INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS CIUDAD DE MÉXICO

Proyectos de Ingeniería II

"Implementación de algoritmos de análisis y reconocimiento de voz esofágica en un DSP"

Realizado por:

Adolfo Abraham Méndez Cervantes
Miguel Arturo Castillo Arroyo
Alan Guillermo Galicia González
Luis Antonio Martínez Arias

Asesor:
Dr. Alfredo Victor Mantilla Caeiros

Sinodales:
Dr. Jorge Brieva Rico
M. en C. Rodrigo Regalado García

Profesora:
Dra. Katya Eugenia Romo Medrano Mora

Mexico, D.F. 2008

Índice

1. Introducción

Desde la aparición del hombre, éste ha tenido la necesidad de comunicarse, ya sea de forma oral o escrita. Se han construido sociedades y civilizaciones, teniendo como común denominador la comunicación.

Es por tal motivo que la comunicación, en especial la oral, es de gran importancia para el hombre, ya que gracias a ella se ha logrado transmitir desde cuentos y leyendas, hasta el mismo conocimiento filosófico y científico.

La tecnología ha ayudado a mejorar la calidad de vida de los hombres, y en nuestro caso queremos aprovechar los avances tecnológicos actuales para volverles a dar la capacidad de comunicarse oralmente a aquellas personas que sufrieron de una laringectomía, operación que consiste en la extirpación de la laringe debido a la presencia de cáncer en este órgano.

Debido a que la laringe es una unión entre la nariz y la boca con la traquea, y ésta a su vez se comunica con los pulmones, la modificación de la traquea es parte fundamental de una laringectomía. La traquea es entonces modificada para que se conecte directamente con un estoma, que es una abertura que se hace en la garganta y es a través de la cual el paciente puede ingresar aire a sus pulmones y respirar. Esto impide la posibilidad de usar el aire proveniente de los pulmones para producir voz usando el aparato fonador. Cuando la laringe es extirpada, se pierden las cuerdas vocales y la capacidad de hablar por medio de ellas, pero no por esto se pierde la capacidad para hablar definitivamente, pues aun es posible hacer pasar aire a través de la boca y la nariz.

La voz esofágica se produce comprimiendo con la ayuda de la lengua, y posteriormente deglutiendo el aire contenido en el tracto vocal. La voz se genera cuando la corriente de aire pasa por el segmento faringo-esofágico y provoca la vibración del músculo cricofaríngeo. Los patrones de voz esofágica producida de un paciente laringectomizado se consideran similares a los de un hablante normal [8].

2. Objetivos

Objetivo General

Programar en un DSP los algoritmos que permitan analizar, reconocer y sintetizar voz esofágica; para de esta forma ayudar a que la gente que se comunica de esta manera, lo haga eficientemente.

Objetivos Específicos

- Primer Semestre:
  - Estudio del marco teórico de los proyectos previamente realizados:
    - Anatomía de la voz y del sistema auditivo humano
    - Bandas críticas y modelo del oído
    - Transformada Wavelet
    - Redes Neuronales
    - Diseño del sistema
  - Familiarización con el DSP C5416 de Texas Instruments
    - Arquitectura
    - Set de instrucciones
  - Implementación de las etapas de análisis, detección de segmentos vocalizados, y extracción de parámetros.

- Segundo Semestre:
  - Implementación de la etapa de reconocimiento.
  - Conjunción de todas las etapas en formato de funcionamiento multitareas.

3. Justificación

Este proyecto es la continuación de trabajos anteriores [4], [5], [6], en los cuales se desarrolló una investigación teórica así como simulaciones realizadas en MatLab; todo esto lo usaremos como base para lograr la implementación del sistema en un DSP (Digital Signal Processor).

La razón por la que se utilizó un DSP para realizar la implementación, es por la ventaja que presenta este tipo de procesador (especializado en el procesamiento digital de señales): la gran velocidad de procesamiento, lo cual permitiría que el desempeño del sistema se logre en tiempo real.

4. Marco Teórico

4.1 Anatomía de la voz

La voz humana se produce por medio del aparato fonador. Éste está formado por los pulmones como fuente de energía en la forma de un flujo de aire, la laringe, que contiene las cuerdas vocales, la faringe, las cavidades bucal y nasal, y una serie de elementos articulatorios: los labios, los dientes, el alvéolo, el paladar, el velo del paladar y la lengua.

Las cuerdas vocales son dos membranas dentro de la laringe orientadas de adelante hacia atrás; cuando se encuentran separadas, la glotis adopta una forma triangular, el aire pasa libremente y prácticamente no se produce sonido. Es el caso de la respiración.

Los fonemas son producidos por los movimientos de los órganos del aparato fonador, movimientos que generan pautas de cambios de presión en el aire que se denominan estímulos o señales acústicas. La fonación se realiza durante la espiración, cuando el aire contenido en los pulmones, sale de éstos y, a través de los bronquios y la tráquea, llega a la laringe. La laringe da paso al aire inspirado y espirado. En la laringe se encuentran las cuerdas vocales, cuando estas vibran al paso del aire inspirado, emiten el sonido laríngeo: la laringe es, entonces, el órgano esencial de la fonación, que es la emisión de sonidos con fines intencionados de comunicación.

A través de la voz, los seres humanos somos capaces de producir sonidos con frecuencias en un rango desde los 100 Hz hasta los 10 kHz. Sin embargo, en una señal de voz tanto la información verbal como la sección más significativa del espectro de la voz se encuentran contenidas a partir de los 300 Hz hasta los 4 kHz. Tomando en cuenta únicamente la inteligibilidad de la voz, entonces el rango de frecuencias que contendría esta información sería desde los 500 Hz a los 2.5 kHz.

La voz se puede clasificar por las características de su espectro de la siguiente forma:

a) *Segmento vocalizado*: Constituye la parte periódica o casi periódica de la señal de voz y está formado fundamentalmente por los fonemas de las vocales y de algunas consonantes que tienen un comportamiento espectral casi periódico.

b) *Segmento no vocalizado*: Corresponde a segmentos no periódicos de la señal, que son fundamentalmente consonantes.

c) *Segmento de silencio*: Es aquel que no contiene ninguna señal de voz.

4.2 Sistema auditivo humano

Como la mayoría de los elementos biológicos, el sistema auditivo humano presenta una gran complejidad. Éste cumple la tarea de proporcionarnos información acerca del sonido, es decir, los cambios en la presión atmosférica debidos a la vibración de un cuerpo. Para llevar a cabo lo anterior, el sistema auditivo se encarga de detectar las oscilaciones que viajan por el aire y convertirlas en impulsos nerviosos que más tarde son interpretados por el cerebro.

El oído se divide en tres partes :

- Oído externo. Se encuentra formado por la aurícula, el canal auditivo y termina con la membrana externa del tímpano. La aurícula es la única parte visible del sistema auditivo y se encarga de recibir las vibraciones transportadas por el aire y canalizarlas hacia el interior del oído. Posee una estructura helicoidal y presenta múltiples surcos y depresiones, que permiten al oído captar una mayor cantidad de sonidos al reducir las pérdidas por reflexión y difracción. Asimismo, su forma le otorga características acústicas que modifican las vibraciones sonoras. Lo anterior permite al sistema nervioso central identificar si la fuente del sonido se encuentra ubicada al frente o detrás.

- Oído medio. Limitando con el tímpano, comienza el oído medio; éste se encuentra constituido esencialmente por los tres huesos más pequeños del cuerpo humano : el martillo, el yunque y el estribo. La función principal del oído medio es la de transportar las vibraciones mecánicas del tímpano hacia el oído interno de manera eficiente. Mientras el oído externo conduce vibraciones presentes en el aire, el oído interno se encuentra lleno de un fluido de mayor viscosidad. El oído medio es entonces el encargado de acoplar ambos elementos evitando pérdidas de energía.

- Oído interno. Es la última etapa del sistema auditivo anterior al sistema nervioso. En él se encuentra la cóclea, la cual está unida al oído medio a través de su ventana oval. La cóclea es un conducto que forma una espiral de aproximadamente 2.75 vueltas. En su interior, ésta se encuentra dividida por dos formaciones: la membrana de Riessner y la membrana basal. La membrana basal se encuentra cubierta en un lado por el órgano de Corti ; éste consiste en un conjunto de vellos que, al ser

doblados, disparan un impulso nervioso transmitido al cerebro por el nervio auditivo. Estas células capilares, al estar unidas a la membrana basal, son afectadas directamente por los movimientos de alguna de las secciones de ésta última. El cerebro entonces, al conocer la posición del vello afectado por un estímulo audible, recibe del oído la información de las componentes espectrales presentes en el sonido percibido.

4.3 Bandas críticas

Si el oído es estimulado con un tono puro, las distintas regiones de la membrana basal responderán en proporciones diferentes. Si ahora se tiene una señal audible que contiene dos o más frecuencias, la respuesta de la membrana basal será la superposición de los efectos de cada uno de sus componentes. En este caso, si se tienen dos constituyentes espectrales, cada uno afectará a una posición particular de la membrana basal y a sus alrededores. Si ambos se encuentran lo suficientemente cerca, será imposible distinguirlos como tonos independientes.

A fin de conocer la resolución aproximada del oído, se definió el concepto de ancho de banda crítico. Éste puede resumirse como la mínima diferencia necesaria en la frecuencia de dos tonos puros para que éstos sean distinguidos como sonidos independientes. El oído puede distinguir 24 bandas críticas, de las cuales solo se consideran 17 para el propósito de este trabajo, debido a la frecuencia de muestreo de la voz humana.

4.4 Transformada Wavelet

Es una herramienta matemática que permite conocer el comportamiento frecuencial de señales no estacionarias, tales como la voz humana. Las señales no estacionarias son aquellas cuyo espectro de frecuencia no es el mismo a través del tiempo, por lo que no se puede usar la transformada de Fourier, sino que se necesita de una transformada que permita conocer cómo el espectro cambia en el tiempo.

Esta transformada nos será útil para realizar la extracción de ciertos parámetros de la señal de voz, que nos servirán para, posteriormente, crear un vector que alimentará a un sistema de redes neuronales, esto para llevar a cabo la etapa de reconocimiento.

A continuación se presenta una definición de la transformada *Wavelet* que formaliza las ideas planteadas en los párrafos anteriores.

La transformada *wavelet* de una señal de energía finita $f(t) \in L^2\{\mathbf{R}\}$ consiste en la descomposición de ésta mediante un conjunto de funciones base $\psi_{\tau,s}(t)$ y se define de la siguiente manera:

$$\gamma(\tau,s) = \langle f, \psi_{\tau,s} \rangle = \int_{-\infty}^{\infty} f(t) \cdot \overline{\psi_{\tau,s}}(t)\, dt \qquad (4.1)$$

En la ecuación anterior, $\langle f, \psi_{\tau,s} \rangle$ se conoce como el producto escalar de $f(t)$ con $\psi_{\tau,s}(t)$ y se calcula de acuerdo con la integral mostrada. Esta operación puede pensarse, en analogía con espacios vectoriales como $\mathbf{R}^3$, como la proyección de la señal $f(t)$ sobre $\psi_{\tau,s}(t)$. De este modo, podemos interpretar a la transformada *wavelet* $\gamma(\tau,s)$ como la similitud entre la señal de análisis y las funciones base empleadas para la descomposición.

El producto escalar definido mediante la integral mostrada da lugar a un espacio de funciones conocido como L^2 o, con mayor precisión, $L^2\{\mathbf{R}\}$ si se trabaja con funciones reales. Este espacio es aquel que contiene a todas las funciones cuya magnitud al cuadrado es integrable, es decir, todas las $f(t)$ para las cuales:

$$\int_{\infty}^{\infty} |f(t)|^2\, dt < \infty \qquad (4.2)$$

De forma similar a los espacios vectoriales comúnmente encontrados en física, $L^2\{\mathbf{R}\}$ puede caracterizarse mediante una base. Con esto nos referimos a una colección de funciones tales que cualquier señal de energía finita puede expresarse como una combinación lineal de los miembros de la base.

La transformada wavelet, como se mencionó anteriormente, es definida a través de la familia de funciones $\psi_{\tau,s}$, a las que se les denomina *wavelets* hijas debido a que son generadas mediante la traslación y escalamiento de una función $\psi(t)$ llamada wavelet madre. En otras palabras, por "wavelet madre" nos referimos a una especie de regla que puede utilizarse para generar funciones a partir de dos parámetros, la escala s y traslación τ. Las funciones generadas a partir de una wavelet madre poseen la característica de formar una base en $L^2\{\mathbf{R}\}$. Lo anterior es un requisito para que la descomposición preserve toda la información con la que se contaba en un inicio.

Las wavelets hijas pueden obtenerse una vez establecido un wavelet madre mediante la siguiente definición:

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \cdot \psi\left(\frac{t-\tau}{s}\right) \tag{4.3}$$

4.5 Modelo del oído

Para realizar el análisis de la voz esofágica, no nos basamos en cómo se produce la voz, sino en cómo escucha el oído; entonces utilizamos un modelo del mismo[6], realizado por Zhang, Xuedong; Heinz, Michael; Bruce, Ian y Carney, Laurel, en el cual se describe la estructura base de algunos comportamientos no lineales inherentes a las fibras del nervio auditivo.

Zhang propone que la dinámica de una sección de la membrana basal que presenta una frecuencia característica f_c, es similar a la de un filtro *gamma-tone* sintonizado a esa misma frecuencia. El nombre de este tipo de filtro proviene de su representación temporal, la cual consiste en el producto de una distribución gama por un tono.

La distribución de probabilidad gama indica la probabilidad de que, desde un tiempo inicial, hayan ocurrido un cierto número de eventos que presentan una distribución Poisson; ésta puede escribirse como:

$$P[\alpha,\theta](x) = \frac{x^{\alpha-1} \cdot e^{\frac{-x}{\theta}}}{\Gamma(\alpha) \cdot \theta^{\alpha}} \quad x > 0 \tag{4.4}$$

donde α y θ son parámetros denominados "forma" y "escala" respectivamente, y $\Gamma(x)$ es la función gama definida de la siguiente manera:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} \cdot e^{-t} dt \tag{4.5}$$

En un filtro *gamma-tone*, la forma de la distribución gama se asocia generalmente con el orden del filtro. Lo anterior implica que $\alpha \in \mathbf{N}$ por lo que la ecuación (3.4) puede rescribirse como:

$$P[\alpha,\theta](x) = \frac{x^{\alpha-1} \cdot e^{\frac{-x}{\theta}}}{(\alpha-1)! \cdot \theta^{\alpha}} \quad x > 0 \tag{4.6}$$

La expresión anterior se conoce como la distribución de Erlang. En este punto introduciremos la restricción $\alpha > 1$ con la finalidad de evitar caer en el caso particular para el cual la distribución anterior se reduce a una distribución exponencial.

Siguiendo con el modelo inicial, ahora multiplicamos la distribución anterior por un tono. Éste es elegido de tal manera que su frecuencia sea igual a la tasa con la que ocurren los eventos Poisson que dan origen a la distribución gama (i.e. el promedio de veces que éstos acontecen por unidad de tiempo).

$$\psi_{\theta}^{\alpha}(t) = \frac{1}{(\alpha-1)!\cdot\theta^{\alpha}} \cdot t^{\alpha-1} \cdot e^{\frac{-t}{\theta}} \cdot \cos\left(2\pi \cdot \frac{1}{\theta} \cdot t\right) \quad t > 0 \tag{4.7}$$

Ahora tomamos un caso particular del resultado anterior al permitir que la escala $\theta = 1$

$$\psi^{\alpha}(t) = \frac{1}{(\alpha-1)!} \cdot t^{\alpha-1} \cdot e^{-t} \cdot \cos(2\pi \cdot t) \quad t > 0 \tag{4.8}$$

La ecuación anterior es el modelo del oído que será utilizado para realizar la implementación, por un lado tenemos el filtro gamma:

$$\text{Gamma} = \frac{1}{(\alpha-1)!} \cdot t^{\alpha-1} \cdot e^{-t}$$

Y por otro lado tenemos el tono:

$$\text{Tono} = \cos(2\pi \cdot t)$$

4.6 Redes neuronales

Una red neuronal es un procesador paralelo y distribuido que tiene una propensión natural para organizar conocimientos experimentales y hacerlos disponibles para su uso. Es similar al cerebro en el sentido de que su conocimiento es adquirido por medio de un proceso de aprendizaje y que éste se guarda en las conexiones interneuronales conocidas como pesos sinápticos.

Nosotros utilizamos un sistema de redes neuronales para la etapa de reconocimiento, es decir, la red neuronal nos indica a qué fonema pertenece el

vector de parámetros que se le da cómo entrada, para de esta forma poder realizar la síntesis.

Las redes neuronales están conformadas a partir de unidades elementales llamadas neuronas y las interconexiones que existen entre ellas. Comúnmente, las neuronas están agrupadas en una capa de entrada, una o varias capas intermedias u ocultas y una capa de salida.

En este modelo se puede apreciar, que la neurona consta de diferentes elementos que le permiten arrojar una salida determinada a partir de las señales de entrada. Cada una de las señales de entrada tienen asociadas a ellas cantidades llamadas fuerzas de conexión o pesos sinápticos. Los pesos pueden ser del tipo excitatorio o inhibitorio, esto quiere decir que los pesos pueden tomar valores tanto positivos como negativos. Las señales de entrada alimentan a otro bloque que se encarga de sumar a todas las señales multiplicadas por su peso. La señal resultante es la entrada de un bloque que contiene una función de activación, esta función de activación tiene la misión de limitar la amplitud de la salida de la neurona para hacerla converger a un valor finito.

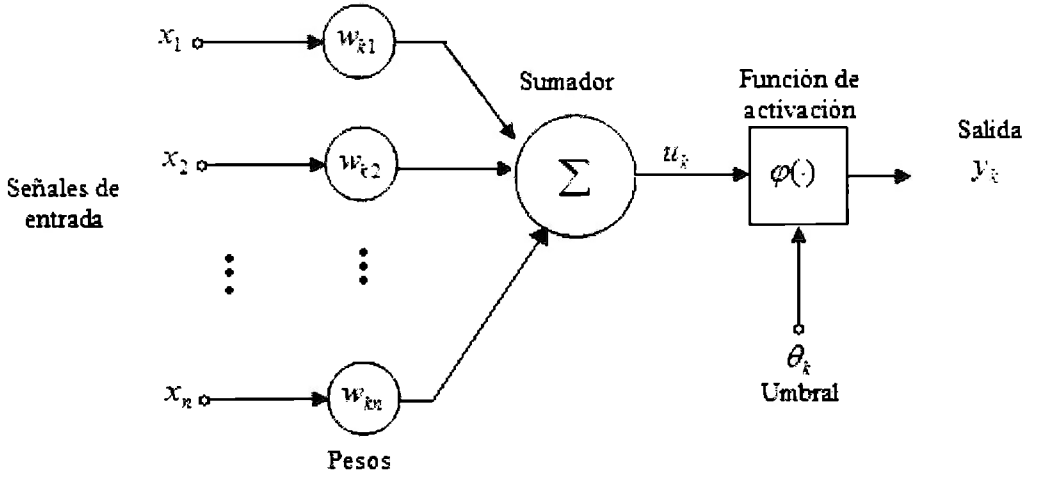


Figura 4.1 Modelo de una neurona

Existen 4 clases principales dentro de las que pueden ser agrupadas las redes neuronales según su arquitectura.

Redes de una sola capa ante-alimentadas.

Esta es la arquitectura más simple de todas; consta de una capa de entrada en la que no se realiza ningún tipo de procesamiento, y una capa de salida. Los nodos en la capa de entrada sólo transfieren información a las neuronas en la

capa de la salida y esta capa es quien se encarga de realizar el procesamiento de la información que genera una respuesta. Las señales dentro de esta red sólo recorren un solo sentido, de la entrada a la salida, es por eso que se le da la clasificación de ante-alimentadas.

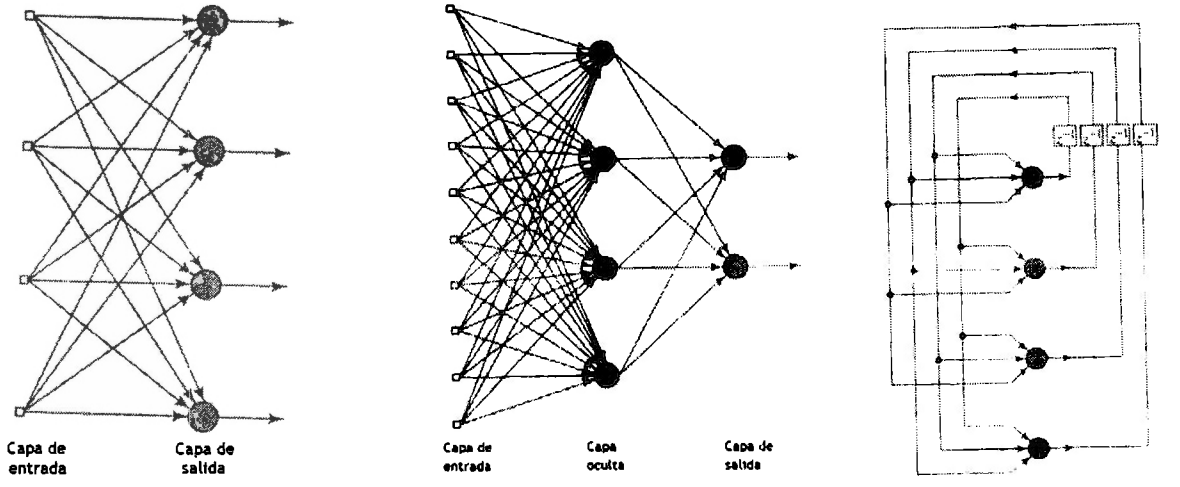
Redes de varias capas ante-alimentadas.

Esta arquitectura está formada por una capa de entrada, una o más capas ocultas y una capa de salida. La función de la capa de entrada es la de proveer a las neuronas de la capa oculta con la información que contienen las señales de entrada. La función de las capas ocultas es la de intervenir entre la entrada externa y la salida de la red, además es en ellas donde sucede el procesamiento de la información que caracteriza a ésta. La capa de salida se encarga de generar la respuesta total de la red que pertenece al patrón de activación que fue dado por los nodos que conforman la capa de entrada. En este tipo de redes, la información sólo fluye de la entrada a la salida, es por eso que también están clasificadas como ante-alimentadas.

Redes recurrentes.

Las redes recurrentes pueden tener la estructura de las redes de una o varias capas pero con la característica especial de que constan de al menos un lazo de retroalimentación.

En la siguiente figura se puede apreciar esquemas de estas arquitecturas.



Red de una sola capa.

Red de varias capas.

Red recurrente.

Figura 4.2 Arquitecturas de redes neuronales

Estructuras matriciales.

Las redes matriciales están conformadas por una capa de nodos de entrada y un arreglo de 1 o más dimensiones. Cada neurona que forma parte de la matriz tiene una o varias fuentes que vienen desde la capa de entrada. Las redes neuronales que forman estructuras matriciales, en realidad son redes del tipo de una o varias capas en donde las neuronas de salida están organizadas en filas y columnas. De igual manera, no existen lazos de retroalimentación.

Entrenamiento y Propagación hacia atrás

El entrenamiento es el proceso por el cual se enseña a la red a reconocer o diferenciar los patrones que se presentan a su entrada a través de la variación de los pesos sinápticos que conectan a las neuronas de la red.

Existe una gran variedad de algoritmos de entrenamiento, pero en general se dividen en dos tipos:

- Entrenamiento supervisado: se muestran patrones a la red y la salida deseada para cada patrón.

- Entrenamiento no supervisado: la red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta.

Uno de los algoritmos más usados en un Perceptrón Multicapa como el que se busca implementar es el de Propagación hacia atrás o Backpropagation.

Propagación hacia atrás

Este algoritmo busca minimizar la función de error que caracteriza a la salida de la red neuronal en el espacio de los pesos usando el método del descenso del gradiente. La combinación de pesos que minimiza la función de error (4.9) se considera una solución del problema de entrenamiento.

$$E = \frac{1}{2}\sum_{i=1}^{P}\left\|o_i - t_i\right\|^2 \tag{4.9}$$

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l}\right) \tag{4.10}$$

El cálculo del gradiente (4.10) sirve para obtener un factor con el cual se puedan modificar los pesos sinápticos (4.11).

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}, i = 1,...,l \tag{4.11}$$

El cálculo del gradiente se hace para cada peso de cada neurona en un proceso iterativo que termina una vez que se ha encontrado un mínimo de la función de error.

Para la capa de salida el error se obtiene directamente, por lo que el cambio en los pesos se obtiene haciendo las siguientes consideraciones:

1. La salida de la red puede expresarse como:

$$o_k = f(net_k) \tag{4.12}$$

donde :

$$net_k = \sum w_{kj} o_j \tag{4.13}$$

es decir, la suma de las entradas a la capa multiplicadas por los pesos.

2. Por regla de la cadena se sabe que:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$
$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum w_{kj} o_j = o_j \tag{4.14}$$

3. Se define:

$$\delta_k = -\frac{\partial E}{\partial net_k} \tag{4.15}$$

y se evalúa, con lo que se obtiene que δk:

$$\delta_k = -\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k} \tag{4.16}$$

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{4.17}$$

$$\frac{\partial o_k}{\partial net_k} = f'_k(net_k) \tag{4.18}$$

4. Y sustituyendo estos resultados se obtiene:

$$\Delta w_{kj} = \eta(t_k - o_k) f'_k(net_k) o_j \tag{4.19}$$

El error en las capas ocultas no puede calcularse directamente, por lo que es necesario considerar todo el error que entra en un nodo y trasladarlo a las capas anteriores. Para la obtención de las expresiones que permitirán hacer cambios en los pesos se sigue el procedimiento que se muestra a continuación:

1. Se aplica la regla de la cadena a la fórmula general del cambio de pesos.

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_j} = -\eta \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_j} \tag{4.20}$$

2. Se sustituye un resultado que también se obtuvo para las capas de salida y se obtiene que:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial net_j} o_i \tag{4.21}$$

3. Se aplica de nuevo la regla de la cadena al diferencial faltante, con lo que se obtiene que:

$$\Delta w_{kj} = \eta\left(-\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\right) o_i \tag{4.22}$$

Y después de sustituir la derivada parcial de la salida con respecto a la evaluación de los pesos en la red se obtiene:

$$\Delta w_{kj} = \eta\left(-\frac{\partial E}{\partial o_j}\right) f'_j(net_j) o_j \tag{4.23}$$

4. La derivada parcial del error con respecto a la salida de la capa j se obtiene como sigue:

$$-\frac{\partial E}{\partial o_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j} \tag{4.24}$$

$$= \sum_k \left(\frac{\partial E}{\partial net_k}\right)\frac{\partial}{\partial o_j}\sum_m w_{km} o_m \tag{4.25}$$

$$= \sum_k \left(\frac{\partial E}{\partial net_k}\right) w_{kj} = \sum_k \delta_k w_{kj} \tag{4.26}$$

5. Se define δj como:

$$\delta_j = f'_j(net_j)\sum_k \delta_k w_{kj} \tag{4.27}$$

6. Finalmente, sustituyendo δj en la expresión general del cambio de pesos se tiene que para las capas ocultas:

$$\Delta w_{kj} = \eta \delta_k o_j \tag{4.28}$$

Los pasos de implementación que sigue el algoritmo de Propagación hacia atrás son los siguientes:

- Inicialización de pesos: se generan pesos aleatorios de valores cercanos a cero.

- Propagación hacia adelante (Feed Forward): se alimenta una entrada x a la red. Se calculan y se almacenan las derivadas de las salidas de cada nodo.

