UNIVERSIDAD POLITECNICA DE VALENCIA

E S C U E L A P O L I T E C N I C A S U P E R I O R DE G A N D I A

Grado en Ing. Sist. de Telecom., Sonido e Imagen

“Implementación en DSP de algoritmo anti-feedback acústico”

TRABAJO FINAL DE GRADO

Autor/a:

Guillermo González Montañana

Tutor/a:

José Marín-Roig Ramón

GANDIA, 2017

ÍNDICE

Resumen **6**
Resum **7**
Abstract **8**
1 -. Introducción **9**
1.1 -. Objetivos 10
1.2 -. Metodología y organización de la memoria 10
1.3 -. Etapas 11
2 -. Tecnologías utilizadas **12**
2.1 -. DSP 12
2.1.1 -. Blackfin (ADSP-BF706) 13
2.1.2 -. SHARC (ADSP-21489) 14
2.2 -. Arquitectura de un DSP 15
2.2.1 -. Arquitectura Harvard 15
2.2.2 -. Arquitectura Super Harvard 16
2.3 -. Crosscore Embedded Studio 17
2.4 -. MATLAB 18
3 -. Fundamentos teóricos **19**
3.1 -. El problema del feedback acústico 19
3.2 -. Métodos de control automático del feedback acústico. 22
3.2.1 -. Métodos de modulación de fase (PM) 22
3.2.2 -. Métodos de reducción de ganancia. 23
3.2.3 -. Métodos de filtrado espacial. 24
3.2.4 -. Métodos de modelado de sala. 24
3.3 -. Control del feedback mediante modulación de la fase (PFC) 25
3.3.1 -. Concepto 25
3.3.2 -. Realización 27
3.4 -. Transformada de Hilbert 28
3.5 -. Formatos numéricos en la programación 29
3.5.1 -. Coma fija 29
3.5.2 -. Coma flotante 30
4 -. Diseño e implementación **30**
4.1 -. Diseño del prototipo usando MATLAB 31
4.1.1 -. Transformada de Hilbert ideal 32
4.1.2 -. Transformada de Hilbert mediante Filtro FIR 32

4.1.3 -. Transformada de Hilbert mediante FFT 34
4.1.4 -. Transformada de Hilbert mediante filtro FIR basado en FFT 36
4.1.5 -. Resultados 37
4.2 -. Implementación del algoritmo en el DSP 40
4.2.1 -. Método PFC en SHARC 40
4.2.2 -. Método PFC en Blackfin 42
5 -. Pruebas y resultados 44
6 -. Problemas y soluciones 47
7 -. Conclusiones 48
8 -. Bibliografía 50

ÍNDICE DE FIGURAS

Figura 1: Funcionamiento del buffer circular. Fuente: http://www.dspguide.com/pdfbook.htm 13
Figura 2: Placa ensambladora del ADSP-BF706. Fuente: www.analog.com 14
Figura 3: Placa ensambladora del ADSP-21489. Fuente: www.analog.com 15
Figura 4: Esquema básico de arquitectura Harvard. Fuente: http://www.dspguide.com/pdfbook.htm 16
Figura 5: Esquema arquitectura Super Harvard. Fuente: http://www.dspguide.com/pdfbook.htm 16
Figura 6: Esquema de la distribución de un sistema de sonorización. Fuente: Propia. 19
Figura 7: Esquema discreto del bucle cerrado. Fuente: Propia. 20
Figura 8: Esquema discreto de un solo canal. Fuente: (van Waterschoot y Moonen, 2011) 21
Figura 9: Comparación de la señal obtenida del filtro FIR con la señal ideal. Fuente: Captura propia. 34
Figura 10: Comparación de la señal obtenida con FFT y la señal ideal. Fuente: Captura propia. 35
Figura 11: Error producido por la técnica de FFT. Fuente: Captura propia. 36
Figura 12: Ejemplo del método de solapamiento. Fuente: http://www.dspguide.com/pdfbook.htm 37
Figura 13: Diagrama ecuación anti-feedback. Fuente: Propia. 41
Figura 14: Circulo unidad. Fuente: Propia. 43
Figura 15: Esquema de la disposición de los elementos en la fase de pruebas. Fuente: Propia. 45

ÍNDICE DE TABLAS

Tabla 1: Resultados RMS del filtro FIR 38
Tabla 2: Resultados RMS de la FFT. 38
Tabla 3: Resultados RMS del filtro FIR basado en FFT. 39
Tabla 4: Valores RMS mínimos obtenidos con diferentes coeficientes. 45

Resumen

Esta memoria recoge información detallada acerca del proceso de diseño, desarrollo e implementación del presente Trabajo Fin de Grado (TFG). El objetivo del proyecto es la implementación de un algoritmo anti-feedback acústico digital utilizando un microprocesador especializado en el procesamiento de señales digitales: DSP. En concreto se trabaja con dos tipos de DSP, denominados Blackfin y SHARC. El estudio e implementación del algoritmo se ha realizado en dos fases: la primera fase, el prototipo, para el cual se ha utilizado el programa MATLAB, y la segunda, implementación del algoritmo final en el DSP, para la cual se ha utilizado el programa CrossCore de Analog Devices.

Palabras Clave

Algoritmo, DSP, Blackfin, SHARC, transformada de Hilbert

Resum

Aquesta memòria recull informació detallada arran del procés de disseny, desenvolupament i implementació del present Treball Fi de Grau (TFG). L'objectiu del projecte és la implementació d'un algoritme antifeedback acústic digital fent ús d'un microprocessador especialitzat en el processament de senyals digitals: DSP. En concret es treballa amb dos tipus de DSP, denominats Blackfin i SHARC. L'estudi i implementació de l'algoritme ha sigut realitzat en dos fases: la primera fase, el prototip, per al qual s'ha utilitzat el programa MATLAB, i la segona, implementació de l'algoritme final, per al qual s'ha fet ús del programa CrossCore d'Analog Devices.

Paraules clau

Algoritme, DSP, Blackfin, SHARC, transformada de Hilbert

Abstract

This report includes detailed information about the design process development and implementation of this TFG. The objective of this proyect is the implementation of a digital antifeedback algorithm using a microprocessor specialized in digital signal processing: DSP. Specifically, we work with two types of DSP, called Blackfin and SHARC. The study and implementation of the algorithm has been done in two stages: the first stage, the prototype, for which the MATLAB program has been used; and the second stage, the implementation of the final algorithm in DSP, for which Crosscore IDE from Analog Devices has been used.

keywords

Algorithm, DSP, Blackfin, SHARC, Hilbert transform

1 -. Introducción

Desde la aparición de los sistemas de sonorización y PA (public addres), el feedback acústico ha sido un problema. Siempre que se utiliza un micrófono para captar una señal de audio, que luego es procesada, amplificada y vuelta a reproducir a través de la PA, es inevitable que la señal del altavoz sea realimentada al micrófono. Cuando esto ocurre, se crea un bucle cerrado que afecta al rendimiento del sistema, deteriorando la calidad del sonido y limitando la amplificación. Dentro de los efectos producidos por el feedback, el acople de ciertas frecuencias o *howling effect* es el más molesto. Este efecto es debido a las reflexiones del sonido producidas por las características de la sala, y no al sonido directo emitido por los altavoces o PAs, ya que en los escenarios se intenta que estos componentes y el micrófono no se encuentren enfrentados.

El control del feedback acústico se refiere al proceso de intentar anular o reducir lo máximo posible el efecto de feedback acústico. Normalmente, el encargado de que no se produzca es el técnico de sonido que, mediante un ecualizador paramétrico y técnicas de ecualización, busca y corrige manualmente las frecuencias afectadas. Gracias a la evolución tecnológica y del mundo digital, es posible diseñar e implementar un algoritmo en un microprocesador que se encargue de esta tarea, llegando al punto de que no haya que preocuparse por este efecto tan molesto.

En este proyecto se trata, desde el punto de vista teórico-práctico, el proceso de diseño e implementación de un algoritmo capaz de reducir el efecto de feedback lo máximo posible, teniendo en cuenta las características del dispositivo en el que se va a implementar y respetando la mejor relación posible entre calidad de audio y carga computacional del DSP.

1.1 -. Objetivos

Además del fin señalado anteriormente, existen diferentes objetivos del presente proyecto que son:

- El estudio de distintos algoritmos de anti-feedback acústico y que características favorables o desfavorables ofrece cada uno.
- Profundizar en la implementación del algoritmo, concretamente en el método PFC (Phase-modulation Feedback Control), teniendo en cuenta las limitaciones que puede ofrecer un modelo comercial de DSP.
- Realizar un estudio en un ámbito práctico del algoritmo utilizado.
- Comprobar las limitaciones del algoritmo implementado, y sopesar futuras líneas de estudio.

1.2 -. Metodología y organización de la memoria

Después de estudiar que necesidades cubrir, se ha dividido en bloques funcionales o subsistemas y se ha definido un listado de objetivos. Primero se ha decidido programar el prototipo en un entorno controlado donde se puedan realizar simulaciones, con el fin de comprobar el buen funcionamiento del código y elegir las características adecuadas. Al tener esta parte acabada, se ha continuado con su diseño esta vez sí, en el DSP, reprogramando el código utilizado para adaptarlo a las características de programación de cada DSP. Como cada uno de los dos DSP utilizados esta optimizado para un formato de programación (fixed-point o floating-point), se ha desarrollado un código distinto para cada uno, buscando las funciones necesarias y su implementación en los manuales de cada dispositivo.

Finalmente, se han programado los dos dispositivos para probar su correcto funcionamiento y pulir detalles.

Con el fin de seguir la metodología anterior, la memoria se ha estructurado de la siguiente manera:

- Capítulo 1: En el que nos encontramos, muestra una introducción de la memoria y se trata los objetivos, metodología y etapas del proyecto.

- Capítulo 2: Se habla de las tecnologías utilizadas en la realización del proyecto.

- Capítulo 3: Se explican los fundamentos teóricos necesarios para la comprensión del proyecto, métodos y técnicas utilizados, herramientas matemáticas, …

- Capítulo 4: Se explica detalladamente los pasos seguidos para la realización del presente proyecto.

- Capítulo 5: Se explican las pruebas realizadas y se comentan los resultados obtenidos tras su realización.

- Capítulo 6: Cuenta los problemas surgidos y como se solucionaron.

- Capítulo 7: Se comentan unas conclusiones generales, sobre lo que ha sido la realización y montaje del trabajo.

1.3 -. Etapas

Tras la exposición de la metodología y organización del proyecto, se pretende comentar las diferentes etapas por las que se ha tenido que pasar desde el comienzo hasta la finalización del trabajo.

1. Investigación sobre los DSP y CrossCore.
2. Organización y búsqueda de las necesidades.
3. Elección de elementos hardware.
4. Investigación y aprendizaje sobre la programación en tiempo real de audio.
5. Familiarización con el lenguaje utilizado mediante prácticas ajenas al proyecto.
6. Investigación y aprendizaje sobre la teoría de control de feedback acústico y los distintos métodos existentes.
7. Elección del método de control de feedback por sus características.
8. Planteamiento del prototipo y su implementación en un entorno controlado (Matlab).
9. Investigación, programación y comprobación de diferentes algoritmos en Matlab. Realizando las pruebas necesarias para obtener información suficiente para la elección del algoritmo final.
10. Implementación del algoritmo, primero en el DSP Blackfin y luego en el SHARK, mediante CrossCore IDE.
11. Realización de pruebas y obtención de resultados.

2 -. Tecnologías utilizadas

En este apartado se estudian las distintas tecnologías que forman parte del proyecto, tales como componentes hardware, software, entornos de programación, etc…

2.1 -. DSP

DSP (Digital Signal Processing) puede hacer referencia bien al área de la ciencia computacional que trabaja con un único tipo de datos, las señales, o bien al dispositivo especializado en esta tarea. En este proyecto, cuando se habla de DSP, se hace referencia al segundo término.

