Memoria del proyecto:

“Audio Fingerprint”

- DSP 2009 -
- Facultad de Ingenieria -
- Universidad de la Republica -

NOMBRE	CI	e-mail
Edgardo Vaz	3.353.188-4	edgardovaz@gmail.com
Melina Rabinovich	3.006.885-0	mrabinovichm@gmail.com
Daniel Aicardi	3.361.507-4	daicav@gmail.com

Tutor: Juan Cardelino

Resumen

En el presente trabajo se detallan los aspectos teóricos y prácticos usados para la realización del proyecto de curso de la asignatura *"Procesadores digitales de señal (DSP)"*.

La idea básica de este proyecto ha sido identificar señales de voz y de audio a partir de su huella característica.

En el desarrollo de este trabajo se expondrá el método utilizado para la solución del problema, así como las dificultades que surgieron tanto en la realización del software como en la implementación en hardware, explicando las soluciones encontradas para lograr el funcionamiento del sistema.

Tabla de contenido:

Introducción

¿A quién no le ha pasado escuchar una canción en la radio y querer saber como se llama?. Que bueno sería tener un medio simple de usar, que nos permitiera de manera rápida saber como se llama esa canción.

La idea de este proyecto ha sido desarrollar un sistema que fuera capaz de reconocer señales de audio, en particular música, a partir de una característica que la identifica de forma única. Esto último es posible generando una huella de audio que la diferencia del resto del universo de canciones.

Este método tiene la gran ventaja de utilizar menor cantidad de memoria a la hora de construir una base de datos de huellas conocidas que nos permita comparar y determinar la canción que estamos buscando. Una posible desventaja es que no podemos reconstruir la señal original a partir de la huella que la caracteriza.

Objetivos

1. Estudiar distintos algoritmos que permitan generar huellas descriptivas de señales de audio.
2. Generar huellas (*fingerprint*) de señales conocidas, que serán almacenadas en memoria.
3. Definir un umbral que evite *falsos positivos* a la hora de reconocer la señal (mediante su *fingerprint*) que se está analizando, contra las que se tienen almacenadas en memoria.

Alcance

- Implementar los algoritmos ZCR (cruces por cero) y YIN (función diferencia) en código C para la generación de *huellas de audio*, en una primera etapa de voz y luego de música.

- Comparación de ambos algoritmos como ser, *falsos positivos* o *falsos negativos* y velocidad en la *búsqueda*.

- Definición de parámetros:
  - *Ancho de banda* de las señales.
  - *Frecuencia de muestreo*.
  - Longitud de los *tramos* de señal (tamaño de *ventana*).
  - Filtrado de la señal de entrada.

- *Interfaz* con el usuario mediante mensajes desplegados en un *display*.

- Diseñar y fabricar una placa de circuito impreso, con la cual conectar un display *LCD16x2* a un puerto *GPIO* del *DSP56303*.

- Generar un módulo en código C con las funciones básicas para la comunicación del controlador *(KS0066)* del display con el DSP .

Algoritmos

Los dos algoritmos estudiados pertenecen al grupo del *dominio del tiempo*. Estos permiten la detección de la frecuencia fundamental en un pequeño intervalo de tiempo.
En una primera etapa los mismos fueron implementados en *Scilab* y luego llevado a código C en el caso del *algoritmo ZCR* para su uso directo sobre el DSP, aunque la idea durante la etapa de especificaciones de este proyecto era poder implementar también el *algoritmo YIN* y poder realizar la comparación del desempeño de ambos.

Cruces por cero (ZCR)

Dentro de la literatura usada se reconoce como el más simple, descartándose en la mayoría de los casos por ser poco preciso cuando se emplea con señales ruidosas. Su principal ventaja es que gracias a su simplicidad tiene bajo consumo de procesamiento.

Su idea básica es contar la cantidad de veces que la señal cruza un nivel de referencia, en este caso cero, de ahí su nombre, aunque hay que tener en cuenta si la señal tiene componente de continua. Cada período de la señal cuenta con dos cruces por cero, por tanto una vez que se cuenta con la cantidad de cruces por cero (en un segundo), se divide por dos para obtener la frecuencia fundamental, f_0.

Función diferencia cuadrática normalizada (YIN)

Este algoritmo se basa en el de *autocorrelación* que usa la función de *correlación* para hallar la similitud de la señal consigo misma desplazada en el tiempo. Cuando la señal es desplazada un tiempo equivalente a su período, la *autocorrelación* presenta un máximo. En el caso de la función diferencia, ésta presenta un mínimo. La diferencia entre dos mínimos consecutivos equivale al período de la señal analizada, y de ahí se puede deducir la frecuencia fundamental, f_0, que se desea hallar. La ecuación de la función diferencia cuadrática viene dada por:

$$(1) \qquad d(\tau) = \sum_{j=0}^{W+\tau_{max}} (x_j - x_{j+\tau})^2$$

Donde τ es la variable de retardo, en el caso que la señal x sea periódica, cuando $\tau = T$ (período de la señal) la función d presenta un cero, en nuestro caso las funciones que manejamos no son exactamente periódicas, entonces la función d presenta un mínimo que es distinto de cero. W es el tamaño de ventana de tiempo donde se quiere hallar la frecuencia fundamental f_0.

Se debe tener en cuenta que el retardo máximo debe ser mayor o igual que el valor del período máximo que se pretende hallar.

La normalización llevada a cabo consiste en dividir la función diferencia por la media acumulada.

$$(2) \qquad d_{norm}(\tau) = \begin{cases} 1, & \text{si } \tau = 0 \\ \dfrac{d(\tau)}{\frac{1}{\tau}\sum_{j=1}^{\tau} d(j)}, & \text{en otro caso} \end{cases}$$

Diseño del sistema

Hardware:

El hardware empleado es el kit de desarrollo *DSP56303EVM* de *Motorola*. El mismo cuenta con el *Codec Crystal CS4215*, el cual es usado para adquirir las muestras de la señal de entrada. A su vez, cuenta también con la interfaz *HI08*, la cual puede ser usada de dos maneras diferentes, una de ellas para la conexión con otro *Microprocesador* o *DSP*, y la otra es para ser usada como puerto de *entrada-salida* de propósito general *(GPIO)*.

En nuestro caso hemos usado esta interfaz como *GPIO* para la conexión de un *display LCD16x2*, *Winstar WH1602D*, este último está basado en un controlador de *Samsung (KS0066)* equivalente al *Hitachi (HD44780)*, el cual se encarga del manejo de la memoria interna y de la matriz generadora de puntos en la pantalla lcd; esto lo hace simple de usar, ya que solo es necesario enviar mensajes de control o de datos a través de sus pines I/O.

➢ *Construcción del PCB:*

Para interconectar el *DSP* con el display fue necesario fabricar una placa de circuito impreso (PCB) de una sola capa, ésta fue diseñada usando la herramienta de diseño *CAD, KiCad,* la cual es *open source GPL.*

A continuación se muestra el esquemático y la lista de componentes empleados en la fabricación.

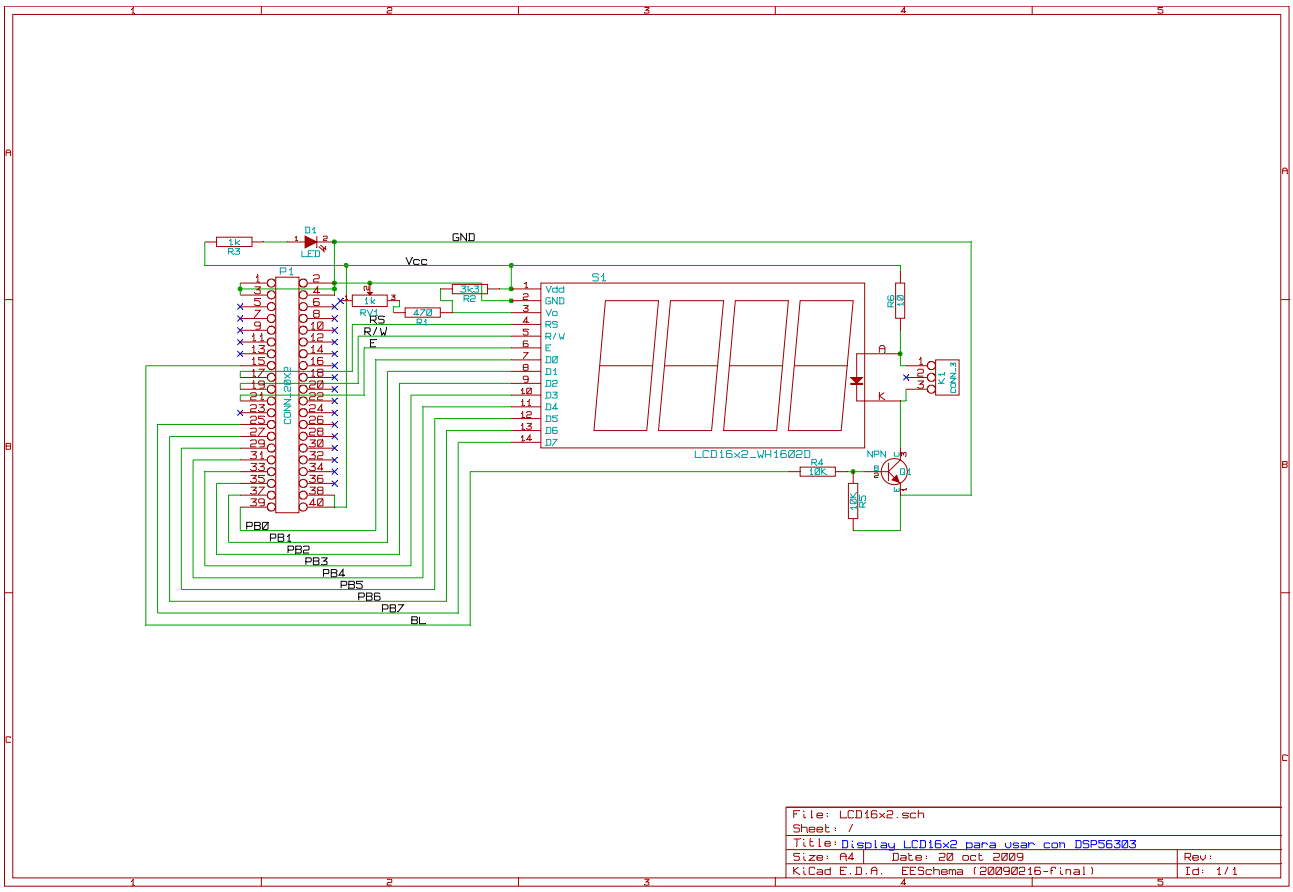

Figura 1. Esquemático de la placa diseñada.

