UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T. Telecomunicación (Sonido e Imagen)

UNIVERSIDAD
POLITECNICA
DE VALENCIA

ESCUELA POLITECNICA
SUPERIOR DE GANDIA

Implementación de efectos de audio para trombón de vara en el dispositivo reprogramable "Chameleon".

TRABAJO FINAL DE CARRERA

Autor/es:
José Ribes Blanco

Director/es:
María Desamparados Girona Coma

GANDIA 2012

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS.

El desarrollo tecnológico ha estado presente en toda la historia de la humanidad, el fuego, la rueda, el motor a vapor, el transistor, etc. Un desarrollo con el fin de facilitar nuestro bienestar en este mundo, realizar el trabajo físico sin apenas esfuerzo. Un desarrollo que ha llegado hasta la era de "lo digital", de poder transformar una serie de palabras, de dibujos, de ondas, en ceros y unos. Por supuesto, las aplicaciones artísticas son una de las grandes aventajadas en este mundo digital, podemos ver cualquier cuadro de pintura y leer nuestra novela favorita en una pantalla, podemos hacer sonar cualquier instrumento con un teclado o sintetizador, y no solo un instrumento sino toda una orquesta sinfónica.

Centrándonos en el ámbito musical, antes de la era digital, las grabaciones de instrumentos musicales se realizaban en soportes magnéticos, en cintas, y la edición era bastante compleja, hasta el punto de tener que recortar trozos de cinta para eliminar cualquier instrumento mal grabado e ir empalmando con nuevos trozos bien grabados previamente, todo un trabajo artesanal. No existían efectos musicales como los que conocemos ahora, y si sonaban con efectos era porque el mismo músico añadía algún artefacto a su instrumento. Un claro ejemplo eran los guitarristas, que añadían trozos de metal a la caja acústica para hacer sonidos más metálicos y estridentes o los trompetistas de Jazz, que añadían tapas o embudos a la campana, enmudeciendo el sonido y haciendo el actualmente conocido “efecto de sordina”.

Hoy en día esta artesanía está olvidada, los unos y ceros nos han facilitado y agilizado el trabajo. Podemos editar cualquier tipo de música e instrumento, pasar cualquier señal por el efecto que deseemos, editar la onda que se describe de un instrumento e incluso dibujarla si está "sucia" o con ruido. Hemos avanzado tanto que no solo podemos filtrar las señales por cualquier efecto, sino que podemos crear hasta nuestros propios efectos para poder solucionar cualquier problema e imprevisto tal como mostraré en este

trabajo.

El objetivo principal en el presente trabajo es el desarrollo e implementación de un efecto digital, en concreto crear el efecto de sordina para Trombón tenor.

Esta idea surge de una situación laboral real. Tras grabar a una banda de música era la hora de la edición, mezcla y masterización para su uso comercial. Pero en el momento de empezar la edición y tras escuchar varios fragmentos el director se dio cuenta de que en un pasaje los trombones tenían que tocar con sordinas y no lo hicieron. Por suerte cada familia de instrumentos estaba grabada por pistas diferentes así que se volvió a grabar a los trombones y así se solucionó el problema. Esta situación me dio que pensar, pensaba en el tiempo y dinero que se había perdido tras este fallo y lo bien que vendría algún efecto o programa que simulara a los trombones sonando con sordina.

Tras explicar brevemente esta situación o idea principal y objetivo primordial, hay varios objetivos a tener en cuenta:

-Uno de ellos es la explicación de fundamentos físicos de la propagación de las ondas en instrumentos musicales y la física de estos.

-A continuación explicaré los fundamentos del audio digital y analógico, las ventajas de cada uno de ellos, los filtros digitales y los diferentes tipos de filtro y ventanas de cada uno de ellos.

-Por último haré hincapié en el entorno de trabajo del chameleon y el desarrollo y procesado de algoritmos para crear los efectos digitales deseados, en mi caso el desarrollo de filtros digitales.

2. FUNDAMENTOS MUSICALES

2.1- INTRODUCCIÓN

Desde un punto de vista físico, el sonido es una vibración que se propaga en un medio elástico (sólido, líquido o gaseoso) , generalmente el aire. Otra definición para el sonido podría ser la sensación producida en el oído por la vibración de las partículas que se desplazan (en forma de onda sonora) a través de un medio elástico que las propaga.

Para que se produzca un sonido se requiere la existencia de un cuerpo vibrante llamado "foco" (una cuerda tensa, una varilla, una lengüeta...) y del medio elástico transmisor de esas vibraciones, las cuales se propagan constituyendo la onda sonora.

Como onda el sonido responde a las siguientes características:

- ***Onda mecánica:***

Las ondas mecánicas no pueden desplazarse en el vacío, necesitan hacerlo a través de un medio material (aire, agua, cuerpo sólido) elástico.
La propagación de la perturbación se produce por la compresión y expansión del medio por el que se propagan. La elasticidad del medio permite que cada partícula transmita la perturbación a la partícula adyacente, dando origen a un movimiento en cadena.

- ***Onda longitudinal:***

El movimiento de las partículas que transporta la onda se desplaza en la misma dirección de propagación de la onda.

- ***Onda esférica:***

Las ondas sonoras son ondas tridimensionales, es decir, se desplazan en tres direcciones y sus frentes de ondas son esferas radiales que salen de la fuente de perturbación en todas las direcciones. El principio de Huygens afirma

que cada uno de los puntos de un frente de ondas esféricas puede ser considerado como un nuevo foco emisor de ondas secundarias también esféricas, que como la originaria, avanzarán en el sentido de la perturbación con la misma velocidad y frecuencia que la onda primaria.

Cualquier sonido sencillo, como una nota musical, puede describirse en su totalidad especificando tres características de su percepción: el tono, la intensidad y el timbre. Estas características corresponden exactamente a tres características físicas: la frecuencia, la amplitud y la composición armónica o forma de onda.

Tono

Los sonidos musicales son producidos por algunos procesos físicos como por ejemplo, una cuerda vibrando, el aire en el interior de un instrumento de viento, etc. La característica más fundamental de esos sonidos es su "elevación" o "altura", o cantidad de veces que vibra por segundo, es decir, su frecuencia.

Intensidad

La distancia a la que se puede oír un sonido depende de su intensidad, que es el flujo medio de energía por unidad de área perpendicular a la dirección de propagación. En el caso de ondas esféricas que se propagan desde una fuente puntual, la intensidad es inversamente proporcional al cuadrado de la distancia, suponiendo que no se produzca ninguna pérdida de energía debido a la viscosidad, la conducción térmica u otros efectos de absorción. Por ejemplo, en un medio perfectamente homogéneo, un sonido será nueve veces más intenso a una distancia de 100 metros que a una distancia de 300 metros. En la propagación real del sonido en la atmósfera, los cambios de propiedades físicas del aire como la temperatura, presión o humedad producen la amortiguación y dispersión de las ondas sonoras.

Timbre

Si el tono permite diferenciar unos sonidos de otros por su frecuencia, y la intensidad los sonidos fuertes de los débiles, el timbre completa las posibilidades de variedades del arte musical desde el punto de vista acústico, porque es la cualidad que permite distinguir los sonidos producidos por los diferentes instrumentos. Más concretamente, el timbre o forma de onda es la característica que nos permitirá distinguir una nota de la misma frecuencia e intensidad producida por instrumentos diferentes. La forma de onda viene determinada por los armónicos, que son una serie de vibraciones subsidiarias que acompañan a una vibración primaria o fundamental del movimiento ondulatorio.

Normalmente, al hacer vibrar un cuerpo, no obtenemos un sonido puro, sino un sonido compuesto de sonidos de diferentes frecuencias. A estos se les llama armónicos. La frecuencia de los armónicos, siempre es un múltiplo de la frecuencia más baja llamada frecuencia fundamental o “primer armónico”. A medida que las frecuencias son más altas, los segmentos en vibración son más cortos y los tonos musicales están más próximos los unos de los otros.

2.2- INSTRUMENTOS DE VIENTO METAL (TROMBÓN)

Este tipo de instrumentos consta de uno o varios tubos sonoros, los cuales contienen una columna gaseosa capaz de producir el sonido al ser convenientemente excitada. Las vibraciones del gas contenido en un tubo sonoro son longitudinales, y de igual manera que en las vibraciones transversales de las cuerdas, se siguen formando ondas estacionarias con zonas de vibración nula (nodos) y zonas de vibración máxima (vientres).

La Acústica musical clasifica en dos grupos a este tipo de instrumentos de tubos sonoros.

Tubos Abiertos: Son aquellos que disponen de dos o más orificios.

Debido al fenómeno de la reflexión se produce una onda estacionaria en el interior del tubo. Esta onda estacionaria proporciona dos Vientres en los extremos, con lo cual el sonido fundamental se produce cuando en el centro se forme un nodo.

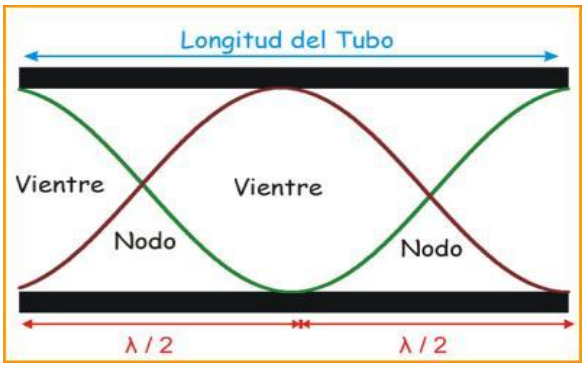


Figura 2.1. Tubo abierto

Tubos Cerrados: Son aquellos que disponen de un solo orificio.

En los Tubos Cerrados se produce un nodo en el extremo cerrado y un vientre en el extremo abierto. El sonido fundamental tiene lugar con un solo nodo y un solo vientre; el nodo para completar la onda estacionaria se forma fuera del tubo.

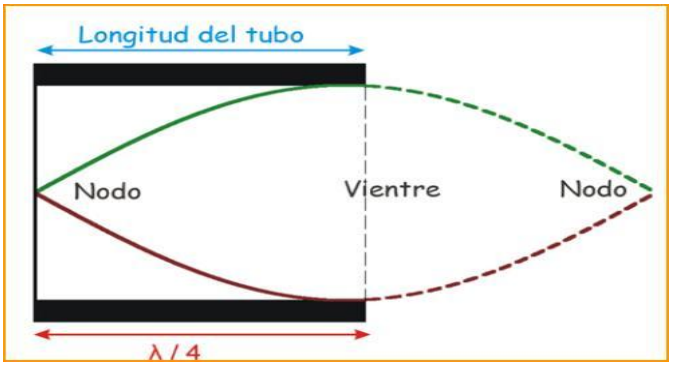


Figura 2.2. Tubo cerrado

La generalidad de instrumentos de viento convencionales está formada por tubos abiertos, quedando los cerrados para casos muy concretos como son ciertos tubos de órgano, el Clarinete, la Flauta de Pan, etc.

La excitación de la columna gaseosa en estos instrumentos se hace por medio de una embocadura, cuya misión es comunicar el movimiento vibratorio a la referida columna. La abertura donde se encuentra la embocadura no puede ser un nodo, pero tampoco debe ser necesariamente un vientre, pudiendo estar el punto de excitación en un lugar intermedio. De la misma forma no es necesario que las aberturas del tubo coincidan con los extremos. Las aberturas situadas a lo largo del tubo tienen por objeto el dividir la columna gaseosa en segmentos, produciendo cada una de ellas una frecuencia propia.

En los extremos abiertos la reflexión que se produce está en función de la anchura del tubo y de la abertura, comparada con la longitud de onda que se propaga por el tubo. En el caso de los instrumentos musicales el tubo es demasiado estrecho y no se puede disipar toda la energía en el extremo abierto, por lo que se produce el fenómeno de la reflexión. La reflexión hace que se produzca un vientre en dicho extremo abierto. Dicho de otra manera: "En todo extremo abierto de un tubo sonoro se produce un vientre". Esto último junto con el fenómeno de la difracción tiene una gran importancia para comprender como se generan los armónicos.

En el caso del trombón tenor perteneciente a la familia de viento-metal, el sonido se produce gracias a la vibración de los labios del intérprete en la boquilla a partir de una columna de aire (flujo del aire). Las diferentes notas se obtienen por el movimiento de un tuno móvil denominado vara, alargando la distancia que el aire en vibración debe recorrer, produciendo de este modo sonidos que también se pueden controlar con una mayor o menor presión del aire soplado por el intérprete. A mayor distancia de la vara, más se alarga la columna de aire y el sonido producido es más grave dependiendo de la presión ejercida siempre.

Figura 2.3. Trombón de vara tenor

2.3- LA SORDINA

La sordina es el nombre que reciben los diversos mecanismos de reducción del volumen o modificación de las cualidades tímbricas del sonido, que puede adoptar diferentes nombres y formas y puede hacerse de diferentes materiales, dependiendo del instrumento. Las más conocidas son las de los instrumentos de viento metal.

La sordina es un mecanismo que sirve para cambiar la calidad y atenuar el sonido producido por el instrumento, en nuestro caso el trombón tenor. Tiene diferentes formas y materiales, como el caucho, el plástico, la madera o el metal, y encaja perfectamente en la campana del instrumento, aunque existen

otros tipos de sordinas que no se acoplan (como la sordina desatascador).

Las sordinas obstruyen los movimientos de las ondas sonoras en los metales, amplifican ciertos armónicos y reducen otros. Los músicos han experimentado con diversas formas de sordinas, desde insertar una mano en el pabellón (técnica ortodoxa en algunos metales clásicos) o incluso cubrir la campana con un bombín.

Las sordinas más comunes son:

- La sordina straight (*straight mute*): produce los sonidos más suaves del instrumento. Cilíndrica, se ajusta al pabellón. Se toca hacia el extremo abierto de la sordina y las ondas sonoras pasan a un material que llena el cilindro y absorbe el sonido. (El efecto programado en Chameleon se basa en esta sordina).

Figura 2.4. Sordina straight

- La sordina cup (*cup mute*): es cónica, con forma de copa en el extremo más ancho, a veces recubierto de fieltro. Reduce el volumen y el tono incisivo del instrumento, pero añade una cualidad más suave. Puede ajustarse la posición de la copa en la parte cónica para alterar la distancia del pabellón.

Figura 2.5. Sordina cup

- La sordina Wah-Wah: inventada en 1965, es un tubo de metal que se ajusta al pabellón y va sellado con un cuello de metal. En esta técnica, se coloca la sordina sobre el pabellón y se varía el grado de cierre para producir un efecto wah-wah. Su timbre etéreo varía, dependiendo de cuánto se cierre la sordina

Figura 2.6. Sordina Wah-Wah

- La sordina Harmon: equivale a la sordina Wah-Wah, pero son el cuello metálico en su parte final, haciendo que los tonos agudos sean más estridentes cuando se toca con el orificio abierto

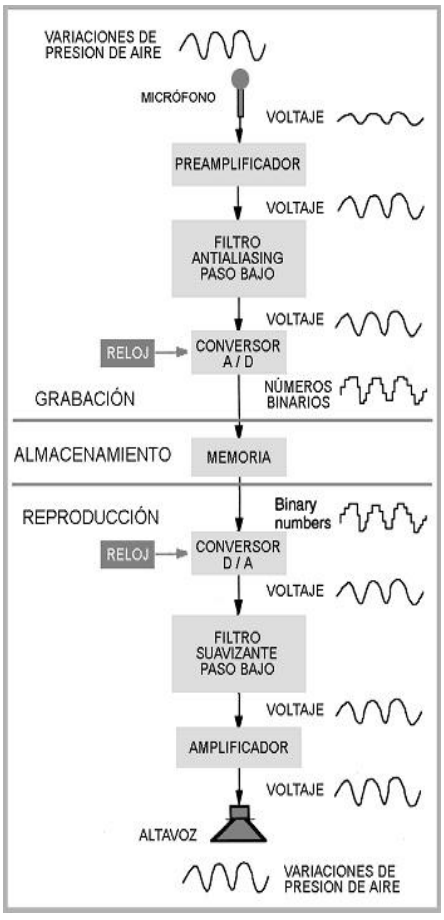
Figura 2.7. Sordina Harmon

3. FUNDAMENTOS DEL AUDIO DIGITAL

3.1- INTRODUCCIÓN

El audio digital es la representación de señales sonoras mediante un conjunto de datos binarios, la codificación de la señal sonora en términos discretos. Un sistema completo de audio digital comienza habitualmente con un transductor mecánico-eléctrico (micrófono) que convierte la onda de presión que excita al transductor en una señal eléctrica.