Un dispositivo DSP es principalmente un microprocesador que se encarga de procesar señales digitales, pero consta de diversas cualidades que lo diferencian del resto de microprocesadores. Los DSPs realizan de forma rápida y repetida operaciones matemáticas basadas en sumas y multiplicaciones (filtros digitales, FFT, …), y están especializados en el procesamiento de señales a “tiempo real”. Una señal de audio es continuamente convertida a digital en un ADC, procesada por el DSP y convertida de nuevo a forma analógica mediante un DCA. Esto es lo que diferencia los DSP del resto de microprocesadores de propósito general, que realizan fundamentalmente tareas de manipulación de datos (por ejemplo, procesadores de texto o bases de datos) que pueden tomar más o menos tiempo de procesamiento. A su vez, un DSP no es capaz de realizar otro tipo de tareas que otros microprocesadores pueden, pero es muy caro fabricar procesadores en todo tipo de tareas y por eso existen procesadores especializados como los DSPs. Otra cualidad es que pueden trabajar muestra a muestra o por bloques de muestras, cosa que afectará a la forma de trabajar con cada DSP, pero se hablará de esto más tarde.

Dicho esto, para procesar rápidamente, por ejemplo, un filtro FIR, se necesita algo más que operaciones de sumas y restas. Es necesario el acceso rápido a los N últimos datos de entrada y a los N coeficientes del filtro, para ello se utiliza un *buffer circular.*

Un buffer circular es una memoria temporal que se rellena de datos y sobrescribe los datos más antiguos, de esta forma no se sobrecarga la memoria. El buffer requiere de un puntero al inicio del buffer en memoria, un puntero de la posición actual, el valor del tamaño del buffer y el valor del ancho de la palabra. Los DSPs manejan muy

eficientemente los buffers circulares, de ahí que tengan tan alta velocidad de procesamiento.

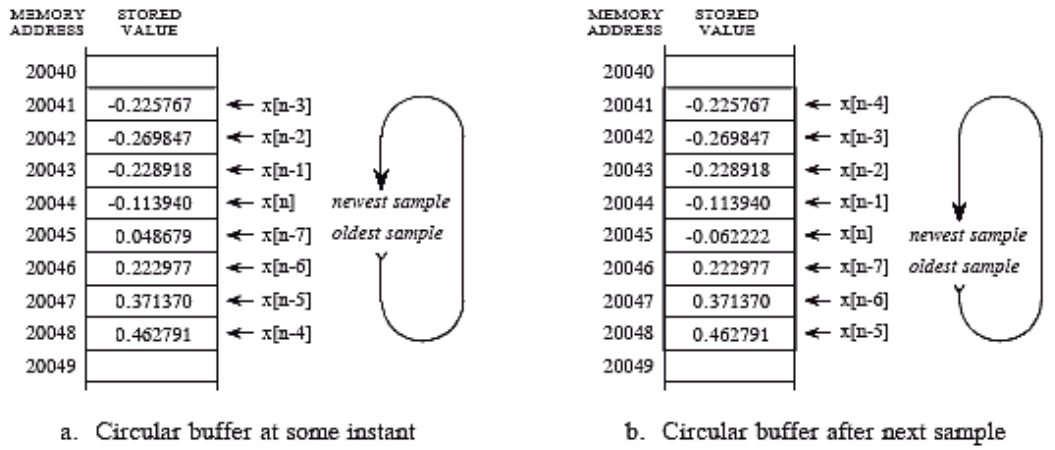

Figura 1: Funcionamiento del buffer circular. Fuente: http://www.dspguide.com/pdfbook.htm

Para este proyecto se han utilizado dos modelos de DSP, el Blackfin y el SHARC, ambos pertenecen a la empresa Analog Devices.

2.1.1 -. Blackfin (ADSP-BF706)

Este dispositivo pertenece a la familia de los ADSP-BF70x, y es uno de los modelos más comerciales de la compañía ya que incorpora la placa ensambladora. Consta de un procesador Blackfin+ con un rendimiento máximo de 400 MHz, también en modo bajo consumo a <100mV. Ofrece soporte MAC (Multiply-accumulate operation) dual 16-bit o single 32-bit, y también complex Max de 16-bit. Incorpora memoria en chip de 136KB L1 RAM con control de bit-multiparidad, 1Mbyte L2 SRAM con protección ECC, y 512 Kbyte L2 de ROM. Incluye conectores para periféricos USB 2.0 HS OTG, 2x canales minijack estéreo I/O, ePPI Video I/O, 2x SPORTs(w/I2S), 2xQuad-SPI/ 1xDual-SPI, I2C, 2xUART, SD/SDIO/MMC (4 bits).

Su uso se limita principalmente a pruebas de laboratorio ya que no incorpora una carcasa que lo proteja de interferencias externas, aun así, debido a sus características y a su reducido tamaño, es ideal para nuevos investigadores que desean adentrarse en la programación de DSPs.

Con respecto a su precio, en internet se pueden encontrar kits con el DSP y la placa ensambladora (ADZS-BF706-EZMINI) con un precio que ronda los 70€, a su vez también se puede conseguir el chip del microprocesador individualmente con un precio de 20€ aproximadamente.

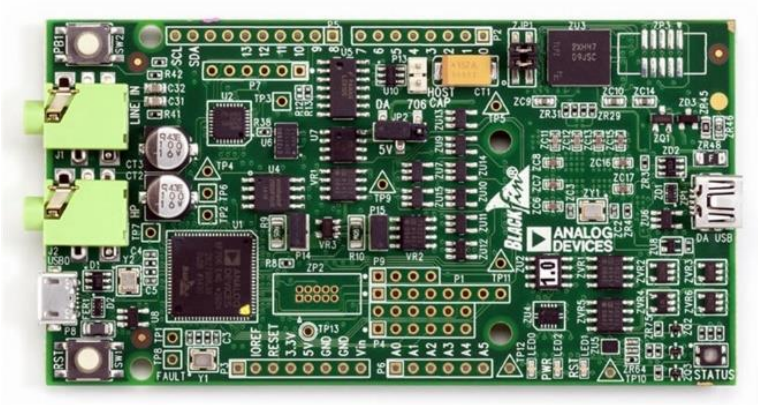

Figura 2: Placa ensambladora del ADSP-BF706. Fuente: www.analog.com

2.1.2 -. SHARC (ADSP-21489)

Este es un modelo más especializado de DSP y ofrece prestaciones mucho mayores en comparación con el modelo Blackfin. Entre sus características se encuentra, un rendimiento máximo de 450MHz, 5 Mbits de RAM en chip, aceleradores de FIR, IRR y FFT, ingeniería DMA, interfaz de memoria externa SDR SDRAM 16-bit, 8 puertos en serie (SPORTs), 2 puertos SPI, interfaz UART y Two-Wire, 16 canales PWM, 3 timers completos, etc….

El tamaño de este dispositivo es considerablemente mayor al Blackfin, y puede incorporar carcasa protectora. Debido a sus prestaciones es un dispositivo muy potente capaz de trabajar con 8 señales de entrada, siendo ideal para programas que requieran una gran carga computacional.

El precio del kit completo con placa ensambladora (ADZS-21489-EZLITE) se puede conseguir por internet por un precio que ronda los 500€, mientras que el microprocesador se puede conseguir individualmente por unos 27€ aproximadamente.

Figura 3: Placa ensambladora del ADSP-21489. Fuente: www.analog.com

2.2 -. Arquitectura de un DSP

Tomando como ejemplo un filtro FIR, uno de los procesos más típicos en el procesamiento de señales, es necesario que en un solo ciclo de reloj se realicen 4 accesos a memoria:

1. Tomar la instrucción.
2. Acceso a un dato, en línea de retardo.
3. Acceso a dato del coeficiente del filtro.
4. Desplazamiento de posición del dato en la línea de retardo.

Debido a esto el acceso a la memoria debe ser rápido y eficiente, y que permita acceder a más de un dato simultáneamente. Para este apartado se ha usado como referencia el documento *Electrónica Aplicada al Audio. Módulo 1: Procesadores Digitales de Señal (Primera Parte)* (Sogorb, 2016).

2.2.1 -. Arquitectura Harvard

La arquitectura Harvard es la arquitectura utilizada por la mayoría de DSPs, consta de dos espacios de memoria separados: uno para programa y otro para datos, y permite acceso a cada espacio por un sistema propio de buses separados.

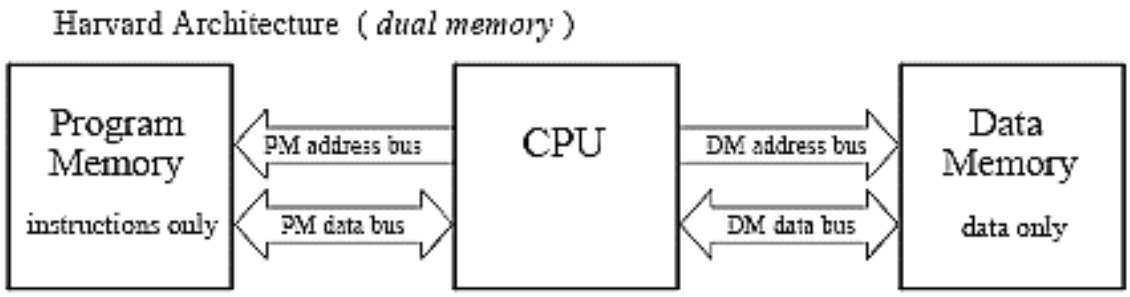

Figura 4: Esquema básico de arquitectura Harvard. Fuente: http://www.dspguide.com/pdfbook.htm

2.2.2 -. Arquitectura Super Harvard

La arquitectura Super Harvard es la que utiliza el SHARC, y la que le da su nombre. Añade nuevas funcionalidades para mejorar las prestaciones, como:

- Una memoria caché de instrucciones, que contiene unas 32 de las instrucciones de programa más recientes. De esta forma se reduce la carga en los buses de memoria y de datos, añadiendo operandos en las instrucciones (por ejemplo, los coeficientes de los filtros).
- Un controlador I/O, que libera al procesador de gestionar periféricos. A su vez, mediante técnicas DMA (Direct Memory Acces) permite que los flujos de datos se transfieran directamente a la memoria, sin la necesidad de pasar por el registro de la CPU.

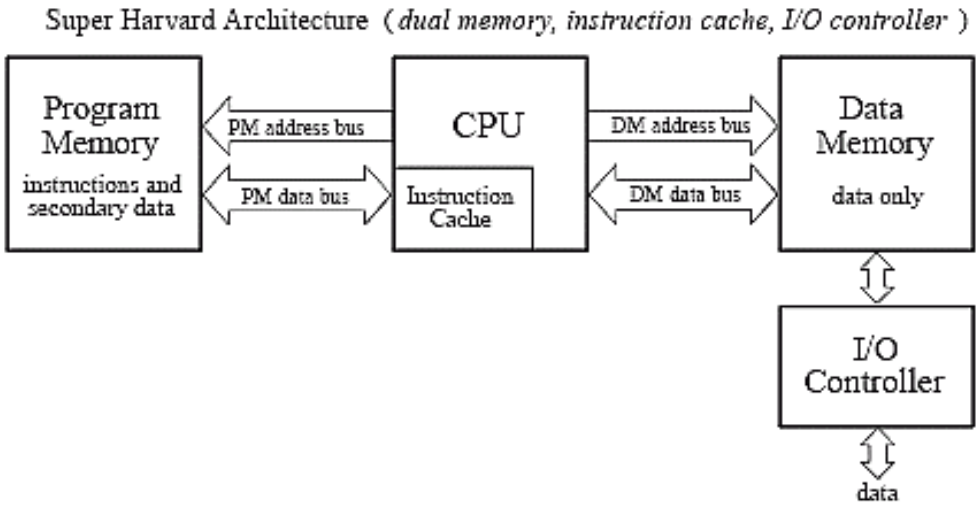

Figura 5: Esquema arquitectura Super Harvard. Fuente: http://www.dspguide.com/pdfbook.htm

2.3 -. Crosscore Embedded Studio

Para programar el microprocesador, Analog Devices proporciona su propio entorno de desarrollo integrado o IDE para las familias de procesadores Blackfin, SHARC y ARM. Se puede descargar desde su web, siendo necesario adquirir una licencia para su uso. En la realización de este proyecto se ha utilizado la licencia educativa proporcionada por la UPV. Este IDE está basado en Eclipse, una plataforma de software compuesta por un conjunto de herramientas de programación de código abierto multiplataforma, típicamente usada para desarrollar IDE en una gran variedad de lenguajes de programación.

