Laboratorio de Sistemas Electrónicos Digitales

Universidad Politécnica de Madrid
Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación

Memoria del proyecto desarrollado en el Laboratorio de Sistemas Electrónicos Digitales (LSED)

Curso 2011/2012

Sistema de procesamiento digital de señales de audio para el MCF5272

Autores: Alfredo Álvarez Senra

Rafael López Martínez

Código de la pareja: LT-21

ÍNDICE GENERAL

1 Introducción

El objetivo del proyecto del Laboratorio de Sistemas Electrónicos Digitales para el curso 2011/2012 es el desarrollo de un sistema de procesamiento digital de señal sobre la plataforma ColdFire MCF5272, que permita **filtrar**, **ecualizar** y añadir una **reverberación** simple a una señal de audio. Para lograr dichos objetivos se muestreará una señal analógica con el ADC propio del ColdFire, se procesará y se entregará al DAC para que entregue la señal digitalmente procesada por la salida analógica.

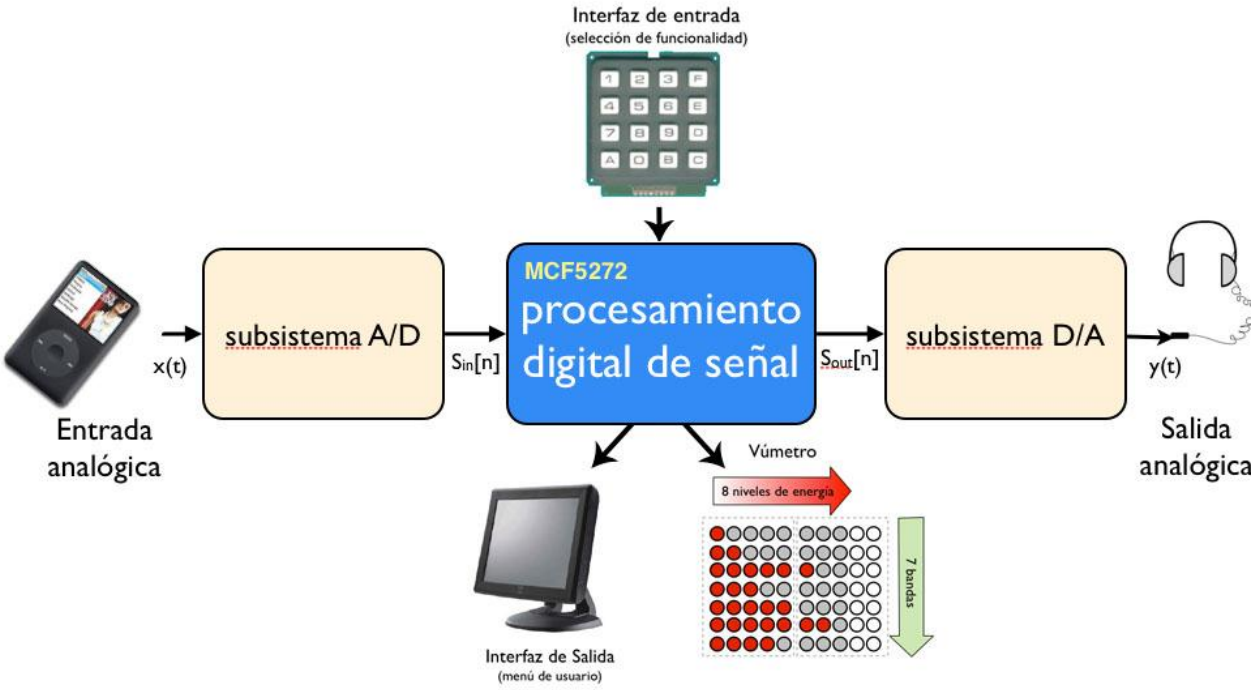
La primera de las tres funciones principales que se implementaran es la **caracterización** de 7 filtros paso banda, que podrá hacerse de forma individual o de forma conjunta. La segunda función principal es la posibilidad de ecualizar la señal de audio, seleccionando una atenuación para cada una de las 7 bandas de frecuencia implementadas. La tercera función principal es la inclusión de un efecto de reverberación simple que podremos configurar seleccionando el retardo y la atenuación de dicho efecto.

Las mejoras que hemos realizado son la implementación de pre-sets de reverberación simple, la aplicación de modos de ecualización pre-establecidos, la visualización de los niveles de ecualización en la matriz de leds, así como poder elegir la visualización de la respuesta en frecuencia de la señal de entrada o de salida.

Las palabras resaltadas en negrita son nuestras palabras clave.

2 Diagrama de subsistemas

2.1 Diagrama general



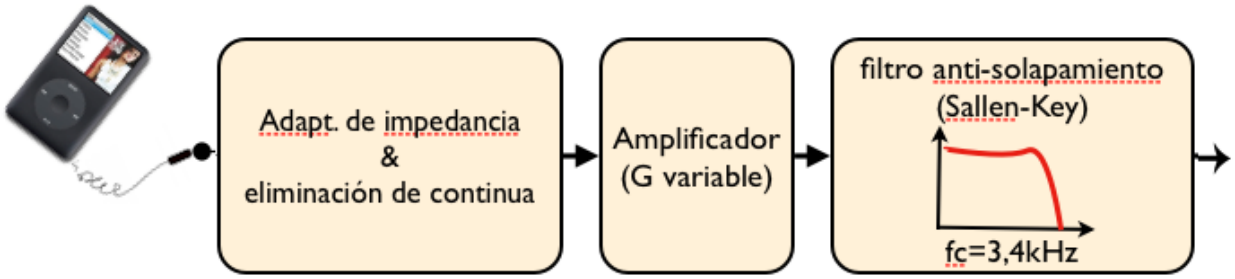
El sistema de procesamiento constará de 3 etapas:

- Una etapa HW analógico digital encargada de acondicionar la señal de audio entregada por un reproductor musical para su posterior procesamiento en el microcontrolador ColdFire MCF5272.
- Una etapa de procesamiento mediante SW utilizando el microcontrolador ColdFire MCF5272 disponible en el laboratorio.
- Una última etapa HW de acondicionamiento de la señal ya procesada para ser escuchada por unos auriculares.

Por otro lado el usuario podrá comunicarse con el sistema mediante una interfaz de 3 elementos:

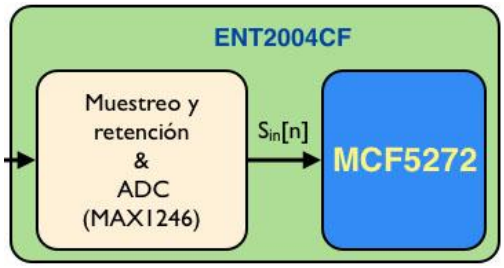
- Un teclado matricial de 16 teclas con el que se podrán seleccionar las diferentes opciones.
- Una pantalla en la que se mostraran los mensajes correspondientes.
- Una matriz de leds encargada de mostrar la energía de la señal o el ecualizador gráfico.

2.2 Diagrama subsistema HW analógico – digital



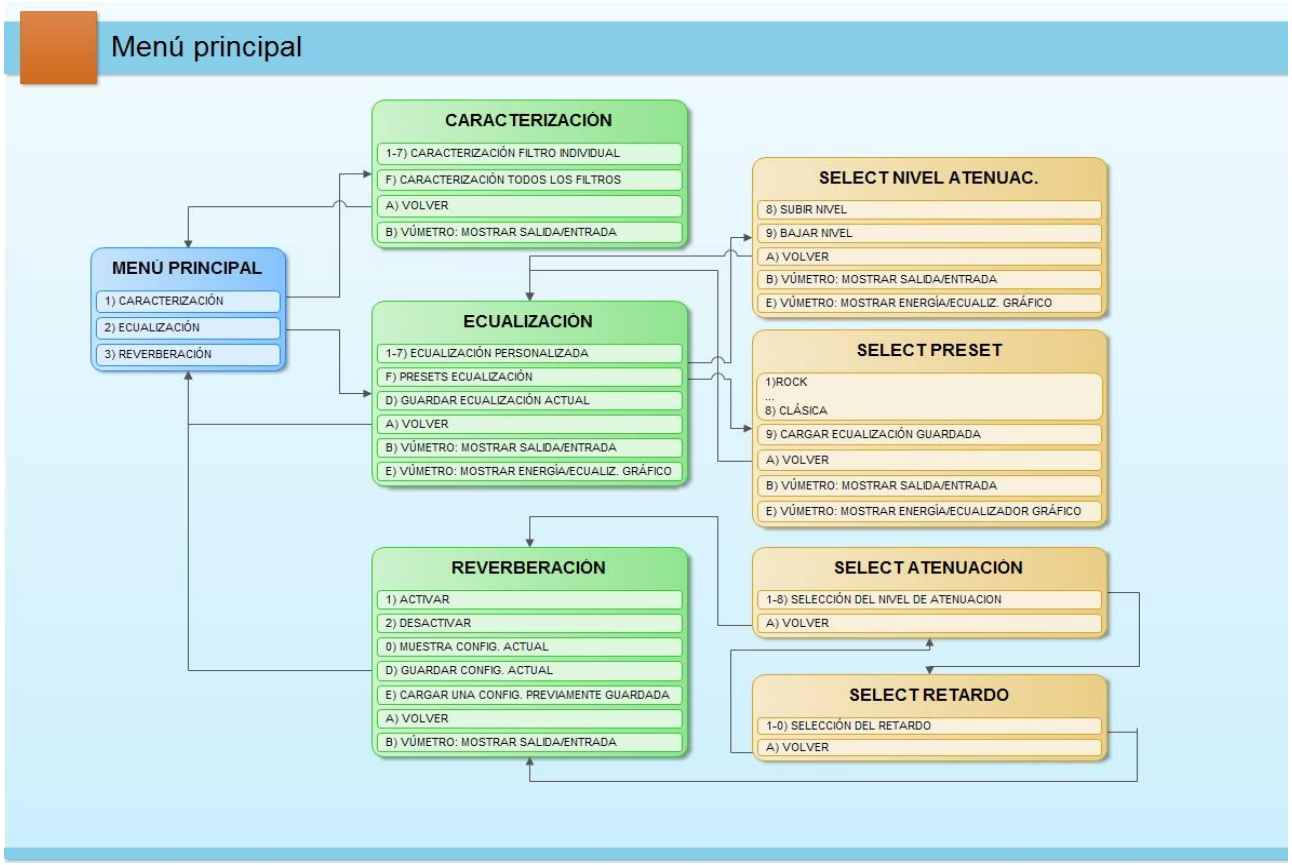
Este subsistema se encarga de acondicionar la señal de audio entregada por un reproductor musical para su posterior procesamiento. Dicho acondicionamiento se basa en la eliminación de la componente continua de la señal y la eliminación de las componentes de frecuencia por encima de la frecuencia de Nyquist. Además se implementa un amplificador de ganancia variable para poder entregar a la salida el nivel de señal deseado.

La señal de salida de esta etapa se entrega a la entrada analógica de la plataforma ENT2004CF que se encargará del muestreo y la conversión analógica-digital a través del ADC MAX1246 disponible en dicha plataforma.



2.3 Diagrama subsistema SW

Dividimos el subsistema SW en dos partes que actúan de forma paralela:



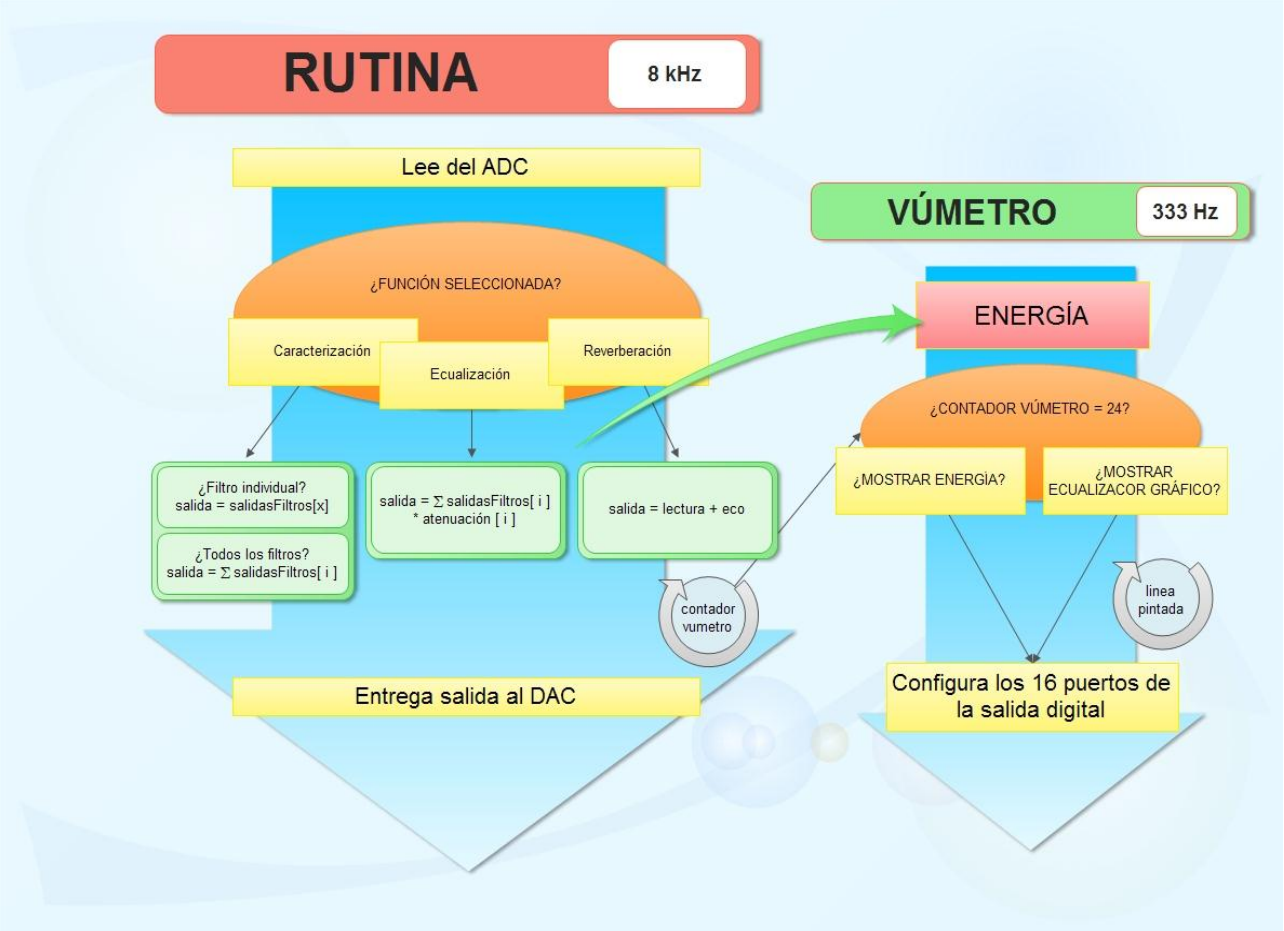
- **Menú principal**: Permite seleccionar la función que deseamos ejecutar y guardar la configuración seleccionada en variables que usará la rutina:

  1) **Caracterización de filtros.**
  2) **Ecualización**.
  3) **Efecto de reverberación.**

✓ **Caracterización de filtros:** En este submenú se modifican las variables de estado correspondientes para que sea posible caracterizar cada uno de los filtros implementados individualmente o de forma conjunta. Además tendremos la opción de mostrar en el vúmetro la señal de entrada o la de salida.

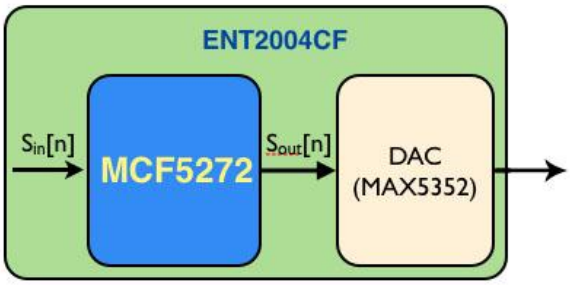
✓ **Ecualización:** En este submenú se modifican las variables de estado correspondientes para que sea posible ecualizar la señal. Se implementa un ecualizador gráfico de octava empleando el banco de filtros. Puesto que no podemos amplificar la señal por cuestiones de saturación en el DAC, la ecualización se implementara seleccionando un nivel de atenuación para cada banda. Los niveles de atenuación podrán ser configurados de forma individual para cada filtro o seleccionando uno de los presets disponibles. También será posible guardar y cargar las configuraciones personalizadas. Por último podremos seleccionar que deseamos visualizar en la matriz de leds, por un lado la energía de la señal de salida o la de entrada, y por otro el ecualizador gráfico.

