DCSE - Grupo especial

Trabajo con herramientas de diseño CAD-VHDL

Memoria del proyecto

Madrid, Noviembre 2012

Autor:

Adrián Pérez Orozco

Índice

1. Introducción

En esta memoria vamos a tratar un proyecto realizado para la asignatura *Diseño de Circuitos y Sistemas Electrónicos* impartida en la ETSI de Telecomunicación de la Universidad Politécnica de Madrid.

El proyecto se propone como trabajo adicional a la asignatura, para tratar los temas de diseño de circuitos digitales de una forma más práctica y obtener conocimientos mucho más amplios sobre tecnologías relacionadas con estos campos.

En concreto, la propuesta consiste en el diseño y simulación de un sistema utilizando VHDL. El sistema propuesto consiste en un ecualizador de audio, del cual se implementarán algunos subsistemas de procesado digital en VHDL. Esta práctica se apoya sobre el trabajo realizado en el año anterior en el *Laboratorio de Sistemas Electrónicos Digitales*, donde se realizó este sistema utilizando un microcontrolador.

El esquema de dicho sistema puede verse en la figura 1. El proyecto que nos ocupa se centrará en la realización del subsistema de procesado digital de audio, cuyo esquema puede verse en la figura 2.

Para la realización del proyecto se ha utilizado la herramienta ModelSim. De forma auxiliar, se utilizará la herramienta MATLAB para el tratamiento de las señales obtenidas de forma sencilla y eficiente, para poder así evaluar el funcionamiento del sistema.

1.1. Descripción de los módulos VHDL

El subsistema digital representado en la figura 2 se va a dividir en los siguientes módulos principales VHDL:

Banco de filtros Este módulo incluye los 7 filtros digitales que separarán la señal de entrada en sus diferentes bandas de frecuencia. Incluye 7 submódulos correspondientes a los 7 filtros digitales. Además, este módulo ofrece la funcionalidad de selección de ganancia, como se explicará posteriormente.

Módulo de reverberación Este módulo sirve para retardar y atenuar una señal. Se utilizará para realimentar en el banco de filtros la salida retardada hasta la entrada, para generar un efecto de reverberación. El subsistema de retardo se ha implementado independientemente como un submódulo.

Vúmetro Proporciona información sobre el nivel de señal de cada una de las bandas de audio. Consiste en 7 elementos vúmetro individuales que se agrupan para obtener el vúmetro de 7 bandas.

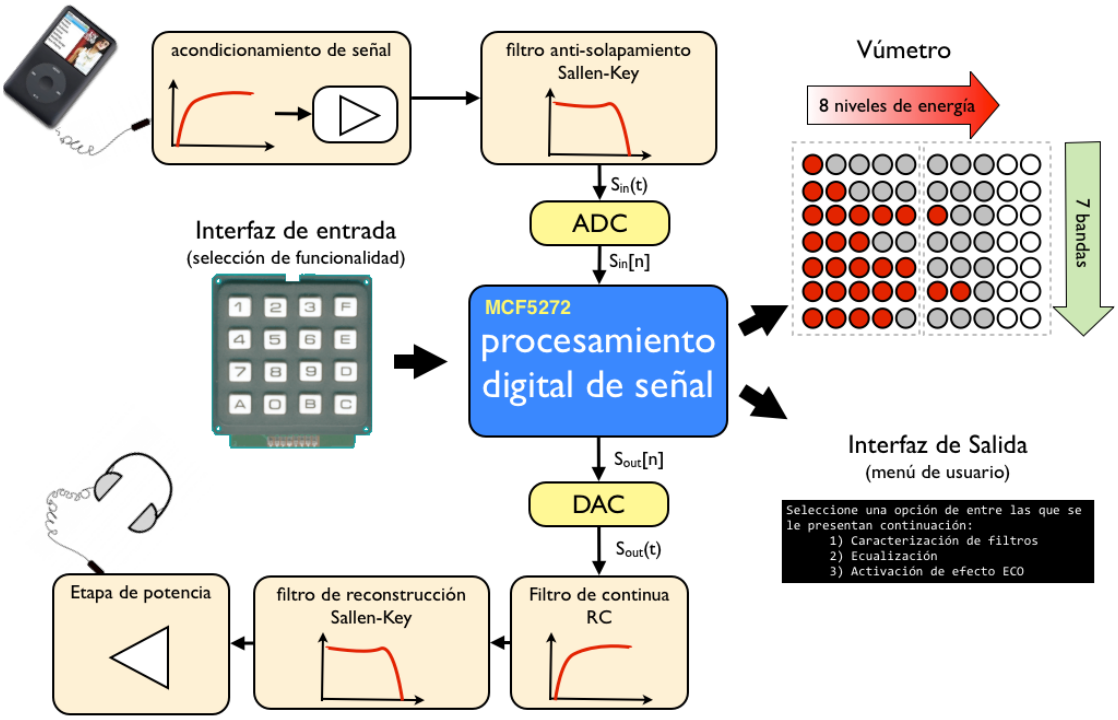

Figura 1: Descripcción del sistema completo

Todo ello se ha agrupado en un único módulo *ecualizador*, para facilitar la simulación y la ejecución de pruebas. La relación de módulos y señales puede verse en la figura 3.

Por último, se han implementado sumadores y multiplicadores con las estructuras estudiadas en clase. Como mejora, se han sustituido las operaciones de suma y multiplicación de señales en el sistema mediante funciones VHDL (operadores + y *) por sumas y productos obtenidos mediante sumadores y multiplicadores más realistas.