Crosscore es compatible con los sistemas operativos Windows y Linux, y ofrece herramientas como edición de idioma de montaje, generador de códigos, soporte de depuración y ensamblador, además de gran cantidad de librerías y ejemplos.

El lenguaje de programación de Crosscore está basado en C/C++, y existen dos tipos de ficheros principales con los que se trabaja:

- Cabecera (.h): Fichero donde se realiza la definición de constantes y la declaración de variables locales o de prototipos, además de la inicialización de librerías.

- Programa (.c): En este fichero se implementa el código del programa, se inicializan las variables y se implementan las funciones. Cada función se ejecuta como un bucle que se ejecuta mientras la placa esté en funcionamiento, siendo la principal característica de la programación a tiempo real.

2.4 -. MATLAB

Para la realización del prototipo se ha utilizado la herramienta de software matemático MATLAB. Este entorno de desarrollo integrado (IDE) utiliza un lenguaje de programación propio, lenguaje M. Es compatible para las plataformas Windows, Unix, Mac OS X y GNU/Linux.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, representación de datos y funciones, implementación de algoritmos, creación de interfaces usuario (GUI) y comunicación con programas en otros lenguajes y con otros dispositivos hardware.

Es un software muy utilizado en universidades y centros de investigación y desarrollo, y en los últimos años ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal (DSP) o crear código VHDL.

Las aplicaciones de MATLAB se desarrollan con un lenguaje de programación propio. Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de *script* (archivo .m). También permite operaciones de vectores y matrices, funciones, cálculo lambda, y programación orientada a objetos.

Como se puede observar, MATLAB es una herramienta muy completa que permite trabajar en un entorno controlado y con gran cantidad de funciones, cosa que la hace perfecta para la realización del prototipo y de las pruebas necesarias, además de la posibilidad de representar gráficamente los resultados, facilitando el análisis de su funcionamiento.

3 -. Fundamentos teóricos

En este apartado se pretende explicar de forma clara los fundamentos teóricos sobre los que se basa este proyecto, de forma que sea posible la comprensión total de todas sus partes. Desde el punto 3.1 hasta el 3.3 se explica la problemática del feedback acústico y los distintos métodos para su control basándose en *"Fifty years of acoustic feedback control: state of the art and future challenges"* (van Waterchoot y Moonen, 2011, pp. 289-301), el punto 3.4 explica el uso de la transformada de Hilbert como herramienta matemática (Wikipedia, 2017), y el último punto, describe las características de los distintos formatos de programación utilizados (Sogorb, 2016).

3.1 -. El problema del feedback acústico

En la siguiente figura se muestra la distribución típica de un sistema de PA en un escenario.

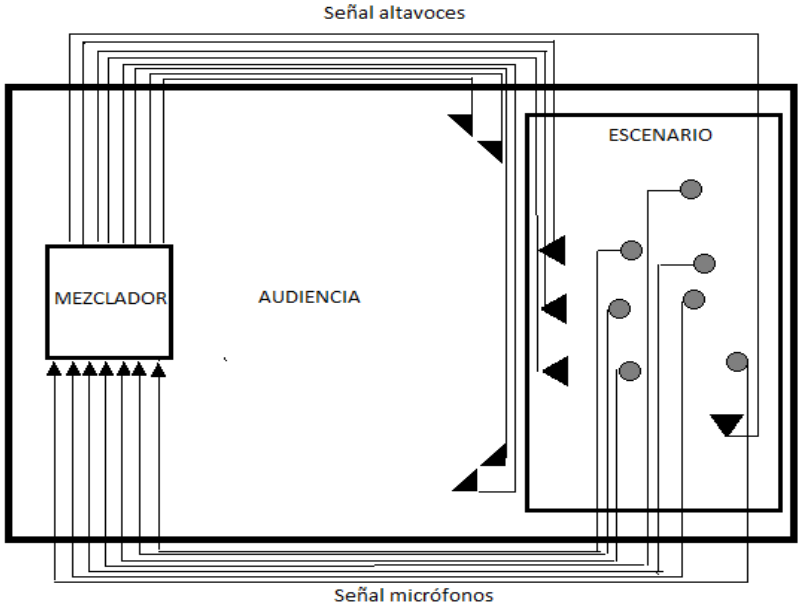

Figura 6: Esquema de la distribución de un sistema de sonorización. Fuente: Propia.

Múltiples micrófonos están orientados para captar distintas fuentes de sonido de interés. La señal de audio captada por los micrófonos es dirigida a un mezclador, donde

se realizada el procesamiento necesario (amplificación, efectos, compresión, etc.) para después enviarla a los altavoces, que suelen estar organizados en grupos de altavoces que emiten señales distintas, ya que no es necesario emitir la misma señal hacia los músicos que hacia el público. Normalmente, los micrófonos y los altavoces suelen estar orientados de forma que, teniendo en cuenta su directividad, la señal emitida por los altavoces no sea recibida directamente por los micrófonos. Aun así, es inevitable que debido a las reflexiones producidas en la sala no se produzca una realimentación. Estas reflexiones constituyen un acoplamiento acústico indirecto entre los altavoces y los micrófonos.

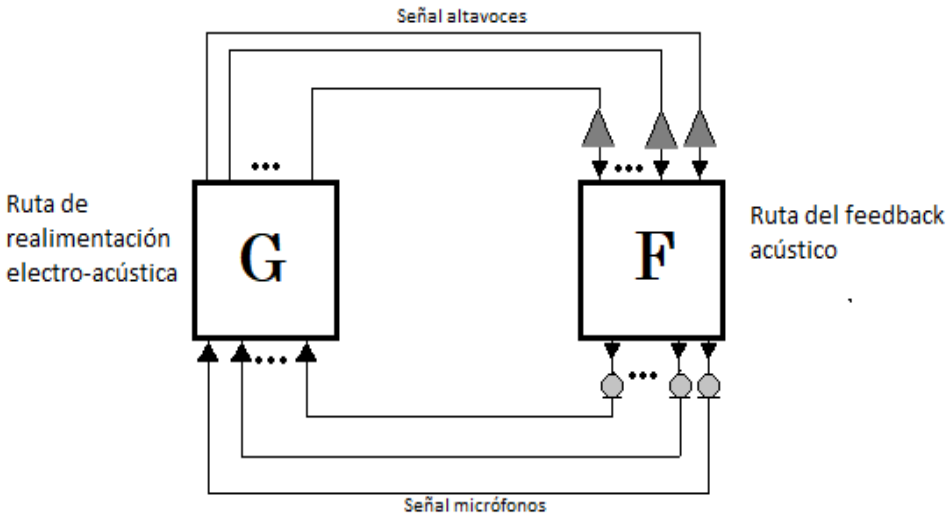

Figura 7: Esquema discreto del bucle cerrado. Fuente: Propia.

Otra manera de representar el sistema de PA es con un modelo en tiempo discreto como se muestra en la figura 7. De esta forma se pueden diferenciar dos rutas que recorre la señal de audio: la ruta del feedback acústico, que incluye las reflexiones de la señal en la sala hasta que es realimentada en los micrófonos, y la ruta electro-acústica, que comprende desde que la señal es captada por los micrófonos, procesada y amplificada para su retorno a los altavoces.

Esta representación está basada en un sistema multicanal, con múltiples micrófonos y altavoces. Para la realización del proyecto se tiene en cuenta un modelo de un solo canal, ya que todos los métodos de control de feedback propuestos hasta la fecha son aplicables a solo un canal. Esto tiene sentido si se piensa que, en un caso real, el acople se suele producir en el micrófono del cantante, que no suele estar estático y es más susceptible a realimentaciones. Así pues, el modelo del sistema de un solo canal

quedaría como en la figura 8, donde se pueden observar las dos rutas del bucle cerrado que recorre la señal y un bloque de procesamiento con la función de transferencia de control del feedback. Hasta ahora se podría suponer que:

- El altavoz tiene respuesta lineal y plana.
- El micrófono tiene respuesta lineal y plana.
- La ruta de amplificación tiene respuesta lineal y plana (suponiendo que no se aplica ningún procesamiento con respuesta infinita)
- La ruta de feedback tiene respuesta lineal pero no plana.

Sabiendo esto, si se aplica el criterio de estabilidad de Nyquist, que dice: si existe una frecuencia radial $\omega = 2\pi(f/fs)$ para la cual

$$\begin{cases} \left| G(\omega,t)F(\omega,t) \right| \geq 1 \\ \angle G(\omega,t)F(\omega,t) = n2\pi, \qquad n \in Z \end{cases}$$

entonces el sistema de bucle cerrado es inestable. Si el sistema inestable es excitado en un frecuencia crítica ω, se producirá una oscilación en esa frecuencia, provocando un acoplamiento o *howling*.

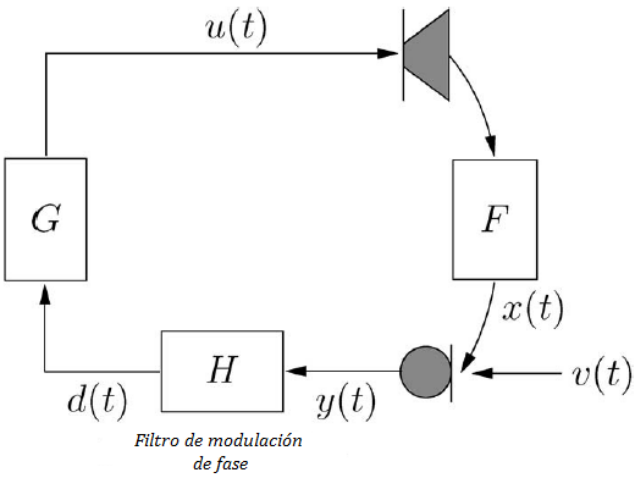

Figura 8: Esquema discreto de un solo canal. Fuente: (van Waterschoot y Moonen, 2011)

3.2 -. Métodos de control automático del feedback acústico.

En este apartado se habla de los diferentes métodos de control del feedback acústicos en el contexto de un sistema de un solo canal, con el fin de explicar brevemente las diferentes técnicas utilizadas, al igual que las ventajas y desventajas de cada uno. Los métodos automáticos de control del feedback se pueden clasificar en cuatro clases.

3.2.1 -. Métodos de modulación de fase (PM)

Una de las primeras propuestas para el control del feedback acústico consiste en el desplazamiento de frecuencia o FS (*Frequency shifting*) de la señal del micrófono antes de ser amplificada y enviada a los altavoces. Este enfoque se le puede atribuir en su gran medida a M.R. Shroeder, quien publicó varios documentos sobre el tema a principio de los 60.

Se ha comprobado en diversos experimentos que, la distancia frecuencial media entre dos picos de magnitud en la respuesta de una sala, es aproximadamente de 10Hz, por lo tanto, el valor óptimo de FS se espera que sea alrededor de 5 Hz.

Existen otras técnicas, como el uso de modulación de fase en la ruta electro-acústica con el objetivo de evitar la condición de fase en el criterio de Nyquist, u otras como la modulación de amplitud (AM) o modulación del retardo (Delay Modulation, DM).

La realización de este proyecto se basa principalmente en la técnica de modulación de la frecuencia (FS) usando un filtro FIR de Hilbert, aunque también se han probado alguna de las otras. Se ha elegido esta técnica debido a su robustez y sencilla implementación comparada con los siguientes métodos, aunque cuenta con algunas desventajas de las que se hablará más adelante, donde se explica esta técnica con más detenimiento.

3.2.2 -. Métodos de reducción de ganancia.

El enfoque más claro para el control del feedback es automatizar las acciones que un técnico haría para prevenir o eliminar el acople en sistemas de refuerzo de sonido. Estas acciones normalmente consisten en reducir la ganancia de la ruta electro-acústica mediante ecualización, con el fin de alejar el sistema de la condición de módulo del criterio de Nyquist.