Por los años 70 Alec Reevers desarrolló un sistema de codificación de señales. El PCM (Pulse Code Modulation) o modulación por impulsos codificados es un procedimiento de modulación utilizado para transformar una señal analógica en una secuencia de bits. Una trama o stream PCM es una representación digital de una señal analógica en donde la magnitud de la onda analógica es tomada en intervalos uniformes, muestras que puede tomar un conjunto finito de valores, los cuales se encuentran codificados.



La señal analógica tras ser procesada se muestrea (se toma un número discretos de valores de la señal analógica), se cuantifica (se asignan valores analógicos discretos a las muestras) y codifica (se asigna una secuencia de bits a cada valor analógico discreto) como veremos más adelante. Un sistema de audio digital suele terminar con el proceso inverso. De la representación digital almacenada se obtiene el conjunto de muestras que representan. Estas muestras pasan por un proceso de conversión digital-analógico

Figura 3.1. Sistema y procesado completo de audio digital.

proporcionando una señal eléctrica/analógica que tras un procesado (amplificación, ecualización, etc.)excita un transductor eléctrico-mecánico, un altavoz.

3.2- CONVERSIÓN ANALÓGICO-DIGITAL

Consiste en la transcripción de señales analógicas en señales digitales, con el propósito de facilitar su procesamiento y hacer la señal resultante más inmune al ruido y otras interferencias a las que son más sensibles las señales analógicas.

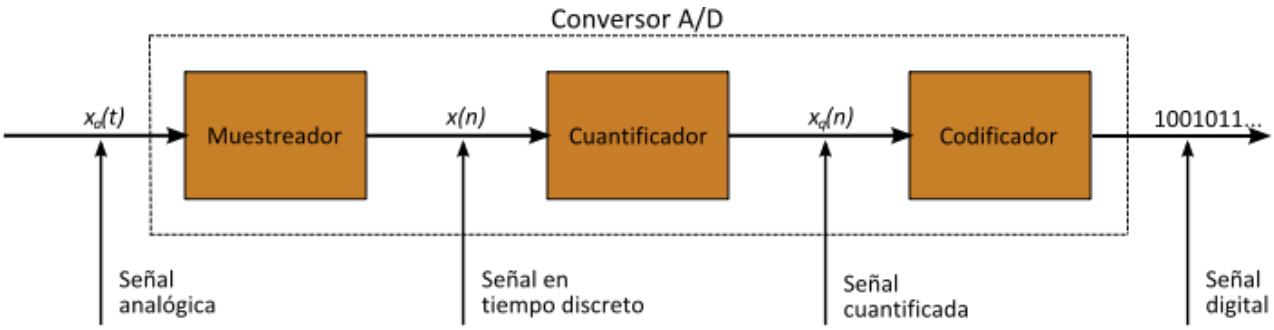


Figura 3.2. Conversor Analógico-Digital

3.3- MUESTREO, FRECUENCIA DE MUESTREO

La frecuencia de muestreo es el número de muestras por unida de tiempo que se toma de una señal continua convirtiéndose así en una señal discreta.

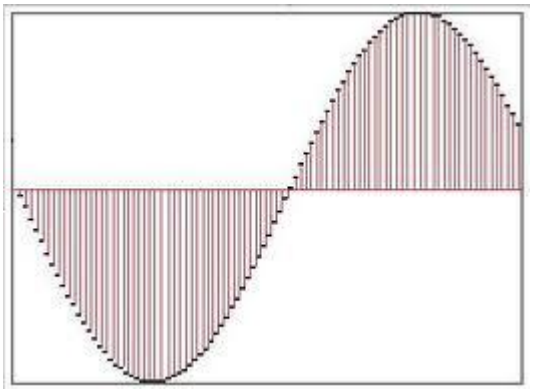


Figura 3.3. Muestreo

Está basado en el teorema de muestreo de Nyquist, el cual expone que para poder replicar con exactitud la forma de onda continua es necesario que la frecuencia de muestreo sea superior al doble de la máxima frecuencia a muestrear. Es necesario que se deje un margen entre la frecuencia máxima que se desea registrar y la frecuencia de Nyquist que resulta de la tasa de muestreo elegida, en concreto para audio la frecuencia máxima de los componentes a registrar y reproducir es de 20 KHz y la frecuencia crítica de la tasa de 44100 muestras por segundo empleada es de 22,05 KHz, un margen necesario del 10% aproximadamente que resulta de las limitaciones físicas del filtro de reconstrucción o filtro antialiasing (Filtro con el propósito de eliminar cualquier “presencia”, antes de hacer el muestreo, de las frecuencias superiores de Fs/2 siendo Fs la frecuencia de muestreo).

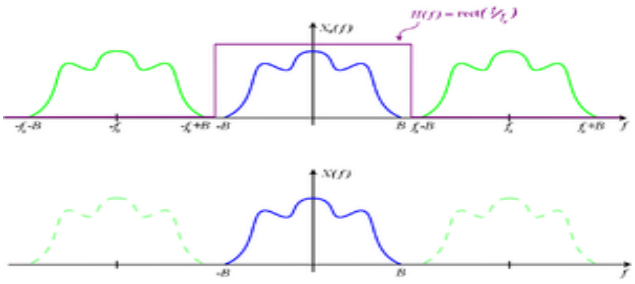
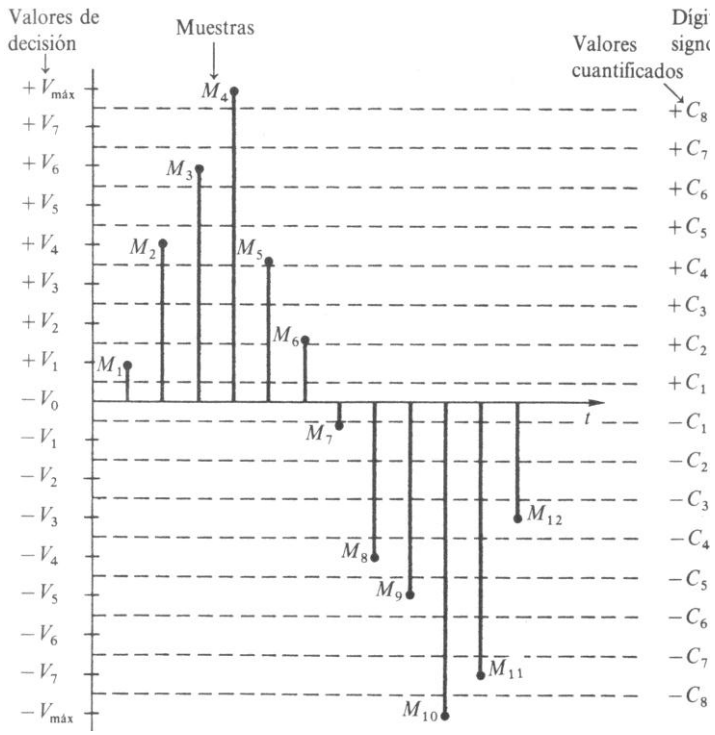


Figura 3.4. Filtro de reconstrucción

3.4- CUANTIFICACIÓN

Una vez muestreada la señal, la sucesión de muestras de amplitud se convierte en valores discretos.



Durante el proceso de cuantificación se mide el nivel de tensión de cada una de las muestras y se les atribuye un valor finito de amplitud seleccionado por aproximación dentro de un margen de niveles previamente fijados. Estos valores se eligen en función de la propia resolución que utilice el código empleado durante la codificación.

Figura 3.5. Cuantificación

Ahora la señal analógica se convierte en digital ya que los valores prestablecidos son finitos, aunque todavía no se traduce al sistema binario. La señal ha quedado representada por valores finitos que durante la codificación será cuando se transforme en una sucesión de ceros y unos tal como explicaré a continuación.

3.5- CODIFICACIÓN

Consiste en la traducción de valores de tensión eléctricas analógicos que ya han sido cuantificados al sistema binario mediante códigos

prestablecidos. La señal analógica queda en un tren de impulsos, el resultado es un sistema binario que está basado en el álgebra de Boole.

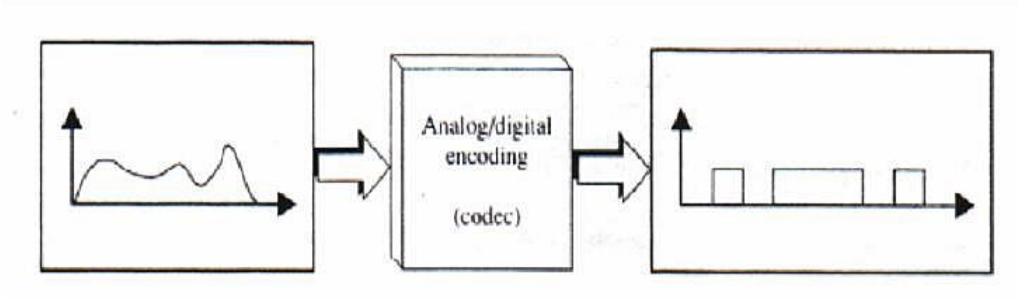


Figura 3.6. Codificación de una señal analógica.

Los métodos de codificación de audio que existen en la actualidad se basan en algoritmos de compresión y en codificación multicanal. Los algoritmos de compresión de audio se fundamentan en aspectos perceptibles al oído humano. Básicamente son dos los fenómenos que han originado los métodos de compresión:

- La curva de sensibilidad del oído
- El fenómeno de enmascaramiento

El oído humano detecta sonidos entre 20 Hz y 20 KHz, pero su sensibilidad depende de la frecuencia el sonido de esta forma, dos frecuencias con la misma potencia son interpretadas por nuestro oído de forma diferente, teniendo la sensación de que una es más fuerte que otra, o incluso oír una y no la otra. La curva que indica cual es la potencia mínima (umbral) que nuestro oído detecta es la curva de sensibilidad.

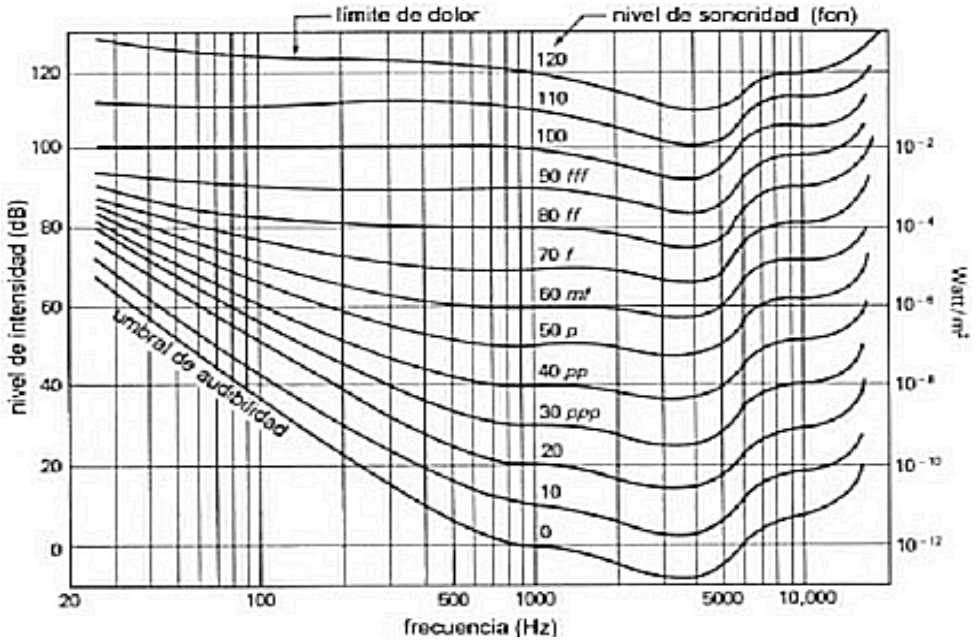


Figura 3.7. Curva de sensibilidad del oído humano.

El enmascaramiento gana importancia cuando los sonidos son cercanos en frecuencia y la frecuencia que enmascara es inferior que la enmascarada. Para poder cuantificar el fenómeno de enmascaramiento surge el concepto de banda crítica como el ancho de banda máxima alrededor de una frecuencia para que no haya enmascaramiento, por lo tanto, sólo se produce éste entre bandas contiguas.

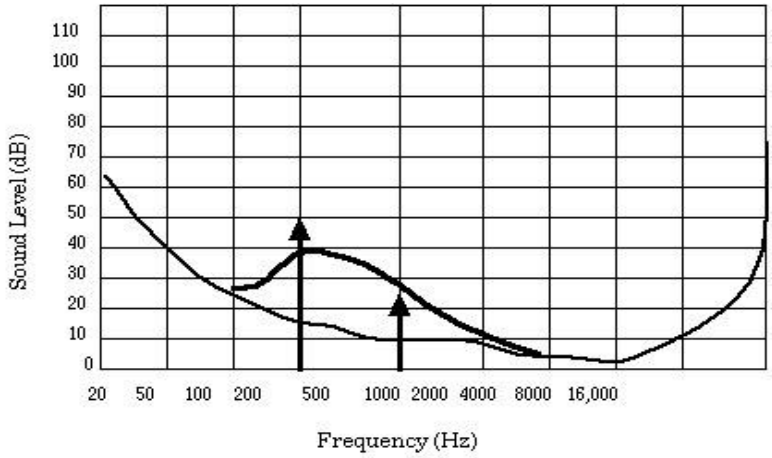


Figura 3.8. Escala psicoacústica de Bark

Además estas bandas están distribuidas siguiendo una escala logarítmica que simula la respuesta perceptiva del oído humano. Una escala de

medida perceptual es la escala Bark (Figura 3.8.) que relaciona las frecuencias acústicas con la resolución perceptual de éstas. A partir de esta escala de bandas de frecuencia y de un modelo psicoacústico se determinará que frecuencias se enmascaran y cuales no. También existe enmascaramiento temporal, cuando oímos un sonido de alta potencia y para de pronto seguimos oyéndolo durante un breve instante de tiempo que puede enmascarar a otras señales.

Un ejemplo de codificación es el MPEG, el cual define tres capas de codificación de audio, cada una añade complejidad a la anterior. La codificación se realiza dividiendo las secuencias de audio en tramas (de 384 muestras), que se filtra para obtener las bandas críticas

- Capa 1: sólo considera enmascaramiento frecuencial.
- Capa 2: considera además el enmascaramiento temporal estudiando tres tramas a la vez.
- Capa 3: utiliza filtros no lineales, elimina redundancias provocadas por muestreo y utiliza codificación de Huffman (algoritmo usado para compresión de datos).

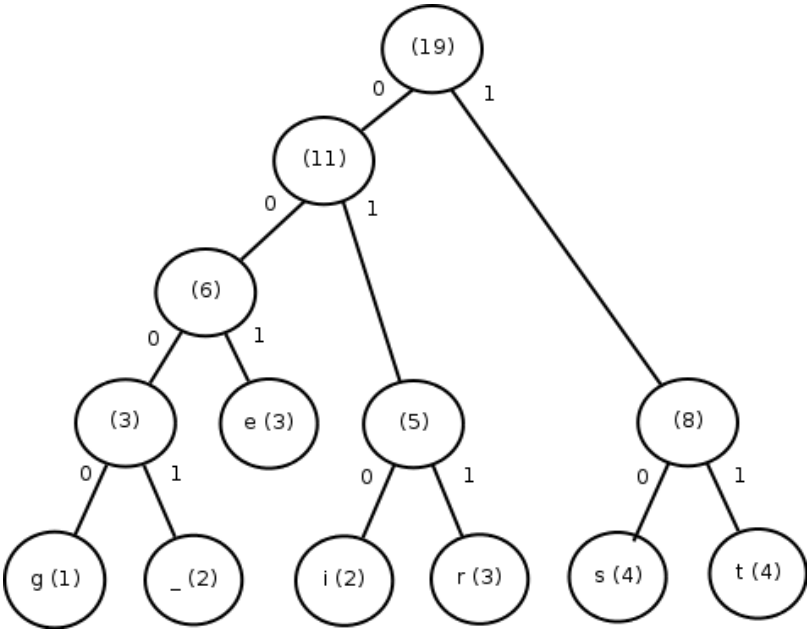


Figura 3.9. Árbol binario del algoritmo de Huffman.