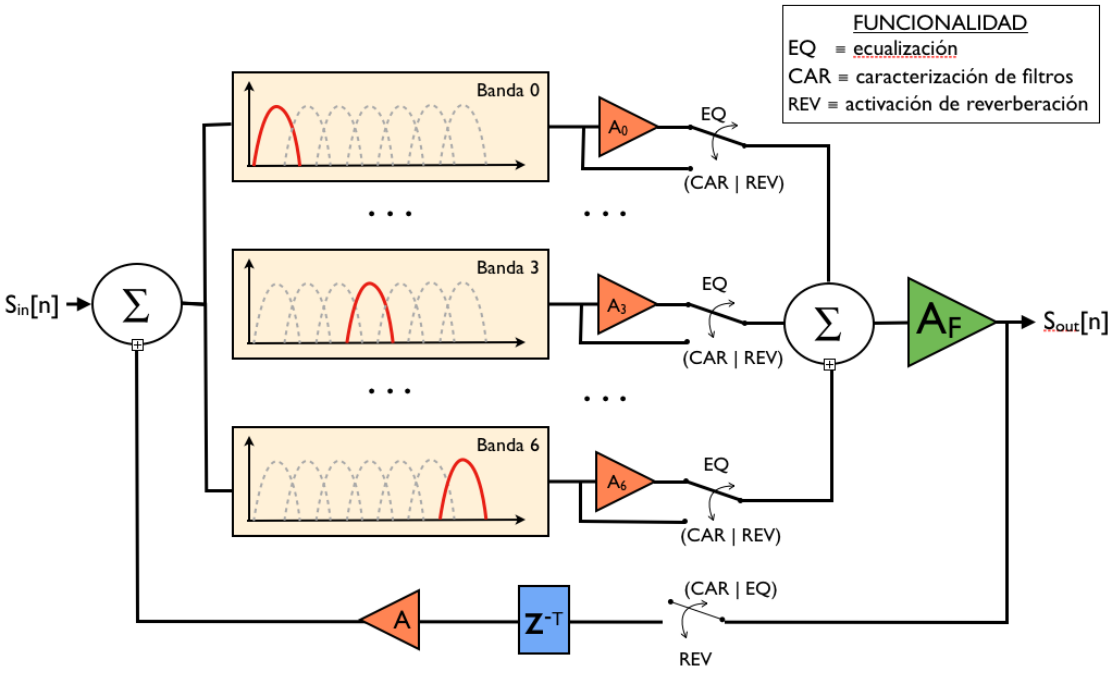

Figura 2: Descripcción del subsistema de procesado digital

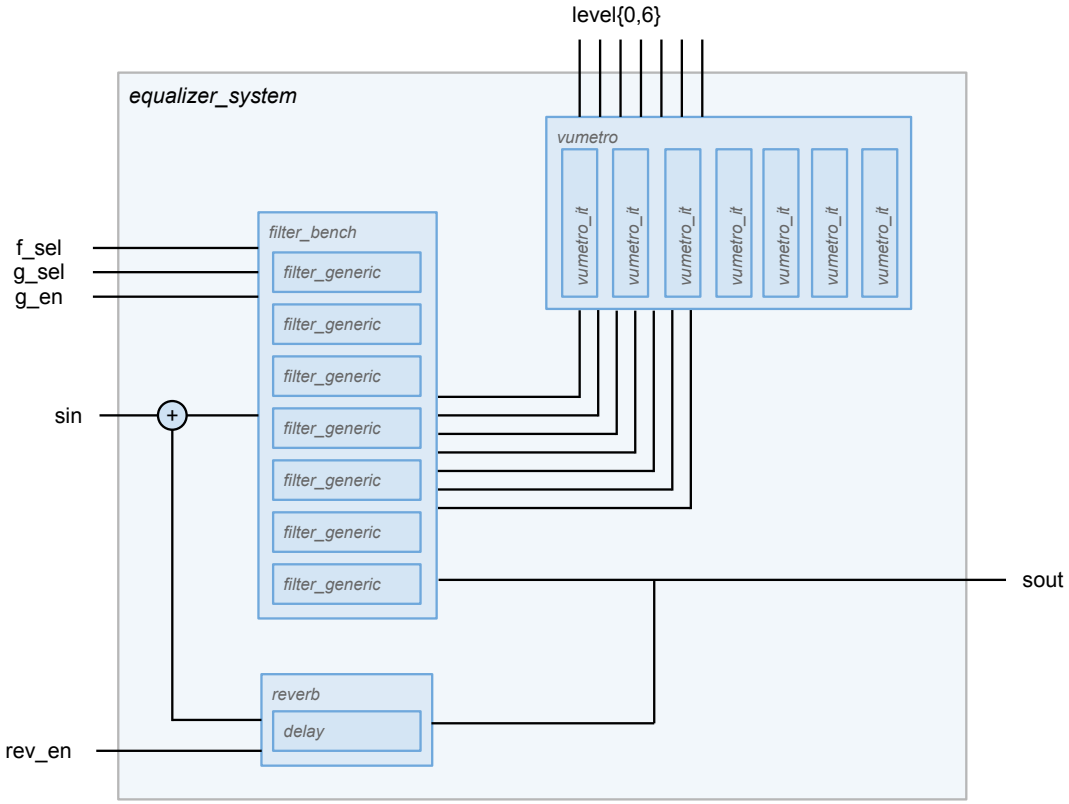

Figura 3: Relación de módulos VHDL

Banda	f_0	f_{c1}	f_{c2}
0	31.25	22.10	44.19
1	62.5	44.19	88.39
2	125	88.39	176.78
3	250	176.78	353.55
4	500	323.55	704.11
5	1000	707.11	1414.21
6	2000	1414.21	2828.43

Cuadro 1: Descripción de las distintas bandas y sus filtros asociados

2. Diseño de banco de filtros

El sistema propuesto se trata de un ecualizador de audio, por lo que las señales de entrada que tendremos estarán comprendidas en el rango de frecuencias audibles por el hombre. En concreto, la especificación del sistema propuesto propone señales cuya frecuencia estará comprendida aproximadamente entre los 22 Hz y los 2.8 KHz.

Este rango de frecuencias se dividirá en 7 bandas, para lo cual utilizaremos filtros IIR de segundo orden. Las bandas de frecuencias se detallan en la tabla 2, y los coeficientes para la realización de los filtros IIR en la tabla 2.

2.1. Implementación de los filtros en VHDL

Para una implementación más sencilla de los filtros, utilizaremos la *Forma directa II*. El detalle de esta implementación se puede ver en la figura 4.

A la hora de implementar estos filtros digitales, tenemos que tener en cuenta una limitación muy importante. Debemos definir un ancho de palabra fijo para la representación de las señales de entrada y de salida. En nuestro caso, hemos escogido un ancho de palabra de *16 bits*, correspondientes a *6 bits enteros* y *10 bits fraccionarios*. Esto es especialmente conveniente a la hora de representar los coeficientes de los filtros, ya que para todos ellos tenemos que $a_0 = 1024$. Normalizar los coeficientes realizando una división por 1024 equivale a desplazar los bits 10 lugares a la derecha (o lo que es lo mismo, la coma decimal 10 lugares a la izquierda). De esta forma, cuando representemos los coeficientes enteros como palabras binarias de 16 bits, bastará como tomar los 10 bits menos significativos

Filtro	Ganancia	a_0	a_1	a_2	b_0	b_1	b_2
0	8	1024	-2029	1006	1024	0	-1024
1	17	1024	-2011	988	1024	0	-1024
2	34	1024	-1970	955	1024	0	-1024
3	66	1024	-1878	890	1024	0	-1024
4	125	1024	-1660	772	1024	0	-1024
5	227	1024	-1115	569	1024	0	-1024
6	392	1024	141	239	1024	0	-1024

Cuadro 2: Descripción de los coeficientes de los filtros IIR

como fraccionarios para tener el coeficiente normalizado a 1024.

2.2. Ruido de cuantificación

El haber elegido un ancho de palabra de 16 bits con 6 bits enteros y 10 fraccionarios nos impone una limitación considerable en cuanto a la resolución de nuestro sistema a la hora de representar señales. Sobretodo en señales de amplitud baja, tendremos un ruido de cuantificación que puede llegar a influir en el correcto funcionamiento de nuestro sistema.