Dependiendo del ancho de la banda de frecuencias en la cual se reduce ganancia, se pueden distinguir entre tres métodos de reducción de ganancia:

1) Métodos de control automático de ganancia(AGC), los primeros de este tipo en estudiarse. Si se detecta acople en alguna frecuencia se reduce la ganancia de toda la banda de frecuencias, y después de un intervalo de tiempo la ganancia se restaura a su estado inicial.
   Con los años se han propuesto mejoras que permiten realizar la reducción de ganancia en sub bandas de frecuencia más centradas en donde se produce el acople. La mayor fortaleza de este método es su fiabilidad, ya que si la ganancia se reduce lo suficiente se asegura que el sistema sea estable. Muchos otros métodos utilizan esta técnica como último recurso en caso de que todos los demás fallen.

2) Métodos de ecualización automática(AEQ), derivan de la mejora del método anterior. Si se detecta acople, la reducción de ganancia se aplica solo a las sub bandas donde se produce.

3) Métodos NHS, se reduce la ganancia en bandas estrechas de frecuencias (filtros noch) donde se produce el acople, frecuencias donde el bucle de ganancia es cercano a la unidad.
   Se pueden dividir en dos categorías: métodos NHS de una etapa o de dos etapas, dependiendo de si la detección del acople y filtrado noch se ejecutan juntos o por separado.
   Los de dos etapas son probablemente los más populares de este tipo de método. Usando la transformada rápida de Fourier (FFT) se realiza un análisis frecuencial de la señal del micrófono, donde los componentes candidatos al acople son detectados usando un algoritmo de detección de pico. La energía de los candidatos al acople es comparada a un umbral absoluto de energía, la media

de la energía de la señal y la de los (sub)harmónicos, para detectar si se produce acoplamiento.

3.2.3 -. Métodos de filtrado espacial.

La meta de estos métodos es alterar la respuesta del bucle $G(\omega, t)F(\omega, t)$ del sistema usando arrays de micrófonos y altavoces en los cuales la señal recibida/transmitida se procesa mediante los llamados *beamforming filters*, que se podría traducir como filtros formadores de haces. El objetivo general es un array de micrófonos en los que el lóbulo principal de captación esté dirigido a la fuente mientras se tiene un nulo en la dirección del altavoz, o lo mismo con un array de altavoces, donde tenga máxima directividad hacia la audiencia y un nulo en la dirección de los micrófonos.

Uno de los métodos de filtrado espacial propuestos es el uso de un cancelador de feedback adaptativo (AFC) junto al array de micrófonos, así se puede alimentar la señal de los micrófonos con la obtenida del AFC, donde se predice y se substrae la componente de feedback de la señal. De esta forma se consigue reducir la influencia de la señal con feedback en el algoritmo formador de haces.

Estos métodos son muy complicados de implementar debido a que es muy difícil obtener un array de micrófonos con un nulo en la dirección de los altavoces ya que la señal no proporciona información sobre la localización de estos. Otra cosa a tener en cuenta son las restricciones espaciales que ofrece cada sala, añadiendo complicidad a este tipo de algoritmos.

Aun así, como se puede observar en las tablas *Performance Measures for Comparative PFC, NHS, and AFC Simulations: Speech Source Signal* y *Performance Measures for Comparative PFC, NHS, and AFC Simulations: Audio Source Signal* (van Waterschoot y Moonen, 2011, pp. 321-322) la calidad de sonido obtenida por estos métodos es la más alta en comparación con los otros.

3.2.4 -. Métodos de modelado de sala.

Estos métodos establecen un modelo de la ruta del feedback acústico ya sea offline, durante la inicialización del sistema de refuerzo sonoro, u online, durante el funcionamiento del sistema de refuerzo. Se pueden distinguir dos métodos de modelado

de sala, dependiendo de cómo se aplique posteriormente el modelo para el control del feedback acústico.

Uno de ellos es el AFC, donde se utiliza la ruta del feedback acústico para predecir la componente de feedback en la señal de micrófono (la parte de la señal de micrófono que proviene del altavoz a través del acoplamiento acústico). La señal predicha se substrae de la señal del micrófono, dando como resultado una señal con compensación de feedback, que de hecho es una estimación de la componente de la fuente en la señal del micro. Si se consigue un modelo preciso de la ruta de feedback, entonces el método AFC puede conseguir una eliminación casi total del acoplamiento acústico.

Por otro lado, se puede obtener e identificar un modelo inverso de la ruta de feedback, que luego puede ser insertado en el bucle cerrado de la señal para obtener una ecualización optima de la señal del micrófono. A este enfoque se le denomina como filtrado inverso adaptativo (AIF).

Basándose en estos dos métodos se han ido desarrollando mejoras utilizando diferentes técnicas de filtrado, implementando una combinación de ambos o simplemente realizando una automatización de un gran número de ecualizadores en la ruta electro-acústica a partir de un modelo offline. La ventaja de estos métodos de modelado de sala es su robustez, aun así, su complicidad reside en la precisión con la que se pueda obtener el modelo de la ruta del feedback acústico.

3.3 -. Control del feedback mediante modulación de la fase (PFC)

Este apartado trata con detenimiento el método para control del feedback que se ha implementado en este proyecto, hablando de sus ventajas e inconvenientes y distintas técnicas que pueden ser utilizadas. Se ha optado por este método debido a que el algoritmo es robusto, con una implementación no muy complicada, dando como resultado una buena calidad de audio sin ofrecer demasiada carga computacional para los DSP utilizados.

3.3.1 -. Concepto

Como ya se ha comentado brevemente en el apartado anterior sobre los métodos de control del feedback, el objetivo del PFC es controlar la fase de la señal del micrófono

de manera que, cada componente frecuencial en la señal de feedback, tenga una fase distinta cada vez que vuelva a ser captada por el micrófono después de haber recorrido un ciclo a través del bucle cerrado de señal. De esta forma se puede garantizar que no ocurra la condición de fase del criterio de Nyquist en dos instantes consecutivos, mejorando la estabilidad del sistema.

Existen cuatros técnicas PM usadas en el contexto del control del feedback acústico:

- Modulación senoidal de fase (Sinusoidal PM): La respuesta de filtro tiene una respuesta en frecuencia caracterizada por una portadora independiente de la frecuencia y una respuesta de banda lateral que se corresponde con las funciones de Bessel de primer tipo y orden n

$$H(\omega, t) = e^{j\beta sin\omega_m t}$$

$$\mathcal{H}(n) = J_n(\beta), \qquad n = 0, \dots, N-1$$

- Modulación senoidal de frecuencia (FM senoidal): El efecto producido por un filtro FM con una frecuencia de modulación $f_m = \omega_m(f_s/2\pi)$ y un índice de modulación Δ_f, se dice que es idéntico al efecto que tiene un filtro PM con la misma frecuencia de modulación f_m y un índice de modulación $\beta = \Delta_f / f_m$.

- Desplazamiento de frecuencia (Frequency Shift FS): Un mecanismo FS también se puede ver como un sistema no-lineal e invariante en el tiempo o como un sistema LPTV (linear periodically time-varying). De la última interpretación, se puede demostrar que una operación FS con un desplazamiento de frecuencia de $f_m = \omega_m(f_s/2\pi)$ Hz corresponde a una operación PM con una función de fase que se incrementa linealmente con el tiempo.

$$H(\omega, t) = e^{j\omega_m t}$$

- Modulación senoidal de retardo (Sinusoidal DM): Un filtro DM senoidal varia el retardo de la señal de entrada senoidalmente alrededor de un tiempo offset τ_0, con una desviación máxima del tiempo de retardo Δ_τ y una frecuencia de modulación ω_m. Siendo esta su respuesta en frecuencia:

$$H(\omega, t) = e^{-j\omega(\tau_0 + \Delta_\tau sin\omega_m t)}$$

3.3.2 -. Realización

Los filtros PM, FM y FS operan normalmente sobre la llamada representación analítica de la señal de micrófono $y(t)$. En tiempo discreto la señal analítica queda representada como $y_a(t) = y(t) + j\hat{y}(t)$, donde $\hat{y}(t)$ es la transformada de Hilbert de $y(t)$.

La señal de salida de los filtros PM, FM y FS se calcula modulando $y_a(t)$ con la respuesta de la función de transferencia de la transformada de Hilbert. De esta forma se obtiene una señal $d(t)$ cuya parte real es la señal original y la parte imaginaria es la señal original desfasada 90º, es decir, la transformada de Hilbert de la señal.

Obteniendo $d(t)$, es posible implementar el algoritmo antifeedback utilizando las técnicas de modulación PM, FM y FS. Esto lleva a,

$$d(t) = y(t)\cos\varphi(t) - \hat{y}(t)\sin\varphi(t) \qquad (1)$$

$$\varphi(t) = \begin{cases} \beta\, sin\omega_m t, & para\ PM \\ \dfrac{\Delta_f}{fm} sin\omega_m t, & para\ FM \\ \omega_m t, & para\ FS \end{cases}$$

La modulación DM se puede realizar directamente operando sobre la señal del micrófono $y(t)$, que luego se introduce en una línea de retardo de longitud variable. Este tipo de líneas de retardo también es utilizado en efectos de audio digitales como vibrato, flanging o chorus.

Para terminar, las mayores desventajas que ofrece este método son: primero, el incremento de ganancia que se consigue es limitado ya que aunque se elimine el acoplamiento de frecuencias, si se aumenta mucho la ganancia la frecuencia moduladora pasa a ser audible; segundo, insertar un filtro PM en la ruta electro-acústica inevitablemente implica distorsión de la señal, cosa que no es bueno para aplicaciones de audio; tercero, en sistemas multicanal se ha comprobado que la mejora de estabilidad obtenida con PFC se reduce conforme aumenta el número de canales.

3.4 -. Transformada de Hilbert

Cuando se habla de la transformada de Hilbert se considera un sistema que no distorsiona la amplitud de la señal de entrada, pero que si distorsiona la componente de fase, de tal manera que produce un desfase de $\pi/2$ en todas las frecuencias de la señal de entrada. Estamos hablando de un sistema LTI (Linear time-invariant) cuya respuesta al impulso es $h(t) = 1/\pi t$.

Otra forma de definirla, donde $s(t)$ es la señal de entrada:

$$\hat{s}(t) = \mathcal{H}\{s\}(t) = (h * s)(t) = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{s(\tau)}{t-\tau} d\tau$$

Utilizando $\hat{s}(t)$, se puede obtener la señal analítica de $s(t)$ como:

$$s_a = s(t) + j\hat{s}(t)$$

Otra cualidad de la transformada de Hilbert es que posee una respuesta en frecuencia dada por la transformada de Fourier. La transformada de Hilbert actúa como un operador multiplicador (Duoandikoetxea, 2000, Capitulo 3), siendo el multiplicador de H, $\sigma_H(\omega) = -j\ sgn(\omega)$ donde sgn es el signo de la función. Entonces,

$$\mathcal{F}(H(s))(\omega) = (-j\ sgn(\omega)) \cdot \mathcal{F}\{s\}(\omega)$$

Donde $\mathcal{F}$ denota la transformada de Fourier. Siendo $sgn(x) = sgn(2\pi x)$, se deduce que este resultado se aplica a las tres definiciones de $\mathcal{F}$.

Usando la fórmula de Euler,

$$\sigma_H(\omega) = \begin{cases} j = e^{+\frac{j\pi}{2}}, & para\ \omega < 0 \\ 0, & para\ \omega = 0 \\ -j = e^{-\frac{j\pi}{2}}, & para\ \omega > 0 \end{cases}$$

Por lo tanto, la transformada de Hilbert produce el efecto de desplazar la fase de las componentes de frecuencias negativas de $s(t)$ +90º ($\pi/2$) y las componentes de frecuencias positivas -90º.

3.5 -. Formatos numéricos en la programación

Este apartado, obtenido de (Sogorb, T. C., 2016), se ha añadido debido a que ha sido uno de los rompecabezas a la hora de implementar el algoritmo en el DSP Blackfin. Por eso se ha decidido explicar brevemente los dos formatos de representación binaria que se utilizan a la hora de programar.

3.5.1 -. Coma fija

En la representación de números en coma fija, que representa valores en complemento a 2 (Ca2), la posición del punto decimal está predeterminada. El ancho de palabra n es normalmente igual a 16 o 32 bit. Si por ejemplo, tenemos n= 16, se obtienen 2^n=2^{16}= 65.535 valores.