Una vez tenemos la señal digital podemos procesarla, editarla, hacer con ella infinidad de cosas pero toca devolverla a su estado original, toca convertir la señal digital en analógica. Para ello se requiere de un conversor digital-analógico (DAC; Digital-to-Analog Conversion) que convierte los números abstractos obtenidos de la conversión analógico-digital en una secuencia concreta de impulsos que luego son procesadas por un filtro de reconstrucción utilizando algún tipo de interpolación para llenar los datos entre los impulsos. Otros métodos como la modulación delta-sigma producen un pulso de señal que puede ser filtrada de una forma similar para producir una señal suavemente variable.

En lugar de los impulsos, por lo general la secuencia de números actualiza el voltaje analógico en uniformes intervalos de muestreo. Estos números se escriben en el DAC, típicamente con una señal de reloj que hace que cada número sea enganchado en secuencia, en cuyo momento la tensión de salida del DAC cambia rápidamente desde el valor anterior al valor representado por el número actual. El efecto de esto es que el voltaje de salida se mantiene con el valor actual en el tiempo hasta que el número de entrada siguiente varía, resultando una salida constante en forma de escalera.

El hecho de que la salida DAC sea una secuencia de valores constantes a trozos o pulsos rectangulares producen armónicos múltiples por encima de la frecuencia de Nyquist. Por lo general, éstos se eliminan con un filtro paso bajo que actúa como un filtro de reconstrucción en las aplicaciones que lo requieran, en nuestro caso para audio.

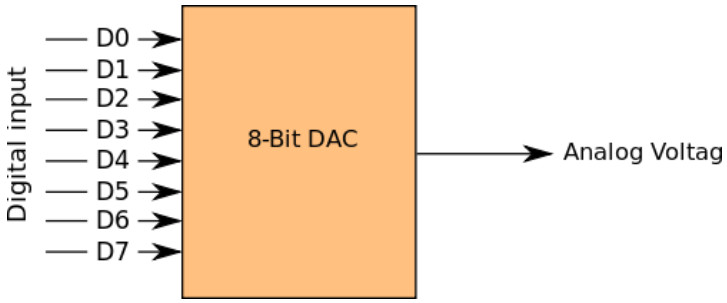


Figura 3.10. Diagrama funcional simplificado de un DAC de 8 bits.

4. FILTROS DIGITALES

4.1- INTRODUCCIÓN

Un filtro es un sistema, que dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Los filtros digitales tienen como entrada una señal digital y a su salida se obtiene otra señal digital, pudiendo haber cambiado su amplitud, frecuencia o fase dependiendo de las características del filtro.

El filtrado digital es parte del procesado de señal digital. Se le denomina digital más por su funcionamiento interno que por su dependencia del tipo de señal a filtrar, así podríamos llamar filtro digital tanto a un filtro que realiza el procesado de señales digitales como a otro que lo haga a señales analógicas.

El filtrado digital consiste en la realización interna de un procesado de datos de entrada. El valor de la muestra de la entrada actual y algunas muestras anteriores (que previamente habían sido almacenadas) son multiplicadas, por unos coeficientes definidos. También podría haber tomado valores de la salida en instantes pasados y multiplicarlos por los coeficientes. Finalmente todos los resultados se todas estas multiplicaciones son sumados, dando una salida para el instante actual. Esto implica que internamente tanto la salida como la entrada del filtrado serán digitales, por lo que puede ser necesario una conversión analógico-digital o digital-analógico para uso de filtros digitales en señales analógicas.

Los filtros digitales se usan frecuentemente para tratamiento digital de la imagen o como en nuestro caso, para el tratamiento del sonido digital.

Hay varios tipos de filtro así como distintas clasificaciones de estos:

-De acuerdo con la parte del espectro que dejan pasar y que actúan tenemos:

- Filtros pasa alto
- Filtros pasa bajo
- Filtros pasa banda:
    - Banda eliminada
    - Multibanda
    - Pasa todo
    - Resonador
    - Oscilador
    - Filtro peine
    - Filtro ranura o filtro rechaza banda (filtro Notch), etc.

-De acuerdo con su orden:

- Primer orden
- Segundo orden, etc.

-De acuerdo con el tipo de respuesta ante entrada unitaria:

- FIR (Finite Impulse Response)
- IIR (Infinite Ipulse Response)
- TIIR (Truncated Infinite Impulse Response)

-De acuerdo con la estructura que se implementa:

- Directa
- Transpuesta
- Cascada
- Fase Lineal
- Laticce

Existen dos métodos básicos para el análisis del comportamiento o respuesta de un sistema lineal a una determinada entrada. Un primer camino se basa en obtener la solución de la ecuación entrada-salida del sistema que en general tiene la forma de las ecuaciones en diferencias lineales a coeficientes constantes a_m b_k.

$$\sum_{m=0}^{N_a-1} a_m y[n-m] = \sum_{k=0}^{N_b-1} b_k x[n-k]$$

(4.1)

Siendo *Na y Nb* los ordenes máximos de las diferencias en la ecuación correspondientes a la salida y entrada del sistema.

El segundo método para el análisis del comportamiento del sistema reside en la aplicación del principio de superposición y consiste en descomponer la señal de entrada en una suma de señales elementales las cuales se escogen de manera que sea conocida la respuesta del sistema a las mismas. Siguiendo

esta línea, una señal a tiempo discreto puede visualizarse como una secuencia pesada de impulsos unitarios.

$$x[n] = \sum_{k=-\infty}^{\infty} x[k].\delta[n-k]$$

(4.2)

Aplicando la propiedad de superposición de los SLTI (Sistemas Lineales e Invariantes en el Tiempo) (Oppenheim y Willsky, 1983), se puede determinar la salida del sistema ante una cierta entrada de la siguiente manera:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k]$$

(4.3)

Siendo h[n] la respuesta o salida del sistema ante una entrada equivalente a un impulso unitario d[n], denominada respuesta al impulso del sistema. El segundo miembro de la expresión representa el producto de convolución de la señal x[n] y la respuesta al impulso del sistema h[n], esto es:

$$y[n] = x[n] * h[n] = h[n] * x[n]$$

(4.4)

4.2- FILTROS FIR

Los filtros digitales de Respuesta Finita Impulsiva por sus siglas en ingles “Finite Impulse Response”, se trata de un tipo de filtro digitales en el que, como

su nombre indica, si la entrada es una señal impulso la salida tendrá un número finito de términos no nulos. La estructura de señal a la salida del filtro se basa solamente en la combinación lineal de las entradas actuales y anteriores, esto es:

$$y[n] = \sum_{k=0}^{N_b-1} b_k x[n-k] - \sum_{m=1}^{N_a-1} a_m y[n-m]$$

(4.5)

En donde *N* es el orden del filtro, que también coincide con el número de términos no nulos y con el número de coeficientes b_k del filtro.

En la figura 4.1, se muestra el diagrama en bloques de la estructura básica del filtro FIR, para una cantidad de 12 coeficientes.

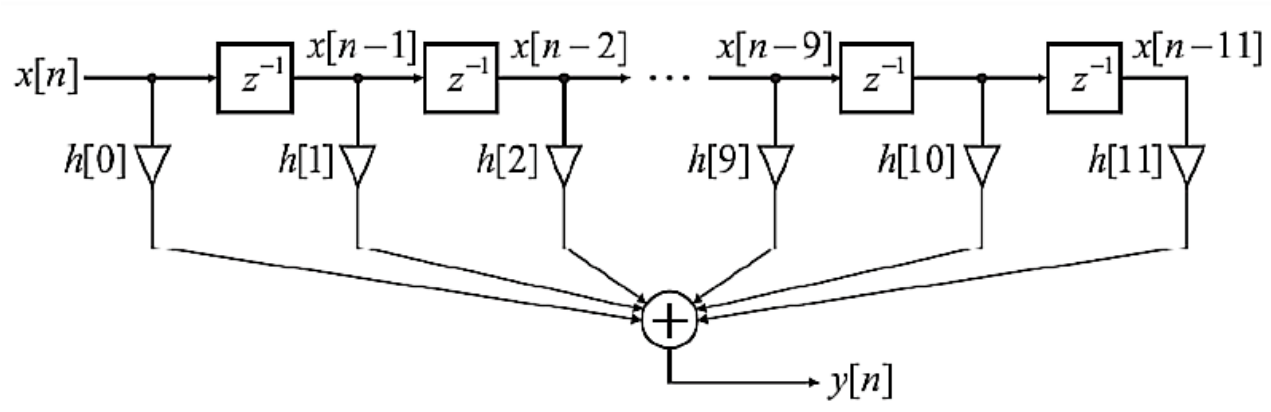


Figura 4.1: Representación en diagrama de bloques del filtro FIR, para un total de 12 coeficientes

4.3- FILTROS IIR

Son sistemas cuya salida depende además de salidas anteriores y que, estando en reposo, al ser estimulados con una entrada impulsional su salida no vuelve al reposo, de ahí el calificativo de filtros de respuesta impulsional infinita (IIR). La ecuación en diferencias general es de la forma:

$$y_n = b_0 x_n + b_1 x_{n-1} + \ldots + b_n x_{n-N} - a_1 y_{n-1} - a_2 y_{n-2} - \ldots - a_m y_{n-M}$$

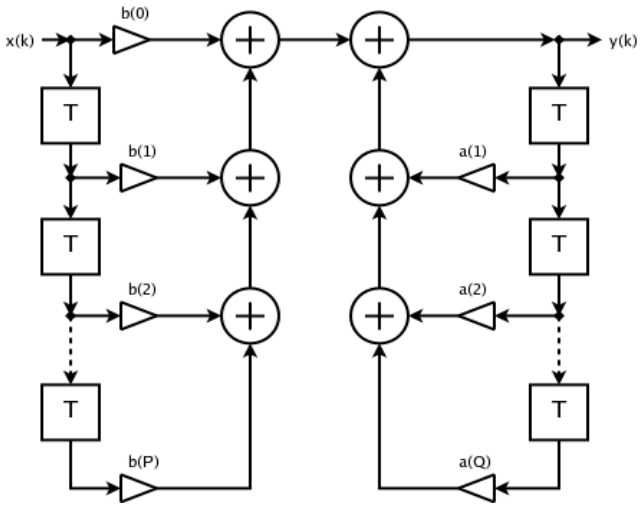
(4.6)

Donde a y b son los coeficientes del filtro y el orden es igual al máximo de M y N.

La función de transferencia en Z del filtro es:

$$H(z) = \frac{\sum_{k=0}^{M} b_k \cdot z^{-k}}{\sum_{k=0}^{N} a_k \cdot z^{-k}}$$

(4.7)



Hay numerosas formas de implementar los filtros IIR. La estructura afecta a las características finales que presentará el filtro como la estabilidad. Otros parámetros a tener en cuenta a la hora de elegir una estructura es el gasto computacional que presenta

Figura 4.2. Representación en diagrama de bloques del filtro IIR

Existen dos filosofías de diseño de filtros IIR:

INDIRECTA: Se basa en aplicar a filtros analógicos diseñados previamente, transformaciones que los conviertan en digitales con las mismas características. Hay tres métodos fundamentales:

- Diseño por impulso invariante.
- Diseño por analogía o aproximación de derivadas.
- Diseño por transformación bilineal

DIRECTA: Se propone el diseño de filtros digitales imponiendo una serie de condiciones a la respuesta para determinar los coeficientes. Nos centramos en dos métodos simples como son:

- Diseño por la aproximación de Padé
- Diseño por aproximación de mínimos cuadrados.

También podemos considerar como método directo de uso limitado el diseño de ubicación de ceros y polos.
Los filtros más generales contienen ceros y polos. Si los coeficientes del filtro son reales, si los ceros o polos son complejos siempre aparecen como pares complejos conjugados. La condición de estabilidad, para sistemas causales implica que los Polos se encuentran en el interior de la circunferencia unidad. Los ceros no tienen efecto sobre la estabilidad del sistema y pueden encontrarse en el interior o exterior de dicha circunferencia.
Cuando los ceros y polos de un sistema se encuentran en el interior de la circunferencia unidad se dice que el sistema es de FASE MÍNIMA.
Cuando todos los ceros y polos están en el exterior de la circunferencia unidad se dice que el sistema es de FASE MÁXIMA.
En general, cuando tenemos ceros y polos en el exterior y en el interior se dice que el sistema es de FASE MIXTA.

Comparado con un FIR, un filtro IIR requiere un orden mucho menor para cumplir las especificaciones de diseño, sin embargo estos últimos no pueden

diseñarse para tener fase lineal. Existen técnicas de compensación de fase mediante la utilización de filtros pasa todo, sin embargo esto aumenta la longitud total del filtro. Son muy eficaces y pueden proporcionar pendientes de corte muy pronunciadas. Por otro lado, al poseer características de realimentación (o feedback), tienen tendencia a entrar en oscilación y en resonancia.

5. CHAMELEON

5.1- INTRODUCCIÓN

Para poder desarrollar el efecto de sordina para trombón he escogido el dispositivo Soundart Chameleon DSP.

Chameleon es un dispositivo versátil diseñado exclusivamente para aplicaciones de audio en tiempo real, procesamiento de señales digitales y MIDI. Concebido como una plataforma abierta (puede ser totalmente programado por el usuario), su funcionalidad específica dependerá del código que se ejecuta cada vez. Este código puede ser generado por el usuario o por medio de las herramientas de desarrollo apropiadas. Los conceptos fundamentales son el control y procesamiento de este sistema.

El control se lleva a cabo por medio del panel frontal programable para la interacción con el usuario y un microcontrolador, que lleva a cabo todas las tareas de intercomunicación del aparato. La parte de procesamiento se realiza por medio del procesador de señales digitales (DSP) que controla todo el procesamiento de audio y/o generación en tiempo real. Las conexiones de dispositivos externos son: dos entradas y dos salidas analógicas no balanceadas, una salida de auriculares estéreo, una entrada MIDI y una salida, además de un estándar RS-232 para fines de depuración.

Éste procesador de señales nos da la opción de crear y configurar cualquier efecto digital, desde un ecualizador hasta un delay para guitarra electrónica, bajo o cualquier instrumente imaginable, en mi caso un trombón tenor previamente grabado. Podemos configurar los potenciómetros de la parte frontal a partir de lenguaje c/c++, control de volumen, selección de efecto, opciones de menús, modificación de parámetros, etc. todo a nuestro gusto y parecer. El aparato viene en formato de rack estándar, se puede acoplar a cualquier armario o cajonera de racks, pudiendo convertirse en una

herramienta eficaz para cualquier estudio de grabación o directos.

5.2- ARQUITECTURA HARDWARE

El chameleon contiene internamente dos procesadores que trabajan de una manera totalmente asíncrona: un micro controlador Motorola ColdFire MCF5206e y un DSP Motorola 56303.