Para intentar minimizar el ruido de cuantificación, se puede intentar ajustar más el numero de bits enteros, ya que las señales con las que trabajaremos están normalizadas a 1, y a priori debería bastarnos con un bit entero y 15 fraccionarios. Este enfoque se ha intentado realizar sin éxito en la implementación, ya que aunque las señales de entrada estén normalizadas a 1, en puntos intermedios de los filtros algunas señales pueden tomar valores más altos, produciendo desbordamiento. Por ello, se ha decidido tomar una postura más conservadora y mantener la representación con 10 bits fraccionarios, que aunque no es óptima en sentido del ruido de cuantificación, nos ofrece mayor seguridad frente a desbordamiento.

Otra forma de abordar este problema sería utilizar palabras de 32 bits en lugar de 16. Esto nos ofrecería una resolución enormemente mayor, mejorando sensiblemente la calidad del sistema. No obstante, trabajar con palabras de 32 bits supone una complejidad mucho mayor a la hora de implementar los sumadores y multiplicadores vistos en clase. Debido a que el objetivo del proyecto es mayormente didáctico y no obtener una calidad de audio excepcional, se ha decidido mantener

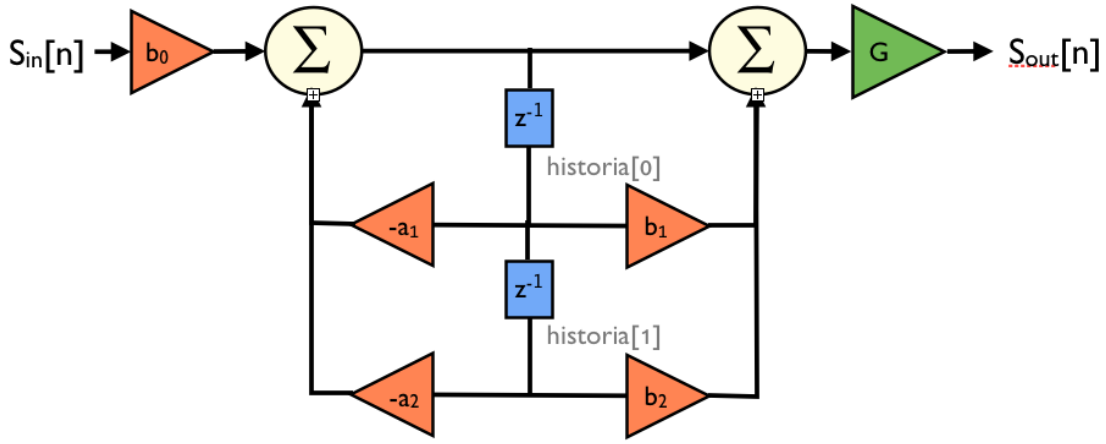

Figura 4: Diagrama de bloques de un filtro IIR de segundo orden

las palabras de 16 bits y asumir los efectos del ruido de cuantificación.

2.3. Ajuste de ganancias

Además de separar la señal en 7 bandas de frecuencias, el banco de filtros de nuestro sistema debe aplicar a cada uno una ganancia ajustable y disponer a la salida de la señal global sumada.

La selección de ganancia para cada banda se realiza mediante 3 entradas adicionales:

Entrada f_sel Entrada de 3 bits mediante la cual seleccionamos el filtro (de 0 a 6) cuya ganancia queremos ajustar.

Entrada g_sel Entrada de 4 bits mediante la cual seleccionamos la ganancia que queremos aplicar al filtro seleccionado. Estas ganancias se encuentran definidas en el propio sistema en forma de tabla, de forma que a cada uno de los valores 0 a 15 le corresponde un valor de ganancia predefinido.

Entrada g_en Señal de *enable* del sistema de selección de ganancia. Cuando se detecta un flanco de subida se aplica la ganancia seleccionada al filtro seleccionado.

Para evitar desbordamiento en la señal al aplicar la ganancia, trabajaremos con posibles valores de ganancia menores que 1 (atenuaremos la señal). De esta forma, la ganancia numero 0 corresponde a una ganancia de 0dB, y de ahí se irá bajando a medida que aumente el índice de la ganancia seleccionada.

3. Diseño de módulo de reverberación

El subsistema de reverberación consiste en un módulo que ofrece a la salida la señal que obtiene a la entrada con un retraso de N posiciones y aplicándole una atenuación determinada.

Para la implementación de este módulo se ha utilizado como base una cola FIFO, en la cual se va introduciendo la señal que posteriormente se va obteniendo por la salida tras N ciclos de reloj. Para ello es importante inicializar la cola para que esté llena de palabras a cero, ya que si no la primera muestra de la señal de entrada ocuparía la primera posición y sería la que se ofrecería a la salida en el siguiente ciclo de reloj.

En nuestro sistema, el numero de palabras en la cola será siempre fijo e igual al número de muestras que queramos retrasar la señal. La implementación de una cola FIFO tiene cierta complejidad derivada del hecho de que el numero de palabras en la cola es variable y puede ser impredecible, conllevando posibles problemas de desbordamiento o cola vacía. Sin embargo, puesto que el funcionamiento de nuestro retardador es mucho más predecible (llegará una muestra por cada ciclo de reloj y sacaremos una muestra por cada ciclo de reloj, resultando en un tamaño constante) podemos simplificar considerablemente este diseño.

Se ha implementado para ello el subsistema *delay*, que consiste en una memoria de N posiciones (tamaño configurable mediante genéricos) con un puntero señalando a una de esas posiciones. En cada ciclo de reloj, se sacará la señal de la posición indicada por el puntero y se añadirá en su lugar la señal que llegue a la entrada. El valor del puntero se va aumentando de forma cíclica, de forma que la señal que acabamos de introducir no se sacará hasta que el puntero vuelva a apuntar a esa misma posición, es decir, tras N ciclos de reloj.

Posteriormente, la señal se atenúa mediante un valor de ganancia también configurable mediante genéricos, y se realimenta a la entrada del banco de filtros.

4. Diseño de vúmetro

El vúmetro ofrece información sobre el nivel de señal disponible en cada una de las bandas de señal. Para abordar la implementación de este módulo, se ha implementado un subsistema vúmetro con una única entrada que ofrece el nivel de la señal que tenga a la entrada. Estos vúmetros individuales se agrupan en un módulo vúmetro de 7 entradas, que conectaremos a la salida de cada uno de los filtros del banco de filtros.

Cada uno de los vúmetros individuales generan a la salida una señal de 8 bits, simulando los leds que tradicionalmente se utilizan en estos sistemas. El número de bits que se pongan a 1 en la señal (leds que se iluminan) indicará el nivel de la

señal en esa banda.

