y0,x:TX_BUFF_BASE+2
              move   #TONE_INPUT,y0
              move         y0,x:TX_BUFF_BASE+3

      include 'vma'           ; macro que multiplica 2 vectores dando como respuesta la suma.

lazo
        jset   #2,x:SSISR,*          ; Espera trama de sincronizacion
        jclr   #2,x:SSISR,*          ; Espera trama de sincronizacion
                                            ; obtencion de nuevas muestras
      move   x:RX_BUFF_BASE,a1       ; canal izquierdo (RX voz+ruido)
      move   x:RX_BUFF_BASE+1,b1     ; canal derecho (RX ruido)

      move   a1,x:d
      move   b1,x:ruido
;---------------------------------------------------------------------------
                           ; ACTUALIZA MATRIZ r
      move   #r+N-2,a0             ; r0 = ultimo-1 = (r+N-1)-1
      move   a0,r0
      inc    a                     ; r1 = ultimo = r+N-1 = r0+1
      move   a0,r1

      do     #N-1,_fin1
      move   x:(r0)-,x0            ; rota la cola de la matriz r
      move   x0,x:(r1)-
_fin1

      move   x:ruido,a1
      move   a1,x:r                ; el primero, nuevo valor de la muestra ingresante
;---------------------------------------------------------------------------
                           ; W1r = W' * r
      move   #W,a0
      move   a0,x:($900)
      move   #r,x0
      move   x0,x:($901)
      move   #W1r,b0
      move   b0,x:($902)
      vma    N
;---------------------------------------------------------------------------
                           ; e = d - W'*r
      clr    a
      clr    b
      move   x:d,a0
      move   x:W1r,b0
      sub    b,a
      move   a0,x:e
;---------------------------------------------------------------------------
                           ; ue = u * e
```

```
        clr  a
        move   #u,x0
        move   x:e,y0
        mpy    x0,y0,a
        move   a,x:ue
;-------------------------------------------------------------------------
                        ; rue = r * ue
        move   #r,r0
        move   x:ue,y0
        move   #rue,r1
        move   x:(r0)+,x0

        do      #N,_fin2
        clr     a
        mpy     x0,y0,a      x:(r0)+,x0
        move    a,x:(r1)+
_fin2
;-------------------------------------------------------------------------
                          ; W = W + rue
        move   #W,r0
        move   #rue,r1
        move   #W,r4

        do    #N,_fin3
        clr    a
        clr    b
        move   x:(r0)+,a0
        move   x:(r1)+,b0
        add    b,a
        move   a0,x:(r4)+
_fin3
;-------------------------------------------------------------------------
        move   x:e,a1
        move   a1,x:TX_BUFF_BASE      ; canal izquierdo (TX voz filtrada).
        move   a1,x:TX_BUFF_BASE+1    ; canal derecho (TX voz filtrada).
        jmp    lazo                   ; vuelve a empezar el lazo infinito

        include 'txrx_isr.asm'
        end
```

VMA.asm

```
        page    132,60
;*********************************************************
;
; Vma.asm multiplicacion de 2 vectores de num. reales
;
; resultado = sum [ vect1(i) * vect2(i) ]     i=1..n
;
; n =  numero de multiplicaciones
; base de vect1 en x:($900)
; base de vect2 en x:($901)
; resultado en x:($902)
;
;*********************************************************
vma         macro     n

          move      x:($900),r0
          move      x:($901),r4
          move      x:($902),r1
          move      x:(r0)+,x0
          move      x:(r4)+,y0
          clr  a
          do          #n,_finmult
          clr         b
          mpy         x0,y0,b       x:(r0)+,x0
          move        x:(r4)+,y0
          move        b1,b0
          add         b,a
```

```
_finmult
        move    a0,x:(r1)

        endm
```

Anexo E

FUNCIONES IMPLEMENTADAS PARA EL CONTROL DEL EQUIPO PANASONIC SC-AK45

	FRASES	FUNCIÓN
1	**TELESTEREO**	Nombre de la emisora en la banda 88.3FM.
2	**RPP**	Nombre de la emisora en la banda 89.7FM.
3	**OK**	Nombre de la emisora en la banda 91.9FM.
4	**STUDIO 92**	Nombre de la emisora en la banda 92.5FM.
5	**RADIO AMERICA**	Nombre de la emisora en la banda 94.3FM.
6	**ZETA**	Nombre de la emisora en la banda 95.5FM.
7	**MIRAFLORES**	Nombre de la emisora en la banda 96.1FM.
8	**DOBLE 9**	Nombre de la emisora en la banda 99.1FM.
9	**STEREO 100**	Nombre de la emisora en la banda 100.1FM.
10	**PANAMERICANA**	Nombre de la emisora en la banda 101.1FM.
11	**RADIOMAR**	Nombre de la emisora en la banda 106.3FM.
12	**PLANETA**	Nombre de la emisora en la banda 107.7FM.
13	**POWER**	Enciende / apaga el equipo.
14	**WOOFER**	Cambia hasta que haya presencia de voz (sin woofer, medio, máximo).
15	**VOLUMEN**	Sube el volumen hasta que haya presencia de voz nuevamente.
16	**BAJA**	Baja el volumen hasta que haya presencia de voz nuevamente.
17	**EQ**	Cambia el ecualizador hasta que haya presencia de voz nuevamente.
18	**AUXILIAR**	Coloca el selector en modo auxiliar (AUX).
19	**DISCOS**	Coloca el selector en modo tocador de discos compactos (CD).
20	**EMISORAS**	Coloca el selector en modo receptor de emisoras (TUNER FM-MW-SW)
21	**SILENCIO**	Baja totalmente el volumen / vuelve al volumen original.
22	**BUSCAR**	Cambia de emisoras en forma secuencial cada 3 segundos hasta que haya presencia de voz
23	**PRIMERO**	Activa el disco compacto ‘1’.
24	**SEGUNDO**	Activa el disco compacto ‘2’.
25	**TERCERO**	Activa el disco compacto ‘3’.
26	**CUARTO**	Activa el disco compacto ‘4’.
27	**QUINTO**	Activa el disco compacto ‘5’.
28	**PISTA**	Cambia de pista sucesivamente hasta que haya presencia de voz.
29	**PLAY**	Reproduce lel sonido de la pista actual
30	**PAUSA**	Detiene la reproducción de sonido.
31	**SIGUIENTE**	Selecciona siguiente pista del disco compacto actual.
32	**ANTERIOR**	Selecciona pista anterior del disco compacto actual.

Anexo F

PROGRAMA ELABORADO EN MATLAB PARA EL ENTRENAMIENTO DE LAS PALABRAS AISLADAS

ENTRENAMIENTO.m

```
%%%%%%%%%%%%%%%%%%%
%  ENTRENAMIENTO  %
%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROGRAMA QUE HALLA EL MODELO A,mu,sigma DE CADA %
% FRASE PARA EL CONTROL DEL EQUIPO POR MEDIO      %
% DE LA VOZ  (SEÑAL de 16 bits a 8000hz)          %
%                                                 %
%          Giovani Said Simon Bendezu             %
%                                                 %
%                  PUCP - 2003                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

% Parametros de analisis
mp = 256;               % Tamaño de ventana
dp = 170;               % desplazamiento de ventana
p = 10;                 % Orden del Cepstrum

% Parametros de entrenamiento
Npal = 32;          % Numero de palabras
Nent = 10;          % Numero de expresiones para entrenamiento
Nest = 20;          % Numero de estados por modelo de palabra

NIT = 20;           % Numeros de iteraciones EM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parametrizacion (Coeficientes cepstrales de¨Potencia estandar). %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(1, 'Procesando la informacion...\n');

ord1=wavread('secuencia1');
ord2=wavread('secuencia2');
ord3=wavread('secuencia3');
ord4=wavread('secuencia4');
ord5=wavread('secuencia5');
ord6=wavread('secuencia6');
ord7=wavread('secuencia7');
ord8=wavread('secuencia8');
ord9=wavread('secuencia9');
ord10=wavread('secuencia10');

l1=length(ord1);l2=length(ord2);l3=length(ord3);l4=length(ord4);l5=length(ord5);
l6=length(ord6);l7=length(ord7);l8=length(ord8);l9=length(ord9);l10=length(ord10);
ord=[ord1; ord2; ord3; ord4; ord5; ord6; ord7; ord8; ord9; ord10];
l=[0 l1 l1+l2 l1+l2+l3 l1+l2+l3+l4 l1+l2+l3+l4+l5 l1+l2+l3+l4+l5+l6 l1+l2+l3+l4+l5+l6+l7
l1+l2+l3+l4+l5+l6+l7+l8 l1+l2+l3+l4+l5+l6+l7+l8+l9];

d(1:Npal,1:Nent*2)=0;
d(1:Npal,1:2)=limites(ord1);
d(1:Npal,3:4)=limites(ord2);
d(1:Npal,5:6)=limites(ord3);
d(1:Npal,7:8)=limites(ord4);
d(1:Npal,9:10)=limites(ord5);
d(1:Npal,11:12)=limites(ord6);
d(1:Npal,13:14)=limites(ord7);
d(1:Npal,15:16)=limites(ord8);
d(1:Npal,17:18)=limites(ord9);
```