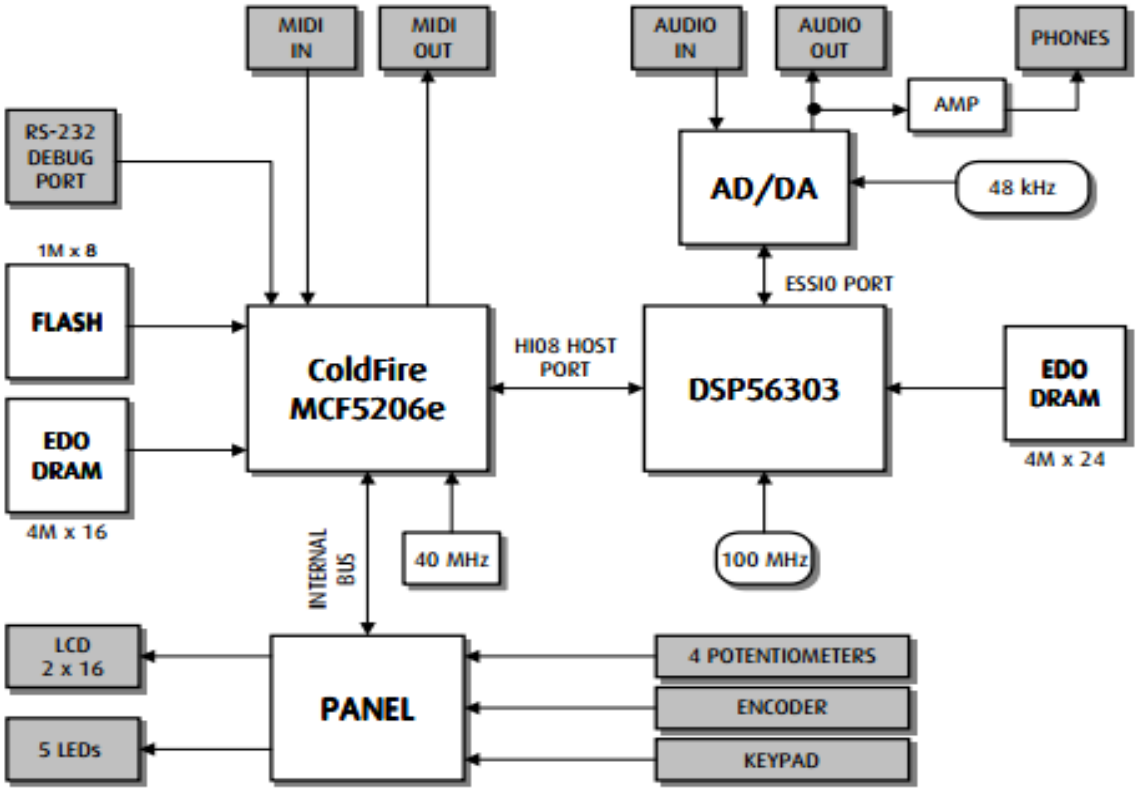


Figura 5.1. Arquitectura del hardware

El DSP se encarga exclusivamente de generar y procesar el audio. Tiene una conexión con el mundo analógico a través de un conversor AD/DA de 24 bits a una velocidad de muestreo de 48 KHz. Los datos de audio se reciben y transmite en forma de serie síncrona intercalada a través del puerto DSP ESSI0 por otra parte, se ha dotado de un tipo de subsistema de memoria externa ESO DRAM disponible para las operaciones de audio.

El ColdFire es responsable de la tramitación de todas las tareas de control del sistema. Este procesador utiliza una longitud de palabra de 32 bits. Posee 8Kbytes de memoria interna y un controlador de memoria caché para reducir el tiempo de espera durante los accesos externos

Ambos procesadores (el DSP56303 y ColdFire MCF5206e) están interconectados por el bus HIO8 HOST PORT.

5.3- PANEL FRONTAL Y POSTERIOR

El panel frontal consiste en la interfaz de comunicación con la aplicación del usuario. A través de este panel los comandos son recibidos junto al estado actual de la aplicación que se ejecuta en la pantalla.

Está compuesto de:

- Cuatro potenciómetros. Los tres de la derecha están concebidos para ser asignados como controladores en tiempo real. El de la izquierda marcado como volumen actúa como controlador del volumen de audio, aunque su funcionalidad puede programarse para intercambiarlos arbitrariamente.

- Un codificador incremental rotativo programable. Se puede utilizar como otro control de propósito general.

- Doce pulsadores. Se dividen en dos grupos y su funcionalidad es totalmente dependiente de la aplicación. El programador se encarga de asignar a cada uno de estos botones su función específica.

- Un display LCD de 2 líneas de 16 caracteres alfanuméricos.

- Cinco diodos LED. Tres de ellos colocados sobre cada uno de los potenciómetros controladores en tiempo real, y los dos restante colocados al lado de “edit” y “shift”, aunque su funcionalidad no está vinculada con estos controles y una vez más es tarea del programador asignar el encendido y apagado de éstos.

- Una salida de auriculares jack 1/4”. Es un puente con la salida en línea de audio del dispositivo.

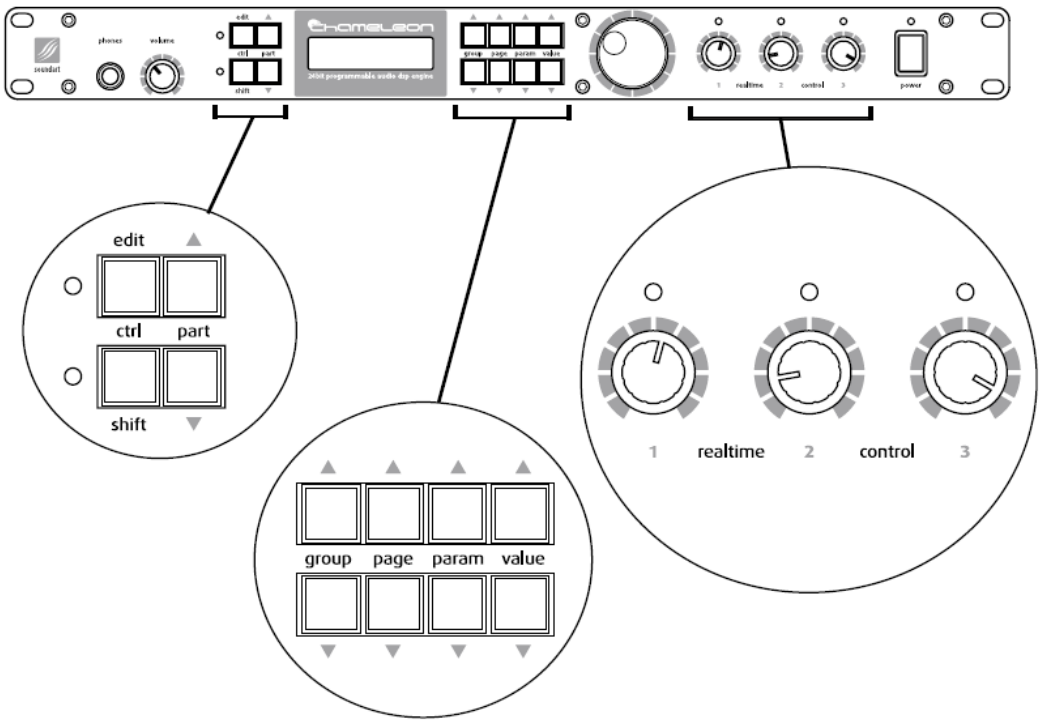


Figura 5.2. Panel frontal del Chameleon.

Desde el punto de vista del programador, el acceso a todos estos controles se realiza a través del ColdFire, por lo tanto se proporciona una biblioteca para la aplicación de todas las funciones necesarias para acceder a estos controles.

En la figura 5.3 se muestra el panel posterior del Chameleon, donde se sitúan todas las conexiones del equipo (excepto los auriculares que se colocan en el panel frontal.

Estas conexiones son:

- Potencia 9VDC/1.2A. Conexionado del suministro de corriente.

- Una entrada MIDI y una salida, con la normativa de conectores DIN 5

- Dos entradas analógicas no balanceadas, con conectores TRS jack 1/4".

- Dos salidas analógicas no balanceadas, con conectores TRS jack 1/4".

- Un conector DN-9 (RS-232 estándar) para la depuración de las funciones.

Figura 5.3. Panel posterior del Chameleon

5.4- *ENTORNO DE TRABAJO DEL CHAMELEON*

El Chameleon es un procesador digital de señales totalmente programable. Para ello el propio aparato viene con su software que se encargará de compilar y ensamblar los códigos que programemos junto a las librerías y macros de programación que nos facilitarán el trabajo a desarrollar.

Primero instalaremos el software que viene en formato CD. Una vez instalado el software, programaremos y desarrollaremos en el cuadro de Chameleon Development Environment la aplicación que deseemos. Una vez programado ejecutaremos la aplicación en el cuadro Chameleon Toolkit. Todo esto lo explicaré con mayor detenimiento a continuación. Además, se dispone de un programa en el cual podemos ver el desarrollo y ejecución del programa que se ha desarrollado.

5.4.1- Chameleon Development Environment.

Este programa es el entorno de trabajo en el cual programaremos y desarrollaremos la aplicación deseada.

Lo primero crearemos un nuevo proyecto y lo guardaremos con el nombre que queramos. Al crear el nuevo proyecto se generan dos archivos, uno **main.c** y otro **main.asm**. Estos dos son en los que vamos a trabajar y desarrollar nuestro código. En el main.c (ver figura 5.4) utilizaremos lenguaje C o C++ y en este cuadro es donde programaremos cada función destinada a los componentes del panel frontal y de cómo el Coldfire va a comunicar la DSP cualquier variación que se produzca en él (códigos de inicialización y finalización del DSP o cambio de valores de parámetros).

El main.asm (ver figura 5.5) es el archivo en el que se desarrolla el código ensamblador. En este se programan las funciones que ha de realizar el DSP con lo que le llega directamente del conversor A/D (ver figura 5.1), funciones como guardar las muestras que le llegan del conversor A/D en un buffer, de que forma se comunicará con el

Coldfire o escalar por un valor que haya recogido el Coldfire procedente de una variación del panel frontal como por ejemplo el volumen entre otros.

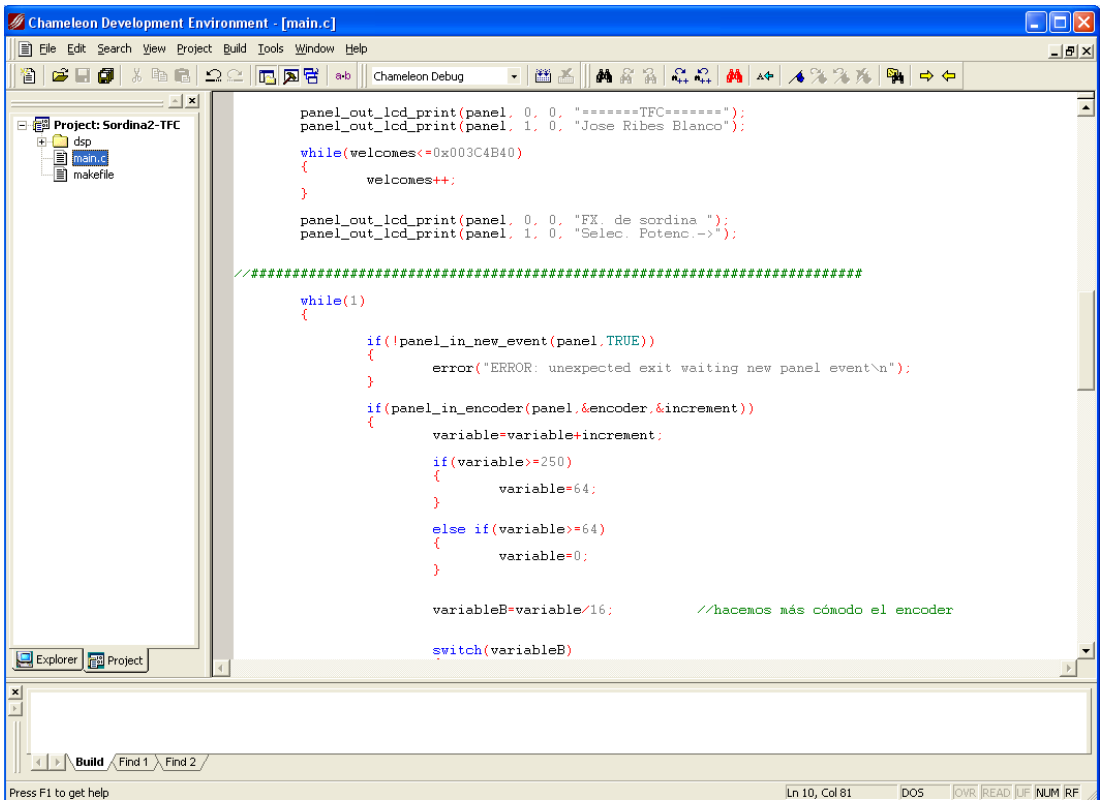


Figura 5.4 Chameleon Development Environment. Entorno de trabajo main.c

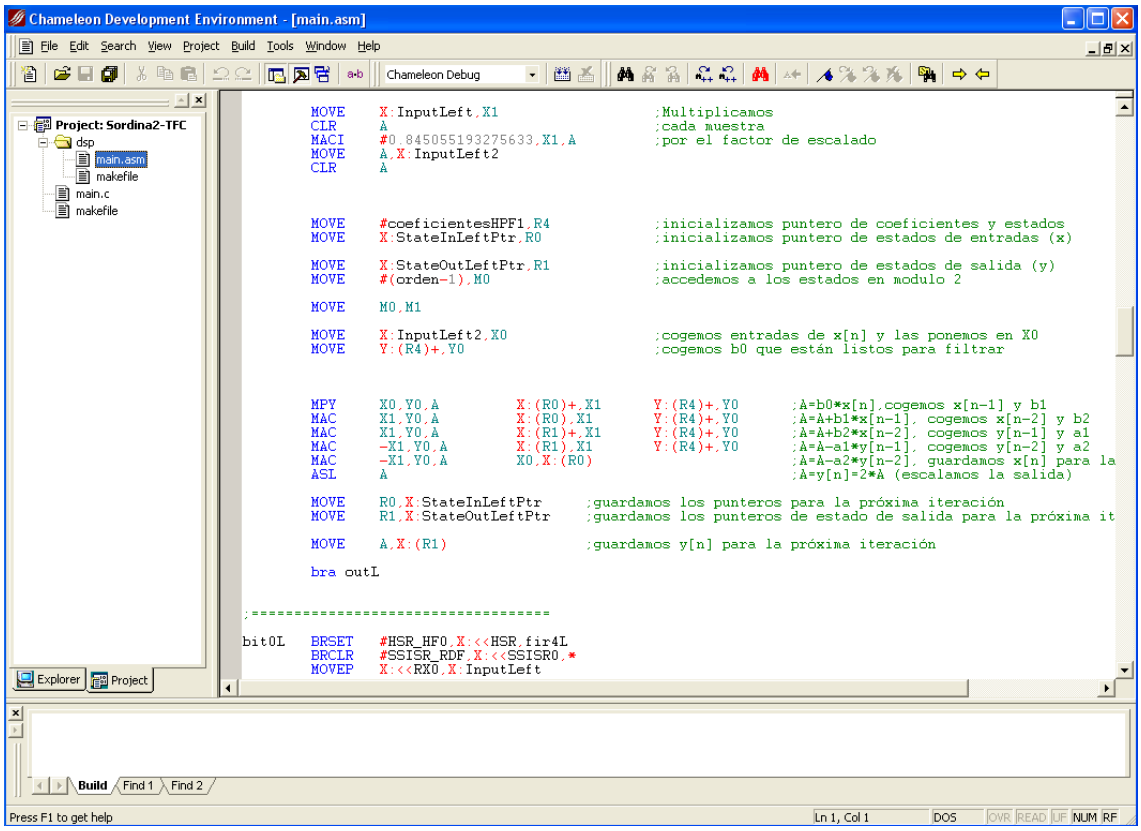


Figura 5.5 Chameleon Development Environment. Entorno de trabajo main.asm

5.4.2- Chameleon Toolkit.

Una vez programada la aplicación deseada en el Chameleon Development Environment lo compilamos y este proceso genera varios archivos, entre uno de estos hay un archivo .elf que deberemos cargar en el Chameleon Toolkit.

El Toolkit será el encargado de transmitir la aplicación que hemos desarrollado al Chameleon a través del puerto-serie RS-232 (ver figura 5.3). En el Toolkit podremos observar las variaciones que hagamos desde el panel frontal siempre y cuando las hayamos programado para que puedan ser vistas en el cuadro.