- ✓ **Efecto de reverberación**: En este submenú se modifican las variables de estado correspondientes para que sea posible añadir un efecto de reverberación simple. Podremos configurar dicho efecto seleccionando la atenuación y el retardo del eco. Tendremos la opción de guardar y cargar distintas configuraciones en varias posiciones de memoria. Por último podremos seleccionar que deseamos visualizar en el vúmetro, la energía de la señal de salida o la de entrada.

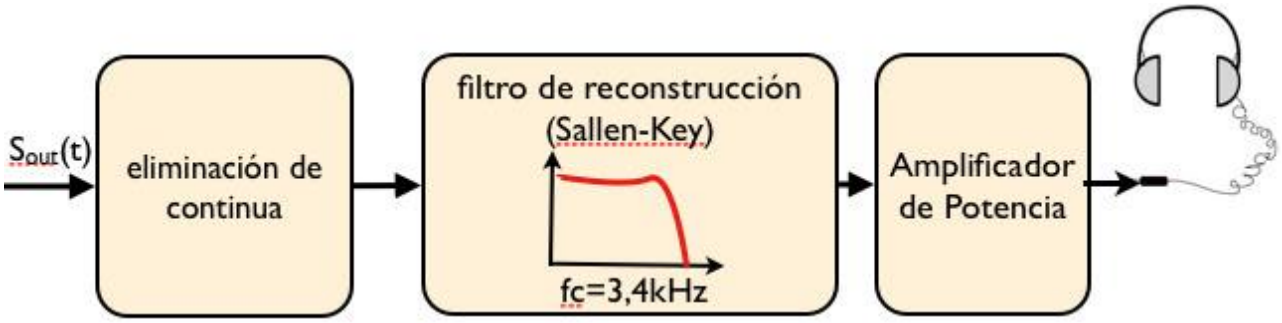


- La rutina tout0 se ejecuta continuamente cada 125 microsegundos, o lo que es lo mismo, con una frecuencia de 8 kHz. Es la encargada de leer la muestra del ADC y procesarla según las variables de estado configuradas en el menú descrito anteriormente, para posteriormente entregarla al DAC. También es la encargada de hacer los cálculos y configuraciones necesarias para mostrar la energía de cada banda en la matriz de leds.

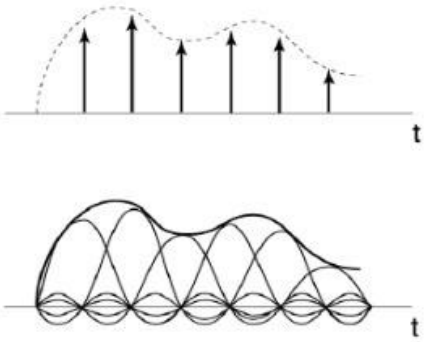
2.4 Diagrama subsistema HW digital - analógico



La señal de entrada de esta etapa es entregada por la salida analógica de la plataforma ENT2004CF a través del DAC MAX5352 disponible en dicha plataforma.



Este subsistema se encarga de acondicionar la señal de audio procesada entregada por el DAC MAX5352 para poder ser escuchada correctamente en unos auriculares. Dicho acondicionamiento se basa en la eliminación de la componente continua de la señal y la reconstrucción de la señal a partir de los pulsos derivados del muestreo.



Además se implementa un amplificador de potencia de ganancia variable, con el que ajustaremos el volumen.

3 Descripción del subsistema Hardware

El HW utilizado en el nuestro sistema de procesamiento de audio consta de tres etapas:

- Un subsistema de conversión Analógico/Digital: encargado de acondicionar la señal de audio de entrada para poder ser procesada digitalmente en la plataforma ENT2004CF.

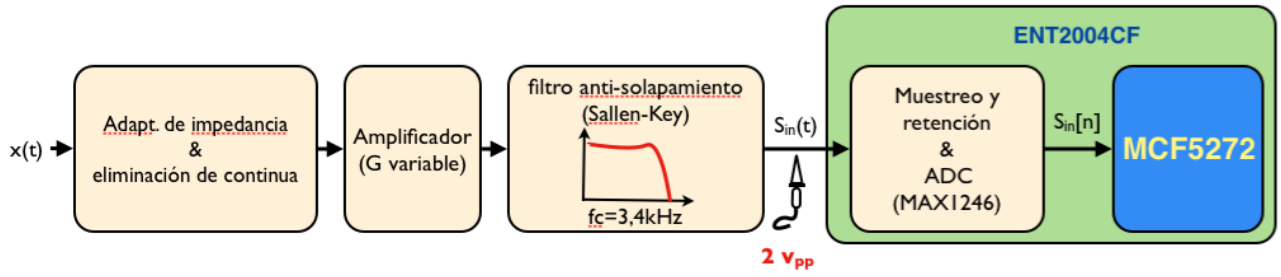
- Un módulo que realice el procesado digital: de esto se encargara el microcontrolador ColdFire, en el que cargaremos nuestro programa para filtrar, ecualizar o añadir efectos (reverberación) a nuestra señal de entrada. A esta etapa le añadiremos un sistema de visualización de energía utilizando la matriz de leds disponible en puesto de trabajo.

- Un subsistema de conversión Digital/Analógico: se encarga de acondicionar señal que nos entrega la platadorma ENT2004CF (a través del DAC MAX5352) para poder reproducirla adecuadamente en los auriculares.

A continuación ofrecemos un desarrollo mas detallado de los montajes hardware que hemos utilizado a lo largo de la práctica.

3.1 Descripción del subsistema de conversión analógico-digital.

Las etapas básicas de nuestro subsistema de conversión Analógico/Digital son:

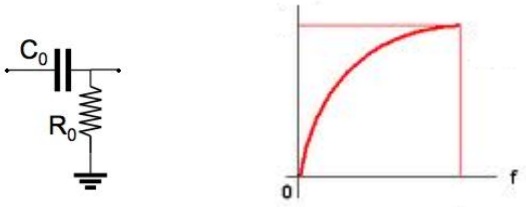


La señal de entrada x(t) proviene de un reproductor de audio que incorporamos a nuestro sistema conectándolo a la entrada de nuestro circuito con un cable con conectores "Jack macho" en ambos extremos.

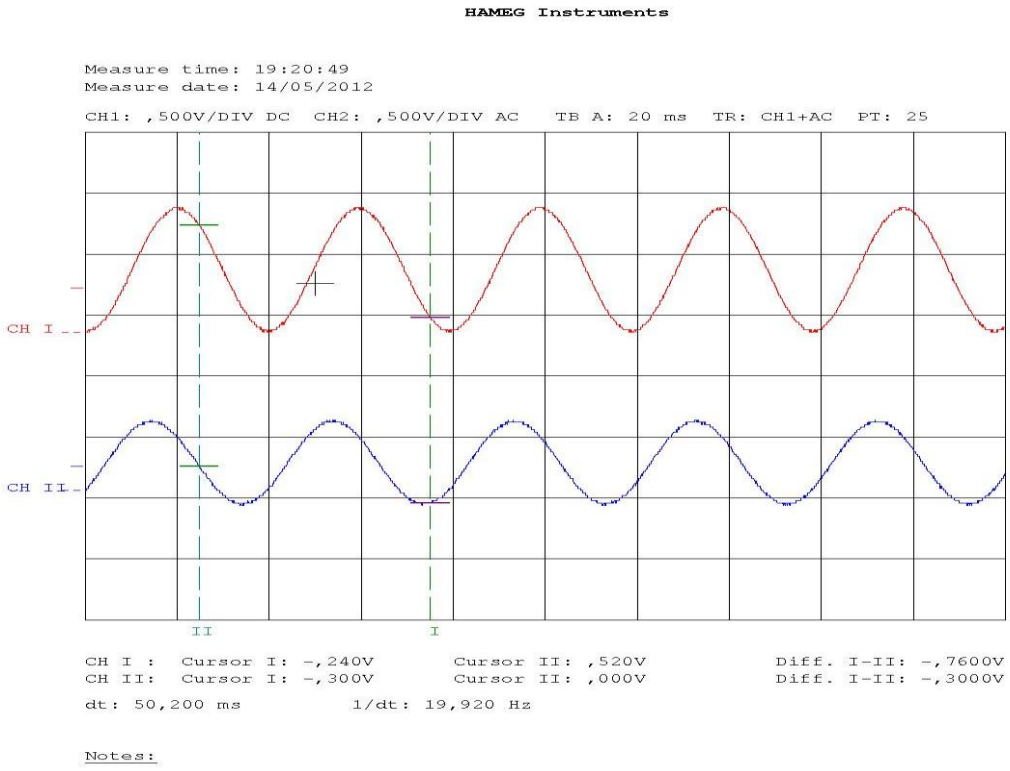
3.1.1 Filtro paso alto:

El primer elemento que aparece en el recorrido de la señal es una resistencia de 10Ω en paralelo al reproductor que simula la impedancia de los auriculares.

Tras la adaptación de impedancias, diseñamos un filtro RC paso alto ($f_c=1/(2.\pi.R.C)$) que limita el paso de corriente continua intentando atenúar lo menos posible el resto de frecuencias. El montaje será como el del siguiente esquema, empleando R_0=12KΩ y C_0=470nF, obteniendo así una frecuencia teórica de corte de 28,22 Hz.

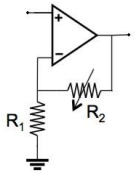


Empíricamente medimos una frecuencia de corte f_c=19,92 Hz, tal y como podemos observar en la siguiente figura (caracterizamos el filtro aislado y empleando una señal sinusoidal de amplitud 1Vp y buscando la frecuencia donde a la salida obtenemos la señal con una amplitud √2 veces la amplitud de la señal de entrada):



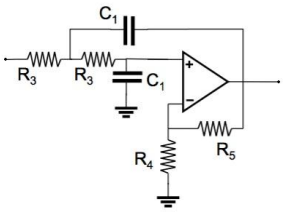
3.1.2 Amplificador no inversor:

Esta etapa se encarga de proporcionar ganancia a la señal para alcanzar el objetivo de entregar al ADC una señal de entrada de 2 Vpp de amplitud. Para ello en R_2 ponemos un potenciómetro que ajustaremos para alcanzar este objetivo. El esquema de montaje será el de la siguiente figura. La R_1 utilizada es de 10KΩ y el potenciómetro de 50KΩ.



3.1.3 Filtro anti-solapamiento Sallen Key:

Este filtro anti-solapamiento será un filtro paso bajo de 2º orden Sallen-Key ($f_c=1/(2.\pi.R_3.C_1)$) que deberá estar por encima de la frecuencia de Nyquist. El montaje será el siguiente:



Para este filtro tenemos que cumplir que el factor de calidad $Q=1/\sqrt{2}$, y para esta restricción tenemos una ganancia de $G=3-\sqrt{2}=1+R_5/R_4=1{,}586$. Esto se cumple para una relación de resistencias de $R_4=2{,}7K\Omega$ y $R_5=1{,}6K\Omega$. la frecuencia de corte $f_c=3{,}386KHz$ para $R_3=9{,}1K\Omega$ y $C_1=4{,}7nF$. A la salida de esta etapa debemos tener una señal de amplitud 2 Vpp. Para ello ajustamos el potenciómetro de la etapa anterior ajustando la ganancia.

Empíricamente los resultados que obtuvimos son los que presentamos en los siguientes diagramas y tablas (caracterizamos el filtro aislado y con sinusoides de 1Vp, a distintas frecuencias):

Medidas obtenidas empíricamente:

f(Hz)	Av	Av(dB)	tiempo desfase (ms)	Periodo (ms)	Fase (grados)
10	1,57	3,917993048	0,3	100,5	1,074626866
20	1,54	3,750414417	0	100,5	0
50	1,54	3,750414417	0,1	20,2	1,782178218
80	1,53	3,693828616	0	10,04	0
100	1,53	3,693828616	0,025	2	4,5
300	1,53	3,693828616	-0,051	1	-18,36
500	1,547	3,789806274	0	0,201	0
700	1,54	3,750414417	-0,005	0,102	-17,64705882
1000	1,53	3,693828616	-0,005	0,0198	-90,90909091
3000	1,346	2,580901198	-0,005	0,0101	-178,2178218
5000	0,785	-2,102606865	-0,001	0,001	-360
7000	0,454	-6,858882943	-0,001	0,001	-360
10000	0,238	-12,46846086	-0,001	0,001	-360

Diagrama del modulo del LPF:

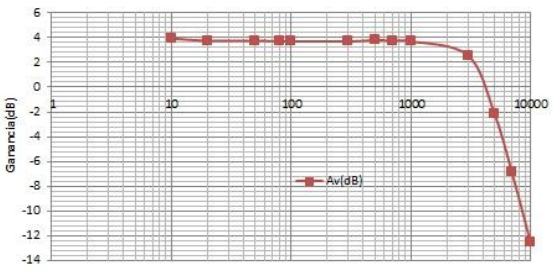
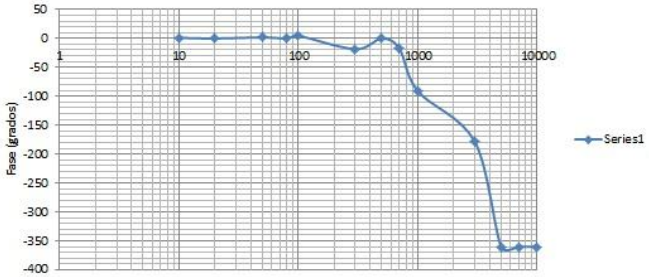


Diagrama de la fase del LPF:



3.1.4 Conversor analógico-digital

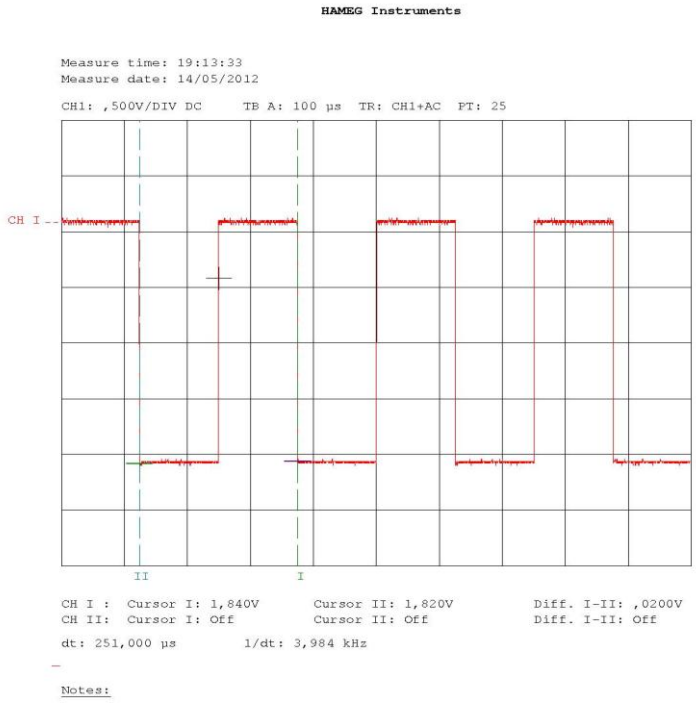
La salida de la etapa anterior se entrega al ADC MAX1246 disponible en la plataforma ENT2004CF que se encargará de muestrearla. Para ello debemos configurar el ADC en modo asimétrico para procesar señales bipolares (entre ±2,5V) con una frecuencia de muestreo de 8kHz. Para realizar todo esto modificamos la librería m5272adc_dac para que acepte señales bipolares, ya que por defecto solo acepta señales entre 0 y 5 V. la línea de código que modificamos quedara de esta forma:

```
mbar_writeShort(MCFSIM_QDR, 0x9797);
```

La interrupcion a 8KHz. La configuramos en nuestro programa principal:

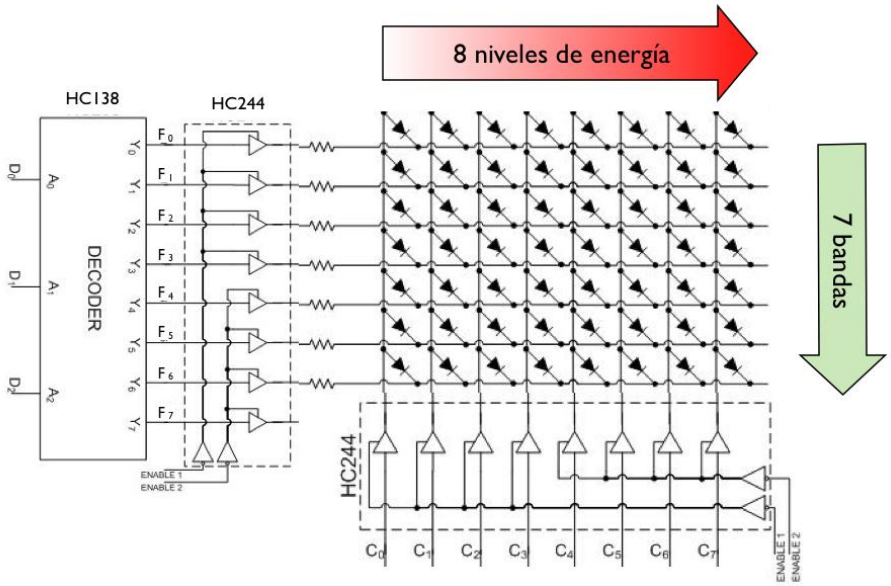
```
#define FREC_INT 8000
```

Y la medimos con el osciloscopio, como se puede ver en la siguiente captura:



3.2 Descripción del subsistema de visualización de energía

Para implementar el vúmetro que mostrará la energía de cada una de las bandas de frecuencia del banco de filtros digitales, utilizaremos la matriz de leds disponible en el puesto de trabajo y un esquema como el siguiente:



Cada una de las 7 filas identificarán las 7 bandas de frecuencia; y cada uno de las 8 columnas identificaran 8 niveles de energía, siendo el nivel máximo cuando tenemos una señal de 1,5 Vp y el mínimo cuando tenemos una señal de 25mV.