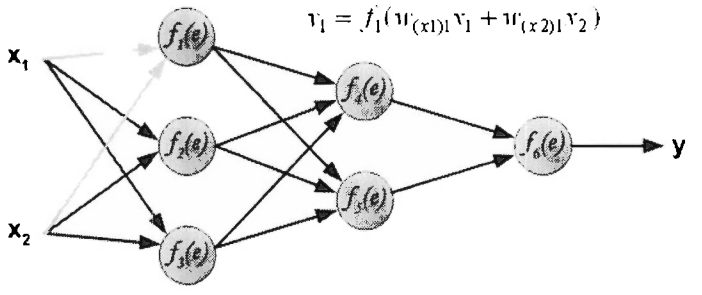


Figura 4.3. Propagación hacia delante en la primera capa oculta

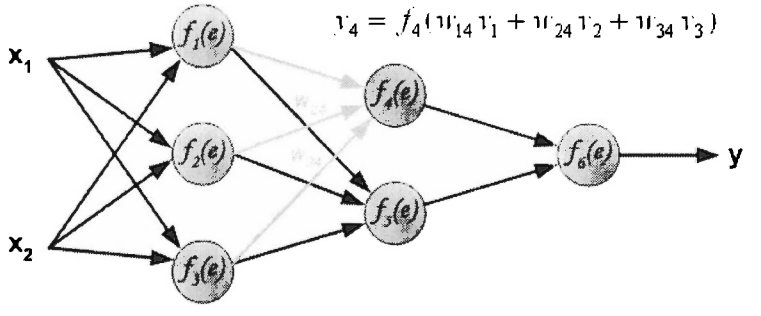


Figura 4.4. Propagación hacia delante en la segunda capa oculta

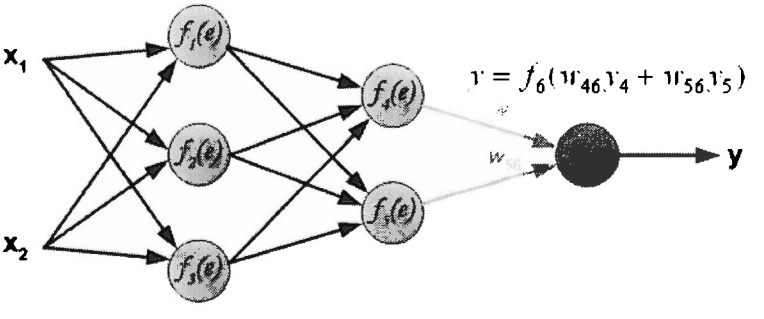


Figura 4.5. Propagación hacia delante en la capa de salida

- Propagación hacia atrás: se presenta la salida deseada en la capa de salida y se calcula el error como la diferencia entre la salida deseada y la salida obtenida. Posteriormente la red se corre en sentido inverso. De este modo se consigue propagar el error a cada sinapsis.

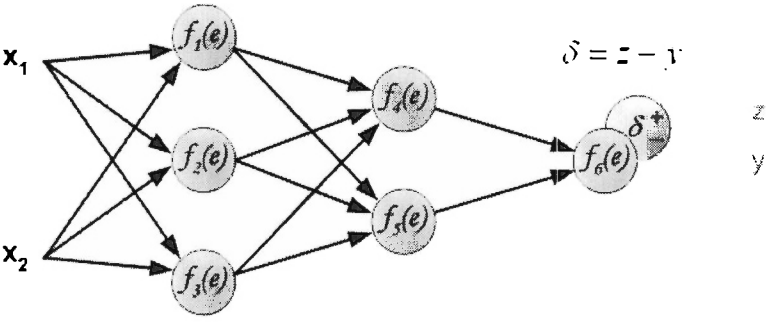


Figura 4.6. Cálculo del error en la capa de salida

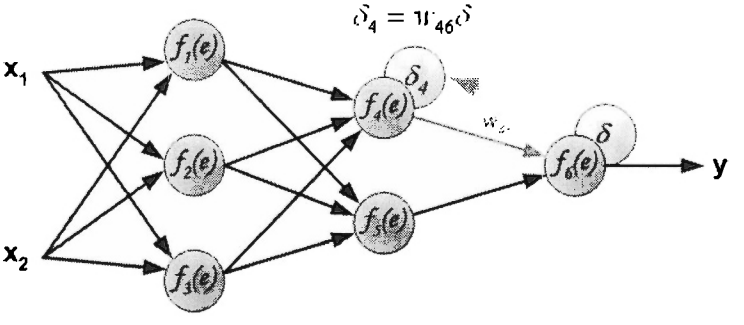


Figura 4.7. Propagación a la primera capa oculta

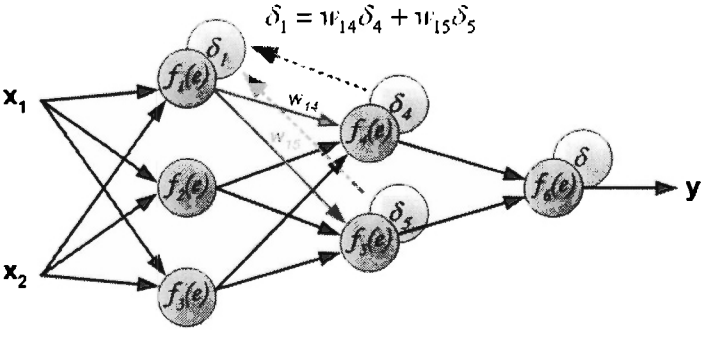


Figura 4.8. Propagación a la segunda capa oculta

- Cambio en los pesos: se modifican los pesos con los cambios obtenidos en la etapa de propagación hacia atrás.

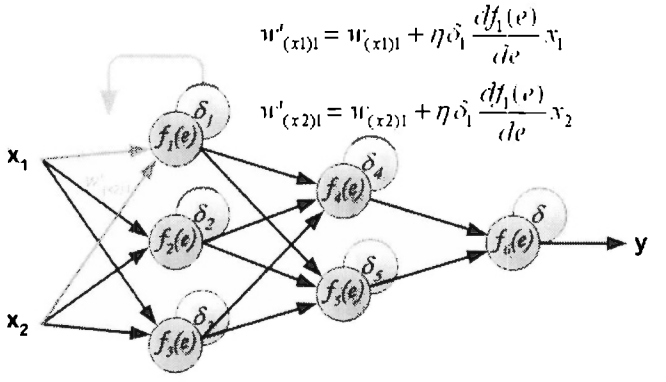


Figura 4.9. Cambio en los pesos de la primera capa oculta

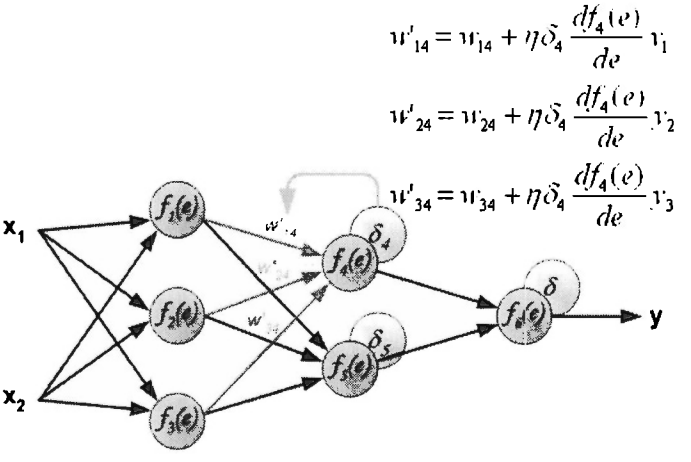


Figura 4.10. Cambio en los pesos de la segunda capa oculta

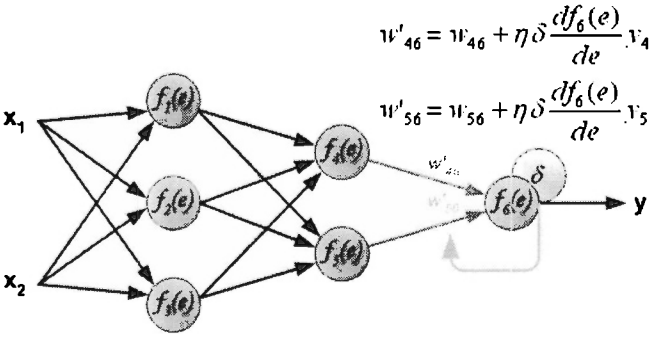


Figura 4.11. Cambio en los pesos de la capa de salida

5. Introducción a los DSP's

Son microprocesadores optimizados para aplicaciones de procesamiento digital. Sus principales características son las siguientes:

- Tienen un set de instrucciones potente y flexible para el procesamiento digital de señales, por lo que implementa de forma eficiente sumas de productos.
- Mayor velocidad de procesamiento debido a sus:
  - Arquitecturas cableadas en hardware (hardwire) que permiten ejecutar operaciones de sumas de productos en un solo ciclo de máquina.
  - Arquitecturas *pipeline* muy potentes.
  - Arquitecturas paralelas de procesamiento.
- Incorporan dentro del chip integrado dispositivos básicos en el procesamiento digital de señales, tales como :
  - DAC's y ADC's
  - Generadores de PWM
  - Controladores de DMA (Direct Memory Access)

El DSP que utilizamos en este proyecto es el C5416 de Texas Instruments, compañía líder en la manufactura de procesadores digitales de señales, y que ofrece varias soluciones enfocadas al área de aplicación de interés de los usuarios, destacando:

La familia 6000, que son DSP's de hasta 1 GHz, pensados para aplicaciones de redes inalámbricas, video digital, infraestructura de telecomunicaciones y procesamiento de imágenes.

La familia 2000 que son DSP's enfocados para aplicaciones de control. Proveen de alta integración con periféricos, y son usados en control de motores, fuentes digitales y sensores inteligentes.

La familia 5000 ofrece el consumo mas bajo de energía y buen desempeño, son usados en dispositivos móviles como celulares y PDA's, reproductores

digitales de música, voz sobre IP, módems, comunicaciones inalámbricas, y equipo médico portátil. Los DSP's de esta familia tienen como características principales:

- Procesador de 16 bits de punto fijo, con un mínimo de consumo de potencia de 40 mW.
- Uno o varios núcleos que alcanzan velocidades de procesamiento de 300 hasta 532 MIPS.
- Tres modos de ahorro de energía.
- Configuraciones de ROM y RAM integradas.

El DSP C5416, viene dentro de una tarjeta de desarrollo, la tarjeta TMS320C5416, que incluye:

- El procesador C5416 a 160 MHz.
- Soporte de JTAG por medio de USB.
- Códec de 16/20 bits stereo de alta calidad.
- Cuatro jacks de 3.5mm.
- 256 kW de memoria flash y 64 kW de RAM.
- Puerto de expansión.
- Fuente de 5V.

5.1 DSP-BIOS

El DSP BIOS es un kernel escalable de tiempo real, diseñado para las familias C5000 y C6000 de TI. Provee de utilidades como la creación de varios hilos de ejecución, un API estándar y consistente, sin sacrificar el potencial de tiempo real del procesador. El DSP BIOS es controlado desde la interfaz del programador, que es el Code Composer Studio, como se muestra en la Figura 5.1.

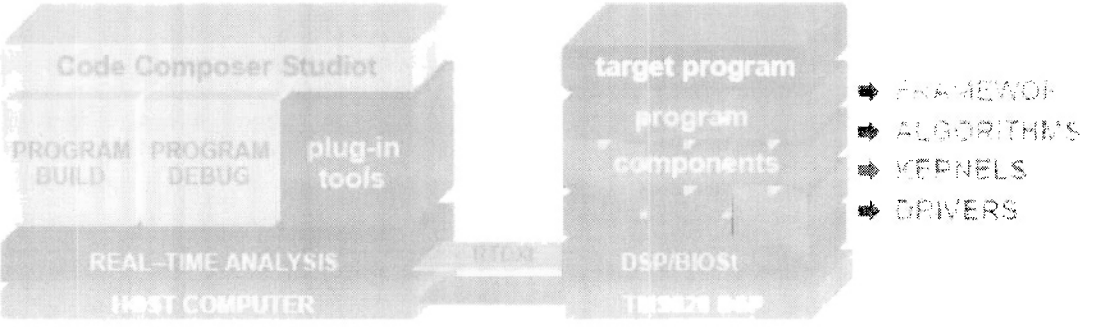


Fig. 5.1 Arquitectura DSP Bios [5]

Dentro del DSP-BIOS, existen mecanismos para el control de procesos y el acceso controlado a recursos, como semáforos, timers, tareas, mensajes, locks, y otras utilidades, como se muestra en la Figura 5.2.

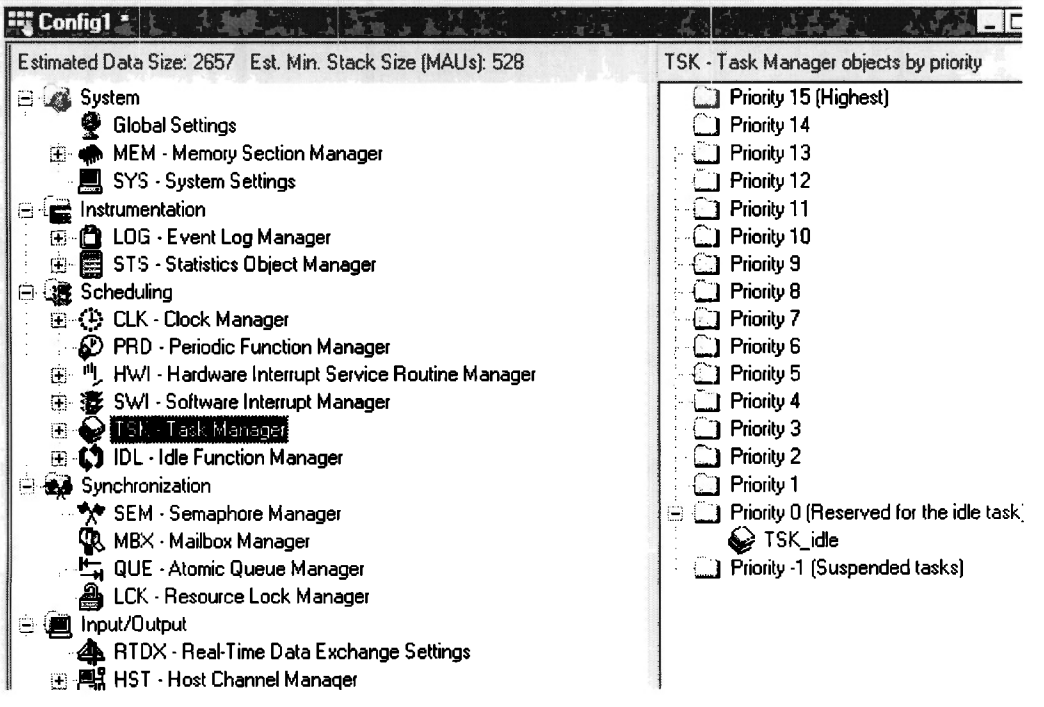


Fig. 5.2 Configuración DSP

5.2 CODEC

La tarjeta de desarrollo cuenta con un convertidor analógico digital conectado a cuatro jacks, dos de salida y dos de entrada. Este convertidor es el circuito PCM3002, y es capaz de muestrear los datos con una resolución de 16 o 20 bits hasta a 48 kHz. Este integrado requiere circuitería adicional que es provista dentro de la tarjeta así como de una lógica de control que es manejada por la CPLD de la tarjeta. Esta provee de una interfaz vía el puerto serial del DSP para que este pueda controlar al códec.

6. Sistema utilizado

La figura 6.1 muestra el diagrama a bloques del sistema implementado; consta de 4 etapas:

- Acondicionamiento: Se segmentará la señal, y cada segmento se filtrará y normalizará, para eliminar las componentes indeseables de la señal, y resaltar sus constituyentes de mayor relevancia para el reconocimiento de voz.

- Detección de segmentos vocalizados: Cada segmento de la señal será analizado mediante un método que comparará su energía con la energía de una vecindad de segmentos vecinos, esto para decidir si el segmento es vocalizado o no.

- Extracción de parámetros: Los segmentos vocalizados localizados en la etapa anterior serán procesados con la transformada wavelet para extraer características que le serán de utilidad a la red neuronal para identificar a qué vocal pertenece dicho segmento.

- Reconocimiento: Cada vector de parámetros entrará a una red neuronal que indicará a qué letra pertenece dicho vector.

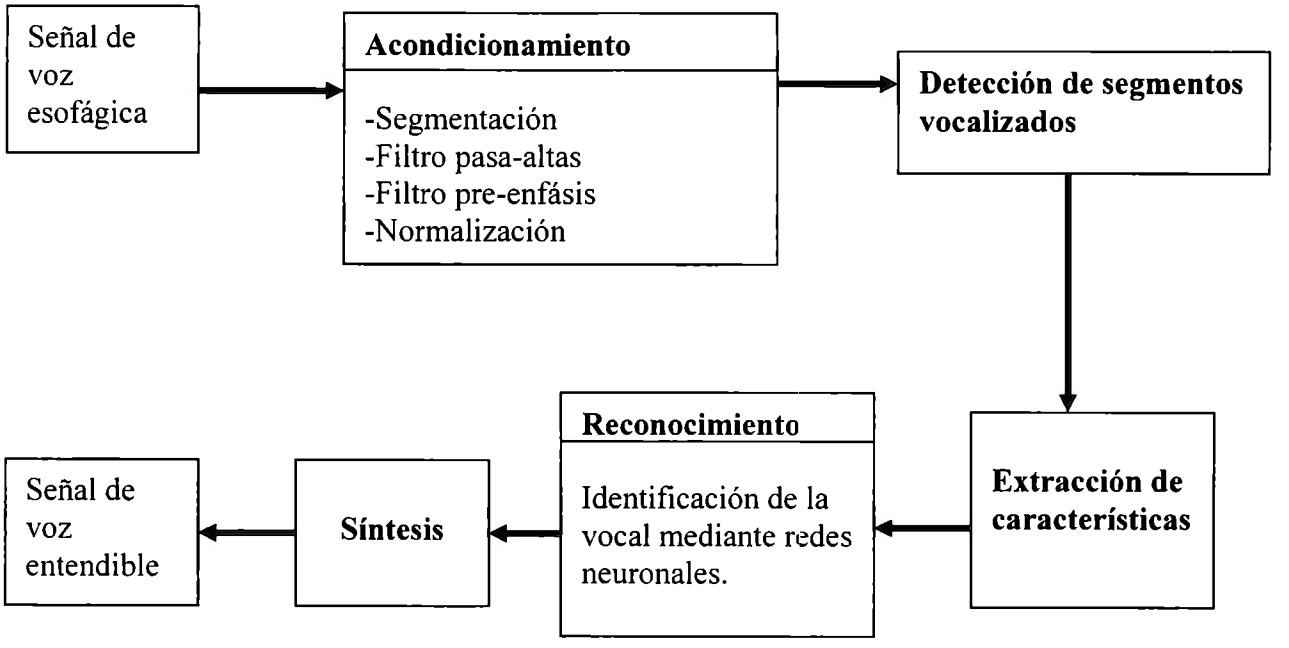


Figura 6.1 Diagrama de bloques del sistema

A continuación se describe mas detalladamente cada una de las etapas del sistema:

6.1 Acondicionamiento

La señal de audio ingresa al sistema en esta etapa (a través del códec del DSP). En ella se espera recibir como entrada el muestreo (a 8 KHz) de las variaciones en la presión atmosférica debidas a la voz esofágica. La información anterior es procesada en esta etapa con la finalidad de modificar sus características espectrales y que las mismas sean óptimas para las siguientes etapas.

El acondicionamiento de la señal comienza con la segmentación de la voz en tramas de 15 ms y 120 muestras cada una, esto permitirá que etapas posteriores no inviertan tiempo de cómputo en aquellas secciones donde no podrían existir vocales, y debido a que la identificación del contenido vocálico se realizará trama a trama; la duración de éstas debe ser tal que, en general, no contengan más de un fonema; asimismo, la información presente en ellas debe ser suficiente para llevar a cabo la identificación de su contenido.

Después de segmentar la voz se procede a filtrarla. El filtrado inicial se realiza por medio de un filtro digital de respuesta finita al impulso (FIR). Se utilizó

un filtro pasa-altas de orden 100 con una frecuencia de corte de 100 Hertz. Con ello se busca eliminar el ruido de 60 Hertz presente en las grabaciones utilizadas para probar el sistema.

Además de eliminar las componentes indeseables de la señal de audio, se desea resaltar sus constituyentes de mayor relevancia para el reconocimiento de voz. Esto se logra mediante otro filtro, esta vez un pasa-altas de dos coeficientes, llamado de pre-énfasis. Un filtro de este tipo presenta una pendiente de 20 decibeles por década, de modo que las componentes de alta frecuencia son amplificadas. Lo anterior cumple con dos tareas significativas: equilibra una atenuación de similar magnitud presente en las secciones vocalizadas de la voz e imita la sensibilidad adicional del oído humano a sonidos de frecuencias altas [1]. El filtro de pre-énfasis utilizado durante la programación de este algoritmo fue seleccionado de acuerdo con los valores típicos encontrados en la literatura sobre el reconocimiento de voz, y presenta la siguiente respuesta al impulso h:

$$h = \begin{bmatrix} 1 & -0.4 \end{bmatrix} \tag{6.1}$$

Finalmente, la etapa de pre-procesamiento de la señal concluye con la normalización de la misma. Con ello se garantiza que señales formadas por componentes semejantes posean, a la salida del bloque de acondicionamiento, la misma energía. Lo anterior será importante en secciones posteriores donde la potencia promedio de la señal se utiliza para tomar decisiones acerca de su contenido fonético. Además esto también evitará *overflow* en las operaciones del DSP.

La etapa de acondicionamiento entrega al resto del sistema una representación temporal de la señal a analizar, adecuada para el reconocimiento de los fonemas vocálicos que pudiera contener.

6.2 Detección de segmentos vocalizados.

Como se mencionó en la sección anterior, para el alcance de este trabajo sólo es necesaria la clasificación de los segmentos con contenido vocálico de las señales de voz. Esta etapa pretende identificar cada una de las secuencias provenientes del bloque de segmentación como vocalizadas o no vocalizadas. Con esta información, será posible descartar los datos que no podrían representar a una vocal antes de que estos ingresen al clasificador.

Un fragmento de voz se considera vocalizado si durante su generación el flujo de aire es alterado por las vibraciones de las cuerdas vocales del locutor. Dicho proceso aporta a la señal resultante características que permiten diferenciarla de las secciones no vocalizadas de la voz. En particular, las componentes vocalizadas presentan una mayor cantidad de energía que los sonidos no vocalizados [2]; la etapa de detección de segmentos vocalizados propuesta para este trabajo aprovecha esta característica.

Para analizar un segmento y detectar si es vocalizado o no, primero se obtiene lo que llamamos "ventana", que consta del segmento a analizar junto con medio segmento posterior y medio anterior (Figura 6.2). A fin de evitar errores debidos a discontinuidades generadas por la segmentación, cada ventana es multiplicada por una ventana de Hamming con una duración de 30 milisegundos [2], definida de acuerdo a la ecuación 6.2, esto para prevenir la pérdida de información en las transiciones de una secuencia a la siguiente.

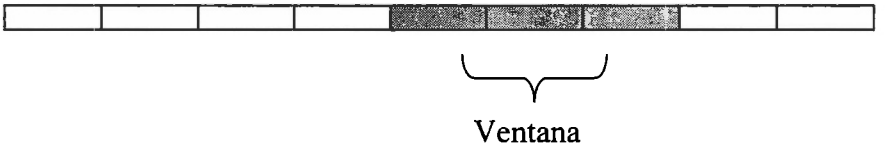


Figura 6.2 Ventana del segmento a analizar

$$W[k+1] = 0.54 - 0.46\cos\left(2\pi\frac{k}{n-1}\right), k = 0,1,\ldots,n-1 \qquad (6.2)$$

n representa al número de muestras contenidas en la ventana y puede calcularse como el producto de la duración de la misma y la frecuencia de muestreo utilizada en la señal de audio.

Se considera la potencia promedio de cada ventana; ésta puede calcularse mediante la siguiente fórmula:

$$P_{prom} = \frac{1}{n}\cdot\sum_{k=1}^{n}|x[k]|^2 \qquad (6.3)$$

donde x representa la señal contenida en el segmento a analizar y n el número de muestras que lo forman.

La energía transportada por la señal de voz depende de las condiciones durante la adquisición de los datos. Factores como el cambio de locutor, el tono al hablar o la presencia de ruido pueden modificarla. Esto hace imposible

definir un intervalo fijo para la potencia promedio que deba tener un segmento a fin de ser considerado como vocalizado. Debido a esta problemática, el algoritmo para la detección de secuencias vocalizadas compara la potencia promedio de la ventana formada con cada segmento con la energía que posee la señal en la cercanía de la misma. Específicamente se considera un intervalo de 200 milisegundos (13 segmentos: el analizado, 6 posteriores y 6 anteriores) alrededor del segmento a analizar, a este intervalo le hemos llamado "marco". Dicho marco se obtuvo de manera empírica como la duración media de una sílaba al hablar a una velocidad normal. Al tomar en cuenta que la señal podría tener intervalos de silencio o regiones largas con un comportamiento vocalizado, por ejemplo en palabras que presenten diptongos, se fijó un límite superior e inferior para el valor con el cual se comparará la potencia promedio de cada ventana. Si la energía del marco es mayor a 0.5 se considera 0.5, si es menor a 0.33 se considera 0.33, y si está entre 0.33 y 0.5 se considera la energía calculada.

Siguiendo con el mecanismo para el etiquetado de los segmentos propuesto por Greenwood, et. al. [2], se fija un umbral. Si el cociente de la potencia promedio de la ventana entre la que presenta el marco (con los límites descritos en el párrafo anterior) es menor al umbral, la trama es considerada no vocalizada, si dicho cociente es mayor al umbral, entonces los datos se clasifican como vocalizados.

La salida de este bloque consiste en una cadena de valores lógicos (uno por cada segmento) que indican si el contenido de la señal es o no predominantemente vocalizado.

6.3 Extracción de parámetros

Esta etapa se encarga de generar la información que se empleará como entrada para el clasificador; para ello se recurre a la transformada *wavelet*, herramienta con la que se obtendrá el vector de parámetros que se pasará como entrada a la red neuronal.

Al trabajar con señales en tiempo discreto, estamos obligados a muestrear el eje del tiempo (traslación en el caso de la transformada *wavelet*), esto quiere decir que el parámetro τ en la ecuación de la transformada *wavelet* hija:

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \cdot \psi\left(\frac{t-\tau}{s}\right) \tag{6.4}$$

no puede tomar cualquier valor real. Este fenómeno es común al utilizar sistemas digitales para el procesamiento de una señal. Esto se realiza con dos objetivos en mente: obtener una discretización de la transformada *wavelet* que pueda ser calculada en una computadora e incorporar el principio psicoacústico de las bandas críticas en la descomposición planteada.

La extracción de características propuesta en este trabajo busca la descomposición de la señal en bandas que asemejen la respuesta en frecuencia de la cóclea. Lo anterior obliga a realizar un muestreo del plano escala-traslación distinto al usual. Específicamente, este proyecto pretende efectuar una descomposición de la señal que muestre de manera directa sus componentes para cada una de las bandas críticas definidas por la escala Bark.

La escala de Bark es una mapeo logarítmico de la frecuencia. Ésta fue creada de forma tal que la resolución espectral del oído humano sea de un Bark para cualquier frecuencia característica. Al definirse con base en un parámetro biológico, no es posible hallar una expresión exacta que relacione la frecuencia en Hertz con la misma en Barks. Sin embargo diversos autores han propuesto interpolaciones basadas en mediciones experimentales. Para fines de este proyecto se empleará la interpolación propuesta por Schroeder et. al. [3].

$$Z = 7\ln\left(\frac{f}{650} + \sqrt{\left(\frac{f}{650}\right)^2 + 1}\right) \tag{6.5}$$

La escala presentada determina a veinticuatro frecuencias críticas. Estas se definen mediante incrementos de 1 Bark a lo largo de toda la banda audible.

Con base en la interpolación mostrada, podemos definir un muestreo de la escala apropiado para la transformada *wavelet* continua de forma tal que la descomposición de la señal de voz entregue de manera directa la información referente a cada una de las bandas críticas del oído humano:

$$s_j = \frac{1}{325} \cdot \frac{e^{\frac{j}{7}}}{e^{\frac{2j}{7}} - 1} \quad j = 1,2,\ldots \tag{6.6}$$

Al efectuar la segmentación de la señal, se eligieron tiempos de trama y ventana adecuados para el reconocimiento de fonemas individuales. Aprovechando esto, se busca crear un vector de parámetros para cada

segmento. En este punto, las porciones de la señal que fueron identificadas como no vocalizadas son descartadas. Esto evitará que el clasificador deba ajustarse a discriminar los sonidos no vocalizados y pueda estar dirigido únicamente a distinguir entre las vocales.