```
d(1:Npal,19:20)=limites(ord10);

n(1:Npal,1:Nent*2)=0;
for j=1:Npal
   for k=1:Nent
      for z=1:2
         n(j,2*(k-1)+z)=fix((d(j,2*(k-1)+z)+l(k)-mp)/dp);
      end;
   end;
end;

fprintf(1, 'Calculando mfcc de las ordenes: \n');
C=(mfcc(ord,dp,mp,p))';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Entrenamiento HMM utilizando mfcc %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(1, 'Modelo de entrenamiento para orden:');
for w = 1:Npal
   A = sparse(0.3333*diag(ones(1,Nest))+0.3333*diag(ones(1,Nest-1),1)+
    0.3333*diag(ones(2,Nest-2),2));
   A(Nest,Nest) = 1;
   fprintf(1, ' %d', w);
    X=[];
    st=[];
    for k=1:Nent
       st = [st; size(X,1)+1];
       X = [X ;  C(n(w,(2*k)-1):n(w,2*k),:)];
    end;
    T=size(X,1);

    [mu,Sigma] = hmm_ini(X, st, Nest);
    Sigma = ones(Nest,1)*mean(Sigma);
    logl = zeros(1, NIT);
    for k = 1:NIT
      [A, logl(k), gamma] = hmm_reest(X, st, A, mu, Sigma);
      [mu, Sigma] = mix_par(X, gamma);
      Sigma = ones(Nest,1)*(sum((sum(gamma)'*ones(1,p)).*Sigma)/T);
    end
    sigma = Sigma(1,:);
    eval(['save modelo_frase_' int2str(w) ' A mu sigma']);
end
fprintf(1,'\n');
```

LIMITES.m

```
function  posiciones = limites(voz)

% Halla las posiciones de inicio y fin de cada palabra
% en un archivo de voz de w palabras
y=voz.*voz;
L=length(y);
umbral=0.003;
i=0;
cont=0;
flag=0;
flag_cont=1;
for j=1:L
   if (flag==0)&(y(j)>umbral)
      flag_cont=0;
      flag=1;
      if cont>3500
         i=i+1
         pos_ini(i)=j-100;
      end
   end
   if (flag==1)&(y(j)<umbral)
      flag_cont=1;
      flag=0;
      pos_fin(i)=j+100;
```

```
   end
   if flag_cont==1
      cont=cont+1;
   else
      cont=0;
   end
end
posiciones = [pos_ini' pos_fin'];
```

MFCC.m

```
function ccep=mfcc(y,M,N,P);

% Calcula coeficientes cepstrales
% de la secuencia de muestras y
% N = tamaño de ventana
% M = tamaño de salto entre ventanas
% P = orden del cepstrum

NYQ=N/2;
% Filtro Triangular
for i=1:10
  fi(i)=(2*i)-1;
end;

fc=fi+2;
fs=fi+4;
fi(11)=23;fi(12)=27;fi(13)=31;fi(14)=35;fi(15)=40;fi(16)=46;fi(17)=53;
fi(18)=61;fi(19)=70;fi(20)=81;
fc(11)=27;fc(12)=31;fc(13)=35;fc(14)=40;fc(15)=46;fc(16)=53;fc(17)=61;
fc(18)=70;fc(19)=81;fc(20)=93;
fs(11)=31;fs(12)=35;fs(13)=40;fs(14)=46;fs(15)=53;fs(16)=61;fs(17)=70;
fs(18)=81;fs(19)=93;fs(20)=108;

long_trama=size(y,1);
m=0;
ini=1;
fin=N;
m=1;
while fin<=long_trama
  x(1)=y(ini);                          % Pre-enfasis
  for i=2:N
     x(i)=y(ini-1+i)-.95*y(ini-2+i);
  end
  ventana=hamming(N).*x(1:N)';          % Hamming
  magspec=abs(fft(ventana));
  for i=1:20
    for j=fi(i):fc(i)
       filtmag(j)=(j-fi(i))/(fc(i)-fi(i));
    end;
    for j=fc(i)+1:fs(i)
       filtmag(j)=1-(j-fc(i))/(fs(i)-fc(i));
    end;
    Y(i)=sum(magspec(fi(i):fs(i)).*filtmag(fi(i):fs(i))');
  end;

  Y=log(Y.^2);                          % Cepstrum

  for i=1:P
    coefwin=cos((pi/20)*i*(linspace(1,20,20)-0.5))';
    ccep(i,m)=sum(coefwin.*Y');
  end;

  m=m+1;
  ini=1+(m-1)*M;
  fin=ini+N-1;
end;
```

HMM_INI.m

```
function [mu, Sigma] = hmm_ini (X, st, N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Halla el modelo inicial de B (mu y sigma) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[KT, p] = size(X);
K = length(st);
mu = zeros(N,p);
cont = zeros(1, N);
Sigma = zeros(N,p);

for i=1:K
  ini = st(i);
  if (i == K)
    fin = KT;
  else
    fin = st(i+1)-1;
  end
  seg = round([ini+(0:N-1)*(fin-ini)/N fin]);
  cont = cont + (seg(2:N+1)-seg(1:N)+1);
  for j=1:N
    mu(j,:) = mu(j,:) + sum(X(seg(j):seg(j+1),:));
    Sigma(j,:) = Sigma(j,:) + sum(X(seg(j):seg(j+1),:).^2);
  end
  mu = mu ./ (cont' * ones(1,p));
  Sigma = Sigma ./ (cont' * ones(1,p)) - mu.^2;
end
```

HMM_REEST.m

```
function [A_, logl, gamma] = hmm_reest (X, st, A, mu, Sigma)

N=length(A(1,:));
pi0=[1 zeros(1,N-1)];
[KT, nc]=size(X);
K=length(st);
A=sparse(A);
A_=sparse(zeros(N));
gamma=zeros(KT, N);
den=0;
logl=0;
for i=1:K
  ini=st(i);
  if (i==K)
    fin=KT;
  else
    fin=st(i+1)-1;
  end
  T=fin-ini+1;
  [alfa,beta,logl_tmp,dens] = hmm_fb (X(ini:fin,:),A,pi0,mu,Sigma,N);
  logl=logl+logl_tmp;
  for j=1:T-1
    A_i=alfa(j,:)'*(dens(j+1,:).*beta(j+1,:)).*A;
    A_i=A_i/sum(sum(A_i));
    A_=A_+A_i;
  end
  gamma(ini:fin,:)=alfa.*beta;
  gamma(ini:fin,:)=gamma(ini:fin,:)./(sum(gamma(ini:fin,:)')'*ones(1,N));
  den=den+sum(gamma(ini:ini+T-2,:));
end
A_= A_./(den'*ones(1,N));
```

HMM_FB.m

```
function [alfa, beta, logl, dens] = hmm_fb (X, A, pi0, mu, Sigma,N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Forward y Backward algorithm %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[T, Nc] = size(X);
dens = gauseval(X, mu, Sigma);
escala = [1 ; zeros(T-1, 1)];
alfa = zeros(T, N);
alfa(1,:) = pi0.*dens(1,:);
for i=2:T
  alfa(i,:) = (alfa(i-1,:) * A) .* dens(i,:);
  escala(i) = sum(alfa(i,:));
  alfa(i,:) = alfa(i,:)/escala(i);
end
logl = sum(log(escala));
beta = zeros(T, N);
beta(T, :) = ones(1, N);
for i=(T-1):-1:1
  beta(i,:) = (beta(i+1,:).* dens(i+1,:)) * A'/escala(i);
end
```

GAUSEVAL.m

```
function logdens = gauslogv(X, mu, Sigma)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Distribucion gausiana multi-variable %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[T, p] = size(X);
[N, Nc] = size(mu);
nm = prod(Sigma');
if (any(nm < realmin))
  error('Determinante es negativo o cero.');
else
  nm = 1 ./ sqrt((2*pi)^p * nm);
end