Para controlar las filas y las columnas de que deben ser iluminadas utilizaremos el puerto de salida disponible en la plataforma.

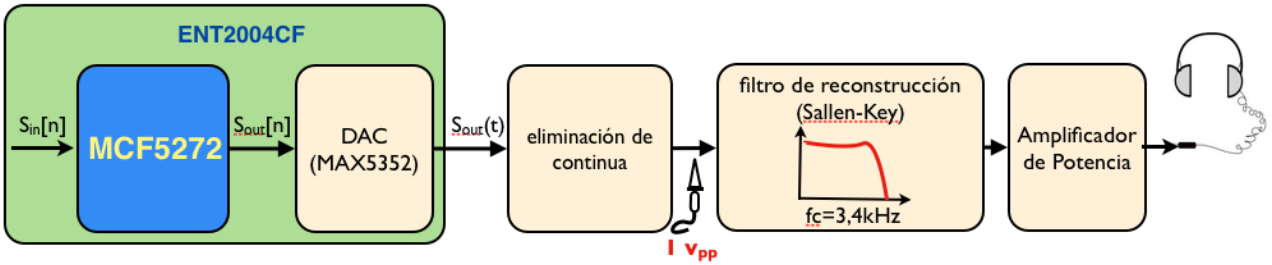
Para el caso de las filas utilizaremos los bits 4, 5 y 6 de dicha salida, un demultiplexor 3a8 HC138 y un banco de buffers triestado HC244. De esta manera y junto con el SW que configura los bits mencionados, aplicaremos 0 V a la fila que queramos iluminar y 5V al resto.

Para el caso de las columnas utilizaremos los bits 8 a 15 de dicha salida y un banco de buffers triestado HC244. De esta manera y junto con el SW que configura los bits mencionados, aplicaremos 5 V a las columnas que queramos iluminar y 0V al resto.

Además utilizaremos unas resistencias de 220Ω para proteger la matriz de leds.

3.3 Descripción del subsistema de conversión digital-analógico

Consta de las siguientes etapas:



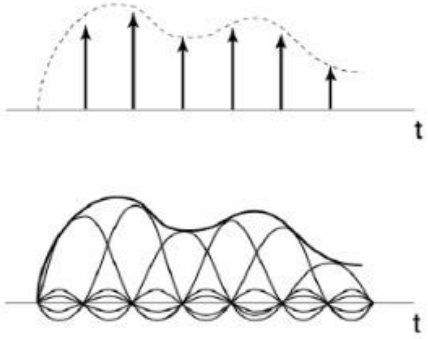
Esta etapa recibe la señal que entrega el DAC (capaz de ofrecer tensiones de salida de 0 a 2,5V) y se encarga de acondicionar dicha señal para ser escuchada con unos auriculares.

3.3.1 Filtro paso-alto:

Es la primera etapa de este subsitema, y es equivalente al montado en el punto 3.1.1 y con el mismo esquema eléctrico, los mismos elementos (R_0=12KΩ y C_0=470nF) e igual frecuencia de corte (f_c=19,92 Hz). Su finalidad es también la eliminación de la componente continua de la señal.

3.3.2 Filtro de reconstrucción.

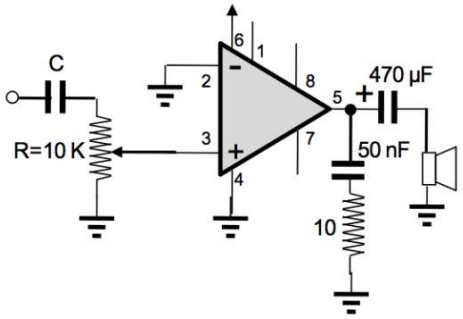
Este filtro es necesario para poder reconstruir la señal analógica a partir de los códigos binarios que la representan. Este es un filtro equivalente al filtro anti-solapamiento del punto 3.1.3 para eliminar las copias espectrales que se producen a la salida del DAC.



El esquema eléctrico será el mismo que el del Sallen-Key, con valores de elementos eléctricos iguales (R_4=2,7KΩ, R_5=1,6KΩ, R_3=9,1KΩ y C_1=4,7nF), y frecuencia de corte teorica f_c=3,386KHz.

3.3.3 Amplificador de potencia.

Esta etapa la conectamos para incrementar la corriente que excita a los auriculares para poder reproducir nuestra señal de audio. Para ello montamos este amplificador de potencia sobre un chip LM386. Además puesto que es necesario eliminar la componente continua antes de amplificar, montamos un nuevo filtro RC paso alto utilizando un condensador de 680 nF y los 10KΩ de un potenciómetro que también utilizamos para varia la ganancia del amplificador. Utilizaremos una resistencia de 10Ω para modelar la impedancia de los auriculares. El esquema eléctrico es el que sigue:

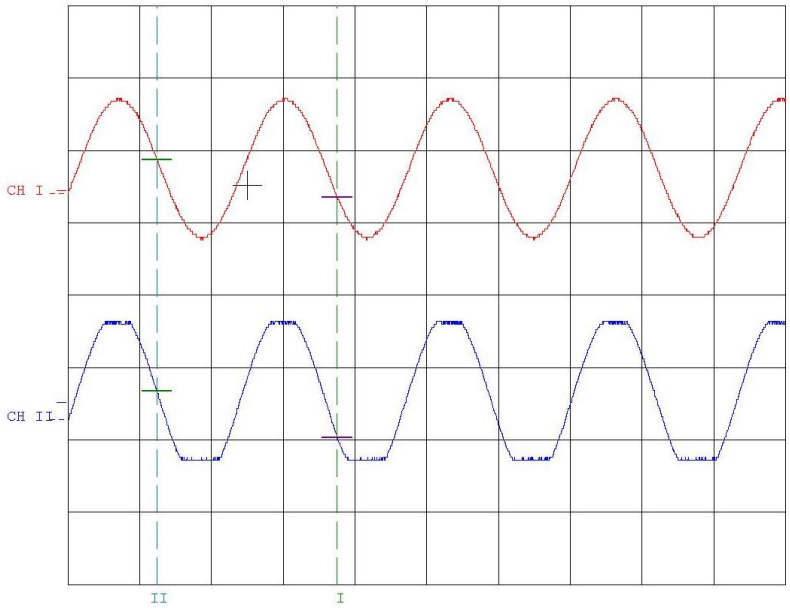


El punto de saturación lo alcanzamos en el punto que nos indica la siguiente figura:

HAMEG Instruments

Measure time: 19:33:17
Measure date: 14/05/2012

CH1: 1,000V/DIV DC CH2: 2,000V/DIV AC TB A: 1 ms TR: CH1+AC PT: 25



CH I : Cursor I: -,080V Cursor II: ,440V Diff. I-II: -,5200V
CH II: Cursor I: -,960V Cursor II: ,320V Diff. I-II: -1,2800V

dt: 2,510 ms 1/dt: 398,406 Hz

Notes:

4 Descripción del subsistema Software

El objetivo es implementar un programa en leguaje de programación C que permita **caracterizar** un banco de 7 filtro de forma independiente o conjunta, **ecualizar** una señal de audio aplicando una determinadas atenuaciones a cada banda de frecuencias (utilizando el banco de filtros) y añadir un efecto de **reverberación** simple.

La estructura principal se basa en "la ejecución de forma paralela de dos programas" (Ver apartado 2.3). Uno es el menú, que permite seleccionar opciones y configurar variables. Por otro lado, una rutina de interrupción de frecuencia 8kHz hace las operaciones correspondientes sobre la señal de audio para procesarla según las variables configuradas con el menú mencionado previamente.

Se han creado diferentes ficheros.c con el fin de dar al código una estructura modular que haga más sencilla su lectura y comprensión. Estos ficheros.c se incluyen en el fichero principal LT-21.c.

Es importante resaltar que debido a que se trata de un programa que funciona en tiempo real, hemos tenido especial cuidado en la optimización de recursos, puesto que de no haberlo hecho podríamos vernos en la situación de que al programa no le diese tiempo a hacer las operaciones necesarias y estropear así la señal de audio que deseamos procesar.

Tras ejecutar el programa y ejecutarse el código de la rutina `__init()` que inicializa, el DAC y el ADC, las direcciones de los vectores de interrupción y habilita y configura las interrupciones y su frecuencia de interrupción (8 kHz), se ejecuta la función bucleMain(), que nos muestra el menú que permite configurar las variables de estado que se utilizarán en la rutina de interrupción para, según estas, aplicar unas operaciones u otras a las muestras que entregue el ADC.

4.1 Proceso del programa principal:

```
void bucleMain(void){
  GestionMenuPrincipal ();
}
```

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: ninguna.

Como ya hemos explicado, muestra el menú que permite configurar las variables de estado. Esta acción se hace llamando a la función GestionMenuPrincipal() que se encuentra definida en el archivo Menú.c.

4.1.1 Función GestionMenuPrincipal

void GestionMenuPrincipal (void){
Inicia el array de estado a cero (todas las funciones desactivadas)
Muestra el menú principal con las 3 funciones del programa y espera a que una de ellas sea seleccionada.
Guarda la opción seleccionada en estado[FUNCION] y llama a la subrutina correspondiente.
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función GestionMenuPrincipal(): Muestra el menú principal, guarda la opción seleccionada y hace las gestiones necesarias para seguir adelante con la configuración.

4.1.2 Función GestionCaracterizacion

void GestionCaracterizacion (void){
Muestra el submenú de caracterización y espera a que se seleccione una opción.
Guarda la opción de seleccionada.
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función GestionCaracterizacion (): Muestra el submenú de caracterización. La caracterización de un filtro individual permite escuchar solo las componentes de la señal que se encuentren en las frecuencias que deje pasar el filtro seleccionado. También es posible caracterizar todos los filtros a la vez. Guarda en estado[CARACTERIZACION] la opción de caracterización seleccionada. En la opción de visualizar en el vúmetro la entrada o la salida, modificamos estado[MESTRAENTRADA] (ver apartado 5).

4.1.3 Función GestionEcualizacion

void GestionEcualizacion (void){
Muestra el submenú de ecualización y espera a que se seleccione una opción.
Guarda la opción de seleccionada.
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función GestionEcualizacion (): Muestra el submenú de ecualización. La ecualización se llevará a cabo aplicando a cada banda de frecuencias la atenuación correspondiente a uno de los 16 niveles seleccionables. Estos niveles de atenuación pueden ser seleccionados de forma individual para cada banda de frecuencia (la banda seleccionada se guarda en estado[CARACTERIZACION]) o seleccionando uno de los presets de ecualización guardados en memoria. El nivel de atenuación seleccionado para cada banda de frecuencias se guardará en el array nivel[]. Existe una opción exclusiva de la función de ecualización que consiste en utilizar la matriz de leds para mostrar el ecualizador grafico, esta opción se guarda en la posición estado[MUESTRAECUALIZADOR] (ver apartado 5).

4.1.4 Función seleccionNivel

void seleccionNivel (int bandaNivel){
Permite seleccionar el nivel de atenuación de una banda determinada
Tras modificar un nivel muestra el nivel de cada banda.
}

Variables de entrada: estado[ECUALIZACION]**.**
Variables de salida: void**.**
Variables generales modificadas: estado[], nivel[]

Función seleccionNivel (): permite seleccionar el nivel de atenuación de la banda seleccionada anteriormente con las teclas 8 y 9. Guarda el nivel seleccionado en nivel[].

4.1.5 Función muestraEstadoEcualizacion

void muestraEstadoEcualizacion (void){
imprime en pantalla y vuelve al punto donde se le invocó.
}

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: ninguna.

Función muestraEstadoEcualizacion (): muestra en pantalla una tabla con los niveles seleccionados actualmente para cada una de las bandas de frecuencia.

4.1.6 Función GestionReverb

void GestionReverb (void){
Muestra el submenú de reverberación y espera a que se seleccione una opcion.
}

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función GestionReverb(): muestra el menú de reverberación que permite activar o desactivar dicho efecto. El estado del efecto se guarda en la posición estado[REVERBERACION] siendo 0 desactivado y 1 activado.

4.1.7 Función selectAtenuacion

void selectAtenuacion (void){
permite selecionar una de las 8 niveles de atenución.
}

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función selectAtenuacion (): permite seleccionar uno de los 8 niveles de atenuacion. Dicho nivel es el factor por el que dividiremos nuestra señal retardada. Simula la atenuación que sufre una señal acústica en el viaje de ida y vuelta y en el momento del rebote. Dicho factor se guardara en estado[ATTREVERB].

4.1.8 Función selectRetardo

void selectRetardo (void){
Permite selecionar el retardo con el que oiremos el eco.
Muestra el resultado final de la configuración
Pone el estado de la reverberacion a "activado"
Vuelve al submenú principal de reverberación
}

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: estado[].

Función selectRetardo (): permite seleccionar el retardo con el que se oira el eco. Dicho retardo está relacionado con la distancia a la que se encuentran las paredes donde rebota la señal acustica. El retardo seleccionado se guardará en estado[TIEMPORETARDO].

4.2 Proceso de la rutina de interrupción

void rutina_tout0(void){
Lee la muestra del ADC y le hace la extencion de signo.
Configura la salida que entregará al DAC modificando la lectura según lo que indiquen las variables que se han manipulado con el menú.
Si CARACTERIZACION: filtra la señal y entrega la componente del filtro seleccionado.
Si ECUALIZACION: filtra y multiplica cada banda por su atenuacion.
Si REVERBERACION: llama a la funcion salidaReverberacion() que suma a la lectura del ADC el eco correspondiente.
Entrega la salida al DAC.

Configura la salida digital para gobernar el vúmetro.
Para ello calcula la energía de cada banda de frecuencias durante 24 interrupciones.
Pinta una linea de leds cada 24 interrupciones.
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: ninguna.

Es la parte más importante del programa, ya que es la encargada de procesar la señal de audio y de gestionar la iluminación del vúmetro. Se ejecuta cada 125 microsegundos y sigue siempre la misma estructura: evaluar variables – hacer operaciones. La primera parte de la rutina realiza las modificaciones necesarias a la muestra entregada por el ADC para entregarla posteriormente al DAC y, tras una etapa HW, ser escuchada en unos auriculares.

Estas modificaciones se harán según hayamos configurado las variables de estado con el menú.