Además de la información que presenta directamente la transformada *wavelet*, resulta útil para el clasificador contar con la energía total de la señal previa a la descomposición. Esto se debe a que, al trabajar con señales físicas, la energía total de un sistema siempre es conservada. Lo anterior, en conjunto con la normalización efectuada en la primera etapa de este algoritmo, permitirá al clasificador realizar una discriminación entre los posibles fonemas vocálicos contenidos en un segmento, considerando las diferencias energéticas involucradas en la producción de las distintas vocales.

En el diseño del vector de parámetros se debe contemplar la naturaleza del clasificador a utilizar. Específicamente, se propone el uso de una red neuronal artificial con una arquitectura de alimentación hacia delante. Al no poseer elementos de retraso, y recordando que las señales de voz no son estacionarias, encontramos la necesidad de incluir en el vector de parámetros información sobre la dinámica espectral de la entrada al sistema. Esto se logra al añadir componentes definidos como el cambio entre el vector anterior y el actual.

Tomando en cuenta lo anterior, podemos ahora definir al vector de parámetros como la concatenación de los siguientes componentes:

- Energía total del segmento
- Energía contenida en cada uno de los niveles de la descomposición usando la transformada wavelet
- Cambio en la energía de la señal con respecto al segmento anterior
- Cambio en la energía contenida en cada descomposición con respecto al segmento previo

La longitud del vector de parámetros dependerá entonces del número de bandas utilizadas para la descomposición. Este valor, como se vio con anterioridad, está a su vez limitado por la frecuencia de muestreo de la señal de entrada. En particular, si se utiliza una frecuencia de muestreo de 8 KHz, el vector de parámetros estará formado por 36 componentes.

6.4 Reconocimiento:

La etapa de reconocimiento se realiza con una Red Neuronal Artificial de varias capas antealimentadas o Perceptrón multicapa con la siguiente topología:

1. Capa de entrada: 36 neuronas
2. Primera capa oculta: 48 neuronas
3. Segunda capa oculta: 16 neuronas
4. Capa de salida: 5 neuronas

La interconexión entre las capas es completa, es decir, todas las neuronas de una capa cualquiera están conectadas con todas las neuronas de las capas anterior y posterior inmediata.

Para garantizar el funcionamiento adecuado de esta etapa es necesario hacer el entrenamiento de la red. Para esto se implementaron tres algoritmos de entrenamiento con el mismo funcionamiento y los mismos parámetros, pero con diferentes esquemas de realización, es decir, todos se basan en las definiciones generales de la Propagación hacia atrás desarrolladas y obtenidas en el capítulo 4.

El primero tiene la peculiaridad de que los pesos de toda la red están almacenados en tres vectores. La implementación de este algoritmo resultó complicada debido a que el error entrante a cada nodo se obtiene a través de la suma de los pesos que lo conectan con la capa siguiente, por lo que fue necesario encontrar una manera adecuada de recorrer los vectores en los que se almacenaban los pesos. Esto era crítico, ya que los cambios obtenidos tenían que ser asignados al peso correspondiente y si no se hacía correctamente se hubiera podido dar el caso en el que el error no se minimizara.

El diagrama de flujo del algoritmo se muestra en la Figura 6.3.

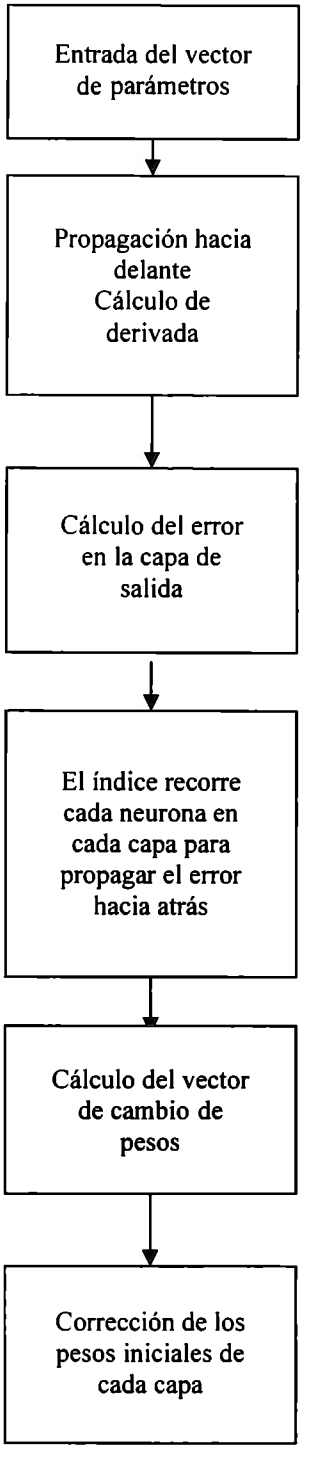


Figura 6.3 Diagrama de flujo del algoritmo del primer programa

El segundo funciona a través de una representación matricial del algoritmo de entrenamiento, que consta de los siguientes pasos:

1. Almacenar las derivadas de la salida en matrices diagonales $\mathbf{D}_n$

2. Almacenar el error en la capa de salida en un vector **e**.

3. El error en los nodos anteriores a la capa de salida es:

$$\delta^{(o)} = \mathbf{D}_o \mathbf{e} \tag{6.7}$$

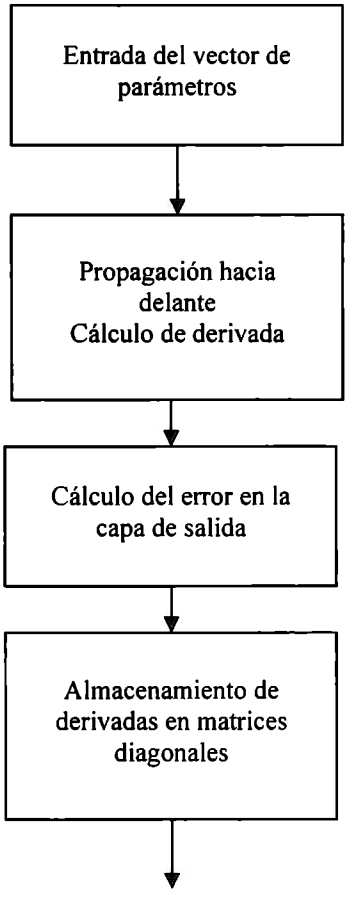
4. Se propaga el error a cada una de las capas ocultas con:

$$\delta^{(l)} = \mathbf{D}_l \mathbf{W}_{l+1} \delta^{(l+1)} \tag{6.8}$$

5. El cambio en los pesos de cada capa se obtiene con:

$$\Delta \mathbf{W}_l^T = - \gamma \, \delta^{(l+1)} \, \mathbf{o}^l \tag{6.9}$$

Éste algoritmo resultó mucho más fácil de implementar y el tiempo de cómputo del entrenamiento fue mucho menor. A continuación se muestra el diagrama de flujo del algoritmo:



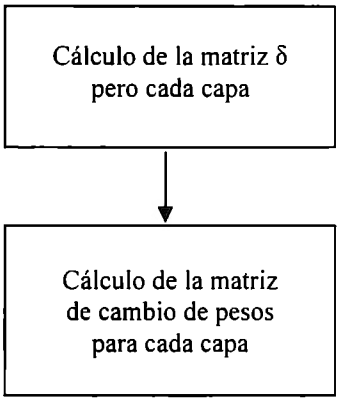


Figura 6.4 Diagrama de flujo del algoritmo del segundo

Por último se programó un algoritmo que funciona con matrices, al igual que el anterior, pero con algunas modificaciones en el acomodo de las matrices que almacenan las derivadas. Esta modificación reduce un paso en el cálculo de las matrices para el cambio de pesos, que se puede calcular directamente como:

$$\mathbf{D}_o = \mathbf{E}$$

$$\mathbf{D}_l = -\gamma \frac{\partial E}{\partial w} \mathbf{W}_{l+1} \mathbf{D}_{l+1} \tag{6.10}$$

$$\Delta \mathbf{W}_l = -\gamma \mathbf{D}_l \mathbf{o}_{l-1} \tag{6.11}$$

Los códigos correspondientes a estas implementaciones se adjuntan en el Anexo B.

7. Formato multitareas

Las etapas anteriores se deben implementar en un formato multitareas, lo que significa que mientras se adquieren datos (voz), éstos se vayan procesando y se obtengan los resultados pertinentes. Este funcionamiento permitiría al sistema desempeñarse en tiempo real.

Para lograr esto en un DSP, se utiliza un *kernel* de tiempo real llamado DSP-BIOS (ya explicado en el capítulo 5). Las problemáticas que se resolvieron para lograr el formato multitareas fueron las siguientes.

7.1 Mapeo de memoria

El primer problema enfrentado para conseguir el formato multitareas fue el agotamiento de la memoria de programa. Esto sucedió al juntar en un solo código todas las etapas implementadas anteriormente.

Para resolver este inconveniente se procedió a investigar el mapa de memoria que el DSP tenía por defecto, para esto ingresamos al proyecto, en la parte de DSP-BIOS, se selecciona "System", y después en "MEM" en ese momento se desplegarán las partes en que se divide la memoria, como se ve en la Figura 7.1.

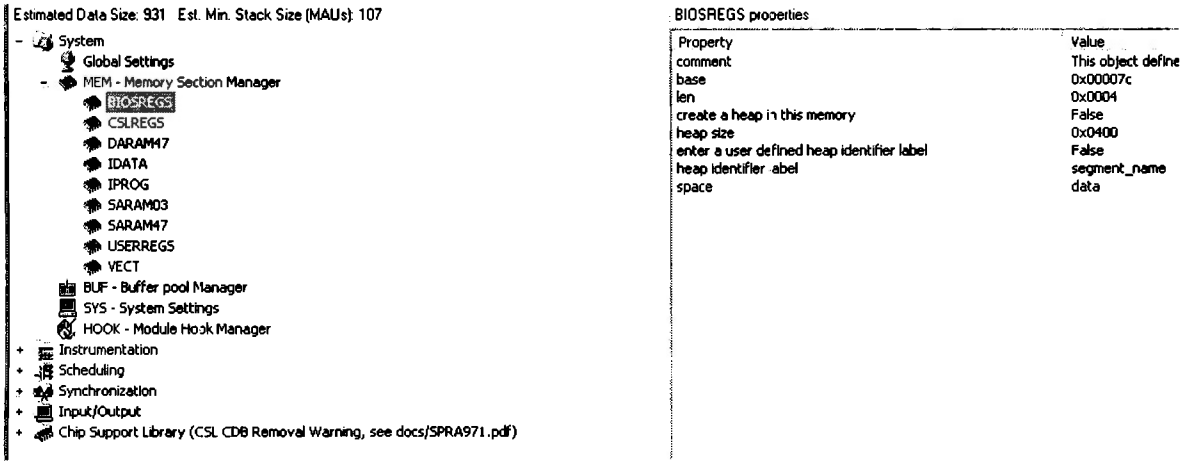


Figura 7.1 Despliegue en DSP/BIOS de las secciones de memoria.

Al seleccionar cualquier opción se abre una ventana en la cual se muestra la dirección de inicio (base) y longitud de memoria (len) de esa sección; con esta información se procedió a realizar un mapa simplificado de la memoria estándar del DSP, el cual se muestra en la Figura 7.2.

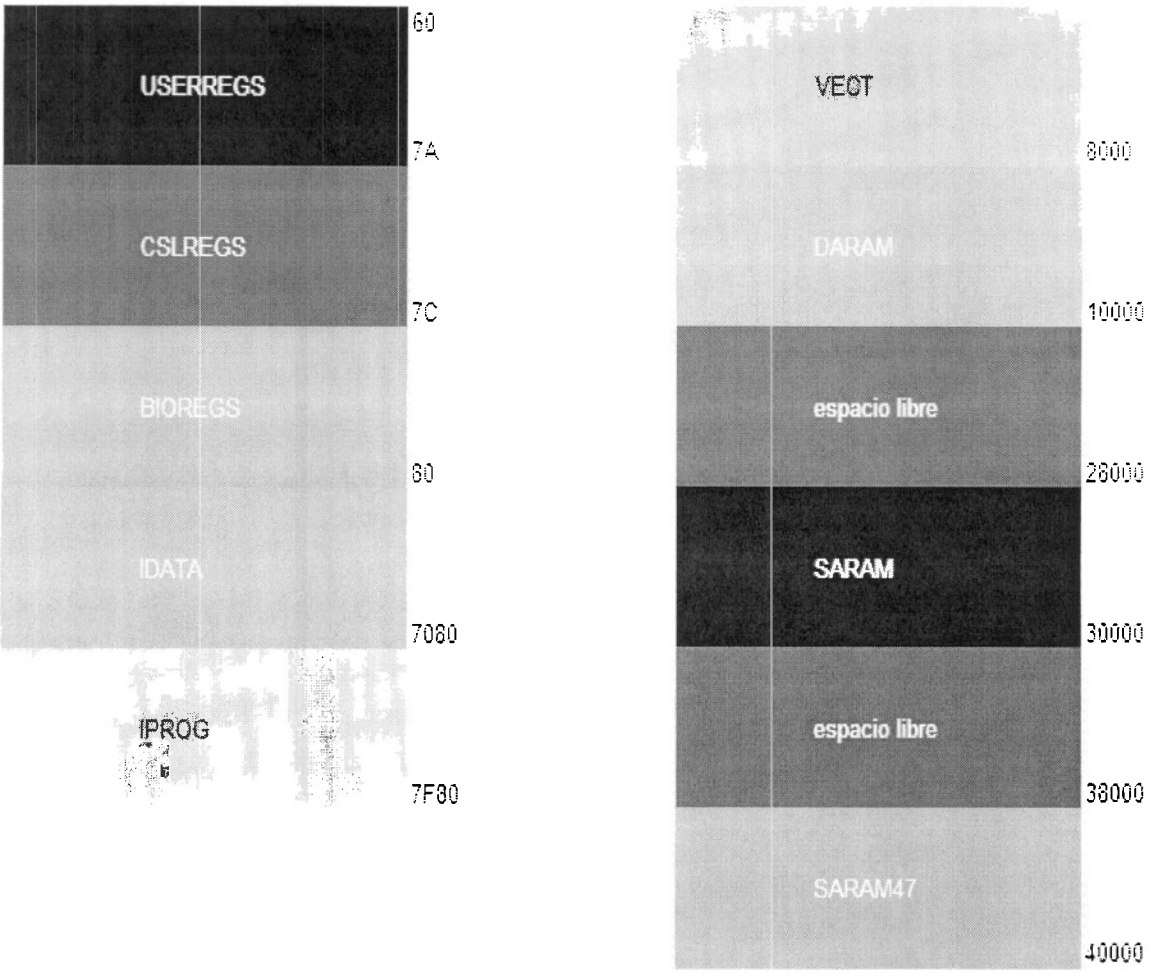


Figura 7.2 Mapa estándar de memoria del DSP

IPROG es la memoria de programa, y debíamos incrementar su tamaño, pero justo después de ella está VECT, así que tuvimos que buscar un lugar donde colocar a IPROG. Finalmente se configuró una forma de memoria extendida en la que se cambió de lugar a IPROG y se aumentó su capacidad. En la Figura 7.3 se observa que la longitud del espacio de memoria del programa (IPROG) era F00h (3840 direcciones), y comenzaba en la dirección 7080h. Lo que hicimos fue cambiar de posición a la memoria de programa (IPROG) en el espacio de memoria de DARAM 4-7(Dual Access RAM), como se muestra en la Figura 7.4; así la nueva dirección es asignada como la 18000h, con una longitud de 8000h, suficiente para almacenar todo el código de nuestro programa.

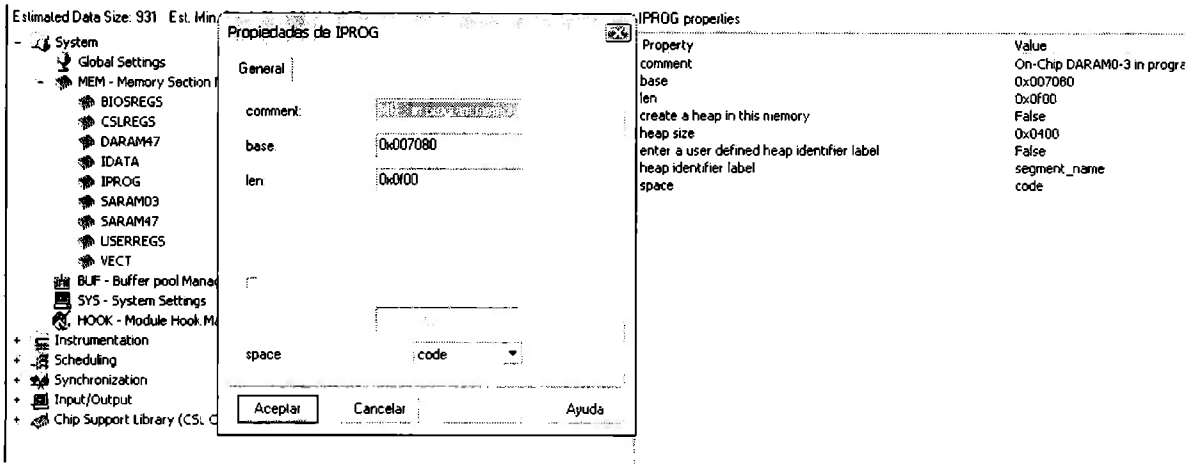


Figura 7.3. Dirección estándar de IPROG en *el DSP.*

Es importante mencionar que modificar las direcciones de memoria no resulta trivial debido a que al mapear en una sección incorrecta se pueden presentar problemas como un mal funcionamiento de interrupciones o de las tareas del DSP (En ocasiones este tipo de errores no son mostrados por el compilador del DSP.).

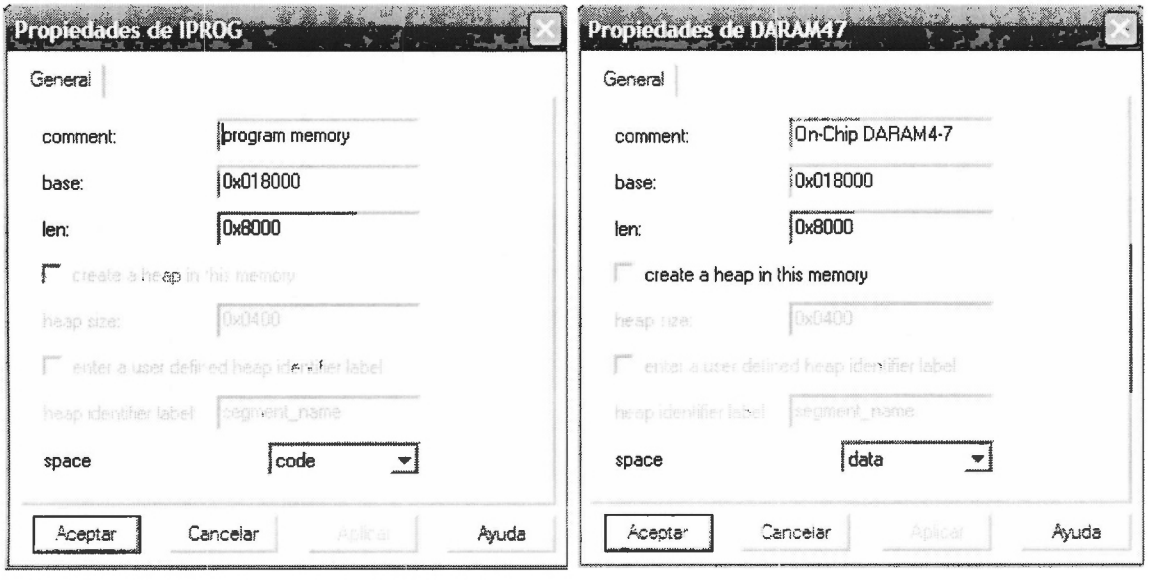


Figura 7.4. Mapeo de memoria de programa (IPROG) y DARAM 4-7

Se muestra en la Figura 7.5 que la solución fue correcta y no se presenta otro problema de ese tipo en la memoria.

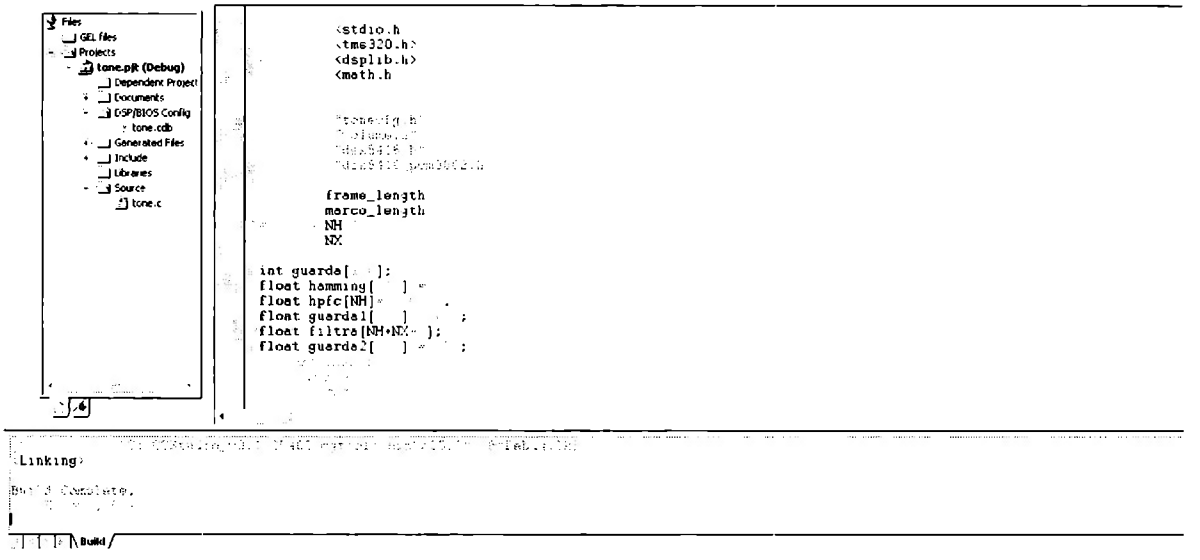


Figura 7.5 No hay error en el redireccionamiento de memoria

7.2 Sincronización de tareas

Ya resuelto el problema de memoria se procedió a investigar cuáles son los tipos de tareas que DSP-BIOS permite crear y cómo se pueden sincronizar entre ellas. A continuación se describen los tipos de tareas que usamos en el proyecto:

- PRD. Son tareas que se realizan periódicamente cada cierto número de *ticks* del reloj. Un *tick* es un periodo configurable del *timer* del DSP.
- TSK. Son tareas que se realizan una sola vez.

Nosotros juntamos todas las etapas del sistema (acondicionamiento, detección de segmentos vocalizados, extracción de parámetros y clasificación) en una sola función del código llamada Análisis. También creamos otra función a la que nombramos Adquisición encargada de tomar cada vez que se ejecuta (125 μs) una muestra de voz a través del códec del DSP. Estas dos funciones deben realizarse simultáneamente, es decir, mientras se realiza el análisis de un segmento se deben estar tomando muestras, ya que si dejáramos de adquirir voz, se perdería información y el funcionamiento sería incorrecto.

Análisis y Adquisición se comienzan a ejecutar después de haberse llevado a cabo la función Arranque, una función de inicio que permite adquirir los primeros trece segmentos necesarios para iniciar el análisis. La Figura 7.6

muestra un diagrama donde se muestra el funcionamiento general del programa.

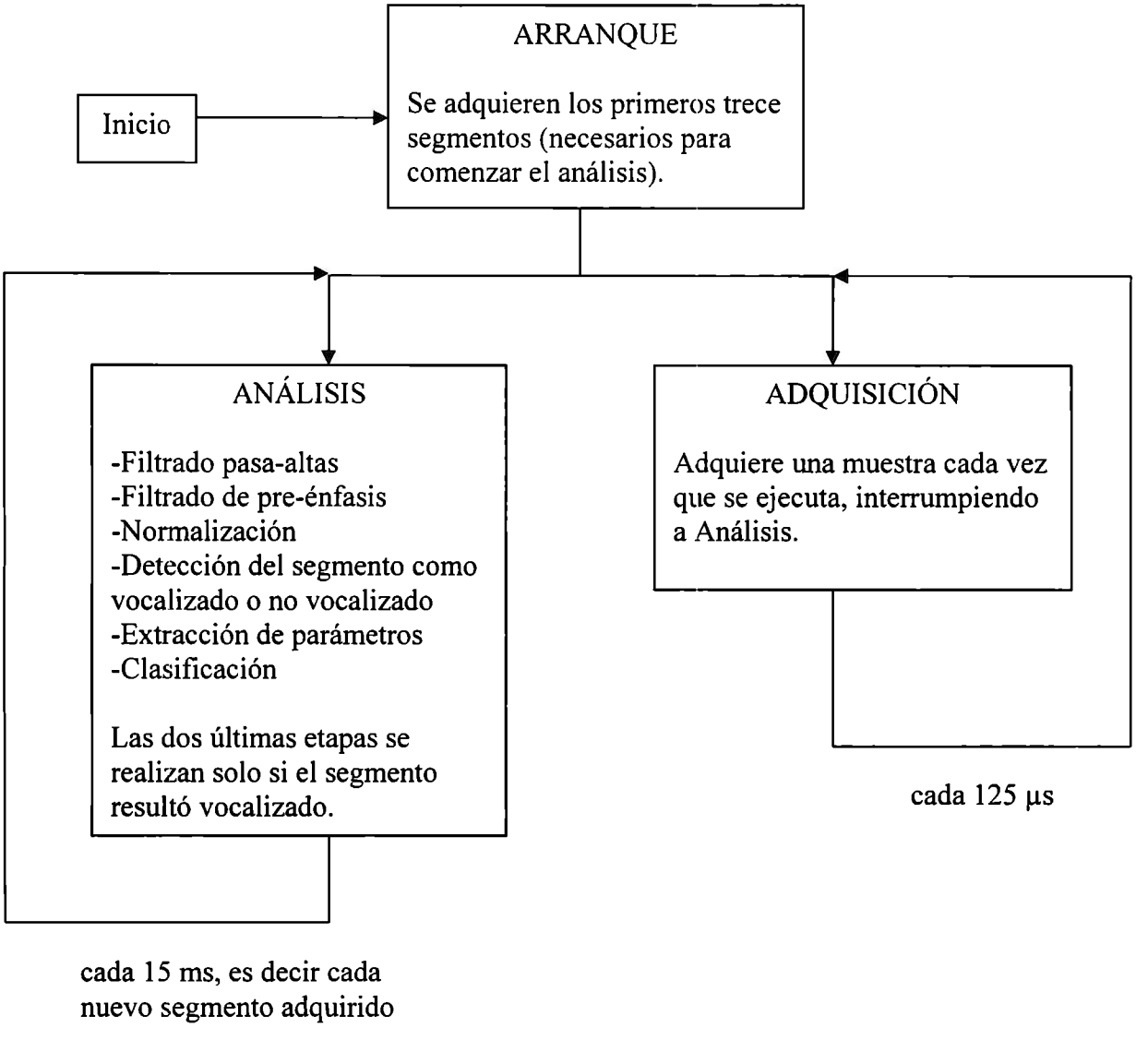


Figura 7.6 Diagrama del funcionamiento del programa

Debido a que tanto Adquisición como Análisis deben estar ejecutándose periódicamente (cada 125 µs y cada 15 ms respectivamente), primero pensamos en activarlas como PRD's, pero surgió la complicación de que cuando entraba Análisis, no dejaba ejecutarse a Adquisición sino hasta que terminara de realizar el análisis al segmento correspondiente. Entonces tuvimos que buscar una forma alterna de sincronizar estas funciones.

Para solucionarlo decidimos cambiar de enfoque; después investigamos que un PRD podía interrumpir a una TSK, por tener una mayor prioridad, así que decidimos dejar a Adquisición como un PRD y activar a Análisis como una TSK, como se ve en la Figura 7.7, así sí podría ser interrumpida para que Adquisición (con una prioridad mayor) entrara cada 125 μs a tomar una muestra.