Descripción	Componente	Valor
Led	D1	-
Conector 3x1	K1	-
Conector 20X2	P1	-
Transistor NPN	Q1	BC547
Resistencia	R1	470Ω
Resistencia	R2	3,3KΩ
Resistencia	R3	1KΩ
Resistencia	R4	10KΩ
Resistencia	R5	10KΩ
Resistencia	R6	10Ω
Preset	RV1	2,2KΩ
LCD16x2	S1	WH1602D

Tabla 1. Componentes empleados en el PCB.

➢ *Configuración GPIO:*

La interfaz *HI08 (port B)* se encuentra disponible en el conector *J8 (ISA 20x2)* del kit. Los pines de este conector son empleados para controlar el *display LCD16x2* y se detallan en las siguientes tablas:

HI08 port pin	Multiplexed address/data bus mode	Non-Multiplexed bus mode	GPIO mode	Nº Pin ISA J8
HAD0–HAD7	HAD0–HAD7	H0–H7	PB0–PB7	39..25(impares)
HAS/HA0	HAS/HAS	HA0	PB8	23
HA8/HA1	HA8	HA1	PB9	21
HA9/HA2	HA9	HA2	PB10	19
HCS/HA10	HA10	HCS/HCS	PB13	17

Table 2 HI08 Signal Pin Definitions for Various Operational Modes

HI08 port pin	Single strobe bus	Dual strobe bus	GPIO mode	Nº Pin ISA J8
HRW/HRD	HRW	HRD/HRD	PB11	11
HDS/HWR	HDS/HDS	HWR/HWR	PB12	9

Table 3 HI08 Data Strobe Signal Pins

HI08 port pin	Vector required	No vector required	GPIO mode	Nº Pin ISA J8
HREQ/HTRQ	HREQ/HREQ	HTRQ/HTRQ	PB14	15
HACK/HRRQ	HACK/HACK	HRRQ/HRRQ	PB15	13

Table 4 HI08 Host Request Signal Pins

En la siguiente tabla se muestra la asociación entre los pines del *puerto B* del *DSP* y los pines del *display Winstar*:

DSP conector ISA J8		**LCD 16x2**	
nº Pin J8	**Nombre**	**Nombre**	**nº Pin LCD**
39	PB0	D0	7
37	PB1	D1	8
35	PB2	D2	9
33	PB3	D3	10
31	PB4	D4	11
29	PB5	D5	12
27	PB6	D6	13
25	PB7	D7	14
21	PB9	E	6
19	PB10	R/W	5
17	PB13	RS	4
15	PB14	Base del transistor	Backlight
-	-	Vdd	1
-	-	Vss	2
-	-	Vo	3

Table 5 Conexionado entre pines del DSP y del Display

La funcionalidad *GPIO* del *puerto B* es controlada con los siguientes registros:

- **Registro HPCR (Host Port Control Register)**
  Permite configurar el *puerto B* en uno de los dos modo posibles. Para usarse como *GPIO* deben escribirse los siguientes bits.

  - Bit 0 (HGEN) = 1 Los pines de *HI08* son usados como *GPIO*.
  - Bits 1 a 6 = 0 Configuración como *GPIO*.

- **Registro HDDR (Host Data Direction Register)**
  Indica la dirección de los datos en los pines configurados como *GPIO*.

  - Los bits configurados a “1” actúan como *salida*.
  - Los bits configurados a “0” actúan como *entrada*.

- **Registro HDR (Host Data Register)**
  Es el registro donde se mantiene el valor de los datos, de los pines configurados como *GPIO*.

En resumen, para configurar la interfaz *HI08* como *GPIO* se deben seguir los siguientes pasos:

1) Bit0 *(HGEN)* del registro *HPRC* se debe establecer a "1".
2) Bits 1 a 6 del registro *HPRC* se deben poner a "0".
3) Los bits de *HDDR* configurados a "1" son *salidas*, los configurados a "0" son *entradas*.

Los datos son escritos o leídos en el registro *HDR*.

Arquitectura de Software:

La arquitectura elegida fue *Round Robin con Interrupciones*, ésta si bien es más compleja que Round Robin (usada en el laboratorio 3), permite un uso más eficiente del hardware.

Se definió una *rutina de atención a interrupciones (ISR)* que se encargara de recibir los datos desde el *codec* y los guardara en *buffers de entrada*, mientras que en la tarea principal se define que buffer llenar y cual de ellos usar para realizar las operaciones necesarias con las muestras de la señal de entrada. Otra rutina pero de menor prioridad, es la encargada de setear una bandera para que el programa principal arranque por orden del usuario al presionar el *switch 2* del kit *EVM*.

En la siguiente figura puede verse un diagrama de bloques conceptual de la aplicación:

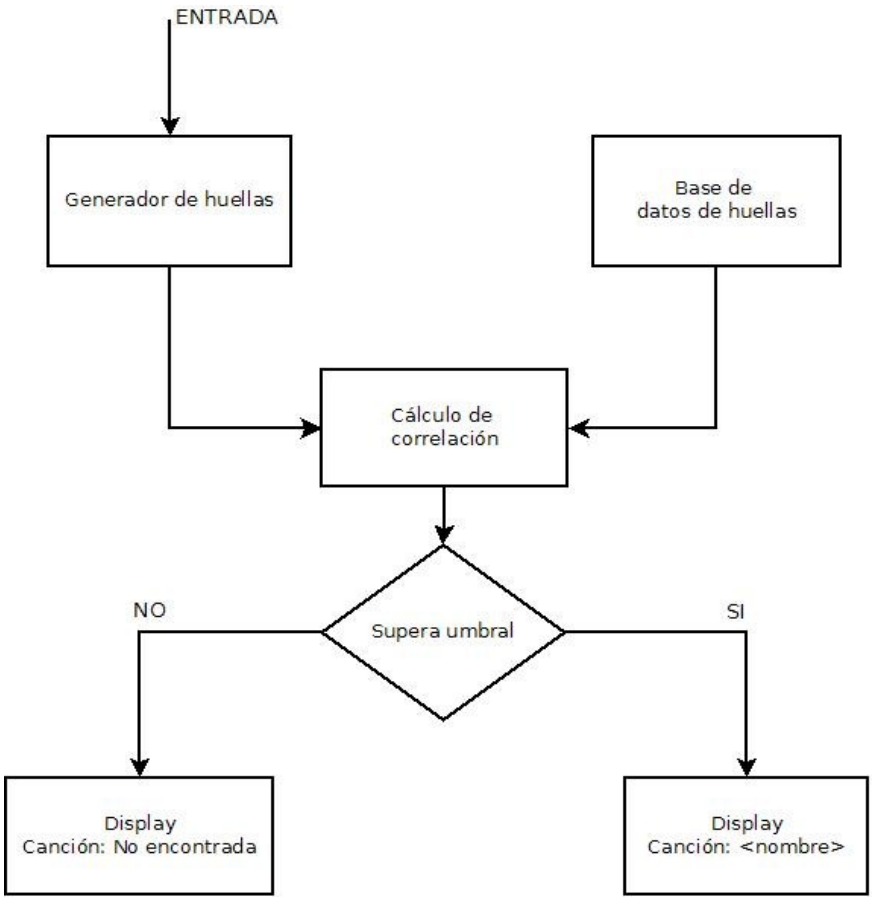

Figura 2. Diagrama de bloques de la aplicación.

Implementación

Para la implementación de este prototipo se usaron herramientas de *software libre*. El sistema operativo usado fue *Linux* (Ubuntu 9.04 y 9.10), y aplicaciones tales como *Scilab* (matemática), *Audacity* (edición de audio), *Open Office* (documentos y presentaciones) fueron empleadas durante el transcurso del proyecto. Sólo se usaron las herramientas propietarias *Tasking* y *EVM* a la hora de compilar y ejecutar la aplicación sobre el *DSP* por no existir herramientas libres que pudieran sustituirlas.

En Scilab:

Se utilizó esta aplicación con el fin de realizar simulaciones, ya que éste tiene buenas herramientas para el procesamiento de señales de audio.

La señal de audio contenida en un archivo de extensión *wav* es cargado en un *buffer de entrada* a través de la función *"wavread"*. En régimen, el programa principal llama a la función *zcr* que calcula los *cruces por cero* de una cantidad de muestras equivalentes a un tramo de *5ms* del *buffer de entrada*. Esta cantidad de *cruces por cero* se suma a los ya hallados en otros dos tramos de *5ms* los cuales se usan como *solapamiento* ente ventanas, obteniendo así los cruces en un intervalo de *15ms*. El anterior valor es dividido por *15ms* para encontrar la cantidad de cruces por segundo y luego dividido por dos (ya que un período contiene dos cruces por cero), con esto obtenemos la *frecuencia fundamental* en una ventana. Lo anterior se repite hasta barrer todo el *buffer de entrada*, momento en el cual se tiene una *huella* completa formada por un vector de *frecuencias fundamentales*.

