GRABADORA Y REPRODUCTORA DE VOZ

Memoria del Proyecto

Antonio Valera | Ximo Gallud

Proyectos de Ingeniería Física I . Grado en Ingeniería Física. UPC.

Contenido

1. Introducción: Especificaciones del proyecto

Atendiéndonos a los objetivos presentados en el documento inicial, el proyecto consiste en desarrollar un aparato (una grabadora reproductora) capaz de reproducir una señal de voz que previamente ha capturado y almacenada en la memoria de un microcontrolador. La señal almacenada en memoria debe ser demodulada y tratada para ser escuchada a través de unos auriculares.

Las especificaciones del proyecto son:

- **Tiempo de duración de la señal de voz:** alrededor de 4 segundos (este tiempo depende de la frecuencia de muestreo, la cual se puede variar con facilidad si se accede al código para obtener mejor calidad de sonido o bien más tiempo de grabación).
- **Máxima intensidad de detección del micrófono**: Alrededor de 70 dB.
- **Ancho de banda:** 0-3.8kHz. Corresponde al ancho de banda empleado usualmente en telefonía.
- **Frecuencia de muestreo**: 8 kHz.
- La señal de salida debe **reproducir fiablemente** sonidos dentro del rango de una conversación humana.
- El aparato debe contar con indicadores visuales del estado de grabación o reproducción.

2. Descripción del funcionamiento general

De cara al usuario, el aparato cuenta con dos pulsadores, dos LEDs (uno azul y otro rojo), un micrófono, un *jack* de audio para conectar bien auriculares o altavoces y una rueda de regulación de un potenciómetro que sirve como regulación del volumen de la señal de salida.

Una vez se pulsa el pulsador SW1, se enciende el LED rojo y el aparato comienza a grabar. El microprocesador irá muestreando la señal de voz del hablante (que registra el micrófono) hasta que la memoria de almacenamiento de datos se agote, en este caso el sistema dejará de grabar y se apagará el LED.

El pulsador SW2 será el activador de la reproducción. Cuando se pulse el botón el sistema emitirá la señal grabada a través del *Jack* de audio, cuyo volumen puede ser regulado a gusto del usuario. Durante este tiempo el LED azul se mantendrá encendido, hasta que la señal se acabe de reproducir.

3. Materiales utilizados

Recursos de la placa test:

- *Jack* estéreo (salida de audio).
- Micro controlador *Pinguino OTG*.
- LED Rojo: Se enciende cuando el sistema está en modo grabación.
- LED Azul: Se enciende en modo reproducción.
- Pulsadores SW1 y SW2: Mientras el SW1 esté pulsado el sistema grabaría audio, que reproduciría al pulsar SW2. Cada vez que se pulsa SW1 la información anterior sustituida por la nueva grabación.
- Micrófono *electret* AMB-714-RC

Recursos del laboratorio:

- Fuente de tensión de 5V DC *Promax FAC-362*: usada para la alimentación de los amplificadores operacionales en los filtros.
- Osciloscopio *Agilent 54622D* y Multímetro *Promax MD-100C*: en el proyecto son usados para el montaje y el testeo (como por ejemplo medida de valores reales de las resistencias a usar, comprobar que un sistema amplificador funciona correctamente, observación de respuestas de filtros, observación de la respuesta del programa implementado…) y no tienen ningún papel en el sistema final.
- Generador de funciones GF-232: Se utiliza el generador de funciones para la generación de señales senoidales puras para comprobar sin necesidad de utilizar el circuito entero (circuito de entrada) todas las demás partes del proyecto (conversor D/A, salida del procesador, salida de los altavoces …)

Recursos propios:

Estos recursos engloban el material empleado en la construcción de los filtros y el todo el cable para las conexiones del proyecto. Todos estos componentes ya los adquirimos con motivo de las prácticas de laboratorio salvo alguna pequeña excepción, así que no hay costes añadidos importantes asociados al proyecto.

- 2 Amplificadores operacionales duales TLC272. Se emplean 4 amplificadores: uno para la amplificación de la señal de salida del micrófono (que es del orden de milivoltios), otro para el filtrado de la señal del micrófono, y los dos últimos los usamos para el filtro suavizador a la salida del conversor Digital-Analógico.
- 2 placas *protoboard*. Por comodidad se ha comprado una segunda placa *protoboard* para acoplar con comodidad el conversor D/A que es bastante aparatoso.
- Resistencias y Condensadores de los Filtros. Todos se encuentran en principio en el rango y número de las resistencias y condensadores comprados para las prácticas de laboratorio.

- Auriculares de salida: Se utilizan auriculares de calidad baja. (Los que reparten en aviones, o en trenes).
- Cables de Conexión: de bobina y BNC.

A continuación se adjunta una lista detallada de los materiales utilizados y el coste económico real que ha tenido.

Lista de materiales

Tipo de material	Tipo/ Especificaciones	Unidades	Precio/ud (€)	Precio total (€)	Presupuestado previamente
Recursos de la placa test	Micrófono electret AMB-714-RC	1	P	0	SÍ
	Jack estéreo (2 entradas)	1	P	0	SÍ
	Pulsador SW1	1	P	0	SÍ
	Pulsador SW2	1	P	0	SÍ
	LED Rojo	1	P	0	SÍ
	LED Azul	1	P	0	SÍ
	Microprocesador ***PINGUINO OTG***	1	P	0	SÍ
Resistencias					
	Serie de 30 resistencias 1K (tol 1%)	1	1.31	1.31	NO
	Serie de 61 resistencias E12 (tol 5%)**	2	1.35	2.7	SÍ
	4.7K	1			
	470	2			
	150	2			
	2.2K	2			
	3.3K	1			
	390K	1			
	2.7K	1			
	120	1			
	1.5K	1			
Condensadores					
	Electrolítico 1µF	2	0.417	0.834	SÍ
	Multicapa 10 nF	2	0.138	0.276	SÍ
	Multicapa 100 nF	2	0.118	0.236	SÍ
Amplificadores					
	TLC272	3	0.97	2.91	SÍ ***
Hilo de cable					
	Aprox 2-2.5 m	1	0.25	0.625	SÍ

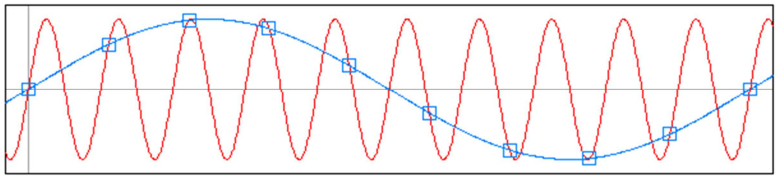
Placas protoboard					
	Protoboard de 800+ bornes	2	5.25	10.5	SÍ
Recursos laboratorio					
	Osciloscopio AGILENT 54622D	1	P	0	SÍ
	Generador funciones PROMAX MD-100C	1	P	0	SÍ
	Multímetro PROMAX MD-100C	1	P	0	SÍ
	Fuente alimentación PROMAX FAC-362	1	P	0	SÍ
	Cables tipo banana	4	P	0	SÍ
	Cables BNC	3	P	0	SÍ
			TOTAL PRECIO	19.39	

*P = proporcionado por la universidad
** Las resistencias usadas se detallan bajo
*** Estaba presupuestado el uso de 2, pero uno se quemó

4. Problemas teóricos en el muestreo de una señal analógica a solventar en el circuito:

Problemas de *aliasing*

En la conversión de una señal de analógico a digital puede aparecer *aliasing*. Este fenómeno está asociado a la frecuencia de muestro de la señal y consiste en los valores del muestreo son compatibles con una señal de frecuencia mayor a la señal analizada (un alias).



Ejemplo de sinusoide confundida con otra de frecuencia mucho mayor debido a una frecuencia de muestreo insuficiente.

Para evitar este fenómeno, la frecuencia de muestreo debe ser suficientemente alta. El Teorema de Nyquist – Shanon establece que si la frecuencia máxima de una señal analógica $x(t)$ es F_{max}, si la frecuencia de muestreo es $F_s > 2F_{max}$ no se produce aliasing y además, la señal $x(t)$ puede recuperarse de la forma:

$$x(t) = \sum_{-\infty}^{\infty} x\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right)$$

$$donde\ \ g(t) = \frac{sen2\pi F_{max}}{2\pi F_{max}}\ y\ x\left(\frac{n}{F_s}\right) = x(nT) = x[n], es\ decir\ el\ muestreo.$$

Para evitar este fenómeno en nuestro proyecto, en el cual no se puede utilizar una frecuencia de muestreo a voluntad sino aquella con que el microcontrolador sea capaz de trabajar, se debe aplicar un filtro *antialiasing* a la entrada. Este filtro será un filtro de paso banda cuya frecuencia de corte mayor sea la mitad de la frecuencia de muestro del chip.