logdens = zeros(T, N);
for i=1:T
  for j=1:N
    logdens(i,j) = sum(((X(i,:)-mu(j,:)).*(X(i,:)-mu(j,:))) ./ Sigma(j,:));
  end
  logdens(i,:) = log(nm) -0.5 * logdens(i,:);
end
```

MIX_PAR.m

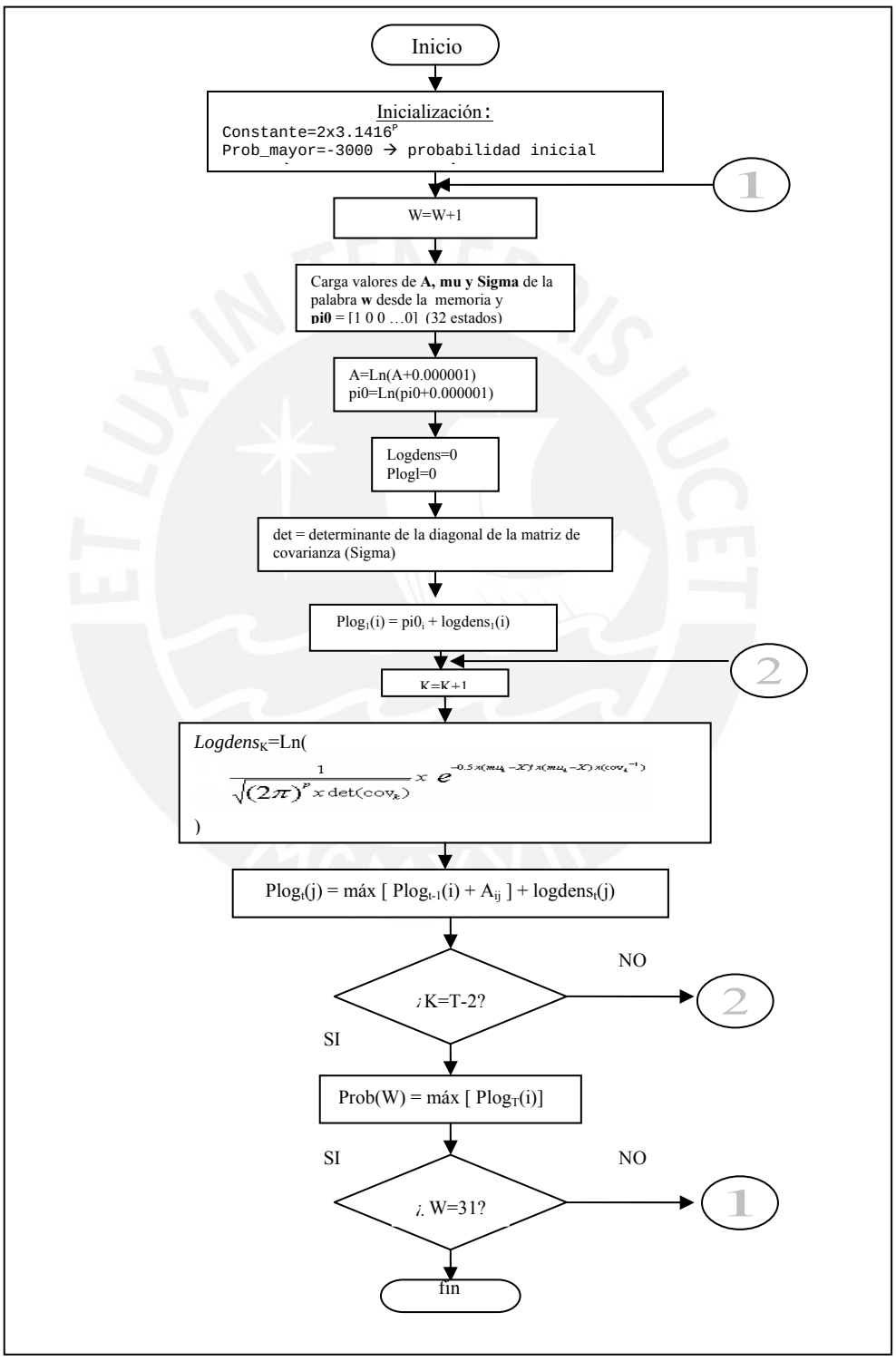
```
function [mu, Sigma] = mix_par (X, gamma)

% Reestimación de los parámetros de las mixturas

[T, N] = size(gamma);
p = length(X(1,:));
mu = zeros(N, p);
Sigma = zeros(N, p);
for i=1:N
  mu(i,:) = sum (X .* (gamma(:,i) * ones(1,p)));
  Sigma(i,:) = sum(X.^2 .* (gamma(:,i) * ones(1,p)));
end
mu = mu ./ (sum(gamma)' * ones(1,p));
Sigma = Sigma ./ (sum(gamma)' * ones(1,p));
Sigma = Sigma - mu.^2;
```

Anexo G

DIAGRAMA DE FLUJO DEL ALGORITMO DEL RECONOCIMIENTO DE PALABRAS AISLADAS: VITERBI



Anexo H

DISTRIBUCIÓN DE LA MEMORIA EN LA TARJETA DE DESARROLLLO TMS320C6711

Rango de Direcciones (Hexadecimal)	Tamaño (Bytes)	Descripción del Bloque de Memoria
0000 0000 - 0000 FFFF	64K	RAM Interna
0001 0000 - 017F FFFF	24M - 64K	Reservado
0180 0000 - 0183 FFFF	256K	Registros de configuración del EMIF
0184 0000 - 0187 FFFF	256K	Registros de control de Caché
0188 0000 - 018B FFFF	256K	Registro de configuración interna de bus HPI
018C 0000 - 018F FFFF	256K	Registros de configuración de McBSP0
0190 0000 - 0193 FFFF	256K	Registros de configuración de McBSP1
0194 0000 - 0197 FFFF	256K	Registros del timer 0
0198 0000 - 019B FFFF	256K	Registros del timer 1
019C 0000 - 019F FFFF	256K	Registros selectores de interrupciones
01A0 0000 - 01A3 FFFF	256K	Registros de configuración del EDMA
01A4 0000 - 1FFF FFFF	1G - 288M	Reservado
3000 0000 - 3FFF FFFF	256M	Data McBSP 0/1
4000 0000 - 7FFF FFFF	1G	Reservado
8000 0000 - 8FFF FFFF	256M	Espacio de memoria CE0
9000 0000 - 9FFF FFFF	256M	Espacio de memoria CE1
A000 0000 - AFFF FFFF	256M	Espacio de memoria CE2
B000 0000 - BFFF FFFF	256M	Espacio de memoria CE3
C000 0000 - FFFF FFFF	1G	Reservado

Anexo K

ESPECIFICACIONES TÉCNICAS DEL SONÓMETRO DIGITAL

Baterías	Alcalina de 9v.
Micrófono	Condensador eléctrico.
Rango	50dB a 126dB.
Exactitud	± 2dB en 114dB SPL.
Referencia	0dB = 0.0002 Micro Bar.
Curva de respuesta en frecuencia	A (500 – 10000Hz) Rango más sensitivo para el oído humano. C (32 - 10000Hz) Rango de medición de niveles de sonido para material musical.
Respuesta en el visualizador	Rápida o Lenta.
Señal de salida	
Voltaje	1vp-p min. (voltio pico – pico mínimo). Circuito abierto. Escala completa en 1kHz.
Impedancia	Mínima carga de 10 Kilohms.
Distorsión	Menor a 2% en 1kHz. Salida de 0.5 V p-p (Entrada: salida del micrófono, Salida: 10Kohm).
Temperatura de operación	32ªF a 122ªF.
Temperatura de almacenamiento	-40ªF a 149ªF.
Dimensiones	159 x 64 x 44mm.
Peso	165g aprox.

Anexo I

PROGRAMAS REALIZADOS EN LA TARJETA DE DESARROLLO TMS320C6711

CONTROL REMOTO DE VOZ.c

```
/*********************************************/
/*        CONTROL REMOTO DE VOZ.c            */
/*                                           */
/*   Sistema de audio Panasonic SC-AK45      */
/*                                           */
/*       Giovani Said Simón Bendezú          */
/*               PUCP-2003                   */
/*********************************************/
#include <c6x.h>
#include <math.h>

#include "c6711dsk.h"
#include "c6711interrupts.h"
#include "c6xdskdigio.h"

#define longDC 1600       // número de muestras utilizadas para el nivel DC.
#define longM 200         // longitud del buffer (tipo cola) de las muestras y Energía.
#define longvoz 12000     // número de muestras en 1.5s a 8kHz que se grabarán ante la.
                          // presencia de sonido (voz).

#define Nm 256            // número de muestras por ventana.
#define dp 170            // desplazamiento entre ventanas.
#define P 10              // número de orden del cepstrum.
#define BANCO 20          // número de filtros triangulares.

#define Npal 32           // número de frases o palabras entrenadas.
#define N 20              // número de estados.

#define luz 0x80160000    // portadora de la señal IR controlada en la interrupcion 10.
#define f_luz 0x80160004  // bandera de la portadora.

#define vit 0x80160008              // bandera que indica inicio del proceso de viterbi.
#define probabilidad 0x8016000c     // probabilidad de la frase que se ha procesado.
#define posicion_fin 0x80160010     // longitud de la frase a comparar.
#define orden 0x80160014            //
#define continuacion 0x8016001c

#define p_A  0x80200000     // tabla de los modelos de la matriz de probabilidad 'A'.
#define p_mu 0x8020c800     // tabla de los modelos 'mu'.
#define p_Sigma 0x80212c00  // tabla de los modelos de la covarianza diagonal 'Sigma'.

#define logdens 0x80240000  // float
#define plogl 0x80250000    // float
#define Prob 0x80260000     // float 32*4
#define Rel 0x80261000      // float 32*4
#define T 0x80270000        // short 1*2

#define voz 0x80140000      // float 12000*4 buffer de voz (1.5s - 12000 muestras)
#define bufferM 0x80150000  // float 200*4   buffer tipo cola, últimas muestras
#define bufferE 0x80150500  // float 200*4   buffer tipo cola, energía de las muestras

#define A 0x80150a00        // float 400*4  N*N*4 A de la palabra en proceso
#define mu 0x80151100       // float 200*4  P*N*4 mu de la palabra en proceso
#define Sigma 0x80151500    // float 200*4  P*N*4 Sigma de la palabra en proceso
#define pi0 0x80151900      // float 20*4   N*4