Para detectar el nivel de la señal, observaremos la palabra a nivel binario. Una palabra de la señal está representada de la siguiente forma:

$$X_{15}X_{14}X_{13}X_{12}X_{11}X_{10}X_9X_8X_7X_6X_5X_4X_3X_2X_1X_0$$

Definiremos cada uno de los 8 niveles de señal dividiendo los 16 bits en conjuntos de dos. Así, el nivel 0 corresponderá a todos los bits a 0, el nivel 1 se alcanzará cuando sólo los bits X_1 o X_0 estén a nivel alto, el nivel 2 si los bits X_4 o X_3 están a 1, y así sucesivamente. cada uno de estos niveles *encenderá* a la salida el led correspondiente a su nivel además de todos los de los niveles anteriores.

Para tener en cuenta las posibles palabras negativas en complemento a dos, es importante tener en cuenta que debemos observar siempre el valor absoluto de las señales, ya que si no siempre obtendríamos el nivel más alto para cualquier entrada negativa.

4.1. Mantenimiento del nivel

Debido a las posibles variaciones rápidas de la señal, es importante que los picos a la salida del vúmetro se mantengan un mínimo tiempo para su correcta visualización.

Para implementar este comportamiento, se ha definido el siguiente procedimiento:

1. Se toma una muestra a la entrada y se mantiene en memoria
2. Se inicia un contador (configurable mediante genéricos) que indica el tiempo mínimo que se han de mantener los niveles a la salida
3. Si a la entrada llega alguna muestra mayor a la que tenemos en memoria, se sustituye y se reinicia el contador, comenzando el proceso desde el principio.
4. Si el contador llega a cero, se elimina la señal de memoria, permitiendo que otras señales más bajas tomen el lugar de la muestra. En este punto el pico anterior deja de mantenerse y el nivel del vúmetro puede bajar
5. La siguiente muestra entra en memoria y se reinicia el proceso

Esto nos permite mantener de forma sencilla los niveles un mínimo de tiempo en pantalla sin perder información de posibles picos que puedan aparecer mientras estamos manteniendo la señal.

5. Mejoras

Para la aplicación de los conocimientos obtenidos en la asignatura *Diseño de Circuitos y Sistemas Electrónicos*, se ha propuesto la realización de varias modificaciones que se corresponden con lo estudiado en la parte digital de la asignatura.

En concreto, se ha propuesto la implementación de sumadores y multiplicadores binarios, correspondiéndose con las configuraciones estudiadas en clase. Estas implementaciones se han de realizar a bajo nivel, mediante la conexión y configuración de *Full Adders*, ilustrando además los efectos de retardo en cada una de las distintas configuraciones.

En concreto, se han implementado las siguientes configuraciones:

- Sumador ripple carry
- Sumador carry bypass
- Multiplicador en array
- Multiplicador carry save

Se ha supuesto para ello un retardo en los *Full Adder* fijo de 1 ns, tanto en la generación de la señal de suma como en la señal de carry. Esto nos permite observar el efecto de los distintos caminos críticos en el retardo en la generación de las señales.

5.1. Sumador Ripple Carry

Este sumador es el más sencillo en términos de implementación. Simplemente se han conectado 16 full adders de forma secuencial de forma que el acarreo se propague de uno a otro.

En la figura 5 se muestra un esquema simplificado (4 bits) de este sumador.

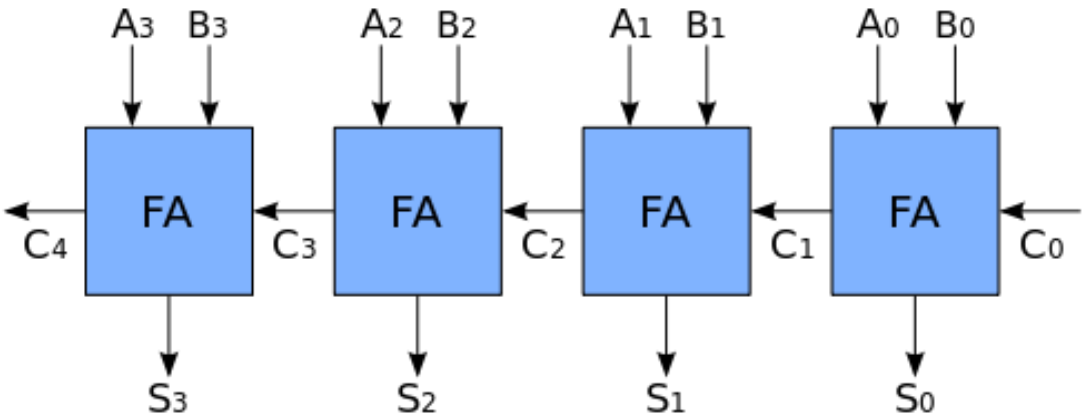

Figura 5: Sumador Ripple Carry de 4 bits

5.2. Sumador Carry Bypass

El sumador *Carry Bypass* (o *Carry Skip*) permite reducir el tiempo de retardo del camino crítico sensiblemente. En la figura 6 puede verse un esquema de este tipo de sumador.

Para la implementación de este tipo de sumadores, se divide la operación en varios bloques. En el esquema de la figura 6 estos bloques son de 4 bits, aunque se puede variar el tamaño de los mismos para optimizar aún más el retardo de este sumador.

En la figura 7 puede verse un detalle de los bloques carry bypass.

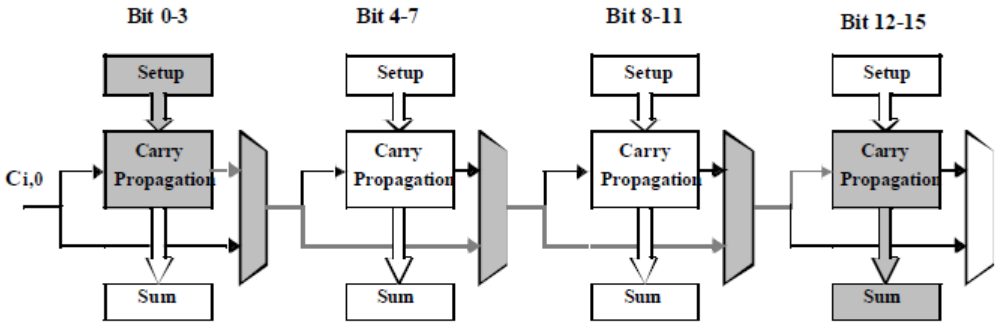

Figura 6: Sumador carry bypass

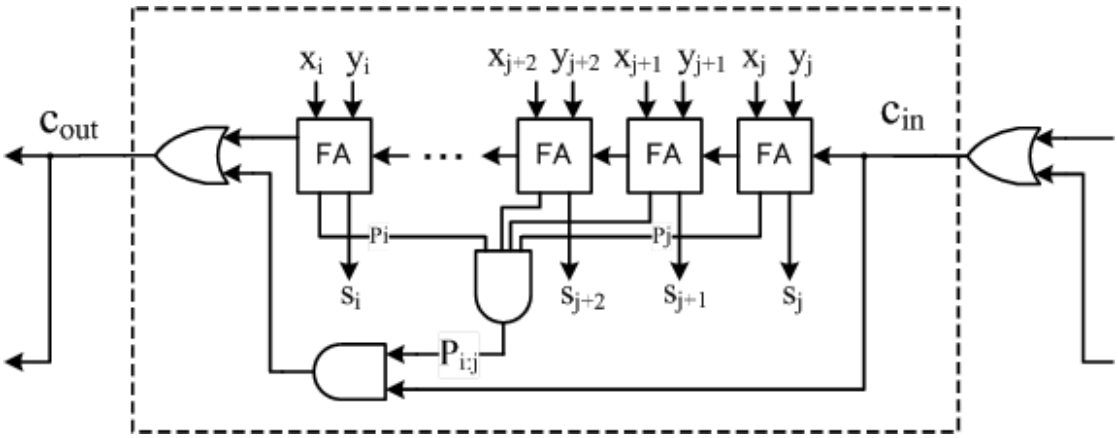

Figura 7: Bloque carry bypass

En nuestro diseño, se ha optado por un tamaño de bloque de 4 bits, lo que resulta en un total de 4 bloques carry bypass para obtener el sumador completo.