Problemas de imaging

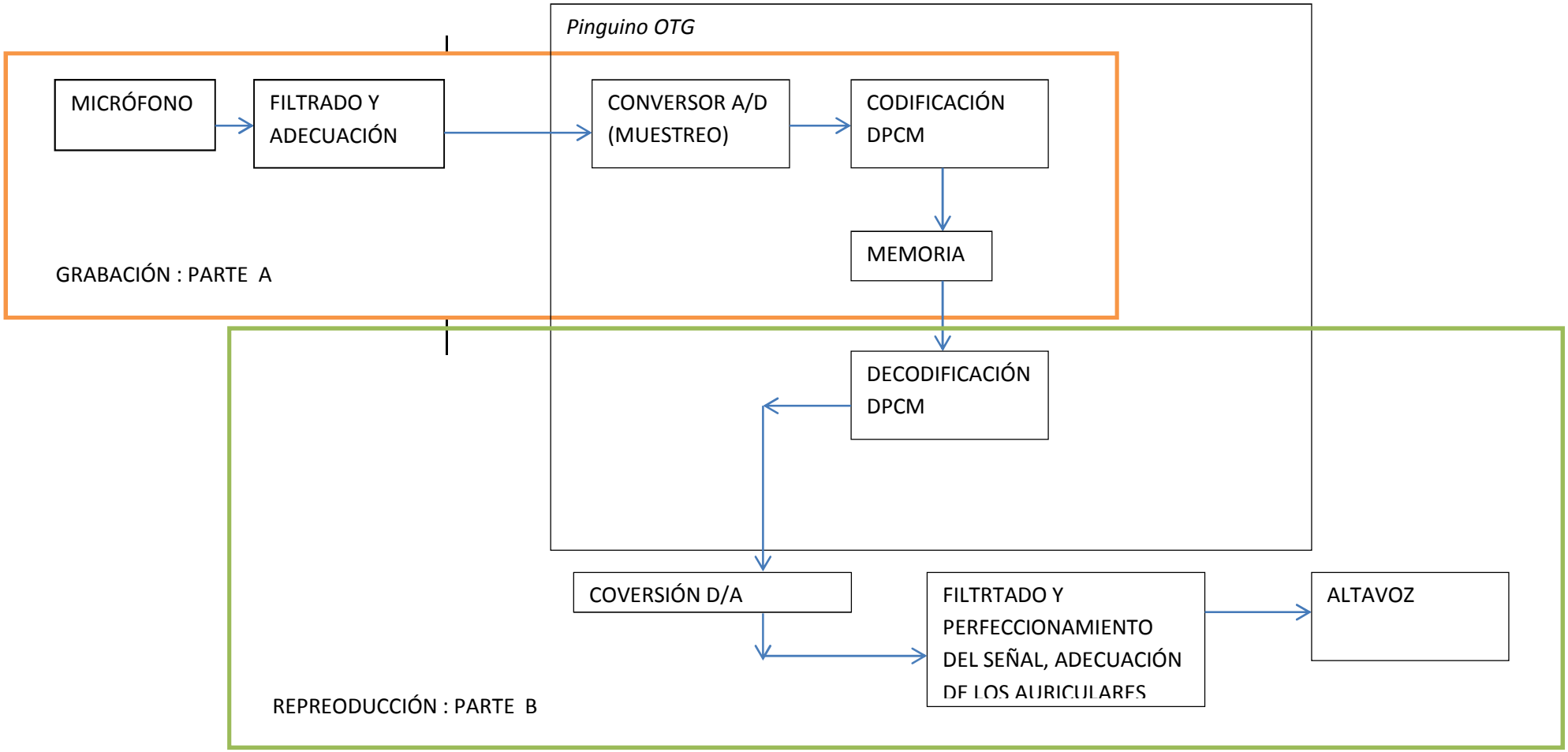
El filtro *anti-imaging* o filtro de reconstrucción consiste en un filtro paso bajo que debe implementarse en la conversión D/A para evitar el fenómeno de *imaging*. Este fenómeno consiste en que en la interpolación del paso de digital a analógico, cierta potencia de frecuencias altas (puede que no audibles, por encima de los 20kHZ) pueden pasar a frecuencias más bajas distorsionando la señal.

5. Metodología para la verificación de las especificaciones y el funcionamiento del circuito

Para desarrollar la grabadora reproductora, se elige seguir una cierta metodología:

- Para los filtros a la entrada y a la salida, primero se hará el cálculo teórico de los parámetros que tiene que tener el filtro para el ancho de banda escogido de 3.8kHz. Se observará en el osciloscopio la respuesta de los filtros a una señal senoidal generada por el generador de funciones, para distintas frecuencias.
- El conjunto del micrófono-filtro se acoplará también al osciloscopio para observar la respuesta obtenida. Se comprueba la **máxima tensión usual del micrófono** pues no se quiere que para niveles de sonido cotidianos, la señal de entrada al microcontrolador sature.
- El conjunto altavoz se probará antes de pasar por el microprocesador, directamente generando una señal senoidal en el generador de funciones y escuchando en el altavoz la respuesta en tiempo real. Seguidamente se le acoplará el micrófono y se observará si el conjunto micrófono-altavoz funciona correctamente para ruido ambiente. Es importante que se haya eliminado en esta etapa el máximo ruido posible.
- Seguidamente se acoplará el microprocesador. Se observará cuál es la respuesta del programa y de la salida cuando le introducimos una señal senoidal generada por el generador de funciones, para diversas frecuencias, se comprueba así el **ancho de banda** real al que estamos trabajando.
- A continuación se observará la respuesta generada por el microprocesador cuando la señal es de voz, acoplándosele el micrófono, sin incorporar la función de almacenamiento en memoria en el microprocesador *(tiempo real),* haciendo uso del osciloscopio.
- Posteriormente se implementan las conexiones al jack de audio. En este punto, aunque sin almacenamiento de la señal se debe obtener ya la salida de audio desada.
- Finalmente se incorporan la función de almacenado en la memoria dentro del microprocesador y todos los elementos accesorios (botones, LEDs) que sirven para activar/desactivar las opciones de grabado/reproducido. Se realizan los testeos finales

6. Diagrama de bloques del circuito.



7. Descripción del circuito[1]

NOTA: En las figuras aparece el circuito idéntico a como ha sido construido, es decir, sin simplificar resistencias en paralelo o en serie, por encontrarlo más ilustrativo del trabajo realizado.

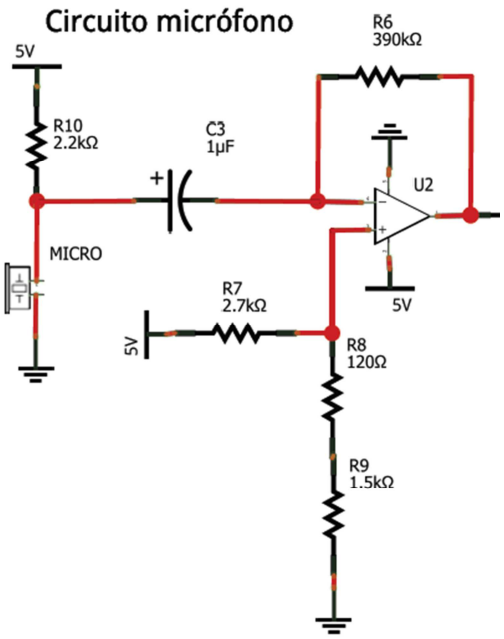
Adecuación de la entrada:

La entrada estaría compuesta por el micrófono, con su correspondiente amplificación y un filtro amplificador pasobajo de segundo orden, que serviría para atenuar las frecuencias de a partir de 3.8 kHz. La señal debe tratarse para que a la entrada del microcontrolador la señal corresponda a 3.8 voltios para 70 dB de intensidad del sonido. 3.8 voltios es el límite efectivo de saturación de los amplificadores TLC272 para una alimentación de 0 a 5 V.

-Circuito del micrófono.

El micrófono proporciona una señal analógica de tensión muy pequeña, del orden de un milivoltio. Es necesario que la señal proporcionada por el micrófono pase por una etapa de amplificación. El circuito del micrófono está diseñado para que a la salida del amplificador tengamos una tensión de 3.8 Voltios cuando la intensidad es la máxima que hemos definido en las especificaciones (70 dB) y 0 para una intensidad muy baja (Se considera que sonidos de mayor nivel de intensidad no son deseados en grabaciones de voz humana, pues corresponden a ruidos estridentes como un tren o un avión). Se coloca un voltaje de offset de 1.9 voltios para que la señal oscile entre el límite superior del amplificador y 0 voltios.

[1] Se incorpora en un anexo especial el esquema del circuito, para que se pueda ver con mayor resolución



-Filtro pasa bajos *antialiasing*.

Para evitar problemas de *aliasing*, es necesario un filtro pasobajo con frecuencia de corte de 3.8kΩ (y por supuesto, de ganancia 1). El filtro que se ha escogido es un filtro de segundo orden tipo Bessel por tener una respuesta suave en la banda de paso.

La posible función de transferencia del filtro es:

$$H(s) = \frac{\omega_0^2}{s^2 + a\omega_0 s + \omega_0^2}$$

Donde *a* en un filtro Bessel tiene el valor de 1.414, y la frecuencia crítica,

$$\omega_0 = \omega_c \cdot K_0$$

Donde ωc es la frecuencia de corte, en nuestro caso 3.8 kHz y K_0 es un parámetro que vale 1.274 para el este filtro.

La función de transferencia aplicada a las especificaciones de nuestro circuito podría ser:

$$H(s) = \frac{9.252 \cdot 10^8}{s^2 + 52680s + 9.252 \cdot 10^8}$$

El filtro responde a una tipología de Sallen-Key, por tanto podemos poner la función de transferencia en función de los elementos del circuito:

$$H(s) = \frac{\frac{1}{R_4 R_6 C_1 C_2}}{s^2 + \frac{R_4 + R_6}{R_4 R_6 C_2} s + \frac{1}{R_4 R_6 C_1 C_2}}$$

Igualando las funciones de transferencia:

$$\frac{1}{R_4 R_6 C_1 C_2} = 9.252 \cdot 10^8$$

$$\frac{R_4 + R_6}{R_4 R_6 C_2} = 52680$$

A continiación se muestra un esquema de los parámetros básicos de un filtro Bessel y cómo se han adaptado al proyecto:

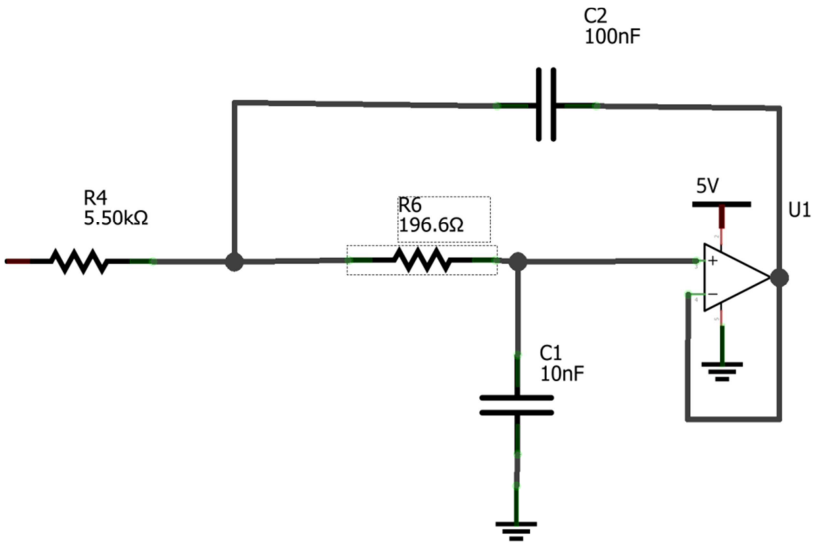


Figura 1: Filtro en tipología básica Sallen-Key

En el esquema anterior se han representado los valores de los parámetros que teóricamente se tendrían que utilizar para tener una frecuencia de corte de 3.8 kHz. En este caso, para unos valores de los condensadores fijados (10 y 100 nF), las resistencias teóricas son:

$$R_4 = 5.50\ k\Omega$$

$$R_6 = 196.6\ k\Omega$$

Las utilizadas en el proyecto son $R'_4 = 5.5\ k\Omega$ y $R'_6 = R_1||(R_3 + R_2) = 187.5\ k\Omega$