El primer punto de divergencia que encontramos es la función que deseamos implementar: **caracterización**, **ecualización** o **reverberación**, que indica la posición estado[FUNCION]. Según esta variable se aplicaran unas transformaciones u otras:

- En el caso de la caracterización, se filtra la lectura del ADC y tras evaluar si se ha seleccionado un solo filtro o todos a la vez, se guarda en la variable "salida" la componente o suma de componentes correspondientes para ser entregada al DAC. En el caso de la caracterización de los 7 filtros a la vez, obtenemos una ganancia de 3dB al realizar la suma de las componentes resultantes del filtrado. Para compensar esta ganancia dividimos entre $\sqrt{2}$. Para no trabajar con decimales (por limitaciones de HW) se ha implementado esta división multiplicando por 1500 y desplazando 11 posiciones a la derecha, o lo que es lo mismo, multiplicar por $((\sqrt{2}/2)\cdot 1024)/1024$ que es equivalente a la división mencionada anteriormente.

- En el caso de la ecualización, se filtra la lectura del ADC se aplica la atenuación correspondiente a cada banda, indicada por el array nivel[], y se guarda en la variable "salida" la suma de las componentes de las 7 bandas para ser entregada al DAC. Resaltamos en este apartado que, puesto que por limitaciones de HW no podemos trabajar con coma flotante, aplicamos los factores de atenuación multiplicados por 1024, y después de utilizarlos deshacemos dicho escalado desplazando a la derecha 10 bits. También aquí tenemos que compensar la ganancia de 3dB mencionada anteriormente.

- En el caso de la reverberación, se llama a la función salidaReverberacion() que devuelve la lectura con el eco sumado, lista para ser entregada al DAC.

En cualquiera de las 3 funciones se calcula la energía de la banda que estemos iluminando en el vúmetro. Dependiendo de si esta activada la opción de mostrar la señal de entrada o la de salida, almacenaremos en salidaVumetro la energía correspondiente. Cuando aplicamos la reverberación, filtramos la señal de salida y no la lectura del ADC, para así poder calcular la energía de la señal con el eco añadido.

Por último, la rutina gestiona la iluminación del vúmetro. Puesto que no es posible pintar todas las bandas a la vez, la idea es crear una ilusión óptica, iluminando cada línea de leds durante 3ms. Esto se llevará acabo con un sistema de 2 contadores, filtroIlluminado que indicará cual de las 7 lineas está siendo pintada y contadorVumetro que llevará la cuenta de 24 interrupciones (24x125µs = 3ms) durante las cuales una línea permanece iluminada. Cada vez que se pinte una nueva línea se llamará a la función gestionVumetro() que configura el puerto de salida.

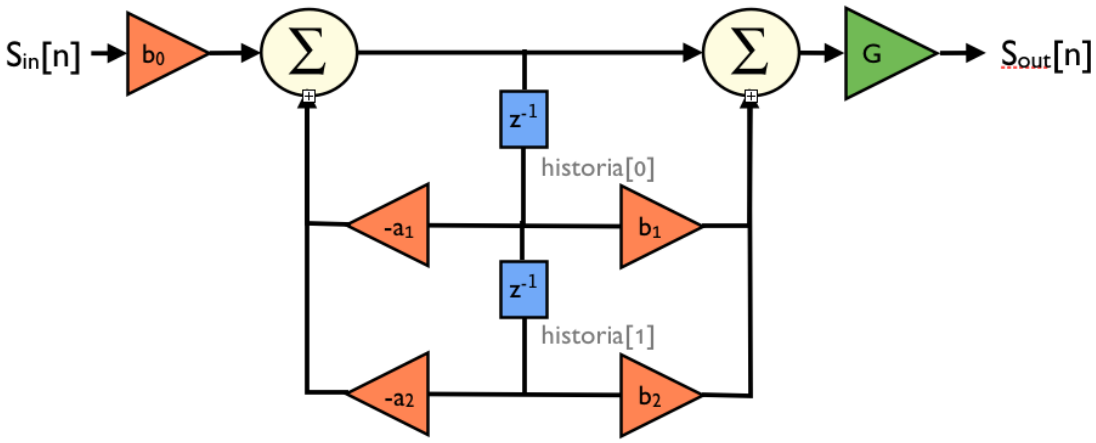
Solo dos excepciones modifican este funcionamiento del vúmetro. Una es cuando se está caracterizando un solo filtro, en ese caso se iluminará continuamente la línea correspondiente al filtro que esta siendo caracterizado. La otra es cuando la opción de mostrar el ecualizador gráfico esta activada (estado[MUESTRAECUALIZADOR] == 1), en ese caso en vez de iluminar conforme a la energía de cada banda, lo haremos conforme al nivel de atenuación seleccionado.

4.2.1 Función calculaSalidasFiltros

void calculaSalidasFiltros (int lectura){
Calcula la salida de cada filtro utilizando "lectura" como señal de entrada.
Guarda la salida de cada filtro en salidasFiltros[]
}

Variables de entrada: lectura.
Variables de salida: void**.**
Variables generales modificadas: salidasFiltros[].

Función calculaSalidasFiltros (): Se encarga del filtrado la muestra "lectura" que le pasamos que se le pasa como parámetro en la llamada al método. El filtrado se realiza se aplicando los cálculos indicados en la siguiente figura:



Los coeficientes a, b y la ganancia se encuentran almacenados en el array coefs[][]. Así mismo en historia[] almacenamos valores de instantes anteriores necesarios para calcular la salida de los filtros.

Cada vez que se llama al método, calcula la salida de los 7 filtros y las almacena en el array salidasFiltros[].

4.2.2 Función salidaReverberacion

int salidaReverberacion (int muestra, int atenuacion, int retardo){
Las variables posicionMuestraActual y posicionMuestraRetardada serviran de indices para trabajar con el array muestrasAlmacenadas[]
Calcula en que posicion del array se encuentra la muestra correspondiente al "retardo" seleccionado
Le suma a "muestra" el eco, que es la muestra de muestrasAlmacenadas[] en la posición posicionMuestraActual dividido por la "atenuacion" seleccionada.
Almacena la muestra con el eco sumado en el array muestrasAlmacenadas[] y la devuelve.
}

Variables de entrada: muestra, atenuacion y retardo
Variables de salida: salidaReverb**.**
Variables generales modificadas: ninguna.

Función salidaReverberacion (): Se encarga de calcular la salida cuando el efecto de reverberación esta activado. Se le pasan como parámetros la muestra a la que sumarle el eco y la atenuación y el retardo anteriormente seleccionados en el menú. Puesto que el retardo máximo es de 1s, debemos tener guardado 1s de música en un array (8000 muestras, ya que muestreamos la señal a 8kHz). Para buscar la muestra que representa el eco dentro del array, configuramos la variable posicionMuestraRetardada como la distancia desde la posición actual a la posición correspondiente a los ms que indica "retardo" (es dedir, 8*retardo, puesto que almacenamos 8 muestras cada ms). Para aplicar la atenuación que afecta al eco, puesto que debemos optimizar al máximo el recurso del tiempo, en vez de hacer una división, hacemos una multiplicación por el factor de atenuación escalado en un factor 256, escalado que desharemos posteriormente desplazando 8 posiciones a la derecha.

4.2.3 Función gestionVumetro

void gestionVumetro (int muestra, WORD teclado, int filtro){
Calcula cuantos leds hay que iluminar en la linea que se esta pintando comparando la energia representada por la variable "muestra" con una una serie de umbrales y lo guarda en nivelEnergia.
Llama a la función configuraPuerto()
}

Variables de entrada: muestra, teclado y filtro
Variables de salida: void**.**
Variables generales modificadas: ninguna.

Función gestionVumetro (): calcula el número de leds que hay que iluminar en la línea que esta siendo pintada para representa la energía de esa banda.

4.2.4 Función configuraPuerto

void configuraPuerto (WORD signalTecladoVumetro, int filtro, int nivelEnergia){
Calcula cuantos leds hay que iluminar en la linea que se esta pintando comparando la energia representada por la variable "muestra" con una una serie de umbrales y lo guarda en nivelEnergia.
Llama a la función configuraPuerto()
}

Variables de entrada: signalTecladoVumetro, filtro y nivelEnergia
Variables de salida: void**.**
Variables generales modificadas: signalSinTeclado.

Función configuraPuerto (): calcula la máscara correspondiente al nivel de energía indicado por "nivelEnergia", y la correspondiente a la banda que se va a pintar, que indica "filtro". Estas se añaden en los bits correspondientes de la salida respetando los 4 primeros bits que se utilizan para excitar el teclado. El contenido de estos 4 primeros bits nos lo indica signalTecladoVumetro, variable que se configura en el método teclado(). Del mismo modo guardamos en la variable general signalSinTeclado el contenido de los bits de las filas y columnas del vúmetro para poder salvar estos bits en la función teclado() cada vez que se configure el puerto de salida. Llamamos al método set16_puertoS() para que saque la señal que hemos configurado con las máscaras.

5 Descripción de las mejoras

Las mejoras que hemos realizado en nuestra práctica so las siguientes:

- Implementación de un sistema hardware para que se escuche el audio (explicado detalladamente en el punto 3 de la memoria).
- Implementación del efecto de reverberación (explicada detalladamente en el punto 4 de la memoria).
- Selección de pre-sets de ecualización para aplicar a nuestra señal.
- Guardado de configuraciones de ecualización que aplicamos a la señal y que queramos conservar para su posterior utilización.
- Guardado de configuraciones de reverberación que aplicamos a la señal y que queramos conservar para su posterior utilización.
- Mostrar la entrada o la salida por la matriz de leds.
- Mostrar la energía de la señal o el ecualizador gráfico en matriz de leds.

A continuación explicaremos detalladamente las siguientes mejoras, a excepción del hardware y del sistema de reverberación, los cuales ya detallamos con anterioridad.

5.1 Selección de pre-sets (Ecualización):

void seleccionPreset(void){
Si pulsamos la tecla F en el menu ecualizacion, accedemos a la opcion de seleccionar unos ajustes predeterminados para la ecualizacion de nuestra señal. Seleccionaremos uno u otro pre-set pulsando las teclas del 1 al 9. Tambien podemos seleccionar alguna configuracion que hayamos guardado pulsando D.
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: nivel[] y estado[].

Estos pre-sets tienen unos valores asignados para cada banda almacenados en memoria (que dan a cada posición de nivel[] un valor) que aplicaremos a nuestra señal, consiguiendo asi el efecto deseado.

Para cargar una configuración que hallamos almacenado, tras pulsar D debemos seleccionar la posición en la que está guardada dicha configuración.

5.2 Guardado y carga de configuraciones de ecualización.

void GestionEcualizacion (void){
Dentro del menú ecualización tenemos la opción de, o bien guardar la configuracion actual (pulsando D), o bien cargar una configuracion guardada con anterioridad (pulsando F)
}

Variables de entrada: void**.**
Variables de salida: void**.**
Variables generales modificadas: nivel[] y nivelesGuardados[][].

Función GestionEcualizacion (): Muestra el submenú de ecualización. Si pulsamos D, salvamos la configuración de ecualización actual. Para ello pasamos en contenido de nivel[] a nivelesGuardados[][]. Podemos seleccionar la posición en la que queremos guardar nuestra configuración (de la 1 a la 9) pulsando las teclas de la 1 a la 9.

Para cargar una configuración guardada con anterioridad, pulsamos F en el submenú ecualización accediendo a la selección de presets. Desde aquí, tal y como explicamos en el punto anterior, pulsando D, seleccionamos la opción de configuraciones guardadas, y pulsando la tecla correspondiente a la posición en la que guardamos la configuración requerida, cargamos la misma, asignandole a nivel[] los valores de la configuración que se almacenan en nivelesGuardados[][].

5.3 Guardado y carga de configuraciones de reverberación.

void GestionReverb (void){
Muestra el submenú de reverberación y espera a que se seleccione una opcion. Podemos, o bien guardar la configuracion actual (pulsando D), o bien cargar una configuracion guardada con anterioridad (pulsando E)
}

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: estado[] y reverbGuardadas[][].

Función GestionReverb(): muestra el menú de reverberación. Si pulsamos D, salvamos la configuración de reverberación actual. Para ello pasamos en contenido de estado[] (en las posiciones ATTREVERB y TIEMPORTARDO) a reverbGuardadas[][]. Podemos seleccionar la posición en la que queremos guardar nuestra configuración (de la 1 a la 9) pulsando las teclas de la 1 a la 9.

Para cargar una configuración guardada con anterioridad, pulsamos E en el submenú reverberación. Posteriormente, pulsando la tecla correspondiente a la posición en la que guardamos la configuración requerida, cargamos la misma, asignandole a estado[ATTREVERB] y a estado[TIEMPORTARDO] los valores de la configuración que se almacenan en reverbGuardadas[][].

5.4 Mostrar entrada o salida por la matriz de leds.

```
void rutina_tout0 (void){
...
if (estado[MUESTRAENTRADA]== 0){
   salidaVumetro += ((salidasFiltros[filtroIluminado]*ganancia[nivel[filtroIluminado]])>>10) *
                    ((salidasFiltros[filtroIluminado]*ganancia[nivel[filtroIluminado]])>>10);
}else {
   salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
}
...
```

Variables de entrada: void.
Variables de salida: void**.**
Variables generales modificadas: estado[] y reverbGuardadas[][].

Esta mejora consiste en mostrar, o la entrada del ADC o la salida hacia el DAC por la matriz de leds. Para ello desde el menú de caracterización, ecualización o reverberación, y sus submenús, pulsado la tecla B, configuramos la posición estado[MUESTRAENTRADA], que según sea 1 o 0 se mostrará la entrada o la salida respectivamente.

Como anteriormente apuntamos, al ser señales distintas, las energías de ambas se calculan de forma diferente en la rutina de interrupción. Para ello, evaluamos la posición estado[MESTRAENTRADA] para que en la rutina se calcule la energía de la señal correspondiente al vúmetro. En el cuadro gris anterior mostramos los 2 cálculos de energía en la función de ecualización.

A la hora de configurar el puerto de salida para la iluminación de la matriz de leds, no existen diferencias de código. La única diferencia es la energía que se utiliza para decidir el número de columnas que se iluminan.

5.5 Mostrar energía o el ecualizador gráfico en la matriz de leds.

void gestionEcualizGrafico (int nivel, WORD teclado, int filtro){
Calcula cuantos leds hay que iluminar en la banda que se esta pintando según el nivel de atenuación seleccionado.
Llama a la función configuraPuerto()
}

Variables de entrada: signalTecladoVumetro, filtro y nivelEnergia
Variables de salida: void**.**
Variables generales modificadas: signalSinTeclado.

Función gestionEcualizGrafico ():calcula el número de leds que hay que iluminar en la línea que esta siendo pintada para representar el nivel de atenuación seleccionado para esa banda. De esta manera representaremos el ecualizador gráfico en la matriz de leds. Puesto que disponemos de 8 leds y de 16 niveles de atenuación, utilizaremos un led para cada 2 niveles. Esta función es la equivalente a la que calcula el nivel de energía cuando representamos la señal.

Para seleccionar esta opción, pulsamos en el menú de ecualización o sus submenús, con la tecla E configuramos valor de estado[MUESTRAECUALIZADOR] que según sea 0 o 1, sacaremos la energía o el ecualizador gráfico respectivamente.

6 Principales problemas encontrados

El problema más importante que nos encontramos durante el desarrollo de la práctica fue el desconocimiento que los 2 miembros de la pareja de trabajo teníamos sobre el lenguaje de programación C. Esto provocó que el primer mes del curso estuviésemos totalmente perdidos y con la sensación de no avanzar. Consecuencia de esto tuvimos un retraso muy grande en el plan de trabajo establecido en el manual de la práctica, con lo que no fuimos capaces de alcanzar la mayoría de los objetivos propuestos para los hitos 1 y 2. Para solucionar este problema dedicamos muchas horas de trabajo tanto en casa como en el laboratorio, llegando incluso a asistir alrededor de unas 40 horas de laboratorio en turno extra. De este modo, a base de horas, conseguimos tener nuestra práctica terminada en la fecha de entrega del hito 3, incluidas todas las mejoras, solo a falta de las tareas de depuración.

En cuanto a aspectos más específicos de la práctica, nos encontramos con muchos problemas a la hora de implementar el banco de filtros, que inicialmente realizamos con la fórmula proporcionada en el enunciado. Con esta implementación los 2 filtros de frecuencias más bajas eran muy inestables, provocando muchas veces que apareciese una señal portadora de alta frecuencia modulada por una señal sinusoidal distinta a la que introducíamos a la entrada. Puesto que no fuimos capaces de solucionar dichos errores decidimos implementar los filtros según el esquema del enunciado, decisión que soluciono todos nuestros problemas.