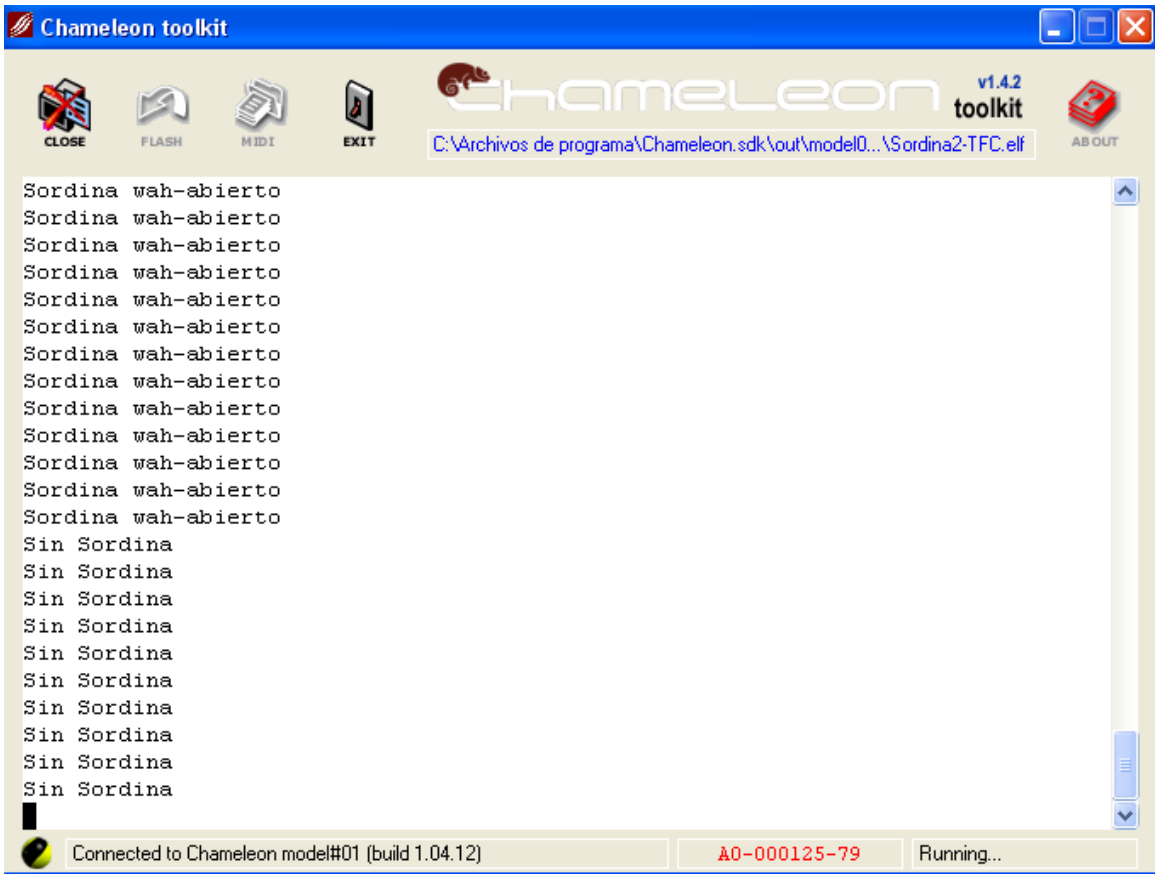


Figura 5.6 Chameleon Toolkit

5.4.3- Simulador DSP GUI56300

El GUI56300 sirve para simular la aplicación que hayamos desarrollado en el ensamblador. Se muestra todos los registros del DSP como memorias, registros, mapa de direcciones, etc. y se puede visualizar paso a paso como va el programa, carga de registros, comandos, etc.

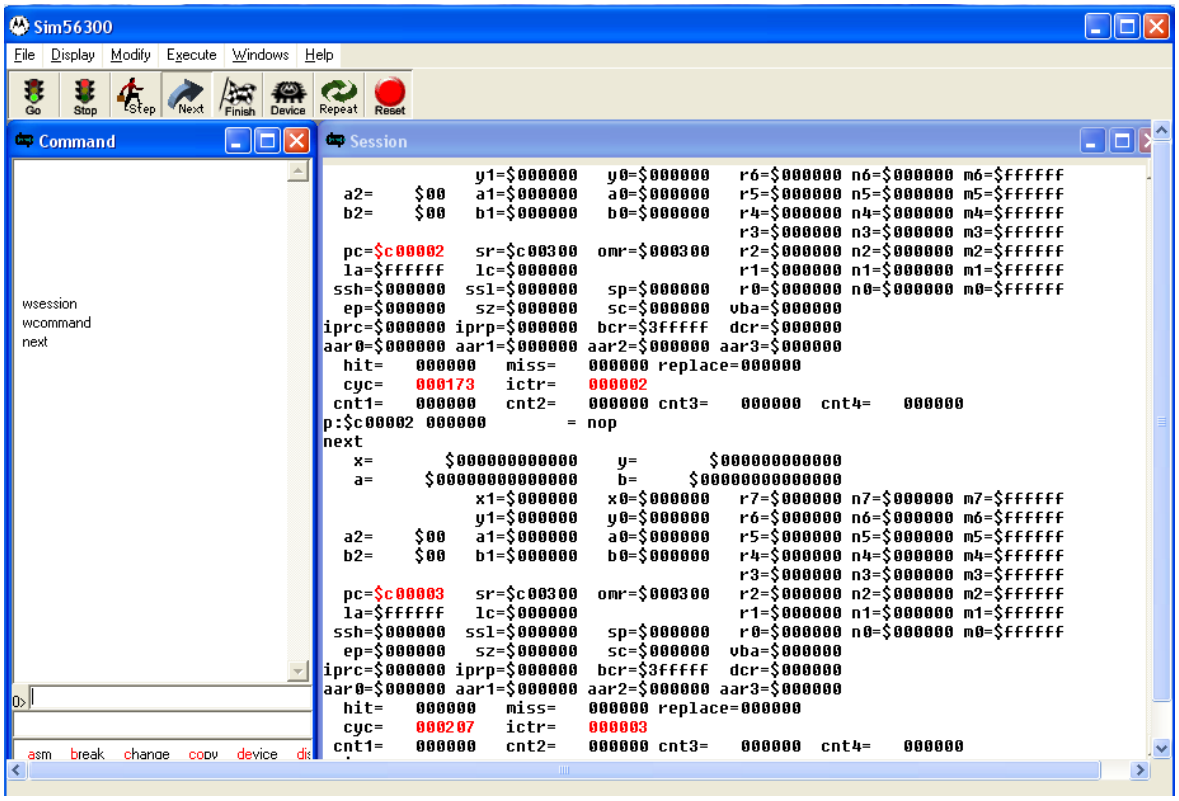


Figura 5.7 Simulador DSP GUI56300

6. Desarrollo del programa.

6.1- INTRODUCCIÓN

Lo primero era pensar en como quería que funcionara el aparato es decir, que potenciómetros utilizar y como. Tenía una idea clara, tenía que utilizar los potenciómetros necesarios para que pasara por diferentes estados, el primer estado tenía que sonar el trombón sin sordina, el siguiente tenía que sonar el trombón con sordina Straight, a continuación sordina Wah-Wah cerrado y finalmente Wah-Wah abierto. Esta era la idea básica que tenía que realizar el aparato.

Una vez clara la idea de lo que quería hacer, era el momento de desarrollar la aplicación que quería programar en el Chameleon. Lo siguiente fue hablar con alguien que tocara el trombón, para ello mi hermano me proporcionó gran ayuda y le pedí que tocara su instrumento sin la sordina y con sordina. Tras escuchar bastantes veces y diferentes notas llegué a la conclusión de que tenía que utilizar y desarrollar filtros para eliminar frecuencias, que es básicamente lo que hacía la sordina al acoplarla a la campana del instrumento. El problema era saber que frecuencias eran las que se enmudecían, bajas, altas o ambas.

6.2- ANÁLISIS ESPECTRAL.

Caí en la cuenta de que una vez realizamos una práctica en audio digital si no recuerdo mal, que trataba de hacer una análisis espectral en frecuencia de una señal de audio, visualizar el espectro de esta y ver las frecuencias fundamentales que se representaban.

Una vez ojeado como se trataba éste método era el momento de ponerlo en práctica. La forma más sencilla y rápida era con el software SoundForge, en concreto con el Spectrum analisys, uno de sus plug-ins predeterminados. Lo podemos encontrar en la barra superior del menú, nos vamos a la Herramienta Tools y encontramos el plug-in deseado (ver Figura 6.1).

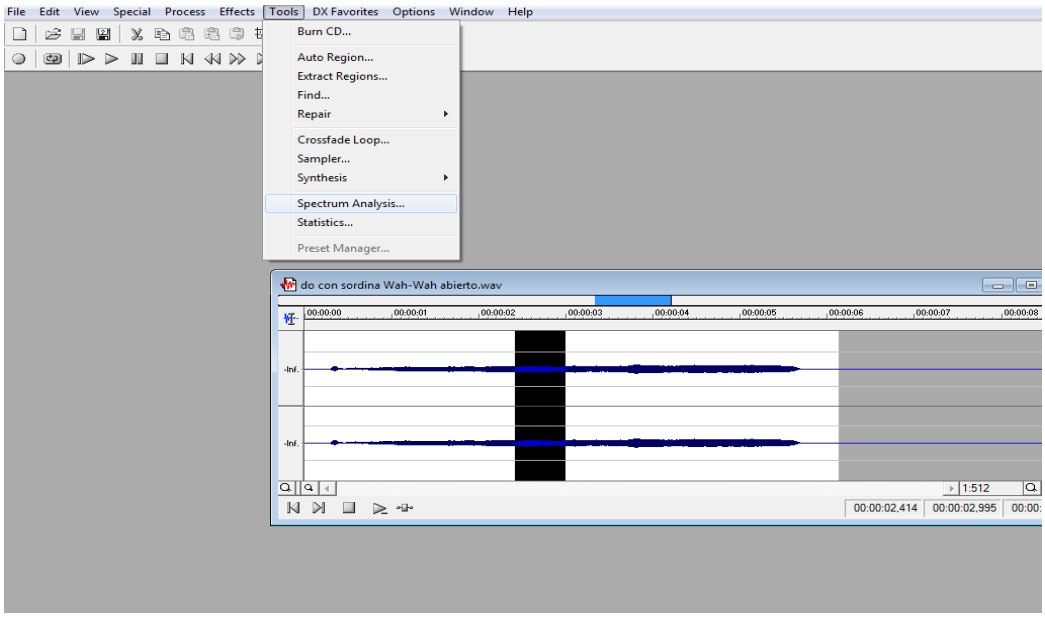


Figura 6.1. Entorno de trabajo de SoundForge

Para realizar el análisis frecuencial lo primero que hice fue abrir cada archivo de audio previamente grabado. Una vez cargado en el software seleccioné una franja de la señal ya que el sonido grabado era una sola nota musical y constante (ver Figura 6.1). A continuación seleccioné el plug-in ya mencionado y analicé las señales del siguiente modo.

La primera señal a analizar fue el archivo de audio “Do sin sordina”. Elegí la nota Do porque es una de las notas fundamentales ya que la vara está desplazada hasta la mitad de su máximo de apertura, hasta la campana en

concreto. Una vez cargado en el plug-in “Spectrum analisys” fui seleccionando varias frecuencias, más o menos por octavas y tomando los valores de cada una de éstas (ver Figuras 6.2, 6.3, 6.4, 6.5 y 6.6).

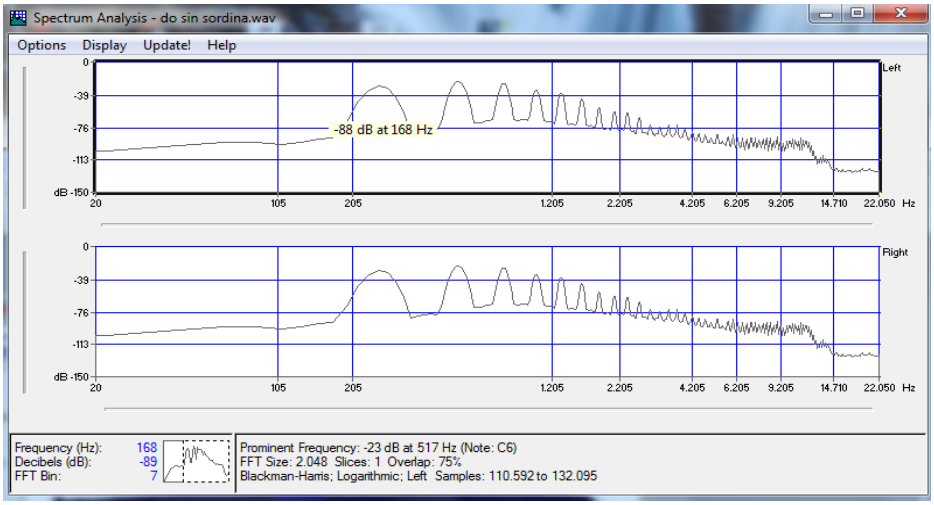


Figura 6.2. Do sin sordina, 168 Hz.

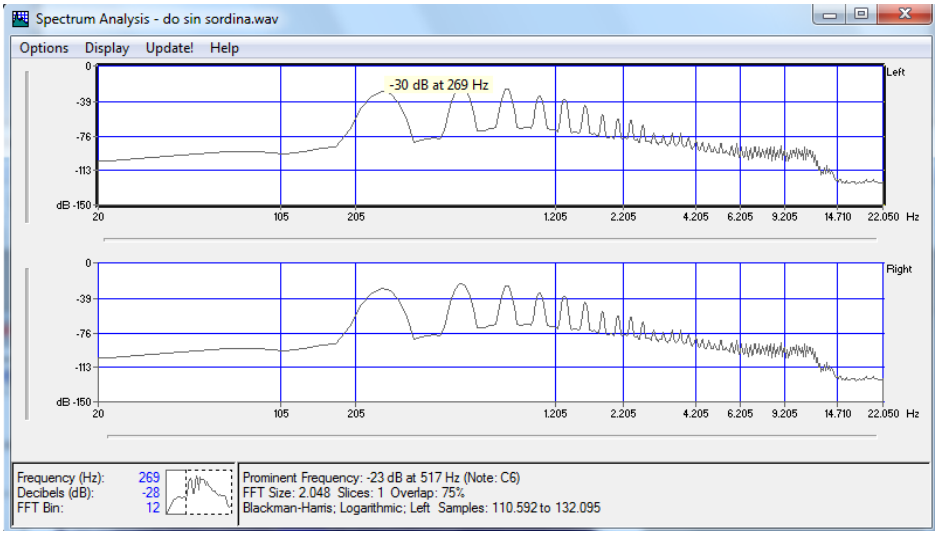


Figura 6.3. Do sin sordina, 269 Hz.

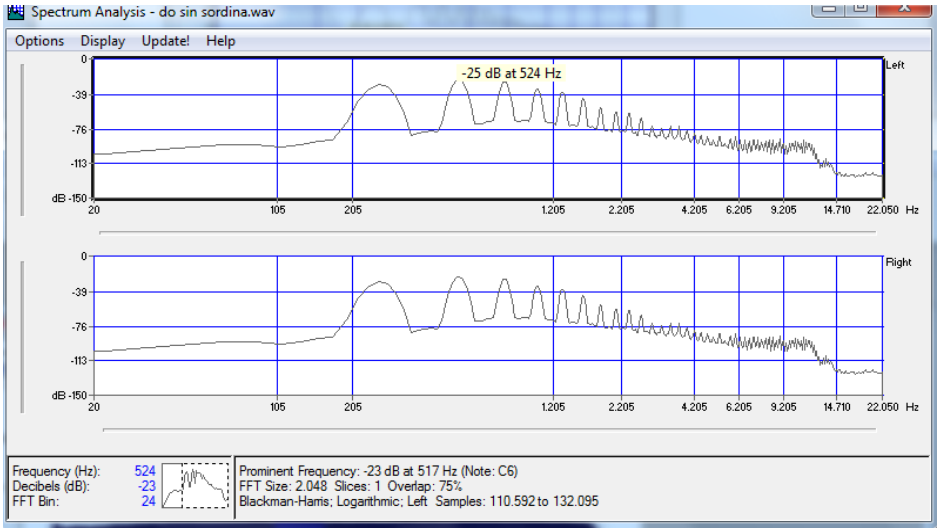


Figura 6.4. Do sin sordina, 524 Hz.