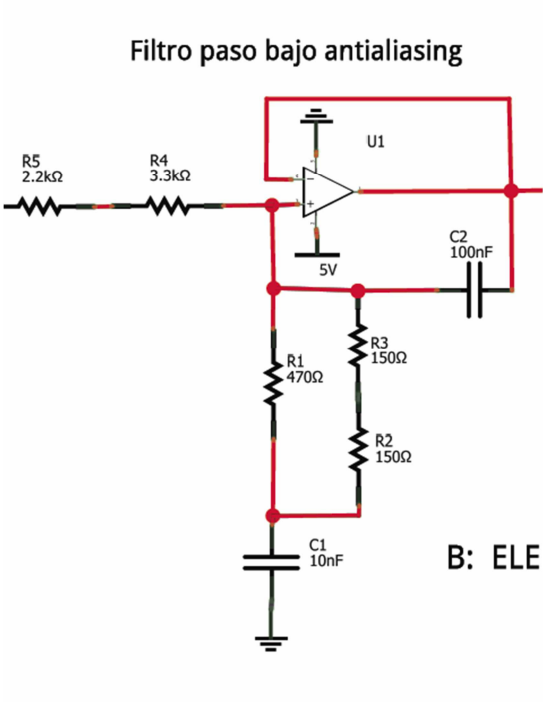


Figura 2: Esquema circuital del filtro paso bajo

El valor de la frecuencia de corte que se obtiene es el esperado. Lo comprobamos mediante un programa realizado en *LabVIEW* para representar diagramas de bode:

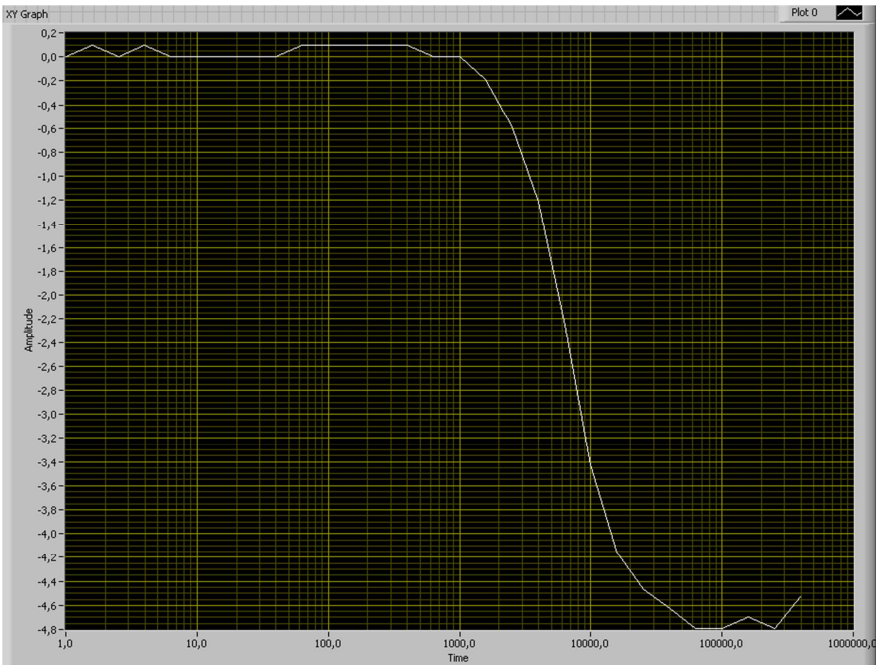


Figura 3: Diagrama de Bode del módulo de la función de transferencia del filtro paso bajo

Se observa claramente que el valor de caída a -3 dB es aproximadamente 3.8 kHz.

- ***Dificultades y cambios en las especificaciones en el circuito de entrada.***

-En primer lugar hemos hecho cambios por lo que concierne a la frecuencia de corte. Debido al límite de memoria que teníamos en el primer procesador (*Arduino*) era más conveniente reducir de 5 kHz a 3.8 kHz, para poder así muestrear a menos frecuencia[2]. La reducción del ancho de banda afecta a la calidad del sonido, pero en este caso no tiene mucha importancia, como ejemplo podemos decir que el ancho de banda de telefonía se sitúa en el orden de magnitud de los 3 kHz.

-En segundo lugar, hemos cambiado el límite de intensidad del micrófono de 100 dB hasta 70 dB. No tenía sentido definir el límite en 100 dB ya que el proyecto no se utilizará para grabar sonidos de estas intensidades (perforadora eléctrica, motor de avión …). El limitar a 70 la intensidad medida del micrófono nos permite aumentar la sensibilidad.

-Cortocircuitamos sin querer un amplificador, se quemó y tuvimos que sustituirlo por otro amplificador.

Tratamiento de la salida

El circuito de salida lo podemos dividir en tres partes: una parte de conversión digital/analógica, otra parte de suavizado de señal mediante filtrado, y finalmente un la incorporación de una entrada de Jack para audio. El volumen lo controlaremos mediante un potenciómetro.

NOTA: Previamente a la exposición del software, y por tanto de los procesos que lleva a cabo el microcontrolador, sólo ha de saberse que a su salida, el microcontrolador tan sólo puede generar señales digitales de 5V en sus pins.

Conversor D/A

Puesto que el microcontrolador no cuenta con uno, se debe construir uno de factura proua. El diseño que hemos utilizado es del tipo R-2R, como se observa en la figura.

[2] La frecuencia de muestreo tiene que ser mínimo 2*(Bandwidth) para que no se produzca *aliasing.*

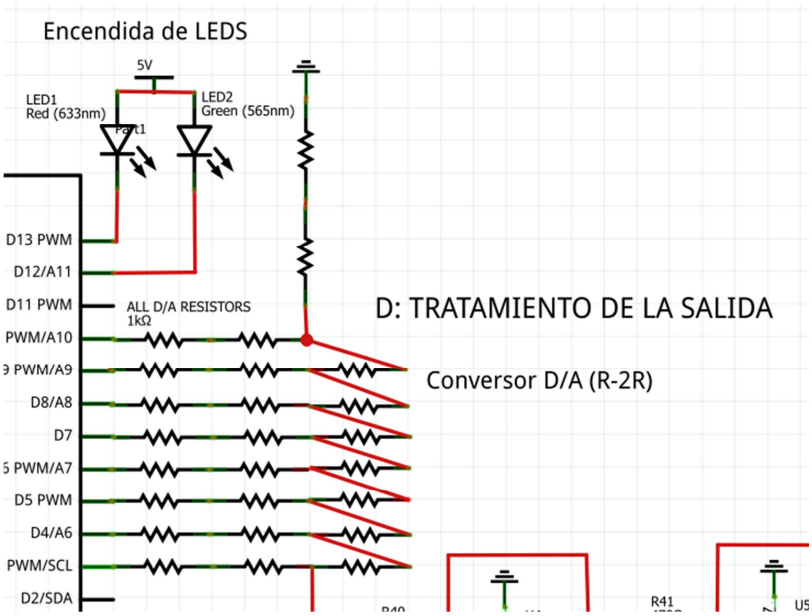


Figura 4: Representación del conversor digital/analógico

Dentro de los tipos de conversores que podíamos construir estaba el conversor de *escalera* y el conversor de tipo *R-2R.* El conversor en *escalera* era bastante incómodo porque necesitábamos resistencias de valores muy concretos del tipo $2^n \cdot R$. La alternativa era usar un conversor de tipo R-2R, con un solo valor de resistencia, en este caso igual a $1\ k\Omega$. Entre las condiciones que tiene el hecho de usar un conversor R-2R está el que las resistencias sean muy precisas y con una dispersión en su valor muy baja. Para ello se ha tenido que comprar resistencias de tolerancia del 1% y usarlas todas de una misma serie. Se ha escogido un conversor de 8 bits precisamente porque en un conversor de 10 bits si la tolerancia de las resistencias es del 1%, el error esperable por la tolerancia ya es mayor que la precisión que se pretende conseguir con los 2 bits menos significativos.

- ***Dificultades y cambios en las especificaciones de la conversión.***

La dificultad más grande que hemos tenido es el montaje físico del conversor en la *protoboard*. Para poder realizar lo menos lioso y lo más claro posible para poder trabajar con él, es intentar que todas las entradas que se unen a los pines digitales estén en el mismo lado de la *protoboard*. Al ser un conversor de 8 bits para sólo 10 líneas con bornes es imposible construir un conversor D/A de este tipo sin colocar en el mismo borne dos resistencias.

Por falta de espacio físico para conectar las resistencias que unen cada dígito (las resistencias R, dentro del conversor R-2R) se han tenido que sacar cables a la parte izquierda de la *protoboard* para poder conectarlas.

El conversor que hemos construido es un poco complicado en el sentido de que hay que intentar mantener separadas las resistencias asociadas a diferentes bits en un espacio de margen de pocos milímetros (lo que separa una línea de bornes de la otra).

No ha sido trivial darse cuenta de por dónde salía la señal convertida. Por todo lo demás el conversor ha sido lo que menos se ha modificado físicamente.

Se ha tenido que comprar una segunda *protoboard* nueva para ubicar el conversor.

Filtro suavizador pasa bajos (anti-imaging)

El filtro suavizador se ubica justo después del conversor para redondear el escalonado que produce la conversión D/A. Se trata de un filtro paso bajo en cascada de segundo orden con el mismo polo.

$$H = \frac{1}{(R_{40}C_4 s + 1)(R_{41}C_5 s + 1)} = \frac{1}{(4.7 \cdot 10^{-5} s + 1)^2}$$

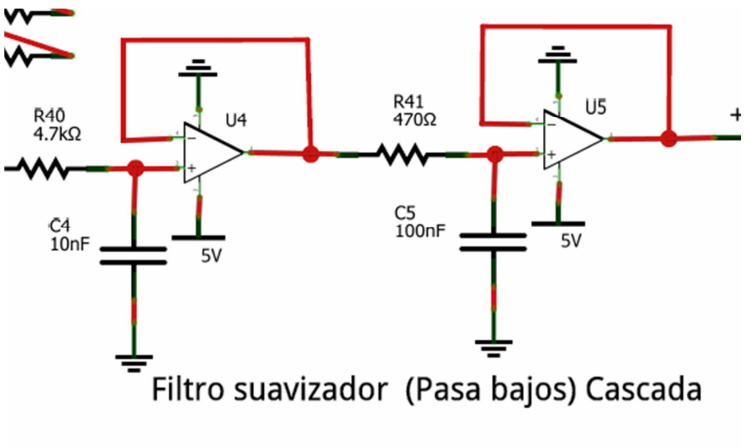


Figura 5: Filtrado de la señal de salida

Entrada del Jack.