Tomando como ejemplo n = 16, se pueden representar números de cuatro formas:

- Enteros sin signo: la coma se encuentra a la izquierda del bit más significativo (MSB), es decir, no hay parte decimal. Se representan enteros entre el 0 y el 65.535.
- Enteros con signo: la coma decimal se encuentra a la derecha del MSB. Se representan en complemento a 2, enteros entre el -2^{15} y el $2^{15}-1$, es decir , entre el -32.768 y el +32.767. El signo depende de si el MSB está a 0 (positivo) o 1 (negativo).
- Fracciones sin signo: la coma decimal está situada a la izquierda del MSB. Permite representar números entre el 0 y el $1-2^{-15} \approx 1$(0.9999847412).
- Fracciones con signo: la coma decimal se encuentra a la derecha del MSB. Permite representar números entre el -1 y el $1-2^{-15} \approx 1$(0.9999847412).

3.5.2 -. Coma flotante

La representación de números en coma flotante está basada en la notación científica normalizada:

$$N = -1^{Signo} * 1, Mantisa * Base^{Exponente-127}$$

Donde

- La Base es 2.
- El MSB es el bit de Signo
- El Exponente se representa en exceso de 127 (se la ha sumado 127).
- El exponente 255 (todo unos) se utiliza para expresar
    - Si la Mantisa ≠ 0 representa NAN (Not A Number): variables no inicializadas, operaciones no válidas.
    - Si Mantisa = 0 representa ∞ (con el bit de signo -∞ o +∞).
    - El cero se representa con Mantisa = 0 y Exponente = 0. Con el bit de signo se puede representar +0 y -0.
    - Rango de representación : $[-2^{127}, 2^{127}]$.

4 -. Diseño e implementación

Para diseñar, desarrollar e implementar este proyecto, se ha tenido que pasar por diversas etapas hasta conseguir el resultado final. Principalmente el desarrollo de este proyecto se puede dividir en dos etapas, la etapa de desarrollo del algoritmo prototipo, y la etapa de implementación en DSP del algoritmo. Aun así, antes de empezar con cada una de las etapas principales, ha hecho falta dedicar tiempo al aprendizaje de los distintos programas, teoría necesaria y formas de programación.

Se comenzó por investigar acerca de CrossCore y del DSP Blackfin, para obtener una idea de sus posibilidades y las características propias de la programación a tiempo real. Tanto el software y hardware ha sido proporcionado por la UPV, por lo tanto, no ha habido necesidad de realizar una búsqueda de cuales se iban a utilizar o cual era su coste. Así pues, una vez obtenido el DSP y el software, se empezó a aprender el lenguaje de CrossCore mediante la realización de programas básicos encontrados en libros en la web, pero sobretodo mediante la realización de las prácticas impartidas en

la asignatura de Electrónica Aplicada al Audio (EAA). Siguiendo la estructura de la asignatura, realizando pequeños programas de efectos de audio y trabajando con líneas de retardo, fue posible coger agilidad, práctica y soltura a la vez que se iba adentrando más en el lenguaje.

Cuando se consideró que el nivel adquirido era suficiente, se empezó con el estudio de los algoritmos anti-feedback, su comprensión y la forma de implementarlos mediante el lenguaje de programación. Tras el estudio del documento de van Waterschoot y Moonen, citados anteriormente, y por la recomendación del tutor del proyecto, se eligió el método de control de feedback acústico basado en el PFC, más específicamente se tomó como referencia el método de FS, con el cual se ha trabajado hasta la implementación final.

Llegados a este punto se observó que había distintas formas de programar el algoritmo, sobre todo a la hora de realizar las transformada de Hilbert, que puede ser obtenida mediante diversos métodos. Así, teniendo métodos distintos, era necesario elegir el que mejor rendimiento y calidad era capaz de obtener, para ello se llegó a la conclusión de que CrossCore no era la plataforma más adecuada para esta tarea, ya que con él no puedes obtener datos objetivos sobre el funcionamiento y rendimiento del algoritmo implementado.

4.1 -. Diseño del prototipo usando MATLAB

Como ya se ha mencionado anteriormente, el programa con el que se va a implementar el código que se introducirá en el DSP es el CrossCore, pero antes de implementar el código en esa plataforma especializada para programación en tiempo real, se decidió realizar un prototipo en un entorno controlado, en el que fuera posible realizar todas las pruebas necesarias para seleccionar el modelo de algoritmo que mejor se adapte después al DSP según sus limitaciones computacionales, comentadas anteriormente.

El programa elegido es Matlab, una de las herramientas matemáticas más potentes. Gracias a Matlab se pueden generar diferentes tipos de señales y procesarlas con los diferentes algoritmos que se vayan a usar, a la vez que se obtienen resultados gráficos y numéricos que facilitan la tarea de análisis de los distintos algoritmos. El único inconveniente es que no se trata de programación a tiempo real por lo que el código tiene que ser adaptado posteriormente para que su uso sea apto en el DSP.

Después de estudiar cómo adaptar la transformada de Hilbert a tiempo discreto y teniendo en mente el documento (van Waterschoot y Moonen, 2011) se implementaron 3 funciones, mediante:

- Filtro FIR
- FFT
- Filtro FIR basado en FFT

Con cada una de estas funciones fue posible obtener la transformada de Hilbert, pero no en todas se obtuvo la misma calidad en la señal de salida, ni la misma facilidad de implementación.

Para realizar las pruebas más rápidamente se creó un documento *.m* con todas las funciones, en el cual era posible cambiar el valor de las variables. Este documento se encuentra en el anexo, y sigue la estructura siguiente.

4.1.1 -. Transformada de Hilbert ideal

Con el fin de observar el comportamiento de la transformada de Hilbert y de la ecuación de anti-feedback (1), se utilizó la función *hilbert()* con la que cuenta Matlab en su librería. Esta función calcula la transformada de una señal de entrada dada x_n con unos parámetros ideales. El resultado es una señal compleja $x_r + i * x_i$, donde la parte real es la señal original de entrada y la parte imaginaria es la señal original desplazada 90°, la transformada de Hilbert.

Una vez se comprobó que las dos señales obtenidas eran correctas, se decidió utilizar esta función como referencia para las posteriores. De esta forma, calculando el Error Cuadrático Medio (RMS) entre esta señal ideal y las resultantes en los otros métodos, era posible obtener un valor de error que luego sirvió para elegir el método a implementar.

4.1.2 -. Transformada de Hilbert mediante Filtro FIR

Para esta función, Matlab cuenta con diversas funciones capaces de implementar el filtro FIR. La función utilizada fue *fdesign.hilbert*, que permite introducir dos parámetros, el número de coeficientes y el ancho de banda de transición (TW).

Esta función diseña el filtro con los parámetros introducidos y guarda los coeficientes, que se pueden obtener añadiendo *.Numerator* a la variable donde se guarda el filtro.

Por ejemplo:

```
d = fdesign.hilbert('N,TW', 50 , 0.1);

Hd = design(d, 'equiripple');

Coefs = Hd.Numerator;
```

Ahora los coeficientes de un filtro de Hilbert de 50 coeficientes y un ancho de banda de transición de 0.1 se encuentran guardados en la variable Coefs.

Como era necesario obtener un archivo externo con el valor de los coeficientes, se utilizó la función *save*, que permite crear un fichero .txt con los valores de los coeficientes en formato ASCII.

Ejemplo:

```
save('Coef50.txt', 'Coefs', '-ascii');
```

A la hora de diseñar el filtro FIR, se tuvo que tener en cuenta un factor muy importante, el retardo generado por el filtro. Siempre que se implementa un filtro FIR, la señal de salida se encuentra retarda N/2 muestras, donde N es el número de coeficientes del filtro. La corrección es necesaria, sobretodo en Matlab, ya que las dimensiones de los vectores tienen que ser las mismas si se desea operar con ellos o representarlos gráficamente. Para ello, se tuvo que eliminar las N/2 ultimas muestras de la señal original y del vector de tiempo, y las N/2 primeras muestras de la señal procesada (transformada).

Ejemplo:

```
retardo = floor(length(Hd.Numerator)/2);

s_FIR = x (1: end-retardo) + 1j * s_hilbert_fir (retardo+1: end);

t_fir = t (1: end-retardo);
```

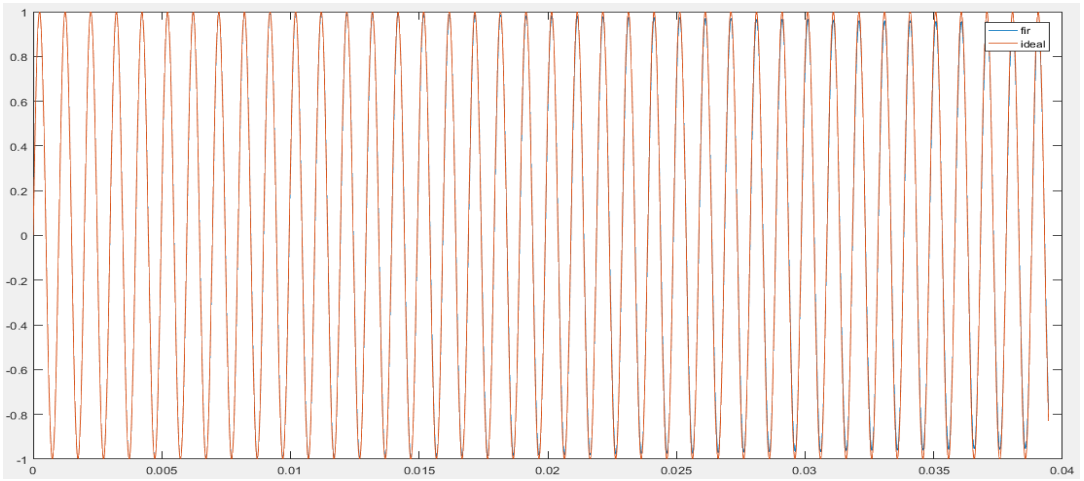

Figura 9: Comparación de la señal obtenida del filtro FIR con la señal ideal. Fuente: Captura propia.

4.1.3 -. Transformada de Hilbert mediante FFT

La función *hilbert(xr,n)* que incorpora Matlab es la misma que la utilizada para calcular la transformada ideal, pero añade una variable *n* que permite elegir el número de puntos de la FFT que realiza la transformada.

Sin embargo, no se utilizó esta función, debido a que el objetivo era implementar el algoritmo que llamaba la función, con el fin de adaptarlo más tarde a lenguaje de DSP. Los pasos del algoritmo se encontraron en *Discrete-time analytic signal using Hilbert transform.* (MathWorks, 2017).

El algoritmo sigue cuatro pasos:

1. Se calcula la FFT de la secuencia de entrada, guardando el resultado en un vector *x*.
2. Se crea un vector *h* cuyos valores serán:

    - 1 para i = 1, (n/2) +1
    - 2 para i = 2, 3, …, (n/2)
    - 0 para i = (n/2) +2, …,

3. Se calcula el producto de los vectores componente a componente de *x* y *h*.
4. Se calcula la inversa de la FFT de la secuencia obtenida en el paso 3 y devuelve los n primeros elementos del resultado.

Como dice el paso 4, solo devuelve los n primeros elementos del resultado. Eso significaba que la señal obtenida era un fragmento de las n primeras muestras procesadas de la señal original. Por lo tanto, si se deseaba obtener la FFT de toda la secuencia era necesario segmentar la secuencia de entrada en bloques de n elementos, procesarlos y juntarlos en un vector final que contuviera el resultado de FFT de toda la secuencia. Para ello solo hizo falta generar un bucle que segmentara la señal en paquetes y realizase los pasos del 1 al 4 en cada paquete, almacenándolos en la posición correspondiente del vector final.

Una vez implementado y después de haber realizada las primeras pruebas de test, se observó que este algoritmo tenía fundamentalmente dos problemas. El primero era que para obtener un buen valor de RMS era necesario un número de FFT alto, cosa que no supondría ningún problema para el DSP SHARC, pero si para el Blackfin, que trabaja con bloques de 16 o 32 muestras.