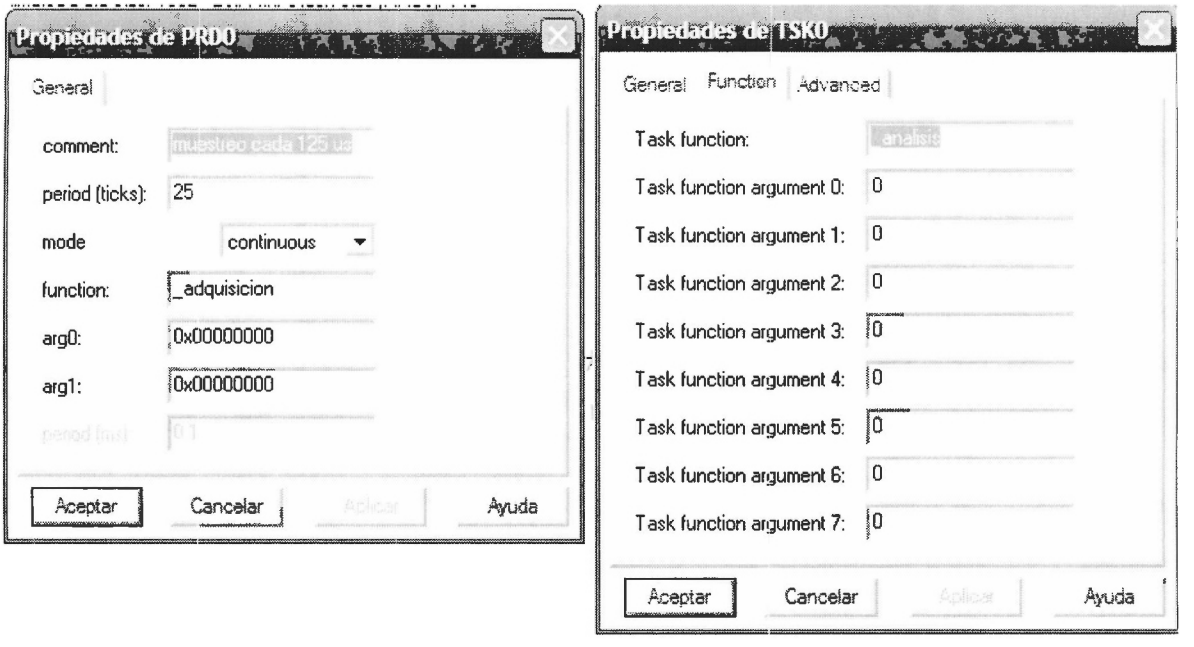


Figura 7.7 Función adquisición() activada como PRD y función análisis() activada como TSK

Pero surgió un nuevo problema: la TSK no es periódica. Esto se resolvió mediante un ciclo infinito dentro del código de Análisis que se mantuviera preguntando por una bandera que activaría Adquisición cuando ésta ya hubiera adquirido 120 muestras, es decir, un segmento completo. En ese momento Análisis ya podría comenzar a analizar dicho segmento, y al terminar se quedaría preguntando nuevamente por la bandera y lo más importante, durante ese tiempo se continuaría adquiriendo mediante interrupciones a la función Análisis.

8. Resultados

8.1 Acondicionamiento

8.1.1 Segmentación

La Figura 8.1 muestra en la gráfica superior un conjunto de muestras de voz adquiridas a través del códec del DSP que conforman 250 ms de grabación. La gráfica inferior muestra el primer segmento de este conjunto de voz; se puede apreciar la duración del segmento, 15 ms. Estas gráficas fueron tomadas cuando el programa adquiría 1.5 segundos de voz, es decir 100 segmentos, y los analizaba todos. Pero cuando el sistema se implementó en formato multitareas ya solo se almacenan trece segmentos (los necesarios para realizar un análisis) que se van recorriendo conforme se adquieren segmentos nuevos, es decir están en un buffer circular que elimina el segmento mas antiguo para alojar al nuevo.

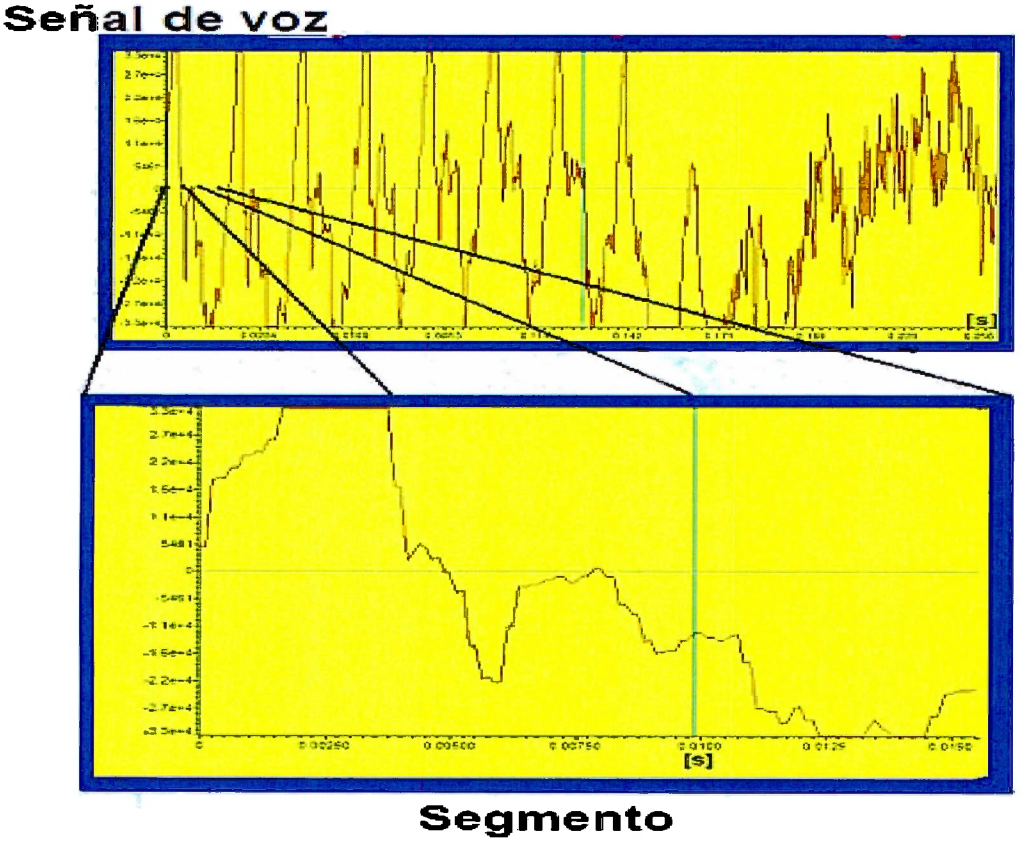


Figura 8.1 Gráficas de voz. Arriba: conjunto de 250 ms. Abajo: Segmento de 15 ms

8.1.2 Filtrado pasa-altas

En la figura 8.2 se muestra el resultado de aplicar el filtro pasa-altas a una señal de prueba que consta de la suma de señales seno a 50, 200 y 500 Hz. Estas gráficas se obtuvieron con MatLab para compararlas con las que obtuvimos en el DSP. Recordamos que este filtro tiene una frecuencia de corte de 100 Hz.

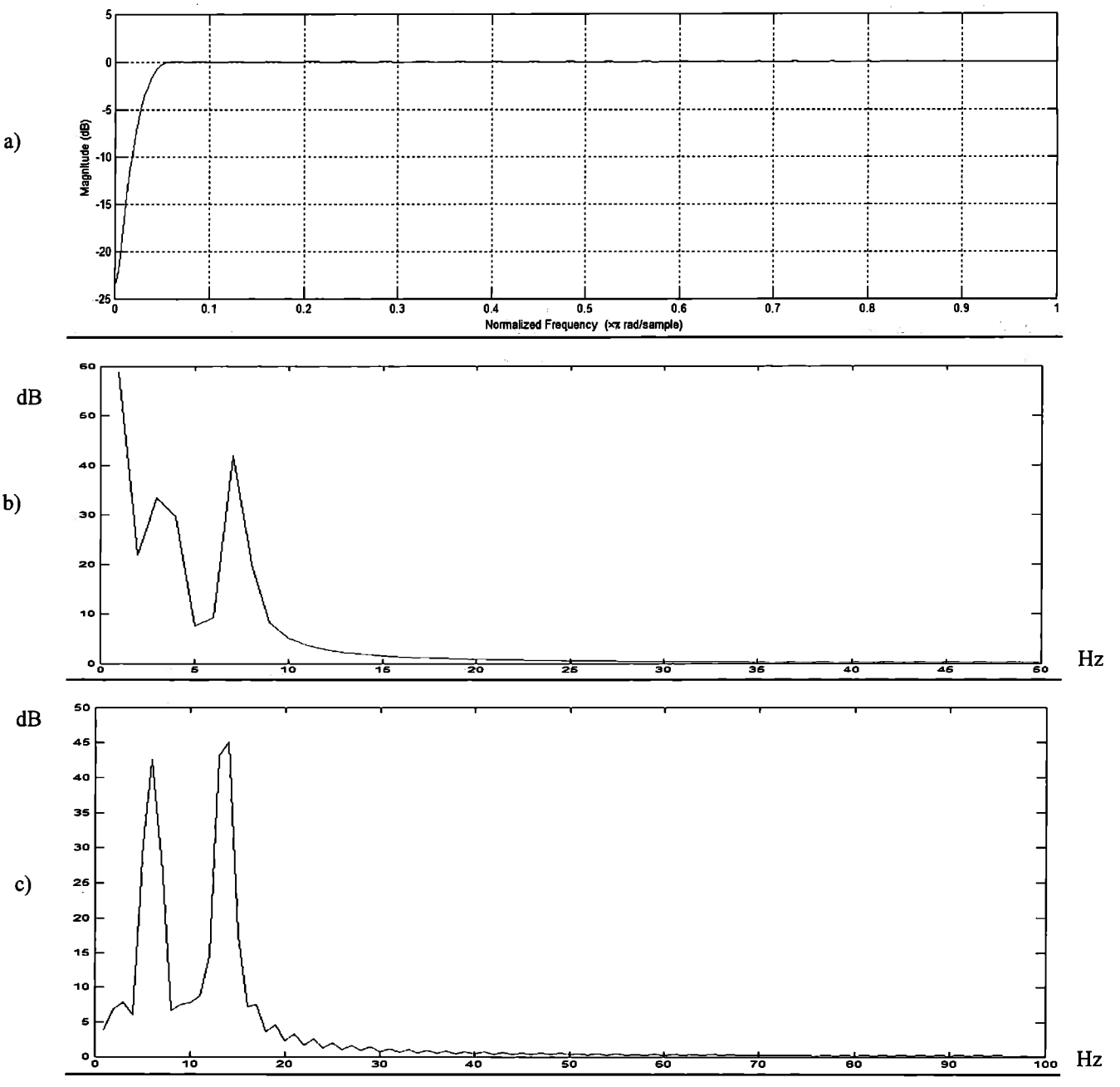


Figura 8.2 Filtrado pasa-altas en MatLab. a) Respuesta del filtro b) Señal de prueba c) Señal filtrada

En la figura 8.3 se muestra el mismo filtro anterior con la misma señal de prueba pero ahora en el DSP.

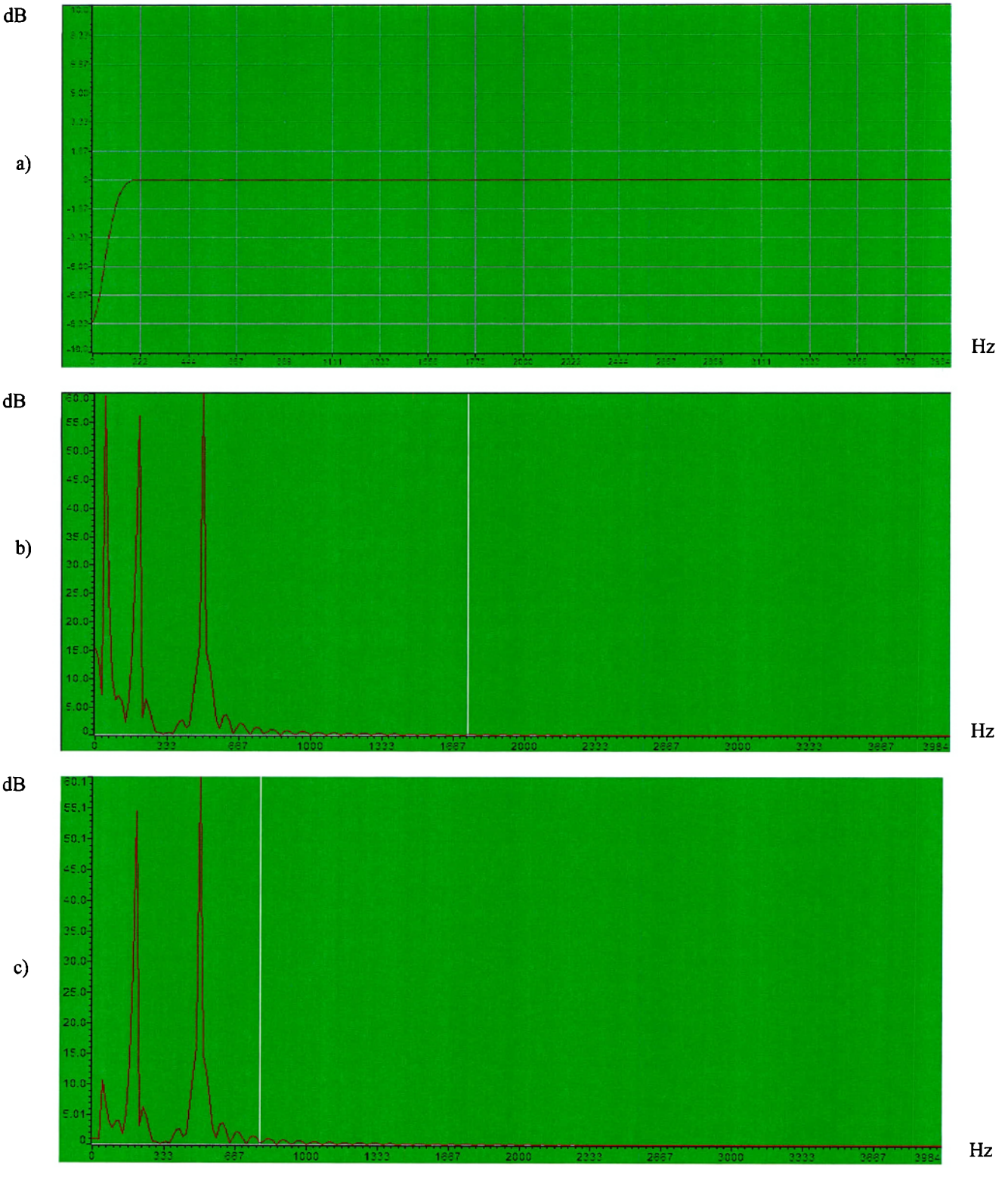


Figura 8.3 Filtrado pasa-altas en el DSP a) Respuesta del filtro b) Señal de prueba c) Señal filtrada

A continuación, en la figura 8.4, se presenta el filtrado de un segmento de señal de voz con el mismo filtro pasa-altas.

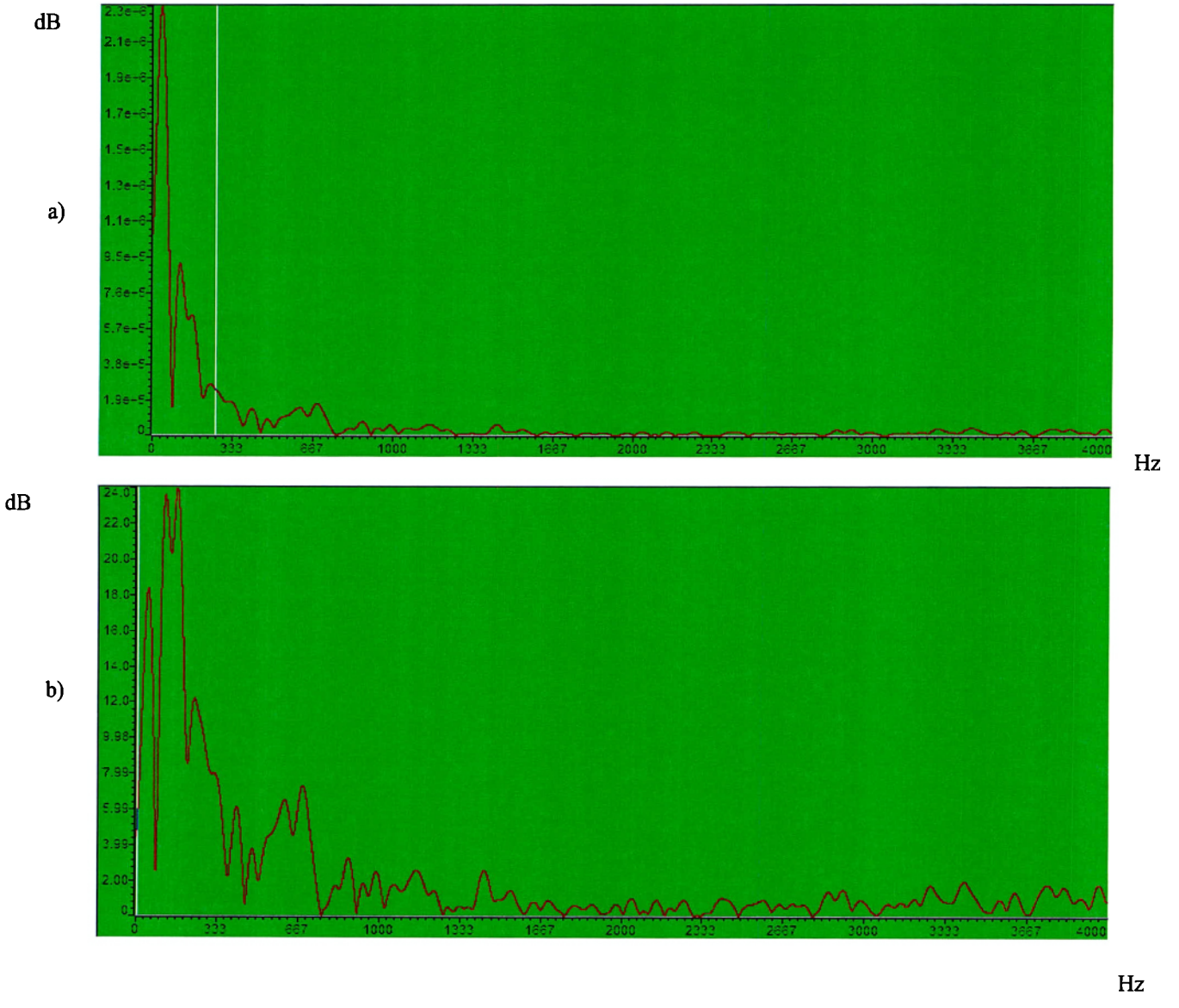


Figura 8.4 Filtrado pasa-altas en el DSP a) Señal de voz original b) Señal de voz filtrada

8.1.3 Filtrado pre-énfasis

En la figura 8.5 se muestra el resultado de aplicar el filtro pre-énfasis a una señal de prueba que consiste de la suma de 3 señales seno de 100, 1000 y 2000 Hz. Recordamos que este filtro tiene la intención de enfatizar los componentes de alta frecuencia de la señal, los cuales presentan secciones vocalizadas de la voz.

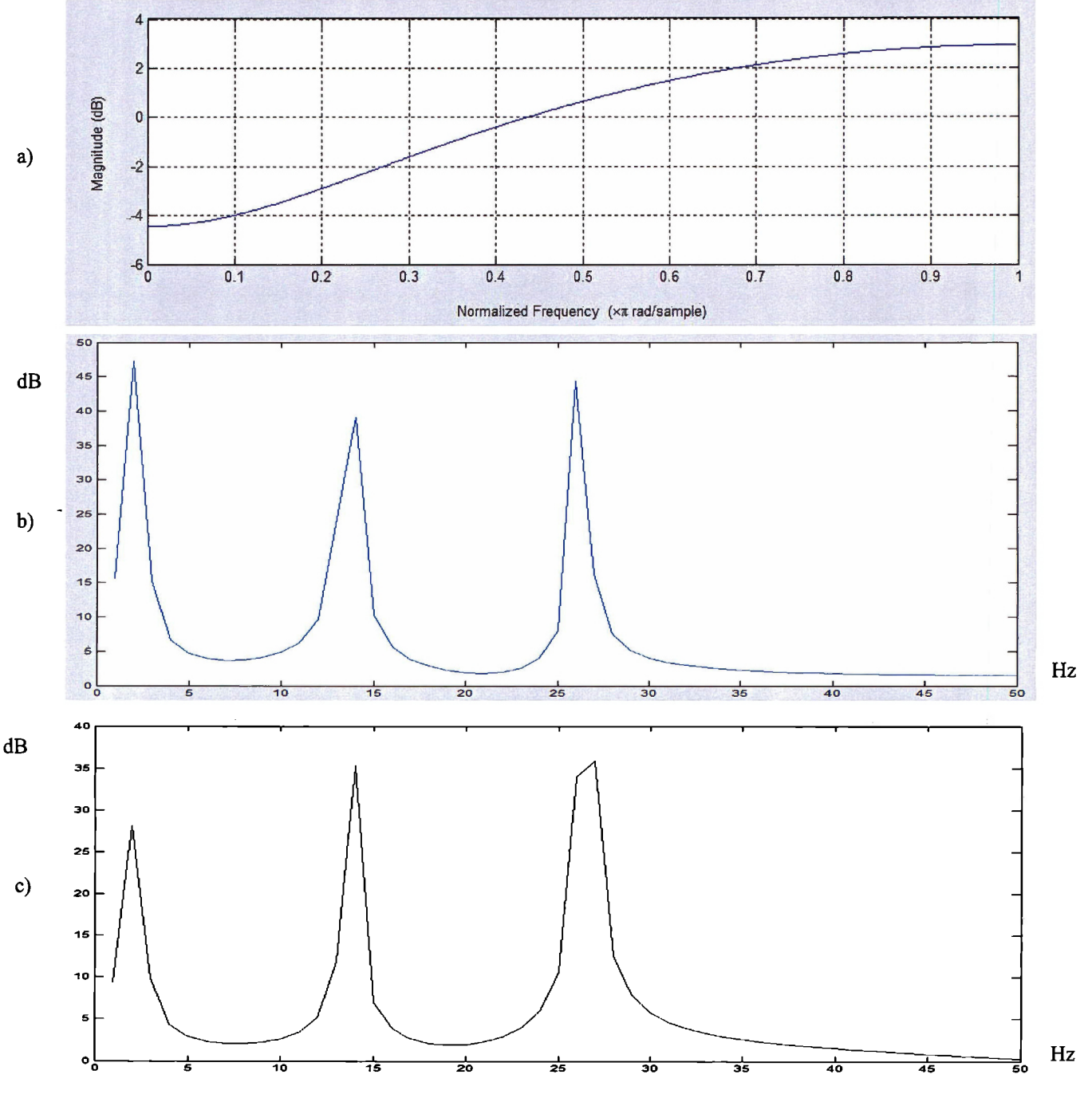


Figura 8.5 Filtrado pre-énfasis en MatLab a) Respuesta del filtro b) Señal de prueba c) Señal filtrada

En la figura 8.6 se muestra el mismo filtro pre-énfasis con la misma señal de prueba, pero ahora en el DSP.

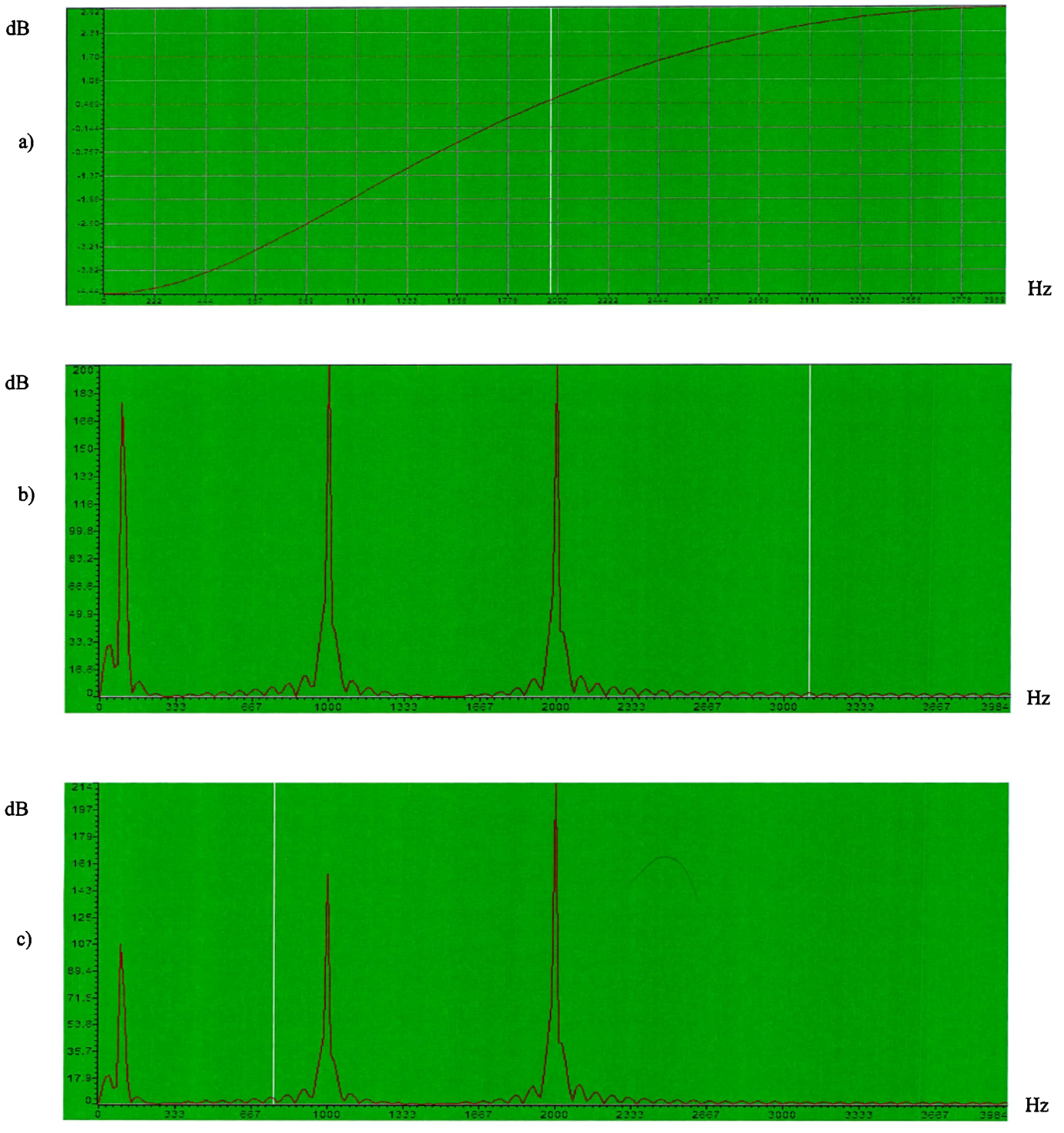


Figura 8.6 Filtrado pre-énfasis en el DSP a) Respuesta del filtro b) Señal de prueba c) Señal filtrada

8.2 Detección de segmentos vocalizados

La figura 8.7 muestra los resultados del buffer que almacena los valores lógicos (0 para no vocalizado y 1 para vocalizado) de la detección de segmentos vocalizados de tres palabras.