#define hamming 0x80151a00  // float 256*4  Nm*4
```

```
#define ventana 0x80151e00            // float 256*4  Nm*4
#define vent_preenfasis 0x80152200    // float 256*4  Nm*4
#define magspec 0x80152600            // float 256*4  Nm*4
#define Y 0x80152e00                  // float 20*4   BANCO*4
#define ccep 0x80152e50               // float 10*4   P*4
#define C 0x80152e80                  // float 1000*4 P*100(reserva)*4
#define filtmag 0x80153e20            // float 2160*4  BANCO*108*4
#define coefwin 0x80155fe0            // float 200*4   BANCO*P*4;
#define plA 0x80156300                // float 20*20*4  N*N*4
#define maximo 0x80156940             // float 20*4   N
#define nm 0x80156990                 // float 20*4   N

float i_o_real[Nm];       // parte real en la transformada de fourier
float i_o_complejo[Nm];   // parte imaginaria = 0
float fi[BANCO];          // inicio del filtro triangular
float fc[BANCO];          // frecuencia central
float fs[BANCO];          // fin del filtro


void mcbsp0_init();
int  mcbsp0_read();
void mcbsp0_write(int out_data);
void AD535_init();
void Interrupt12();

float VentanasSenoidales(int p, int banco);
float Preenfasis(int n);
float ValoresHamming(int n);
float VentanaHamming (int n, float i_o_real[], float i_o_complejo[]);
float CoeficientesCepstrales(float fi[], float fs[]);
void fft(int longitud, float i_o_real[], float i_o_complejo[]);
void magnitud(int longitud, float i_o_real[], float i_o_complejo[]);
void FiltrosTriangulares(float fi[], float fc[], float fs[]);

void hmm_vit(int Npalabras);

void timer0_setup(int clock_source, int output_mode, int timer_period);
void timer0_start(void);
void timer1_setup(int clock_source, int output_mode, int timer_period);
void timer1_start(void);
void delay_millisec(int delay);
void retardo(int delay,int valor);
void transmision_IR(int base);


volatile int sample;      // variable que contiene la muestra capturada 16 bits (int)

unsigned int CNTL0;       // salida LED IR
unsigned int CNTL1;       // salida LED verde (presencia de voz y viterbi)

float sumaE;              // suma de la energía total en el 'buffer' de energía


int main()
{
   int pos_ini,pos_fin,ini,fin,flag,long_trama,j,base_IR,accion;
   float E,umbral2;
   char transmite;

   *(short*)vit=0;
   *(short*)continuacion=0;
   *(unsigned volatile int *)IO_PORT = 0;  /* Enciende leds de la tarjeta */

  // DSP inicializacion
  CSR=0x100;                  /* Desabilita todas las interrupciones */
  IER=1;                      /* Desabilita todas las interrupciones excepto NMI   */
  ICR=0xffff;                 /* limpìa todas las interrupciones pendiente         */

  *(unsigned volatile int *)EMIF_GCR = 0x3300;    /* EMIF global control      */
  *(unsigned volatile int *)EMIF_CE1 = CE1_32;    /* EMIF CE1 control, 32bit  */
  *(unsigned volatile int *)EMIF_SDCTRL = 0x57227000; /* EMIF SDRAM control   */
```

```
*(unsigned volatile int *)EMIF_CE0 = 0x30;      /* EMIF CE0 control         */
*(unsigned volatile int *)EMIF_SDRP = 0x61a;    /* EMIF SDRM refresh period */
*(unsigned volatile int *)EMIF_SDEXT= 0x54529;  /* EMIF SDRM extension      */


 /*********************************/
 /*   DEFINICIONES PRELIMINARES   */
 /*********************************/
 ValoresHamming(Nm);             // ventana de hamming
 FiltrosTriangulares(fi,fc,fs);  // posición de los filtros dentro en el banco
 VentanasSenoidales(P, BANCO);   // 'ventanas' que se utilizará en el cepstrum


 /*********************************/
 /*  INICIALIZACIÓN DE BUFFERS    */
 /*********************************/
 for (j=0;j<longvoz;j++)  *(float*)(voz+j*4)=0;
 for (j=0;j<longM;j++)
 {
  *(float*)(bufferM+j*4)=0;
  *(float*)(bufferE+j*4)=0;
 }
 sumaE=0;
 accion=0;
 /*************************************************************************/
 /*  INICIALIZACIÓN DE LA CAPTURA DE LA SEÑAL DE VOZ Y DE LA SEÑAL IR  */
 /*************************************************************************/
 ICR=0x100;
 IER=1;
 ICR=0x100;
 mcbsp0_init();
 AD535_init();
 Interrupt12();

 /*******************/
 /*  LAZO INFINITO  */
 /*******************/

 for(;;)
 {
  if(*(short*)vit==1)
   {
      /*******************/
      /*     LIMITES     */
      /*******************/
      umbral2=0.01;
      pos_ini=0;
      j=longvoz-1;
      flag=0;
      while (flag==0)
      {
        E=(*(float*)(voz+j*4))*(*(float*)(voz+j*4));
        if (E>umbral2)
        {
          pos_fin=j+200;
          flag=1;
        }
        if (j==0)
        {
          pos_fin=10000;
          flag=1;
        }
        j--;
      }
      *(short*)(posicion_fin)=pos_fin;

      /***********************/
      /*  PREPROCESO - mfcc  */
      /***********************/
      long_trama = pos_fin - pos_ini + 1;
```

```
ini=0;
fin=Nm-1;
*(short*)T=1;

while (fin <= long_trama)
{
 for (j=0;j<Nm;j++)
    *(float*)(ventana+j*4)=*(float*)(voz+(pos_ini+ini+j)*4);

  Preenfasis(Nm);
  VentanaHamming(Nm, i_o_real, i_o_complejo);
  fft(Nm, i_o_real, i_o_complejo);
  magnitud(Nm, i_o_real, i_o_complejo);
  CoeficientesCepstrales(fi, fs);

  for (j=0;j<P;j++)
       *(float*)(C+((*(short*)T-1)*P+j)*4)=*(float*)(ccep+j*4);

  *(short*)T=*(short*)T+1;
  ini=(*(short*)T-1)*dp;
  fin=ini+Nm-1;
}

/*************/
/*   VITERBI */
/*************/
hmm_vit(Npal);

  transmite=0;

  switch(*(short*)orden){
  case 0:if ((pos_fin<7200)&&(pos_fin>3800)&&(*(float*)(Rel)<-
         0.10)&&(*(float*)(Rel)>-0.24))
         {base_IR=0x802f0000;transmite=1;break;} // Telestereo
  case 1:if ((pos_fin<6500)&&(pos_fin>1000)&&(*(float*)(Rel+4)<-
         0.16)&&(*(float*)(Rel+4)>-0.24))
         {base_IR=0x802f00c0;transmite=1;break;} //RPP
  case 2:if ((pos_fin<4500)&&(pos_fin>1000)&&(*(float*)(Rel+8)<-
         0.16)&&(*(float*)(Rel+8)>-0.24))
         {base_IR=0x802f0180;transmite=1;break;} //Ok
  case 3:if ((pos_fin<8500)&&(pos_fin>5000)&&(*(float*)(Rel+12)<-
         0.14)&&(*(float*)(Rel+12)>-0.22))
         {base_IR=0x802f0240;transmite=1;break;} //Studio92
  case 4:if ((pos_fin<8500)&&(pos_fin>3700)&&(*(float*)(Rel+16)<-
         0.14)&&(*(float*)(Rel+16)>-0.24))
         {base_IR=0x802f0300;transmite=1;break;} //Radio America
  case 5:if ((pos_fin<4500)&&(pos_fin>1800)&&(*(float*)(Rel+20)<-
         0.14)&&(*(float*)(Rel+20)>-0.24))
         {base_IR=0x802f03c0;transmite=1;break;} //Zeta
  case 6:if ((pos_fin<5000)&&(pos_fin>3200)&&(*(float*)(Rel+24)<-
         0.14)&&(*(float*)(Rel+24)>-0.24))
         {base_IR=0x802f0480;transmite=1;break;} //Miraflores
  case 7:if ((pos_fin<7600)&&(pos_fin>4000)&&(*(float*)(Rel+28)<-
         0.14)&&(*(float*)(Rel+28)>-0.24))
         {base_IR=0x802f0540;transmite=1;break;} //Doble9
  case 8:if ((pos_fin<8000)&&(pos_fin>3300)&&(*(float*)(Rel+32)<-
         0.14)&&(*(float*)(Rel+32)>-0.24))
         {base_IR=0x802f0600;transmite=1;break;} //Stereo100
  case 9:if ((pos_fin<9000)&&(pos_fin>4000)&&(*(float*)(Rel+36)<-
         0.14)&&(*(float*)(Rel+36)>-0.24)) {transmite=2;accion=10;break;}
         //Panamericana
  case 10:if ((pos_fin<6000)&&(pos_fin>2300)&&(*(float*)(Rel+40)<-
         0.14)&&(*(float*)(Rel+40)>-0.24)) {transmite=2;accion=11;break;}
         //Radiomar
  case 11:if ((pos_fin<5500)&&(pos_fin>2500)&&(*(float*)(Rel+44)<-
         0.14)&&(*(float*)(Rel+44)>-0.24)) {transmite=2;accion=12;break;}
         //Planeta
  case 12:if ((pos_fin<4000)&&(pos_fin>1600)&&(*(float*)(Rel+48)<-
         0.14)&&(*(float*)(Rel+48)>-0.24))
         {base_IR=0x802f0840;transmite=1;break;} //Power
```