El segundo problema apareció cuando se observaba el gráfico de la señal ideal junto a la señal con FFT. En la segunda, se podía ver que cada cierto número de muestras, aparecía un pico que no correspondía con la señal ideal. Estos picos aparecían cada n muestras, correspondiendo con la transición entre un segmento y el siguiente, cosa que se traducía en un chasquido a la hora de reproducir la señal.

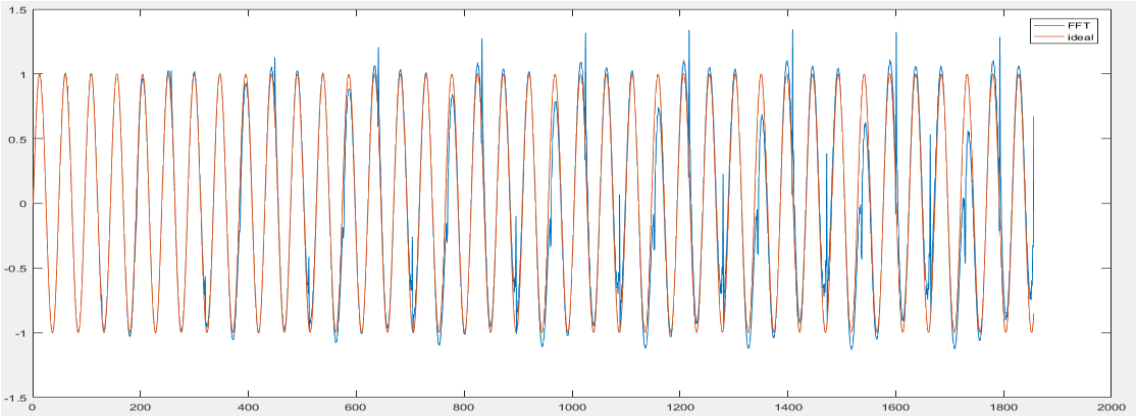

Figura 10: Comparación de la señal obtenida con FFT y la señal ideal. Fuente: Captura propia.

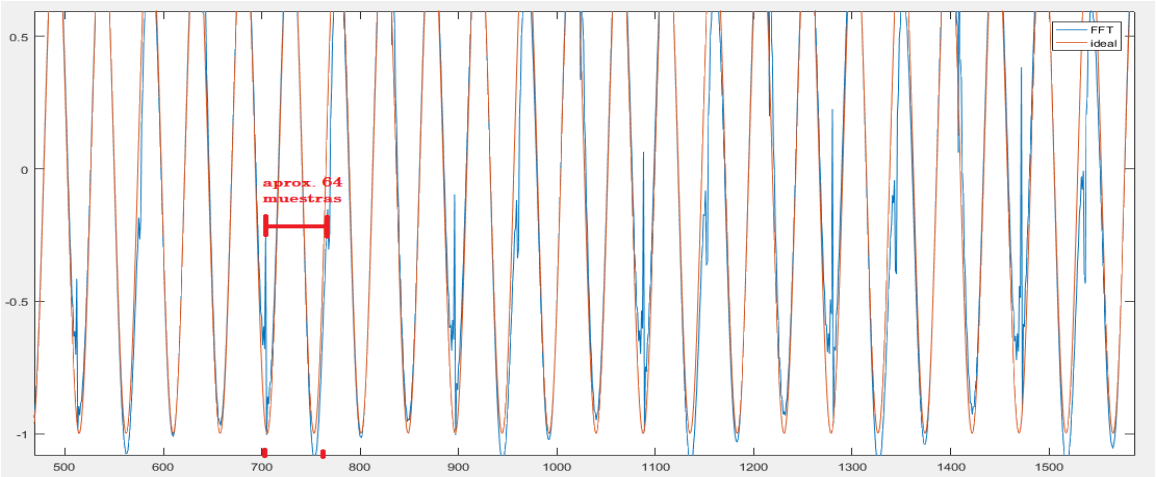

Figura 11: Error producido por la técnica de FFT. Fuente: Captura propia.

Investigando se encontró otro método de FFT con solapamiento, que ponía solución al último problema, y del cual se hablará en el siguiente punto.

4.1.4 -. Transformada de Hilbert mediante filtro FIR basado en FFT

Esta técnica es muy parecida a la anterior. La señal es fragmentada en bloques de n muestras, cada segmento se transforma al dominio frecuencial mediante FFT y se añaden M-1 ceros al final de cada uno, donde M es el número de coeficientes del filtro utilizado. Cada segmento se convoluciona con la FFT de los coeficientes del filtro (su respuesta en frecuencia) y después se realizada la IFFT al mismo, dando como resultado un nuevo segmento de longitud n + M – 1. Cada nuevo segmento resultante se suma con el siguiente, solapando las M-1 muestras finales del primero con las primeras respectivas del segundo. Al solapar cada segmento con el siguiente se obtiene una señal final procesada en la cual se soluciona el problema de método de FFT. En la figura 12 es posible visualizar mejor el procedimiento. *Computing the Discrete –Time Analitic Signal via FFT* (Marple, 1999).

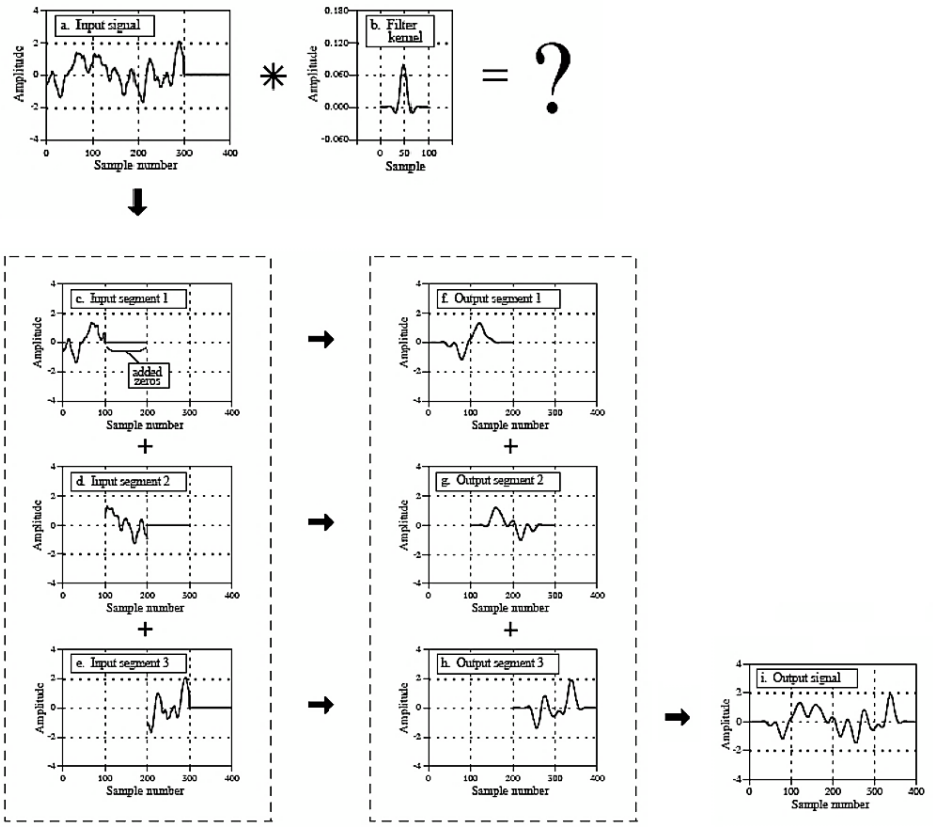

Figura 12: Ejemplo del método de solapamiento. Fuente: http://www.dspguide.com/pdfbook.htm

Antes de implementar este método, se decidió realizar primero las pruebas con la función de Matlab *fftfilt(D,x,Nfft)* donde D es el filtro FIR, x la señal de entrada y Nfft el número de puntos de la fft. Tras realizar las pruebas, se observó que los picos de la señal habían desaparecido y se obtenía una calidad de audio final aceptable.

4.1.5 -. Resultados

Para el código de pruebas se creó un documento de Matlab en el que se implementaron conjuntamente los diferentes métodos para obtener la transformada de Hilbert, se realizaba la ecuación de anti-feeback (1) y con la señal obtenida se calculaba el valor RMS comparándola con la señal anti-feedback ideal, obtenida mediante la función *hilbert()*. Se realizaron diversas pruebas generando diferentes tipos de señales simples (senoidal, cuadrada, sawtooth), grabaciones de voz y música, las cuales se

procesaban utilizando distintos parámetros de cada algoritmo implementado. Los resultados se reunieron en 3 tablas, una por cada método implementado.

Tabla 1: Resultados RMS del filtro FIR

FIR RMS	10	20	40	50	60	80	100	120	140	160	180	200
F=1000; TW=0,1; Fs= 48000; t = 15s												
S. senoidal	0,2027	0,122	0,0448	0,0264	0,0188	0,0083	0,0037	0,0017	7,91E-04	3,69E-04	1,74E-04	8,26E-05
S. cuadrada	0,2681	0,1578	0,0581	0,0353	0,0261	0,0143	0,0103	0,009	0,0084	0,0081	0,079	0,0076
S. sawtooth	0,138	0,0823	0,0295	0,0182	0,0136	0,008	0,0062	0,0055	0,0053	0,0051	0,0049	0,0048
t=15s; Fs=44100Hz												
S. Voz Masc.	0,1057	0,0958	0,0823	0,077	0,0742	0,0684	0,064	0,0605	0,0577	0,0554	0,0534	0,0517
S.Voz Fem.- Música	0,0542	0,0495	0,0438	0,0416	0,0405	0,0381	0,0363	0,0347	0,0335	0,0323	0,0314	0,0305

La primera tabla muestra los resultados obtenidos utilizando un filtro FIR con diferente número de coeficientes. En ella se puede apreciar que con esta técnica se obtiene valores de error bastante pequeños en todos los tipos de señales utilizadas, que, como era de esperar, se van reduciendo conforme aumenta el número de coeficientes.

El resultado auditivo también fue satisfactorio, obteniendo una buena calidad de audio sin mucha distorsión, aunque sí que se podía notar las oscilaciones producidas por la frecuencia moduladora de 5 Hz.

Tabla 2: Resultados RMS de la FFT.

FFT RMS	16	32	64	128	256	512	1024
F=1000; Fs= 48000; t = 15s							
S. senoidal	0,5039	0,3235	0,2265	0,1629	0,1171	0,0836	0,0594
S. cuadrada	0,6815	0,453	0,3217	0,2324	0,167	0,1192	0,0848
S. sawtooth	0,3826	0,2765	0,2037	0,149	0,1074	0,0769	0,0547
t=15s; Fs=44100Hz							
S. Voz Masc.	0,142	0,1372	0,1105	0,0805	0,0591	0,0378	0,0294
S.Voz Fem.- Música	0,0733	0,0679	0,0577	0,0443	0,0331	0,0222	0,0141

Con el uso de la técnica de FFT, también se pueden observar valores de error pequeños, aunque no tan buenos como en el caso anterior. Como era de esperar, conforme se incrementa el número de puntos de la FFT el valor de error se reduce.

En este caso el resultado auditivo no fue tan bueno. Debido a los picos de distorsión que se comentan en el apartado de la FFT, a la hora de reproducir una señal de audio

se puede apreciar un chasquido que ensucia la señal y no la hace nada agradable para el oído.

Tabla 3: Resultados RMS del filtro FIR basado en FFT.

FIR-FFT32 RMS	10	20	40	50	60	80	100	120	140	160	180	200
F=1000; TW=0,1; Fs= 48000; t = 15s												
S. senoidal	0,0276	0,3819	0,3487	0,044	0,2702	0,4467	0,0643	0,4665	0,1621	0,3761	0,3439	0,1969
S. cuadrada	0,2545	0,5157	0,5084	0,1808	0,4069	0,5918	0,2352	0,6478	0,3033	0,5042	0,4793	0,3401
S. sawtooth	0,1988	0,3006	0,3061	0,125	0,2658	0,3338	0,1675	0,3302	0,1973	0,296	0,2882	0,2253
t=15s; Fs=48000Hz												
S. audio música-voz	0,146	0,1334	0,1276	0,1218	0,1166	0,1239	0,1315	0,13	0,1278	0,1271	0,1186	0,1145

Tras realizar las pruebas con la técnica del filtro FIR basado en FFT, que resolvió el problema de la técnica anterior, se obtuvieron unos valores de error muy dispares. Como se puede observar en la tabla 3, los valores están bastante alejados del cero, aun incrementando el número coeficientes (el número de puntos de FFT tiene un valor fijo de 32 debido a que el DSP Blackfin trabaja con un máximo de 32 muestras).