La siguiente etapa es la de *búsqueda*, la cual se hace mediante el uso de la función *correlación*. Lo que se hace es que a través de esta función, se compare la huella generada con huellas de señales conocidas almacenadas previamente en una *base de datos*. Cuando las *huellas* están correlacionadas, esta función presenta un valor alto en el intervalo donde hay coincidencia, pudiéndose observar un pico en la gráfica de *correlación*. Sabiendo cual es el resultado de comprar una *huella* consigo misma y contra otras distinta, es que podemos definir un valor de *umbral* para decidir sobre el acierto de una *huella*.

En DSP56303:

La aplicación que controla el sistema está escrita en *lenguaje C*. Los factores principales para elegir este lenguaje son: en la primera parte del curso se ve una introducción a este lenguaje, es más simple de mantener, depurar y escalar que el lenguaje ensamblador.

La aplicación cuenta con varios módulos, entre ellos: *Codec*, *Interrupciones switch*, *Display, Algoritmo ZCR*. La tarea principal se mantiene en un *loop* infinito y hace uso de las funciones definidas en los módulos. Entraremos un poco más en detalle sobre la aplicación en lo que sigue.

La tarea principal cuenta con tres *buffers circulares de entrada* y un cuarto *buffer circular* usado como auxiliar, éste último evita que se sobreescriban datos en memoria; esto nos sucedió cuando sólo teníamos 3 *buffers* (no circulares) y la rutina de atención a la *interrupción* continuaba incrementando el puntero luego que se dejaban de usar los tres primeros. En los tres *buffers* es donde se almacenan las muestras de la señal de entrada que llegan desde el *codec* a través de una

ISR (rutina de atención a la interrupción) llamada "*ssi_receive_ls(void)*", lo cual lleva a cabo mediante un puntero. Dentro de la tarea principal se le indica al mismo a que *buffer* debe apuntar. Para explicar el llenado y procesado de los *buffers*, asumiremos que el sistema ya está en régimen, en este caso se tiene la cantidad de *cruces por cero* de la señal en cada uno de los *buffers de entrada*, y cada uno de estos valores fueron almacenados en una posición distinta de un arreglo definido a tal fin, la suma de estos tres valores permite hallar la *frecuencia fundamental* de un tramo de la señal de entrada. En cada ciclo se llena uno de los *buffers* mientras otro se procesa buscando la cantidad de *cruces por cero*, al tener este nuevo valor es posible sumarlo con los otros dos valores calculados en los dos ciclos anteriores, esto nos permite tener un solapamiento de dos tercios de su longitud total entre cada uno de los tramos de la señal.

Cuando se ha completado el arreglo de *frecuencias fundamentales* que forman la *huella* de la señal de entrada, se indica mediante una bandera que la misma ha sido generada y que debe salirse de este ciclo. El siguiente paso es realizar la búsqueda de la *huella generada* en una *base de datos* (de *huellas conocidas*), que se encuentra almacenada en memoria, esta *búsqueda* se hace usando la función de *correlación* que compara una a una hasta superar cierto *umbral de decisión*. Una vez que este *umbral* es superado la función de *búsqueda* devuelve el número de posición en la *base de datos* que coincide con la *huella generada*; este valor es el que permite direccionar el mensaje que será desplegado en el display conteniendo el nombre de *autor* y nombre de la *canción*.

Durante la ejecución del ciclo del programa se van desplegando mensajes en el *display* para indicar a el usuario que es lo que está sucediendo, logrando una interfaz amigable. Esto se consigue haciendo uso de la función *"dato_lcd(mensaje, largo)"*, a la que se le pasa como parámetros un puntero hacia el mensaje y la longitud de dicho mensaje.

Por más detalles, ver código fuente que se adjunta en el anexo.

Pruebas

Durante la primera etapa de este proyecto se definió realizar las primeras pruebas en alguna herramienta que permitiera investigar como se comportarían los algoritmos a implementar. La herramienta usada para tales efectos ha sido *Scilab*.

Entre los parámetros a definir para hacer uso de los algoritmos se encontraban:

1) *La frecuencia de muestreo*: la cual iba a depender de las señales a estudiar y de los valores que permitiera configurar el codec. Como primer valor se tomó *48kHz*.
2) *El ancho de banda*: esto está asociado con el espectro de las señales de audio y en una primera instancia pensamos en tomar el espectro que va desde *60Hz* hasta los *22kHz*.
3) *Longitud de los tramos*: la señal sería dividida en tramos de una o algunas decenas de milisegundos *(ventana)* de donde se hallaría la *frecuencia fundamental*. Este valor fue elegido en *15ms* aunque no descartamos aumentarlo a *20* o *30ms* de ser necesario. La *frecuencia fundamental* más baja que permite hallar este valor es *66,6Hz (1/15ms)*.
4) *Solapamiento de tramos*: elegimos un solapamiento que equivale a *2/3* entre un tramo y el siguiente, o sea que cada *5ms* obtendríamos un nuevo valor de *frecuencia fundamental*.
5) *Largo de la huella*: en este caso optamos por generar una huella cuyo largo fuese equivalente a *3 segundos* de la señal original. O sea que sería necesario recabar *600 muestras* de *frecuencia fundamental* f_0, *(n * 5ms = 3s ---» n = 600 muestras)*.

6) *Umbral de decisión*: este valor lo definimos en *0,9* lo que equivale a un *90%* de coincidencia entre la *huella* incógnita y alguna de las almacenadas en la *base de datos*.

Una vez asignados estos valores a los parámetros anteriores tuvimos la posibilidad de ejecutar ambos algoritmos, tanto *ZCR* como *YIN*, obteniendo *huellas* de algunas señales como ser la de la canción *"Only Time"* de Enya, *"Headshrinker"* de Oasis y *"Eruption"* de Van Halen. Las gráficas de las *huellas* anteriores nos permitieron observar como era la variación de *frecuencia fundamental* en el tiempo y cual era su valor máximo, éste último de importancia a la hora de definir el *ancho de banda* que usaríamos en la aplicación con el *DSP*.

Es de destacar que las pruebas sobre *Scilab* tenían tiempos de ejecución del orden del minuto, lo cual a priori puede llevar a pensar que no funcionarían bien sobre el *DSP*, pero no hay que perder de vista que en los *DSP* las operaciones están optimizadas a tal punto que es posible cargar factores desde memoria en paralelo, multiplicar y sumar, todo en un solo ciclo de máquina.

El siguiente paso sobre *Scilab* fue probar un algoritmo de *búsqueda*, el cual se basa en hacer uso de la función de *correlación*, con una pequeña variante para que esté normalizada. De esta manera pretendíamos poder definir el valor del *umbral de decisión*, que evitara *falsos positivos* o *falsos negativos* en el resultado de una *búsqueda*. Si el resultado de la *correlación normalizada* era igual o superior a *0,9* se daba por encontrada la *huella* buscada.

Previo a pasarnos a realizar pruebas sobre el *DSP*, era necesario contar con archivos de audio en formato *wav* desde donde se obtendrían las señales que luego se inyectarían en el *DSP*. Definimos que estas señales serían *Mono*, ya que solo usamos uno de los canales del *codec* (el izquierdo), aunque sería bueno trabajar con señales *Stereo* (ambos canales), ya que la información que contiene cada uno de ellos es distinta y sería una forma de hacer un segundo control cuando sólo se llega al mínimo del umbral de decisión aceptado con solo un canal, evitando *falsos positivos*.

Como el tamaño en muestras de estos archivos *wav* era equivalente a tres o cuatro minutos de *canción*, decidimos recortarlos a un equivalente de *ocho segundos* el cual por lo general contiene el *estribillo de la canción* (en el caso de la canción elegida para Van Halen no hay *estribillo*). De estos fragmentos de *canción* es de donde se generaron las *huellas* que formaron parte de la *base de datos*, cada una de ellas equivalente a seis de los ocho segundos de duración de audio. El proceso para crear estas *huellas* era inyectar la señal de ocho segundos de duración a la entrada del *codec* y producir una *huella* de largo equivalente a *seis segundos*, que era posible extraer desde la memoria del *DSP*, mediante el comando "*SAVE x:$------ $------ «nombre_archivo.LOD» -f*" de la aplicación *EVM* de *Motorola*. Esto producía un archivo de texto que contenía un vector de frecuencias que formaban una *huella*, la cual luego la transformamos en un arreglo en *lenguaje C* con el que creamos la *base de datos de huellas*.

Una vez que la *base de datos* estaba lista era momento de comenzar a realizar pruebas, generando *huellas* en "tiempo real" y efectuando la *búsqueda* en la *base de datos*.

Comenzaremos describiendo los valores que efectivamente le asignamos a los parámetros que utilizamos en los algoritmos.

De las pruebas realizadas en *Scilab* pudimos observar que en ninguna de las *huellas generadas* con esta aplicación se superaban valores de *frecuencia fundamental* de *8kHz*, por esta razón definimos esta frecuencia como la máxima que queremos detectar, el *Ancho de Banda* es entonces de *8kHz*. A partir del anterior valor, la *frecuencia de muestreo* quedó establecida en *16kHz*.

La *longitud* de cada *tramo* o *ancho de ventana* quedó definido en *30ms*, y el *solapamiento* mantiene la proporción de *2/3*, o sea que cada nuevo tramo coincide con los últimos *20ms* del tramo anterior, teniendo cada *10ms* un nuevo valor de *frecuencia fundamental*. Los anteriores valores terminan por definir el tamaño en muestras de cada buffer de entrada, esto es, *10ms * 16kHz = 160 muestras*.

El *largo de la huella* de la señal incógnita a la entrada del sistema se genera de un largo de *200 muestras*, que equivalen a *2 segundos* de la señal a la entrada *(n * 10ms = 2s ---» n = 200 muestras de f_0)*.