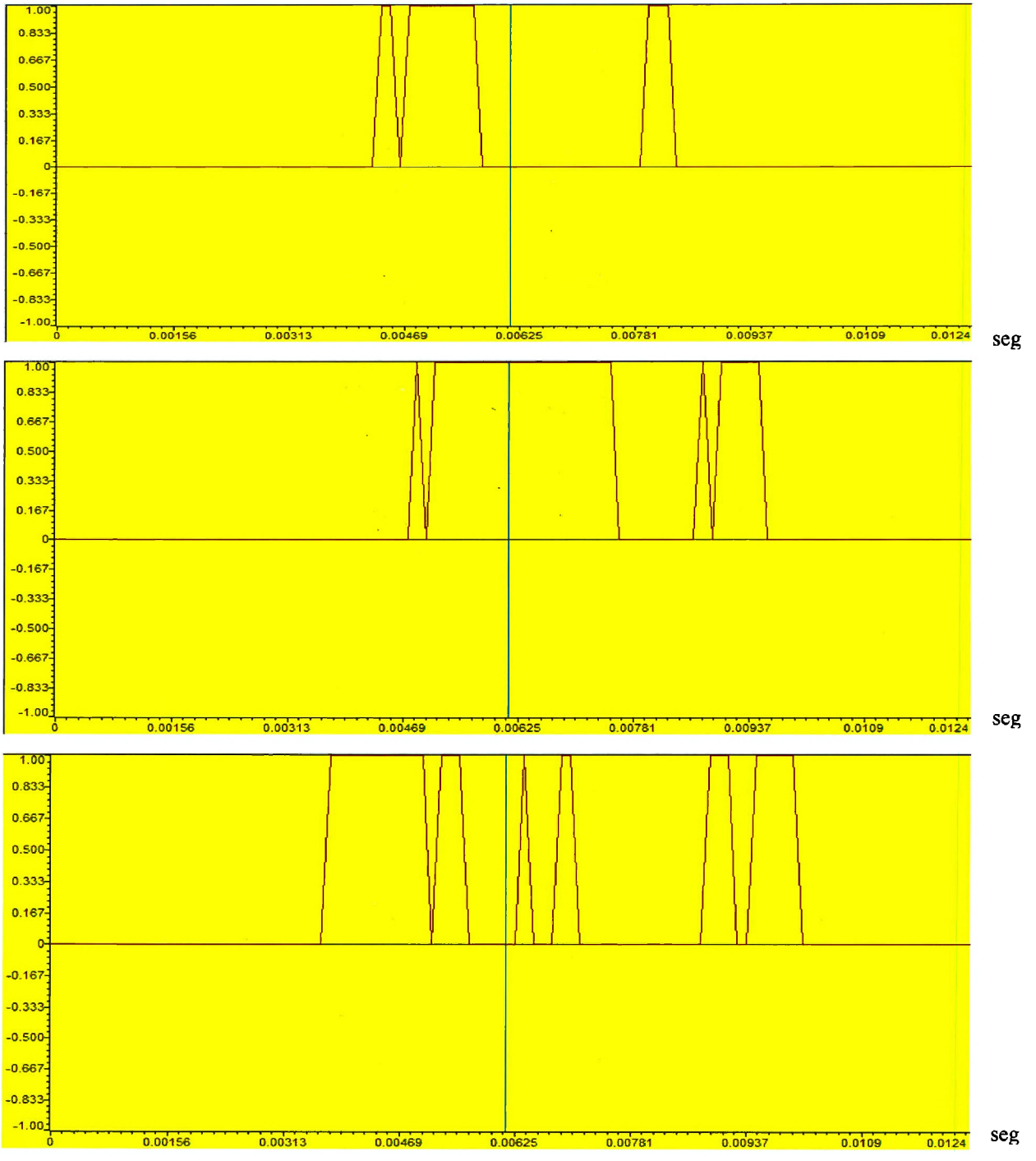


Figura 8.7 Detección de segmentos vocalizados para las palabras: a) pato b) huevo c) perico

8.3 Extracción de Parámetros

Los datos que ingresan a esta sección son los segmentos que han sido etiquetados como vocalizados por las etapas anteriores.

Las *wavelets* hijas se obtienen desplazando en frecuencia a la wavelet madre, con las frecuencias que se obtuvieron de las interpolaciones de la escala de Bark, y después muestreándolas a 8 KHz. Utilizando las ecuaciones 4.3 y 4.8 obtenemos las *waveltes* hijas.

En las ecuacione 4.3 eje "s" representa al muestreo o descomposición en frecuencia, mientras que el eje τ al muestreo en el tiempo, las wavelets hijas obtenidas se pueden observar en el anexo A.

El procedimiento para obtener el vector de parámetros consiste en:

1. Hacer la correlación entre el segmento de entrada y cada una de las wavelets hijas.

2. Hacer la correlación entre el segmento vocalizado anterior y las wavelets hijas.

3. Obtener la energía de cada una de estas funciones evaluando la autocorrelación en cero.

4. Calcular la energía del segmento vocalizado.

5. Calcular la energía del segmento anterior.

6. Formación del vector de parámetros.

Los primeros 17 valores del vector de parámetros se obtienen con las energías de las funciones de correlación del segmento vocalizado con las wavelet hijas, los 17 siguientes son las diferencias entre las energías de las correlaciones del segmento vocalizado con las *wavelet* hijas y las energías de las correlaciones del segmento anterior con las *wavelet* hijas. El penúltimo valor del vector es la energía del segmento vocalizado, y el último valor del vector es la diferencia entre la energía del segmento vocalizado y la energía del segmento anterior.

Para comprobar el funcionamiento de este bloque del sistema se requiere integrar a la red neuronal, por lo que se decidió únicamente verificar que nuestro algoritmo de implementación fuera correcto comparando los resultados con las funciones de la biblioteca de Matlab. Para hacer esto se extrajeron segmentos que previamente habían sido clasificados como vocalizados de las etapas anteriores y se usaron como entrada para esta sección. En las figuras 8.8 y 8.9. para el segmento presente y en las figuras 8.10 y 8.11. para el segmento anterior, se presentan las correlaciones con la primer *wavelet* hija.

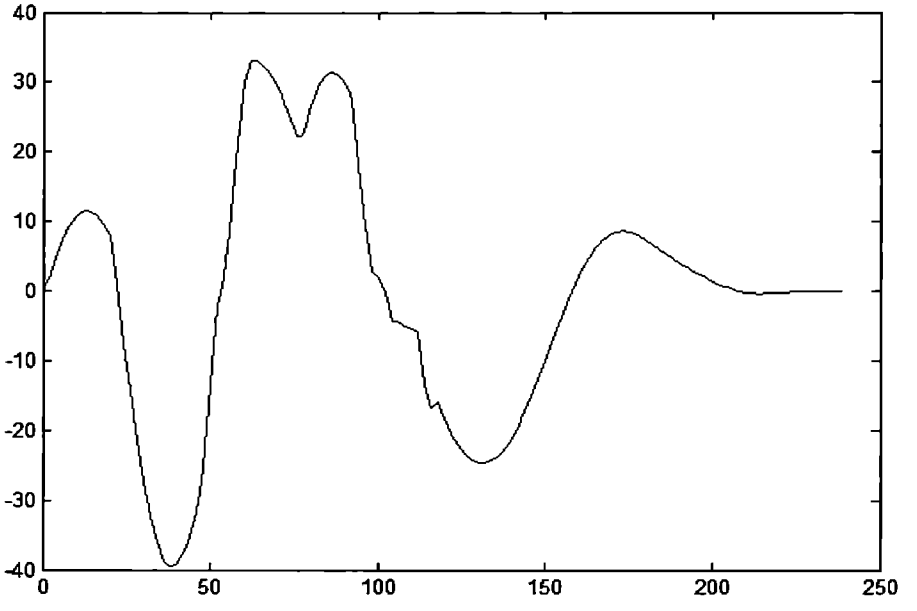


Figura 8.8 Correlación del segmento vocalizado con la primera wavelet hija (MatLab)

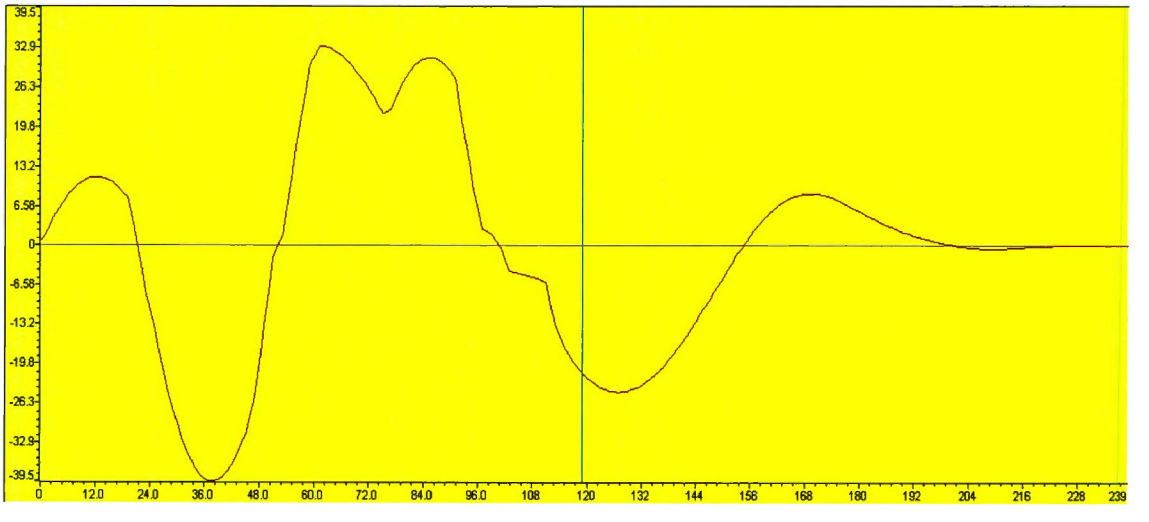


Figura 8.9 Correlación del segmento actual con la primera wavelet hija (DSP)

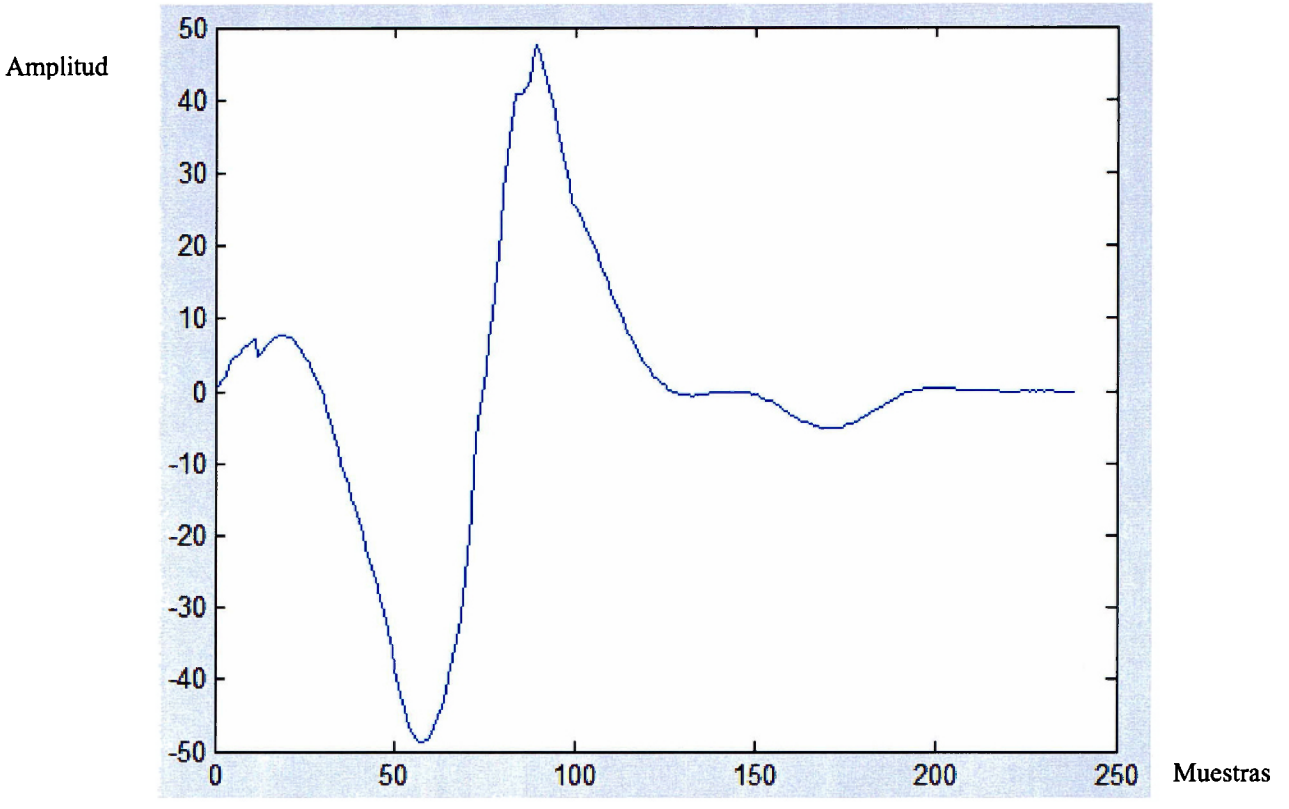


Fig.8.10 Correlación del segmento anterior con la primera wavelet hija (MatLab)

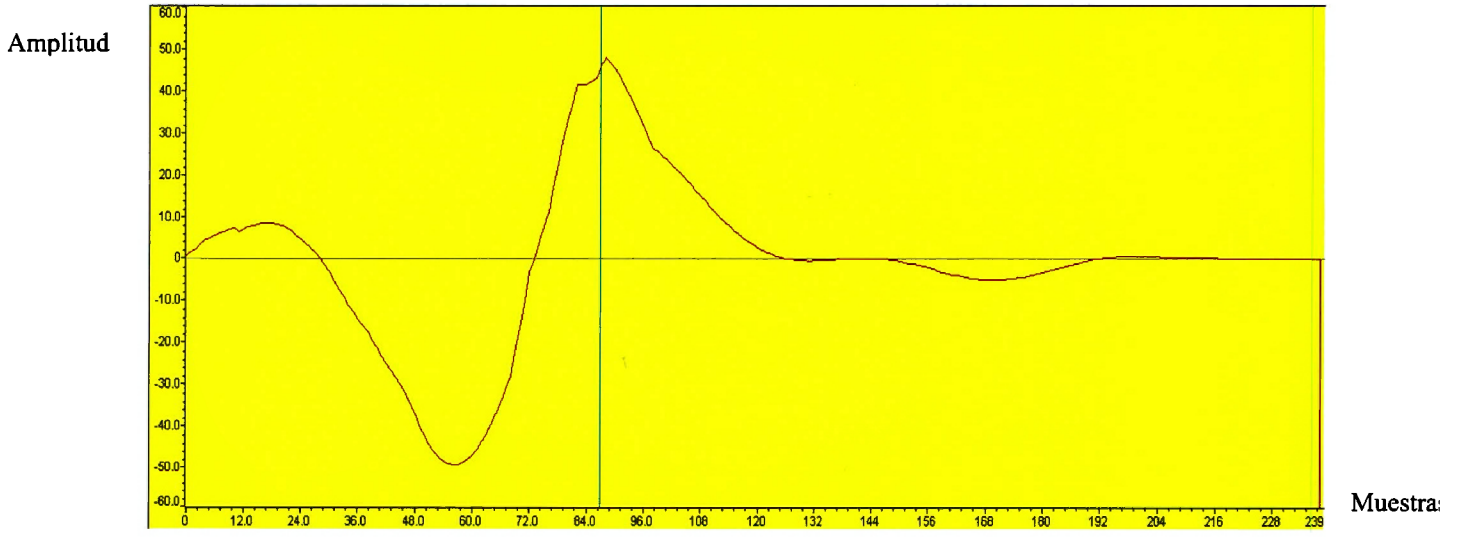


Fig.8.11 Correlación del segmento anterior con la primera wavelet hija (DSP)

Como se puede apreciar, nuestras gráficas son prácticamente iguales a las que arrojó MatLab, lo cual indica que nuestro algoritmo es correcto.

8.4 Reconocimiento

La red neuronal del sistema está conformada por 4 capas, una capa de entrada, dos capas ocultas y una capa de salida. La capa de entrada tiene un número de neuronas igual al tamaño del vector de parámetros (36), la primera capa oculta conformada por 48 neuronas, la segunda capa oculta por 16 neuronas, y la de salida por 5 neuronas.

Para validar el algoritmo de entrenamiento y la propagación hacia adelante se realizó, en Matlab, la implementación de una función lógica XOR, la cual es sugerida en los libros de textos como un problema clásico que debe resolver un Perceptrón Multicapa como el que se está implementando.

Para realizar la XOR con la red neuronal el tamaño de la entrada cambió momentáneamente de 36 a 2 nodos y el tamaño de la salida de 5 a 1 neurona, para hacer las combinaciones de la tabla de la verdad.

La red se entrenó y se hicieron las combinaciones de la tabla de la verdad de la función lógica XOR. Los resultados obtenidos se presentan en la figura 8.12.

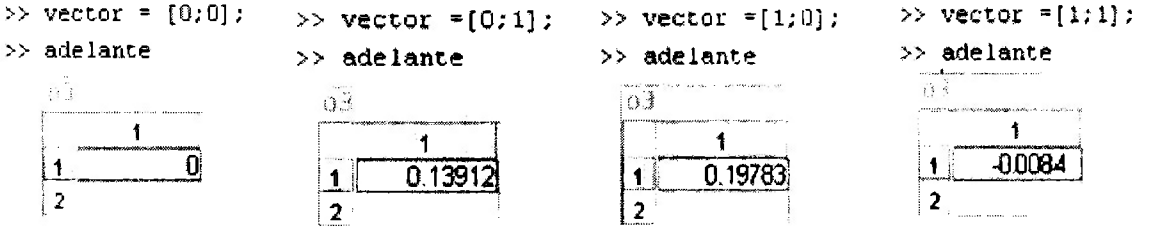


Figura 8.12 Prueba de la tabla de la verdad de la XOR

Con estos resultados se logra comprobar la tabla de la verdad, aunque no se alcanzan los valores máximos, pero con esto es suficiente para identificar cuándo se presentan entradas cuyas salidas deben ser uno ó cero.

Aunque los resultados del entrenamiento de la XOR son fiables, verificamos que el algoritmo de la red neuronal en su propagación hacia delante estuviera bien implementado. Esto se hizo porque la bibliografía sugiere que la programación de la red se realice de manera matricial, pero debido a que el DSP en sus matrices utiliza el formato Q15 y todos los datos que usamos en las etapas anteriores están en formato *float*, se implementaron algoritmos que hacen lo mismo que las operaciones matriciales. Para confirmar el correcto funcionamiento, ambas redes reciben el mismo vector de entrada. En la figura

8.13 se muestran los resultados de las salidas de cada una de las redes y se observa que se obtiene los mismos resultados.

redmatricial	redneuronal
salida =	salidac3
0.7498	0.7498
-0.0828	-0.0828
0.1596	0.1596
0.2709	0.2709
-0.8686	-0.8686

Figura 8.13 Comparación de red matricial y red con ciclos anidados

Una vez que se comprobaron el correcto funcionamiento de la propagación hacia adelante y la propagación hacia atrás de la red neuronal, es necesario implementar las etapas previas a la clasificación en Matlab para el entrenamiento de la red neuronal del sistema, ya que el vector de parámetros es generado por la etapa de extracción de parámetros.

En la figura 8.14 se muestra palabra "ojo" sin procesar por el sistema.

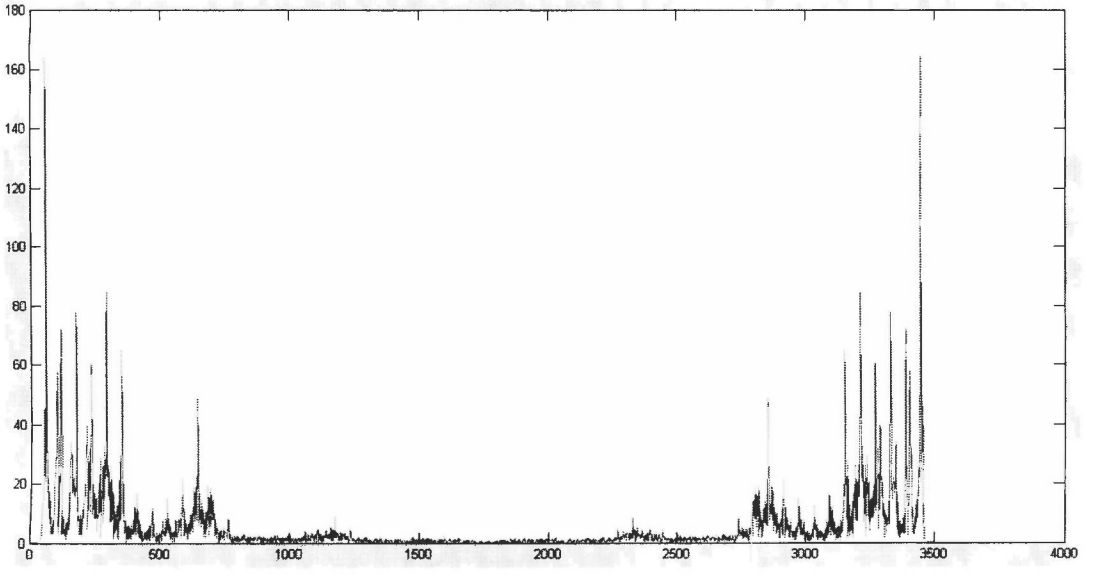


Figura 8.14 Señal Original. (Palabra Ojo)

Seguidamente la señal pasa por el filtro pasa-altas, obteniendo la figura 8.15 como resultado.

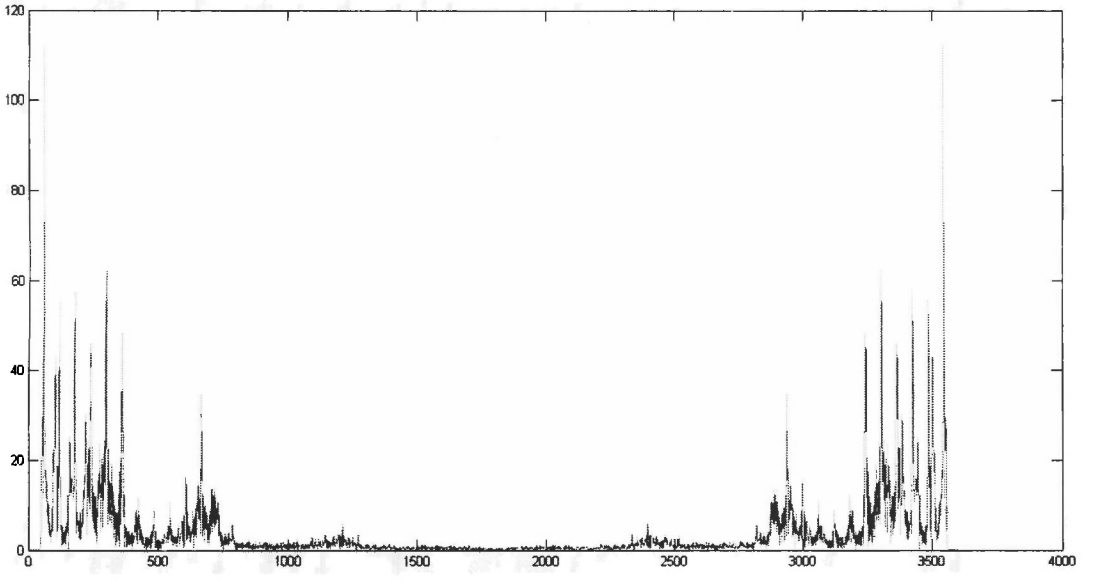


Figura 8.15 Señal filtrada con pasa-altas

Posteriormente se pasa la señal filtrada por el filtro de pre-énfasis obteniendo el resultado de la figura 8.16.

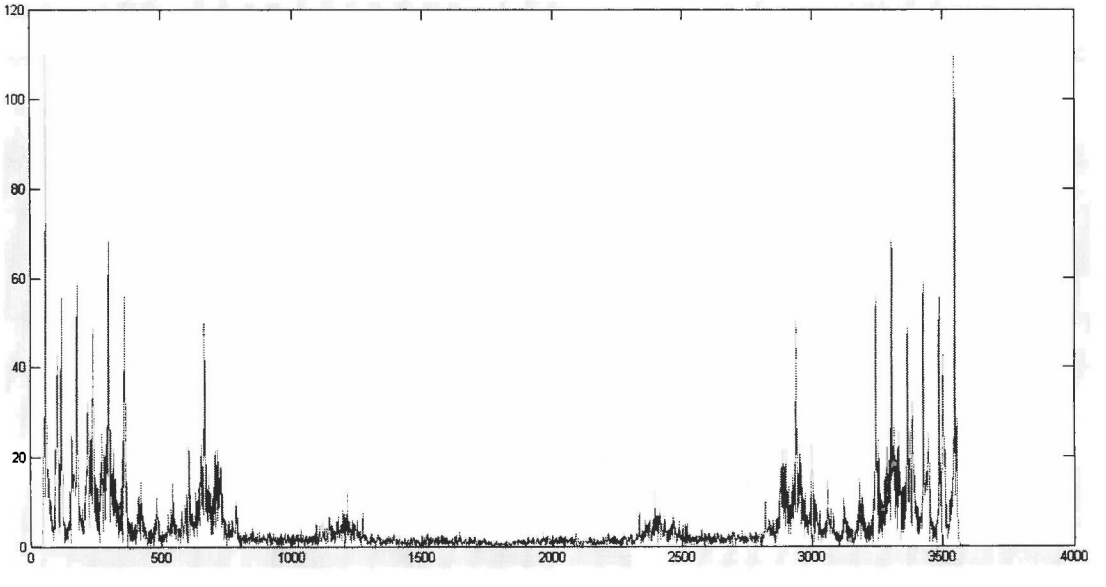


Figura 8.16 Señal filtrada con el filtro pre-énfasis

En esta simulación de MatLab, las etapas de segmentación y detección de segmentos vocalizados se realizaron juntas. En la figura 8.17 se muestra la

detección de segmentos vocalizados para la palabra "ojo" de MatLab comparados contra el resultado del DSP implementado.

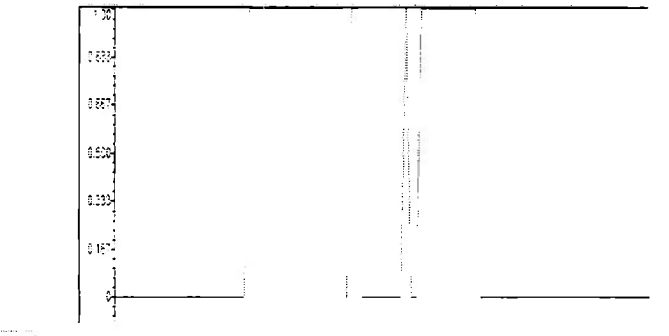


Figura 8.17 Detección de los segmentos vocalizados de la palabra "Ojo"

Una vez que se obtuvieron los segmentos vocalizados de una variedad de palabras, se agruparon las vocales en diferentes bancos, dando como resultado 5 bancos, uno por cada vocal. Posteriormente, de cada uno de estos bancos se toman 36 elementos, correspondientes a un vector de parámetros, y se usan como entrada a la red.

La clasificación se hace tomando la letra correspondiente a la neurona de salida que presenta el máximo de las cinco. En el siguiente ejemplo presentado en la figura 8.18, la entrada de la red esta dado por un vector de parámetros generado por la vocal "a" dando como resultado el máximo de las cinco, logrando identificar adecuadamente esta.

```
>> vector = ases(1:1:36,1);
>> adelante
```

	1
1	0.29298
2	0.27274
3	0.25559
4	0.28108
5	0.27586

Figura 8.18 Detección de la letra "a"

De la misma forma se introduce a la red un vector de parámetros propio de la vocal "e" y como se ve en la figura 8.19, y se aprecia que el segundo renglón es mucho mas grande que los otros 4, con lo cual se ve que se realiza la detección de manera adecuada.

```
>> vector = eses(1:1:36,1);
>> adelante
```

	1
1	0.16699
2	0.17883
3	0.13362
4	0.17325
5	0.16368

Figura 8.19 Detección de la letra "e"

Desafortunadamente con las vocales "i", "o" y "u", no se realiza la detección, como se muestra la figura 8.20, ya que otras vocales muestran salidas de mayor magnitud a la deseada:

```
>> vector = ises(1:1:36,1);
>> adelante
```

	1
1	0.1756
2	0.18117
3	0.11775
4	0.16913
5	0.14502

```
>> vector = oses(1:1:36,1);
>> adelante
```

	1
1	0.1341
2	0.13838
3	0.079599
4	0.13632
5	0.095265

```
>> vector = uses(1:1:36,1);
>> adelante
```

	1
1	0.11901
2	0.10636
3	0.091741
4	0.10052
5	0.10513

Figura 8.20 Errónea detección de las vocales "i", "o" y "u"

8.5 Formato Multitareas

Una de las herramientas que provee DSP-BIOS es la gráfica de ejecución, en la Figura 8.21 se muestra dicho diagrama de tiempos en el que se muestra cómo es la ejecución del programa. Primero se ejecuta la función Arranque, que en el diagrama se ve como "Other Threads", debido a que esta tarea es una subfunción perteneciente a la función Main, misma que se ejecuta inmediatamente iniciado el programa. Después se ejecuta la función Análisis que está activada como la TSK0, y es interrumpida después por Adquisición, activado como el PRD0, que comienza a entrar cada 125 µs.