Aun así, la prueba auditiva fue satisfactoria comparada con los datos obtenidos. La señal reproducida mantenía una calidad bastante buena, que iba mejorando conforme aumentaba el número de coeficientes. Hay que decir que la componente tonal se distorsionaba, sobre todo con una señal de audio musical, y que la oscilación moduladora era apreciable, pero podía ser reducida con valores más bajos de *fm*. El chasquido producido en la técnica de FFT desaparecía, haciendo su escucha mucho más agradable.

Observando los resultados de las tres técnicas para obtener la transformada de Hilbert se concluyó que, el método que se implementará para realizar el algoritmo anti-feedback en el DSP será mediante la técnica del filtro FIR, debido a que es la que menor valor de error y mayor calidad de audio proporciona, además de ser la más sencilla de implementar en el lenguaje de CrossCore y la que menos recursos de DSP consumirá.

También se podría deducir de los resultados anteriores que, no por tener menor valor de error, la calidad de audio final sea mejor. Aquí es donde entra en juego la subjetividad de cada uno, ya que el oído humano es más susceptible a unas frecuencias y variaciones que a otras, y aunque matemáticamente una técnica puede ser mejor que otra, hasta que no es recibida por nuestro oído no es posible establecer un criterio de calidad de audio.

4.2 -. Implementación del algoritmo en el DSP

Ha llegado el momento de implementar el algoritmo en lenguaje C++ utilizando Crosscore IDE. Tras los anteriores resultados se puede afirmar que el algoritmo a implementar estará basado en el método PFC, más concretamente el de FS, y que la transformada de Hilbert se realizará mediante filtro FIR de N coeficientes.

Cuando se llevó a cabo esta fase se comenzó con la implementación del código en el DSP Blackfin, pero tras ciertas dificultades con la programación en punto fijo, de las que se hablará posteriormente, se decidió implementar primero el código en el DSP SHARC, cuya programación en coma flotante hace más sencillo el proceso de adaptación del código.

4.2.1 -. Método PFC en SHARC

El primer paso fue comprobar el correcto funcionamiento del DSP y del programa. Para ello se cargó una plantilla con un *talkthrough* que viene incorporada en la carpeta de ejemplos del programa. Esta plantilla incorpora los ficheros y funciones necesarias para conseguir a la salida la misma señal de entrada, cosa que consiste en enviar al buffer de salida las muestras tomadas a la entrada, sin ningún procesamiento. Reutilizando esta plantilla, se comenzó a implementar el algoritmo de control de feedback siguiendo los siguientes pasos:

1. Obtener las muestras de la señal de entrada.
2. Procesar las muestras mediante un filtro FIR utilizando los N coeficientes obtenidos con Matlab.
3. Corregir el retardo generado por el filtro retardando la señal de entrada N/2 muestras.
4. Aplicar la ecuación anti-feedback (1) con la señal retardada, la salida del filtro y generando un seno y un coseno con la frecuencia moduladora.
5. Guardar las muestras procesadas en una variable de salida consiguiendo la señal anti-feedback.

Se creó una función llamada AntiLarsen dentro del *.c* donde se realizaron los pasos anteriores. Esta función tiene como entrada las muestras de la señal de entrada y, como salida, las muestras procesadas.

A partir de las muestras de entrada se obtuvo la transformada de Hilbert de la señal utilizando la función *fir (dataIn, coeffs, state, TAPS)* que se encuentra dentro de la librería <filters.h> proporcionada por CrossCore. Como se puede ver, esta función consta de cuatro parámetros: la entrada del filtro, los valores de los coeficientes del filtro, el estado del filtro (serie de parámetros que es necesario inicializar antes de llamar a la función), y el número de coeficientes del filtro.

En el parámetro *coeffs* se introdujo el valor de los coeficientes correspondientes a un filtro de Hilbert de N coeficientes obtenido en Matlab. Con esto se consiguió que a la salida del filtro se obtuviera la transformada de Hilbert de la señal.

El siguiente paso era corregir el retardo producido por el filtro. Para ello fue necesario crear un array de longitud N/2, donde se iban almacenando las muestras de la señal de entrada una a una. La señal retardada se consiguió almacenando, en otra variable, el valor de la posición de escritura del array, antes de sobrescribirlo con el valor de la nueva muestra. Así la nueva variable de salida contenía el valor más antiguo almacenado en el array, es decir, la N/2 muestra más antigua.

Obtenidas la señal retardada y la salida del filtro (transformada de Hilbert), se procedió a la implementación de la ecuación de anti-feedback:

$$d(t) = y(t)\cos\varphi(t) - \hat{y}(t)\sin\varphi(t)$$

Donde $\varphi(t) = \omega_m t$ para FS.

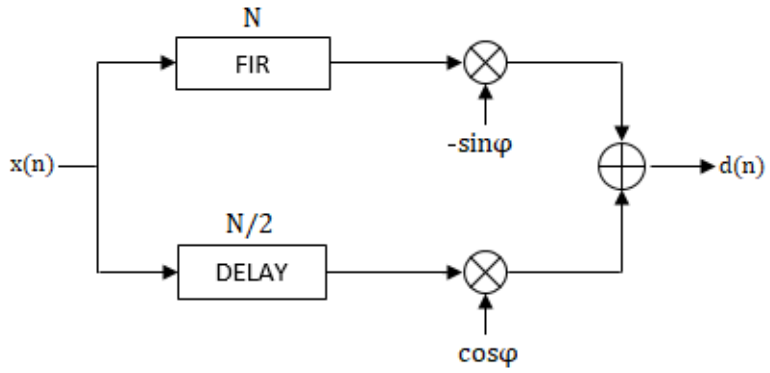

Figura 13: Diagrama ecuación anti-feedback. Fuente: Propia.

Para generar el seno y el coseno moduladores se utilizó los funciones *sinf()* y *cosf()* que incorpora la librería <math.h>. La variable ω_m se obtuvo de la siguiente forma, $(2\pi * fm)/Fs$, donde Fs es la frecuencia de muestreo, con un valor de 48kHz. Para conseguir la variable de tiempo t, en CrossCore es necesario ejecutar un bucle contador de muestras que vaya incrementando su valor con la frecuencia de muestreo.

Una vez implementada la ecuación anti-feedback solo era necesario llamar a la función AntiLarsen e introducir una señal de audio de entrada.

Es importante recalcar que al programar en formato de coma flotante se trabaja principalmente con dos tipos de variables: variables flotantes (float) y enteros (int). La interacción entre estos dos tipos de variables es compatible, causa principal de la sencillez de este tipo de programación.

4.2.2 -. Método PFC en Blackfin

Los pasos a seguir en la implementación del código en el Blackfin son exactamente los mismos que en el apartado anterior. La diferencia reside en que en este caso se trabaja con variables fraccionarias (fract) que representan valores entre -1 y 1. Este tipo de variables no es compatible con los otros tipos, por lo tanto, siempre que se desea operar con ellas es necesario realizar una conversión de formato mediante funciones de librería del Blackfin. Dentro de las variables de tipo fraccionario existen diversos tipos dependiendo del número de bits que utilice, pudiendo distinguir entre *fract16* y *fract32*. Trabajar con este tipo de variables tiene aún más complejidad en las operaciones entre ellas, por lo que CrossCore incluye un tipo de variable fraccionaria nativa que hace más sencillo su uso, las variables tipo *fract* y *long fract*, correspondientes a valores en Ca2 de 16 y 32 bits.

También hay que tener en cuenta que en el SHARC se estaba trabajando muestra a muestra, mientras que el Blackfin trabaja en bloques de 64 muestras (32 para cada canal L y R), influyendo en la ejecución de funciones, como lo función *fir*.

De vuelta al proceso de implementación, el primer paso fue generar la función AntiLarsen, donde se recibían las muestras del buffer de entrada y se realizaba la separación de las muestras en dos canales L y R. Como las muestras recibidas se encontraban por defecto en formato *fract32*, fue necesario realizar la conversión a

formato nativo mediante la función *lrbits(fract32)*. Esta función de librería convierte patrones de bits de números fraccionarios a diversos formatos nativos, la abreviatura delante de la palabra bits decidirá cuál es el formato final (*lr = long fract*).

Cuando se tuvo las muestras de los dos canales por separado en el formato correcto, se prosiguió con la implementación del filtro de Hilbert. Para ello se utilizó la función de librería *fir_fx32(In, Out, Muestras_Canal, estado).* Esta función, al contrario que la utilizada en coma flotante, realiza el filtrado por paquetes de 32 muestras en vez de muestra a muestra, de ahí que sea necesario introducir el número de muestras de cada paquete. El estado del filtro se inicializa independientemente con una serie de parámetros, como el valor de los coeficientes (formato long fract) o el número de coeficientes, entre otros. Como solo se trabajó sobre un canal mono, no era necesario implementar el filtrado en los dos canales, pero si se tuvo que tener en cuenta por qué canal entraba la señal.

Para la corrección del retardo del filtro se utilizó el mismo código que en el apartado anterior, solo que en este caso fue necesario crear un bucle que fuera guardando una a una las muestras de cada paquete de entrada de 32 muestras, primero en el array de retardo y luego en un array de salida de 32 muestras. De esta forma el número de muestras de cada paquete a la salida del filtro coincidía con el de la señal retardada.

Obtenidas la transformada de Hilbert de la señal y la señal corregida, el último paso fue implementar la ecuación anti-feedback (1) a las dos señales. En este caso no existía una función de librería que generara un seno o coseno completo, sino que la función existente solo generaba la señal senoidal por cuadrantes. Para poder generar la señal completa fue necesario crear una función que comparara el valor del argumento del seno o coseno con el valor de cada cuadrante del circulo unidad.

Como se trabajaba con números fraccionarios, el valor de la señal senoidal debía de estar siempre entre -1 y 1. Para que esto fuera posible, además de obtener la frecuencia digital (*f/fs*), se implementó un contador que se reiniciaba cuando llegaba al número de muestras de la frecuencia (*fs/f*), en vez de llegar hasta la frecuencia de muestreo *fs*. De esta forma, al multiplicarlo por la frecuencia digital, se obtenían siempre valores entre -1 y 1.

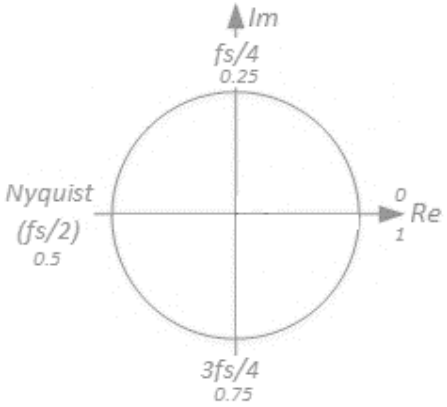

Figura 14: Circulo unidad. Fuente: Propia.

Para realizar las operaciones entre arrays y seno/coseno, fue necesario dividir la formula, guardando en dos variables el producto de las dos señales y luego restando

las dos variables entre sí. Estas operaciones se realizaron de esta forma para evitar problemas de compatibilidad de formato y también, para tener más independencia al variar valores en la ecuación anti-feedback.

Como el resultado se encontraba en formato fraccional nativo, se realizó una conversión inversa utilizando la función *bitslr(long fract)*, para así guardar la señal obtenida en el buffer de salida.

5 -. Pruebas y resultados

En este apartado se explica el proceso de testeo y los resultados obtenidos tras haber implementado el código en los dos DSPs. Las pruebas fueron realizadas intentando recrear una situación de realimentación acústica.