Por último el valor del *umbral* de decisión lo establecimos en *0,7*, a priori puede parecer un poco bajo, pero es bastante mayor que los valores que resultaron de efectuar la *correlación* de una *huella* con las que se encuentran en la *base de datos* y no se produce coincidencia, este valor ronda en el mejor de los casos en *0,4*, para la reducida base que pudimos probar (ocho canciones). Cuando la *huella* incógnita coincide con alguna de las almacenadas en la *base de datos*, el valor de *correlación* es próximo a *1*. Este valor de *umbral* puede evitar *falsos negativos* en presencia de micro cortes en la señal de entrada, por ej., en el caso de la señal de Soda Stereo produjimos dos (y hasta tres) cortes de *50ms* de duración dentro de un intervalo de *dos segundos* de *canción* y la inyectamos a la entrada del *codec*, la *huella* que se producía con ese intervalo se podía detectar de todas formas, evitando un *falso negativo*.

La *base de datos* de *huellas* pertenecen a ocho canciones de diferentes géneros, como ser, música *clásica, folclore, rock, new age y murga*. Sus nombres y autores o interpretes se detallan a continuación:

1) *"Only Time"*, Enya.
2) *"Headshrinker"*, Oasis.
3) *"Eruption"*, Van Halen.
4) *"Los dos Gallos"*, Los Olimareños.
5) *"A Jose Artigas"*, Alfredo Zitarrosa.
6) *"Cuple IMM"*, Curtidores de Hongos.
7) *"Marcha Turca"*, Mozart.
8) *"De Musica Ligera"*, Soda Stereo.

Si bien el tamaño de la base es muy pequeño, aproximadamente *5Kbytes* de datos, tuvimos el inconveniente de no poder cargarla toda en memoria, ya que por defecto el kit *EVM* utiliza la memoria interna del *DSP* y no nos fue posible configurar la memoria extendida de *32Kbytes* mediante el comando *"CHANGE"* del software *EVM*. Esto hizo que sólo pudiéramos cargar de a dos *huellas* en memoria, lo que hace que se reduzca aún más, nuestro pequeño universo de *búsqueda*. En ninguna de las pruebas que efectuamos obtuvimos falsos positivos.

Conclusiones

La principal cosa a destacar es que el *algoritmo ZCR* implementado en *lenguaje C* para ser ejecutado desde el *DSP* funcionó mejor de lo esperado. Tal vez nuestra falta de expectativa en este algoritmo se debió a lo estudiado en la literatura empleada.

El valor seleccionado para los parámetros empleados en el algoritmo parecen ser los correctos luego de realizadas las pruebas, aunque debemos ser cautos con el valor del *umbral* de decisión ya que el tamaño de la *base de datos* utilizada no es representativo del universo de canciones que existen en el mundo; lo que hace que no hayamos tenido *falsos positivos*.

En lo que concierne al tamaño de la *huellas*, generarlas de longitud equivalente mayor a tres segundos es recabar información innecesaria, mientras que generarlas de longitud menor a un segundo puede ser insuficiente información. Una *longitud de huella* equivalente a *dos segundos* de audio parece ser una medida óptima.

Referido al *DSP56303*, podemos decir que tiene potencia de cálculo suficiente para ejecutar la aplicación que hace uso del *algoritmo ZCR*, aunque sería bueno contar con más memoria para poder hacer uso de una *base de datos* más grande. Ésto en parte hubiese sido posible si hubiésemos logrado configurar la memoria externa de *32Kbytes* con los que cuenta el kit de desarrollo.

Por último, podemos decir que logramos alcanzar casi en su totalidad los cometidos definidos durante la especificación del proyecto, a excepción de la implementación del *algoritmo YIN* directamente en el *DSP*.

Anexo

Planificación del Proyecto

En el comienzo del proyecto se realizó un cronograma tentativo de los puntos más importantes que se debían llevar a cabo. Por diversas razones nos vimos obligados a cambiar algunos puntos durante el transcurso del mismo.

El retraso más importante se debió a la falta de herramientas adecuadas de desarrollo, las mismas son propietarias, y debido a la falta de licencia nos vimos obligados a usar una versión trial por *15 días* ya que no era posible el uso de la demo estudiantil (misma que se usa en los laboratorios), ya que no permitía compilar los módulos implementados en *C*. La restricción de *15 días* fue resuelta instalando una *máquina virtual de Windows* (a la cual se le instaló la herramienta de desarrollo) sobre un PC con *Linux*, y de la que se guardaba su estado.

Cronograma:

Nombre	fecha	Descripción
Entrega prop. definitiva	**08/10/09**	
Algoritmo ZCR	12/10/09	Implementación y pruebas en Scilab
Algoritmo YIN	18/10/09	Implementación y pruebas en Scilab
Depurado	22/10/09	Prueba de huellas creadas con los algoritmos (previo a DSP)
1º Informe de avance	**02/11/09**	
Código C	09/11/09	Pasaje a código C para uso en el DSP
Pruebas en DSP	16/11/09	Búsqueda de las huellas en el DSP
2º Informe de avance	**16/11/09**	
Correcciones	18/11/09	Depurado y corrección de errores
Redacción de info final	20/11/09	
Entrega de documentación	23/11/09	
Presentación	**30/11/09**	Defensa oral

Cronograma corregido:

Nombre	fecha	Descripción
Algoritmo ZCR	12/10/09	Implementación y pruebas en Scilab
Algoritmo YIN	18/10/09	Implementación y pruebas en Scilab
Presentación	**20/10/09**	Presentación del proyecto
Depurado	30/10/09	Prueba de huellas creadas con los algoritmos (previo a DSP)
Código C	04/11/09	Pasaje a código C para uso en el DSP
Pruebas en DSP	04/12/09	Búsqueda de las huellas en el DSP
Correcciones y pruebas	07/12/09	Depurado y corrección de errores
Redacción documentación	20/12/09	
Página web	20/12/09	
Presentación	**22/12/09**	Defensa oral

Bibliografía

- *Digital Processing of Speech Signals*.
  Autores: L.R. Rabiner / R.W. Schafer

- *Separation of voiced and unvoiced using zero crossing rate and energy of the speech signal*.
  Autores: Kopparthi, Adapa, Barkana.
  Descripción: Método que utiliza los cruces por cero y la energía de la señal para poder distinguir el umbral de decisión sonoro-sordo.

- *YIN, a fundamental frequency estimator for speech and music.*
  Autores: Alain de Cheveigne, Hideki Kawahara.
  Descripción:

- *A Review of Audio Fingerprinting*.
  Autores: Pedro Cano, Eloi Batlle, Ton Kalker, Jaap Haitsma.

- *Construcción de la prótesis táctil "Pauper Tango".*
  Autores: Rozo, Perry, Villegas.
  Descripción: Extracción de frecuencia fundamental utilizando correlación.

- *A Highly Robust Audio Fingerprinting System*.
  Autores: Ton Kalker, Jaap Haitsma.

- *Audio Fingerprinting. "A New Technology To Identify Music*
  Autor: J. A. Haitsma.

- *Pitch Extraction and Fundamental Frequency: History and Current Techniques.*
  Autor: David Gerhard.

- Manuales Motorola:
  - DSP56303EVMUM/AD
  - DSP56303UM/AD
  - DSP56300FMO

- *El lenguaje de Programación C*.
  Autor: Brian W. Kernighan.

Software utilizado:

- *Kicad*
  Descripción: Herramienta para el diseño de circuitos impresos.
  Sitio web: www.lis.inpg.fr/realise_au_lis/kicad

- *Dia*
  Descripción: Herramienta para el diseño de diagramas.
  Sitio web: projects.gnome.org/dia/

- *Open Office*
  Descripción: Editor de texto, hojas de cálculo, diapositivas y mucho más.
  Sitio web: http://es.openoffice.org/

- *Subversion*
  Descripción: Herramienta que permite llevar un control de versiones del programa.
  Sitio web: subversion.tigris.org/

- *Notepad++*
  Descripción: Editor de código fuente.
  Sitio web: notepad-plus.sourceforge.net/es/site.htm

- *Audacity*
  Descripción: Excelente editor de audio.
  Sitio web: audacity.sourceforge.net/

- Scilab
  Descripción: Herramienta para efectuar cálculo numérico.
  Sitio web: www.scilab.org/

- Tasking EDE
  Descripción: Compilador para la arquitectura de DSP's Motorola.
  Sitio web: www.tasking.com/resources/technologies/en/ede.cfm

- EVM
  Descripción: Interfaz entre el DSP Motorola y el PC.
  Sitio web: iie.fing.edu.uy/cursos/mod/resource/view.php?id=992

Agradecimientos

A nuestras familias, por su apoyo incondicional.

Archivos fuentes

Módulos:

Codec:	*3xx_4215.c, cs4215.h, loopback.h.*
Interrupciones switch:	*int.c, int.h.*
Display:	*lcd16x2.c, lcd16x2.h.*
Algoritmo ZCR:	*zcr_func.c, zcr_func.h, bdh.h*
Tarea principal:	*main.c*

Codec:

3xx_4215.c:

```
/************************************************************************
**
**  VERSION:          @(#)3xx_4215.c  1.10    02/04/25
**
**  IN PACKAGE:       DSP563xx/6xx fftscope
**
**  COPYRIGHT:        Copyright 1997-2004 Altium BV
**
**  DESCRIPTION:
**
**     Routines for CS4215 codec connected to DSP563xx ESSI0.
**     The codec initialisation routine is the C equivalent
**     of the routines supplied by Motorola.
**
************************************************************************/

//#include <c56.h>
//#include <reg56303.h>

#include "cs4215.h"
#include "loopback.h"

#ifndef CPUCLK
#define CPUCLK      67000000L   /* Hz */
#endif

#if FSAMPLE >= 48000          /* select appropriate sampling freq.  */
#define FS_VALUE SAMP_RATE_48
#elif FSAMPLE >= 32000
#define FS_VALUE SAMP_RATE_32
#elif FSAMPLE >= 27000
#define FS_VALUE SAMP_RATE_27
#elif FSAMPLE >= 16000
#define FS_VALUE SAMP_RATE_16
#elif FSAMPLE >= 9000
#define FS_VALUE SAMP_RATE_9
#elif FSAMPLE >= 8000
#define FS_VALUE SAMP_RATE_8
#endif
```

```
#define CTRL_WD_12  (NO_PREAMP+HI_PASS_FILT+FS_VALUE+STEREO+DATA_16)
#define CTRL_WD_34  (IMMED_3STATE+XTAL1_SELECT+BITS_64+CODEC_MASTER)
#define CTRL_WD_56  0x000000
#define CTRL_WD_78  0x000000

#define TONE_INPUT      MIC_IN_SELECT+(15*MONITOR_ATTN)
#define TONE_OUTPUT     HEADPHONE_EN+LINEOUT_EN+(0*LEFT_ATTN)+(0*RIGHT_ATTN)

#define setvect(n, v)   *((_P unsigned short*)(n*2)) = (0x0d0000+(unsigned short)(v)); \
                        *((_P unsigned short*)(n*2+1)) = 0

_Y _circ CS4215DATA txdata, rxdata;
_Y _circ _fract *txdata_p = txdata.b;
_Y _circ _fract *rxdata_p = rxdata.b;

_fract _circ *ptr_buffer;

/*-------------------------------------------------------------------------
 *  PURPOSE: Handle SSI receive (A/D converter) interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(IRQ_ESSI0_RCV)
ssi_receive(void)
{
   *rxdata_p++ = RX0;
}


/*-------------------------------------------------------------------------
 *  PURPOSE: Handle SSI receive with exception (A/D converter) interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(IRQ_ESSI0_RCE)
ssi_receive_err(void)
{
   SSISR0.B.ROE = 0;
   *rxdata_p++ = RX0;
}


/*-------------------------------------------------------------------------
 *  PURPOSE: Handle SSI receive last slot interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(IRQ_ESSI0_RCL)
ssi_receive_ls(void)
{
   rxdata_p = rxdata.b;
          *ptr_buffer++ = rxdata.d.i.audio_left;
}
```

```
/*-----------------------------------------------------------------------------
 *  PURPOSE: Handle SSI transmit (D/A converter) interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-----------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(IRQ_ESSI0_TRM)
ssi_transmit(void)
{
   TX00 = *txdata_p++;
}



/*-----------------------------------------------------------------------------
 *  PURPOSE: Handle SSI transmit with exception (D/A converter) interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-----------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(IRQ_ESSI0_TRE)
ssi_transmit_err(void)
{
   SSISR0.B.TUE = 0;
   TX00 = *txdata_p++;
}


/*-----------------------------------------------------------------------------
 *  PURPOSE: Handle SSI transmit last slot interrupt during initialisation
 *           (transmitted data is not retrieved from the output buffers).
 *  INPUT  : -
 *  OUTPUT : -
 *-----------------------------------------------------------------------------*/
static void _internal _reentrant _long_interrupt(IRQ_ESSI0_TRL)
ssi_transmit_ls_init(void)
{
   txdata_p = txdata.b;
}


/*-----------------------------------------------------------------------------
 *  PURPOSE: Handle SSI transmit last slot interrupt.
 *  INPUT  : -
 *  OUTPUT : -
 *-----------------------------------------------------------------------------*/
void _near _internal _reentrant _long_interrupt(-1)
ssi_transmit_ls(void)
{
   txdata_p = txdata.b;
}


/*-----------------------------------------------------------------------------
 *  PURPOSE: Handle SSI receive last slot interrupt.
 *  INPUT  : -
 *  OUTPUT : -
```

```
 *-------------------------------------------------------------------------*/
static void _near _internal _reentrant _long_interrupt(-1)
ssi_receive_ls_dummy(void)
{
   rxdata_p = rxdata.b;
}


/*-------------------------------------------------------------------------
 *  PURPOSE: Send input gain settings to codec.
 *  INPUT  : -
 *  OUTPUT : -
 *-------------------------------------------------------------------------*/
void ssi_set_gains(void)
{
   int i, gain;

   for(i = 0;  i < IN_MAX;  i++)
   {
#if 0
      gain = chan[i].params.gain * 2 / 3; /* 1.5 dB/step   */
#else
                        gain = 0;
#endif
      if(gain <  0)  gain = 0;          /* CS4215 limits */
      if(gain > 15)  gain = 15;
      if(i == IN_LEFT)
         txdata.d.b.lgain = gain;
      else
         txdata.d.b.rgain = gain;
   }
}



#define CRA0_VALUE   0x103800 + ((CPUCLK/(2*2500000)) - 1)
#define CRB0_CVALUE  0xFF313C /* value for control mode */
#define CRB0_DVALUE  0xFF310C /* value for data    mode */
#define PRRC_VALUE   0x000003
#define PCRC_VALUE   0x00003C
#define PDRC_NRESET  0x000001
#define PDRC_DATA    0x000002
#define IPRP_SSIMASK 0x00000C

/*-------------------------------------------------------------------------
 *  PURPOSE: Initialize (interrupt driven) ESSI interface between codec and DSP.
 *  INPUT  : -
 *  OUTPUT : -
 *-------------------------------------------------------------------------*/
void ssi_init(void)
{
      int i, j;

   /* First, the SSI port is programmed as master to be able to write    */
   /* to the CS4215's control registers.                                 */
```

```
/* 'Sampling frequency' here is 2 MHz/(4*16) = 39 kHz (immaterial).   */
/* Switch to control mode and reset the CS4215.                       */

PCRC   = 0x0000;        /* Turn off SSI port */
CRA0.I = CRA0_VALUE;    /* (CPUCLK/4)/N = 2MHz SCLK, WL=16 bits, 4W/F */
CRB0.I = (int)CRB0_CVALUE; /* RIE,TIE,RE,TE, NTWK, SYN, FSR/RSR->bit  */
PRRC   = PRRC_VALUE;    /* setup pc2, pc3 and pc4 as outputs          */
PDRC   = 0x0000;        /* D/C~ = 0, RESET~ = 0 ==> reset mode        */

/* Allow for power supply stabilization before codec calibration.     */

for(i = 0;  i < 1000;  i++)            /* 100 ms total delay:        */
{
   for(j = 0;  j < (int)(100*CPUCLK/(2*1E6));  j++)     /* 100 us    */
   {
      _nop();
   }
}

txdata.c.i.ctrl_wd_12 = CTRL_WD_12;
txdata.c.i.ctrl_wd_34 = CTRL_WD_34;
txdata.c.i.ctrl_wd_56 = CTRL_WD_56;
txdata.c.i.ctrl_wd_78 = CTRL_WD_78;

setvect(IRQ_ESSI0_TRL, ssi_transmit_ls_init);

PDRC    = PDRC_NRESET;   /* D/C~ = 0, RESET~ = 1 ==> control mode     */
IPRP.I |= IPRP_SSIMASK;  /* set SSI interrupt priority level to 3     */

enable_interrupts();

PCRC = PCRC_VALUE;          /* Turn on SSI port                       */

/* The CS4215's control register are now written from the interrupt   */
/* routine ssi_trm_init() with the control word values above.         */

while(!SSISR0.B.RFS)  ;     /* wait for synchronization               */
while( SSISR0.B.RFS)  ;     /* wait for one complete buffer to be     */
while(!SSISR0.B.RFS)  ;     /* written to the CS4215                  */

while(rxdata.c.b.clb)  ;    /* wait for received clb to go low        */
                            /*  indicating data has been accepted     */

txdata.c.b.clb = 1;         /* set clb to finalize transfer           */

for(i = 0;  i < 4;  i++)    /* wait four transmit cycles              */
{   while(!SSISR0.B.TFS)  ;
    while( SSISR0.B.TFS)  ;
}

/* Now switch the SSI port to slave mode. The CS4215 will drive the   */
/* SCLK and FS lines from now on. Sampling frequency is determined by */
/* the clock source of the CS4215 and the divide ratio selected.      */

PCRC   = 0x0000;            /* Turn off SSI port                      */
```

```
   CRA0.I = CRA0_VALUE;       /* (CPUCLK/4)/N = 2MHz SCLK, WL=16, 4W/F  */
   CRB0.I = (int)CRB0_DVALUE;  /* rx,tx & int ena,netwk,syn,sclk==i/p,msb 1st*/
   PDRC   = PDRC_DATA|PDRC_NRESET;/* D/C~ = 1  ==> data mode            */

   txdata.d.i.audio_left     = 0;
   txdata.d.i.audio_right    = 0;
   txdata.d.i.output_setting = TONE_OUTPUT;
   txdata.d.i.input_setting  = TONE_INPUT;
   ssi_set_gains();

   setvect(IRQ_ESSI0_TRL, ssi_transmit_ls);

   PCRC = PCRC_VALUE;         /* Turn on SSI port                    */

}


/*---------------------------------------------------------------------------
 *  PURPOSE: Change interrupt handler for receive-last-frame interrupts.
 *  INPUT  : Flag: 1 = run mode, 0 = init mode.
 *  OUTPUT : -
 *---------------------------------------------------------------------------*/
void ssi_enable_receive(int flag)
{
   setvect(IRQ_ESSI0_RCL, flag ? ssi_receive_ls : ssi_receive_ls_dummy );
}


/*---------------------------------------------------------------------------
 *  PURPOSE: Toggle an I/O line for timing measurements.
 *  INPUT  : -
 *  OUTPUT : -
 *---------------------------------------------------------------------------*/
void ssi_toggle_indicator(void)
{
   TCSR0.B.TDIR = 1;           /* set to 'output'                     */
   TCSR0.B.TDO ^= 1;           /* toggle TIO0, LED next to audio connector  */
}

/* END OF FILE */
```