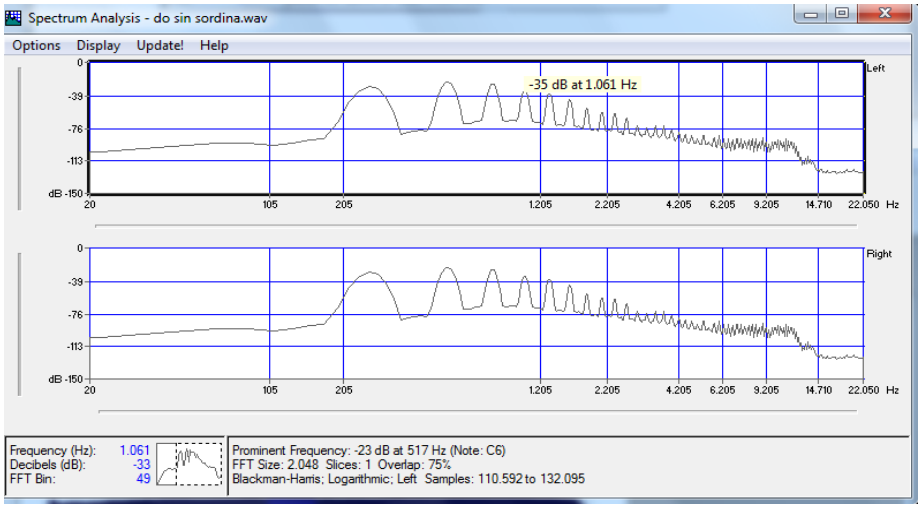


Figura 6.5. Do sin sordina, 1061 Hz.

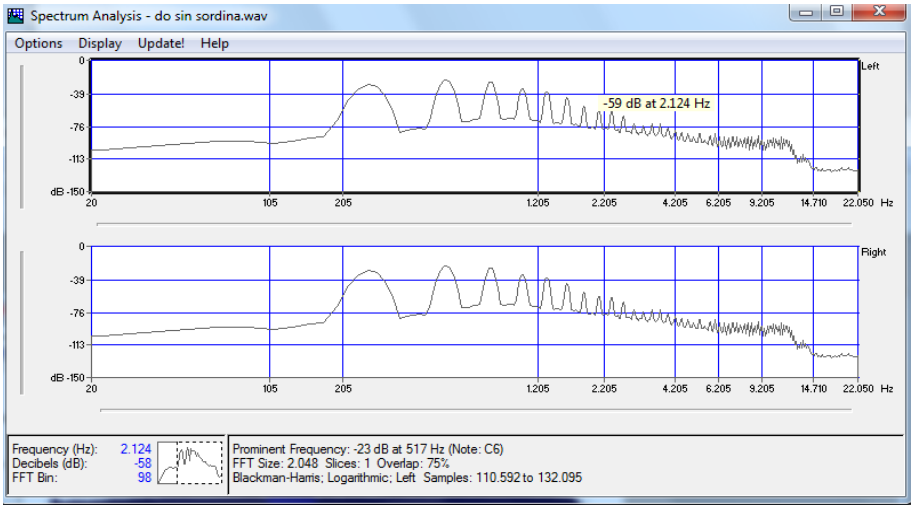


Figura 6.6. Do sin sordina, 2124 Hz.

Una vez tomado los valores, analicé el siguiente archivo de audio “Do con sordina Straight” (ver Figura 6.7, 6.8, 6.9, 6.10 y 6.11) y repetí el proceso descrito anteriormente para poder comparar con el “Do sin sordina”.

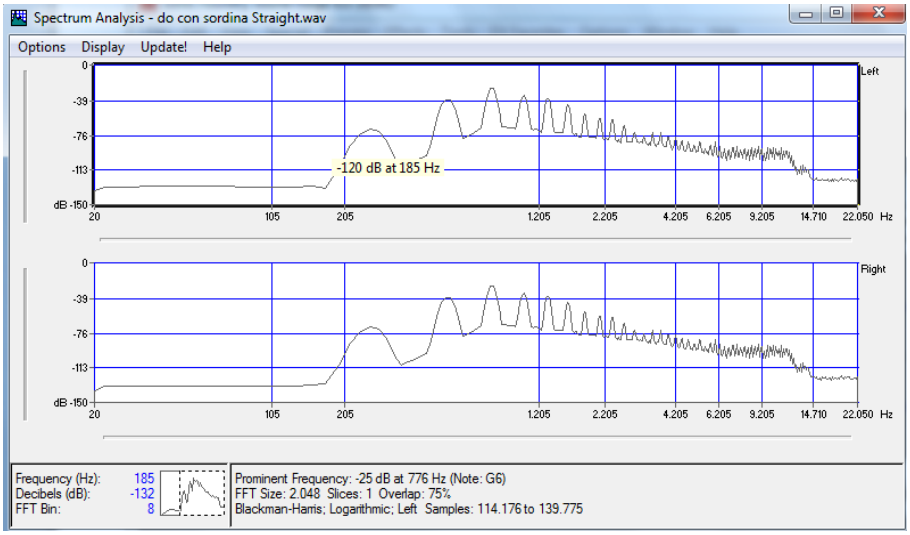


Figura 6.7. Do con sordina Straight, 185 Hz.

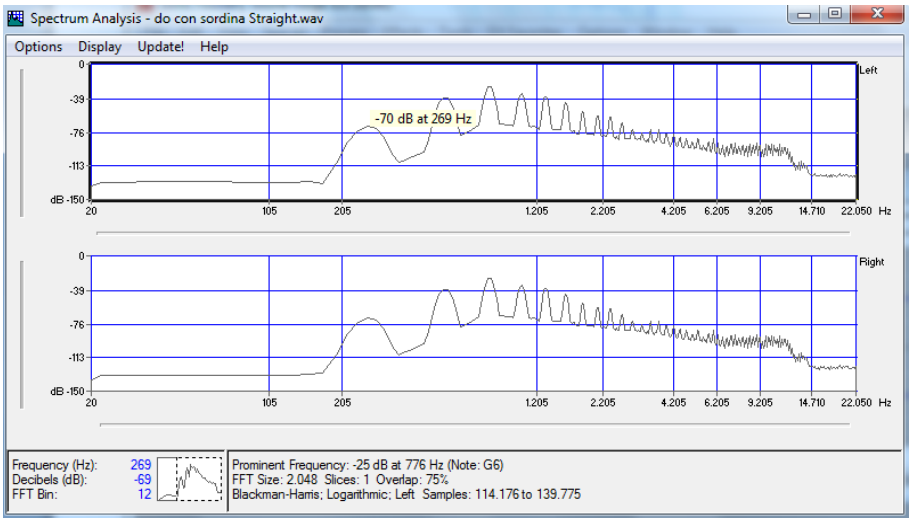


Figura 6.8. Do con sordina Straight, 269 Hz

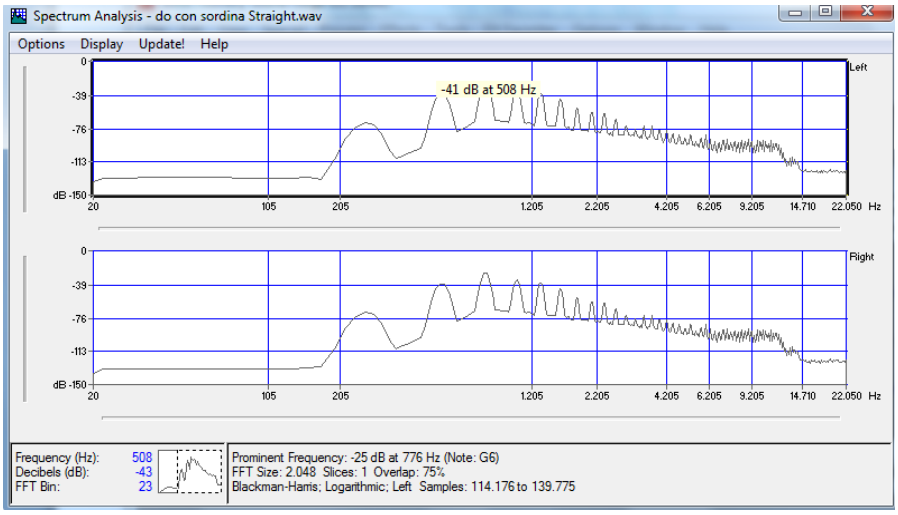


Figura 6.9. Do con sordina Straight, 508 Hz.

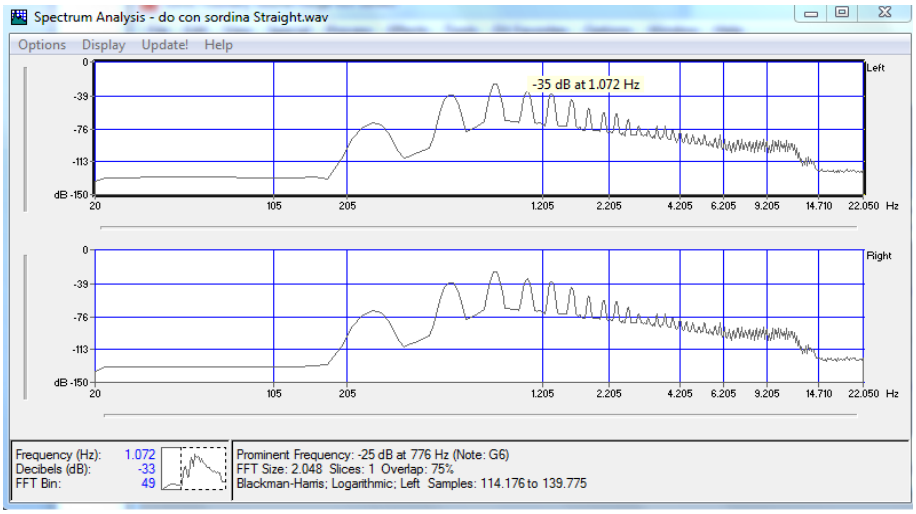


Figura 6.10. Do con sordina Straight, 1072 Hz.

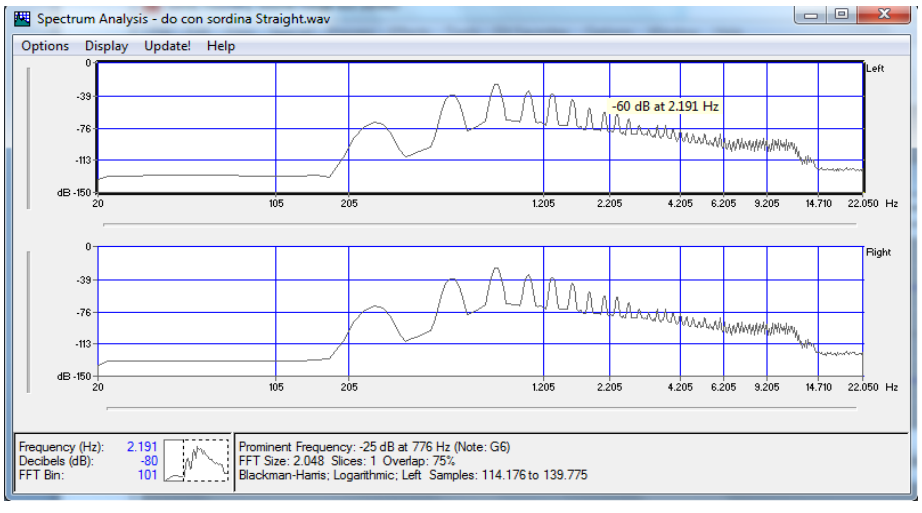


Figura 6.11. Do con sordina Straight, 2191 Hz

No todas las frecuencias tomadas entre las dos muestras coinciden, ya que era bastante impreciso atinar justo en la frecuencia deseada, pero apenas hay distancia entre las frecuencias correspondientes a las dos muestras por lo que me era válido, ya que no había mucha variación de decibelios que me supusiera un problema a la hora de diferenciar entre las dos señales.

Dicho esto era el momento de continuar analizando las siguientes señales con el mismo proceso (ver Figura 6.12 y 6.13).

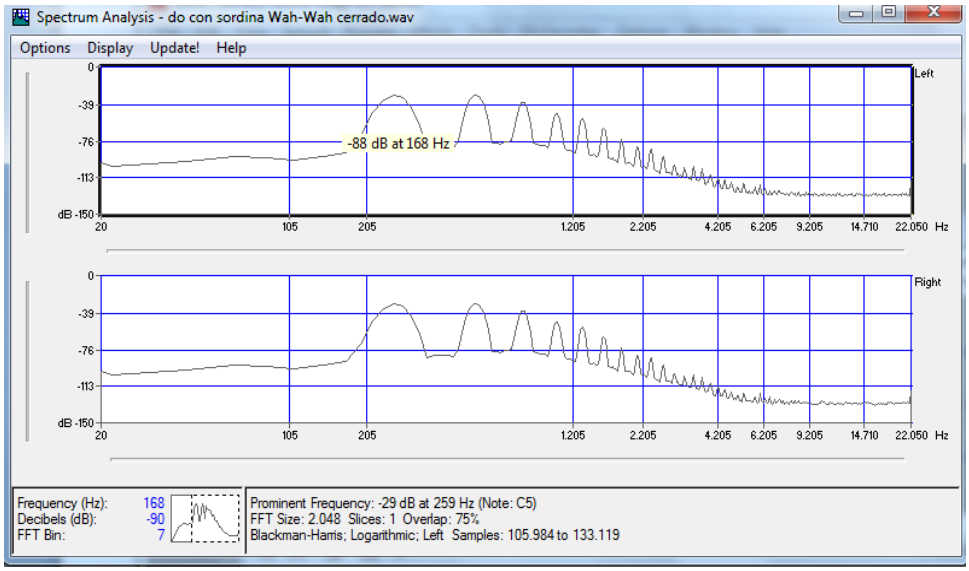


Figura 6.12. Do con sordina Wah-Wah cerrado, 168 Hz.

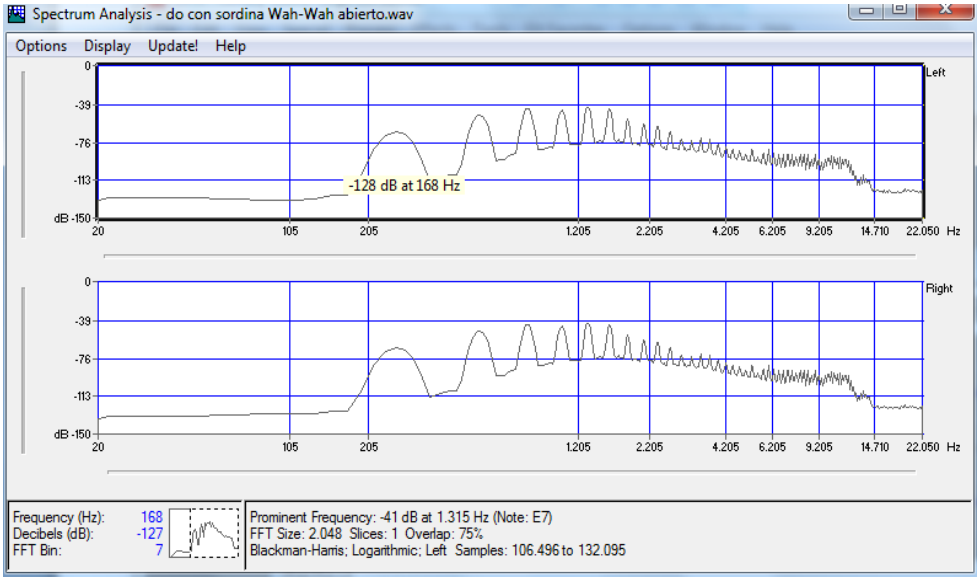


Figura 6.13. do con sordina Wah-Wah abierto, 168 Hz.