Para trabajar con las señales P y G que se han explicado en teoría, se ha implementado una versión modificada del full adder que trabaja con las señales P y G en lugar de los sumandos originales.

5.3. Multiplicador en array

El multiplicador en array corresponde a la implementación más directa de una multiplicación binaria. Para ello, simplemente se generan los productos parciales y se van sumando en distintas filas de sumadores. Esto conlleva un retardo considerablemente alto, que como podremos ver, se puede reducir mediante otras configuraciones de multiplicadores.

En la figura 8 se puede ver un esquema general de un multiplicador en array. Para un multiplicador de 16 x 16 bits como el que necesitamos en nuestro sistema, tendríamos que utilizar un total de 256 full adders. Debido a que esta cantidad es inmanejable a la hora de realizar conexiones de señales en VHDL, se ha optado por dividir la estructura del multiplicador en filas de 16 full adders, de forma que solo tengamos que diseñar una fila y después tratar con 16 filas. El diseño de estas filas (*bloques de multiplicador en array*) puede verse en la figura 9. Es importante destacar que esta agrupación en filas no conlleva ninguna modificación en el funcionamiento del multiplicador, y que equivale exactamente a haber conectado entre si los 256 full adders en un único módulo.

Para realizar multiplicaciones con signo, se calcula previamente el signo del resultado mediante un XOR de los MSB de los factores. Posteriormente se realiza la multiplicación del valor absoluto de los factores, invirtiéndose posteriormente el signo del resultado si se espera un resultado negativo.

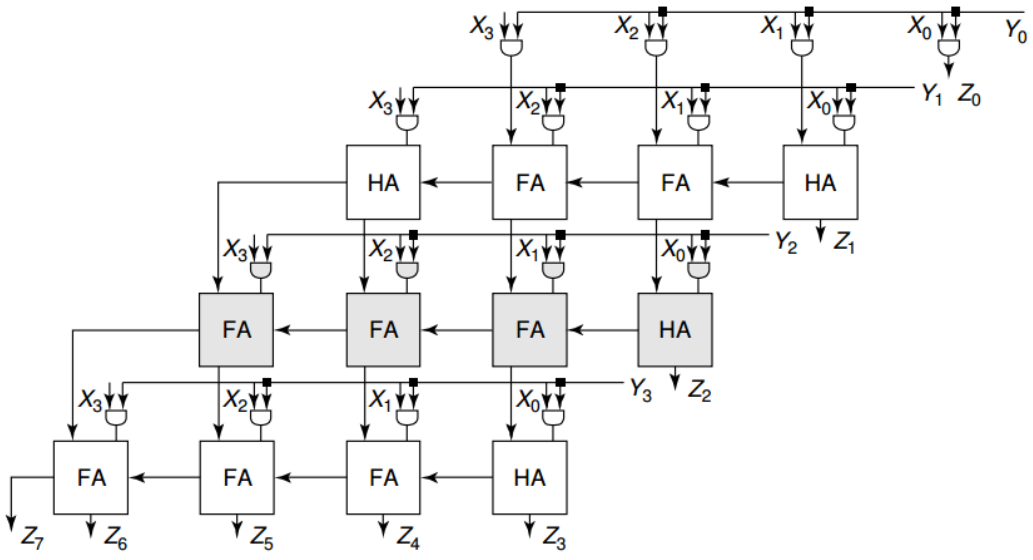

Figura 8: Multiplicador en array

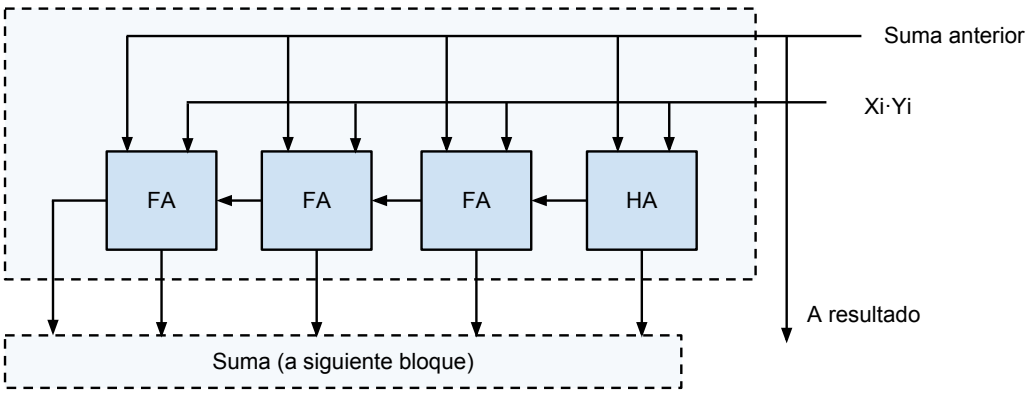

Figura 9: Bloque de multiplicador en array

5.4. Multiplicador carry save

El multiplicador carry save es una estrucutra de multiplicador modificada que permite reducir el tiempo de retardo en la generación del producto. Para ello, los acarreos de una fila de full adders no son propagados dentro de una misma fila, sino que se propagan a la fila siguiente como un sumando adicional.

El esquema de este tipo de multiplicador puede verse en la figura 10.