Los materiales utilizados para estas pruebas fueron:

- PC portátil.
- Tarjeta de sonido Focusrite Scarlett de 2 entradas.
- DSP: Blackfin y SHARC.
- Mesa de mezclas.
- Sistema de monitores (imitando PA en un escenario)
- Micrófono Behringer de condensador omnidireccional (ECM8000).
- Fuente de sonido: Voz o Altavoz.

Todos los materiales se conectaron y se situaron creando un bucle cerrado como se puede ver en la figura 15. De esta forma se forzaba a que la señal se realimentara produciendo acople.

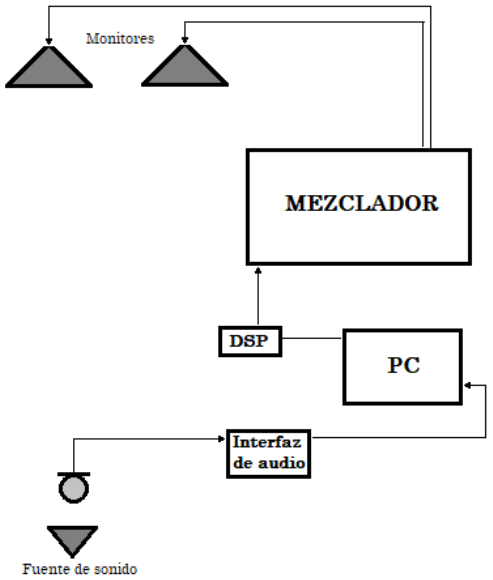

Figura 15: Esquema de la disposición de los elementos en la fase de pruebas. Fuente: Propia.

Con estas pruebas se pretendía comprobar la fiabilidad del algoritmo implementado, y observar que calidad de audio se obtenía variando el número de coeficientes del filtro de Hilbert. Utilizando Matlab se crearon diferentes archivos con diferente número de coeficientes, intentando obtener el menor valor RMS variando el ancho de banda de transición.

Tabla 4: Valores RMS mínimos obtenidos con diferentes coeficientes.

Nº de coeficientes	TW	RMS
50	0,08	0,0064
60	0,08	0,0053
80	0,08	0,0048
100	0,06	0,0043
120	0,06	0,0041
140	0,05	0,0038
160	0,04	0,0036
180	0,04	0,0035
200	0,04	0,0034

Como señales de prueba se utilizaron locuciones de radio y música descargadas de internet, a la vez que pruebas de voz reales.

Antes de empezar con el testeo, se igualaron los niveles de entrada de la señal, procurando que la señal que entraba al DSP tuviera el mismo nivel de ganancia, con y sin el anti-feedback. Esto se hizo porque cuando se llamaba a la función AntiLarsen, la señal de entrada se reducía debido al producto con la señal moduladora, por lo que hubo que amplificar la entrada. Ahora, con las dos entradas al mismo nivel y con los coeficientes generados, era posible obtener unos resultados válidos.

Las primeras pruebas se realizaron con señales de voz reales, es decir, voz en directo. Después, se instaló delante del micro un altavoz por donde se reprodujo primero una locución de radio y por último una grabación musical. Todas las señales se procesaron con 10 coeficientes hasta 200 coeficientes, con una frecuencia moduladora de 5 Hz y utilizando los dos DSP.

Los resultados de las pruebas fueron los siguientes:

- 10 – 50 coef.: Se nota distorsión en la señal procesada, aunque es claramente entendible. Presencia de modulación de fondo.
- 50 – 80 coef.: Reducción de la distorsión y calidad de audio aceptable. Continua modulación de fondo.
- 80 – 150 coef.: Reducción notable de la distorsión y mejora de la calidad de audio. La modulación de fondo persiste, pero menos notablemente.
- 150 – 200 coef.: Calidad de audio buena, mejorando conforme aumenta el número de coeficientes. La modulación de fondo se reduce mucho, casi no tiene presencia, pero persiste.

Estos son compartidos para los dos DSP. Cosa que era de esperar ya que contienen el mismo código y algoritmo. Podría decirse que la calidad del SHARC es mayor que la del Blackfin, pero no es algo tan notable como para afirmarlo.

Los resultados con las grabaciones musicales fueron muy parecidos a los obtenidos con las grabaciones de voz, pero en este caso sí que hay que recalcar una distorsión en el tono de los instrumentos que era mayor conforme menor era el número de coeficientes. En el documento *"Fifty years of acoustic feedback control: state of the art and future challenges,"* (van Waterschoot y Moonen, 2011, p. 301) se habla del problema de la técnica FS con señales musicales y propone utilizar las técnicas PM o FM para este tipo de señales.

Sabiendo esto, se hicieron pruebas utilizando la técnica PM, con un índice de modulación $\beta = 3.8$, el recomendado en el documento citado, al igual que una frecuencia de modulación inferior a 5 Hz. Tras las pruebas se observó una pequeña

mejoría en la calidad de la señal instrumental, sobre todo cuando se reducía la frecuencia de modulación a 1-2 Hz. Aun así, no se podría considerar una señal de alta calidad de audio.

Para dejar constancia de las pruebas se realizaron una serie de grabaciones donde se puede escuchar grabaciones con y sin el anti-feedback (enlaces incluidos en el anexo). En dos de ellas se utilizó una locución de radio que incluye también música de fondo, y las otras se grabaron con voz real. En las grabaciones sin anti-feedback se escucha claramente como, al aumentar a penas 2 dBs de ganancia, se produce un acople sobre los 400 Hz, emitiendo un sonido muy desagradable. Por otro lado, en las grabaciones con procesamiento, fue posible aumentar hasta unos 10 dBs hasta que la modulación se hace demasiado presente, deteriorando la señal. Con esto se puede comprobar como el uso del algoritmo anti-feedback permite un aumento de ganancia mayor que sin él, haciendo posible una mejor escucha de la señal de audio.

Estas pruebas han servido, además de valorar la calidad de audio utilizando distintos parámetros, para conocer las deficiencias de este tipo de algoritmo. Aun logrando su correcto funcionamiento eliminando la componente de feedback de la señal y permitiendo un aumento de ganancia, la calidad de audio de la señal procesada no es totalmente fiel a la original, seguramente pudiéndose mejorar variando los parámetros de la función o los niveles de la señal de entrada y las señales moduladoras. De todas formas, se puede decir que las pruebas fueron un éxito.

6 -. Problemas y soluciones

A lo largo de la realización del trabajo han ido surgiendo problemas por diferentes motivos, pero tras un periodo de análisis, se ha conseguido encontrar y solucionar los fallos.

Debido a que estamos hablando de dos tipos de DSP que, aun siendo muy versátiles, cada uno cuanta con ciertas limitaciones. En algunos casos la solución a los problemas aparecidos no consistía en una aclaración de un experto, sino en buscar una alternativa. Otro de los problemas que se ha podido comprobar es que el mundo del DSP está muy protegido, cada empresa patenta sus creaciones y no se pueden encontrar ejemplos de código abierto fácilmente, cosa que ralentiza la tarea de solución de problemas. A continuación, se detallan algunos problemas surgidos:

- A lo largo de toda la programación en CrossCore han surgido infinidad de comportamientos no esperados a la hora de realizar algunas funciones, los cuales se resolvieron realizando continuas revisiones de la programación, depuraciones del código y observación del comportamiento del algoritmo en el osciloscopio.
- Se tuvo muchos problemas con la adaptación del código a formato de punto fijo. El primer intento se realizó con el formato no nativo del DSP, resultando en problemas de operación entre variables y generando una señal de salida con ruido. Esto se solucionó investigando a fondo el formato nativo, aplicando y adaptando las funciones a ese tipo de formato. Entre las funciones que más problemas dio se encuentran la del seno y coseno, debido a la necesidad de ir generándolos por cuadrantes.
- Problemas con el nivel de ganancia de la señal de entrada y la moduladora, siendo necesario establecer unos niveles de señal adecuadas para que la modulación no repercutiera en exceso en la calidad de la señal final.

7 -. Conclusiones

El proyecto ha conseguido cumplir el objetivo de implementar un algoritmo para el control del feedback acústico en dos modelos de DSP distintos. A su vez, se ha conseguido estudiar a fondo las posibilidades que ofrece trabajar con CrossCore Embedded Studio y con el DSP como herramienta de procesamiento de audio. De esta forma se cumple el objetivo de punto de partida del proyecto.

Respecto a los dos modelos de DSP se ha podido conocer las capacidades que ofrece cada uno. El modelo Blackfin, más limitado computacionalmente, pero perfecto para nuevos usuarios que quieran comenzar en la materia, además es adaptable para programar en coma flotante, aunque se consume más recursos. El modelo SHARC, mucho más profesional y con unas características que lo hacen capaz de procesar una gran cantidad de datos, además, su formato nativo de programación es coma flotante, facilitando mucho su uso.

A partir de los resultados obtenidos del algoritmo anti-feedback se puede concluir que, el método PFC FS es eficaz, robusto y de fácil implementación, con un buen rendimiento en señales simples de voz, pero aun así cuenta con limitaciones, como la

distorsión en la calidad de audio final, sobre todo en señales musicales, y en la cantidad de incremento de señal estable que puedes obtener sin llegar a percibir la moduladora.

Mirando hacia futuras líneas de estudio, creo que sería posible mejorar este método, optimizándolo, y ejecutándolo conjuntamente con otra técnica que sea capaz de suplir sus carencias. Además, otro objetivo de interés sería conseguir implementar estos métodos de control del feedback acústico en sistemas multicanal, cosa que aumentaría su versatilidad.

Finalmente, pienso que los métodos de control de feedback son una herramienta que puede ser de mucha utilidad en sistemas de refuerzo de sonido, pero aún queda mucho para que sea completamente rentable su uso, sean capaces de adaptarse a las diferentes situaciones de sonorización y producir mejores resultados que la labor que hace un técnico de sonido para eliminar ese efecto tan molesto llamado feedback acústico.

8 -. Bibliografía

- Analog Devices, CrossCore® Embedded Studio 2.5.0 C/C++ Compiler and Library Manual for Blackfin Processors (Rev. 1.9), en http://www.analog.com/media/en/dsp-documentation/software-manuals/cces-BlackfinCompiler-library-manual.pdf [Consulta: Junio,10,2017]

- Analog Devices, CrossCore® Embedded Studio 2.6.0 C/C++ Library Manual for SHARC Processors (Rev. 1.9), en http://www.analog.com/media/en/dsp-documentation/software-manuals/cces-SharcLibrary-manual.pdf [Consulta:Junio,11, 2017]

- Analog Devices, ADSP-BF706 en http://www.analog.com/en/products/processors-dsp/blackfin/adsp-bf706.html#product-overview

- Analog Devices, ADSP-21489 en http://www.analog.com/en/products/processors-dsp/sharc/adsp-21489.html#product-overview

- DUOANDIKOETXEA, J. (2000), Fourier Analysis, American Mathematical Society

- MARPLE, S. L. "Computing the Discrete-Time Analytic Signal via FFT." IEEE Transactions on Signal Processing. Vol. 47, 1999.

- MathWorks, FFT-based FIR filtering using overlap-add method, en https://es.mathworks.com/help/signal/ref/fftfilt.html [Consulta: Marzo, 2017]

- MathWorks, Discrete-time analytic signal using Hilbert transform, en https://es.mathworks.com/help/signal/ref/hilbert.html [Consulta: Marzo, 2017]

- OPPENHEIM, A. V. Y SCHAFER, R.W. (2010). Discrete-Time Signal Processing, 3rd Edition, Pearson Education

- SMITH, S. W. (1997). The Scientist and Engineer's Guide to Digital Signal Processing, en http://www.dspguide.com/pdfbook.htm [Consulta: Junio, 2017]

- SOGORB, T. C. (2016). Electrónica Aplicada al Audio. Módulo 1: Procesadores Digitales de Señal (Primera Parte). Escuela Politécnica Superior de Gandía.

- VAN WATERSCHOOT, T. Y MOONEN, M. (2011). "Fifty years of acoustic feedback control: state of the art and future challenges," Proc. IEEE, vol. 99, no. 2, pp. 288-327, Feb. 2011.

- Wikipedia, Hilbert Transform, en https://en.wikipedia.org/wiki/Hilbert_transform [Consulta: Agosto, 2017]