A continuación, después del filtrado de salida, se coloca un condensador electrolítico para eliminar la componente residual de corriente continua y a través de un potenciómetro regulador de intensidad se envía la señal a un conector Jack, a los que se conectará unos auriculares para la audición.

Elementos de control

Botones de selección.

Se han dispuesto dos botones de control para configurar el programa del microprocesador en modo grabación y reproducción. La disposición de los botones es la siguiente:

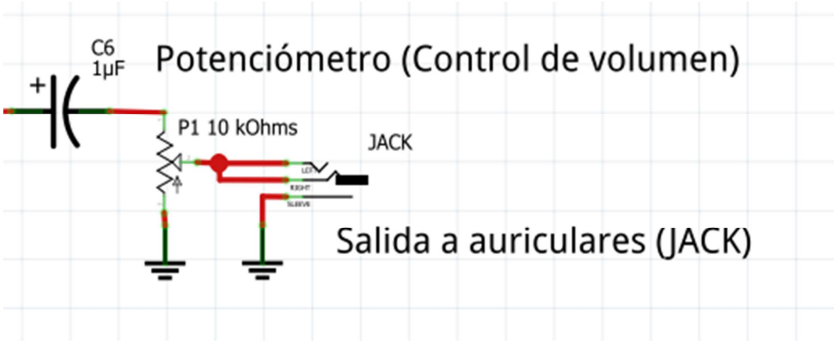
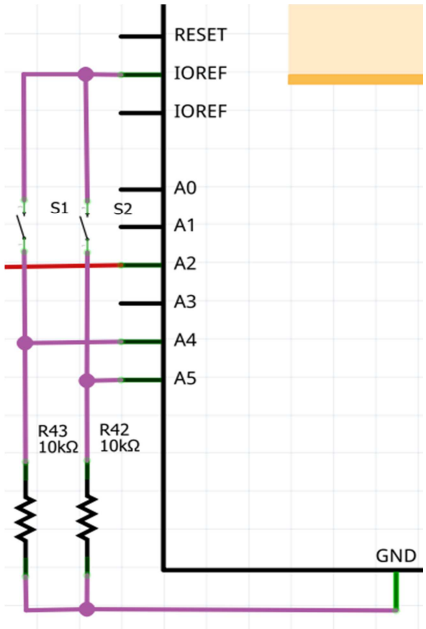


Figura 6: Circuito de entrada del Jack.



Podemos observar que se han dispuesto dos resistencias de pulldown de 10 kΩ, para la descarga de la corriente cuando se pulsa el botón. SI se pulsa S1 se empieza a grabar (se enciende el LED azul), si por el contrario se pulsa S2, se reproduce lo que había almacenado en la memoria (se enciende el LED rojo).

Encendido de LEDs.

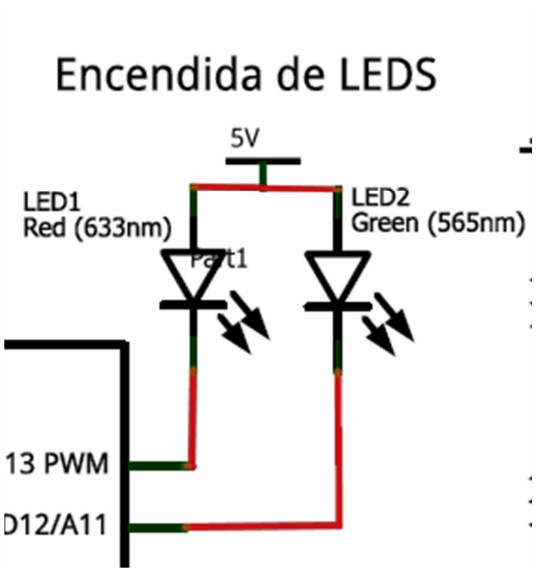


Figura 7: Esquema de la conexión de los LEDs a la placa *Pinguino*.

Se han dispuesto dos LEDs indicadores de si nos encontramos en modo grabado o reproducción, encendiéndose el LED azul y rojo respectivamente. El común está conectado a 5V de forma que si la tensión de salida en el pin del microcontrolador es nula el LED se encenderá y si es de 5V se mantendrá apagado.

8. Gestión de la memoria. Algoritmo DPCM (Differential pulse-code modulation). Primer diseño

La memoria RAM del microcontrolador *Arduino* Leonardo es de tan sólo 2.5 Kbytes. Y si la frecuencia de muestreo es de 8 KHz, almacenar directamente el valor proporcionado por el conversor analógico-digital del propio controlador (es un conversor de 10 bits) supondría ocupar 2 bytes por medida (una variable int usa 2 bytes en *Arduino*). Si se quiere un tiempo de grabación mínimo se debe optimizar esta fórmula.

Para ello se implementará un algoritmo DPCM (comúnmente usado en almacenamiento de audio), que se caracteriza por no guardar los valores medidos sino la diferencia entre la nueva medida y la anterior, con la esperanza de que sea un valor mucho menor que la medida original y que, por tanto, requiera menos bits para almacenarse. A continuación se describe el funcionamiento básico y se ejemplifica en una tabla.

- A un valor "anterior" se le suma el error de la iteración anterior
- Se calcula la diferencia entre la medida actual y el valor "anterior"
- Si la diferencia es menor que -7 o mayor que 8 se guarda el -7 u 8 y también el error cometido
- Al array de almacenamiento se añaden 4 bits conteniendo el valor de la diferencia más 7 (es decir, un número entre 0 y 15). Al dedicar 4 bits a cada medida se está reduciendo en un factor la memoria necesaria, y por tanto, se cuadriplica el tiempo de grabación.

Anterior	Muestra actual	Diferencia real	Diferencia admisible	Desviación	Nuevo anterior
172	181	+3	+3	0	181
181	190	+9	+7	2	189
189	194	+5	+5	0	194
194	187	-7	-7	0	187

A la diferencia admisible se suma 8 para obtener un valor que está ente 0 y 15 obteniendo 11, 15, 13 y 1 respectivamente. Es decir: 1011, 1111, 1101 y 0001. De estas cuatro medidas se obtendrían los 2 bytes 1011111 y 11010001 que corresponden a los números 191 y 209.

9. Problemas en el planteamiento inicial y solución: compresión no lineal

El método para establecer las diferencias entre medidas para luego obtener un valor de 10 bits tras la demodulación presenta un inconveniente serio. En teoría, la señal de entrada puede pasar en un caso extremo del valor mínimo (0 voltios) al valor máximo (3.9 voltios aproximadamente) en n muestras. Del programa se espera por tanto que sea capaz de reproducir este cambio al menos en un número aceptable de iteraciones, esto es, por ejemplo, en entre n y 2n muestras, pues en caso contrario la señal de salida se desviará demasiado de la señal que se quiere reproducir.

A una frecuencia de muestreo de 8kHz y suponiendo una frecuencia de corte de la entrada de 4kHz se deberían de poder representar dos cambios de mínimo a máximo por muestra en casos extremos. Pasar de 0 a 3.9V corresponde a una diferencia (en digital) de 800 mientras que la diferencia máxima admisible es de 8 (un 1%). Esto es inadmisible.

Si se quisiera aumentar la frecuencia de muestreo hasta la diferencia real máxima entre muestras fuera de 8, la frecuencia sería ridículamente alta. De esta manera, para solucionar el problema primero se reescalan las medidas a 8 bits y se ajusta el rango a 0-3.9V de 0-5V, pasando de una diferencia máxima de 800 a una de 256 (para ello basta con multiplicar por $\frac{5}{3.9 \cdot 4}$). Y además, se empleará una compresión no lineal.

Se seguirán almacenando valores comprendidos entre 0 y 15 pero que significarán una diferencia de entre 0 y 15 sino que harán referencia a otros valores de una tabla de referencia.

TABLA DE REFERENCIA	
Posición	**Valor**
0	-128
1	-64
2	-32
3	-16
4	-8
5	-4
6	-1
7	0
8	1
9	4
10	8
11	16
12	32
13	64
14	128
15	256

Cuando el programa calcula la diferencia entre dos medidas, busca dentro de la tabla la posición que corresponde con el valor más cercano y menor en valor absoluto. Por supuesto, al quedarnos con la aproximación de la tabla se está cometiendo un error. Para solventarlo, cuando se calcule la nueva diferencia se restará a la nueva medida el valor que se obtendría al demodular usando las referencias a la tabla. Nótese que con esta compresión no lineal, se puede reproducir una diferencia de tensión de 0 a 3.9 V (que representan todo el rango del conversor, es decir, 256 valores) en tan sólo 2 iteraciones. A continuación se muestra una tabla que ejemplifica el algoritmo:

Anterior	Muestra actual	Diferencia real	Diferencia admisible	Desviación	Nuevo anterior
23	45	22	16	6	39
39	73	34	32	2	71
71	62	-9	-8	-1	63
63	63	0	0	0	63

Los valores almacenados corresponderán a las posiciones de 16, 32, -8 y 0 dentro de la tabla de referencia, es decir, 11, 12, 4 y 7 respectivamente (en binario, 1011, 1100, 0100, 0111). De las cuatro medidas se obtendrían los 2 bytes 10111100 y 01000111 que corresponden a los números 188 y 71.

Para la implementación del sistema en el programa se construye tres funciones. La primera de ellas, la función “conversión”, devuelve para un cierto entero “valor” la posición dentro de la tabla de referencia que corresponde al valor más cercano siempre menor en valor absoluto. Esto quiere decir que si superpusiéramos la señal de entrada con la señal reconstruida, la reconstruida siempre quedaría “dentro” de la original y estaría acotada por ella.

```
int conversion(int valor)
{
if (valor==0) return 7;
if (valor > 0)
   for (kk = 15; kk > 7; kk--)
     if (valor > funcion[kk])
       return kk;
if (valor < 0)
   for (kk = 0; kk < 7; kk++)
     if (valor < funcion[kk])
       return kk;
}
```

Donde “funcion” es un array de enteros que se corresponde con la tabla de referencia. Obsérvese que no se recorre el array entero sino la parte correspondiente a valores negativos o a positivos según corresponda para mayor eficiencia.