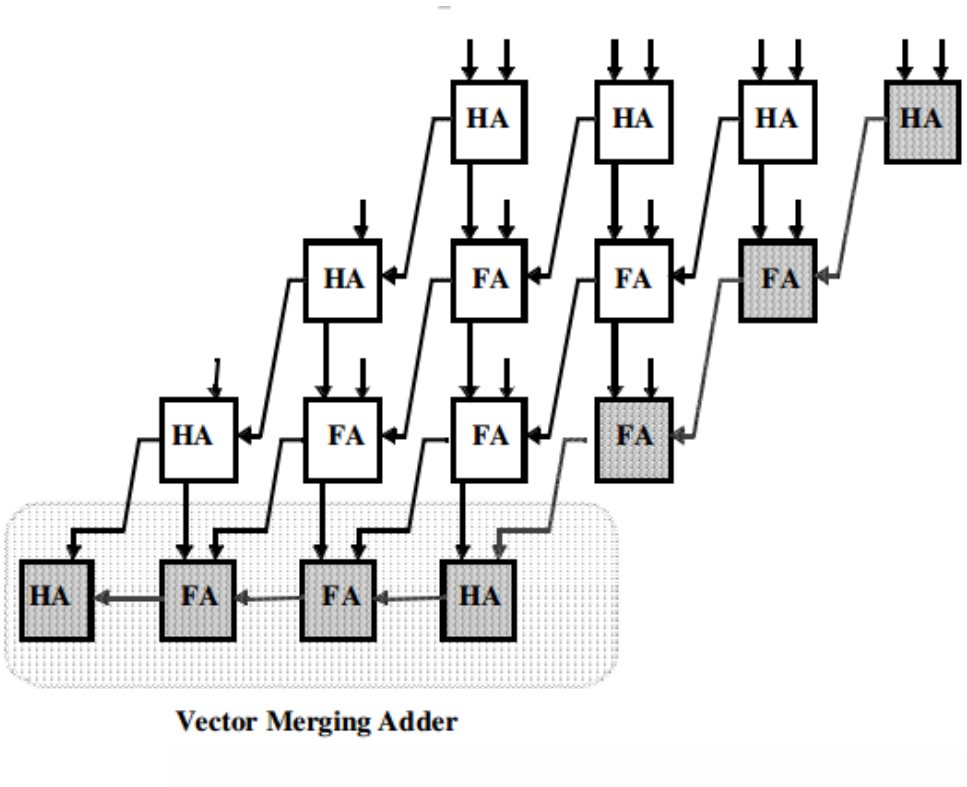

Figura 10: Multiplicador carry save

De forma análoga a lo que sucede con el multiplicador en array, realizar un multiplicador carry save de 16x16 bits conlleva la utilización de un número inmanejable de full adders. Para evitar este problema, se ha optado por una solución similar a la tomada para la implementación del multiplicador en array: dividir el sistema en bloques para cada fila de full adders.

El diseño elegido para estos bloques puede verse en la figura 11

Para realizar multiplicaciones con signo, se calcula previamente el signo del resultado mediante un XOR de los MSB de los factores. Posteriormente se realiza la multiplicación del valor absoluto de los factores, invirtiéndose posteriormente el signo del resultado si se espera un resultado negativo.

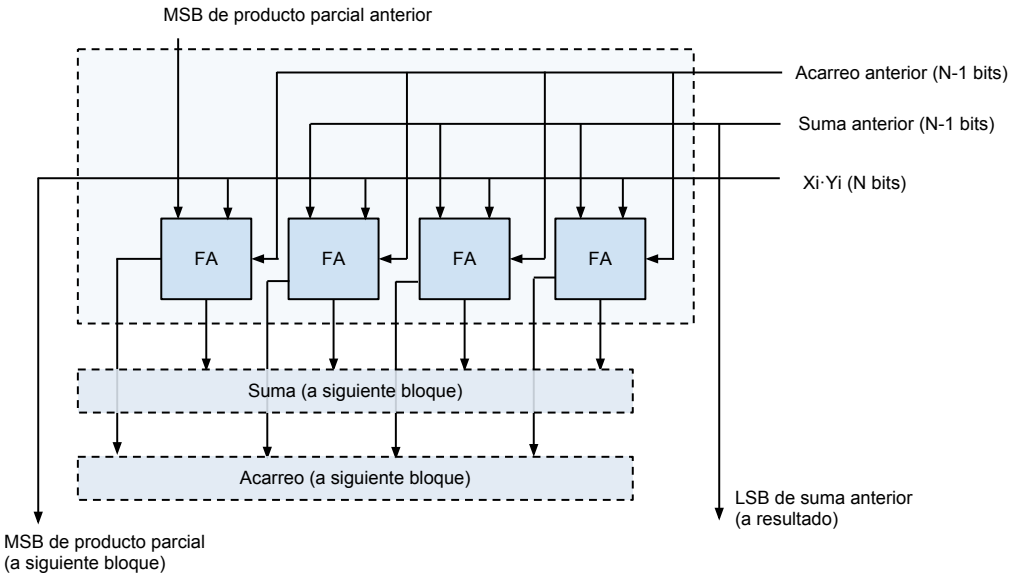

Figura 11: Bloque del multiplicador carry save

6. Pruebas y resultados

En esta sección se van a describir los procedimientos utilizados para evaluar el funcionamiento de los sistemas implementados

6.1. Comprobación del funcionamiento de los filtros

Para comprobar el correcto funcionamiento de los filtros, se ha comparado la respuesta al impulso obtenida en la simulación del sistema VHDL con la respuesta al impulso teórica que debería tener un filtro IIR con los coeficientes indicados en las especificaciones.

Es importante tener en cuenta que al aplicar la atenuación a la salida de los filtros (para que todas las bandas presenten una ganancia de 0dB) se aumenta considerablemente el ruido de cuantificación de la señal de salida. Para evitar este efecto a la hora de comprobar la correcta realización del filtro, se van a comparar las respuestas al impulso sin aplicar la atenuación en primer lugar, y posteriormente aplicándola.

Las gráficas detalladas con los resultados de esta prueba se encuentran en el apéndice A.

Observando los resultados, se puede ver que las respuestas al impulso que se observa sin atenuar la salida de los filtros corresponde bastante bien con la respuesta teórica esperada (calculada con MATLAB). No obstante, si observamos las respuestas al impulso con la salida atenuada, vemos que el ruido de cuantificación es considerablemente alto, especialmente en los filtros de frecuencias bajas.

Debido a que esta atenuación se aplica a la entrada (para evitar saturación en puntos intermedios) este ruido nos afecta también a los valores de realimentación dentro de la estructura del filtro, lo que nos modifica la forma de la respuesta al impulso en los filtros 0 y 1.

El diseño de los filtros es correcto, como se observa en las respuestas al impulso sin atenuar. El funcionamiento pobre de los filtros 0 y 1 se debe por tanto únicamente al error de cuantificación, y por tanto para mejorar el funcionamiento en este aspecto debería trabajarse con un mayor tamaño de palabra.

6.2. Prueba de la función de reverb

Para comprobar el funcionamiento de la función de reverb se ha realizado una prueba sencilla. Se ha generado un estímulo de entrada de corta duración (un periodo de una señal sinusoidal) y se ha ajustado el número de muestras del retardo de reverberación para que el *eco* se produzca justo después de finalizar la señal original, pudiendo así apreciarse mejor.

Se ha comparado la señal obtenida con la función de reverb activada con la obtenida con la función de reverb desactivada. Mientras que en la señal filtrada sin reverb se pueden apreciar únicamente pequeñas oscilaciones derivadas de la respuesta de los filtros tras el ciclo sinusoidal de la señal original, en la señal con reverb se pueden apreciar claramente réplicas atenuadas de este ciclo a lo largo del tiempo.