Otro de los problemas más importantes nos lo encontramos a la hora de implementar el vúmetro, ya que en nuestro primer diseño pretendíamos filtrar dos veces la señal, una para configurar la salida que entregamos al DAC y otra para calcular la energía de las bandas. Esto provocó que al programa no le diese tiempo a realizar todas las operaciones necesarias, así que las planes de pruebas daban resultados muy extraños, e incluso lo que funcionaba anteriormente ya no lo hacia, por lo que nos costó mucho detectar la causa de este error.

6.1 Calificación personal

Los aspectos que creemos son importantes en la calificación de nuestra práctica son:

1. La forma de afrontar el mal comienzo, por culpa del gran desconocimiento en cuanto al lenguaje de programación C, a base de trabajo duro y muchas horas de laboratorio en turno extra, incluso llegando a tener la práctica y las mejoras terminadas en el hito 3.
2. El cuidado de la estructura del código, intentando dotarlo de una estructura de fácil comprensión.
3. Se ha tratado de comentar de la mejor manera posible el código, llegando a tener alrededor del 80% de líneas comentadas, sin llegar a parafrasear el código.
4. Hemos tratado de realizar una memoria muy completa a la vez que concisa, de forma que las funciones más importantes han sido comentadas, incluyendo muchas gráficas y diagramas, cuidando al máximo la estructura, para que dicha memoria sea fácilmente legible.

Módulos	Puntuación máxima (sobre 100)	Puntuación esperada (sobre 100)
Requisitos mínimos	**35/40**	25
Memoria	**15**	12
Software	**25**	20
Hardware	**5**	4
Mejoras	**24**	24
TOTAL	**104**	80~85

7 Manual de usuario

Al iniciar el sistema, tenemos 3 posibles opciones que escoger:

- Si pulsamos 1, entramos en el submenú de caracterización de filtros. Tras seleccionar esta opción se nos abre un submenú en el que, las teclas de 1 al 7 se corresponden con los 7 filtros disponibles, la tecla F selecciona todos los filtros, con la tecla B mostramos la entrada del ADC o la salida hacia el DAC y con la tecla A volvemos al menú anterior.

- Si pulsamos 2, entramos en el submenú de ecualización de la señal. Tras seleccionar esta opción se nos abre un submenú en el que, las teclas de 1 al 7 se corresponden con las 7 bandas de frecuencia, la tecla F selecciona los pre-sets de ecualización, con la tecla B mostramos la entrada del ADC o la salida hacia el DAC, con la tecla A mostramos las opciones que tenemos activadas (es decir nos muestra el "estado" en el que nos encontramos), con la tecla E mostramos la energía de la señal o la ecualización de la misma, la tecla C desactiva la ecualización, la tecla D almacena la configuración actual de la ecualización en memoria (tras pulsar la D pulsamos las teclas 1 a 9 para seleccionar la posición en la que queremos guardar nuestra configuración) y la tecla 0 muestra los niveles de ecualización de las bandas de frecuencia.

  Una vez seleccionada una de las bandas de ecualización, pulsando 8 y 9, subimos y bajamos respectivamente el nivel de ecualización. La tecla A nos devuelve al submenú anterior y las teclas E y B tienen la misma función que expliqué antes.

  Si habíamos pulsado la F (selección de pre-sets), si ahora pulsamos las teclas 1 a 9, seleccionamos uno de los pre-sets de ecualización prestablecidos. Si pulsamos D accedemos a las configuraciones guardadas (tras pulsar la D pulsamos las teclas 1 a 9 para seleccionar la posición de la que queremos cargar la configuración). Con la tecla A volvemos al menú anterior y las teclas E y B tienen las funciones anteriormente descritas.

- Si pulsamos 3, entramos en el submenú del efecto de reverberación. Tras seleccionar esta opción se nos abre un submenú en el que, la tecla 1 activa la reverberación y la tecla 2 la desactiva. Con la tecla B mostramos la entrada del ADC o la salida hacia el DAC y con la tecla 0 mostramos la configuración actual de la reverberación. La tecla D almacena la configuración actual de la reverberación en memoria (tras pulsar la D pulsamos las teclas 1 a 9 para seleccionar la posición en la que queremos guardar nuestra configuración). La tecla E nos permite cargar una configuración de la reverberación almacenada en memoria (tras pulsar E pulsamos las teclas 1 a 9 para seleccionar la posición de memoria desde la que queremos cargar la configuración de la reverberación deseada). Y la tecla A nos devuelve al menú anterior.

  Si hemos pulsado 1, ahora con las teclas 1 a 8 seleccionamos la atenuación que queremos darle a la reverberacion, y posteriormente las teclas 0 a 9 para seleccionar el retardo a aplicarle a la misma. La tecla A me devuelve al submenú anterior.

8 Bibliografía

[1] Bian W. Kernighan y Dennis M. Ritchie “El lenguaje de programación C”. Segunda edición.

[2] Información general del Laboratorio de Sistemas Electrónicos Digitales (LSED).
http://lsed.die.upm.es

[3] R. San Segundo et al, “Introducción a los Sistemas Digitales con el micro-controlador MCF5272”, Ed. Marcombo S.A., 2006a.
Sobre todo los apartados dedicados a los tutoriales realizados y al uso y configuración del DAC y ADC de la plataforma.

[4] *R. San Segundo et al, “Entorno de desarrollo EDColdFire V3.0” Ed. ETSIT, 2006.*
Describe el manejo del programa EDColdFire.
http://lsed.die.upm.es/public/docs/tutorialBasicoEDColdFire_v7.htm
http://lsed.die.upm.es/public/docs/EDColdFire_ayuda.htm

[5] *Javier Ferreiros et al, “Aspectos Prácticos de Diseño y Medida en Laboratorios de Electrónica”,Ed. ETSIT. 2002*
Capítulo 4.

[6] *J.M Montero et al. “Dudas y preguntas frecuentes del LSED Programación en C”.*
Muy útil en las primeras fases de nuestro desarrollo
http://lsed.die.upm.es/

[7] *J.M Montero et al. “Transparencias sobre programación en C para alumnos de Java”.*
Muy útiles para aprender a programar.
http://lsed.die.upm.es/

[8] *“Hojas de características del 74HC244”.*
http://lsed.die.upm.es/

[9] *“Hojas de características del amplificador LM386”.*
http://lsed.die.upm.es/

[10] *Enunciado de la Práctica del Laboratorio de Circuitos Electrónicos (LCEL), Curso 2011-2012*
http://neirol-vm-50.die.upm.es/~profes/lcel/1112/public/docs/enunciado1112.pdf

[11] *Enunciado de la práctica estándar del Laboratorio de Sistemas Electrónicos Digitales (LSED),*
*Curso 2008-2009, “*Piano Hero*: un juego musical basado en el MCF5272”.*
http://neirol.die.upm.es/~profes/lsed/0809/public/docs/Enunciado_lsed0809-v1_4.pdf

[12] *Esquemas de salida y entrada del entrenador.*
http://lsed.die.upm.es/public/docs/salidasEntrenador.jpg
http://lsed.die.upm.es/public/docs/entradasEntrenador.jpg

[13] *Fotos sobre cómo conectar el HW de la práctica a la placa TL04.*
http://lsed.die.upm.es/public/fotos/hw1.jpg
http://lsed.die.upm.es/public/fotos/hw2.jpg