```
   case 13:if ((pos_fin<4000)&&(pos_fin>2000)&&(*(float*)(Rel+52)<-
           0.14)&&(*(float*)(Rel+52)>-0.24))
           {*(short*)continuacion=1;accion=3;break;}//S.Woofer
   case 14:if ((pos_fin<6000)&&(pos_fin>2500)&&(*(float*)(Rel+56)<-
           0.15)&&(*(float*)(Rel+56)>-0.23))
           {*(short*)continuacion=1;accion=1;break;} //Volumen
   case 15:if ((pos_fin<4000)&&(pos_fin>1500)&&(*(float*)(Rel+60)<-
           0.16)&&(*(float*)(Rel+60)>-0.24))
           {*(short*)continuacion=1;accion=2;break;} //baja
   case 16:if ((pos_fin<3000)&&(pos_fin>1000)&&(*(float*)(Rel+64)<-
           0.16)&&(*(float*)(Rel+64)>-0.27))
           {*(short*)continuacion=1;accion=15;break;} //EQ
   case 17:if ((pos_fin<6000)&&(pos_fin>3000)&&(*(float*)(Rel+68)<-
           0.14)&&(*(float*)(Rel+68)>-0.24))
           {base_IR=0x802f0c00;transmite=1;break;} //auxiliar
   case 18:if ((pos_fin<4000)&&(pos_fin>1500)&&(*(float*)(Rel+72)<-
           0.17)&&(*(float*)(Rel+72)>-0.26))
           {base_IR=0x802f0d80;transmite=1;break;} //discos
   case 19:if ((pos_fin<5300)&&(pos_fin>2000)&&(*(float*)(Rel+76)<-
           0.13)&&(*(float*)(Rel+76)>-0.22))
           {base_IR=0x802f0e40;transmite=1;break;} //emisoras
   case 20:if ((pos_fin<4800)&&(pos_fin>3000)&&(*(float*)(Rel+80)<-
           0.14)&&(*(float*)(Rel+80)>-0.24))
           {base_IR=0x802f0f00;transmite=1;break;} //silencio
   case 21:if ((pos_fin<4500)&&(pos_fin>1800)&&(*(float*)(Rel+84)<-
           0.14)&&(*(float*)(Rel+84)>-0.24))
           {*(short*)continuacion=1;accion=13;break;} //Buscar
   case 22:if ((pos_fin<4500)&&(pos_fin>1800)&&(*(float*)(Rel+88)<-
           0.15)&&(*(float*)(Rel+88)>-0.25)) {transmite=2;accion=4;break;}
           //Primero
   case 23:if ((pos_fin<4500)&&(pos_fin>1800)&&(*(float*)(Rel+92)<-
           0.15)&&(*(float*)(Rel+92)>-0.26)) {transmite=2;accion=5;break;}
           //Segundo
   case 24:if ((pos_fin<4500)&&(pos_fin>1800)&&(*(float*)(Rel+96)<-
            0.15)&&(*(float*)(Rel+96)>-0.26)) {transmite=2;accion=6;break;}
            //Tercero
   case 25:if ((pos_fin<4500)&&(pos_fin>1200)&&(*(float*)(Rel+100)<-
            0.15)&&(*(float*)(Rel+100)>-0.27)) {transmite=2;accion=7;break;}
            //Cuarto
   case 26:if ((pos_fin<4000)&&(pos_fin>1800)&&(*(float*)(Rel+104)<-
            0.15)&&(*(float*)(Rel+104)>-0.25)) {transmite=2;accion=8;break;}
            //Quinto
   case 27:if ((pos_fin<3500)&&(pos_fin>1800)&&(*(float*)(Rel+108)<-
            0.14)&&(*(float*)(Rel+108)>-0.24))
            {*(short*)continuacion=1;accion=14;break;}//Pista
   case 28:if ((pos_fin<4000)&&(pos_fin>1800)&&(*(float*)(Rel+112)<-
            0.14)&&(*(float*)(Rel+112)>-0.24))
            {base_IR=0x802f1080;transmite=1;break;} //Play
   case 29:if ((pos_fin<4000)&&(pos_fin>1000)&&(*(float*)(Rel+116)<-
            0.14)&&(*(float*)(Rel+116)>-0.22))
            {base_IR=0x802f1080;transmite=1;break;} //Pausa
   case 30:if ((pos_fin<5500)&&(pos_fin>2500)&&(*(float*)(Rel+120)<-
            0.14)&&(*(float*)(Rel+120)>-0.26))
            {base_IR=0x802f1200;transmite=1;break;} //Siguiente
   case 31:if ((pos_fin<5500)&&(pos_fin>2500)&&(*(float*)(Rel+124)<-
            0.14)&&(*(float*)(Rel+124)>-0.24)) {transmite=2;accion=9;break;}
            //Anterior

  }

/*********************/
/*  TRANSMISION IR   */
/*********************/
while(*(short*)continuacion==1)
 {
   if (accion==1)//volumen
   {
    transmision_IR(0x802f09c0);
    delay_millisec(600);
   }
   if (accion==2)//baja
```

```
            {
             transmision_IR(0x802f0a80);
             delay_millisec(600);
            }
            if (accion==3)//woofer
            {
             transmision_IR(0x802f0900);
             delay_millisec(1000);
            }
            if (accion==13)//buscar
            {
             transmision_IR(0x802f1080);
             delay_millisec(3000);
            }
            if (accion==14)//pista
            {
             transmision_IR(0x802f1200);
             delay_millisec(1500);
            }
            if (accion==15)//EQ
            {
             transmision_IR(0x802f0b40);
             delay_millisec(1500);
            }
          }

         if (transmite==1)
            transmision_IR(base_IR);
         if (transmite==2)
          {
            if (accion==10)  //panamericana
             {
               transmision_IR(0x802f0780);delay_millisec(220);transmision_IR(0x802f0000);de
               lay_millisec(220);transmision_IR(0x802f06c0);}
            if (accion==11)  //radiomar
             {
               transmision_IR(0x802f0780);delay_millisec(220);transmision_IR(0x802f0000);de
               lay_millisec(220);transmision_IR(0x802f0000);}
            if (accion==12)  //planeta
             {
               transmision_IR(0x802f0780);delay_millisec(220);transmision_IR(0x802f0000);de
               lay_millisec(220);transmision_IR(0x802f00c0);}
            if (accion==3)  //s.woofer
             { transmision_IR(0x802f0900);delay_millisec(220);transmision_IR(0x802f0900);}
            if (accion==4)  //primero
             { transmision_IR(0x802f0fc0);delay_millisec(220);transmision_IR(0x802f0000);}
            if (accion==5)  //segundo
             { transmision_IR(0x802f0fc0);delay_millisec(220);transmision_IR(0x802f00c0);}
            if (accion==6)  //tercero
             { transmision_IR(0x802f0fc0);delay_millisec(220);transmision_IR(0x802f0180);}
            if (accion==7)  //cuarto
             { transmision_IR(0x802f0fc0);delay_millisec(220);transmision_IR(0x802f0240);}
            if (accion==8)  //quinto
             { transmision_IR(0x802f0fc0);delay_millisec(220);transmision_IR(0x802f0300);}
            if (accion==9)  //anterior
             { transmision_IR(0x802f1140);delay_millisec(220);transmision_IR(0x802f1140);}
          }
           *(unsigned volatile int *)IO_PORT = 0;   /* apaga led (verde) */
         *(short*)vit=0;
        }

    }
}

/*****************************************************************************/
/* mcbsp0_init()-->Función de inicialización del puerto (McBSP0) Serial Port 0' */
/*****************************************************************************/
void mcbsp0_init()
{
    *(unsigned volatile int *)McBSP0_SPCR = 0;      /* reset serial port */
    *(unsigned volatile int *)McBSP0_PCR = 0;       /* set pin control $ */
```

```
    *(unsigned volatile int *)McBSP0_RCR = 0x10040; /* set receive control */
    *(unsigned volatile int *)McBSP0_XCR = 0x10040; /* set transmit control */
    *(unsigned volatile int *)McBSP0_DRR = 0;
    *(unsigned volatile int *)McBSP0_SPCR = 0x12001;   /* setup SP control $ */
}
/*****************************************************************/
/* mcbsp0_write()-->Escribe datos del registro de transmision */
/*****************************************************************/
void mcbsp0_write(int out_data)                          // funcion para escribir
{
  int temp;
  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
  while ( temp == 0)
  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
 *(unsigned volatile int *)McBSP0_DXR = out_data;
}

/*****************************************************/
/* mcbsp0_read()-->Lee datos del registro de recepción */
/*****************************************************/
int mcbsp0_read()                                        // funcion para leer
{
  int temp;
  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
  while ( temp == 0)
  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
  temp = *(unsigned volatile int *)McBSP0_DRR;
  return temp;
}

/*********************************************************************/
/* AD535_init()-->control del codificador TLC320AD535 (16-bit, 8 kHz)   */
/*********************************************************************/
void AD535_init()
{
   /* Registro de control 3 */
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();
   mcbsp0_write(1);
   mcbsp0_read();
   mcbsp0_write(0x0386);   /* Inicializa el software del canal de voz */
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();

   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();
   mcbsp0_write(0);

   mcbsp0_read();
   mcbsp0_write(1);
   mcbsp0_read();
   mcbsp0_write(0x0306);   /* Selecciona 'mic preamp' como entrada en el ADC */
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();

     /* Registro de control 4 - ganancia de entrada */

   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();
   mcbsp0_write(1);
   mcbsp0_read();
   mcbsp0_write(0x0479);
```

```
   mcbsp0_read();
   mcbsp0_write(0);
   mcbsp0_read();

}
/************************************/
/* Interrupt12(): INT12-->RINT0      */
/*** *********************************/
void Interrupt12()
{

  config_Interrupt_Selector(12, RINT0); /*Interrupción 12 recepción de muestras
(RINT0)*/
  enableSpecificINT(12);
  enableNMI();
  enableGlobalINT();
  mcbsp0_read();
  mcbsp0_write(0);
}