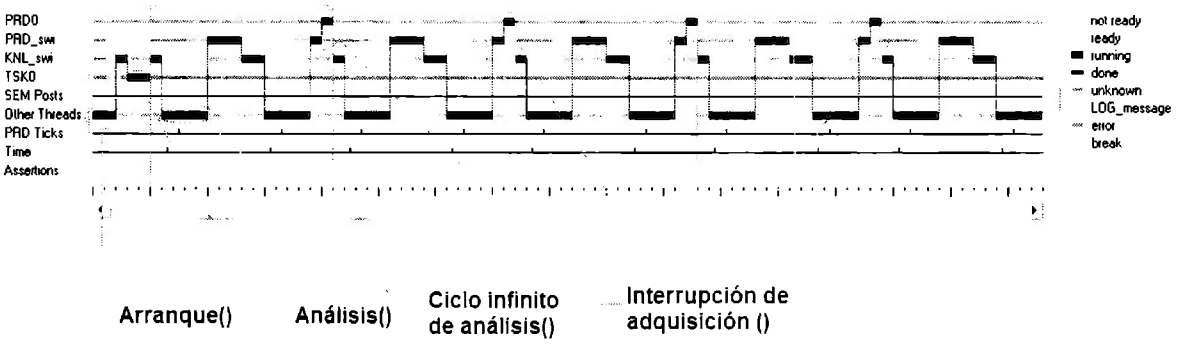


Figura 8.21 Diagrama de tiempos de la ejecución del programa.

Continuando con la adquisición de voz, se observa que al tener las 120 muestras se inicia el análisis de dicho segmento, como se muestra en la Figura 8.22. El diagrama de ejecución se encuentra justamente cuando se tienen 120 muestras. Cuando se inicia el análisis éste es interrumpido por la función Adquisición. Para comprender mejor el diagrama cabe mencionar que las pequeñas interrupciones realizadas por PRD_swi y KNL_swi son debido a que el DSP consulta en cada fracción de *tick* si hay alguna interrupción, pero como no hay tal, en ese momento regresa a la tarea que estaba realizando. Sin embargo se aprecia que cuando dicha consulta tiene una interrupción, se habilita a la función de PRD.

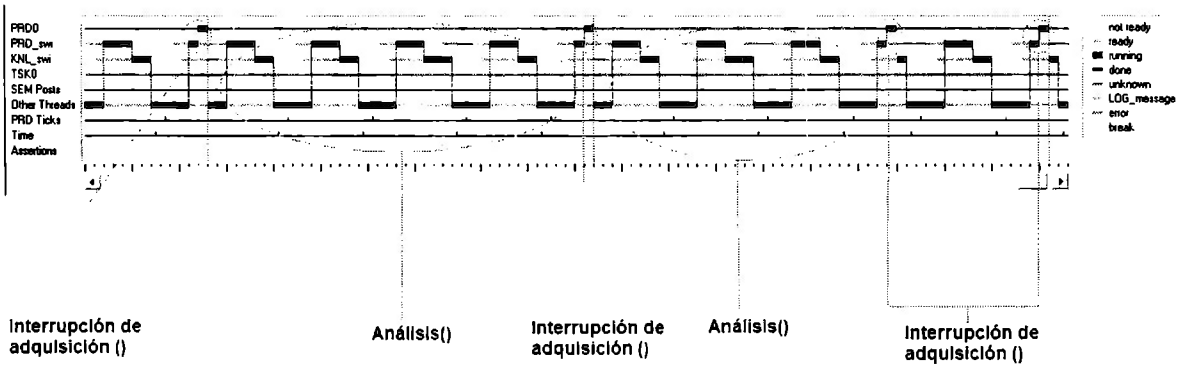


Figura 8.22 Diagrama de tiempos de la ejecución del programa.

Idealmente se debe de tener un tiempo de análisis menor que el tiempo de adquisición de un segmento (15 ms), de lo contrario se comienza a tener una pérdida de datos, lo cual evita el correcto funcionamiento del sistema. Para determinar este tiempo, consideramos ciertas pruebas, las cuales nos muestran algunas contradicciones, debido a que cada una de ellas arroja tiempos diferentes y además éstas no coinciden con nuestras mediciones hechas en tiempo real.

La primera prueba realizada consiste en declarar variables de prueba que son dependientes del reloj asignado a la tarea PRD (Adquisición), es decir, cuando se adquiere una muestra un contador (num_muestras) se incrementa, lo mismo sucede con otro contador (num_segmentos) cuando se tiene un segmento completo, y finalmente otro contador (finanalisis) que nos indica cuando se termina el análisis.

Mediante la aplicación *Watch Window* del DSP (Figura 8.11), podemos seguir los cambios que presentan estos contadores. En teoría el contador finanalisis debe ser siempre menor o igual al contador num_segmentos. En ese momento (Figura 8.23) se tienen 119 muestras; cuando se obtenga una más se comenzará con el análisis, y mientras el análisis se realiza, la adquisición también continua, hasta que el contador finanalisis se incrementa indicando el término del análisis. Podemos observar en la figura 8.24 que le toma aproximadamente 108 muestras de 125µs completar el análisis, es decir 13.5 ms. Es importante mencionar que este tiempo incluye todas las etapas (acondicionamiento, detección del segmento como vocalizado, extracción de parámetros y reconocimiento).

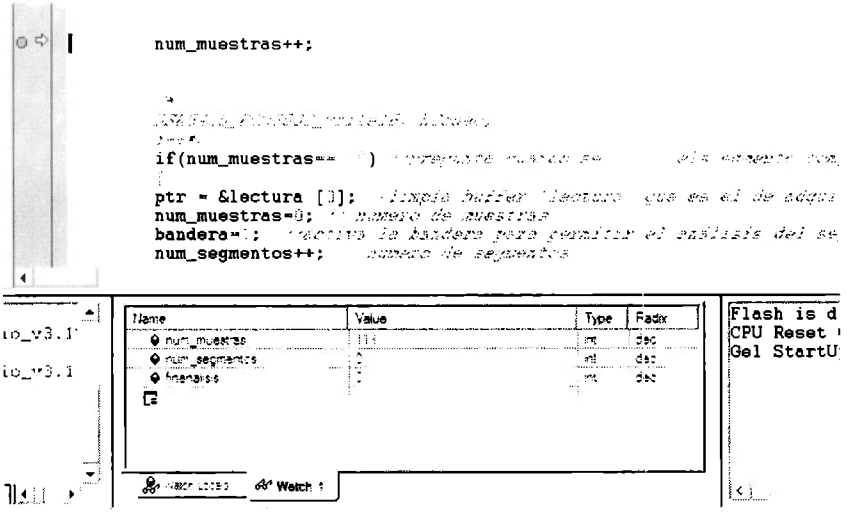


Figura 8.23 Momento en que inicia Análisis

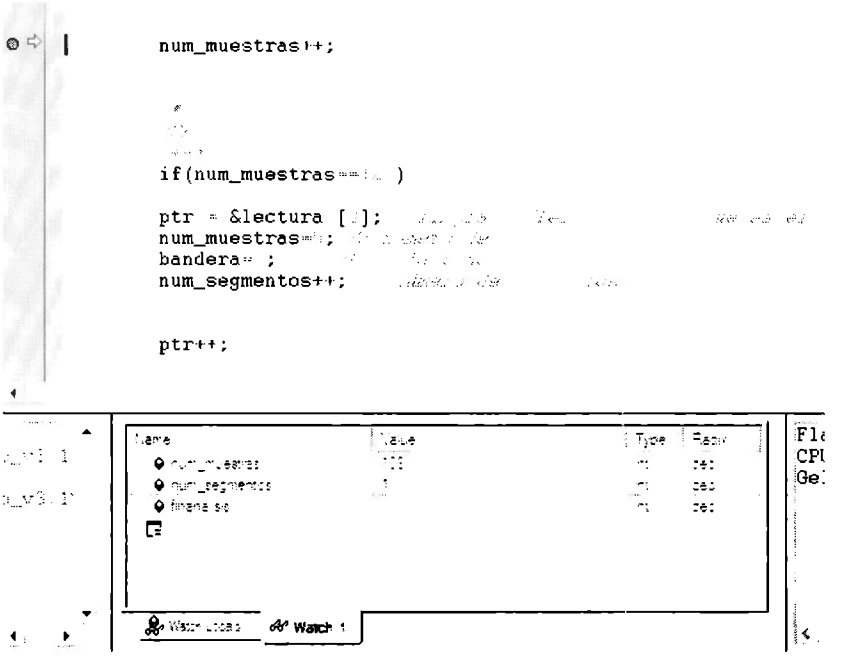


Figura 8.24 Momento en que termina Análisis

Sin embargo, recordemos que Code Composer Studio es un emulador, por lo que los tiempos calculados no corresponden a tiempo real.

9. Conclusiones y trabajo futuro

Se lograron implementar en el DSP las primeras tres etapas (acondicionamiento, detección de segmentos vocalizados y extracción de parámetros) de forma correcta así como la conjunción de éstas en un formato multitareas.

Completar la cuarta etapa (reconocimiento) queda como trabajo futuro, se trabajó arduamente en ella pero no se consiguió entrenar a la red neuronal por completo. Esto debido a la poca información y escasos ejemplos claros de implementación disponibles tanto en la literatura como en proyectos anteriores. Siendo un tema totalmente nuevo para nosotros fue complicado entenderlo e implementarlo de forma correcta, sin embargo el algoritmo de la red sí se realizó, faltando así solo terminar el entrenamiento de la misma para el reconocimiento de los segmentos vocalizados. La obtención de patrones de entrenamiento adecuados representó el punto crítico del entrenamiento de la red neuronal.

En cuanto al formato multitareas, se deja un esquema funcionando correctamente sobre los tiempos del DSP, mismos que no corresponden con el tiempo real. El trabajo futuro aquí es encontrar una herramienta de medición de tiempo que corrobore los tiempos calculados con nuestras variables.

Adicionalmente nos percatamos que algunas de las herramientas incluidas en DSP-Bios no detectan errores en la configuración del DSP, lo que hace complicado encontrar fallas que afectan el funcionamiento del sistema.

El manejo de números en formato Q15 disminuye la memoria utilizada, pero complica el manejo del programa, por la cual se optó por utilizar el formato que punto flotante, ya que la memoria no representa una limitante cuando el sistema funciona en tiempo real.

10. Anexo A

En este capítulo está el código final en el cual juntamos todos los algoritmos realizados hasta el momento (adquisición, segmentación, filtrados, normalización, detección de segmentos vocalizados, extracción de parámetros y parte del reconocimiento), así como las tablas de las *wavelet* hijas que utilizamos.

```
/*

Programa en formato multitarea, con tres funciones:


Arranque(): Adquiere las primeros 13 segmentos para realizar el primer análisis, es llamado en main

Analisis():Esta declarado como función Task, esta ciclado de forma infinita preguntando por la bandera
        que se activa en adquisicion() cuando se tienen 120 muestras, una vez que se activa dicha
                bandera se hace el análisis de detección de segmentos vocalizados y wavelet.

Adquisición():Declarado como una función PRD que toma la muestra cada 125us, esta interrumpe a la función
        Analisis() aún cuando se esta realizando el análisis completo.


*/

#include <stdio.h>

#include <tms320.h>




#include "volume.h"

#include "dsplib.h"

#include "tonecfg.h"

#include "dsk5416.h"

#include "math.h"

#include "dsk5416_pcm3002.h"
```

```
#define frame_length      1

#define marco_length      2873

#define NH 102

#define NX 120


int guarda[130];

float hamming[441] = {0.0800, 0.0802, 0.0804, 0.0807, 0.0812, 0.0817, 0.0823, 0.0830, 0.0838, 0.0846,
0.0856, 0.0867, 0.0878, 0.0891, 0.0904, 0.0918, 0.0934, 0.0950, 0.0967, 0.0985, 0.1003, 0.1023, 0.1044,
0.1065, 0.1087, 0.1111, 0.1135, 0.1160, 0.1185, 0.1212, 0.1239, 0.1268, 0.1297, 0.1327, 0.1358, 0.1389,
0.1422, 0.1455, 0.1489, 0.1524, 0.1559, 0.1596, 0.1633, 0.1671, 0.1710, 0.1749, 0.1789, 0.1830, 0.1872,
0.1914, 0.1957, 0.2001, 0.2045, 0.2090, 0.2136, 0.2182, 0.2229, 0.2277, 0.2325, 0.2374, 0.2424, 0.2474,
0.2525, 0.2576, 0.2628, 0.2680, 0.2733, 0.2787, 0.2841, 0.2896, 0.2951, 0.3006, 0.3062, 0.3119, 0.3176,
0.3233, 0.3291, 0.3350, 0.3408, 0.3467, 0.3527, 0.3587, 0.3647, 0.3708, 0.3769, 0.3830, 0.3892, 0.3954,
0.4016, 0.4078, 0.4141, 0.4204, 0.4267, 0.4331, 0.4395, 0.4459, 0.4523, 0.4587, 0.4651, 0.4716, 0.4781,
0.4846, 0.4910, 0.4976, 0.5041, 0.5106, 0.5171, 0.5237, 0.5302, 0.5367, 0.5433, 0.5498, 0.5563, 0.5629,
0.5694, 0.5759, 0.5824, 0.5890, 0.5954, 0.6019, 0.6084, 0.6149, 0.6213, 0.6277, 0.6341, 0.6405, 0.6469,
0.6533, 0.6596, 0.6659, 0.6722, 0.6784, 0.6846, 0.6908, 0.6970, 0.7031, 0.7092, 0.7153, 0.7213, 0.7273,
0.7333, 0.7392, 0.7450, 0.7509, 0.7567, 0.7624, 0.7681, 0.7738, 0.7794, 0.7849, 0.7904, 0.7959, 0.8013,
0.8067, 0.8120, 0.8172, 0.8224, 0.8275, 0.8326, 0.8376, 0.8426, 0.8475, 0.8523, 0.8571, 0.8618, 0.8664,
0.8710, 0.8755, 0.8799, 0.8843, 0.8886, 0.8928, 0.8970, 0.9011, 0.9051, 0.9090, 0.9129, 0.9167, 0.9204,
0.9241, 0.9276, 0.9311, 0.9345, 0.9378, 0.9411, 0.9442, 0.9473, 0.9503, 0.9532, 0.9561, 0.9588, 0.9615,
0.9640, 0.9665, 0.9689, 0.9713, 0.9735, 0.9756, 0.9777, 0.9797, 0.9815, 0.9833, 0.9850, 0.9866, 0.9882,
0.9896, 0.9909, 0.9922, 0.9933, 0.9944, 0.9954, 0.9962, 0.9970, 0.9977, 0.9983, 0.9988, 0.9993, 0.9996,
0.9998, 1.0000, 1.0000, 1.0000, 0.9998, 0.9996, 0.9993, 0.9988, 0.9983, 0.9977, 0.9970, 0.9962, 0.9954,
0.9944, 0.9933, 0.9922, 0.9909, 0.9896, 0.9882, 0.9866, 0.9850, 0.9833, 0.9815, 0.9797, 0.9777, 0.9756,
0.9735, 0.9713, 0.9689, 0.9665, 0.9640, 0.9615, 0.9588, 0.9561, 0.9532, 0.9503, 0.9473, 0.9442, 0.9411,
0.9378, 0.9345, 0.9311, 0.9276, 0.9241, 0.9204, 0.9167, 0.9129, 0.9090, 0.9051, 0.9011, 0.8970, 0.8928,
0.8886, 0.8843, 0.8799, 0.8755, 0.8710, 0.8664, 0.8618, 0.8571, 0.8523, 0.8475, 0.8426, 0.8376, 0.8326,
0.8275, 0.8224, 0.8172, 0.8120, 0.8067, 0.8013, 0.7959, 0.7904, 0.7849, 0.7794, 0.7738, 0.7681, 0.7624,
0.7567, 0.7509, 0.7450, 0.7392, 0.7333, 0.7273, 0.7213, 0.7153, 0.7092, 0.7031, 0.6970, 0.6908, 0.6846,
0.6784, 0.6722, 0.6659, 0.6596, 0.6533, 0.6469, 0.6405, 0.6341, 0.6277, 0.6213, 0.6149, 0.6084, 0.6019,
0.5954, 0.5890, 0.5824, 0.5759, 0.5694, 0.5629, 0.5563, 0.5498, 0.5433, 0.5367, 0.5302, 0.5237, 0.5171,
0.5106, 0.5041, 0.4976, 0.4910, 0.4846, 0.4781, 0.4716, 0.4651, 0.4587, 0.4523, 0.4459, 0.4395, 0.4331,
0.4267, 0.4204, 0.4141, 0.4078, 0.4016, 0.3954, 0.3892, 0.3830, 0.3769, 0.3708, 0.3647, 0.3587, 0.3527,
0.3467, 0.3408, 0.3350, 0.3291, 0.3233, 0.3176, 0.3119, 0.3062, 0.3006, 0.2951, 0.2896, 0.2841, 0.2787,
0.2733, 0.2680, 0.2628, 0.2576, 0.2525, 0.2474, 0.2424, 0.2374, 0.2325, 0.2277, 0.2229, 0.2182, 0.2136,
0.2090, 0.2045, 0.2001, 0.1957, 0.1914, 0.1872, 0.1830, 0.1789, 0.1749, 0.1710, 0.1671, 0.1633, 0.1596,
0.1559, 0.1524, 0.1489, 0.1455, 0.1422, 0.1389, 0.1358, 0.1327, 0.1297, 0.1268, 0.1239, 0.1212, 0.1185,
0.1160, 0.1135, 0.1111, 0.1087, 0.1065, 0.1044, 0.1023, 0.1003, 0.0985, 0.0967, 0.0950, 0.0934, 0.0918,
0.0904, 0.0891, 0.0878, 0.0867, 0.0856, 0.0846, 0.0838, 0.0830, 0.0823, 0.0817, 0.0812, 0.0807, 0.0804,
0.0802, 0.0800};

float   hpfc[NH]={0.0004,0.0002,0.0002,0.0002,0.0002,0.0002,0.0001,   0.0001,0.0001,0,0,-0.0001,-0.0002,-
0.0003, -0.0005, -0.0007,-0.0009,-0.0011,-0.0014,-0.0017,-0.0020, -0.0024,-0.0028,-0.0032,-0.0036,-0.0041,-
```

```
0.0046,-0.0052, -0.0057,-0.0063,-0.0069,-0.0075,-0.0081,-0.0087,-0.0093, -0.0099,-0.0105,-0.0111,-0.0116,-
0.0122,-0.0127,-0.0131, -0.0135,-0.0139,-0.0142,-0.0145,-0.0147,-0.0149,-0.0150, -0.0150,0.9846,-0.4148,-
0.0148,-0.0146,-0.0144,-0.0141, -0.0138,-0.0134,-0.0130,-0.0125,-0.0120,-0.0115,-0.0109, -0.0103,-0.0097,-
0.0091,-0.0085,-0.0079,-0.0073,-0.0067, -0.0061,-0.0055,-0.0050,-0.0045,-0.0040,-0.0035,-0.0030,-0.0026,-
0.0022,-0.0019,-0.0016,-0.0013,-0.0010,-0.0008, -0.0006,-0.0004,-0.0003,-0.0002,-0.0001,0,0,
0.0001,0.0001,0.0001,0.0002,0.0002,0.0002,0.0002, 0.0002,0.0002,0.0002,-0.0001};

float guarda1[100] = {0};

float filtra[NH+NX-1];

float guarda2[120] = {0};

int archivaldo=0;

int num_muestras=0;

int finanalisis=0;

int inf=0;

int bandera=0;

int num_segmentos=0;

//float energy_buffer[150]={0};

//int contaenergy=0;

//float total_energy=0;

//float energia=0;

int j=0;

int i=0;

int y=0;

int z=0;

int m=0;

int numseg=0;

int r=0;

int g=0;

int ka=0;

int buffer[NX]={0};

int ncmax=220;

//int nm=220;
```

```
float h=0;

int nf=0;

int nl=0;

float mayor;

float kk;

float x1 = 0;

float h1 = 0;

float buffer_hamming[441] = {0};

int length = NH + NX - 1;

float buffer13[2873] = {0};

int buffersinfiltrado[1560]={0};

//int buffer_voiced[100]={0};

float ev = 0;

float em = 0;

float relacion = 0;

int lim_inferior = 0;

int lim_superior = 0;

Int16 lectura[120]={0};

int guardaperro[120]={0};

Int16 *ptr = &lectura [0];

Int16 *ptr1 = &lectura [0];

//variables wavelet

float mayor = 0;

float segmento[120] = {0};

float                                    guarda3[120]                                    =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0};

float anterior[120] = {0};

float                                    guarda4[120]                                    =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0};

int nmax = 0;

float x = 0;

int n = 0;


//Wavelet

float corseg[240]= {0};

float corant[240]= {0};


float vector[36] = {0};


int pos = 0;

int s = 0;


float energia;

float energiaanterior;

float enercorseg;

float enercorant;

float inter = 0;

float inter2 = 0;


int hw = 0;

int v = 0;


int enemaseme = 0;


DSK5416_PCM3002_Config setup = {

   0x01FF, // Set-Up Reg 0 - Left  channel DAC attenuation

   0x01FF, // Set-Up Reg 1 - Right channel DAC attenuation
```

```
  0x0000, // Set-Up Reg 2 - Various ctl e.g. power-down modes
  0x0000  // Set-Up Reg 3 - Codec data format control
};


void filtrar()
{

    for ( r = 0; r < length; r++) filtra[r] = 0;   /*limpia filtra*/

    for ( g = 0; g < length; g++)
        {
      for (ka = 0; ka <= g ; ka++)
                    {

        if( ka > NX )
                        {
                            x1 = 0;
                        }

                    if ( (g-ka) > NH)
                    {
                        h1 = 0;
                    }

                        else
                        {
                        h1 = hpfc[g-ka];
                        x1 = lectura[ka];
```

```
            }

                        filtra[g] = filtra[g] + (h1 * x1);

                        }
            }

}

void normalizar()
{
    for( r = 0; r < NX+NH-1 ; r++ )
            {

      if( filtra[r] < 0 )
      {
       kk = filtra[r]* -1;
      }
      else
                  {
                  kk=filtra[r];
                  }

      if(kk>mayor)
                  {
                    mayor=kk;
                  }
```

```
    }

    for( r = 0; r < 221; r++ )
    {
     filtra[r] = filtra[r] / mayor;
    }

    mayor=0;
}

void redneuronal()
{

//Suma para la primera capa, con i=numero de neurona y j=nodo de entrada
conta = 1;
for( i = 1;i<=neuronascapa1;i++);
  {
        for (j = 1;i<=tamanoentrada;i++);
                        {
        salidacapa1[i] = salidacapa1[i]+(W1s[conta]*vector[j]);
        conta ++;
        }

            salidacapa1[i] = salidacapa1[i] + B1[i];
            salidacapa1[i] = tanh(salidacapa1[i]);
  }
```

```
conta = 1;
for (i = 1;i<=neuronascapa2;i++)
        {
        for (j = 1;i<=neuronascapa1;i++)
                {
        salidacapa2[i] = salidacapa2[i] + (W2s[conta]*salidacapa1[j]);
        conta++;
        }

                salidacapa2[i] = salidacapa2[i] + B2[i];
                salidacapa2[i] = tanh(salidacapa2[i]);


        }



conta = 1;
for (i = 1;i<=neuronascapasalida;i++)
  {
           for (j = 1;i<=neuronascapa2;i++)
                {
        salida[i] = salida[i] + (W3s[conta]*salidacapa2[j]);
        conta++;
        }
```

```
                salida[i] = salida[i] + B3[i];
                salida[i] = tanh(salida[i]);


        }


}


void waveletfuncion()
{
  //Pasa a segmento y anterior  lo del buffer sin filtrado

        n = 0;
                        for ( r = 720  ; r < 840 ; r++)
                        {
                                segmento[n] = buffersinfiltrado[r];
                                n++;
                        }

        n = 0;
                        for ( r = 600 ; r < 720 ; r++)
                        {
                                anterior[n] = buffersinfiltrado[r];
```

```
                    n++;
                }

//normalizacion segmento presente

    mayor = 0;

    for ( i = 0; i < 120 ; i ++)
        {

            if(segmento[i] < 0)
            {
                inter = segmento[i] * -1;

                if ( inter > mayor )
                {
                    mayor = inter ;
                }
            }

            if (segmento[i] > 0)
            {
                inter = segmento[i];

                if ( inter > mayor )
                {
                    mayor = inter ;
                }
            }
```

```
        }


        for ( i = 0 ; i < 120 ; i++)
                {
                        segmento[i] = segmento[i] / mayor;
                }

    //normalización segmento anterior
mayor=0;
        for ( i = 0; i < 120 ; i ++)
                {

                if(anterior[i] < 0)
                {
                        inter = anterior[i] * -1;

                        if ( inter > mayor )
                        {
                                mayor = inter ;
                        }
                }

                if (anterior[i] > 0)
                {
                        inter = anterior[i];
```

```
                        if ( inter > mayor )
                        {
                                mayor = inter ;
                        }
                }


        }

for ( i = 0 ; i < 120 ; i++)
        {
                anterior[i] = anterior[i] / mayor;
        }

for ( i = 0; i < 36; i++ )
        {
                vector[i] = 0;
        }


        //inicia wavelet
pos=0;
for ( j = 0 ; j < 17 ; j+−)
        {

                s = 0;
                while ( s < 120)
                {
                        wavelet[s] = WH[pos];
```

```
            s++;
            pos++;
        }

        //correlaciones con WH
        for ( i = 0; i < 239; i++ )
        {
            corseg[i] = 0;
        }

        for ( i = -119 ; i <= 120 ; i++)
        {
            if (i < 0)
            {
                v = 120 + i;
                g = v;

                for ( hw = 0; hw < g ; hw++)
                    {
                        corseg[i+119] = corseg[i+119] + (segmento[hw] * wavelet[120-
v]);
                        v--;
                    }
            }

            if ( i >= 0)
            {
                v = 120-i;
                g = v;
```

```
                    for ( hw = 0; hw < g ; hw++)
                    {
                        corseg[i+119] = corseg[i+119] + (segmento[120-v] *
wavelet[hw]);
                        v--;
                    }
                }


            }


            for ( i = 0; i < 239; i++ )
            {
                corant[i] = 0;
            }

            for ( i = -119 ; i <= 120 ; i++)
            {
                if (i < 0)
                {
                    v = 120 + i;
                    g = v;

                    for ( hw = 0; hw < g ; hw++)
                    {
                        corant[i+119] = corant[i+119] + (anterior[hw] * wavelet[120-v]);
                        v--;
                    }
                }
```

```
        if ( i >= 0)
        {
                v = 120-i;
                g = v;

                for ( hw = 0; hw < g ; hw++)
                {
                        corant[i+119] = corant[i+119] + (anterior[120-v] * wavelet[hw]);
                        v--;
                }
        }


}

enercorseg = 0;
for ( i = 0 ; i < 239 ; i++)
{
        enercorseg = enercorseg + (corseg[i] * corseg[i]);
}

vector[j] = enercorseg;

enercorant = 0;
for ( i = 0 ; i < 239 ; i++)
{
        enercorant = enercorant + (corant[i] * corant[i]);
}
```

```
                vector[j + 17] = enercorseg - enercorant;
        }

        energia = 0;
        for ( i = 0; i < 120 ; i++)
        {
                energia = (segmento[i] * segmento[i]) + energia;
        }

        energiaanterior = 0;
        for ( i = 0 ; i < 120 ; i++)
        {
                energiaanterior = energiaanterior + (anterior[i] * anterior[i]);
        }

        vector[34] = energia;
        vector[35] = energia-energiaanterior;
        n=n;
  //normalización de vector


mayor = 0;
    for( r = 0; r < 36 ; r++ )
          {


      if( vector[r] < 0 )
      {
```

```
        kk = vector[r]* -1;
      }
      else
                  {
                  kk=vector[r];
                  }

      if(kk>mayor)
                  {
                    mayor=kk;
                  }


    }

    for( r = 0; r < 36; r++ )
    {
     vector[r] = vector[r] / mayor;
    }

     mayor=0;



}
```

```
void arranque()          //Adquiere 13 segmentos de 120 muestras, se filtran y normalizan para realizar el primer análisis
{

  for ( z = 0 ; z < 13 ; z++)
  {

   DSK5416_PCM3002_CodecHandle hCodec;
   int j;
   /* Start the codec */
   hCodec = DSK5416_PCM3002_openCodec( 0, &setup );


                DSK5416_PCM3002_setFreq(hCodec, 8000);


     for ( j = 0 ; j < 120 ; j++ )
     {
       while ( !DSK5416_PCM3002_read16( hCodec, ptr ) );
       ptr++;
     }


                for ( r = (z*NX) ; r < ((z*NX)+NX) ; r++) //Guarda 1 segmento SIN filtrado en buffer13 no filtrado
                {
                        buffersinfiltrado[r] = lectura[n];
                        n++;
                }
```

```
                    filtrar();   //fILTRA LECTURA SEGMENTO DE 120 MUESTRAS
                    normalizar();

                    n = 0;
                    for ( r = (z*length) ; r < ((z*length)+length) ; r++) //Guarda 1 segmento filtrado en
buffer13
                    {
                            buffer13[r] = filtra[n];
                            n++;
                    }
        n=0;
                    ptr = &lectura [0];

          }

    archivaldo=0;

 } //Termina arranque


void deteccion()
{
  /*
  Realiza el acondicionamiento con un suavizado de la ventana, que consta del segmento a determinar
  si es vocalizado, más la mitad del segmento anterior y la mitad del posterior.
  Después se obtiene la energía promedio de la ventana y del marco, para posteriormente comparlos con los
  criterios de energía y determinar si es vocalizado.
  */
```

```
//pasa buffer a la ventana  a buffer_hamming

        n = 0;
        for ( r = 1216 ; r < 1656 ; r++)
        {
                buffer_hamming[n] = buffer13[r] ;
                n++;
        }

        ev = 0;


//Hace el suavizado con ventana de hamming y calcula energia

        for ( r = 0 ; r < 441 ; r++)
        {
                buffer_hamming[r] = buffer_hamming[r] * hamming[r];
                ev = ev + (buffer_hamming[r] * buffer_hamming[r]);
        }

        ev = ev / 441;

//calcula energia marco y primer criterio
        em = 0;
        for ( r = 0 ; r < 2873 ; r++ )
        {
                em = em + (buffer13[r] * buffer13[r]);
        }
```

```
em = em / 2873;


if ( em < 0.33 )
{
        em = 0.33;
}


if ( em > 0.5 )
{
        em = 0.5;
}


relacion = ev / em;


 waveletfuncion();
redneuronal();
if ( relacion > 0.125 ) // Si la energía es menor, significa que es vocalizado, por lo tanto se aplica wavelet y red neuronal para determinar que vocal es
{
// waveletfuncion();
 //redneuronal();


}


else
{



}
```

```
}

void analisis()
 {

    for ( inf = 0 ; inf < 10 ; inf++) //ciclado infinito en Task
          {
     inf=0;  //se limpia la variable "inf" para que no se cumpla la condición anterior del "for" para un ciclado infinito
     if(bandera==1) // se pregunta por la bandera que se encuentra en adquisicion la cual se activa al tener 120 muestras (1 segmento)
     {
   //Se guarda el nuevo segmento sin filtrado para el prceso de wavelet y red neuronal en caso de ser vocalizado
      n=0;
      for ( r = 1440 ; r < 1560 ; r++)
                              {
                                      buffersinfiltrado[r] = lectura[n];
                                      n++;
                              }


        filtrar();    //fILTRA LECTURA que tiene el nuevo SEGMENTO DE 120 MUESTRAS
                              normalizar(); //Normaliza Lectura que tiene el nuevo segmento de 120 muestras
```

```
            n=0;
for ( r = 2652 ; r < 2873 ; r++)
                {
                    buffer13[r] = filtra[n];
                    n++;
                }

                deteccion();

            //SHIFT DE BUFFERS, ajusta la vecindad para el  próximo segmento que se va adquirir
                for ( r = 120 ; r < 1560 ; r++)
                {
                    buffersinfiltrado[r-120] = buffersinfiltrado[r];
                }

                for ( r = 221 ; r < 2873 ; r++)
                {
                    buffer13[r-221] = buffer13[r];
                }

    bandera=0;
    finanalisis++;  // "finanalisis" es un contador de prueba que se incrementa cuando se termina todo el
analisis, incluyendo wavelet y redneuronal
    r=0;
   }

}
```

```
}

void adquisicion()
{

  DSK5416_PCM3002_CodecHandle hCodec;

  /* Start the codec */

  hCodec = DSK5416_PCM3002_openCodec( 0, &setup );

     !DSK5416_PCM3002_read16( hCodec, ptr ) ; //Adquisicion de la muestra

                num_muestras++;

     /*
```

```
                DSK5416_PCM3002_write16( hCodec, lectura[j]);
                j++*/
                if(num_muestras==120) //pregunta cuando se tiene els egmento completo, es decir 120
muestras
                {
                ptr = &lectura [0]; //limpia buffer "lectura" que es el de adquisición
                num_muestras=0; // número de muestras
                bandera=1;  //activa la bandera para permitir el análisis del segmento
                num_segmentos++; // número de segmentos
                }

                ptr++;
}



void main()
{

  DSK5416_init();



   for (  r= 0 ; r < 120 ; r++) guardaperro[r]=0;


        arranque();  //Llama a dicha función para adquirir los primeros 13 segmentos necesarios para el
primer análisis
```