La función “modDPCM” almacena la diferencia entre una medida Y[0] y “anterior” y posteriormente entre otra medida Y[1] y el nuevo “anterior” siguiendo la compresión no lineal expuesta anteriormente. Si es la primera medida de todas (nn = 0), se almacena sin procesar pues será necesaria para la reconstrucción. Si se alcanza la capacidad del array de bytes “medida” (nn == NUMBYTES) el aparato sale de modo grabación y se apaga el correspondiente *LED*.

```
void modDPCM()
{
if (nn == 0)
   {
   medida[0] = anterior1 = (Y[1] * 5 / 4) / 4;
   }
  else
   {
   for (kkk = 0; kkk < 2; kkk++)
       {
       D[kkk] = conversion((Y[kkk] * 5 / 4) / 4 - anterior1);
       anterior1 = anterior1 + funcion[D[kkk]];
       }
   medida[nn]=(D[1] << 4) | D[0];
   }
nn++;
if (nn==NUMBYTES)
     {
     modo=MODO_NADA;
     digitalWrite(11,HIGH);
     }
}
```

De forma análoga, también se genera una función demoduladora “demodDPCM” que a partir de un byte del array “medidas” y un valor “anterior2” almacenado en la memoria, reconstruye dos valores de la señal YY[0] e YY[1].

```
void demodDPCM(){
 if (jj == 0){
  YY[0] = YY[1] = anterior2 = medida[0];
 }
 else{
  for (k = 0; k < 2; k++){
    DD[k] = (medida[jj] >> 4*k) & 15;
    YY[k] = anterior2 + funcion[DD[k]];
    anterior2 = YY[k];
  }
 }
 jj++;
 if (jj==NUMBYTES)
    {
    modo=MODO_NADA;
    digitalWrite(10,HIGH);
    }
}
```

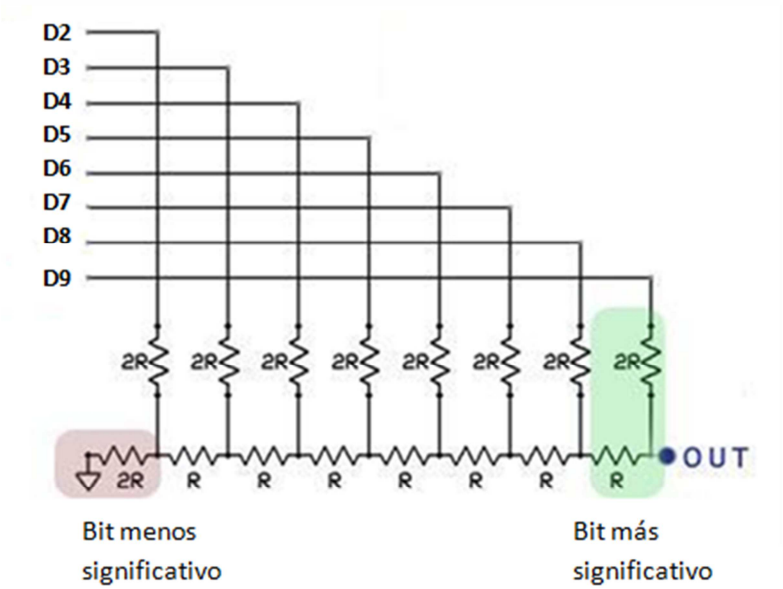
10. Modulación PWM frente a conversión analógica

El microcontrolador no cuenta con conversor digital-analógico. Frente a la construcción de un conversor propio (lo que consistiría en copiar su diseño de cualquier libro de texto) nos parece interesante utilizar señales de modulación PWM (Pulse Width Modulation)[3]. La ventaja de este método es que una vez implementada la modulación en el programa del microcontrolador no es necesario ningún montaje adicional, pues si la frecuencia PWM es suficientemente alta, el sonido generado por un altavoz al conectar a la señal PWM será muy similar al resultante de conectarlo a la señal analógica.

Nótese que para que sea efectiva, la señal PWM debe tener una frecuencia muy superior a la de muestreo (80KHz sería una frecuencia adecuada para nuestro caso). A su vez, la calidad del sonido dependerá de la precisión que se consiga alcanzar en la modulación. Supongamos que queremos una precisión "equivalente" a un conversor de 8 bits: cada periodo debe poder subdividirse en 255 partes lo que equivale a una "frecuencia de resolución" de $80 \cdot 255 = 20400 KHz$. Dejando a un lado que *Arduino* no cuenta con relojes preprogramados con precisión de nanosegundos (lo que nos forzaría a programar en bajo nivel), si se tiene en cuenta que la frecuencia del procesador es de 16MHz alcanzar esta resolución es teóricamente imposible.

La alternativa, como se ha mencionado, es construir un conversor digital-analógico de 8 bits. Además de por las razones expuestas en el apartado anterior "Problemas en el planteamiento inicial y solución", no se opta por una resolución superior (de 10 bits) porque la tolerancia de las resistencias debería de ser muy pequeña (del 0.1%) para obtener realmente esa precisión. El conversor (tal y como ya se ha detallado anteriormente) se ha construido con resistencias de 1KΩ con tolerancia del 1% según el diseño que se muestra esquemáticamente a continuación.

[3] Como ya se ha explicado en entregas anteriores del proyecto, este sistema consiste en encriptar la información en el ciclo de trabajo de una señal cuadrada. Por ejemplo, si para valores de voltaje entre 0 y 5V se quieren representar 3V. En un periodo de la señal, el 60% habrá tensión de 5V y en el 40% tensión nula.



Así pues, en el programa se incluye la función void CDA(int salida) que para un número entero "salida" produce en los pins D2-D9 5 ó 0 voltios según el bit asociado a cada pin sea 1 ó 0. Esto se consigue comparando el entero "salida" mediante el operador bit a bit AND con 1, 2, 4, 8, 16, etc.

```
void CDA(int salida){
  digitalWrite(2,salida & 1);
  digitalWrite(3,(salida & 2)>>1);
  digitalWrite(4,(salida & 4)>>2);
  digitalWrite(5,(salida & 8)>>3);
  digitalWrite(6,(salida & 16)>>4);
  digitalWrite(7,(salida & 32)>>5);
  digitalWrite(8,(salida & 64)>>6);
  digitalWrite(9,(salida & 128)>>7);
}
```

Con el osciloscopio se determina que el tiempo de ejecución de esta función en el *Arduino* es de 30 microsegundos aproximadamente. Este tiempo podría reducirse drásticamente usando programación de bajo nivel; pues mientras que la función digitalWrite() es muy lenta, el procesador Atmega 32u4 del *Arduino* cuenta ya con 5 puertos de entrada/salida de 8 bits numerados de la B a la F, conectados el B y el D a 8 lineas. Usando estos dos puertos[4] se reduce la operación a dos ciclos del procesador (se necesita tan solo un ciclo para reescribir cada uno de los puertos).

[4] Se necesitan los dos porque aunque ambos están conectados a 8 líneas, sólo 4 de cada uno son líneas no usadas.

11. Problemas de RAM y de tiempo de ejecución del programa.

La memoria RAM del *Arduino* Leonardo es de 2.5kBytes. Si se tiene en cuenta que la frecuencia de muestreo es de 8kHz y que en un byte se almacenan 2 medidas, el tiempo máximo de grabación es de $\frac{2\left(\frac{medida}{byte}\right)}{8kHz} 2.5kBytes = 0.625s$ sin tener en cuenta que no toda la memoria se puede dedicar a almacenamiento sino que una parte será necesaria para variables de cálculo.

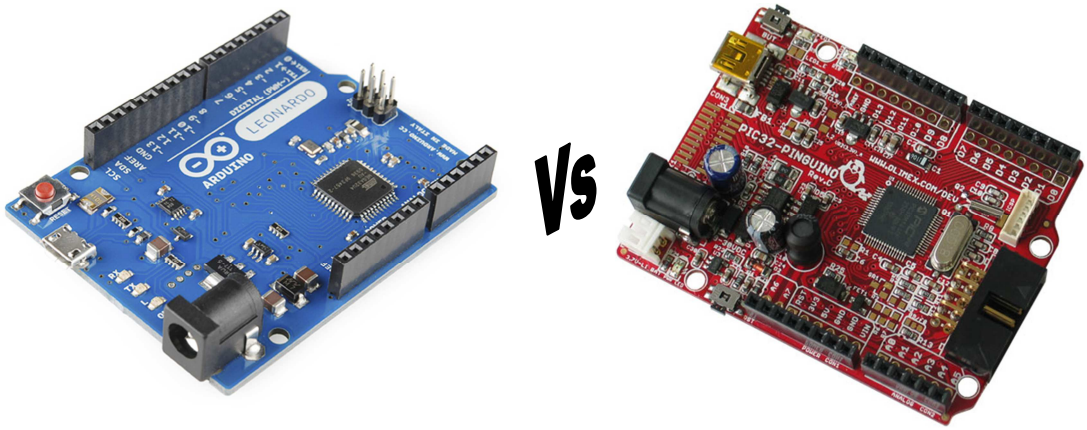
Medio segundo es un tiempo de grabación muy insatisfactorio. Sin renunciar a calidad de sonido (es decir, sin reducir la frecuencia de muestro), la solución pasa por aumentar la memoria RAM, para lo cual hay dos opciones: incorporar una tarjeta ScanDisk de memoria adicional (lo que traería la dificultad añadida de acceder a dicha memoria en el programa) o usar un modelo distinto de microcontrolador (con los posibles problemas de compatibilidad para el funcionamiento del código desarrollado en *Arduino* en la nueva plataforma).

Otro posible problema por incapacidad del hardware de *Arduino* sería el tiempo de ejecución del programa. El programa tiene un reloj de microsegundos y a cada ejecución se comprueba si han pasado o no 125 microsegundos (periodo correspondiente a una frecuencia de 8kHz) desde la última medida. A cada ejecución el programa deberá tomar una medida o reproducir un valor y cada dos ejecuciones comprimir o descomprimir dos medidas según se esté grabando o reproduciendo. Si el tiempo que necesita el procesador para realizar estas operaciones es superior a 125 microsegundos no se estará muestreando a la frecuencia deseada.