Las gráficas que muestran este resultado pueden verse en el apéndice B.

6.3. Prueba de ajuste de ganancia

Para comprobar el funcionamiento de la función de ajuste de ganancia se ha observado la evolución de la señal de salida mientras se va modificando la ganancia. Para ilustrar este funcionamiento, se ha tomado una señal sinusoidal centrada en la frecuencia del filtro 3, y se ha ido aumentando la atenuación progresivamente en las bandas 2, 3 y 4. En la salida puede verse claramente como la señal va atenuándose a medida que vamos variando la ganancia.

Observar de forma gráfica la evaluación temporal de la señal es la forma más adecuada de comprobar este funcionamiento. Las gráficas que muestran este resultado pueden verse en el apéndice C.

6.4. Prueba del vúmetro

El vúmetro es un subsistema enormemente sencillo. Debido a ello, para la comprobación de su funcionamiento es suficiente con una comprobación cualitativa de las señales de nivel generadas.

Para comprobar el funcionamiento, se han generado impulsos de diferente amplitud y comprobado la respuesta del vúmetro a ellos. Se ha comprobado tanto el mantenimiento temporal de los picos como la adecuación de la señal de nivel ante señales tanto positivas como negativas.

Debido a la sencillez del procedimiento, una simple observación en la simulación es suficiente y no es necesaria la generación de gráficas.

6.5. Prueba de mejoras

Para comprobar el funcionamiento de los sumadores y multiplicadores se han realizado varias operaciones y se ha comparado el resultado que ofrecían nuestros sumadores y multiplicadores con el que resultaría de utilizar las funciones + y * de VHDL. Se pretende así comprobar que el uso de nuestros bloques operacionales corresponde exactamente al uso de las funciones VHDL. Para ello se ha elegido un conjunto de señales de prueba a sumar, teniendo cuidado en comprobar las operaciones con señales negativas.

Los resultados son satisfactorios y se obtienen en todo caso los valores esperados.

En las gráficas incluidas en el apéndice D puede observarse la comparación de estas señales.

Observación sobre retardos

Como es de esperar, al utilizar nuestros sumadores y multiplicadores implementados con Full Adders con retardo, observamos los efectos de la propagación de las señales de suma y de acarreo. Esto se traduce en una oscilación en los valores del resultado que ofrecen nuestros bloques, que tarda unos nanosegundos en estabilizarse y ofrecer el valor correcto.

Este efecto se elimina del resto del sistema mediante el uso de una señal de reloj. Utilizando un periodo de reloj mayor que el necesario para la generación de las sumas y multiplicaciones evitamos estas oscilaciones. En la simulación se ha establecido un periodo de reloj de 100ns, suficientemente alto ya que el retardo de los full adders es de 1 ns.

En las gráficas incluidas en el apéndice D puede además verse la diferencia en el retardo de las diferentes configuraciones. Así, observamos como el sumador ripple carry en muchas ocasiones tarda sensiblemente más en estabilizar su resultado que el sumador carry bypass. También se observa esta diferencia en los multiplicadores. El multiplicador carry save tiene un retardo sensiblemente mejor que el del multiplicador en array. Esto corresponde con lo estudiado en teoría en clase.

7. Problemas encontrados y comentarios finales

La realización de este proyecto ha resultado enormemente didáctica para el aprendizaje tanto del lenguaje de programación VHDL como del uso de herramientas de simulación (Modelsim). Además, en su realización no sólo se han aprendido conceptos relativos a estas propias herramientas sino que también se ha podido comprobar de primera mano el funcionamiento de algunos de los sistemas estudiados en clase de forma teórica. Esto ha supuesto una experiencia de aprendizaje muy gratificante en ambos sentidos, que ha permitido una comprensión de los módulos estudiados mucho más profunda que mediante su estudio teórico.

En la realización de este proyecto, se han encontrado ciertas dificultades que han podido ser superadas con cierto esfuerzo. En primer lugar, las peculiaridades del lenguaje VHDL hacen que comenzar a trabajar sea más costoso de lo habitual. Más allá de las características de los sistemas con los que se han trabajado, muchas veces el punto de dificultad ha resultado venir dado por el propio lenguaje. No obstante, debido a la enorme cantidad de información disponible en internet, estos

obstáculos han podido ser superados sin ningún problema.

En segundo lugar, se han encontrado dificultades a la hora de escoger un ancho de bits adecuado para las señales que se manejan en el sistema. Tanto es así, que aunque se ha conseguido definir un ancho de palabra que evita problemas como desbordamientos, no se ha conseguido optimizar la elección de forma que se minimice el error de cuantificación. No obstante, puesto que la calidad de sonido no es el objetivo de esta práctica, se ha considerado que el error de cuantificación presente en el sistema es asumible. Aumentar el ancho de palabra a 32 bits sería una solución aceptable para minimizar este error, pero aumentaría considerablemente la complejidad de los sumadores y multiplicadores desarrollados como mejoras. Por tanto, a pesar de conllevar un error de cuantificación considerable, se ha decidido mantener la elección de 6 bits enteros y 10 bits fraccionarios.

En los demás aspectos, se ha podido diseñar e implementar el resto de subsistemas sin dificultades destacables.

En conclusión, se ha desarrollado un sistema funcional y altamente didáctico.

Apéndices

A. Gráficas de evaluación de filtros

Estas gráficas corresponden a los resultados de los procedimientos descritos en la sección 6.1