***Wavelets* hijas**

muestra	T	WH0	WH1	WH2	WH3	WH4	WH5
0	0	0	0	0	0	0	0
1	0.000125	0.000645352	0.00366988	0.010266807	0.02156467	0.0387337	0.06285662
2	0.00025	0.002531014	0.01386708	0.036582598	0.07035906	0.11066375	0.14560409
3	0.000375	0.005553111	0.02877515	0.068794519	0.11133722	0.12399743	0.05005015

4	0.0005	0.009572583	0.04592363	0.093807968	0.10404988	0.00836007	0.35288632
5	0.000625	0.014418652	0.06238806	0.097792819	0.01465094	0.32622664	1.01686704
6	0.00075	0.019892627	0.07500624	0.068384712	0.17397072	0.79708229	1.70623156
7	0.000875	0.025771983	0.08060007	0.003349718	0.45577632	-1.315542	2.07891611
8	0.001	0.031814673	0.07619253	0.121378448	0.79995913	1.72708925	-1.8297908
9	0.001125	0.037763607	0.05921015	0.283744545	1.15432877	1.86916696	0.83445616
10	0.00125	0.04335125	0.02766209	-0.48226549	1.45305533	1.61828082	0.76582456
11	0.001375	0.04830428	0.01971145	0.702911389	1.62743063	-0.9304034	2.57466415
12	0.0015	0.052348253	0.08332758	0.926819569	1.61784589	0.13621262	4.04784042
13	0.001625	0.055212228	-0.1626901	1.131846941	1.38501937	1.42231177	4.66365512
14	0.00175	0.056633282	0.25636239	1.294515919	0.91864391	2.69368705	4.10002513
15	0.001875	0.056360892	0.36198459	1.392177684	0.24202683	3.68683502	2.35872776
16	0.002	0.054161116	0.47633393	1.405200181	0.5881039	4.16446944	-0.2062172
17	0.002125	0.049820537	0.59542571	1.318988014	1.48674606	3.96881604	2.96867286
18	0.00225	0.04314993	0.71465039	1.125656772	2.35012921	3.06095827	5.19753924
19	0.002375	0.033987614	0.82894153	0.825213551	3.0683686	1.53737893	6.26235517
20	0.0025	0.022202456	0.93296764	-0.42613583	3.53961084	0.38043127	5.81682299
21	0.002625	0.007696492	1.02134066	0.054711037	3.68379902	2.38188047	3.90614317
22	0.00275	0.009592848	1.08883285	0.592825392	3.45422101	4.12207192	0.96638959
23	0.002875	0.029690945	1.13059393	1.157445705	2.84528092	5.28504283	2.28459671
24	0.003	0.052584632	1.14236011	1.713459667	1.89541053	5.64329589	5.03449889
25	0.003125	-0.07822129	1.12064715	2.223713172	0.68464877	5.10217657	6.59013148
26	0.00325	-0.1065084	1.06292017	2.651556006	0.67291612	3.72056426	6.5589725
27	0.003375	0.137313558	0.96773362	2.96344284	-2.0410099	1.70385655	4.95006942
28	0.0035	-0.17046495	-0.8348362	3.131403125	3.27597918	0.62956683	2.16864808
29	0.003625	0.205752277	0.66523633	3.13520186	4.24282062	2.90231315	1.09209799
30	0.00375	0.242928125	0.46122557	2.964034135	4.83056153	4.74260546	4.03132317
31	0.003875	0.281709758	0.22635845	2.617628074	4.96519686	5.84797799	5.94417064

32	0.004	-0.321781328	0.03461117	2.106671158	-4.6186735	6.036773	-6.39549969
33	0.004125	-0.362796454	0.3158356	1.452520711	-3.81287369	5.27852732	-5.32016963
34	0.00425	-0.404381181	0.61051784	0.686207677	-2.61812504	3.69822975	-3.02699916
35	0.004375	-0.44613725	0.91109087	-0.153209828	-1.14638928	1.55411473	-0.10916543
36	0.0045	-0.487645678	1.20942629	-1.020843379	0.46012033	-0.80672507	2.71247565
37	0.004625	-0.528470596	1.49706618	-1.868585221	2.04363697	-3.00689013	4.77090526
38	0.00475	-0.568163314	1.76547079	-2.647969843	3.44724112	-4.70114528	5.61286255
39	0.004875	-0.606266571	2.00627485	-3.313106165	4.53117966	-5.63218138	5.09854081
40	0.005	-0.642318932	2.21154421	-3.823495706	5.18734733	-5.67049304	3.41953841
41	0.005125	-0.675859294	2.3740252	-4.14655933	5.35039548	-4.8321704	1.03507272
42	0.00525	-0.706431444	2.48737875	-4.259713652	-5.00431797	-3.27232293	-1.45328486
43	0.005375	-0.733588651	2.54639209	-4.151866614	-4.18386601	-1.25601098	-3.45412546
44	0.0055	-0.756898219	2.5471614	-3.824238104	2.97070553	0.88775876	-4.52714041
45	0.005625	-0.775945984	2.48723954	-3.290453508	1.48478996	2.8208183	-4.47894116
46	0.00575	-0.790340703	2.36574418	-2.575902844	-0.12808268	4.25011828	-3.39342508
47	0.005875	-0.799718287	2.18342268	1.716403045	-1.71117742	4.97268051	-1.59365474
48	0.006	-0.803745857	1.94267153	-0.756243022	-3.11261451	4.90430401	0.4509318
49	0.006125	-0.802125563	1.64750906	0.254272051	-4.20040779	4.08806496	2.24587417
50	0.00625	-0.79459815	1.30350218	1.261631445	-4.8752268	2.68212175	3.39045291
51	0.006375	-0.780946228	0.91764837	2.21211859	-5.0796314	0.92972205	3.66559648
52	0.0065	-0.760997214	0.49821637	3.054823841	-4.80292119	-0.88294611	3.07218456
53	0.006625	-0.734625922	0.05454934	3.744494992	-4.08121382	-2.47372908	1.8143271
54	0.00675	-0.70175678	-0.40316406	4.2440613	-2.99286289	-3.60856743	0.23580353
55	0.006875	-0.662365642	-0.8641446	4.526687464	-1.64979748	-4.13576505	-1.27158409
56	0.007	-0.616481183	-1.31729647	4.577242832	-0.18576136	-4.00552176	-2.36376659
57	0.007125	-0.564185863	-1.75149305	4.39310626	1.2572857	-3.27240548	-2.82123959
58	0.00725	-0.50561645	2.15586654	3.984266349	2.54319602	-2.0812902	-2.59031976
59	0.007375	-0.44096409	-2.52009367	3.372717803	3.55468447	-0.6398907	-1.7820017
60	0.0075	-	-	2.591194985	4.20413802	0.81702097	-

		0.370473921	2.83466979				0.63222469
61	0.007625	0.294444237	3.09116379	1.681321047	4.44118722	2.0656049	0.56381975
62	0.00775	0.213225196	3.28244699	0.691283164	4.25639921	2.92704409	1.52564579
63	0.007875	0.127217095	3.40288983	0.326830354	3.68086016	3.29261121	2.05158179
64	0.008	0.036868204	3.44852099	-1.31987824	2.78183074	3.13631964	2.05882469
65	0.008125	0.057327816	3.41714465	2.236538919	1.6550404	2.51393158	1.59216348
66	0.00825	0.154834882	3.30841271	3.030048038	0.41449695	1.54931921	0.80212345
67	0.008375	0.255077854	3.12384993	3.660651605	0.81910081	0.41110014	0.09959023
68	0.0085	0.357446238	-2.8668314	4.097643955	1.93016441	0.71619156	0.89485327
69	0.008625	0.461298218	2.54251268	4.320885215	2.81907379	-1.6615553	1.41034238
70	0.00875	0.565964993	2.15771458	4.321723876	3.41107428	-2.29297	1.55357309
71	0.008875	0.670755363	1.72076534	4.103284387	3.66242791	2.53514242	1.32675865
72	0.009	0.77496056	1.24130446	3.680115586	3.56333554	2.37731196	0.81734479
73	0.009125	0.87785926	0.73005309	3.077231093	3.13748136	1.87056878	0.16964031
74	0.00925	0.978722759	0.19855686	2.328605537	2.43838671	1.11581239	0.45403967
75	0.009375	1.076820261	0.34109218	1.475218991	1.54306212	0.24485781	0.91266154
76	0.0095	1.171424245	0.87654753	0.562764678	0.54368694	0.60191279	1.11700954
77	0.009625	1.261815871	1.39550662	0.36084916	0.46179535	1.29765573	1.04522499
78	0.00975	1.347290372	1.88600827	1.248064141	1.37921349	1.74753078	0.74121117
79	0.009875	1.427162416	2.33672146	2.054163869	2.12691484	1.9009339	0.29796866
80	0.01	1.500771365	2.73721855	2.739561093	2.64291937	1.75605798	0.16877169
81	0.010125	1.56748642	3.07822592	3.271762007	2.88986986	1.35663338	-0.549531
82	0.01025	1.626711596	3.35184605	3.626910154	2.85740874	0.78192065	0.76569702
83	0.010375	1.677890493	3.55174558	3.790837574	2.5618768	0.13199151	0.78450974
84	0.0105	1.720510826	3.67330487	3.759579227	2.04349197	-0.489089	0.62192693
85	0.010625	1.754108687	3.71372557	3.539336746	1.36140018	0.98943736	-0.3339464
86	0.01075	1.778272489	3.67209375	3.145907537	0.58717504	1.30244681	0.00042032
87	0.010875	1.792646581	3.54939733	2.603623471	0.20253711	1.39502512	0.29972209
88	0.011	1.796934492	3.34849762	1.943868461	0.93348382	-1.2700805	0.49906548
89	0.011125	1.790901781	3.07405596	1.20326483	1.54066906	0.96331505	0.56330969

90	0.01125	1.774378478	2.73241759	0.421633502	1.97399027	0.53525572	0.49201936
91	0.011375	1.747261089	2.33145584	0.360157868	2.20217824	0.06011321	0.3158993
92	0.0115	1.709514143	1.88038062	1.102342059	2.21475421	0.38657803	0.08643455
93	0.011625	1.661171287	1.38951634	-1.76828165	2.02192306	0.73955143	-0.1381527
94	0.01175	1.602335897	0.87005466	2.326277104	1.65252192	0.95291881	0.30699772
95	0.011875	1.533181209	0.33378834	2.751064942	1.15032254	1.00560371	0.38748434
96	0.012	1.453949974	0.20716727	-3.0249365	0.56912593	0.90256021	0.37076659
97	0.012125	1.364953617	0.74065828	3.138429963	0.03282137	0.67192897	0.27122771
98	0.01225	1.266570927	1.25477499	3.090572439	0.59852445	0.35889191	0.12065745
99	0.012375	1.159246266	1.73812658	2.888673276	1.07750632	0.01743088	0.04093782
100	0.0125	1.04348732	2.18010121	2.547693507	1.43017	0.29858171	0.17540979
101	0.012625	0.919862397	2.57110556	2.089237808	1.63089534	0.54354698	0.2550125
102	0.01275	0.788997301	-2.9027782	1.540233727	1.66964963	0.68637462	0.26761852
103	0.012875	0.651571783	3.16817213	0.931377273	1.55204419	0.71399781	0.21777407
104	0.013	0.508315613	3.36190236	0.295433585	1.29792009	0.63185781	0.12380998
105	0.013125	0.360004284	3.48025556	0.334513947	0.93868039	0.46153619	0.01210979
106	0.01325	0.207454375	3.52125943	0.92677319	0.51369205	0.23613369	0.08983181
107	0.013375	0.051518623	3.48471088	1.452623829	0.06615001	0.00571224	0.15983054
108	0.0135	0.106919291	3.37216257	1.887691152	0.36117643	0.22614618	0.18549728
109	0.013625	0.266950186	3.18686883	2.21304877	0.72990831	0.39376046	0.16597337
110	0.01375	0.427645031	-2.9336927	2.416002365	1.00937721	0.48780138	0.11082716
111	0.013875	0.588060757	2.61897668	2.490524943	1.17902758	0.50040115	0.03668539
112	0.014	0.747246195	2.25038082	2.437333267	1.22973004	0.43666467	0.03737172
113	0.014125	0.904248098	1.83669225	2.263614261	1.16394795	0.31278222	0.09445475
114	0.01425	1.058117216	1.38761111	1.982428056	0.99481026	0.15262437	0.12347918
115	0.014375	1.207914367	0.91351814	1.611830262	0.74424384	0.01653263	0.12103769
116	0.0145	-1.35271648	0.42522959	1.173769047	0.44039738	0.16842688	0.09130046
117	0.014625	-1.49162256	0.06625456	0.692822306	0.11464139	0.28169263	0.04419196

118	0.01475	-1.62375954	0.55000324	0.194846065	0.20154871	0.34266189	0.00753992
119	0.014875	1.748287975	1.01540175	0.294392802	0.47952747	0.34675528	0.05156347

WH6	WH7	WH8	WH9	WH10	WH11	WH12
0	0	0	0	0	0	0
0.09452141	0.13294701	0.17432047	0.20888146	0.21621929	0.15850249	0.02745167
0.15091122	0.08345203	-0.1242792	0.56137404	1.31110895	2.37108639	3.49224166
-0.1933457	-0.699736	1.51717519	2.52191001	3.22210333	2.62108611	0.4457317
1.01443059	1.93518502	2.71967274	-2.4727076	0.05026123	4.64543097	8.37720089
1.93449565	2.51234648	1.66938656	1.55247404	6.10609837	7.01721468	1.23998529
2.29393859	1.39232161	1.90291599	6.08813962	5.69209377	3.53671362	-11.276502
1.55223053	1.37327479	5.4697351	5.37484472	3.28494404	10.4219373	2.0429256
0.32725281	4.37634775	5.62861448	1.43669162	9.69987225	0.27199499	11.9643912
2.74129677	5.62846888	1.29151445	8.11905305	4.10360283	11.0245079	2.60490584
4.65537821	3.88735336	4.87228114	7.54448087	7.18990857	4.60954684	11.1295415
5.06861483	0.37469007	8.24353006	0.45463146	9.77135792	8.86694511	2.85102787
3.51215391	5.05118217	5.92231354	8.38763197	0.30370806	7.65072304	9.51723363
0.33587202	7.54044874	0.79175827	8.54948591	9.15958899	5.14401412	-2.8187953
3.36493541	6.26891985	7.19960514	0.78258093	7.06723337	8.63010175	7.67272855
6.16517665	1.67366481	8.69135006	7.44662349	3.20765597	1.27072671	2.59159116
6.89371113	3.95711239	4.19238411	8.49815821	8.67941746	7.74531193	5.91993876
5.13442445	7.70585454	3.07259216	1.83326958	3.42243036	1.72999746	2.25603321
1.42854604	7.61166231	8.00202248	5.93213502	5.10577523	5.71752101	4.41366609
2.91047713	3.72646559	7.31704918	7.71383987	6.57291269	3.42556636	1.88195647
6.29568009	1.93279284	1.78354432	2.50351251	0.31463451	3.35296957	3.20060805
7.47686946	6.48113823	4.55468882	4.31915001	5.30142038	3.88184338	1.51704202
6.01864011	7.68690911	7.38782107	6.53978986	3.97694826	1.26511083	2.26784746
2.46026639	5.09256724	5.11166667	2.77910037	1.59866395	3.44975851	1.18891709
1.89815529	0.16584043	0.30790087	2.8840725	4.36687158	0.22332348	1.57539823
5.49168909	4.54439991	4.98746331	5.24989598	1.72635271	2.55692399	0.91001775

-7.07164758	6.75202501	-5.92182179	2.73882986	-2.33286901	-1.05020743	-1.07554631
-6.15019511	5.55681275	-2.90205218	-1.74488823	2.99137312	-1.56674718	0.68267179
-3.14825193	1.79219385	-1.64979196	4.02527386	-0.19316951	1.32381134	0.72297532
-0.79459742	-2.51879659	4.58517095	-2.49016914	-2.24134756	-0.71583675	-0.50330322
-4.26163637	5.25440111	-4.19343069	0.91754271	1.68405681	-1.2261225	0.47913813
6.07739649	-5.2410323	1.15013739	2.96397034	0.61345894	0.11165851	0.36547253
-5.71042011	2.74028417	-2.22906089	2.13260537	-1.73531232	0.93948788	-0.31337598
-3.42211855	0.81270366	-3.71685902	-0.36397606	0.70019843	-0.23644212	-0.26185356
-0.12458916	3.61861984	-2.61453218	-2.10264902	0.87286235	-0.60549987	-0.20240826
-2.9821934	4.41962448	-0.00420044	-1.74215611	-1.13236229	-0.37625065	-0.18538495
-4.84920691	3.04885992	-2.23948171	-0.02534185	-0.09184687	-0.31193371	-0.12915731
-4.93217915	0.39389838	-2.71411505	-1.43889656	-0.80847657	-0.3766922	-0.12984609
-3.34193869	-2.14446553	-1.38680531	-1.36877487	-0.61489326	-0.09807025	-0.08143214
-0.75443928	-3.37989108	-0.62047598	-0.15838256	-0.20428069	-0.30337853	-0.09006645
-1.86849109	-2.8820445	-1.92118414	-0.94928444	-0.60584205	-0.03141543	-0.05072261
-3.63425595	-1.09281171	-1.79611079	-1.04010442	-0.25113157	-0.20617982	-0.06192274
-4.02016898	-0.98826615	-0.55071006	-0.23878476	-0.29115903	-0.09129387	-0.03119976
-3.01996524	-2.34650165	-0.84281792	-0.60200632	-0.38486526	-0.11608881	-0.04222875
-1.09921221	-2.43491152	-1.47453866	-0.76737325	-0.03751039	-0.10346767	-0.01893686
-1.00738059	-1.37422044	-1.06824555	-0.25575937	-0.26486635	-0.04769821	-0.02858313
-2.5719487	-0.18952549	-0.05386008	-0.36466558	-0.20492016	-0.08896041	-0.01132763
-3.12067123	-1.45798874	-0.8236991	-0.55115977	-0.06214088	-0.00404856	-0.01921276
-2.56999123	-1.87692041	-1.03168764	-0.23776034	-0.19485656	-0.06394618	-0.00666566
-1.21609604	-1.36472418	-0.55266448	-0.20839813	-0.08319948	-0.0183801	-0.01283073
-0.40397623	-0.28728406	-0.1922427	-0.38604649	-0.09011468	-0.03874734	-0.00384793
-1.71898771	-0.77480438	-0.68615426	-0.2036483	-0.12185773	-0.02570756	-0.00851669
-2.32031769	-1.32828638	-0.66098707	-0.10953806	-0.01384792	-0.01857298	-0.00217013
-2.08286251	-1.18646186	-0.22397673	-0.26393649	-0.0813343	-0.02408798	-0.00562086
-1.17692201	-0.5137248	-0.2762498	-	-0.064183	-0.00498025	-0.0011879

			0.16498905			
-0.02123408	-0.3010291	0.51308803	-0.04981294	-0.01773211	0.0184101	0.0036896
-1.07870888	-0.85833163	-0.38461116	-0.17617252	-0.05921509	-0.00261723	-0.00062428
-1.65796929	-0.93638368	-0.03769343	-0.12820755	-0.02607227	-0.01191548	-0.00240945
-1.61968677	-0.56891299	-0.26966141	0.0157765	-0.02645417	-0.00574379	-0.00030873
-1.04714276	-0.00786634	-0.35135356	-0.11473286	-0.03669257	-0.0063576	-0.00156574
-0.19284904	-0.49519996	-0.1974645	-0.09634866	-0.00475722	-0.00606659	-0.00013763
-0.62567428	-0.68030821	0.0518187	-0.00206326	-0.02383853	-0.00239494	-0.00101268
-1.13991337	-0.52269524	-0.22222415	-0.07278703	-0.01921829	-0.00497444	-4.8898E-05
-1.21458288	-0.14716698	-0.22146574	-0.07039277	-0.00481942	-1.5101E-05	-0.00065201
-0.87718279	-0.23917896	-0.08206444	-0.01015375	-0.01726116	-0.00343059	-5.9218E-06
-0.28941469	-0.45574851	-0.08246097	-0.04484918	-0.00783986	-0.00111665	-0.00041795
-0.32299091	-0.42834379	-0.16419767	-0.05017128	-0.00746525	-0.00199118	-1.2499E-05
-0.75345352	0.207306	-0.127608	-0.01271045	-0.01064161	0.0014203	-0.00026677
-0.88121844	-0.07506862	-0.01814275	-0.02670546	-0.00154844	-0.00090006	-1.8373E-05
-0.70116289	-0.27863241	0.0817068	-0.03496297	-0.0067451	-0.00126869	-0.00016957
-0.31125737	-0.32162234	0.1112628	-0.01237997	-0.00556121	-0.00020054	-1.8315E-05
-0.13289029	-0.20943443	-0.06555328	-0.01523479	-0.00125922	-0.00093386	-0.00010734
-0.47709591	-0.01841936	-0.01228358	-0.02385603	-0.00487369	-0.00016875	-1.5835E-05
-0.61979499	-0.15069522	-0.06733131	-0.01074813	-0.00228335	-0.00058276	-6.768E-05
-0.53923293	-0.22331908	-0.06959479	-0.00819683	-0.00204291	-0.00030593	-1.2696E-05
-0.29058795	-0.18136804	-0.02795155	-0.01594919	-0.0029977	-0.00029746	-4.2502E-05
-0.02219578	-0.06259921	-0.02298365	-0.00871044	-0.00048331	-0.00030583	-9.7094E-06
-0.28723049	-0.06581714	-0.04956212	-0.00402822	-0.00185656	-0.00010265	-2.6585E-05
-0.4229112	-0.14309797	-0.03997255	-0.010449	-0.00156658	-0.00024174	-7.1865E-06
-0.40112437	-0.14191405	-0.00733819	-0.00673121	-0.00031809	-9.1507E-06	-1.6564E-05
-0.24964051	-0.07559241	-0.02340786	-0.00166656	-0.00134178	-0.00016136	-5.1919E-06
-0.03559889	-0.014659	-0.03344863	-0.00670463	-0.00064829	-5.8901E-05	-1.0279E-05
-0.16194389		-0.02063466	-0.00501772		-9.0334E-05	-3.6811E-06