 /************************************/
 /*  FUNCIONES DE PRE-PROCESAMIENTO  */
 /************************************/

float ValoresHamming(int n)
{

        int i=0;
        for (i = 0; i < n + 1 ; i++)
        {
         *(float*)(hamming+i*4) = 0.54 - 0.46 * cos(2*3.14159*i/n);
        }
        return (1);
}


float Preenfasis(int n)
{
        int i=0;
        *(float*)vent_preenfasis = *(float*)ventana;
        for (i = 1; i < n; i++)
        {
      *(float*)(vent_preenfasis+i*4) = *(float*)(ventana+i*4) - .95 *
(*(float*)(ventana+(i-1)*4));
        }
    return (1);
}

float VentanaHamming (int n, float i_o_real[], float i_o_complejo[])
{
    int i=0;
    for (i = 0; i < n; i++)
    {
     i_o_real[i] = *(float*)(vent_preenfasis+i*4) * (*(float*)(hamming+i*4));
     i_o_complejo[i] = 0;
    }
    return(1);
}

void fft(int n, float x[], float y[])
{
    float a, e, xt, yt;
    int  n1, n2, i, j, k, m;


    if ( frexp((float)n, &m) != 1.0 ) {
        m--;
    }

    /* ---------------------- FFT -------------------------- */
```

```
    /* Ajuste de parámetros */
    --y;
    --x;

    /* Funcion */
    n2 = n;
    for (k = 1; k <= m; ++k) {
        n1 = n2;
        n2 /= 2;
        e = (float)6.28318530717958647692528676655900576839 / n1;
        a = 0.0;
        for (j = 1; j <= n2; ++j) {
            float c = cos(a);
            float s = sin(a);
         a = j * e;

            for (i = j; n1 < 0 ? i >= n : i <= n; i += n1) {
               int q = i + n2;

               xt = x[i] - x[q];
               x[i] += x[q];
               yt = y[i] - y[q];
               y[i] += y[q];
               x[q] = c * xt + s * yt;
               y[q] = c * yt - s * xt;
            }
        }
    }

    /* ------------ 'DIGIT REVERSE' ----------------- */

    j = 1;
    n1 = n - 1;
    for (i = 1; i <= n1; ++i) {
        if (i < j) {
            xt = x[j]; x[j] = x[i];   x[i] = xt;
            xt = y[j]; y[j] = y[i];   y[i] = xt;
        }
        k = n / 2;
        while (k<j) {
            j -= k;
            k /= 2;
        }
        j += k;
    }
}

void magnitud(int n, float xr[],float xc[])
{

    int i;
    float xr2, xc2, mag2;
    for (i=0;i<n;i++)
    {
      xr2 = xr[i] * xr[i];
      xc2 = xc[i] * xc[i];
      mag2 = xr2 + xc2;
      *(float*)(magspec+i*4) = sqrtf(mag2);
    }
}

void FiltrosTriangulares(float fi[], float fc[], float fs[])
{
    int i,j;
    for (i=0; i<10; i++)
    {
      fi[i] = 2*(i+1)-1;
      fc[i] = fi[i] + 2;
      fs[i] = fi[i] + 4;
    }
```

```
    fi[10]=23; fi[11]=27; fi[12]=31; fi[13]=35; fi[14]=40;
    fi[15]=46; fi[16]=53; fi[17]=61; fi[18]=70; fi[19]=81;
    fc[10]=27; fc[11]=31; fc[12]=35; fc[13]=40; fc[14]=46;
    fc[15]=53; fc[16]=61; fc[17]=70; fc[18]=81; fc[19]=93;
    fs[10]=31; fs[11]=35; fs[12]=40; fs[13]=46; fs[14]=53;
    fs[15]=61; fs[16]=70; fs[17]=81; fs[18]=93; fs[19]=108;

    for (i=0;i<BANCO;i++)
    {
      for (j=0; j<108; j++)
      {
        *(float*)(filtmag+(108*i+j)*4) = 0;
      }
    }

    for (i=0;i<BANCO;i++)
    {
      for (j=fi[i]; j<fc[i]+1; j++)
      {
        *(float*)(filtmag+(108*i+j-1)*4) =(j-fi[i])/(fc[i]-fi[i]);
      }
      for (j=(fc[i]+1); j<fs[i]+1; j++)
      {
        *(float*)(filtmag+(108*i+j-1)*4) = 1-(j-fc[i])/(fs[i]-fc[i]);
      }
    }

}


float CoeficientesCepstrales(float fi[], float fs[])
{
 int i,j;
 for (i=0; i<BANCO; i++)
     {
      *(float*)(Y+i*4)=0;
      for (j=fi[i]; j<fs[i]+1; j++)
          {
           *(float*)(Y+i*4) = *(float*)(Y+i*4) + *(float*)(magspec+j*4) *
           *(float*)(filtmag+(108*i+j)*4);
          }
      *(float*)(Y+i*4) = *(float*)(Y+i*4) * (*(float*)(Y+i*4));
      *(float*)(Y+i*4) = logf(*(float*)(Y+i*4));
     }

 for (i=0; i<P; i++)
     {
      *(float*)(ccep+i*4)=0;
      for (j=0; j<BANCO; j++)
          {
           *(float*)(ccep+i*4) = *(float*)(ccep+i*4) + *(float*)(coefwin+(BANCO*i+j)*4)
* (*(float*)(Y+j*4));
          }
     }
 return(1);
}

float VentanasSenoidales(int p, int banco)
{
 int i,k;
 for (i=0; i<p; i++)
 {
   for (k=1; k<banco+1; k++)
   {
     *(float*)(coefwin+(20*i+k-1)*4) = cos((3.14159/banco)*(i+1)*(k-0.5));
   }
 }
 return(1);
}
```

```
/*************************************/
/*  AtoD(): RUTINA DE INTERRUPCION 12  */
/*************************************/

interrupt void AtoD()
{
   static int contDC=0;
   static int contvoz=0;
   static int hay_voz=0;
   static float nivelDC=0;
   static int grabar_voz=0;
   float valor,muestra,Energia;
   int j;

   float umbral=0.4;


   // Captura de la muestra de 16 bits tipo 'int'
   sample = mcbsp0_read();  /* ADC */

   //transformación del valor de la muestra de int a float
   valor = sample;
   muestra = valor / 32768;

   // Hallar nivel DC en las primeras 'longDC' muestras
    if (contDC < longDC)
    {
     contDC++;
     nivelDC = nivelDC + muestra/longDC;
    }

   // Quitar nivel DC a cada muestra;
   muestra = muestra - nivelDC;

   // Si no hay proceso y no se graba voz entonces empieza nuevo ciclo
   if ((*(short*)vit==0)&&(grabar_voz==0))
      hay_voz=0;

   // Solo para funciones que requieren presencia de voz para culminar el ciclo
   if ((*(short*)continuacion==1)&&(*(short*)vit==1))
    {
      for (j=1;j<20;j++)
       {
        Energia = muestra * muestra;
        *(float*)(bufferE+(j-1)*4) = *(float*)(bufferE+j*4);
        *(float*)(bufferE+(20-1)*4) = Energia;
       }
      sumaE = sumaE + *(float*)(bufferE+(20-1)*4) - *(float*)bufferE;

      if (sumaE > 0.04)
       {
        sumaE=0;
        hay_voz=2;
        for (j=0;j<20;j++)
            {
              *(float*)(voz+j*4) = *(float*)(bufferM+j*4);
              *(float*)(bufferE+j*4) = 0;
            }
        *(short*)continuacion=0;
        delay_millisec(1000);
       }
    }

   // Para funciones que no requieren doble presencia de voz
   if ((contDC == longDC)&&(hay_voz==0))
    {
      for (j=1;j<longM;j++)
       {
        *(float*)(bufferM+(j-1)*4) = *(float*)(bufferM+j*4);
        *(float*)(bufferM+(longM-1)*4) = muestra;
        Energia = muestra * muestra;
```

```
          *(float*)(bufferE+(j-1)*4) = *(float*)(bufferE+j*4);
          *(float*)(bufferE+(longM-1)*4) = Energia;
         }
       // ¿Hay presencia de voz?
         sumaE = sumaE + *(float*)(bufferE+(longM-1)*4) - *(float*)bufferE;


         if (sumaE > umbral)
          {
             // Si hay voz
             hay_voz=1;
             contvoz=0;
             grabar_voz=1;
             sumaE=0;

             CNTL1 = 0x8;
             *(unsigned volatile int *)IO_PORT = CNTL1<<25; /* enciende led verde */

             for (j=0;j<longM;j++)
              {
                *(float*)(voz+j*4) = *(float*)(bufferM+j*4);
                *(float*)(bufferE+j*4) = 0;
              }

          }
      }

        // Hay presencia de voz, grabar 2s de voz = 16000 muestras.
        if ((grabar_voz==1)&&(contvoz+longM<longvoz))
         {
           *(float*)(voz+(contvoz+longM)*4) = muestra;
           contvoz++;
         }

        if ((grabar_voz==1)&&(contvoz+longM == longvoz))
         {

           grabar_voz=0;
           *(short*)vit=1;
         }

}


/*******************************************************/
/*  carrier(): PORTADORA - RUTINA DE INTERRUPCION  10  */
/*******************************************************/

interrupt void carrier()
{
 if (*(unsigned volatile char *)luz==0)
     *(unsigned volatile char *)f_luz=0;
 if (*(unsigned volatile char *)luz==1)
     *(unsigned volatile char *)f_luz=1;
 if (*(unsigned volatile char *)f_luz)
     *(unsigned volatile char *)luz=0;
 else
     *(unsigned volatile char *)luz=1;
}


/*******************************/
/*  FUNCIONES DE TEMPORIZACION */
/*******************************/

void delay_millisec(int delay)
{
  /* Assume 150 MHz CPU, timer period = 4/150 MHz */
  int timer_limit = (delay*37500);  /* calculate the end point of the required delay */
  int time_start, time_now;
```