cs4215.h:

```
/*****************************************************************************
**
**  VERSION:          @(#)cs4215.h    1.6     02/04/25
**
**  IN PACKAGE:       DSP563xx/6xx fftscope
**
**  COPYRIGHT:        Copyright 1997-2004 Altium BV
**
**  DESCRIPTION:      Defines for CS4215 codec.
**
*****************************************************************************/

#ifndef _CS4215_H
#define _CS4215_H

typedef union
{   union
   {   struct
      {   int           : 8; /* CTRL_WD_12 in most significant 16 bits */
         unsigned df    : 2;
         int     st     : 1;
         unsigned dfr   : 3;
         int            : 1;
         int  hpf       : 1;
         int            : 2;
         int  clb       : 1;
         int  olb       : 1;
         int  mlb       : 1;
         int            : 3;

         int            : 8; /* CTRL_WD_34 in most significant 16 bits */
         int  dad       : 1;
         int     enl    : 1;
         unsigned test  : 6;
         int  xen       : 1;
         int  xclk      : 1;
         unsigned bsel  : 2;
         unsigned mck   : 3;
         int  its       : 1;

         int            : 8; /* CTRL_WD_56 in most significant 16 bits */
         int            : 8;
         int            : 6;
         int     pio0   : 1;
         int     pio1   : 1;

         int            : 8; /* CTRL_WD_78 in most significant 16 bits */
         int            : 8;
         unsigned ver   : 4;
```

```
            int             : 4;
        } b;
        struct
        {   unsigned int ctrl_wd_12;
            unsigned int ctrl_wd_34;
            unsigned int ctrl_wd_56;
            unsigned int ctrl_wd_78;
        } i;
    } c;    /* control mode */
    union
    {   struct
        {   _fract   audio_left;
            _fract   audio_right;

            int             : 8; /* output_setting in most significant 16 bits */
            unsigned ro     : 6;
            int  se         : 1;
            int  adi        : 1;
            unsigned lo     : 6;
            int  le         : 1;
            int  he         : 1;

            int             : 8; /* input_setting in most significant 16 bits */
            unsigned rgain  : 4;
            unsigned matt   : 4;
            unsigned lgain  : 4;
            int  is         : 1;
            int  ovr        : 1;
            int  pio0       : 1;
            int  pio1       : 1;
        } b;
        struct
        {   _fract  audio_left;
            _fract  audio_right;
            unsigned int output_setting;
            unsigned int input_setting;
        } i;
    } d;    /* data mode */
    _fract b[4];    /* (circular) buffer */
} CS4215DATA;

#define NO_PREAMP       0x100000
#define LO_OUT_DRV      0x080000
#define HI_PASS_FILT    0x008000
#define SAMP_RATE_9     0x003800
#define SAMP_RATE_48    0x003000
#define SAMP_RATE_32    0x001800
#define SAMP_RATE_27    0x001000
#define SAMP_RATE_16    0x000800
#define SAMP_RATE_8     0x000000
#define STEREO          0x000400
#define DATA_8LIN       0x200300
#define DATA_8ALAW      0x200200
#define DATA_8ULAW      0x200100
#define DATA_16         0x200000
```

```
#define IMMED_3STATE    0x800000
#define XTAL1_SELECT    0x100000 /* 24.576  MHz */
#define XTAL2_SELECT    0x200000 /* 16.9344 MHz */
#define BITS_64         0x000000
#define BITS_128        0x040000
#define BITS_256        0x080000
#define CODEC_MASTER    0x020000
#define CODEC_TX_OFF    0x010000

#define HEADPHONE_EN    0x800000
#define LINEOUT_EN      0x400000
#define LEFT_ATTN       0x010000 /* 63*LEFT_ATTN   = -94.5 dB, 1.5 dB steps */
#define SPEAKER_EN      0x004000
#define RIGHT_ATTN      0x000100 /* 63*RIGHT_ATTN  = -94.5 dB, 1.5 dB steps */
#define MIC_IN_SELECT   0x100000
#define LEFT_GAIN       0x010000 /* 15*LEFT_GAIN    = 22.5 dB, 1.5 dB steps */
#define MONITOR_ATTN    0x001000 /* 15*MONITOR_ATTN = mute,    6   dB steps */
#define RIGHT_GAIN      0x000100 /* 15*RIGHT_GAIN   = 22.5 dB, 1.5 dB steps */

#endif /* _CS4215_H */


/* END OF FILE */
```

loopback.h:

```
/************************************************************************
 * funciones del CODEC usadas por el programa principal
 ************************************************************************/
#ifndef LOOPBACK_H
#define LOOPBACK_H

#include <c56.h>
#include <reg56303.h>


#define OUTPOLARITY (-1)     /* invert audio output (56xxxEVM HDPHNE output)  */
#define FSAMPLE     16000    /* sampling frequency, Hz   */

#define ON  1
#define OFF 0

#define disable_interrupts() _asm("ori   #$03,mr")
#define enable_interrupts()  _asm("andi  #$FC,mr")

typedef enum    { IN_LEFT,  IN_RIGHT,  IN_MAX  }        inbuf_t;
typedef enum    { OUT_LEFT, OUT_RIGHT, OUT_MAX }        outbuf_t;

void ssi_init(void);
void ssi_set_gains(void);
void ssi_enable_receive(int flag);
void ssi_toggle_indicator(void);
void ssi_loopback(void);
```

```
#endif /*LOOPBACK_H*/


/* END OF FILE */
```

Interrupciones switch:

int.c:

```
#include "int.h"
short sw2;

/ ************************************************************************************************ */
/* ************* Configura el Timer 0 como GPIO para usar los switch SW2, SW3 y el led D12 *************** */
/* ************************************************************************************************ */
void init_sw(void)
{
        /* configuracion de prioridades de interrupcion */
        IPRC.B.IAL =1;
        IPRC.B.IDL =1;
        /* configuracion de timer como GPIO */
        TCSR0.B.TEN   =0;       /* deshabilita el timer                 */
        TCSR0.B.TCTRL =0;       /* modo 0 GPIO                          */
        TCSR0.B.TINV  =0;       /* apaga el inversor a la salida        */
        TCSR0.B.TDIR  =1;       /* configura DO como salida             */
        TCSR0.B.TDO   =0;       /* salida a cero (led D12 apagado)      */
        /* el resto quiero que valga cero (no me importan) */
        TCSR0.B.TOIE =0;
        TCSR0.B.TCIE =0;
        TCSR0.B.TRM    =0;
        TCSR0.B.TDI    =0;
        TCSR0.B.TPCE =0;
        TCSR0.B.TOF    =0;
        TCSR0.B.TCF    =0;
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ****************** ISR para manejar las interrupciones del switch SW2 de la placa EVM ***************** */
/ ************************************************************************************************ */
void _long_interrupt(IRQ_A) switch2(void)
{
        /* se apreto el SW2 */
        TCSR0.B.TDO =ON;                /* led D12 ON */
        sw2 = 1;
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ****************** ISR para manejar las interrupciones del switch SW3 de la placa EVM ***************** */
/ ************************************************************************************************ */
/* void _fast_interrupt(IRQ_D) switch3(void)
{
        //se apreto el SW3
```

```
        TCSR0.B.TDO =OFF;               //led D12 OFF
}  */
/ ********************************************************************************************** */
/* END OF FILE */
```

int.h:

```
#ifndef INT_H
#define INT_H

#include <reg56303.h>
#define ON 1
#define OFF 0

/* ************* Configura el Timer 0 como GPIO para usar los switch SW2, SW3 y el led D12 ************** */
void init_sw(void);

/* ****************** ISR para manejar las interrupciones del switch SW2 de la placa EVM ***************** */
void _long_interrupt(IRQ_A) switch2(void);

/* ****************** ISR para manejar las interrupciones del switch SW3 de la placa EVM ***************** */
/* void _fast_interrupt(IRQ_D) switch3(void); */

#endif /*INT_H*/

/* END OF FILE */
```

Display:

lcd16x2.c:

```
#include "lcd16x2.h"

/ ************************************************************************************************ */
/* ************************* Inicializa el puerto B(HI08) en modo GPIO ****************************** */
/ ************************************************************************************************ */

void init_gpio(void)
{
        HPCR.B.HGEN  = 1;               /* 0: Host GPIO Port Enable      */

        HPCR.B.HA8EN = 0;               /* 1: Host Address Line 8 Enable */
        HPCR.B.HA9EN = 0;               /* 2: Host Address Line 9 Enable */
        HPCR.B.HCSEN = 0;               /* 3: Host Chip Select Enable    */
        HPCR.B.HREN  = 0;               /* 4: Host Request Enable        */
        HPCR.B.HAEN  = 0;               /* 5: Host Acknowledge Enable    */
        HPCR.B.HEN   = 0;               /* 6: Host Enable                */


        HDDR = 0xFFFF;                          /* Configuro bits de I/O como salidas */
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ********************** Escribe un cartacter en la memoria del display ********************************* */
/ ************************************************************************************************ */

void write_lcd(unsigned char simbolo, short ctrl_dat)
{
        HDR = simbolo;                          /* caracter a escribir en el lcd */
        HDR |= ctrl_dat;                /* palabra de control */
        HDR &= 0xFDFF;                                  /* flanco de bajada(E=0) */
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ***************************** Lee un dato o el estado desde el display ***************************** */
/ ************************************************************************************************ */
/*falta probar que funcione*/
//char read_lcd(char ctrl_dat)
//{
//        REGISTRO_HDR dato;

//        HDDR = 0xFF00;                                  /*Configuro 8 bits de datos de I/O como entradas*/
//        dato.bytes.byte1 = ctrl_dat; /*R/!W=1; E=1; RS= 0(ctrl), 1(dato); BL encendido*/
```

```
//          HDR = dato.I;
//          dato.bits.bit1 = 0;             /*flanco de bajada(E=0)*/
//          HDR = dato.I;
//          dato.I = HDR;                                   /*lee dato desde lcd*/

//          HDDR = 0xFFFF;                                          /*Configuro bits de I/O como salidas*/

//          return dato.bytes.byte0;
//}
/ ********************************************************************************************** */


/ ********************************************************************************************** */
/* ************************** Produce un retardo proporcional a 100us ***************************** */
/ ********************************************************************************************** */

void delay(int factor)
{
          int i, j;

          for(i = 0;  i < factor;  i++)     /* factor*100us = total delay: */
   {
      for(j = 0;  j < (int)(100*CPUCLK/(2*1E6));  j++)   /* 100 us */
      {
         _nop();
      }
   }
}
/ ********************************************************************************************** */


/ ********************************************************************************************** */
/* ********************** Inicializa el display LCD16x2, pag.45 HD44780U.pdf ************************** */
/ ********************************************************************************************** */

void init_lcd(void)
{
          delay(150);                                             /* esperar 15ms */
          write_lcd(FUN_SET, CTRL_WR);                  /* comando Function set */
          delay(50);                                              /* esperar 5ms */
          write_lcd(FUN_SET, CTRL_WR);                  /* comando Function set */
          delay(1);                                               /* esperar 100us */
          write_lcd(FUN_SET, CTRL_WR);                  /* comando Function set */
          delay(1);                                               /* esperar 100us */
          write_lcd(FUN_SET, CTRL_WR);                  /* comando Function set */
          delay(1);                                               /* esperar 100us */
          write_lcd(DPLY_OFF, CTRL_WR);                 /* comando Dply off */
          delay(1);                                               /* esperar 100us */
          write_lcd(CLEAR, CTRL_WR);                    /* comando Clear Dply */
          delay(100);                                   /* esperar 10ms */
          write_lcd(ETY_MOD_SET, CTRL_WR);      /* comando Entry mode set */
          delay(1);                                                       /* esperar 100us */
          write_lcd(DPLY_ON, CTRL_WR);                  /* comando Dply on */
          delay(1);                                               /* esperar 100us */
}
```

```
/ ********************************************************************************************** */


/ ********************************************************************************************** */
/* ****************************** Verifica si el display esta ocupado  ****************************** */
/ ********************************************************************************************** */
/*falta probar que funcione para que sustituya a la funcion delay()*/
//int busy(void)
//{
//        int bf = 1;
//        int i = 0;
//        char status;
//        status = read_lcd(CTRL_RD);
//        status = status & 0x80;                                                 /*me deja todo a 0 salvo el
DB7 = BF*/
//        if (status == 0x00)
//                return bf = 0;
//        else
//                {
//                        while ((status != 0x00) || (i == 50000))        /*me quedo mientras este ocupado el lcd
evitando el loop*/
//                                {
//                                        status = read_lcd(CTRL_RD);
//                                        status = status & 0x80;                 /*me deja todo a 0 salvo el
DB7 = BF*/
//                                        i++;
//                                }
//                        if (status == 0x00)
//                                return bf = 0;
//                        return bf;
//                }
//}
/ ********************************************************************************************** */


/ ********************************************************************************************** */
/* ***************** Envia los datos que se escribiran en el display y la longitud de los mismos ***************** */
/ ********************************************************************************************** */

void dato_lcd(_Y unsigned char *dato, short len)
{
        short i;

        if(len <= LCD_16)
        {
                for(i=0; i<len; i++)
                {
                        write_lcd(dato[i], DATO_WR);
                        delay(1);                                               /* esperar 100us */
                }
        }
        else
        {
                for(i=0; i<LCD_16; i++)
```

```
                {
                        write_lcd(dato[i], DATO_WR);
                        delay(1);                                   /* esperar 100us */
                }
                write_lcd(SDA_LIN, CTRL_WR);
                delay(1);                                           /* esperar 100us */
                for(i=LCD_16; i<len; i++)
                {
                        write_lcd(dato[i], DATO_WR);
                        delay(1);                                   /* esperar 100us */
                }
        }
}
/ ************************************************************************************************ */
/* END OF FILE */
```

lcd16x2.h:

```
#ifndef LCD16x2_H
#define LCD16x2_H

#include <reg56303.h>


#ifndef CPUCLK
#define CPUCLK      67000000L   /* Hz */
#endif

#define LCD_16 16           /*Long de una linea del Dply*/

/* ************************** Lineas de control RS, R/W, E y Backlight ****************************** */
#define CTRL_WR 0x4200 /* x10xx01x BL=1(backlight on), RS=0, R/W=0, E=1 */
#define CTRL_RD 0x4600 /* x10xx11x BL=1(backlight on), RS=0, R/W=1, E=1 */
#define DATO_WR         0x6200 /* x11xx01x BL=1(backlight on), RS=1, R/W=0, E=1 */
#define DATO_RD         0x6600 /* x11xx11x BL=1(backlight on), RS=1, R/W=1, E=1 */

/* ***************************** Palabras de control del LCD16x2 ********************************** */
#define CLEAR               0x01    /* 00000001 Borra Dply y vuelve cursor al inicio          */
#define RET_HOME        0x02    /* 00000010 Contador de direcciones a cero                */
#define ETY_MOD_SET     0x06    /* 00000110 I/D=1 inc DDRAM, S=0 sin salto de Dply    */
#define DPLY_ON         0x0C    /* 00001100 D=1 Dply on,  C=0 sin cursor, B=0 sin blink */
#define DPLY_OFF        0x08    /* 00001000 D=0 Dply off, C=0 sin cursor, B=0 sin blink */
#define CSOR_SHFT       0x18    /* 00011000                                               */
#define FUN_SET         0x38    /* 00111000 DL=1 8 bits, N=1 2_lineas, F=don't care       */
#define SDA_LIN         0xC0    /* 11000000 Posiciona el cursor en la 2da linea           */

/ ************************************************************************************************ */

/* typedef union           */
/* {                    */
/*          struct{             */
/*                          int bit0  : 1;  */
/*                          int bit1  : 1;  */
/*                          int bit2  : 1;  */
```

```
/*                              int bit3  : 1;  */
/*                              int bit4  : 1;  */
/*                              int bit5  : 1;  */
/*                              int bit6  : 1;  */
/*                              int bit7  : 1;  */
/*                              int bit8  : 1;  */
/*                              int bit9  : 1;  */
/*                              int bit10 : 1;  */
/*                              int bit11 : 1;  */
/*                              int bit12 : 1;  */
/*                              int bit13 : 1;  */
/*                              int bit14 : 1;  */
/*                              int bit15 : 1;  */
/*          } bits;             */
/*                      */
/*          struct{             */
/*                              int byte0  : 8; */
/*                              int byte1  : 8; */
/*          } bytes;            */
/*                      */
/*          int I;              */
/*                      */
/* } REGISTRO_HDR;          */


/* Inicializa el puerto B(HI08) en modo GPIO */
void init_gpio(void);

/* Escribe un cartacter en la memoria del display */
void write_lcd(unsigned char simbolo, short ctrl_dat);

/* Lee un dato o el estado desde el display */
/*char read_lcd(char ctrl_dat);*/

/* Produce un retardo proporcional a 100us */
void delay(int factor);

/* Inicializa el display LCD16x2 */
void init_lcd(void);

/* Verifica si el Dply esta ocupado */
/*int busy(void);*/

/* Envia los datos que se escribiran en el display */
void dato_lcd(_Y unsigned char *dato, short len);

#endif /*LCD16x2_H*/

/* END OF FILE */
```

Algoritmo ZCR:

zcr_func.c:

```
#include "zcr_func.h"
#include "math.h"


_fract *ptr_huella;
_fract *ptr_h_fin;
/* _fract correl[q-LHG+1]; */          /* vector de correlacion */

/ ************************************************************************************************ */
/* ***************** Funcion que halla el signo del valor que es pasado como parametro ******************** */
/ ************************************************************************************************ */

int signo(_fract x)
{
        _fract cero = 0;

        if(x > cero)  {return 1;}
        if(x < cero)  {return -1;}
        if(x == cero) {return 0;}

        return 2;
                /*caso no valido, evita warning al compilar*/
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ************** Funcion que halla los cruces por cero de una señal que es pasada como parametro ************  */
/ ************************************************************************************************ */

int zcr(_fract *s, int largo)
{
        int i, sg, z;
        _fract p;

        z=0;

        for (i=1; i<largo; i++)
        {
                p  = (*(s+i)) * (*(s+i-1));
                sg = signo(p);
                if ((sg == -1) || (*(s+i) == 0))
                        z++;
        }
        return z;
}
```

```
/ ************************************************************************************************ */



/ ************************************************************************************************ */
/* ******************* Funcion que halla la frecuencia fundamental de un tramo de señal ****************** */
/ ************************************************************************************************ */

short fundamentales(int *ceros)
{
        float zero;

        zero = ceros[0] + ceros[1] + ceros[2];                  /* cruces por cero en un tramo de 30ms */
        *ptr_huella++ = (_fract)(zero/480);             /* normalizacion para no saturar por uso de _fract */
                                                /* 480 = cruces por cero equiv a 8kHz(max f0 detectable) */

        if(ptr_huella < ptr_h_fin)
                return 0;
        return 1;                                                       /* la huella generada esta lista */
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ******************************** Funcion que halla el promedio  ******************************** */
/ ************************************************************************************************ */

_fract promedio(_fract *p, int largo)
{
        int i;
        float suma;
        _fract prom;


        suma = 0;

        for (i=0;i<largo;i++) suma += (float) *(p+i)/largo;

        prom = (_fract)suma;

        return prom;
}
/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ********************** Funcion que halla la correlacion entre dos señales ************************** */
/ ************************************************************************************************ */

short correlacion(_fract *px, _Y _fract *py, _fract media)
{
        float r,suma,sigmax,sigmay,d;
        int k,m,largo;
        _Y _fract *py0;
        double mult;
```

```
        py0 = py;

        sigmax = 0;
   for (k=0; k<LHG; k++)
        sigmax += (*(px+k)) * (*(px+k));


        largo = q-LHG+1;
        for (m=0; m<largo; m++)
   {
                sigmay   = 0;
        suma    = 0;
                py0 = py+m;
        for (k=0; k<LHG; k++)
        {
                        suma += (*(px+k)) * (*(py+k+m)-media);
                /* ******************** */
                sigmay += (*(py+k+m)-media) * (*(py+k+m)-media);
                }

                mult = (double)(sigmax*sigmay);
        d = (float)sqrt(mult);

        r = suma/d;

                /* correl[m] = (_fract) r; */           /* genera el vector de correlacion para ver su grafica en EVM */

                if (r>=UMBRAL) return 1;
        }

        return 0;
}
/ ********************************************************************************************** */


/ ********************************************************************************************** */
/* *************** Funcion que busca la huella pasada como parametro en la base de datos ****************** */
/ ********************************************************************************************** */

short busqueda(_fract *h, _Y _fract **base_de_datos, short tam_bd)
{
        short i;
        _fract media;

        /* calcula la media de la señal */
        media = promedio(h, LHG);

        /* calculo de señal sin valor medio */
        for (i=0; i<LHG; i++)
        {
                *(h+i) -= media;
        }
```

```
        for (i=0; i<tam_bd; i++)
   {
      if (correlacion(h, base_de_datos[i], media) == 1)
         return i;                                          /* indice donde esta la huella coincidente */
   }

   return -1;                                               /* si no la encontro va un -1 */
}
/ ************************************************************************************************ */
/* END OF FILE */
```