	0.08333701			0.00054541		
	-	-	-			
0.27982318	0.10235645	0.00248458	0.00040838	-0.000825	6.9446E-05	6.3528E-06
	-		-	-	-3.8609E-	-2.5708E-
0.28962649	0.07115084	-0.0194296	0.00420807	0.00014587	05	06
	-	-	-		-5.9662E-	-3.9104E-
0.20250959	0.01239693	0.02086954	0.00363308	0.00049976	05	06
	-		-			
0.06022933	0.04229006	0.00901844	0.00020001	0.00043183	6.7026E-06	1.7731E-06
-				-7.7939E-		
0.08278884	0.06841985	0.00606293	0.00257761	05	4.2579E-05	2.3969E-06
-				-		-1.2101E-
0.17918826	0.05873449	0.01432435	0.00256617	0.00036191	9.3391E-06	06
-				-	-2.5755E-	-1.4629E-
0.20342115	0.02368066	0.01199573	0.00044306	0.00018027	05	06
-	-				-1.4701E-	
0.15726661	0.01636481	0.0026713	-0.0015352	0.00014267	05	8.183E-07
-	-	-	-			
0.06553922	0.04214818	0.00642228	0.00177318	0.00022276	1.2614E-05	8.8881E-07
	-	-	-			-5.4896E-
0.0352577	0.04421929	0.00966892	0.00049361	4.2835E-05	1.4035E-05	07
	-	-		-	-3.9471E-	-5.3749E-
0.1106087	0.02563232	0.00623756	0.00088369	0.00013208	06	07
				-	-1.0753E-	
0.1391365	0.00157198	0.00037356	0.00120075	0.00011692	05	3.6567E-07
					-8.3628E-	
0.11787256	0.02343027	0.00539982	0.00045276	1.8549E-05	07	3.2343E-07
						-2.4203E-
0.06070642	0.03080945	0.00603773	-0.0004866	9.5972E-05	6.9764E-06	07
-			-			
0.00853349	0.02278816	0.00279025	0.00079766	4.9262E-05	2.832E-06	-1.936E-07
-		-	-	-3.6685E-	-3.7779E-	
0.06533351	0.00568145	0.00152427	0.00037691	05	06	1.5926E-07
-	-	-		-5.9199E-	-3.1344E-	
0.09269321	0.01109963	0.00400261	0.00025153	05	06	1.1523E-07
-	-	-		-1.2294E-		-1.0425E-
0.08569913	0.01990598	0.00348184	0.00052002	05	1.5237E-06	07
	-	-				-6.8158E-
-0.0514627	0.01806544	0.00090479	0.0002954	3.4373E-05	2.6034E-06	08
-	-		-		-1.7882E-	
0.00508974	0.00825633	0.00170123	0.00011726	3.1182E-05	07	6.7905E-08
			-	-4.2859E-	-1.8079E-	
0.03642966	0.00364464	0.00271022	0.00033263	06	06	4.0044E-08
			-	-2.5087E-		
0.06009689	0.01181472	0.00182616	0.00022186	05	-4.669E-07	-4.403E-08
		-8.4857E-		-1.3264E-		-2.3348E-
0.06063478	0.01315138	06	4.4107E-05	05	1.0626E-06	08
		-				
0.04113994	0.00822682	0.00145603	0.00020858	9.2941E-06	6.5839E-07	2.8428E-08
		-			-4.9951E-	
0.01087602	0.00038796	0.00169772	0.00016127	1.5521E-05	07	1.3498E-08
-	-	-	-6.9001E-		-6.0379E-	-1.8281E-
0.01865794	0.00626267	0.00083483	06	3.4605E-06	07	08
-	-		-	-8.8282E-		-7.7261E-
0.03783947	0.00888237	0.00036606	0.00012802	06	1.3912E-07	09
-	-	0.00108845	-	-8.21E-06	4.4993E-07	1.1711E-08

0.04183016	0.00697391		0.00011414			
-0.03148907	-0.00220432	0.00098391	-9.9479E-06	9.5916E-07	5.2677E-08	4.3713E-09
-0.01227485	0.00273866	0.00029065	7.6692E-05	6.4778E-06	-2.8437E-07	-7.4749E-09
-0.00821335	0.00555891	-0.00043739	7.8951E-05	3.5266E-06	-1.2748E-07	-2.4389E-09
-0.02304873	0.00533688	-0.00074096	1.5832E-05	-2.3243E-06	1.4908E-07	4.7547E-09
-0.02816377	0.00270972	-0.00052091	-4.4655E-05	-4.0226E-06	1.3364E-07	1.3373E-09
-0.02327907	0.00069489	-2.4077E-05	-5.3499E-05	-9.5803E-07	-5.6449E-08	-3.0143E-09
-0.01144859	0.00317987	-0.00038287	-1.6241E-05	2.2416E-06	-1.0769E-07	-7.1718E-10
-0.00242874	0.00376931	-0.0004664	2.5095E-05	2.1378E-06	2.8621E-09	1.9048E-09
-0.01349057	0.00252869	-0.00024301	3.5568E-05	-2.0679E-07	7.2926E-08	3.7328E-10

WH13	WH14	WH15	WH16
0	0	0	0
-0.43503514	-1.17807058	-2.34207687	-3.85801154
-3.95919171	-2.51568137	-2.01394401	-8.70349012
-5.84831585	-9.73059419	-4.73005827	-9.98841983
-4.49329437	-8.28923686	-12.6173152	-7.72317585
-11.9085488	-3.64054161	-15.0555558	-3.61081026
-1.46966177	-13.4725811	-10.2339955	-0.61713853
-12.1210479	-10.5707777	-1.4644846	-3.86146259
-7.98320996	-1.42504557	-6.21287761	-5.71655896
-6.94401771	-10.2201171	-9.47977703	-6.27313066
-10.4571195	8.6284793	-8.07239826	-5.87368326
-0.53160744	-0.35913356	-3.98898811	-4.91973554
-8.61361275	-5.94534676	-0.21188622	-3.75702882
-3.76033751	5.6468972	-2.79199386	-2.62838263
-4.70320226	-0.97503954	-3.32080911	-1.67150068
-5.00523016	-2.93206933	-2.37765596	-0.93979691
-1.05700787	-3.22196242	-0.91643418	-0.43027909
-4.05848273	-0.90938428	0.2823324	-0.10916119
-1.13582967	-1.27157008	-0.86345927	-0.06885408

-2.28968803	-1.66890585	-0.86556106	-0.14828658
1.82489159	0.63650562	0.53883822	0.16648975
-0.72112713	-0.48730338	-0.15367511	-0.15145596
-1.55144053	-0.8024126	-0.1159306	0.12216422
-0.22002888	-0.38267839	0.2186743	0.09017285
-0.92525015	-0.16023981	0.19123135	0.06158291
-0.55854668	-0.36276319	0.10400569	-0.0388695
-0.3529844	0.20830689	0.01812052	0.0223526
-0.51847256	-0.04007323	0.03419144	0.01125431
-0.00305351	-0.15533564	0.04872787	0.00438479
-0.33041756	-0.10545226	-0.03782791	-0.00053825
0.14788066	0.00283484	0.01778099	-0.0013141
-0.14425722	-0.06320237	-0.00063333	0.0019592
-0.15676848	-0.05042833	-0.00845638	-0.00194903
-0.02186605	-0.00528017	0.0099118	0.00164031
-0.10774521	-0.02442144	-0.00688574	-0.00124363
-0.03347935	-0.02300427	-0.00272104	0.00086916
-0.05238353	-0.00498384	-0.00043265	-0.00056366
-0.04359068	-0.00891558	-0.00186637	0.00033751
-0.01339791	-0.01007394	-0.00187982	-0.00018286
-0.03268786	-0.00319914	-0.00117017	8.4834E-05
-0.00595719	-0.00303658	-0.00036742	-2.7729E-05
-0.01745548	-0.00425189	-0.00017508	-2.0126E-06
-0.01120293	-0.001758	-0.00037957	1.4829E-05
-0.00580339	-0.00093782	-0.00033675	-1.8086E-05
-0.0093278	-0.00173355	-0.00018691	1.6593E-05
-0.00050131	-0.00088285	-4.1128E-05	-1.3246E-05
-0.00543879	-0.00024391	-4.6847E-05	9.6278E-06
-0.00264668	-0.00068326	-7.248E-05	-6.4775E-06

0.00214551	0.00041672	5.7448E-05	4.0455E-06
-	-3.9416E-	-2.8084E-	-2.3245E-
0.00251843	05	05	06
-			
0.00020905	0.00026011	2.8985E-06	1.1951E-06
	-		-5.0861E-
0.00160305	0.00018762	1.0584E-05	07
-		-1.3151E-	
0.00056239	8.671E-06	05	1.2738E-07
-			
0.00071916	9.538E-05	9.3782E-06	5.8605E-08
	-8.1259E-	-3.9453E-	-1.2931E-
0.00064452	05	06	07
		-2.1789E-	
0.00015363	1.328E-05	07	1.3844E-07
-			-1.1946E-
0.00045021	3.3497E-05	2.1652E-06	07
0.00010098	-3.403E-05	-2.285E-06	9.1477E-08
			-6.4258E-
0.00022473	9.0871E-06	1.4685E-06	08
-		-5.0942E-	
0.00015598	1.1147E-05	07	4.1886E-08
-6.6817E-	-1.3823E-	-1.5118E-	-2.5316E-
05	05	07	08
0.00012099	5.0256E-06	4.1355E-07	1.3998E-08
-1.2011E-		-3.8218E-	-6.8173E-
05	3.4418E-06	07	09
-6.6465E-	-5.4546E-		
05	06	2.2057E-07	2.6086E-09
		-5.7934E-	-3.7485E-
3.5408E-05	2.4989E-06	08	10
		-4.2791E-	-6.4084E-
2.4171E-05	9.4033E-07	08	10
-3.1172E-	-2.0917E-		
05	06	7.4971E-08	9.6623E-10
-9.8671E-		-6.1736E-	-9.4254E-
07	1.1616E-06	08	10
1.8774E-05	1.9659E-07	3.1691E-08	7.7405E-10
-7.3974E-	-7.7871E-	-5.0849E-	
06	07	09	-5.722E-10
-7.8928E-		-9.6529E-	
06	5.1428E-07	09	3.9012E-10
			-2.4709E-
7.6957E-06	7.4727E-09	1.3027E-08	10
	-2.8075E-	-9.6488E-	
1.3655E-06	07	09	1.4476E-10
-5.0918E-			-7.6964E-
06	2.191E-07	4.3266E-09	11
		-1.1648E-	
1.3594E-06	-2.424E-08	10	3.5254E-11
	-9.7564E-	-1.9438E-	-1.1646E-
2.4049E-06	08	09	11
-1.8145E-			-3.0468E-
06	9.0354E-08	2.1834E-09	13
-6.3731E-	-1.9751E-	-1.4597E-	5.3012E-12

07	08	09	
	-3.2404E-08		-6.5112E-12
1.3301E-06		5.5362E-10	
-1.91E-07	3.6195E-08	9.1545E-11	5.9166E-12
-6.9524E-07	-1.1475E-08	-3.6421E-10	-4.661E-12
4.0553E-07	-1.0126E-08	3.5435E-10	3.3398E-12
2.3363E-07	1.4111E-08	-2.1355E-10	-2.2155E-12
-3.3498E-07	-5.805E-09	6.4472E-11	1.3653E-12
5.6145E-09	-2.8833E-09	3.1249E-11	-7.7519E-13
1.9249E-07	5.3572E-09	-6.4775E-11	3.9506E-13
-8.459E-08	-2.7125E-09	5.5826E-11	-1.6801E-13
-7.6127E-08	-6.8886E-10	-3.0122E-11	4.4013E-14
8.1274E-08	1.9793E-09	6.3395E-12	1.5531E-14
9.8802E-09	-1.2008E-09	7.385E-12	-3.7849E-14
-5.1323E-08	-9.6474E-11	-1.1057E-11	4.0733E-14
1.5926E-08	7.1034E-10	8.5484E-12	-3.4973E-14
2.3029E-08	-5.103E-10	-4.0731E-12	2.6575E-14
-1.8939E-08	2.8351E-11	3.8715E-13	-1.8509E-14
-5.3498E-09	2.4669E-10	1.5069E-12	1.1963E-14
1.3217E-08	-2.0971E-10	-1.8239E-12	-7.1744E-15
-2.4752E-09	3.5575E-11	1.2725E-12	3.9424E-15
-6.6E-09	8.2352E-11	-5.2238E-13	-1.9155E-15
4.2111E-09	-8.3702E-11	-3.3067E-14	7.4009E-16
2.0398E-09	2.2699E-11	2.8277E-13	-1.2231E-16
-3.2911E-09	2.6117E-11	-2.9192E-13	-1.565E-16
2.08E-10	-3.2527E-11	1.8394E-13	2.4588E-16
1.8117E-09	1.1919E-11	-6.2228E-14	-2.4085E-16
-8.8244E-10	7.6914E-12	-1.9126E-14	1.973E-16
-6.734E-10	-1.232E-11	5.0127E-14	-1.4518E-16
7.9196E-10	5.67E-12	-4.545E-14	9.8471E-17
5.4912E-11	1.9975E-12	2.5757E-14	-6.2055E-17

-4.7927E-10	-4.5473E-12	-6.5696E-15	3.6201E-17
1.699E-10	2.5337E-12	-4.9928E-15	-1.9203E-17
2.0445E-10	3.8852E-13	8.5132E-15	8.82E-18
-1.837E-10	-1.6334E-12	-6.8928E-15	-2.9813E-18
-4.0403E-11	1.0822E-12	3.4775E-15	3.9991E-20
1.2261E-10	3.6491E-15	-5.3291E-16	1.1889E-18
-2.8285E-11	-5.6934E-13	-1.0563E-15	-1.4938E-18
-5.8578E-11	4.4601E-13	1.3964E-15	1.3622E-18
4.0852E-11	-5.2997E-14	-1.0185E-15	-1.072E-18
1.6487E-11	-1.9156E-13	4.4891E-16	7.6611E-19
-3.0377E-11	1.7835E-13	-8.6387E-18	-5.0666E-19
3.2891E-12	-4.0106E-14	-2.0132E-16	3.1136E-19
1.605E-11	-6.1659E-14	2.223E-16	-1.7648E-19

11. Anexo B

En este capítulo está el código de Matlab que se uso para entrenar la red neuronal. Los códigos aparecen en el orden en que aparecen en el documento en el capitulo 6.

```
%Indices
conta2 = 1;
conta3 = 120;
fda = input('Factor de aprendizaje ');
error=0;
x = 0;
%inicializacion de numero de neuronas
tamanoentrada = 36;
neuronascapa1 = 36;
neuronascapa2 = 10;
neuronascapasalida = 5;

vectorerrorsalida = zeros(5,1);
vectorerrorcapa2 = zeros(10,1);
vectorerrorcapa1 = zeros(36,1);

ro1 = zeros((neuronascapa2*neuronascapasalida),1);
ro2 = zeros((neuronascapa2*neuronascapa1),1);
ro3 = zeros((neuronascapa1*tamanoentrada),1);

segmentopresente = zeros(120,1);
segmentoanterior = zeros(120,1);

vectordelta50 = zeros((neuronascapasalida*neuronascapa2),1);
vectordelta360 = zeros((neuronascapa2*neuronascapa1),1);
vectordelta1296 = zeros((tamanoentrada*neuronascapa1),1);

iteraciones = input('Iteraciones ');
```

```
%numero de veces que se va a entrenar
for i3 = 1:1:iteraciones


  %envio de datos al vector de parametros


  if (conta3 > 3240)
     conta2 = 1;
     conta3 = 120;
  end


  for i4 = 1:1:120
     segmentoanterior(i4,1) = a(conta2,1);
     segmentopresente(i4,1) = a(conta3,1);
     conta2 = conta2+1;
     conta3 = conta3+1;
  end


  %llama a vectordeparametros para obtener las entradas de la red
  %neuronal
  vectordeparametros
  %llama a redneuronal para calcular la salida en base al vector de
  %parametros de entrada
  redneuronal


  %calculo del error y propagacion hacia atras


     %en la capa de salida
     vectorerrorsalida = zeros (5,1);
     vectorerrorcapa2 = zeros (10,1);
     vectorerrorcapa1 = zeros (36,1);
```

```
%en la capa de salida
for i5 = 1:1:neuronascapasalida

   vectorerrorsalida(i5,1)=deseada(i5,1)-salida(i5,1);

end


for i5 = 1:1:neuronascapasalida


   for i6 = 1:1:neuronascapa2

      indice = (i5*neuronascapa2)+i6-neuronascapa2;
      vectorerrorcapa2(i6,1) = vectorerrorcapa2(i6,1)+(W3(indice,1)*vectorerrorsalida(i5,1));
   end

end


for i5=1:1:neuronascapa2

   for i6 = 1:1:neuronascapa1

      indice = (i5*neuronascapa1) + i6-neuronascapa1;
      vectorerrorcapa1(i6,1) = vectorerrorcapa1(i6,1)+(W2(indice,1)*vectorerrorcapa2(i5,1));

   end

end
```

```
    for i5=1:1:neuronascapa1
       ro3(i5,1) = vectorerrorcapa1(i5,1)*fprima1(i5,1);
       ro3(i5,1) = fda*ro3(i5,1);
    end

    for i5=1:1:neuronascapa2
       ro2(i5,1) = vectorerrorcapa2(i5,1)*fprima2(i5,1);
       ro2(i5,1) = fda*ro2(i5,1);
    end

    for i5=1:1:neuronascapasalida
       ro1(i5,1) = vectorerrorsalida(i5,1)*fprima(i5,1);
       ro1(i5,1) = fda*ro1(i5,1);
    end


    W1 = W1+ro3;
    W2 = W2+ro2;
    W3 = W3+ro1;

end

salida


%red1
tamano = 2;
nc1 = 48;
nc2 = 16;
nc3 = 1;
fda = .00001;
deseada = 0;
%error = zeros(5000,1);
```

```
vector = [0;0];
delayRatio = 0.95;
dW1=zeros(size(W1));
%dB1=zeros(size(B1));
dW2=zeros(size(W2));
%dB2=zeros(size(B2));
dW3=zeros(size(W3));
%dB3=zeros(size(B3));

for iteraciones = 1:1:5000
  %propagacion hacia adelante
   o1 = (W1 * vector);
  fprima1 = sech(o1).*sech(o1);
  Z1 = o1;
  o1 = tanh(o1);
  o2 = (W2 * o1);
  fprima2 = sech(o2).*sech(o2);
  Z2 = o2;
  o2 = tanh(o2);
  o3 = (W3 * o2);
  fprima3 = sech(o3).*sech(o3);
  Z3 = o3;
  o3 = tanh(o3);
  %vector de error
  e = o3-deseada;
  D3=e;
  D2=(1-(o2.*o2)).*(W3'*D3);
  D1=(1-(o1.* o1)).*(W2'*D2);

    h=(1-delayRatio)*fda;
    %h1 = .0001;
    %h2 = .0001;
    dW1=delayRatio*dW1+h*(D1*vector');
```

```
%dB1=delayRatio*dB1;
%dB1 = dB1 + h*(D1);
dW2=delayRatio*dW2+h*(D2*Z1');
%dB2=delayRatio*dB2+h*(D2);
dW3=delayRatio*dW3+h*(D3*Z2');
%dB3=delayRatio*dB3+h*(D3);

%Obtiene los nuevos pesos y desplazamientos
W1=W1+dW1;
%B1=B1+dB1;
W2=W2+dW2;
%B2=B2+dB2;
W3=W3+dW3;
%B3=B3+dB3;
switch mean(vector)
  case 0
    vector = [0;1];
    deseada = 1;



 case 0.5

    if vector(1,1) == 0

      vector = [1;0];
      deseada = 1;

    else

      vector = [1;1];
      deseada = 0;
```

```
            end

        case 1

            vector = [0;0];
            deseada = 0;

    end

end

%red2
%Factor de aprendizaje
        learningRatio=0.001;

  %diferencia entre esta clasificacion y la esperada
  o1=tanh(W1*vector);
  o2=tanh(W2*o1);
  o3=W3*o2;
  E=deseada-o3;


  %Determina la diferencia entre la salida inicial y la deseada para cada
  %nodo de la red. La derivada de la funcion de activacion esta dada como:
  %diff(tanh(x)) = 1-tanh(x)^2
  D3=E;
  D2=(1-(o2.*o2)).*(W3'*D3);
  D1=(1-(o1.* o1)).*(W2'*D2);
```

```
for iteraciones = 1:1:100000


  %propagacion hacia adelante


  o1 = (W1 * vector);
  fprima1 = sech(o1).*sech(o1);
  Z1 = o1;
  o1 = tanh(o1);


  o2 = (W2 * o1);
  fprima2 = sech(o2).*sech(o2);
  Z2 = o2;
  o2 = tanh(o2);


  o3 = (W3 * o2);
  fprima3 = sech(o3).*sech(o3);
  Z3 = o3;
  %o3 = tanh(o3);


  %vector de error
  e = o3-deseada;
  D3=e;
  D2=(1-(o2.*o2)).*(W3'*D3);
  D1=(1-(o1.* o1)).*(W2'*D2);


     h=learningRatio;
     dW1=delayRatio*dW1+h*(D1*vector');
```

```
    dW2=delayRatio*dW2+h*(D2*o1');


    dW3=delayRatio*dW3+h*(D3*o2');
        %Obtiene los nuevos pesos y desplazamientos
    W1Tmp=W1+dW1;
    W2Tmp=W2+dW2;
    W3Tmp=W3+dW3;
      W1=W1Tmp;
      %B1=B1Tmp;
      W2=W2Tmp;
      %B2=B2Tmp;
      W3=W3Tmp;
      %B3=B3Tmp;
      %y los parametros de entrenamiento
      o1=o1Tmp;
      o2=o2Tmp;
end
```

12. Bibliografía

[1] Howard, David, et. al., "ACOUSTICS AND PSYCHOACOUSTICS". Ed: Focal Press. Second Edition, 2001

[2] Greenwood, Mark, et. al., "SUVing: AUTOMATIC SILENCE/UNVOICED/VOICED CLASSIFICATION OF SPEECH", Department of Computer Science, University of Sheffield. (http://plaza.ufl.edu/dongsoo7/speech1.pdf)

[3] Schroeder MR, et. al., "OBJECTIVE MEASURE OF CERTAIN SPEECH SIGNAL DEGRADATIONS BASED ON MASKING PROPERTIES OF THE HUMAN AUDITORY PERCEPTION", Frontiers of Speech Communication Research, Academic Press, 1979.

[4] Treviño, Jorge, et. al. "Reconocimiento de voz esofágica empleando redes neuronales artificiales", 2005

[5] Vizcaíno, Rogelio, et. al. "Reconocimiento de voz esofágica en tiempo real mediante un DSP"

[6] Treviño López Jorge Alberto y Ortal Vite Patricia Isabel "Reconocimiento de voz esofágica empleando redes neuronales artificiales", ITESM-CCM, México DF 2005.

[7] http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html Septiembre 2007

[8] Vázquez de la Iglesia, et al. "Voz esofágica", Revista Médica Universidad de Navarra, Vol. 50, No. 3, 2006.

IMPLEMENTACIÓN DE ALGORITMOS DE ANÁLISIS Y RECONOCIMIENTO DE VOZ ESOFÁGICA EN UN DSP

Autores: Adolfo Abraham Méndez Cervantes 952938
Miguel Arturo Castillo Arroyo 954785
Alan Guillermo Galicia González 1105511
Luis Antonio Martínez Arias 1107173

Asesor: Dr. Alfredo Victor Mantilla Caeiros

Profesora: Dra. Katya Eugenia Romo Medrano Mora

Campus Ciudad de México
División de Ingeniería y Arquitectura
Departamento de Ingeniería Electrónica y Mecánica

IEC

Abril 2008

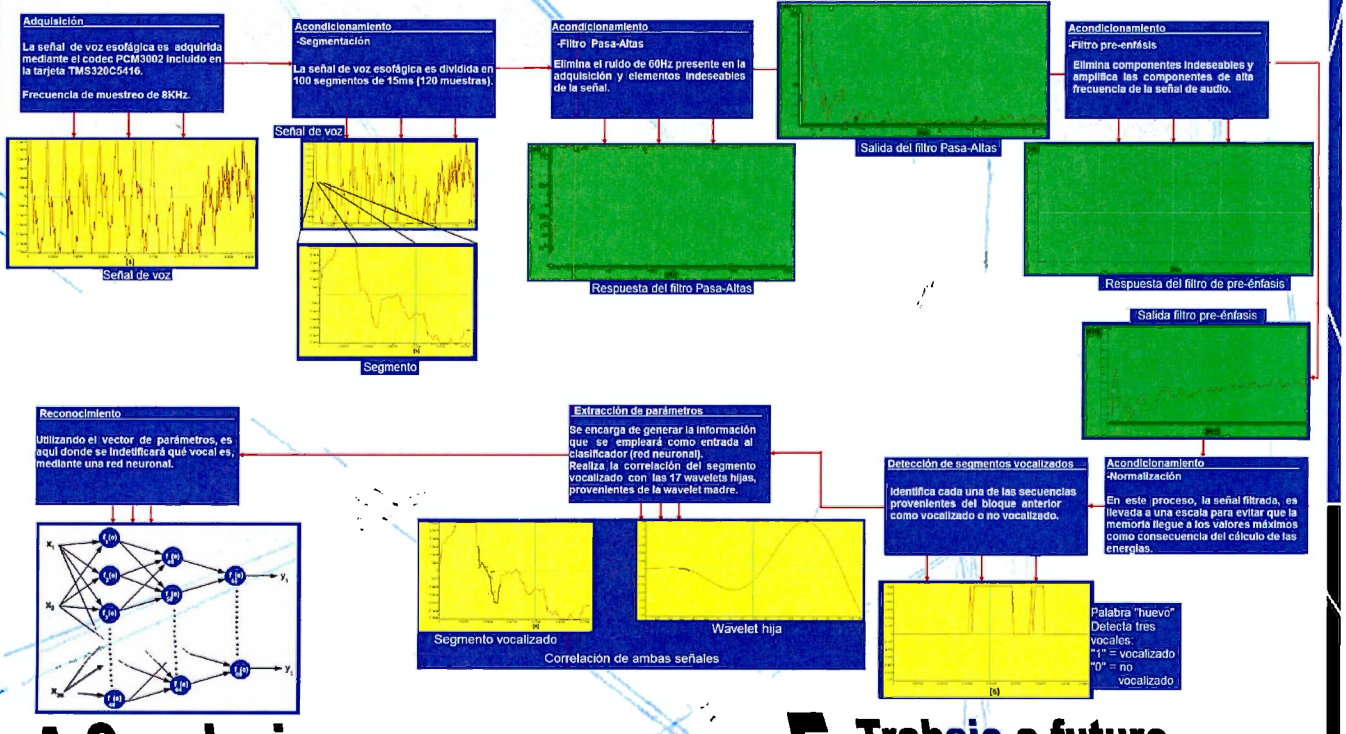
1 Descripción del problema

El cáncer de laringe es la aparición de células malignas en ésta, como consecuencia se le practica al paciente una laringectomía, procedimiento que consiste en extirpar el órgano infectado, el cual contiene las cuerdas vocales encargadas de generar el sonido de la voz.
Debido a que el paciente pierde la posibilidad de comunicarse, se le enseña a hablar mediante el esófago, a esta técnica se le conoce como voz esofágica, caracterizada por ser ronca e ininteligible sobre todo las vocales.

2 Objetivo

Programar en un DSP (Digital Signal Processor) los algoritmos que permitan analizar y reconocer voz esofágica para ayudar a que las personas que se comunican de esta manera lo hagan eficientemente.

3 Sistema propuesto y resultados



4 Conclusiones

Las etapas del sistema propuesto han sido implementadas con éxito en el DSP, los resultados han sido corroborados con Matlab y han sido similares.
La implementación se realiza con un formato multitareas, ha presentado problemas como el mapeo de memoria y la interrupción de tareas, los cuales han sido resueltos satisfactoriamente.

5 Trabajo a futuro

Realizar la etapa de síntesis, la cual consiste en sustituir la vocal reconocida por una de mejor calidad y más entendible.
Terminar la implementación utilizada de formato multitareas para que pueda ser en tiempo real.