9 ANEXO I: Código del programa del proyecto final


```
//-----------------------------------------------------------------
// LT-21.c
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//-----------------------------------------------------------------


//---------------------------
//INCLUSION DE OTROS FICHEROS
//---------------------------

#include "m5272adc_dac.c"
#include "m5272lib.c"
#include "m5272lib.h"
#include "m5272gpio.c"
#include "m5272.h"
#include "Vumetro.c"
#include "Menu.c"
#include "Filtros.c"
#include "Reverberacion.c"

//--------------------------------------
//DEFINICIÓN DE CONSTANTES Y DIRECCIONES
```

```
//--------------------------------------

#define V_BASE 0x40                                     // Dirección de inicio de los vectores de interrupción
#define DIR_VTMR0 4*(V_BASE+5)                          // Dirección del vector de TMR0
#define FREC_INT 8000                                   // Frecuencia de interrupción TMR0 = 8000 Hz (cada 125 us)
#define PRESCALADO 2
#define CNT_INT1 MCF_CLK/(FREC_INT*PRESCALADO*16)       // Valor de precarga del temporizador de interrupciones TRR0
#if CNT_INT1>0xFFFF
#error PRESCALADO demasiado pequeño para esa frecuencia (CNT_INT1>0xFFFF)
#endif

#define BORRA_REF 0x0002                          // Valor de borrado de interrupciones pendientes de tout1 para TER0
volatile ULONG cont_retardo;

#define FONDO_ESCALA 0xFFF                                                  // Valor de lectura máxima del ADC
#define V_MAX 5                                                             // Valores de tensión máxima del ADC

//-------------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------

//-------------------------------------------------------
// void bucleMain(void)
// Descripción:
//      Función del programa principal.
//      Llama a la función GestionMenuPrincipal
//      que nos mostrará el menú.
//-------------------------------------------------------
void bucleMain(void){
        GestionMenuPrincipal ();                                    //muestra el menú en pantalla
}

//-------------------------------------------------------
// void __init(void)
// Descripción:
//      Función por defecto de inicialización del sistema
//-------------------------------------------------------
void __init(void){
        DAC_ADC_init();                                             // Inicializa el DAC y el ADC.
        mbar_writeByte(MCFSIM_PIVR,V_BASE);                         // Fija comienzo de vectores de interrupción en V_BASE.
```

```
        ACCESO_A_MEMORIA_LONG(DIR_VTMR0)= (ULONG)_prep_TOUT0;   // Escribimos la dirección de la función para TMR0
        mbar_writeShort(MCFSIM_TMR0, (PRESCALADO-1)<<8|0x3D);   // TMR0: PS=2-1=1 CE=00 OM=1 ORI=1 FRR=1 CLK=10 RST=1
        mbar_writeShort(MCFSIM_TCN0, 0x0000);                   // Ponemos a 0 el contador del TIMER0
        mbar_writeShort(MCFSIM_TRR0, CNT_INT1);                 // Fijamos la cuenta final del contador
        mbar_writeLong(MCFSIM_ICR1, 0x8888C888);                // Marca la interrupción del TIMER0 como no pendiente
                                                                // y de nivel 4.
        cont_retardo = 0;
        sti();                                                  // Habilitamos interrupciones
}

//-----------------------------------------------------
// Definición de rutinas de atención a las interrupciones
//-----------------------------------------------------
void rutina_int2(void){}
void rutina_int3(void){}
void rutina_int1(void){}
void rutina_int4(void){}
void rutina_tout0(void){
        static int filtroIluminado =0;                  // variable que guarda cual de las 7 bandas debe pintar.
        static int contadorVumetro=0;                   // variable que sirve como contador de las 24 interrupciones de
                                                        // 8kHz durante las que se mantiene una linea de leds pintada.
        static int salidaVumetro=0;                     // variable para guardar el sumatorio de la energia de la banda
                                                        // en las 24 interrupciones
        int lectura = 0;                                // variable para guardar la lectura del ADC
        int salida = 0;                                 // variable para guardar la salida que entregaremos al DAC

        mbar_writeShort(MCFSIM_TER0,BORRA_REF);         // Reset del bit de fin de cuenta.

        lectura = ADC_dato();                           // lee del ADC
        if (lectura >= 0x800){                          // Hace la extensión de signo
                lectura |= 0xFFFFF000;
        }

        //-------------------------------------------------
        //CONFIGURACIÓN DE LA SALIDA QUE SE ENTREGA AL DAC
        //-------------------------------------------------
        salida = 0;

        //Salida caracterización
        //-----------------------
```

```
if(estado[FUNCION] == CARACTERIZACION){         // si está dentro del menú de caracterización calcula la salida
                                                // correspondiente.
        calculaSalidasFiltros(lectura);         // filtra la señal entregada por el ADC
        if (estado[CARACTERIZACION] == TODOS_FILTROS){
                int i;
                for(i=0; i<7; i++){
                        salida += salidasFiltros[i];    // si está caracterizando todos los filtros a la vez,
                                                        // salida es la suma las 7 componentes.
                }
                salida = (salida*1500)>>11;     // divide entre raiz(2) para compensar la ganancia de 3 dB. En
                                                // realidad multiplica por raiz(2)/2 escalado 1024 para no
                                                // trabajar con decimales.
                salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];     //guarda
                                                //la energia de la componente correspondiente al filtro que
                                                //estamos pintando en el vúmetro
        } else{
                salida = salidasFiltros[estado[CARACTERIZACION]-1];     // si está caracterizando un solo filtro,
                                                                        //salida es solo la componente
                                                                        //correspondiente al filtro seleccionado.
                if (estado[MUESTRAENTRADA]== 0){
                        salidaVumetro += salida * salida;
                }else {
                        salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
                }
        }

//Salida ecualización
//--------------------
} else if(estado[FUNCION] == ECUALIZACION){     // si está dentro del menú de ecualización calcula la salida
                                                //correspondiente.
        int i;

        calculaSalidasFiltros(lectura);         // filtra la señal entregada por el ADC
        for(i =0; i<7; i++){
                salida += (salidasFiltros[i]*ganancia[nivel[i]])>>10;   // salida es la suma las 7 componentes,
                                                                        //una de cada filtro, cada una atenuada
                                                                        //según el nivel de atenuación
                                                                        //seleccionado.
        }
```

```
                salida = (salida*1500)>>11;     //divide entre raiz(2) para compensar la ganancia de 3 dB. En realidad
                                                //multiplica por raiz(2)/2 escalado 1024 para no trabajar con decimales.
                if (estado[MUESTRAENTRADA]== 0){
                        salidaVumetro += ((salidasFiltros[filtroIluminado]*ganancia[nivel[filtroIluminado]])>>10) *
((salidasFiltros[filtroIluminado]*ganancia[nivel[filtroIluminado]])>>10);       //guarda la energia de la componente
                                                                                //ecualizada correspondiente al filtro
                                                                                //que estamos pintando en el vúmetro
                }else {
                        salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
                }

        //Salida reverberación
        //--------------------
        } else if(estado[FUNCION] == REVERBERACION){    // si está dentro del menú de reverberación calcula la salida
                                                        //correspondiente.
                salida = salidaReverberacion(lectura, estado[ATTREVERB], estado[TIEMPORETARDO]);
                                                        //llama a la función definida en Reverberacion.c que calcula la
                                                        //salida
                if (estado[MUESTRAENTRADA]== 0){
                        calculaSalidasFiltros(salida);  // filtra la salida con el eco para poder seleccionar la energía
                                                        //de la banda que está pintando en el vúmetro
                        salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
                }else {
                        calculaSalidasFiltros(lectura);
                        salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
                }

        //Salida por defecto
        //------------------
        } else{                         // si está en el menú principal, entrega al DAC la lectura del ADC
                calculaSalidasFiltros(lectura); // filtra para poder seleccionar la energía de la banda que está pintando
                                                //en el vúmetro
                salida = lectura;
                salidaVumetro += salidasFiltros[filtroIluminado] * salidasFiltros[filtroIluminado];
        }

        //--------
        //VÚMETRO
        //--------
        if (contadorVumetro<24){                //si el contador de 24 interrupciones es menor de 24, se incrementa.
```

```
                contadorVumetro++;
        }else{                                  // cuando el contador valga 24, pinta la siguiente banda del vúmetro
                if (estado[MUESTRAECUALIZADOR]== 1){
                        gestionEcualizGrafico(nivel[filtroIluminado] , signalTeclado, filtroIluminado);
                                                // si está seleccionada la opción de mostrar el ecualizador gráfico en el
                                                //vúmetro, llama al método que lo muestra.
                }
                else if (estado[CARACTERIZACION] != TODOS_FILTROS){    // si está caracterizando una sola banda de
                                                                       //frecuancia, pinta constantemente dicha banda.
                        gestionVumetro(salidaVumetro , signalTeclado, estado[CARACTERIZACION]-1);
                }
                else{
                        gestionVumetro(salidaVumetro , signalTeclado, filtroIluminado);
                                                                // en todas las demás funciones pinta la energía de cada
                                                                //banda. Cada 24 interrupciones pinta una banda.
                }
                contadorVumetro = 0;                    //contador de 24 interrupciones
                filtroIluminado++;                      //después de pintar la banda correspondiente, incrementa el
índice que indica que banda que debe pintarse.
                salidaVumetro =0;
        }
        if (filtroIluminado == 7){                      // si ya ha pintado las 7 bandas, vuelve a la primera banda.
                filtroIluminado = 0;
        }

        salida += 0x0800;
        DAC_dato(salida);

}
void rutina_tout1(void){}
void rutina_tout2(void){}
void rutina_tout3(void){}
```


```
//-------------------------------------------------------------
// Menú.c
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//-------------------------------------------------------------

//---------------------------
//INCLUSION DE OTROS FICHEROS
//---------------------------
#include "teclado.c"
#include "m5272lib.c"
#include "m5272gpio.c"
#include "m5272.h"

//-------------------------
//DEFINICIÓN DE CONSTANTES
//-------------------------
#define TODOS_FILTROS 0
#define FUNCION 0
#define CARACTERIZACION 1
#define ECUALIZACION 2
#define REVERBERACION 3
#define MUESTRAENTRADA 4
#define MUESTRAECUALIZADOR 5
#define ATTREVERB 6
#define TIEMPORETARDO 7
```

```
//----------------------------------------
//DECLARACIÓN DE VARIABLES Y ARRAYS GENERALES
//----------------------------------------
int estado[8]={0,0,0,0,0,0,0,0};// array que indica el estado en el que está, es decir, que opciones se han seleccionado
                                //en el menú
                                // PRIMERA POSICIÓN (funcion):(1) gestión caract, (2) gestión ecualiz...
                                // SEGUNDA POSICIÓN (filtro seleccionado):(1)filtro 1,(2)filtro 2...(0) todos los filtros
                                // TERCERA POSICIÓN (banda seleccionada para ecualizar): (1)banda 32Hz,(2)banda 64Hz, ...
                                // CUARTA POSICIÓN (estado reverberacion): (1) activada, (0) desactivada.
                                // QUINTA POSICIÓN (muestra entrada/salida): (0) muestra salida, (1) muestra entrada
                                // SEXTA POSICIÓN (muestra ecualizador grafico/ energia): (0) muestra energia (1) muestra
                                // ecualizador grafico
                                // SEPTIMA POSICIÓN (nivel atenuación de la reverberación)
                                // OCTAVA POSICIÓN (tiempo retardo de la reverberacióm).
int flagsVolver[2] = {0,0};     // dos flags utilizados para volver atrás en los submenús.

// VARIABLES DE ECUALIZACIÓN
int ganancia[16] = {1024,790,610,471,364,281,217,167,129,99,77,59,46,35,27,21}; // valores de las ganancias que se
                                                                                // aplicaran en la ecualización.
int nivel[7] = {0,0,0,0,0,0,0};         // nivel de ganancia de las 7 bandas de frecuencia. nivel 0 = 1024, nivel 15 = 21
int nivelesGuardados[9][7] = {{0,0,0,0,0,0,0},  // array que funciona como memoria en la opción de guardar la
                              {0,0,0,0,0,0,0},   // configuración. Es posible guardar 9 configuraciones.
                              {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}};

// VARIABLES DE REVERBERACIÓN
int reverbGuardadas[9][2] = {{0,0},     // array que funciona como memoria en la opción de guardar la configuración. Es
                             {0,0},     // posible guardar 9 configuraciones.
                             {0,0},
                             {0,0},
                             {0,0},
                             {0,0},
                             {0,0},
                             {0,0}};
```

```
//----------------------
//DECLARACIÓN DE MÉTODOS
//----------------------

void GestionMenuPrincipal(void);

void GestionCaracterizacion(void);

void GestionEcualizacion(void);
void seleccionNivel(int nivel);
void seleccionPreset(void);
void muestraEstadoEcualizacion(void);

void GestionReverb(void);
void selectAtenuacion(void);
void selectRetardo(void);

//-------------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------

//-----------------------------------------------------
// void GestionMenuPrincipal(void)
// Descripción:
//   Muestra el menú principal y nos permite escoger una de las
//   tres opciones: CARACTERIZACIÓN DE FILTROS, ECUALIZACIÓN GRÁFICA,
//                  INCORPORACIÓN DE REVERBERACION SIMPLE.
//-----------------------------------------------------
void GestionMenuPrincipal (void){
        char tecla;
        estado[FUNCION] = 0;                    // inicializa el array estado a cero
        estado[CARACTERIZACION] = 0;
        estado[ECUALIZACION] = 0;
        estado[REVERBERACION] = 0;
        estado[MUESTRAENTRADA] = 0;
        estado[MUESTRAECUALIZADOR] = 0;
        estado[ATTREVERB] = 0;
        estado[TIEMPORETARDO] = 0;
```

```
	output ("-------------------------------------------------------------------\n");
	output ("|  BIENVENIDO AL SISTEMA DE PROCESAMIENTO DIGITAL DE AUDIO LT-21  |\n");
	output ("-------------------------------------------------------------------\n\n");
	output ("Seleccione una de las siguientes opciones:\n");
	output ("\t1 - Caracterización de filtros.\n");
	output ("\t2 - Ecualizador gráfica.\n");
	output ("\t3 - Incorporación de reverberación.\n");
	tecla = teclado();
	switch(tecla){
		case '1':
			output ("Ha seleccionado: ");
			output ("CARACTERIZACIÓN DE FILTROS\n\n");
			estado[FUNCION] = CARACTERIZACION;
			while(estado[FUNCION] == CARACTERIZACION){
				GestionCaracterizacion();
			}
			break;
		case '2':
			output ("Ha seleccionado: ");
			output ("ECUALIZACIÓN GRÁFICA\n\n");
			estado[FUNCION] = ECUALIZACION;
			while (estado[FUNCION] == ECUALIZACION){
				GestionEcualizacion();
			}
			break;
		case '3':
			output ("Ha seleccionado: ");
			output ("INCORPORACIÓN DE REVERBERACION SIMPLE\n\n");
			estado[FUNCION] = REVERBERACION;
			while(estado[FUNCION] == REVERBERACION){
				GestionReverb();
			}
			break;
		default:
			output ("Tecla no válida.\n\n\n");
			break;
	}
}
```

```
//-----------------------------
// SUBMENÚ DE CARACTERIZACIÓN
//-----------------------------



//-------------------------------------------------------
// void GestionCaracterizacion(void)
// Descripción:
//   Muestra el submenú de "Caracterizacón principal" y nos permite
//   seleccionar el filtro (individualmente o todo el banco de filtros)
//   que queremos utilizar
//-------------------------------------------------------
void GestionCaracterizacion(void){
        char tecla;

        output ("\t1,7 - Seleccione el filtro que quiere caracterizar (1-7).\n");
        output ("\tF - Todos los filtros.\n");
        output ("\tA - Volver.\n\n");
        output ("\tB - VúMETRO: Mostrar salida/Mostrar entrada.\n\n");

        tecla = teclado();
        switch(tecla){
                case '1':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 1 \n\n");
                        estado[CARACTERIZACION]= 1;
                        break;
                case '2':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 2 \n\n");
                        estado[CARACTERIZACION]= 2;
                        break;
                case '3':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 3 \n\n");
                        estado[CARACTERIZACION]= 3;
                        break;
                case '4':
                        output ("Ha seleccionado: ");
```

```
                        output ("FILTRO 4 \n\n");
                        estado[CARACTERIZACION]= 4;
                        break;
                case '5':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 5 \n\n");
                        estado[CARACTERIZACION]= 5;
                        break;
                case '6':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 6 \n\n");
                        estado[CARACTERIZACION]= 6;
                        break;
                case '7':
                        output ("Ha seleccionado: ");
                        output ("FILTRO 7 \n\n");
                        estado[CARACTERIZACION]= 7;
                        break;
                case 'F':
                        output ("Ha seleccionado: ");
                        output ("TODOS LOS FILTROS\n\n");
                        estado[CARACTERIZACION]= TODOS_FILTROS;
                        break;
                case 'B':
                        if (estado[MUESTRAENTRADA] == 0){
                                estado[MUESTRAENTRADA]= 1;
                                output ("VUMETRO: muestra entrada\n\n");
                        }else {
                                estado[MUESTRAENTRADA]= 0;
                                output ("VUMETRO: muestra salida\n\n");
                        }
                        break;
                case 'A':
                        estado[FUNCION] = 0;
                        break;
                default:
                        output ("Tecla no válida.\n\n");
                        break;
        }
}
```

```
//---------------------------
// SUBMENÚS DE ECUALIZACIÓN
//---------------------------



//-------------------------------------------------------------
// void GestionEcualizacion(void)
// Descripción:
//   Muestra el submenú de "Gestión ecualización" y nos permite
//   seleccionar la banda de frecuencias que queremos ecualizar,
//   seleccionar una configuración preestablecida, mostrar la
//   configuración actual o desactivarla.
//   Además siempre tendremos la opción de seleccionar que
//   queremos visualizar en la matriz de leds, la energía de
//   las bandas o el ecualizador gráfico.
//-------------------------------------------------------------
void GestionEcualizacion(void){
        int i;
        char tecla;
        output ("Seleccione que ecualización desea implementar.\n");
        output ("\t1,7 - Ecualización personalizada: Seleccione la banda que quiere ecualizar (1-7).\n");
        output ("\tF - Presets de ecualización.\n");
        output ("\tD - Guardar la configuración.\n");
        output ("\t0 - Mostrar estado actual.\n");
        output ("\tC - Desactivar ecualización.\n");