```
  timer0_setup(1,1,0xFFFFFFFF); /* set the maximum period length ie the end value */
  timer0_start();  /* start the timer running */

  time_start  = *(unsigned volatile int *)TIMER0_COUNT; /* read start value */
  time_now = time_start;  /* set initial value */
  /* stay in while loop until you reach the end value */

  while ((time_now - time_start) < timer_limit)
  {
    time_now = *(unsigned volatile int *)TIMER0_COUNT; /* read current time */
  }
}


void retardo(int delay,int valor)
{
  // timer period = 4/150 MHz */
  int timer_limit = delay;
  int time_start, time_now;

  timer0_setup(1,1,0xFFFFFFFF);
  *(unsigned volatile int *)TIMER0_COUNT=0;
  timer0_start();
  time_start  = *(unsigned volatile int *)TIMER0_COUNT;
  time_now = time_start;
  if (valor==0)
  {
    CNTL0 = 0;
    *(unsigned volatile int *)IO_PORT = CNTL0<<24;
    while ((time_now - time_start) < timer_limit)
    {
      time_now = *(unsigned volatile int *)TIMER0_COUNT;
    }
  }

  if (valor==1)
  {
    while ((time_now - time_start) < timer_limit)
    {
     if (*(unsigned volatile char *)luz)
     {
      CNTL0 = 0x8;
      *(unsigned volatile int *)IO_PORT = CNTL0<<24;
     }
     else
     {
      CNTL0 = 0;
      *(unsigned volatile int *)IO_PORT = CNTL0<<24;
     }

      time_now = *(unsigned volatile int *)TIMER0_COUNT;
    }
  }
}


void timer0_setup(int clock_source, int output_mode, int timer_period)
{
 *(unsigned volatile int *)TIMER0_CTRL &= 0xFF3F;/* hold the timer  */
 *(unsigned volatile int *)TIMER0_CTRL |= 0x01;/* tout = timer output pin */

 if (clock_source == 0)
   {
       *(unsigned volatile int *)TIMER0_CTRL &= 0xFDFF; /* use external clock */
   }
   else
   {
       *(unsigned volatile int *)TIMER0_CTRL |= 0x200; /* use internal clock */
   }

 if (output_mode == 0)
```

```
    {
        *(unsigned volatile int *)TIMER0_CTRL |= 0x100; /* clock mode - tstat = 50%  */
    }
     else
    {
     *(unsigned volatile int *)TIMER0_CTRL &= 0xFEFF; /* pulse mode */
    }

 *(unsigned volatile int *)TIMER0_PRD   = timer_period; /* set for a longish period*/

}

void timer0_start(void)
{
*(unsigned volatile int *)TIMER0_CTRL |= 0xC0; /* start the timer */
}

void timer1_setup(int clock_source, int output_mode, int timer_period)
{
 *(unsigned volatile int *)TIMER1_CTRL &= 0xFF3F;/* hold the timer  */
 *(unsigned volatile int *)TIMER1_CTRL |= 0x01;/* tout = timer output pin */

 if (clock_source == 0)
    {
        *(unsigned volatile int *)TIMER1_CTRL &= 0xFDFF; /* use external clock */
    }
    else
    {
        *(unsigned volatile int *)TIMER1_CTRL |= 0x200; /* use internal clock */
    }

 if (output_mode == 0)
    {
        *(unsigned volatile int *)TIMER1_CTRL |= 0x100; /* clock mode - tstat = 50%  */
    }
     else
    {
     *(unsigned volatile int *)TIMER1_CTRL &= 0xFEFF; /* pulse mode */
    }

 *(unsigned volatile int *)TIMER1_PRD   = timer_period; /* set for a longish period*/

}


void timer1_start(void)
{
*(unsigned volatile int *)TIMER1_CTRL |= 0xC0; /* start the timer */
}

/*****************************/
/* TRANSMISION DE LA SEÑAL IR */
/*****************************/

void transmision_IR(int base)
{
       unsigned int estado;
       int j,k;
       *(unsigned volatile char *) luz=0;
       *(unsigned volatile char *) f_luz=0;

       CSR=0x100;     /* Desabilita todas las interrupciones */
       IER=1;         /* Desabilita todas las interrupciones excepto NMI   */
       ICR=0xffff;
       config_Interrupt_Selector(10, TINT1); /*Interrup. 10 frec. de portadora 32kHz*/
       enableSpecificINT(10);
       enableNMI();
       enableGlobalINT();

       timer1_setup(1,1,0xFFFFFFFF);
       *(unsigned volatile int *)TIMER1_COUNT=0;
```

```
        retardo(100000,0);

        k=586;

        *(unsigned volatile int *)TIMER1_PRD=k;
        timer1_start();

        // Cabecera (bit en alta)
        retardo(126000,1);
        // Cabecera (bit en baja)
        retardo(65700,0);

        for(j=0;j<48;j++)
        {
         retardo(16000,1);
         estado=*(unsigned int*)(base+j*4);

         if (estado==1)
            retardo(48988,0);
         if (estado==0)
            retardo(16300,0);
        }

        retardo(16000,1);
        retardo(200000,0);

        CSR=0x100;    /* Desabilita todas las interrupciones              */
        IER=1;        /* Desabilita todas las interrupciones excepto NMI  */
        ICR=0xffff;
        Interrupt12();

}


/************************/
/* ALGORITMO DE VITERBI */
/************************/

void hmm_vit(int Npalabras)
{
 int dm_A,dm_mu,dm_Sigma;
 int w;
 float det, snm, nms;
 float prob_mayor;
 int i,j,k,g,h;
 float constante = pow(2*3.14159, P);


         prob_mayor=-3000;
         *(short*)orden=0;

         dm_A=N*N*4;
         dm_mu=P*N*4;
         dm_Sigma=P*4;
         for (w=0; w<Npalabras; w++)
         {
           for (j=0; j<N*N; j++)
           {
             *(float*)(A+j*4) = *(float*)(p_A+w*dm_A+j*4);
           }
           for (j=0; j<N*P; j++)
           {
             *(float*)(mu+j*4) = *(float*)(p_mu+w*dm_mu+j*4);
           }
           for (h=0; h<N; h++)
           {
             for (j=0; j<P; j++)
             {
               *(float*)(Sigma+(h*P+j)*4) = *(float*)(p_Sigma+w*dm_Sigma+j*4);
             }
```

```
 }
 *(float*)pi0=1;
 for (j=1; j<N; j++)
   *(float*)(pi0+j*4)=0;

 // Aqui empieza hmm_vit

 for (j=0; j<N*N; j++)
 {
   *(float*)(A+j*4) = logf(*(float*)(A+j*4) + 0.000001);
 }
for (j=0; j<N; j++)
 {
   *(float*)(pi0+j*4) = logf(*(float*)(pi0+j*4) + 0.000001);
 }
for (i=0; i<(*(short*)T-1)*N; i++)
 {
   *(float*)(logdens+i*4) = 0;
   *(float*)(plogl+i*4) = 0;
 }
det=1;
for (i=0; i<P; i++)
{
  det = det * *(float*)(Sigma+i*4);
}
snm = constante * det;
nms = logf(1 / sqrtf(snm));

for (j=0; j<N; j++)
 {
   *(float*)(nm+j*4) = nms;
 }

for (j=0; j<N; j++)
 {
   *(float*)(logdens+j*4)=0;
   for (i=0; i<P; i++)
    {
      *(float*)(logdens+j*4) = *(float*)(logdens+j*4) + ((*(float*)(C+i*4)-
      (*(float*)(mu+(j*P+i)*4))) * (*(float*)(C+i*4)-(*(float*)(mu+(j*P+i)*4)))
      /(*(float*)(Sigma+(j*P+i)*4)));
    }
   *(float*)(logdens+j*4) = *(float*)(nm+j*4) - 0.5*(*(float*)(logdens+j*4));
   *(float*)(plogl+j*4) = *(float*)(pi0+j*4) + *(float*)(logdens+j*4);
 }



for (k=1; k<(*(short*)T-1); k++)
 {
    for (j=0; j<N; j++)
     {
       for (i=0; i<P; i++)
        {
          *(float*)(logdens+(k*N+j)*4) = *(float*)(logdens+(k*N+j)*4) +
          ((*(float*)(C+(k*P+i)*4)-(*(float*)(mu+(j*P+i)*4))) *
          (*(float*)(C+(k*P+i)*4)-(*(float*)(mu+(j*P+i)*4)))
          /(*(float*)(Sigma+(j*P+i)*4)));
        }
       *(float*)(logdens+(k*N+j)*4) = *(float*)(nm+j*4) -
       0.5*(*(float*)(logdens+(k*N+j)*4));
     }


     for (h=0; h<N; h++)
     {
       for (g=0; g<N; g++)
        {
          *(float*)(plA+(N*h+g)*4)=*(float*)(plogl+((k-1)*N+h)*4);
        }
     }
```

```
            for (h=0; h<N; h++)
             {
               *(float*)(maximo+h*4)=*(float*)(plA+h*4)+*(float*)(A+h*4);
             }

            for (h=0; h<N; h++)
             {
               for (g=0; g<N; g++)
                {
                  if ((*(float*)(plA+(g*N+h)*4)+*(float*)(A+(g*N+h)*4)) >
                      *(float*)(maximo+h*4))
                   {
                     *(float*)(maximo+h*4) =
                      *(float*)(plA+(g*N+h)*4)+*(float*)(A+(g*N+h)*4);
                   }
                }
             }


            for (j=0; j<N; j++)
             {
               *(float*)(plogl+(k*N+j)*4) = *(float*)(maximo+j*4);
               *(float*)(plogl+(k*N+j)*4) = *(float*)(plogl+(k*N+j)*4) +
                *(float*)(logdens+(k*N+j)*4);
             }