Para estudiar este fenómeno, la primera versión funcional del programa grababa y reproducía a la vez, comprimiendo y descomprimiendo en la misma ejecución pero sin almacenar. Es decir, las medidas se incluían siempre en un mismo byte que, por tanto, a cada par de muestras se actualizaba. Nótese que en esta versión las distintas operaciones (modular, demodular, enviar al conversor D/A, etc.) las realizaban funciones independientes, por lo que el código era reutilizable. Probando para distintas frecuencias de muestreo (partiendo de frecuencias bajas) y observando en el osciloscopio la salida del conversor, se observó que el tiempo que se mantenían los valores correspondientes a las ejecuciones con modulación-demodulación no bajaba de un valor fijo muy superior a los 125 microsegundos (en torno a 350 microsegundos).

Llegados a este punto, y aunque hubiera sido posible pulir el código para reducir suficientemente el tiempo de modulación, decidimos cambiar el *Arduino* Leonardo por un microcontrolador más potente: el PIC32 *Pinguino* OTG; solucionando así directamente todos los problemas anteriores.

12. Nuevo microcontrolador y adaptación del programa. Peculiaridades de PIC32 *Pinguino* OTG



La RAM insatisfactoria del *Arduino* y su incapacidad de computar las operaciones de modulación y demodulación a la frecuencia de muestreo deseada (esto se comprobó experimentalmente) motivaron la utilización de un microcontrolador superior, el *Pinguino* OTG PIC 32. Aun así, tanto el circuito como el programa han sido desarrollados sobre la placa con *Arduino*; el material debe ser adaptado posteriormente a la nueva plataforma.

	***Arduino* Leonardo**	***Pinguino* OTG PIC 32**
Frecuencia del procesador	16 MHz	80 MHz
Memoria RAM	2.5 KBytes	32KBytes
Memoria FLASH	32 KBytes	256KBytes

De cara al diseño del programa final, hay diferencias entre ambos microcontroladores que han llevado consigo cambios obligados en el código:

- En lo relativo a la gestión de los interruptores, en el entorno *Arduino*, al declarar una línea como línea de entrada se puede especificar por ejemplo "pinMode(1,INPUT_PULLUP);" y se puede conectar directamente a ese pin el interruptor conectado a su vez a un tensión dada (5V), mientras que en el entorno *Pinguino* no existe una instrucción para generar un pullup interno y se deben implementar resistencias de pullup externas para los pulsadores.
- En lo relativo a la llamada de funciones mediante interrupciones, mientras que en *Arduino* existen funciones como attachInterrupt() que llaman a una función cuando se produce una interrupción (baja o aumenta la tensión) en un determinado pin o MsTimer que permite introducir una llamada periódica a otra función no existe ninguna de ellas en *Pinguino*. Esto no supone un gran problema, el programa usará una variable "modo" de cuyo valor dependerá

qué funciones llama el bucle. A cada ejecución del mismo, el valor de "modo" se actualiza en función según su propio valor en la ejecución anterior y según la tensión en dos pins de control (conectados a los pulsadores con pullup externo) obteniendo el mismo resultado que con las funciones de interrupción de *Arduino*.

- Existen otras diferencias menores como que la IDE de *Pinguino* no permite usar como tamaño de arrays variables const (por esa razón, en el código final NUMBYTES está declarado con #define).

Nótese que:

- Aunque la frecuencia del nuevo procesador posibilitara (en teoría) generar señales PWM con 80kHz de frecuencia y una resolución de 8 bits, en el diseño final se emplea el conversor desarrollado para las pruebas con el *Arduino* por simplicidad y su funcionamiento satisfactorio.
- Aunque en el apartado "Modulación PWM frente a conversión analógica" se apunta a la posibilidad de reescribir el código del conversor DA en lenguaje de bajo nivel para aprovechar más eficientemente los puertos, esto no será necesario pues el nuevo procesador es suficientemente rápido (el aparato funciona correctamente incluso para frecuencias de muestreo de 20kHz). Así pues, el intervalo de señal errónea desde la ejecución del primer digitalWrite() hasta el último (en este intervalo la señal toma un valor aleatorio que puede estar muy alejado del valor real) es imperceptible.

13. Métodos de testeo y algunos bugs importantes durante el desarrollo

Se ha intentado desarrollar el proyecto de manera modular, es decir, en módulos que cumplen una función específica dentro del aparato y que pueden ser testeados por separado. De cara a al software, en este sentido, cada uno de los procedimientos ha sido implementado en una función que ha sido testeada de forma que en la versión final sólo habría que llamarlas en el bucle.

Además, como se ha comentado anteriormente, se desarrolló una primera versión del programa que grababa y reproducía simultáneamente. Esto es, las funciones modDPC, demodDPC, conversion y CDA mencionados han sido programadas y se las llama en el bucle para grabar, modular, demodular y reproducir a través del conversor a cada iteración. El objetivo era poder reproducir el funcionamiento completo de la grabadora final pero sin hacer uso de la memoria para almacenar datos.

Teniendo en cuenta que sólo se dispuso del microcontrolador *Pinguino* OTG PIC 32 hasta los últimos días de desarrollo, era fundamental poder testear tanto el código

como el circuito analógico y llegar a una versión final operativa en la placa con el *Arduino* Leonardo que tan sólo hubiera que trasladar al nuevo microcontrolador. Así pues, con dicha versión del programa se eliminan los problemas de memoria del *Arduino* y se llegó a un montaje plenamente operativo (aún con una cierta limitación en la frecuencia de muestreo de la grabadora debido a la inferior frecuencia del procesador).

Por otro lado, cabe destacar un bug en particular detectado al final del proyecto (ya con el *Pinguino* y el sistema de pulsadores y el uso de memoria implementados). Probando con los auriculares se observó que la grabación sonaba más aguda que el sonido real. Observando con detenimiento la señal de salida del conversor en el osciloscopio se observó que las iteraciones en las que se modula (1 de cada dos), el tiempo empleado era mayor que en el resto y también que el periodo de muestreo deseado (de 125 microsegundos). De esta forma, mientras que la frecuencia en la reproducción era la teórica, la de grabación era superior.

Llegados a este punto se testeó el tiempo de ejecución de cada una de las líneas de las funciones modDPCM y demodDPCM. Para ello se agregaron las siguientes líneas antes y después:

```
digitalWrite(12,HIGH);
// Instrucción
digitalWrite(12,LOW);
```

Donde el pin 12 está conectado al osciloscopio. Midiendo el tiempo del escalón, se observó la inmensa mayoría del tiempo de la iteración se empleaba ejecutando la instrucción:

```
D[kkk] = conversion((Y[kkk] * 5 / 3.9) / 4 - anterior1);
```

Pues aparece una operación con un número en coma flotante (3.9) al reescalar la medida a 8 bits. Aunque estas operaciones son más costosas que con enteros, se subestimó su impacto. Una vez detectado el bug, se sustituyo 3.9 por 4 lo que deriva en mínima pérdida de precisión pero que soluciona el problema. De hecho, se ha comprobado que el aparato funciona satisfactoriamente (después del cambio) para frecuencias de muestreo de 50 microsegundos como mínimo.

14. Programa final

El estado del aparato está gestionado por la variable “modo” que modificarán los pulsadores. Según su valor 0, 1, ó 2, el aparato estará en espera, grabando o reproduciendo respectivamente.

A cada ejecución del bucle se comprueba el tiempo (en microsegundos) que ha pasado desde que está corriendo el programa y la tensión actual en los pins asociados a los pulsadores. Si la tensión en el pin pasa de un nivel bajo a un nivel alto, la variable “modo” se modifica para indicar grabación o reproducción según corresponda. Nótese que cuando esto sucede, el programa también inicializa a cero todas las variables que emplean las funciones necesarias para grabar o reproducir (por ejemplo, cada vez que se pulsa el botón de grabar, el programa borrará el contenido del array “medidas” que almacenará los datos.

Si el modo es de grabación o reproducción, el programa comprueba cuanto tiempo ha pasado desde la última iteración de grabación o reproducción propiamente. Si es mayor o igual que el periodo de muestreo, se llaman las funciones necesarias para grabar:

- Leer el pin de entrada y guardar el valor en el array Y
- Cada dos valores medidos, se ejecuta modDPCM.

Y para reproducir:

- Se ejecuta CDA para reproducir a la salida del conversor la tensión equivalente a un valor del array YY
- Cada dos iteraciones, se ejecuta demodDPCM, que actualiza los valores de YY.

Cuando se recorre el vector “medidas” por completo, se pasa a modo de espera y se apaga el *LED* que estuviera encendido.

A continuación se muestra el código completo:

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

Author:  --<ANTONIO VALERA, XIMO GALLUD>
Date: Sun Dec 22 20:34:45 2013
Description: PROYECTO GRABADORA REPRODUCTORA

-----------------------------------------------*/


// Longitud del array de bytes de almacenamiento máximo que permite el microcontrolador
#define NUMBYTES 14000

// Distintos estados de funcionamiento del aparato
#define MODO_NADA 0
#define MODO_LECTURA 1
#define MODO_REPRODUCCION 2

int modo=MODO_NADA; // Variable indicadora del estado del aparato

const unsigned int periodo = 125; // Periodo de muestreo

unsigned long previousMicros = 0; // Reloj de microsegundos

// Contadores
int nn = 0; int mm = 0; int jj = 0; int kk = 0;
int kkk = 0; int k = 0;

// Tabla de referencia
int funcion[16] = {-128, -64, -32, -16, -8, -4, -1, 0, 1, 4, 8, 16, 32, 64, 128, 256};

// Variables para modulación y demodulación
int Y[2]; int D[2]; int YY[2]; int DD[2];
int anterior1,anterior2 = 0;

byte medida[NUMBYTES]; // Array de guardado de datos

// Variables que almacenan el estado de los pins de los pulsadores
int actRep, prevRep = LOW, actGrab, prevGrab = LOW;

// Función que devuelve la posición de la tabla de referencia del valor inmediatamente
// Menor en valor absoluto al de la variable "valor"
int conversion(int valor)
{
if (valor==0) return 7;
if (valor > 0)
   for (kk = 15; kk > 7; kk--)
     if (valor > funcion[kk])
       return kk;
if (valor < 0)
   for (kk = 0; kk < 7; kk++)
     if (valor < funcion[kk])
       return kk;
```

```
}

// Función moduladora
void modDPCM()
{
if (nn == 0)
   {
   medida[0] = anterior1 = (Y[1] * 5 / 4) / 4;
   }
  else
   {
   for (kkk = 0; kkk < 2; kkk++)
       {
       D[kkk] = conversion((Y[kkk] * 5 / 4) / 4 - anterior1);
       anterior1 = anterior1 + funcion[D[kkk]];
       }
   medida[nn]=(D[1] << 4) | D[0];
   }
nn++;
if (nn == NUMBYTES)
    {
    modo=MODO_NADA;
    digitalWrite(11,HIGH);
    }
}