Como he comentado el proceso era el mismo, así que he ahorrado en imágenes y he resumido en la siguiente tabla (ver Tabla 6.1) todos los valores obtenidos de las distintas sordinas empleadas en el proyecto:

dB / Frecuencia	168-185 Hz	269-272 Hz	508-529 Hz	1061-1072	2124-2191
Do sin sordina	-89 dB	-28 dB	-23 dB	-33 dB	-58 dB
Sordina Straight	-132 dB	-69 dB	-43 dB	-33 dB	-80 dB
Wah-cerrado	-90 dB	-29 dB	-29 dB	-48 dB	-82 dB
Wah-abierto	-127 dB	-65 dB	-49 dB	-44 dB	-59 dB

Tabla 6.1. Valores por frecuencia de cada señal.

Una vez obtenidos los valores que buscábamos era el momento de diseñar los filtros para poder programar en el Chameleon.

6.3- DISEÑO DE FILTROS

Era el momento de decidir que tipo de filtros diseñar para este trabajo. En primer lugar me decanté por diseñar filtros FIR por el tema de la estabilidad y demás pero finalmente opté por los filtros IIR ya que pueden cumplir las mismas exigencias que los FIR pero con un orden menor, lo cual es importante a la hora de implementarlo con el Chameleon. En concreto decidí diseñarlo de orden 2 porque son más fáciles de implementar y los resultados que se obtienen son satisfactorios.

Para diseñar lo filtros busqué en las prácticas de “sistemas discretos” y de “audio digital”. Finalmente, tras buscar las distintas formas de obtener los

coeficientes de filtros digitales en Matlab encontré el comando FDATool, en el cual puedes diseñar todo tipo de filtros y obtener sus coeficientes, el diagrama polar con los ceros y polos entre otras gráficas.

Lo siguiente era comparar las tres señales que tenían sordina con la de sin sordina. Para ello lo primero que hice fue coger la tabla 6.1, en la que se muestran los niveles de intensidad en dB por frecuencia. El primer caso era claro, “Sin sordina” no requería de filtro alguno así que este lo descartamos.
El segundo caso se trataba de la “sordina Straight”. Comparando los datos obtenidos del análisis frecuencial entre esta señal y la de “sin sordina”, me fijé que a partir de 1KHz más o menos era cuando dejaba de haber diferencia de niveles entre las dos señales, optando así por un filtro pasa-alto con frecuencia de paso de 1KHz.
El tercer caso se trataba de la “sordina Wah-Wah cerrado”. Al igual que en el caso anterior, comparé esta señal con la de “sin sordina” y llegué a la conclusión de que a partir de 500 Hz en adelante variaban los niveles entre las dos señales constituyendo un filtro pasa-bajo. Con el cuarto caso igual, comparando la señal de “sordina Wah-Wah abierto” con la de “sin sordina” obtuve que los niveles empezaban a diferenciarse entre 1-2KHz siendo como en el segundo caso un filtro pasa-alto.

Tras obtener los coeficientes de varias frecuencias de paso para los tres distintos filtros y tras probarlos finalmente en el programa que diseñé en el Chameleon para ver como sonaban y si simulaban con efectividad el efecto de sordina, llegué a la conclusión de que las frecuencias más efectivas eran:

- 800 Hz para el filtro pasa alto, simulando la sordina Straight. En la imagen 6.14 y 6.15 se muestra el programa FDATool de Matlab y el diseño del filtro para Fps=800Hz (frecuencia de paso).

- 500 Hz para el filtro pasa bajo, simulando la sordina Wah-Wah cerrado.

- 1500 Hz para el filtro pasa alto, simulando la sordina Wah-Wah abierto.

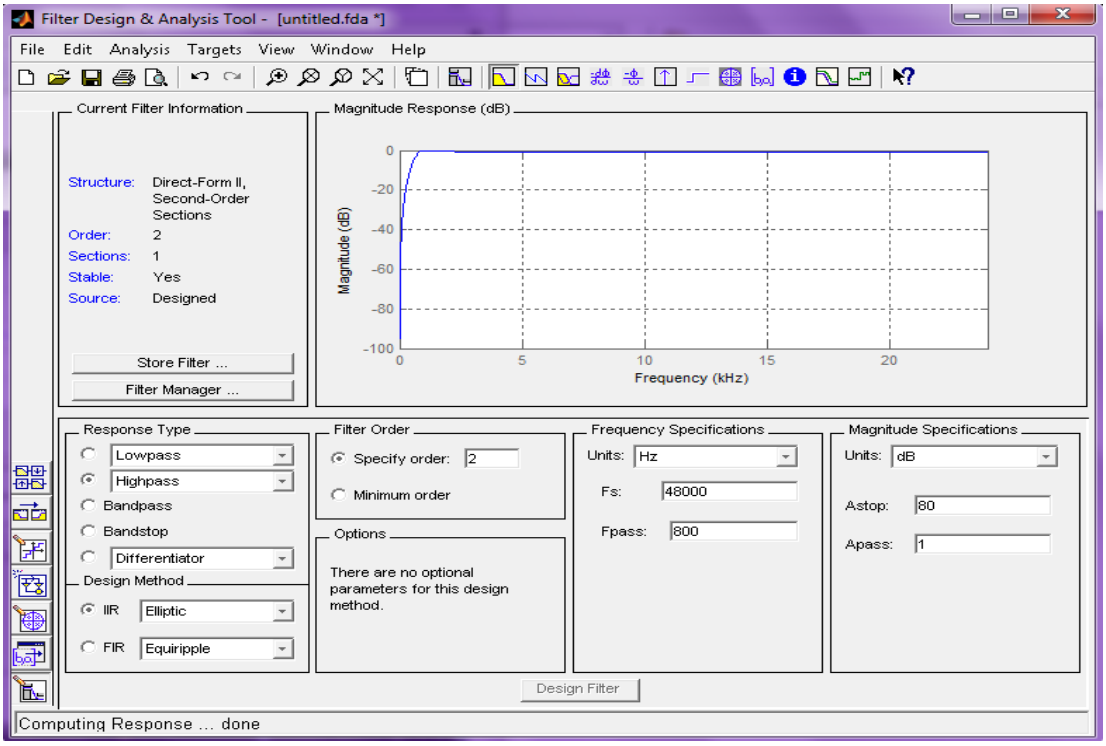


Figura 6.14. Diseño del filtro pasa-alto para Fps=800 Hz.

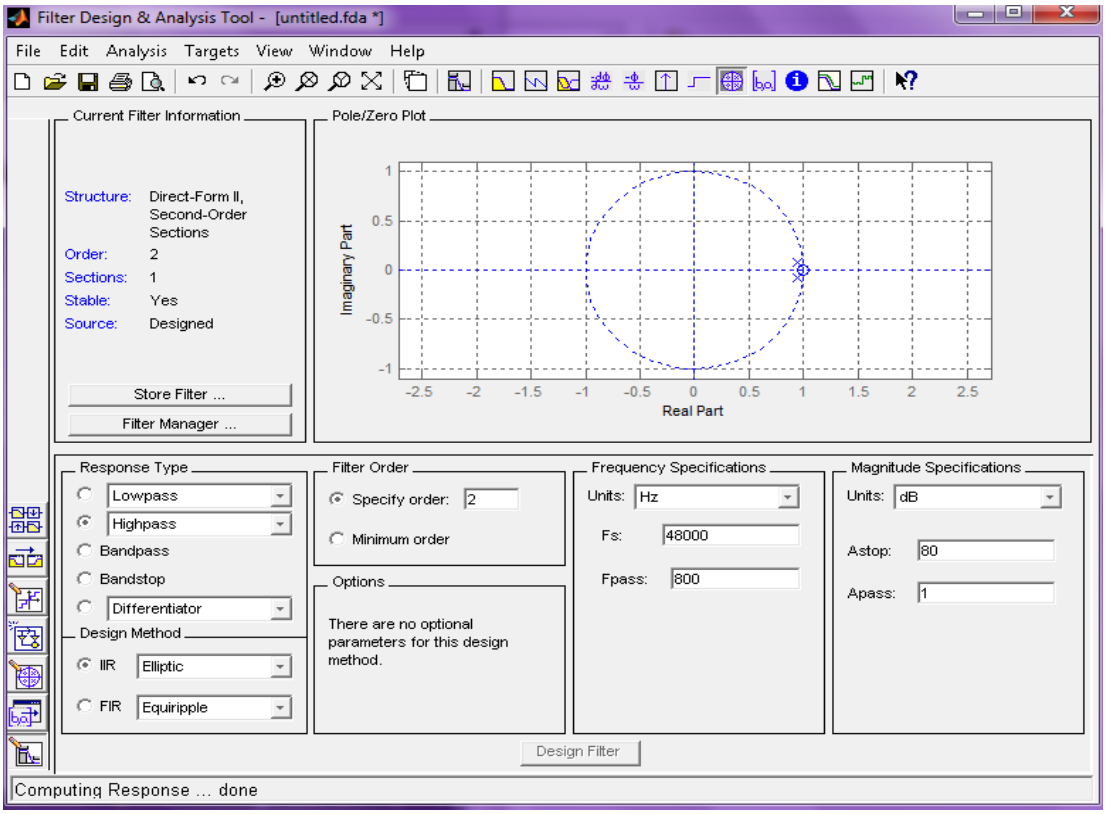


Figura 6.15. Diagrama polar de polos y ceros para filtro pasa-alto Fps=800Hz

Para el filtro pasa-bajo de 500 Hz y para el pasa-alto de 1500 Hz el proceso con el FDATool es el mismo. Los tres filtros son IIR de orden 2 ya que son más fáciles de implementar en el Chameleon, si fueran de un orden mayor el proceso de cálculo en el lenguaje ensamblador sería más largo, costoso, complejo y la sensación sonora sería prácticamente la misma que la que obtendremos con el orden de filtro 2. Por último escogí una frecuencia de muestreo de 48000 Hz que proporcionará más muestras en comparación a la de 44100 Hz.

6.4- IMPLEMENTACIÓN EN EL CHAMELEON

Lo primero fue buscar en las librerías que proporciona el Software una vez instalado, filtros diseñados por lo creadores del Chameleon. Había varios ejemplos bastante complejos de entender así que opté por empezar con el manual de usuario para poder comprender como funciona el Chameleon. Una vez estudiado lo que me podía servir de ayuda como almacenamiento de buffers, edición y control de potenciómetros, envío a la salida de la señal procesada por canal, etc. era el momento de programar basándome en un ejemplo de diseño de filtros, mi propio programa para filtrar a las frecuencias deseadas. Empecé con lo más básico, programar las funciones que quería que realizaran los potenciómetros de la parte frontal del Chameleon. A continuación explicaré los parámetros que cambié para que el panel frontal realice las funciones específicas.

```
##########################################################################
while(1)
    {

        if(!panel_in_new_event(panel,TRUE))
        {
            error("ERROR: unexpected exit waiting new panel event\n");
        }
```

```
if(panel_in_encoder(panel,&encoder,&increment))
{
    variable=variable+increment;
```
-A medida que se incrementa (rueda de potenciómetro)

```
    if(variable>=250)
```
-Si aumenta tanto hasta sobrepasar el valor de 250

```
    {
        variable=64;
    }
```
-El valor de variable será de 64

```
    else if(variable>=64)
```
-Mientras incrementemos y el valor no llegue a ser mayor de 250, se mantenga hasta este valor, el valor de variable será de cero

```
    {
        variable=0;
    }

    variableB=variable/16;
```
-Nuestra variableB será igual a nuestro valor obtenido dividido entre 16, esto es para hacer el encoder más cómodo el encoder.

-En resumidas cuentas, lo que hacemos en este código descrito es que al dar "x" vueltas a la rueda y llegar a los valores de arriba al incrementar este potenciómetro, cambiará de filtro siguiendo un orden tal y como se especifica a continuación.

```
    switch(variableB)
```
-Una vez obtenido variableB con un máximo de 4 como valor, diseñamos nuestro switch para tener nuestro potenciómetro de cuatro opciones

```
{
                    .
    case 0:
        panel_out_lcd_print(panel, 0, 0, "==FX. Sordina==");
```
- Sacamos por pantalla

```
        panel_out_lcd_print(panel, 1, 0, "==Sin Sordina==");

        dsp_write_flag1(dsp,FALSE);
```
-Nuestro primer caso en el que seleccionamos el primer filtro (false-false)

```
        dsp_write_flag0(dsp,FALSE);
        TRACE("Sin Sordina\n");
```
-Lo que aparece en la ventana toolkit

```
        break;

    case 1:
        panel_out_lcd_print(panel, 0, 0, "==FX. Sordina==");
```
- Sacamos por pantalla

```
        panel_out_lcd_print(panel, 1, 0, "===Straight.===");.
        dsp_write_flag1(dsp,FALSE);
```
-Nuestro segundo caso en el

```
                                                    que seleccionamos el
                                                    segundo filtro (false-true)que
                                                    seleccionamos el segundo
                dsp_write_flag0(dsp,TRUE);
                TRACE("Sordina Straight\n");        -Lo que aparece en la
                                                    ventana toolkit
                break;

            case 2:
                panel_out_lcd_print(panel, 0, 0, "==FX. Sordina==");
- Sacamos por pantalla
                panel_out_lcd_print(panel, 1, 0, "wah-wah cerrado");
                dsp_write_flag1(dsp,TRUE);          -Nuestro tercer caso en el que
                                                    seleccionamos el tercer filtro
                                                    (true-false)
                dsp_write_flag0(dsp,FALSE);
                TRACE("Sordina wah-cerrado\n");     -Lo que aparece en la
                                                    ventana toolkit
                break;

            case 3:
                panel_out_lcd_print(panel, 0, 0, "==FX. Sordina==");
- Sacamos por pantalla
                panel_out_lcd_print(panel, 1, 0, "wah-wah abierto");
                dsp_write_flag1(dsp,TRUE);          - Cuarto caso en el que
                                                    seleccionamos el cuarto filtro
                                                    (true-true)
                dsp_write_flag0(dsp,TRUE);
                TRACE("Sordina wah-abierto\n");     -Lo que aparece en la
                                                    ventana toolkit
                break;


            }

        }


    }
```

-Todo este código descrito ya estaba creado para más opciones, no para cuatro. Lo adapté para que cumpliera mi objetivo de selección de cuatro opciones: Sin sordina, sordina Straight, sordina Wah-Wah abierto y sordina Wah-Wah cerrado

##########

Código 6.1. main.c

Todo este código está al completo en el archivo PDF Doc.1, en la carpeta “documentos” dentro de la carpeta “ficheros”.

Ahora era el turno de desarrollar los filtros digitales para simular el efecto de sordina.