zcr_func.h

```
#ifndef ZCR_FUNC_H
#define ZCR_FUNC_H

/ ************************************************************************************************ */
/* ********************** Parametros para la generacion y busqueda de huellas ************************** */
/ ************************************************************************************************ */

#define  UMBRAL      0.70                          /* umbral de decision al comparar huellas */
#define  S           160          /* a fs=16Khz, un intervalo de 10ms, equivale a 160 muestras */
#define  q           597                           /* largo de las huellas de la base de datos */
#define  LHG         199          /* Largo de la Huella Generada de la señal de entrada */
#define  T           3                             /* cantidad de tramos de 10ms */

/ ************************************************************************************************ */


/ ************************************************************************************************ */
/* ***************** Funciones usadas en el algoritmo ZCR y en la busqueda de huellas ******************** */
/ ************************************************************************************************ */

int zcr(_fract *s, int largo);
short fundamentales(int *ceros);
short busqueda(_fract *h, _Y _fract **base_de_datos, short tam_bd);

/ ************************************************************************************************ */

#endif

/* END OF FILE */
```

Tarea principal

main.c:

```
/ ********************************************************************************************** */
/*                                    - PROYECTO AUDIO FINGERPRINT -                              */
/*                              - DANIEL AICARDI - EDGARDO VAZ - MELINA RABINOVICH -              */
/*                                    - DSP - FING - UDELAR -                                     */
/*                                    - 2009 -                                                    */
/ ********************************************************************************************** */

#include "cs4215.h"
#include "loopback.h"
#include "lcd16x2.h"
#include "zcr_func.h"
#include "int.h"
#include "bdh.h"

#define LARGO  32


/* ************************ Mensajes a desplegar y sus largos en display ***************************** */
        _Y unsigned char dsp_afp[]     = "DSP 2009       AUDIOFINGERPRINT";
        _Y unsigned char sw_2[]        = "Presione SW2    para  comenzar  ";
        _Y unsigned char busq[]        = "Buscando       huella...       ";
        _Y unsigned char h_enc[]       = "Huella         encontrada      ";
        _Y unsigned char h_no_enc[]    = "Huella         no encontrada   ";
        _Y unsigned char error[]       = "Error                          ";

        _Y unsigned char enya[]      = "Enya:          Only Time       ";
        _Y unsigned char oasis[]     = "Oasis:         Headshrinker    ";
        _Y unsigned char van_halen[] = "Van Halen:     Eruption        ";
        _Y unsigned char losolima[]  = "Los Olima      Los dos Gallos  ";
        _Y unsigned char zitarrosa[] = "Zitarrosa:     A Jose Artigas  ";
        _Y unsigned char curtidores[] = "Curtidores:    Cuple IMM       ";
        _Y unsigned char mozart[]    = "Mozart:        Marcha Turca    ";
        _Y unsigned char sodastereo[] = "Soda Stereo:    De Musica Ligera";

        _Y unsigned char *dply[NRO_HUELLAS] = {enya/*, oasis, van_halen, losolima, zitarrosa, curtidores,
mozart*/, sodastereo};
/ ********************************************************************************************** */


/* ********************* Buffers y punteros para los datos de entrada desde el codec ********************** */
        _fract _circ buffer_in0[S+1];
        _fract _circ buffer_in1[S+1];                           /*+1 para cumplir el while del llenado de los buffers*/
        _fract _circ buffer_in2[S+1];

        _fract _circ buffer_aux[2*S];                           /*buffer auxiliar para no sobreescribir memoria*/

        extern _fract _circ *ptr_buffer;
```

```
	_fract *ptr_fin;
	_fract *ptr_ini;

/ ************************************************************************************************ */
/* ************************* Buffers y punteros para el manejo de huellas ***************************** */
	_fract huella[LHG];                                  /*muestras de frecuencias fundamentales*/
	int ceros_tramo[T];                                  /*cantidad de ceros en c/tramo de 10ms*/

	extern _fract *ptr_huella;
	extern _fract *ptr_h_fin;
/ ************************************************************************************************ */

short h_lista ;                                          /*indica que la huella generada esta lista*/
extern short sw2;


int main(void)
{
	short resultado;                                     /*0 = no encontrada, 1 = encontrada*/
	short uso_buff;                                      /*indica numero de buffer en uso*/


/* ************************** Inicializa puerto, display, switch2, interrupciones  ************************ */
	init_gpio();                                         /*inicializa puerto donde se conecta el lcd*/
	init_lcd();                                          /*inicializa el display, ver HD44780.pdf*/
	init_sw();                                           /*inicializa switch 2 y led D12*/
	ssi_init();                                          /*inicializa codec*/


	/*mensaje de bienvenida*/
	disable_interrupts();
	dato_lcd(dsp_afp, LARGO);
	delay(3000);
	write_lcd(CLEAR, CTRL_WR);                           /*comando Clear Dply*/
	delay(100);

	for(;;)
	{
		sw2 = 0;
		resultado = -1;
		uso_buff = 0;
		h_lista = 0;
		ptr_huella = huella;                             /*inicio el puntero al arreglo de frec f0*/
		ptr_h_fin = ptr_huella + LHG;
		ptr_buffer = buffer_aux;

		dato_lcd(sw_2, LARGO);
		delay(1000);
		enable_interrupts();
		while(!sw2);                                     /*sw2=1 arranca la busqueda*/
		disable_interrupts();
		write_lcd(CLEAR, CTRL_WR);                       /*comando Clear Dply*/
		delay(100);
		dato_lcd(busq, LARGO);
		enable_interrupts();
```

```
/ ********************************************************************************************** */
/* ************** Calculo los ceros en 2 tramos de 10ms y luego el tercero dentro del loop ******************* */
/* ************ la funcion fundamentales() calcula la frecuencia fundamental de 3 tramos de  ***************** */
/* *************************** 10ms c/u y la guarda en el arreglo huellas[] ***************************** */
/ ********************************************************************************************** */

        ptr_buffer = buffer_in0;                                    /*inicializo puntero buffer0*/
        ptr_fin = buffer_in0 + (S-1);
        ptr_ini = buffer_in0;

        while(ptr_buffer < ptr_fin);                                /*espero hasta llenar el buffer*/
        ptr_buffer = buffer_in1;                                    /*inicializo puntero buffer1*/
        ptr_fin = buffer_in1 + (S-1);
        ceros_tramo[1] = zcr(ptr_ini, S);                                   /*ceros en un tramo 0*/
        ptr_ini = buffer_in1;
        while(ptr_buffer < ptr_fin);                                /*espero hasta llenar el buffer*/
        ptr_buffer = buffer_in2;                                    /*inicializo puntero buffer1*/
        ptr_fin = buffer_in2 + (S-1);
        ceros_tramo[2] = zcr(ptr_ini, S);                                   /*ceros en un tramo 1*/


        while(!h_lista)
        {
                while(ptr_buffer < ptr_fin);                        /*espero hasta llenar el buffer*/
                switch(uso_buff)
                {
                        case 0:  ptr_buffer = buffer_in0;           /*inicializo puntero buffer0*/
                                 ptr_ini = buffer_in2;
                                 break;
                        case 1:  ptr_buffer = buffer_in1;           /*inicializo puntero buffer1*/
                                 ptr_ini = buffer_in0;
                                 break;
                        case 2:  ptr_buffer = buffer_in2;           /*inicializo puntero buffer2*/
                                 ptr_ini = buffer_in1;
                                 break;
                        default: break;
                }
                ptr_fin = ptr_buffer + (S-1);           /*no puedo sumar S porque ptr_buffer es circ*/
                ceros_tramo[uso_buff] = zcr(ptr_ini, S);                    /*ceros en un tramo*/
                h_lista = fundamentales(ceros_tramo);               /*indica si la huella esta lista*/
                uso_buff++;
                uso_buff = uso_buff % 3;
        }
        disable_interrupts();
        resultado = busqueda(huella, (_fract **)h_conocidas, NRO_HUELLAS);
        write_lcd(CLEAR, CTRL_WR);                                  /*comando Clear Dply*/
        delay(100);
        if(resultado != -1)
        {
                dato_lcd(h_enc, LARGO);
                delay(2000);
```

```
                        write_lcd(CLEAR, CTRL_WR);                      /*comando Clear Dply*/
                        delay(100);
                        dato_lcd(dply[resultado], LARGO);
                }
                else
                        dato_lcd(h_no_enc, LARGO);
                delay(1000);
                write_lcd(CLEAR, CTRL_WR);                              /*comando Clear Dply*/
                delay(100);
                TCSR0.B.TDO = OFF;                              /*led D12 OFF, termina la busqueda*/
        } /*Fin loop ppal*/
} /*Fin main*/

/* END OF FILE */
```