// Función demoduladora
void demodDPCM(){
 if (jj == 0){
  YY[0] = YY[1] = anterior2 = medida[0];
 }
 else{
  for (k = 0; k < 2; k++){
    DD[k] = (medida[jj] >> 4*k) & 15;
    YY[k] = anterior2 + funcion[DD[k]];
    anterior2 = YY[k];
  }
 }
 jj++;
 if (jj==NUMBYTES)
    {
    modo=MODO_NADA;
    digitalWrite(10,HIGH);
    }
}

// Función genera a en el conversor una tensión equivalente al valor "salida"
void CDA(int salida){
 digitalWrite(2,salida & 1);
 digitalWrite(3,(salida & 2)>>1);
 digitalWrite(4,(salida & 4)>>2);
 digitalWrite(5,(salida & 8)>>3);
```

```
  digitalWrite(6,(salida & 16)>>4);
  digitalWrite(7,(salida & 32)>>5);
  digitalWrite(8,(salida & 64)>>6);
  digitalWrite(9,(salida & 128)>>7);
}

// Condiciones iniciales
void setup(){
  pinMode(2,OUTPUT); // 8 pins para el conversor de 8 bits
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT); // Dos pins para los LEDs
  pinMode(11,OUTPUT);
  digitalWrite(10,HIGH); // Los LEDs comienzan apagados
  digitalWrite(11,HIGH);
}

// Bucle
void loop()
{
unsigned long currentMicros; // Reloj de microsegundos

actRep=digitalRead(0); // Se comprueba el estado actual de los pins de los pulsadores
actGrab=digitalRead(1);

// Iniciar grabacion si en esta iteración se ha activado el pulsador correspondiente
if ((prevGrab==LOW)&&(actGrab==HIGH))
      {
      mm=0;
      nn=0;
      modo=MODO_LECTURA;
      digitalWrite(11,LOW);
      }

// Iniciar reproduccion si en esta iteración se ha activado el pulsador correspondiente
if ((prevRep==LOW)&&(actRep==HIGH))
      {
      mm=0;
      jj=0;
      modo=MODO_REPRODUCCION;
      YY[0]=0;
      YY[1]=0;
      digitalWrite(10,LOW);
      }

prevRep=actRep; // Se actualizan las variables del estado previo de los pins de los pulsadores
prevGrab=actGrab;
```