Una vez sacadas las frecuencias de paso de prueba, me propuse a diseñar los filtros en el programa Chameleon. Por suerte éste viene con una librería de filtros ya configurados y diseñados que me sirvieron de gran ayuda. A continuación explicaré las principales funciones del código programado.

```
;#####################################################################
##########

    org Y:$000000

;Filtro IIR d2 segundo orden HPF 800Hz

coeficientesHPF1    dc  1*0.5                       -coeficiente b0
                    dc  -1.999998881987597*0.5      -coeficiente b1
                    dc  1*0.5                       -coeficiente b2

                    dc  -1.891610602557218*0.5      -coeficiente a1
                    dc  0.901058424415623*0.5       -coeficiente a2

escaladoHPF1        dc  0.845055193275633           -factor de escala


;Filtro IIR de segundo orden LPF 500 Hz

coeficientesLPF     dc  1*0.5                       -coeficiente b0
                    dc  1.6531108854947349*0.5      -coeficiente b1
                    dc  1*0.5                       -coeficiente b2

                    dc  -1.9261465340632251*0.5     -coeficiente a1
                    dc  0.93070380628122584*0.5     -coeficiente a2

escaladoLPF         dc   0.0011118395435932432      -factor de escala



;Filtro IIR de segundo orden HPF 1500 Hz

coeficientesHPF2    dc  1*0.5                       -coeficiente b0
                    dc  -1.999996051324449*0.5      -coeficiente b1
                    dc  1*0.5                       -coeficiente b2



                    dc  -1.791028010721931*0.5      -coeficiente a1
                    dc  0.822823254687701*0.5       -coeficiente a2

escaladoHPF2        dc  0.805212877524140           -factor de escala
```

```
;#########################################################################
```

Código 6.2. main.asm

En el código de arriba lo que hice fue sustituir el valor de los coeficientes que obtuve en el FDATool (b0, b1, b2, a1, a2 y factor de escalado) para los tres filtros. Cada valor de los coeficientes de los distintos filtros se corresponde con HPF1, coeficientes LPF y coeficientes HPF2, de esta forma llamamos a los coeficientes para hacer las operaciones necesarias como a continuación se explica

En el “Código 6.3” se expone el proceso de filtrado:

```
;#########################################################################
;#########################################################################
; Separamos dos bancos de memoria, uno para el
; canal izquierdo y otro para el canal derecho.
;#########################################################################

;####################
;# Canal Izquierdo #
;####################


coefL:  BRSET #HSR_HF1,X:<<HSR,bit0L
        BRSET #HSR_HF0,X:<<HSR,fir2L
        bra directL

;============================

fir2L   ;MOVE X:punteroL,R0
        BRCLR #SSISR_RDF,X:<<SSISR0,*
        MOVEP X:<<RX0,X:InputLeft


    ;Filtro IIR HPF1 canal izquierdo


    MOVE    X:InputLeft,X1              -Multiplicamos
    CLR     A                           cada muestra
    MACI    #0.845055193275633,X1,A     por el factor de escalado
    MOVE    A,X:InputLeft2
    CLR     A


    MOVE    #coeficientesHPF1,R4        -inicializamos puntero de
                                        coeficientes y estados

    MOVE    X:StateInLeftPtr,R0         -inicializamos puntero de
                                        estados de entradas (x)
```

```
    MOVE    X:StateOutLeftPtr,R1        -inicializamos puntero de
                                        estados de salida (y)
    MOVE    #(orden-1),M0               -accedemos a los estados en
                                        modulo 2
    MOVE    M0,M1

    MOVE    X:InputLeft2,X0             -cogemos entradas de x[n] y
                                        las ponemos en X0
    MOVE    Y:(R4)+,Y0                  -cogemos b0 que están listos
                                        para filtrar


    MPY X0,Y0,A     X:(R0)+,X1  Y:(R4)+,Y0  -A=b0*x[n],cogemos x[n-1]
                                            y b1

    MAC X1,Y0,A     X:(R0),X1   Y:(R4)+,Y0  -A=A+b1*x[n-1], cogemos
                                            x[n-2] y b2

    MAC X1,Y0,A     X:(R1)+,X1  Y:(R4)+,Y0  -A=A+b2*x[n-2], cogemos
                                            y[n-1] y a1

    MAC -X1,Y0,A    X:(R1),X1   Y:(R4)+,Y0  -A=A-a1*y[n-1], cogemos
                                            y[n-2] y a2

    MAC -X1,Y0,A    X0,X:(R0)               -A=A-a2*y[n-2], guardamos
                                            x[n] para la próxima
                                            iteración

    ASL A                                   -A=y[n]=2*A (escalamos la
                                            salida)

    MOVE    R0,X:StateInLeftPtr         -guardamos los punteros para
                                        La próxima iteración
    MOVE    R1,X:StateOutLeftPtr        -guardamos los punteros de
                                        estado de salida para la
                                        próxima iteración
    MOVE    A,X:(R1)                    -guardamos y[n] para la
                                        Próxima iteración
    bra outL


;==================================

bit0L   BRSET   #HSR_HF0,X:<<HSR,fir4L
        BRCLR   #SSISR_RDF,X:<<SSISR0,*
        MOVEP   X:<<RX0,X:InputLeft


    ;Filtro IIR LPF canal izquierdo


    MOVE    X:InputLeft,X1              -Multiplicamos
    CLR     A                           cada muestra
    MACI    #0.0011118395435932432,X1,A por el factor de escalado
    MOVE    A,X:InputLeft2
    CLR A

    MOVE    #coeficientesLPF,R4         -inicializamos puntero de
                                        coeficientes y estados
    MOVE    X:StateInLeftPtr,R0         -inicializamos puntero de
                                        estados de entradas (x)
```

```
    MOVE    X:StateOutLeftPtr,R1        -inicializamos puntero de
                                        estados de salida (y)

    MOVE    #(orden-1),M0               -accedemos a los estados en
                                        modulo 2

    MOVE    M0,M1

    MOVE    X:InputLeft2,X0             -cogemos entradas de x[n] y
                                        las ponemos en X0

    MOVE    Y:(R4)+,Y0                  -cogemos b0 que están listos
                                        para filtrar


    MPY X0,Y0,A     X:(R0)+,X1  Y:(R4)+,Y0  -A=b0*x[n],cogemos x[n-1]
                                            y b1

    MAC X1,Y0,A     X:(R0),X1   Y:(R4)+,Y0  -A=A+b1*x[n-1], cogemos
                                            x[n-2] y b2

    MAC X1,Y0,A     X:(R1)+,X1  Y:(R4)+,Y0  -A=A+b2*x[n-2], cogemos
                                            y[n-1] y a1

    MAC -X1,Y0,A    X:(R1),X1   Y:(R4)+,Y0  -A=A-a1*y[n-1], cogemos
                                            y[n-2] y a2

    MAC -X1,Y0,A    X0,X:(R0)               -A=A-a2*y[n-2], guardamos
                                            x[n] para la próxima
                                            iteración

    ASL A                               -A=y[n]=2*A (escalamos la
                                        salida)

    MOVE    R0,X:StateInLeftPtr         -guardamos los punteros para
                                        la próxima iteración

    MOVE    R1,X:StateOutLeftPtr        -guardamos los punteros de
                                        estado de salida para la
                                        próxima iteración

    MOVE    A,X:(R1)                    -guardamos y[n] para la
                                        próxima iteración

    bra outL


;====================================


fir4L   BRCLR   #SSISR_RDF,X:<<SSISR0,*
        MOVEP   X:<<RX0,X:InputLeft


    ;Filtro IIR HPF2 canal izquierdo

    MOVE    X:InputLeft,X1              -Multiplicamos
    CLR A                               cada muestra
```

```
    MACI    #0.805212877524140,X1,A     por el factor de escalado
    MOVE    A,X:InputLeft2
    CLR A


    MOVE    #coeficientesHPF2,R4        -inicializamos puntero de
                                        coeficientes y estados

    MOVE    X:StateInLeftPtr,R0         -inicializamos puntero de
                                        estados de entradas (x)

    MOVE    X:StateOutLeftPtr,R1        -inicializamos puntero de
                                        estados de salida (y)

    MOVE    #(orden-1),M0               -accedemos a los estados en
                                        modulo 2

    MOVE    M0,M1



    MOVE    X:InputLeft2,X0             -cogemos entradas de x[n] y
                                        las ponemos en X0
    MOVE    Y:(R4)+,Y0                  -cogemos b0 que están listos
                                        para filtrar

    MPY X0,Y0,A     X:(R0)+,X1  Y:(R4)+,Y0  -A=b0*x[n],cogemos x[n-1]
                                            y b1

    MAC X1,Y0,A     X:(R0),X1   Y:(R4)+,Y0  -A=A+b1*x[n-1], cogemos
                                            x[n-2] y b2

    MAC X1,Y0,A     X:(R1)+,X1  Y:(R4)+,Y0  -A=A+b2*x[n-2], cogemos
                                            y[n-1] y a1

    MAC -X1,Y0,A    X:(R1),X1   Y:(R4)+,Y0  -A=A-a1*y[n-1], cogemos
                                            y[n-2] y a2

    MAC -X1,Y0,A    X0,X:(R0)               -A=A-a2*y[n-2], guardamos
                                            x[n] para la próxima
                                            iteración

    ASL A                                   -A=y[n]=2*A (escalamos la
                                            salida)

    MOVE    R0,X:StateInLeftPtr         -guardamos los punteros para
                                        la próxima iteración
    MOVE    R1,X:StateOutLeftPtr        -guardamos los punteros de
                                        estado de salida para la
                                        próxima iteración

    MOVE    A,X:(R1)                    -guardamos y[n] para la
                                        próxima iteración

    bra outL

;================================================

directL:    BRCLR #SSISR_RDF,X:<<SSISR0,*
            MOVEP X:<<RX0,X1
            clr a
```

```
            move x1,a
            BRA outL


outL:       BRCLR #SSISR_TDE,X:<<SSISR0,*   -Espera hasta que el
                                            registro de salida  este
                                            vacío

            MOVEP A,X:<<TX00                -Saca la salida por el
                                            canal correspondiente, en
                                            este caso el izquierdo




;#################
;# Canal derecho #
;#################


coefR:  BRSET #HSR_HF1,X:<<HSR,bit0R
    BRSET #HSR_HF0,X:<<HSR,fir2R
    bra directR




;====================================

fir2R
    BRCLR   #SSISR_RDF,X:<<SSISR0,*
    MOVEP   X:<<RX0,X:InputRight

;#####################################################################
```

Código 6.3. main.asm

En el Código 6.3, se muestra a la derecha la explicación del proceso que sigue el código programado.

Lo último que me quedaba era escuchar y probar los coeficientes que había obtenido de las distintas frecuencias de paso para cada sordina. Finalmente y tras hacer varias pruebas me quedé con las frecuencias ya mencionadas en el apartado 6.3.

Dentro de mi experiencia personal, los errores que cometí a la hora de programar y que más me costaron solucionar fueron el de no multiplicar los valores de los coeficientes por 0.5. Esto se traducía en un pitido agudo constante a la hora de ejecutar el programa en el Toolkit que conseguí solucionar multiplicando cada valor por 0.5. Otro error que cometí fue a la hora de multiplicar las muestras por el factor de escalado ya que tenía que poner directamente el valor del factor y no "llamarlo" como hago con los valores de coeficiente (ver Código 6.4.).

```
;#####################################################################
##########
    MOVE    X:InputLeft,X1              -Multiplicamos
    CLR A                               cada muestra
    MACI    #0.805212877524140,X1,A     por el factor de escalado
    MOVE    A,X:InputLeft2
    CLR A
;#####################################################################
##########
```

Código 6.4. Ejemplo correcto de la multiplicación de las muestras por el factor de escalado.

6.5- ANÁLISIS DE RESULTADOS

Tras corregir los errores cometidos en los algoritmos desarrollados, el programa funcionaba a la perfección. El potenciómetro principal y más grande había sido el escogido para que al ser rodado pasara por los diferentes estados.

El primer estado "Sin Sordina" no procesaba el audio que entraba al aparato por lo que se escuchaba perfectamente el instrumento sin ninguna variación.

El segundo estado “Sordina Straight” filtraba la señal de entrada a la perfección, un filtro pasa alto que simulaba con bastante exactitud la sordina Straight.

El tercer estado “Sordina Wah-Wah cerrado” filtraba la señal de tal forma que se escuchaba las frecuencias bajas, tal y como tiene que hacer esta sordina en modo cerrado y el resultado comparándolo con una sordina de este tipo en directo es bastante similar por lo que el resultado es satisfactorio.

El último caso es el que más problemas dio ya que filtraba la señal bien pero como la frecuencia de paso es más alta y elimina gran parte de las frecuencias bajas, hace que pierda calidez el instrumento en el proceso de filtrado y parezca más artificial la simulación. El efecto en sí está logrado pero no queda tan real como en los casos anteriores.

En apariencia y en una situación real tal como se explica en el apartado 1-INTRODUCCIÓN Y OBJETIVOS, la simulación de los trombones con sordina mediante Chameleon sería una opción adecuada y rápida para solucionar el problema ya descrito.

7. Conclusiones

Después de haber desarrollado y explicado el procedimiento que he seguido para concluir con satisfacción el trabajo mencionaré las conclusiones a las que he llegado.

Lo primero es mencionar los conocimientos adquiridos tras leer varios libros sobre la física del sonido y en especial a Daniel Ribes Blanco por las explicaciones sobre los distintos tipo y funcionamiento de sordinas, basados en tapones cuya función es impedir la correcta salida de aire y vibración del instrumento, traduciéndose o transformándose en distintos tipos de filtros para audio digital que simulan este efecto. También destacar el repaso realizado sobre el proceso y tratamiento que sigue el audio, desde que empieza o se reproduce el audio analógico, se digitaliza, se procesa y vuelve a su estado natural así como los distintos tipos de filtros digitales, en especial los IIR que son los utilizados para este proyecto. No paso por alto el conocimiento adquirido en Matlab y la gran variedad de herramientas y entornos gráficos que proporciona a la hora de diseñar filtros digitales, habiendo hecho el trabajo realizado más sencillo y práctico a la hora de entender el funcionamiento de estos.

Lo aprendido sobre el sistema Chameleon y las múltiples herramientas que proporciona es la principal tarea a destacar en este trabajo. El aprendizaje sobre la creación y desarrollo de filtros es solo una pequeña parte de lo que te enseña Chameleon ya que proporciona una gran variedad de efectos con opciones de desarrollarlos aún más para un uso exclusivo y particular. Todo lo aprendido sobre lenguaje ensamblador será de gran ayuda ya no sólo por este sistema, sino porque existen muchos otros más nuevos y a su vez complejos con el mismo propósito que el Chameleon, desarrollar tus propios efectos digitales para audio.

Finalmente he llegado a la conclusión tras haber realizado este trabajo la unión que existe y que existirá entre los elementos artísticos y tecnológicos,

elementos que se complementan creando un sinfín de posibilidades creativas que facilitan al desarrollo de nuevas formas de ver y escucha el arte.

8. Bibliografía

- “Principios de audio digital” Ken C. Pohlmann, Ed. McGraw-Hill Professional

- “Audio digital y MIDI” Sergi Jordà Puig, Ed. Anaya Multimedia

- “Dispositivo reprogramable chameleon: aplicaciones prácticas y docentes”, Proyecto final de carrera José Francisco Bellver Molla. Universidad Politécnica de Valencia. Escuela Politécnica Superior de Gandía.

- “Procesadores digitales de señales: prácticas de laboratorio utilizando chameleon (dispositivo dsp reprogramable de 24 bits)” Francisco Carlos Marcos Laserna. Universidad Politécnica de Valencia. Escuela Politécnica Superior de Gandía.

- “Principles of vibration and sound” Thomas D. Rossing, Ed. Springer

- “Manual imprescindible de C/C++” Miguel Ángel Acera García, Ed. Anaya Multimedia

- http://www.mathworks.com/ - Consultas de Matlab

- http://www.soundonsound.com/ - Consultas de Chameleon

- http://www.chameleon.synth.net/ - Consultas de Chameleon

- http://www.chameleon.synth.net/ - Manual de usuario del Chameleon

9. Anexos

El presente trabajo se entrega en formato digital. A continuación se describe los archivos y documentos de interés que se encuentran dentro del CD.

Dentro del CD encontraremos la Memoria y una carpeta que pone ficheros Anexos. Dentro de esta tenemos:

- Carpeta “Archivos de audio”. En esta encontramos las pistas de audio que se pueden utilizar para hacer la simulación. Hay una gran variedad de audios, desde el trombón sonando con diferentes sordinas hasta distintas notas para poder comparar en el Chameleon.
- Carpeta “Códigos”. En esta se encuentra dos PDF, uno es el código programado en C (main.c) y el otro en lenguaje ensamblador (main.asm).
- Carpeta “Sordina 2-TFC”. Dentro de esta encontramos el proyecto creado en Chameleon listo para ejecutar. Para ejecutarlo hay que abrir el programa Chameleon Development Environment , abrir el proyecto y al ejecutarlo se creará un archivo.elf, este habrá que abrirlo en el ChameleonToolkit y ya podremos simular el efecto de sordina.