        output ("\tA - Volver.\n\n");
        output ("\tE - VÚMETRO: Mostrar ecualizador gráfico/Mostrar energía.\n");
        output ("\tB - VÚMETRO: Mostrar salida/Mostrar entrada.\n\n");

        tecla = teclado();
        switch(tecla){
                case '1':
                        output ("Ha seleccionado: ");
                        output ("Banda 32Hz \n\n");
                        estado[ECUALIZACION]= 1;
                        flagsVolver[0]= 0;
                        while(flagsVolver[0] == 0){
```

```
                seleccionNivel(estado[ECUALIZACION]);
        }
        break;
case '2':
        output ("Ha seleccionado: ");
        output ("Banda 64Hz \n\n");
        estado[ECUALIZACION]= 2;
        flagsVolver[0]= 0;
        while(flagsVolver[0]== 0){
                seleccionNivel(estado[ECUALIZACION]);
        }
        break;
case '3':
        output ("Ha seleccionado: ");
        output ("Banda 125Hz \n\n");
        estado[ECUALIZACION]= 3;
        flagsVolver[0]= 0;
        while(flagsVolver[0]== 0){
                seleccionNivel(estado[ECUALIZACION]);
        }
        break;
case '4':
        output ("Ha seleccionado: ");
        output ("Banda 250Hz \n\n");
        estado[ECUALIZACION]= 4;
        flagsVolver[0]= 0;
        while(flagsVolver[0]== 0){
                seleccionNivel(estado[ECUALIZACION]);
        }
        break;
case '5':
        output ("Ha seleccionado: ");
        output ("Banda 500Hz \n\n");
        estado[ECUALIZACION]= 5;
        flagsVolver[0]= 0;
        while(flagsVolver[0]== 0){
                seleccionNivel(estado[ECUALIZACION]);
        }
        break;
case '6':
```

```
                output ("Ha seleccionado: ");
                output ("Banda 1kHz \n\n");
                estado[ECUALIZACION]= 6;
                flagsVolver[0]= 0;
                while(flagsVolver[0] == 0){
                        seleccionNivel(estado[ECUALIZACION]);
                }
                break;
        case '7':
                output ("Ha seleccionado: ");
                output ("Banda 2kHz \n\n");
                estado[ECUALIZACION]= 7;
                flagsVolver[0]= 0;
                while(flagsVolver[0]== 0){
                        seleccionNivel(estado[ECUALIZACION]);
                }
                break;
        case '0':
                muestraEstadoEcualizacion();
                break;
        case 'F':
                flagsVolver[0] = 0;
                while(flagsVolver[0] == 0){
                        seleccionPreset();
                }
                break;
        case 'D':
                output ("¿Dónde desea guardarlo? Seleccione una de las 9 posiciones de memoria.\n\n");
                tecla = teclado();
                while ((tecla < '1') || (tecla > '9')){
                        output ("Tecla no válida\n\n");
                        tecla = teclado();
                }
                output ("Ha seleccionado: ");
                outch (tecla);
                output("\n\n");
                nivelesGuardados[tecla-'0'][0] = nivel [0];     //guarda la configuración actual en la posicion
                                                                //de memoria seleccionada
                nivelesGuardados[tecla-'0'][1] = nivel [1];
                nivelesGuardados[tecla-'0'][2] = nivel [2];
```

```
                nivelesGuardados[tecla-'0'][3] = nivel [3];
                nivelesGuardados[tecla-'0'][4] = nivel [4];
                nivelesGuardados[tecla-'0'][5] = nivel [5];
                nivelesGuardados[tecla-'0'][6] = nivel [6];
                break;
        case 'C':
                output ("ECUALIZACIÓN DESACTIVADA\n\n");
                for(i=0; i<7; i++){                             // para desactivarla borra el array nivel
                        nivel[i]=0;
                }
                muestraEstadoEcualizacion();
                break;
        case 'E':
                if (estado[MUESTRAECUALIZADOR]== 0){
                        estado[MUESTRAECUALIZADOR]= 1;
                        output ("VUMETRO: muestra ecualizador gráfico\n\n");
                }else{
                        estado[MUESTRAECUALIZADOR]= 0;
                        output ("VUMETRO: muestra energía\n\n");
                }
                break;
        case 'B':
                if (estado[MUESTRAENTRADA]== 0){
                        estado[MUESTRAENTRADA]= 1;
                        output ("VUMETRO: muestra entrada\n\n");
                }else {
                        estado[MUESTRAENTRADA]= 0;
                        output ("VUMETRO: muestra salida\n\n");
                }
                break;
        case 'A':
                estado[FUNCION] = 0;
                break;
        default:
                output ("Tecla no válida.\n\n");
                break;
        }
}
```

```
//-------------------------------------------------------
// void seleccionNivel(int nivel)
// Descripcion:
//      Permite seleccionar el nivel de ecualización con
//      las teclas 8 y 9 (subir y bajar respectivamente)
//      y muestra el estado de ecualizacion de cada banda.
//      Además siempre tendremos la opción de seleccionar que
//      queremos visualizar en la matriz de leds, la energía de
//      las bandas o el ecualizador gráfico.
//-------------------------------------------------------
void seleccionNivel(int bandaNivel){
        char tecla;

        output ("Seleccione el nivel de ecualización: nivel 0 mínima atenuación - nivel 15 máxima atenuación\n");
        output ("\t8 - Subir nivel.\n");
        output ("\t9 - Bajar nivel.\n");
        output ("\tA - Volver.\n\n");
        output ("\tE - VÚMETRO: Mostrar ecualizador gráfico/Mostrar energía.\n");
        output ("\tB - VÚMETRO: Mostrar salida/Mostrar entrada.\n\n");
        tecla = teclado();
        switch(tecla){
                case '8':
                        if (nivel[bandaNivel-1] < 15){
                                nivel[bandaNivel-1]++;  // sube uno a uno la posción correspondiente al filtro
                                                        // seleccionado en el array nivel.
                                muestraEstadoEcualizacion();
                        }
                        break;
                case '9':
                        if (nivel[bandaNivel-1] > 0){
                                nivel[bandaNivel-1]--;  // baja uno a uno la posción correspondiente al filtro
                                                        // seleccionado en el array nivel.
                                muestraEstadoEcualizacion();
                        }
                        break;
                case 'E':
                        if (estado[MUESTRAECUALIZADOR]== 0){
```

```
                                estado[MUESTRAECUALIZADOR]= 1;
                                output ("VUMETRO: muestra ecualizador gráfico\n\n");
                        }else {
                                estado[MUESTRAECUALIZADOR]= 0;
                                output ("VUMETRO: muestra energía\n\n");
                        }
                        break;
                case 'B':
                        if (estado[MUESTRAENTRADA]== 0){
                                estado[MUESTRAENTRADA]= 1;
                                output ("VUMETRO: muestra entrada\n\n");
                        }else {
                                estado[MUESTRAENTRADA]= 0;
                                output ("VUMETRO: muestra salida\n\n");
                        }
                        break;
                case 'A':
                        flagsVolver[0]= 1;
                        break;
                default:
                        output ("Tecla no válida.\n");
                        break;
        }
}

//-------------------------------------------------------------
// void seleccionPreset(void)
// Descripción:
//      Permite seleccionar uno de los presets de ecualización.
//      o cargar una de las configuraciones previamente guardada
//      en una de las posiciones de memoria.
//      Además siempre tendremos la opción de seleccionar que
//      queremos visualizar en la matriz de leds, la energía de
//      las bandas o el ecualizador gráfico.
//-------------------------------------------------------------
void seleccionPreset(void){
        char tecla;

        output ("Seleccione una de las siguientes opciones de ecualización:\n");
        output ("\t1 - Rock.\n");
```

```
output ("\t2 - Rap.\n");
output ("\t3 - Grunge.\n");
output ("\t4 - Metal.\n");
output ("\t5 - Dance.\n");
output ("\t6 - Tecno.\n");
output ("\t7 - Jazz.\n");
output ("\t8 - Clásica.\n");
output ("\tD - Ecualizaciones guardadas.\n");
output ("\tA - Volver.\n\n");
output ("\tE - VÚMETRO: Mostrar ecualizador gráfico/Mostrar energía.\n");
output ("\tB - VÚMETRO: Mostrar salida/Mostrar entrada.\n\n");
tecla = teclado();
switch(tecla){
        case '1':
                output ("Ha seleccionado: ROCK \n\n");
                nivel[0] = 4;
                nivel[1] = 3;
                nivel[2] = 2;
                nivel[3] = 0;
                nivel[4] = 2;
                nivel[5] = 3;
                nivel[6] = 1;
                muestraEstadoEcualizacion();
                break;
        case '2':
                output ("Ha seleccionado: RAP \n\n");
                nivel[0] = 3;
                nivel[1] = 2;
                nivel[2] = 1;
                nivel[3] = 0;
                nivel[4] = 1;
                nivel[5] = 2;
                nivel[6] = 1;
                muestraEstadoEcualizacion();
                break;
        case '3':
                output ("Ha seleccionado: GRUNGE \n\n");
                nivel[0] = 5;
                nivel[1] = 3;
                nivel[2] = 1;
```

```
        nivel[3] = 3;
        nivel[4] = 2;
        nivel[5] = 1;
        nivel[6] = 0;
        muestraEstadoEcualizacion();
        break;
case '4':
        output ("Ha seleccionado: METAL \n\n");
        nivel[0] = 5;
        nivel[1] = 3;
        nivel[2] = 2;
        nivel[3] = 2;
        nivel[4] = 1;
        nivel[5] = 1;
        nivel[6] = 0;
        muestraEstadoEcualizacion();
        break;
case '5':
        output ("Ha seleccionado: DANCE \n\n");
        nivel[0] = 4;
        nivel[1] = 2;
        nivel[2] = 0;
        nivel[3] = 3;
        nivel[4] = 4;
        nivel[5] = 3;
        nivel[6] = 1;
        muestraEstadoEcualizacion();
        break;
case '6':
        output ("Ha seleccionado: TECNO \n\n");
        nivel[0] = 6;
        nivel[1] = 3;
        nivel[2] = 1;
        nivel[3] = 3;
        nivel[4] = 4;
        nivel[5] = 3;
        nivel[6] = 1;
        muestraEstadoEcualizacion();
        break;
case '7':
```

```
        output ("Ha seleccionado: JAZZ \n\n");
        nivel[0] = 3;
        nivel[1] = 3;
        nivel[2] = 2;
        nivel[3] = 1;
        nivel[4] = 0;
        nivel[5] = 1;
        nivel[6] = 2;
        muestraEstadoEcualizacion();
        break;
case '8':
        output ("Ha seleccionado: CLÁSICA \n\n");
        nivel[0] = 4;
        nivel[1] = 0;
        nivel[2] = 1;
        nivel[3] = 2;
        nivel[4] = 3;
        nivel[5] = 4;
        nivel[6] = 4;
        muestraEstadoEcualizacion();
        break;
case 'D':
        output ("¿Cuál desea cargar? Seleccione una de las 9 posiciones de memoria.\n\n");
        tecla = teclado();
        while ((tecla < '1') || (tecla > '9')){ //si pulsamos una tecla distinta del 1,2,3,4,5,6,7,8 o 9
                                                //nos indica el error y espera que pulsemos una tecla.
          output ("Tecla no válida\n\n");
          tecla = teclado();
        }
        output ("Ha seleccionado: ");
        outch (tecla);
        output("\n\n");
        nivel [0] = nivelesGuardados[tecla-'0'][0];
        nivel [1] = nivelesGuardados[tecla-'0'][1];
        nivel [2] = nivelesGuardados[tecla-'0'][2];
        nivel [3] = nivelesGuardados[tecla-'0'][3];
        nivel [4] = nivelesGuardados[tecla-'0'][4];
        nivel [5] = nivelesGuardados[tecla-'0'][5];
        nivel [6] = nivelesGuardados[tecla-'0'][6];
        muestraEstadoEcualizacion();
```

```
                        break;
                case 'E':
                        if (estado[MUESTRAECUALIZADOR]== 0){
                                estado[MUESTRAECUALIZADOR]= 1;
                                output ("VUMETRO: muestra ecualizador gráfico\n\n");
                        }else {
                                estado[MUESTRAECUALIZADOR] = 0;
                                output ("VUMETRO: muestra energía\n\n");
                        }
                        break;
                case 'B':
                        if (estado[MUESTRAENTRADA]== 0){
                                estado[MUESTRAENTRADA]= 1;
                                output ("VUMETRO: muestra entrada\n\n");
                        }else {
                                estado[MUESTRAENTRADA] = 0;
                                output ("VUMETRO: muestra salida\n\n");
                        }
                        break;
                case 'A':
                        flagsVolver[0]= 1;
                        break;
                default:
                        output ("Tecla no válida.\n\n");
                        break;
                }
}

//-----------------------------------------------------
// void muestraEstadoEcualizacion(void)
// Descripcion:
//      Muestra una tabla con el estado de ecualizacion
//      de cada filtro.
//-----------------------------------------------------
void muestraEstadoEcualizacion(void){
        int i;
        output("-----------------------------------------------------\n");
        output("BANDA:\t32Hz\t64Hz\t125Hz\t250Hz\t500Hz\t1kHz\t2kHz\n");
        output("GANANCIA:\t");
        for (i=0; i<7; i++){
```

```
                outNum(10, ganancia[nivel[i]],0);
                output("\t");
        }
        output("\n");
        output("NIVEL:\t");
        for (i=0; i<7; i++){
                outNum(10, nivel[i],0);
                output("\t");
        }
        output("\n-----------------------------------------------------");
        output("\n\n");
}

//---------------------------
// SUBMENÚS DE REVERBERACIÓN
//---------------------------

//-------------------------------------------------
// void GestionReverb(void)
// Descripción:
//   Muestra el submenú de "Gestión de reverberación" y nos permite
//   activarla, mostrar la configuración actual, guardarla
//   en una de las 9 posiciones de memoria disponibles,cargar
//   una de las configuraciones previamente guardada o desactivar
//   la función de reverberación
//-------------------------------------------------
void GestionReverb(void){
        char tecla;

        output ("Activación de la reverberación.\n");
        output ("\t1 - Activar.\n");
        output ("\t2 - Desactivar.\n");
        output ("\t0 - Muestra configuración actual.\n");
        output ("\tD - Guardar configuración actual.\n");
        output ("\tE - Cargar una configuración previamente guardada.\n");
        output ("\tA - Volver.\n\n");
        output ("\tB - VÚMETRO: Mostrar salida/Mostrar entrada.\n\n");

        flagsVolver[0]= 0;
        tecla = teclado();
```

```
switch(tecla){
        case '1':
                while (flagsVolver[0]== 0){
                        selectAtenuacion();
                }
                break;
        case '2':
                output ("Reverberación desactivada.\n\n");
                estado[REVERBERACION]= 0;
                estado[ATTREVERB] =0;
                estado[TIEMPORETARDO]=0;
                break;
        case '0':
                output ("Configuración actual:\n");
                output ("\tNIVEL DE ATENUACIÓN: ");
                outNum(10, estado[ATTREVERB],0);
                output ("\n\tRETARDO: ");
                outNum(10, estado[TIEMPORETARDO], 0);
                output("\n\n");
                break;
        case 'D':
                output ("¿Dónde desea guardarlo? Seleccione una de las 9 posiciones de memoria.\n\n");
                tecla = teclado();
                while ((tecla < '1') || (tecla > '9')){
                        output ("Tecla no válida\n\n");
                        tecla = teclado();
                }
                output ("Ha seleccionado: ");
                outch (tecla);
                output("\n\n");
                reverbGuardadas[tecla-'0'][0] = estado[ATTREVERB];
                reverbGuardadas[tecla-'0'][1] = estado[TIEMPORETARDO];
                output ("Configuración guardada.");
                break;
        case 'E':
                output ("¿Cuál desea cargar? Seleccione una de las 9 posiciones de memoria.\n\n");
                tecla = teclado();
                while ((tecla < '1') || (tecla > '9')){
                        output ("Tecla no válida\n\n");
                        tecla = teclado();
```

```
                        }
                        estado[ATTREVERB]= reverbGuardadas[tecla-'0'][0];
                        estado[TIEMPORETARDO]= reverbGuardadas[tecla-'0'][1];
                        output ("Reverberación activada.\n");
                        output ("\tNIVEL DE ATENUACIÓN: ");
                        outNum(10, estado[ATTREVERB],0);
                        output ("\n\tRETARDO: ");
                        outNum(10, estado[TIEMPORETARDO],0);
                        output("\n\n");
                        break;
                case 'B':
                        if (estado[MUESTRAENTRADA]== 0){
                                estado[MUESTRAENTRADA]= 1;
                                output ("VUMETRO: muestra entrada\n\n");
                        }else {
                                estado[MUESTRAENTRADA]= 0;
                                output ("VUMETRO: muestra salida\n\n");
                        }
                        break;
                case 'A':
                        estado[FUNCION] = 0;
                        break;
                default:
                        output ("Tecla no válida.\n\n");
                        break;
        }
}

//-------------------------------------------------------
// void selectAtenuacion(void)
// Descripcion:
//      Permite ajustar el parámetro de atenuación de la
//      reverberación. Dicho parametro indica por cuanto
//      se divide la señal en cada rebote.
//-------------------------------------------------------
void selectAtenuacion(void){
        char tecla;

        output("\t1,8 - Seleccione el nivel de atenuación: de 1 a 8\n");
        output("\tA - Volver.\n\n");
```

```
flagsVolver[1]= 0;
tecla = teclado();
switch(tecla){
        case '1':
                estado[ATTREVERB]= 1;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
        case '2':
                estado[ATTREVERB]= 2;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
        case '3':
                estado[ATTREVERB]= 3;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
        case '4':
                estado[ATTREVERB]= 4;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
        case '5':
                estado[ATTREVERB]= 5;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
        case '6':
                estado[ATTREVERB]= 6;
                while (flagsVolver[1] == 0){
                        selectRetardo();
                }
                break;
```

```
                case '7':
                        estado[ATTREVERB]= 7;
                        while (flagsVolver[1] == 0){
                                selectRetardo();
                        }
                        break;
                case '8':
                        estado[ATTREVERB]= 8;
                        while (flagsVolver[1] == 0){
                                selectRetardo();
                        }
                        break;
                case 'A':
                        flagsVolver[0] = 1;
                        break;
                default:
                        output ("Tecla no válida.\n\n");
                        break;
        }


}

//-------------------------------------------------------
// void selectRetardo(void)
// Descripcion:
//      Permite seleccionar el retardo del efecto de
//      reverberación. Dicho parámetro indica cuanto
//      tarda el eco en volver desde que se emite un
//      sonido.
//-------------------------------------------------------
void selectRetardo(void){
        char tecla;

        output ("Seleccione el retardo:\n");
        output ("\t1 - 100 ms\n");
        output ("\t2 - 200 ms\n");
        output ("\t3 - 300 ms\n");
        output ("\t4 - 400 ms\n");
        output ("\t5 - 500 ms\n");
```

```
output ("\t6 - 600 ms\n");
output ("\t7 - 700 ms\n");
output ("\t8 - 800 ms\n");
output ("\t9 - 900 ms\n");
output ("\t0 - 1 s\n");
output ("\tA - Volver\n\n");

tecla = teclado();
switch(tecla){
        case '1':
                output ("Reverberación activada.\n");
                output ("\tNIVEL DE ATENUACIÓN: ");
                outNum(10, estado[ATTREVERB],0);
                output ("\n\tRETARDO: 100ms\n\n");
                estado[REVERBERACION]= 1;
                estado[TIEMPORETARDO]= 100;
                flagsVolver[0] = 1;
                flagsVolver[1] = 1;
                break;
        case '2':
                output ("Reverberación activada.\n");
                output ("\tNIVEL DE ATENUACIÓN: ");
                outNum(10, estado[ATTREVERB],0);
                output ("\n\tRETARDO: 200ms \n\n");
                estado[REVERBERACION]= 1;
                estado[TIEMPORETARDO]= 200;