```
// Este módulo se ejectua sólo si ha pasado al menos "periodo" microsegundos desde su
// última ejecución
currentMicros=micros();
if (currentMicros - previousMicros > periodo)
     {
     previousMicros = currentMicros;
     if (modo==MODO_LECTURA)  // LECTURA
       {
       Y[mm] = analogRead(A0); // Se mide la tensión en el pin A0
       mm = (mm + 1) % 2;
       if (mm == 0) modDPCM(); // Cada dos medidas se modula.
       }
     if (modo==MODO_REPRODUCCION) // REPRODUCCIÓN
       {
       CDA(YY[mm]); // Se genera el valor reconstruido a la salida del conversor
       mm = (mm + 1) % 2;
       if (mm == 0) demodDPCM(); // Cada dos iteraciones se demodula.
       }
     }
}
```

15. Planificación final

El proyecto total lo hemos dividido en dos partes, tal y como se planificó, la A (GRABACIÓN → circuito y programación de la entrada y almacenamiento) y la B (REPRODUCCIÓN → Circuito y programación de la salida). Además incluimos dos etapas inicial y final (0 y 1) para la planificación de la compra de materiales y para tests globales y modificaciones, respectivamente.

Esquema de tareas:

PARTE 0.

0.1. Selección y compra de material.

PARTE A. GRABACIÓN.

AA.1[5]. Diseño final del circuito de filtrado y del micrófono.

AA.2. Implementación y construcción del circuito de entrada en la *protoboard.*

AA.3. Observación de la respuesta del circuito en el osciloscopio y posibles modificaciones.

AA.4. Diseño de los circuitos auxiliares. Diodo LED e interruptor SW1.

AD.1. Programación e implementación del muestreo.

AD.2. Programación e implementación del algoritmo DPCM y el almacenamiento en la memoria.

AD.3. Programación de los circuitos auxiliares. Diodo LED e interruptor SW1.

Parte B. REPRODUCCIÓN.

BD.1. Diseño del programa descodificador DPCM.

BD.3. Diseño de programa de control de circuitos auxiliares. LED y botón SW2.

BA.1. Diseño e implementación del filtro paso bajo adecuador de señal de salida.

BA.2. Diseño e implementación del conversor D/A y su representación en el programado.

Parte 1. TEST GLOBAL y REDACCIÓN del INFORME.

[5] *Código de enumeración:*
Primer carácter: Indica la parte (A o B).
Segundo carácter. Indica si forma parte de una implementación Digital (programada) (D) o analógica (montada) (A).
Número: Indica el número de identificación de la tarea.

1.0. Realización de tests generales y modificaciones.
1.1 Escritura del informe final.

Como se puede observar se ha sustituido la tarea de generación de PWM por una etapa de conversión D/A, tal y como se predijo en caso de que la primera opción no funcionase.

La modificación que se ha hecho en la planificación semanal ha sido básicamente el retraso de la redacción de la memoria del proyecto para navidad y para las dos primeras semanas de Enero, y el alargamiento del testeo, que prácticamente se ha realizado a la par con el desarrollo del proyecto.

A continuación se adjunta la tabla actualizada con la planificación temporal modificada[6]:

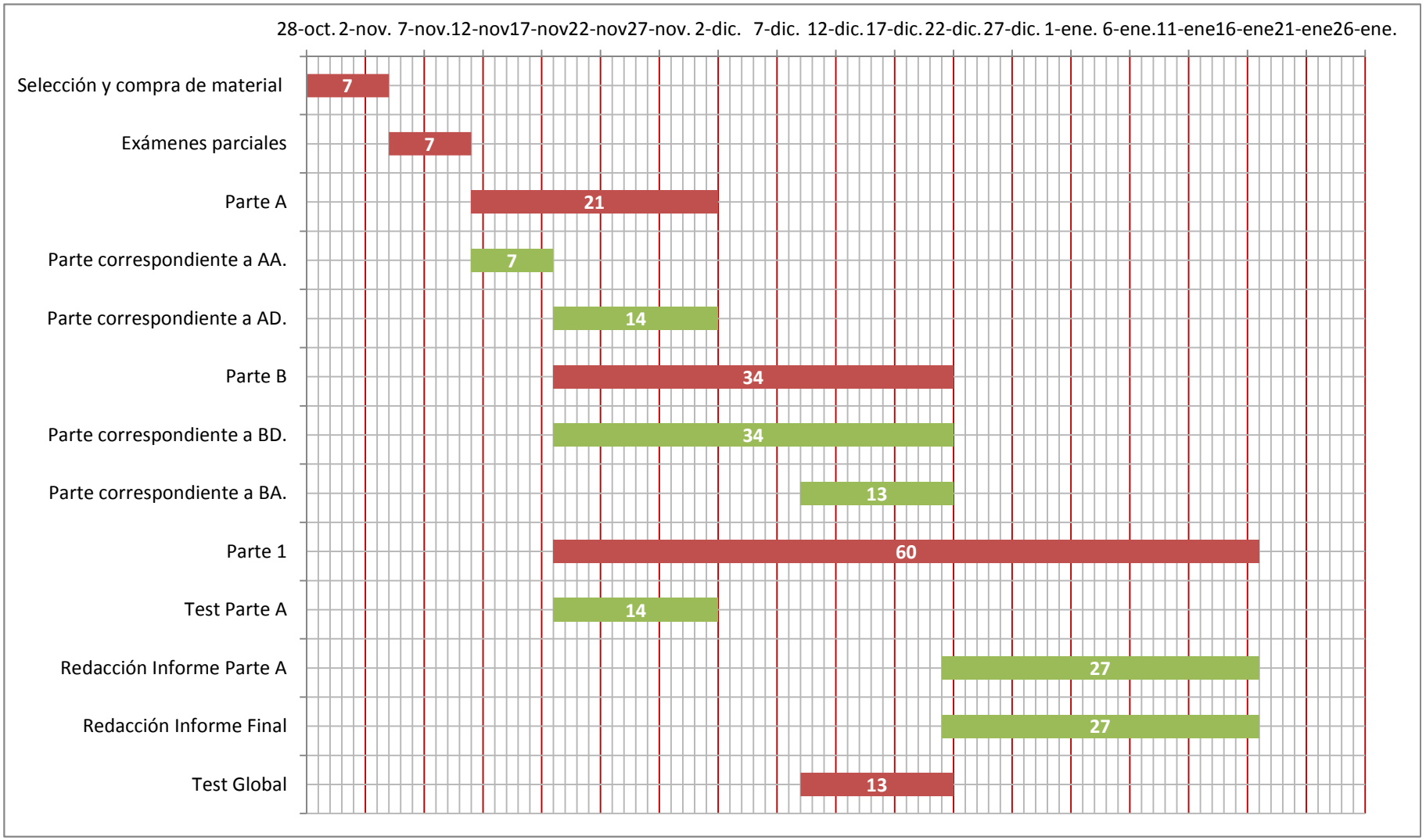
[6] Para ver con más detalle cuál ha sido la planificación temporal en cada semana véase documento *Planificación del proyecto.*

Tabla 1. Esquema de tareas ordenadas

TAREAS	ACTIVIDADES	FECHA INICIO	DIAS	FECHA FIN
Selección y compra de material necesario	**elección y compra de material necesario**	10/28/2013	7	11/3/2013
Exámenes parciales	**Exámenes parciales**	11/4/2013	7	11/10/2013
PARTE A				
Parte correspondiente a AA	AA.1. Diseño final del circuito de filtrado y del micrófono.	11/11/2013	7	11/17/2013
	AA.2. Implementación y construcción del circuito de entrada en la protoboard.	11/11/2013	7	11/17/2013
	AA.3. Observación de la respuesta del circuito en el osciloscopio y posibles modificaciones	11/11/2013	7	11/17/2013
Parte correspondiente a AD.	AD.1. Programación e implementación del muestreo.	11/18/2013	14	12/1/2013
	AD.2. Programación e implementación del algoritmo DPCM y el almacenamiento en la memoria.	11/18/2013	14	12/1/2013
	AD.3. Programación de los circuitos auxiliares. Diodo LED e interruptor SW1.	11/18/2013	14	12/1/2013
PARTE 1	Test de la parte A	11/18/2013	34	12/21/2013
	1.1 Redacción del informe. Parte correspondiente a AA.	12/21/2013	27	1/16/2014
	1.2 Redacción del informe. Parte correspondiente a AD	12/21/2013	27	1/16/2014
PARTE B				
Parte correspondiente a BD.	BD.1. Diseño del programa descodificador DPCM.	11/18/2013	14	12/1/2013
	BD.3. Diseño de programa de control de circuitos auxiliares. LED y botón SW2.	12/9/2013	13	12/21/2013
	BD.2. Test conversor y programa.	12/2/2013	7	12/8/2013
Parte correspondiente a BA.	BA.1. Diseño e implementación del filtro paso bajo adecuador de señal de salida.	12/9/2013	13	12/21/2013
	BA2. Diseño e implementación de un convertidor D/A	12/9/2013	13	12/21/2013
	Parte 1. Tests globales	12/9/2013	13	12/21/2013
PARTE 1	Finalización del informe	12/21/2013	27	1/16/2014

TAREAS	FECHA INICIO	DIAS	FECHA FIN
Selección y compra de material	28-oct.	7	03-11-13
Exámenes parciales	4-nov.	7	10-11-13
Parte A	11-nov.	21	01-12-13
Parte correspondiente a AA.	11-nov.	7	17-11-13
Parte correspondiente a AD.	18-nov.	14	01-12-13
Parte B	18-nov.	34	21-12-13
Parte correspondiente a BD.	18-nov.	34	21-12-13
Parte correspondiente a BA.	9-dic.	13	21-12-13
Parte 1	18-nov.	60	16-01-14
Test Parte A	18-nov.	14	01-12-13
Redacción Informe Parte A	21-dic.	27	16-01-14
Redacción Informe Final	21-dic.	27	16-01-14
Test Global	9-dic.	13	21-12-13

Gráfica 1: Tabla de Gantt

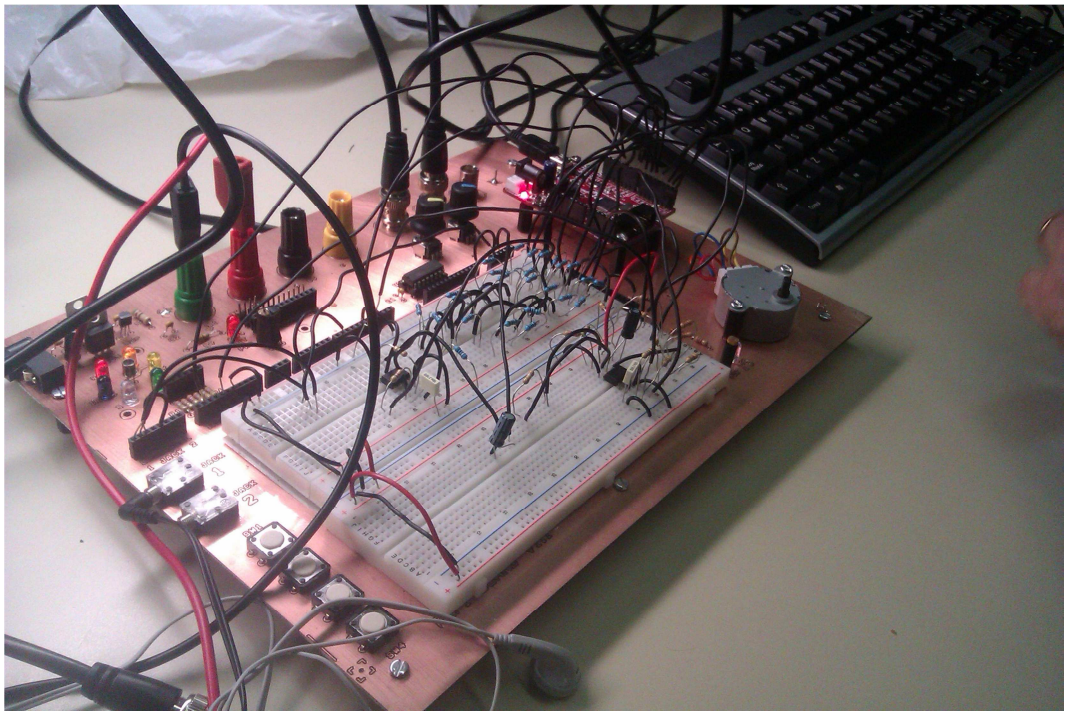


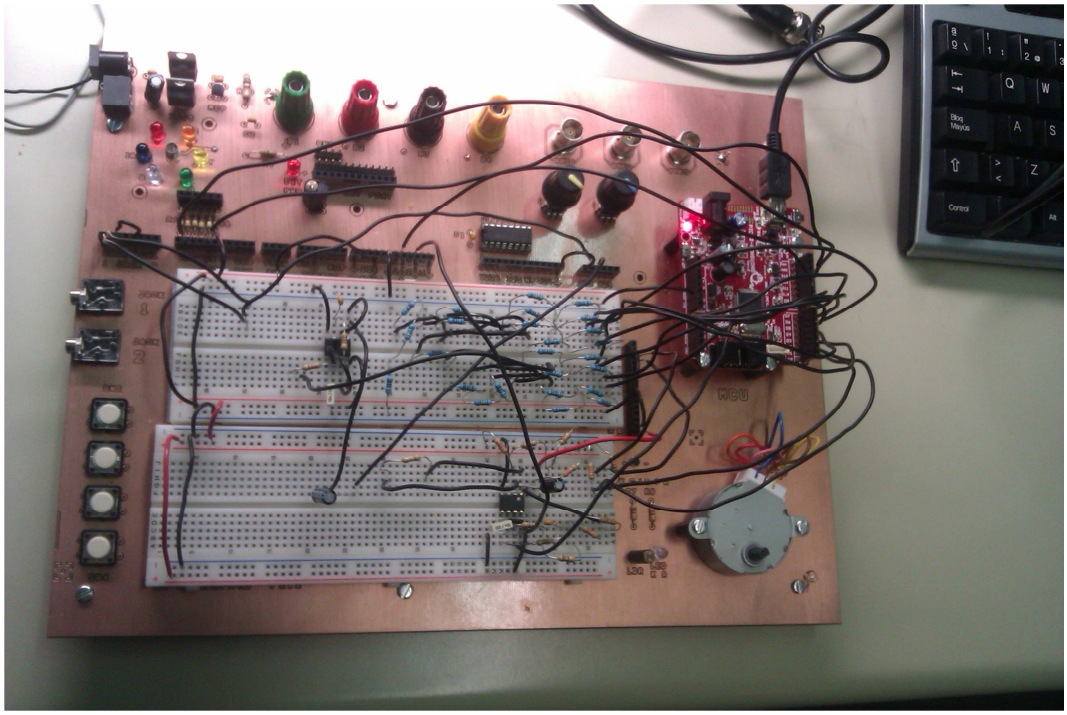
16. Conclusión y posibles mejoras del proyecto.

Aunque el resultado ha sido satisfactorio, el aparato graba 4 segundos aproximadamente de sonido mientras que el audio de salida es de calidad sorprendentemente buena (además, el volumen adecuado se encuentra dentro del rango de variación del potenciómentro del jack de audio y los pulsadores y *LED*s de control funcionan correctamente), es posible realizar algunas mejoras al proyecto.

- Una parte importante de la desviación de la señal de salida respecto a la de entrada es debida a la precisión del conversor. Emplear resistencias de tolerancia del 0.1% es más caro pero llevará a un resultado algo mejor. También se podría construir un conversor de 10 bits (que necesitaría obligatoriamente esa tolerancia en las resistencias del 0.1%) o comprar uno comercial.
- Por simplicidad, las funciones de modulación y demodulación se han estructurado para ejecutarse cada dos iteraciones para usar dos medidas. Esto se ha hecho así porque el tipo de variable en *Pinguino* y *Arduino* de menos memoria es el byte, de 8 bits, es decir, dos medidas. De esta forma, cada dos medidas, modDPCM y demodDCPM llenan o leen un byte. El problema es que si se quisiera llegar a frecuencias de muestreo mucho mayores que 8kHz, el tiempo de ejecución de estas funciones puede ser un factor limitante. Para solventarlo se podrían emplear a fondo los operadores bit a bit de c++ y modular y demodular en todas las iteraciones pero sólo una medida.
- Una ventaja de la grabadora reproductora desarrollada es que el microcontrolador empleado soporta para el mismo programa frecuencias de muestreo superiores a 8kHz (comprobado experimentalmente) y además, para ello sólo hay que cambiar el valor del parámetro “periodo” dentro del programa. Una mejora de cara a la búsqueda de mayor precisión o tiempo de grabado sería incorporar una interfaz para mostrar en el ordenador en la que se puede especificar la frecuencia de muestreo deseada sin tener que acceder al código.
- La distribución de valores la Tabla de referencia sigue aproximadamente las potencias de 2. El resultado es eficaz pero no se han explorado otros diseños durante el desarrollo. Un estudio del error derivado de ésta y otras propuestas de tablas (por ejemplo siguiendo una distribución exponencial) podría llevarnos a una tabla de referencia más eficiente.

17. Imágenes del proyecto





Se puede observar en las imágenes la configuración circuital. A la derecha de la placa se encuentra el microprocesador *Pingüino OTG.* En la *protoboard* superior se implementa el conversor D/A y los filtros antiimaging y los LED. La *protoboard* inferior sirve de soporte a los circuitos de entrada (micrófono y botones).

18. Anexos: Esquema del circuito

Se puede consultar el esquema general del circuito incluido en la carpeta general del proyecto. El esquema está realizado mediante la herramienta informática *Fritzing.*

19. Referencias

http://www.dialogic.com/~/media/products/docs/appnotes/10532_Dialogic_ADPCM_Algorithm_an.pdf

http://www.cs.columbia.edu/~sedwards/classes/2004/4840/reports/manic.pdf

http://www.ipcsit.com/vol37/045-ICINT2012-I3213.pdf

http://*Arduino*.cc/es/Tutorial/PWM

http://www.cika.com/soporte/TechComm/CTC-046_AudioMicrosPWM.pdf

http://playground.*Arduino*.cc/Code/PCMAudio

https://www.olimex.com/

Prácticas de Electrónica (Práctica A4). Proyectos ingeniería física 1. Grado en Ingeniería Física.

Apuntes de electrónica. Cirucuitos Electrónicos. Grado en Tecnologías Aeroespaciales.