          }

          *(float*)probabilidad = *(float*)(plogl+(*(short*)T-2)*N*4);
          for (h=0; h<N; h++)
           {
             if (*(float*)(plogl+((*(short*)T-2)*N+h)*4) > *(float*)probabilidad)
               {
                 *(float*)probabilidad = *(float*)(plogl+((*(short*)T-2)*N+h)*4);
               }
           }

          *(float*)(Prob+w*4)=*(float*)probabilidad;
          *(float*)(Rel+w*4)=*(float*)(Prob+w*4)/(*(short*)posicion_fin);
          if (*(float*)(Prob+w*4)>prob_mayor)
           {
             prob_mayor=*(float*)(Prob+w*4);
             *(short*)orden=w;
           }
        }
 }

/*******/
/* FIN */
/*******/
```

BOOT.asm

```
        .title  "Rutina de inicialización del sistema mediante la memoria FLASH"
        .option D,T

BOOT_SIZE     .equ    0x800       ; longitud del código de inicialización en bytes
FLASH_START   .equ    0x90000000  ; dirección de inicio en la memoria FLASH
BOOT_START    .equ    0x00000000  ; dirección de inicio de L2 SRAM
CODE_SIZE     .equ    0xf800      ; longitud del código de la aplicación en bytes
CODE_START    .equ    0x800       ; dirección de inicio de la aplicación
long_modelos  .equ    0x13100     ; longitud de los modelos de todas las frases
MODELOS_flash .equ    0x9000a000  ; dirección de inicio de los modelos en la FLASH
MODELOS_ram   .equ    0x80200000  ; dirección de inicio de los modelos en la memoria RAM
long_ir       .equ    0x600       ; longitude de los códigos IR
IR_flash      .equ    0x9001d100  ; dirección de inicio de los códigos IR en la FLASH
IR_ram        .equ    0x802f0000  ; dirección de inicio de los códigos IR en la RAM
```

```
LEDS          .equ    0x90080000  ; dirección de los leds
EMIF_GCR      .equ    0x01800000  ; EMIF global control
EMIF_CE1      .equ    0x01800004  ; dirección del registro de control de EMIF CE1
EMIF_CE0      .equ    0x01800008  ; dirección del registro de control de EMIF CE0
EMIF_SDCTRL   .equ    0x01800018  ; control EMIF SDRAM
EMIF_SDRP     .equ    0x0180001c  ; periodo de refresco EMIF SDRAM
EMIF_CE1_8    .equ    0xffffff03  ; 8 bits para espacio CE1
EMIF_CE1_32   .equ    0xffffff23  ; 32 bits para espacio CE1
EMIF_CE0_V    .equ    0xffffff33  ; EMIF CE0control   ;0x30
EMIF_SDCTRL_V .equ    0x07117000  ; EMIF SDRAM control ;0x73380000

 .sect "boot_load"
 .global _boot
 .ref _c_int00

_boot:
           ; Configurando L2 modo
           ZERO    .D2     B4
           MVKH    .S2     0x1840000,B4
           LDW     .D2T2   *B4,B5
           ZERO    .D2     B4
           MVKH    .S2     0x1840000,B4
           NOP             2
           SET     .S2     B5,8,9,B5
           STW     .D2T2   B5,*B4
           NOP             2

           MVK     0x100,A0     ; CSR=0x100 Desabilita todas las interrupciones
           MVC     A0,CSR
           MVK     0x01,A0      ; IER=1 Desabilita las interrupciones excepto NMI
           MVC     A0,IER
           ZERO    A0           ; ICR=0xffff limpiar todas las interrupciones pendientes;
           SET     A0,0x0,0xf,A0
           MVC     A0,ICR
;-------------------------------------------------------------------------------------------
--------
            mvkl  EMIF_GCR,A4       ; EMIF_GCR
      ||    mvkl  0x3300,B4
            mvkh  EMIF_GCR,A4
      ||    mvkh  0x3300,B4
            stw   B4,*A4
            mvkl  EMIF_CE0,A4       ; EMIF_CE0
      ||    mvkl  EMIF_CE0_V,B4
            mvkh  EMIF_CE0,A4
      ||    mvkh  EMIF_CE0_V,B4
            stw   B4,*A4
            mvkl  EMIF_SDCTRL,A4    ; EMIF_SDCTRL
      ||    mvkl  EMIF_SDCTRL_V,B4
            mvkh  EMIF_SDCTRL,A4
      ||    mvkh  EMIF_SDCTRL_V,B4
            stw   B4,*A4
      ||    mvkl  EMIF_SDRP,A4      ; EMIF_SDRP
      ||    mvkl  0x61a,B4
            mvkh  EMIF_SDRP,A4
      ||    mvkh  0x61a,B4
            stw   B4,*A4
;-------------------------------------------------------------------------------------------
; LOS PRIMEROS 1024 BYTES (0X400) DE LA MEMORIA FLASH SE COPIAN AUTOMATICAMENTE AL
; ENERGIZAR LA TARJETA

            mvkl  BOOT_START+1024,A4   ; ram dirección de inicio + 0x400
      ||    mvkl  FLASH_START+1024,B4  ; flash dirección de inicio + 0x400
            mvkh  BOOT_START+1024,A4
      ||    mvkh  FLASH_START+1024,B4
            zero  A1
_boot_loop1:
            ldb   *B4++,B5
            mvkl  BOOT_SIZE-1024,B6    ; BOOT_SIZE -1024  (LA LONGITUD MENOS 0X400)
            add   1,A1,A1
      ||    mvkh  BOOT_SIZE-1024,B6
            cmplt A1,B6,B0
```

```
            nop
            stb   B5,*A4++
      [B0]  b     _boot_loop1
            nop   5
;-------------------------------------------------------------------------------------------
            mvkl  CODE_START,A4         ; copia la aplicación
            mvkh  CODE_START,A4
            zero  A1
_boot_loop2:
            ldb   *B4++,B5
            mvkl  CODE_SIZE-4,B6
            add   1,A1,A1
      ||    mvkh  CODE_SIZE-4,B6
            cmplt  A1,B6,B0
            nop
            stb   B5,*A4++
      [B0]  b     _boot_loop2
            nop   5
;-------------------------------------------------------------------------------------------
            mvkl  MODELOS_ram,A4        ; copia los modelos de harkov a la RAM
      ||    mvkl  MODELOS_flash,B4
            mvkh  MODELOS_ram,A4
      ||    mvkh  MODELOS_flash,B4
            zero  A1
_lazo_modelos:
            ldb   *B4++,B5
            mvkl  long_modelos,B6
            add   1,A1,A1
      ||    mvkh  long_modelos,B6
            cmplt A1,B6,B0
            nop
            stb   B5,*A4++
      [B0]  b     _lazo_modelos
            nop   5

;-------------------------------------------------------------------------------------------
            mvkl  IR_ram,A4           ; copia los códigos IR a la RAM
      ||    mvkl  IR_flash,B4
            mvkh  IR_ram,A4
      ||    mvkh  IR_flash,B4
            zero  A1
_lazo_ir:
            ldb   *B4++,B5
            mvkl  long_ir,B6
            add   1,A1,A1
      ||    mvkh  long_modelos,B6
            cmplt A1,B6,B0
            nop
            stb   B5,*A4++
      [B0]  b     _lazo_ir
            nop   5
;-------------------------------------------------------------------------------------------
            mvkl  EMIF_CE1,A4         ; CE1 Para poder usar los leds
      ||    mvkl  0xffffff23,B4
            mvkh  EMIF_CE1,A4
      ||    mvkh  0xffffff23,B4
            stw   B4,*A4
;-------------------------------------------------------------------------------------------
            mvkl .S2 _c_int00, B0     ; salta al inicio del código de la aplicación
            mvkh .S2 _c_int00, B0
            B    .S2 B0
            nop   5

;  fin de la rutina de inicialización
```

VECTORS.asm

```
        .ref    _boot
        .ref    _AtoD
        .ref    _carrier
        .sect   "vectors"
rst:            mvkl    .s2     _boot, b0
                mvkh    .s2     _boot, b0
                 b      .s2     b0
                nop

                nop
                nop
                nop
               nop

nmi:            nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

resv1:          nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop

resv2:          nop

                nop
                nop
                nop
               nop
                nop
                nop

int8:           nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop

int9:           nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

int10:          b       _carrier
                nop
                nop
                nop
                nop
               nop
                nop
                nop


int11           nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

int4:           nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop

int5:           nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop

int6:           nop
                nop
                nop
                nop
               nop
                nop
                nop
                nop

int7:           nop
                nop
                nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop

int12:          b       _AtoD
                nop
                nop
                nop
                nop
               nop
                nop
                nop

int13:          nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

int14:          nop
                nop
                nop
                nop
                nop
               nop
                nop
                nop
```

```
                                                                    nop
int15:          nop                                                 nop
                nop                                                 nop
                nop                                                 nop
                nop
```

LNK.cmd

```
-c
-lrts6700.lib
MEMORY
{
    BOOT_RAM:   o = 00000000h   l = 00000800h
    IRAM:       o = 00000800h   l = 0000f800h
    CE0:        o = 80000000h   l = 01000000h
}
SECTIONS
{
    "vectors"   >       BOOT_RAM
    "boot_load" >       BOOT_RAM
    .text       >       IRAM
    .stack      >       IRAM
    .bss        >       IRAM|CE0
    .cinit      >       IRAM
    .const      >       IRAM
    .data       >       IRAM
    .far        >       IRAM
    .switch     >       IRAM
    .sysmem     >       IRAM
    .tables     >       IRAM
    .cio        >       IRAM
}
```