                flagsVolver[0] = 1;
                flagsVolver[1] = 1;
                break;
        case '3':
                output ("Reverberación activada.\n");
                output ("\tNIVEL DE ATENUACIÓN: ");
                outNum(10, estado[ATTREVERB],0);
                output ("\n\tRETARDO: 300ms \n\n");
                estado[REVERBERACION]= 1;
                estado[TIEMPORETARDO]= 300;
                flagsVolver[0] = 1;
                flagsVolver[1] = 1;
                break;
        case '4':
```

```
            output ("Reverberación activada.\n");
            output ("\tNIVEL DE ATENUACIÓN: ");
            outNum(10, estado[ATTREVERB],0);
            output ("\n\tRETARDO: 400ms \n\n");
            estado[REVERBERACION]= 1;
            estado[TIEMPORETARDO]= 400;
            flagsVolver[0] = 1;
            flagsVolver[1] = 1;
            break;
    case '5':
            output ("Reverberación activada.\n");
            output ("\tNIVEL DE ATENUACIÓN: ");
            outNum(10, estado[ATTREVERB],0);
            output ("\n\tRETARDO: 500ms \n\n");
            estado[REVERBERACION]= 1;
            estado[TIEMPORETARDO]= 500;
            flagsVolver[0] = 1;
            flagsVolver[1] = 1;
            break;
    case '6':
            output ("Reverberación activada.\n");
            output ("\tNIVEL DE ATENUACIÓN: ");
            outNum(10, estado[ATTREVERB],0);
            output ("\n\tRETARDO: 600ms \n\n");
            estado[REVERBERACION]= 1;
            estado[TIEMPORETARDO]= 600;
            flagsVolver[0] = 1;
            flagsVolver[1] = 1;
            break;
    case '7':
            output ("Reverberación activada.\n");
            output ("\tNIVEL DE ATENUACIÓN: ");
            outNum(10, estado[ATTREVERB],0);
            output ("\n\tRETARDO: 700ms \n\n");
            estado[REVERBERACION]= 1;
            estado[TIEMPORETARDO]= 700;
            flagsVolver[0] = 1;
            flagsVolver[1] = 1;
            break;
    case '8':
```

```
                        output ("Reverberación activada.\n");
                        output ("\tNIVEL DE ATENUACIÓN: ");
                        outNum(10, estado[ATTREVERB],0);
                        output ("\n\tRETARDO: 800ms \n\n");
                        estado[REVERBERACION]= 1;
                        estado[TIEMPORETARDO]= 800;
                        flagsVolver[0] = 1;
                        flagsVolver[1] = 1;
                        break;
                case '9':
                        output ("Reverberación activada.\n");
                        output ("\tNIVEL DE ATENUACIÓN: ");
                        outNum(10, estado[ATTREVERB],0);
                        output ("\n\tRETARDO: 900ms \n\n");
                        estado[REVERBERACION]= 1;
                        estado[TIEMPORETARDO]= 900;
                        flagsVolver[0] = 1;
                        flagsVolver[1] = 1;
                        break;
                case '0':
                        output ("Reverberación activada.\n");
                        output ("\tNIVEL DE ATENUACIÓN: ");
                        outNum(10, estado[ATTREVERB], 0);
                        output ("\n\tRETARDO: 1s \n\n");
                        estado[REVERBERACION]= 1;
                        estado[TIEMPORETARDO]= 1000;
                        flagsVolver[0] = 1;
                        flagsVolver[1] = 1;
                        break;

                case 'A':
                        flagsVolver[1]= 1;
                        break;
                default:
                        output ("Tecla no válida.\n\n");
                        break;
        }

}
```


```
//---------------------------------------------------------------
// Filtros.c
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//---------------------------------------------------------------

//----------------------------------------
//DECLARACIÓN DE VARIABLES Y ARRAYS GENERALES
//----------------------------------------
int salidasFiltros[7]={0,0,0,0,0,0,0};                          //Array para guardar las salidas de cada filtro y asi
poder acceder a ellas cuando sea necesario. Una columna para cada filtro.

//----------------------
//DECLARACIÓN DE MÉTODOS
//----------------------
void calculaSalidasFiltros(int lectura);


//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------




//-----------------------------------------------------
```

```
// void calculaSalidasFiltros(int lectura)
// Descripcion:
//      Calcula la salida de todos los filtros y las
//      las guarda en su posicion correspondiente del
//      array salidasFiltros[]
//-------------------------------------------------------
void calculaSalidasFiltros(int lectura){
        int i;
                                                        // Array con los coeficientes de los filtros. 7 filas,
                                                        // una para cada banda. 6 columnas para cada coeficiente
                                                        // ordenados de la forma: G, b0, b1, b2, a1, a2.
static int coefs[7][6] = {{8,1024,0,-1024,-2029,1006},  //PRIMERA LINEA DEL ARRAY = banda 0 - frecuencia central 31,25Hz
                          {17,1024,0,-1024,-2011,988},  //SEGUNDA LINEA DEL ARRAY = banda 1 - frecuencia central 62,50Hz
                          {34,1024,0,-1024,-1970,955},  //TERCERA LINEA DEL ARRAY = banda 2 - frecuencia central 125Hz
                          {66,1024,0,-1024,-1878,890},  //CUARTA LINEA DEL ARRAY = banda 3 - frecuencia central 250Hz
                          {125,1024,0,-1024,-1660,772}, //QUINTA LINEA DEL ARRAY = banda 4 - frecuencia central 500Hz
                          {227,1024,0,-1024,-1115,569}, //SEXTA LINEA DEL ARRAY = banda 5 - frecuencia central 1000Hz
                          {392,1024,0,-1024,141,239}};  //SEPTIMA LINEA DEL ARRAY = banda 6 - frecuencia central 2000Hz

        static int historia[7][2] =    {{0,0},  //Array para guardar las historias necesarias para carlcular filtrado
                                        {0,0},  //Una linea para cada filtro. Una columna para cada instante
                                        {0,0},
                                        {0,0},
                                        {0,0},
                                        {0,0},
                                        {0,0}};

        for(i=0; i<7; i++){     //Hace las operaciones necesarias para calcular las salidas de cada filtros y se
                                //almacenan en su posición correspondiente del array salidasFiltros
                int operando1=0;
                int operando2=0;
                int operando3=0;
                int operando4=0;
                int operando5=0;
                int suma1=0;
                int suma1temp=0;
                int suma2=0;


                operando1 = lectura*coefs[i][1];
```

```
                operando2 = historia[i][0]*(-coefs[i][4]);
                operando3 = historia[i][1]*(-coefs[i][5]);
                operando4 = historia[i][0]*coefs[i][2];
                operando5 = historia[i][1]*coefs[i][3];
                suma1 = operando1 + operando2 + operando3;
                suma1temp = suma1>>10;          //Utiliza esta variable temporal para guardar las historias escaladas.
                historia[i][1] = historia[i][0];
                historia[i][0] = suma1temp;
                suma2 = (suma1 + operando4 + operando5)>>10;    //Desplaza 10 posiciones para eliminar el escalado por
                                                                //1024 de los coeficientes a y b.
                suma2 = suma2*coefs[i][0];
                suma2 = suma2>>10;      //Desplaza 10 posiciones para eliminar el escalado por 1024 de la ganancia
                salidasFiltros[i] = suma2;      //Almacena en la posición correspondiente del array salidasFiltros
        }
}
```


```
//-------------------------------------------------------------
// Reverberacion.c
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//-------------------------------------------------------------

//----------------------
//DECLARACIÓN DE MÉTODOS
//----------------------
int salidaReverberacion(int muestra, int atenuacion, int retardo);

//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------

//-----------------------------------------------------
// int salidaReverberacion(int muestra, int atenuacion, int retardo)
// Descripción:
//      calcula la salida correspondiente al efecto de
//      reverberación.
//      Los parametros son:
//              muestra: la muestra a la que se le suma el eco
//              atenuacion: factor de atenuación del eco
//              retardo: tiempo que tarda en llegar el eco
//      Devuelve la muestra con el eco sumado
//-----------------------------------------------------
```

```
int salidaReverberacion(int muestra, int atenuacion, int retardo){
	static int muestrasAlmacenadas[8000];	// array que guarda 8000 muestras equivalentes a 1 segundo.
	static int posicionMuestraActual=0;	// variable que funciona como índice de la posición donde se debe guardar
						// la muestra entrante dentro del array
	static int posicionMuestraRetardada=0;	// variable que funciona como índice de la posición donde se encuentra la
						// muestra antigua que se sumará como eco
	int salidaReverb=0;
	int retardoAtenuado=0;

	if (posicionMuestraActual == 8000){	// si llegamos al final del bucle circular donde se guardan las muestras
						// volvemos al principio
		posicionMuestraActual = 0;
	}
	if(estado[REVERBERACION] == 1){		// si la reverberación esta activada
		posicionMuestraRetardada = (posicionMuestraActual) - 8*retardo;
				// Puesto que retardo es un múltiplo de 100ms y cada 100ms guardamos 8 muestras,
				// la muestra antigua correspondiente al retardo seleccionado estará 8*retardo
				// posiciones más atras de la actual.
		if (posicionMuestraRetardada < 0){	// si posicionMuestraRetardada es un número negativo, le suma el
							// tamaño total del array
			posicionMuestraRetardada += 8000;
		}

		if(atenuacion== 1){
			retardoAtenuado = muestrasAlmacenadas[posicionMuestraRetardada];
					//selecciona la muestra antigua y atenua el factor correspondiente
			salidaReverb = muestra + retardoAtenuado;
		}else if(atenuacion == 2){
			retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada]>>1);	// si atenuacion = 2,
												// divide por 2.
			salidaReverb = muestra + retardoAtenuado;
		}else if(atenuacion == 3){
			retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada]*85)>>8;
					// si atenuacion = 3, divide por 3. (utiliza un escalado de 256 para no trabajar
					// con decimales. Después desplaza 8 posiciones para eleminar dicho escalado)
			salidaReverb = muestra + retardoAtenuado;
		}else if(atenuacion == 4){
			retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada])>>2;
			salidaReverb = muestra + retardoAtenuado;
		}else if(atenuacion == 5){
```

```
                        retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada]*51)>>8;
                        salidaReverb = muestra + retardoAtenuado;
                }else if(atenuacion == 6){
                        retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada]*43)>>8;
                        salidaReverb = muestra + retardoAtenuado;
                }else if(atenuacion == 7){
                        retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada]*37)>>8;
                        salidaReverb = muestra + retardoAtenuado;
                }else if(atenuacion == 8){
                        retardoAtenuado = (muestrasAlmacenadas[posicionMuestraRetardada])>>3;
                        salidaReverb = muestra + retardoAtenuado;
                }

        }else{                          //En caso de que la reverberación no este activada saca la muestra entrante.
                salidaReverb = muestra;
        }
        muestrasAlmacenadas[posicionMuestraActual]=salidaReverb;        // guarda la muestra con el eco añadido para su
                                                                        // posterior utilización.
        posicionMuestraActual++;                //incrementa el índice que indica la posición donde debe guardar la
                                                //siguiente muestra
        return salidaReverb;
}
```


```
//-------------------------------------------------------------
// Vumetro.c
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//-------------------------------------------------------------

//----------------------------------------
//DECLARACIÓN DE VARIABLES Y ARRAYS GENERALES
//----------------------------------------
int signalSinTeclado=0;         // variable que servirá de máscara en la funcion teclado() para salvar los bits de la
                                // salida correspondientes a la matriz de leds

//----------------------
//DECLARACIÓN DE MÉTODOS
//----------------------
void configuraPuerto(WORD signalTecladoVumetro, int filtro, int nivelEnergia);
void gestionVumetro(int muestra, WORD teclado, int filtro);
void gestionEcualizGrafico(int nivel, WORD teclado, int filtro);


//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
```

```
//-------------------------------------------------------
// void gestionVumetro(int muestra, int teclado, int filtro)
// Descripcion:
//      Compara el nivel energía de la muestra con unos niveles
//      umbral (multiplicados por 24) para saber los leds debe iluminar
//      en el vúmetro. Configura el puerto de salida al vúmetro
//      para encender los leds necesarios llamando al método
//      configuraPuerto
//-------------------------------------------------------
void gestionVumetro(int muestra, WORD teclado, int filtro){
        int nivelEnergia=0;             // variable que guarda cuantos de los 8 niveles de energía debe pintar.

        if (muestra>= 6464496){
                nivelEnergia = 8;
        }else if (muestra>= 2309352){
                nivelEnergia = 7;
        }else if (muestra>= 824976){
                nivelEnergia = 6;
        }else if (muestra>= 294720){
                nivelEnergia = 5;
        }else if (muestra>= 105264){
                nivelEnergia = 4;
        }else if (muestra>= 37608){
                nivelEnergia = 3;
        }else if (muestra>= 13416){
                nivelEnergia = 2;
        }else if (muestra>= 4800){
                nivelEnergia = 1;
        }else if (muestra< 4800){
                nivelEnergia =0;        // en el caso de tener la mínima energía no se encenderá ningún led
        }
        configuraPuerto(teclado, filtro, nivelEnergia);

}
```

```
//-------------------------------------------------------
// void configuraPuerto(int signalTecladoVumetro, int filtro, int nivelEnergia)
// Descripcion:
//      Configura el puerto de salida al vúmetro mediante el
//      empleo de máscaras para la iluminación de los leds,
//      según el nivel de energía que tenga la banda.
//-------------------------------------------------------
void configuraPuerto(WORD signalTecladoVumetro, int filtro, int nivelEnergia){
        int signalPuerto =0;
        int mascaraColumnas = 0;
        int mascaraFilas = filtro;                      //MASK filtro1 (B4,B5,B6)= 000
                                                        //MASK filtro2 (B4,B5,B6)= 001
                                                        //MASK filtro3 (B4,B5,B6)= 010
                                                        //MASK filtro4 (B4,B5,B6)= 011
                                                        //MASK filtro5 (B4,B5,B6)= 100
                                                        //MASK filtro6 (B4,B5,B6)= 101
                                                        //MASK filtro7 (B4,B5,B6)= 110
                                                        //MASK filtro8 (B4,B5,B6)= 111

        if (nivelEnergia == 0){
                mascaraColumnas = 0;            //MASK nivelEnergia0 (B8,B9,B10,B11,B13,B14,B15)= 0000 0000
        }else if (nivelEnergia == 1){
                mascaraColumnas = 1;            //MASK nivelEnergia1 (B8,B9,B10,B11,B13,B14,B15)= 0000 0001
        }else if (nivelEnergia == 2){
                mascaraColumnas = 3;            //MASK nivelEnergia2 (B8,B9,B10,B11,B13,B14,B15)= 0000 0011
        }else if (nivelEnergia == 3){
                mascaraColumnas = 7;            //MASK nivelEnergia3 (B8,B9,B10,B11,B13,B14,B15)= 0000 0111
        }else if (nivelEnergia == 4){
                mascaraColumnas = 15;           //MASK nivelEnergia4 (B8,B9,B10,B11,B13,B14,B15)= 0000 1111
        }else if (nivelEnergia == 5){
                mascaraColumnas = 31;           //MASK nivelEnergia5 (B8,B9,B10,B11,B13,B14,B15)= 0001 1111
        }else if (nivelEnergia == 6){
                mascaraColumnas = 63;           //MASK nivelEnergia6 (B8,B9,B10,B11,B13,B14,B15)= 0011 1111
        }else if (nivelEnergia == 7){
                mascaraColumnas = 127;          //MASK nivelEnergia7 (B8,B9,B10,B11,B13,B14,B15)= 0111 1111
        }else if (nivelEnergia == 8){
                mascaraColumnas = 255;          //MASK nivelEnergia8 (B8,B9,B10,B11,B13,B14,B15)= 1111 1111
        }
        mascaraFilas = mascaraFilas << 4;       //Desplaza 4 bits. Los correspondientes a las filas son del 5 al 7
        mascaraColumnas = mascaraColumnas <<8;  //Desplaza 8 bits. Los correspondientes a las columnas son del 9 al 16
```

```
        signalSinTeclado = 0|mascaraFilas;
        signalSinTeclado = signalSinTeclado |mascaraColumnas;   // Mediante estas dos operaciones 'OR' consigue en un
                                                                //mismo parámetro las dos máscaras sin que se machaquen
                                                                //la una a la otra. Se utiliza en el método teclado()
        signalPuerto = signalSinTeclado |signalTecladoVumetro;  // Añade la máscara del teclado, correspondiente a los 4
                                                                //primeros bits de la salida, para no interferir en el
                                                                //funcionamiento de este.
        set16_puertoS(signalPuerto);            // Asignamos el valor de la máscara al puerto para que el en la matriz de
                                                //leds se enciendan los leds que corresponden.
}

//-------------------------------------------------------
// void gestionEcualizGrafico(nivel[filtroIluminado] , signalTeclado, filtroIluminado)
// Descripcion:
//      Muestra el ecualizador gráfico en el matriz de leds
//      Dedica un led para cada dos niveles de ecualización.
//      Configura el puerto de salida al vúmetro para encender los leds
//      necesarios llamando al método configuraPuerto
//-------------------------------------------------------
void gestionEcualizGrafico(int nivel, WORD teclado, int filtro){
        int nivelEnergia =0;
        if (nivel<= 1){
                nivelEnergia = 7;
        }else if (nivel<= 3){
                nivelEnergia = 6;
        }else if (nivel<= 5){
                nivelEnergia = 5;
        }else if (nivel<= 7){
                nivelEnergia = 4;
        }else if (nivel<= 9){
                nivelEnergia = 3;
        }else if (nivel<= 11){
                nivelEnergia = 2;
        }else if (nivel<= 13){
                nivelEnergia = 1;
        }else if (nivel<= 15){
                nivelEnergia = 0;
        }
        configuraPuerto(teclado, filtro, nivelEnergia);
}
```


```
//-------------------------------------------------------------------------
// teclado.c
//
//   Programa para el manejo de un teclado matricial.
//
// Autores: Rafael López Martínez
//          Alfredo Álvarez Senra
//-------------------------------------------------------------------------
#include "m5272lib.c"
#include "m5272gpio.c"

#define NUM_FILAS 4
#define NUM_COLS 4
#define EXCIT 1

int signalTeclado = 0;

char teclado(void);

//-----------------------------------------------------
// char teclado(void)
//
// Descripción:
//   Explora el teclado matricial y devuelve la tecla
//   pulsada
//-----------------------------------------------------
```

```
char teclado(void){
  //char tecla;
  BYTE fila, columna, fila_mask;
  static char teclas[4][4] = {{"123C"},
                              {"456D"},
                              {"789E"},
                              {"A0BF"}};
  int signalPuerto =0;
  signalPuerto=0;
  // Bucle de exploración del teclado
  while(TRUE){

    // Excitamos una columna
    for(columna = NUM_COLS - 1; columna >= 0; columna--){
      signalTeclado = EXCIT << columna;
      signalPuerto = signalTeclado | signalSinTeclado;
      set16_puertoS(signalPuerto);              // Se envía la excitación de columna
      retardo(1150);                            // Esperamos respuesta de optoacopladores

      // Exploramos las filas en busca de respuesta
      for(fila = NUM_FILAS - 1; fila >= 0; fila--){
        fila_mask = EXCIT << fila;              // Máscara para leer el bit de la fila actual
        if(lee16_puertoE() & fila_mask){                // Si encuentra tecla pulsada,
          while(lee16_puertoE() & fila_mask);   //   Esperamos a que se suelte
          retardo(1150);                        //   Retardo antirrebotes
          return teclas[fila][columna];         //   Devolvemos la tecla pulsada
        }
      }
      // Siguiente columna
    }
    // Exploración finalizada sin encontrar una tecla pulsada
  }
  // Reiniciamos exploración
}
```