A.1. Sin aplicar atenuación

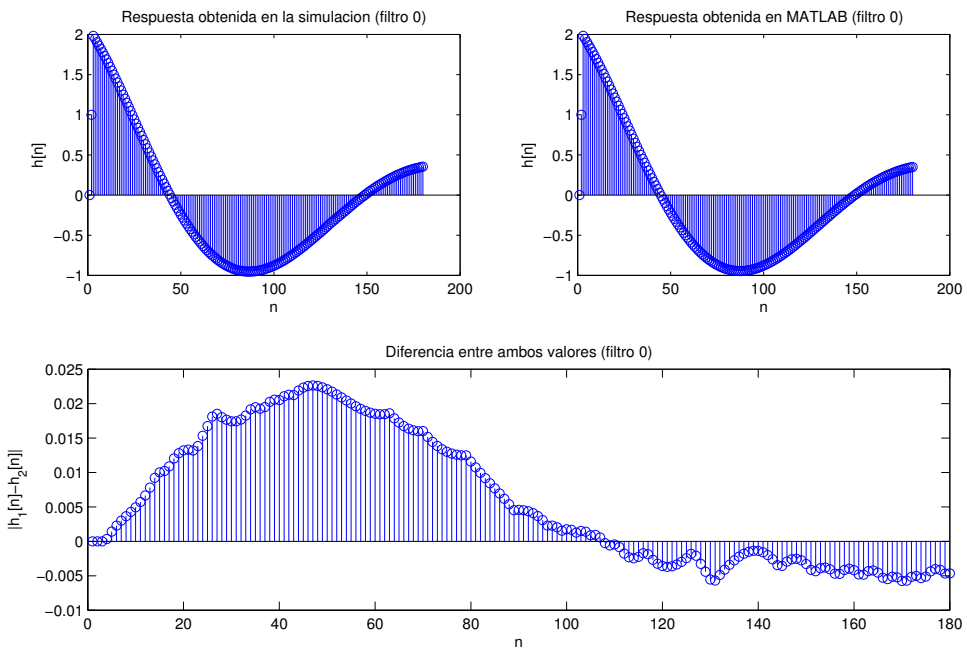

Figura 12: Comprobación del filtro 0

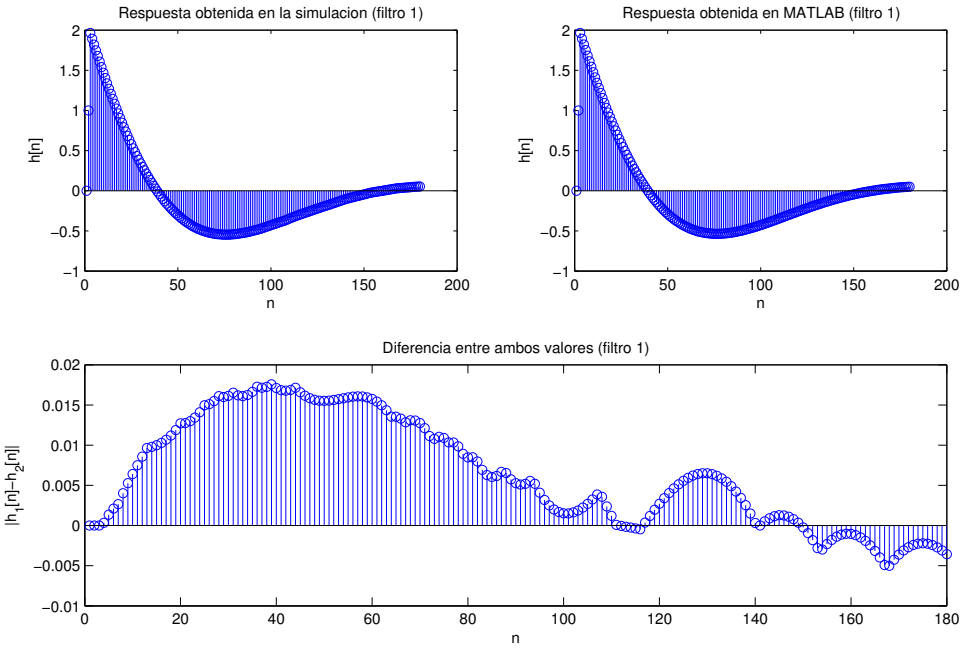

Figura 13: Comprobación del filtro 1

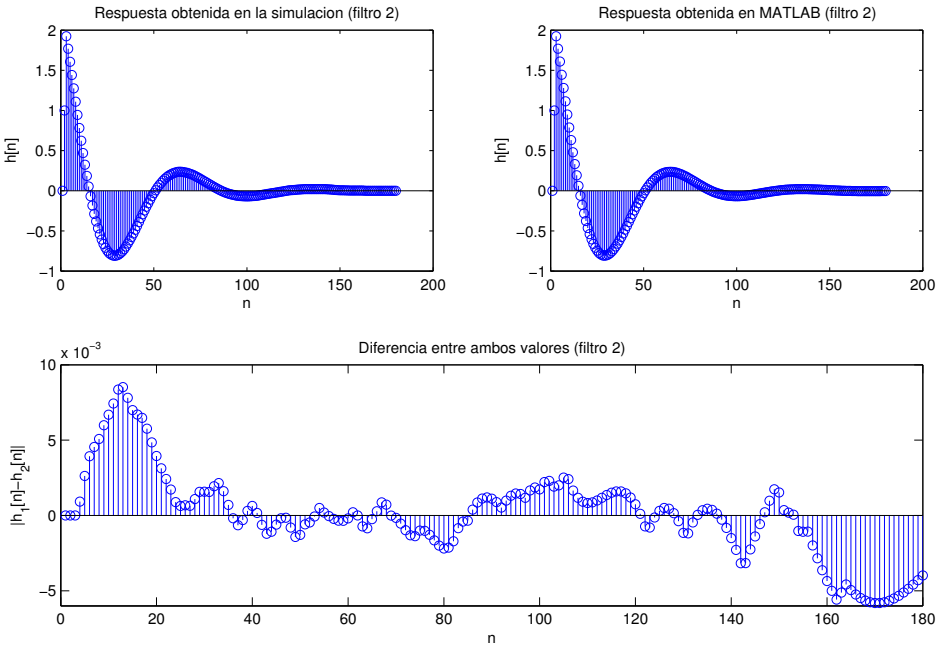

Figura 14: Comprobación del filtro 2

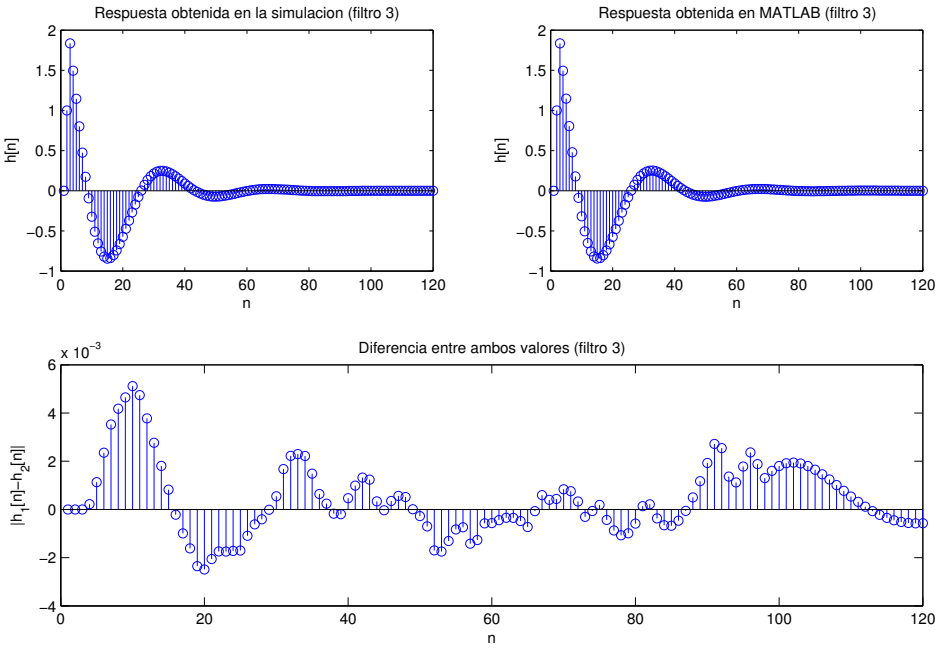

Figura 15: Comprobación del filtro 3

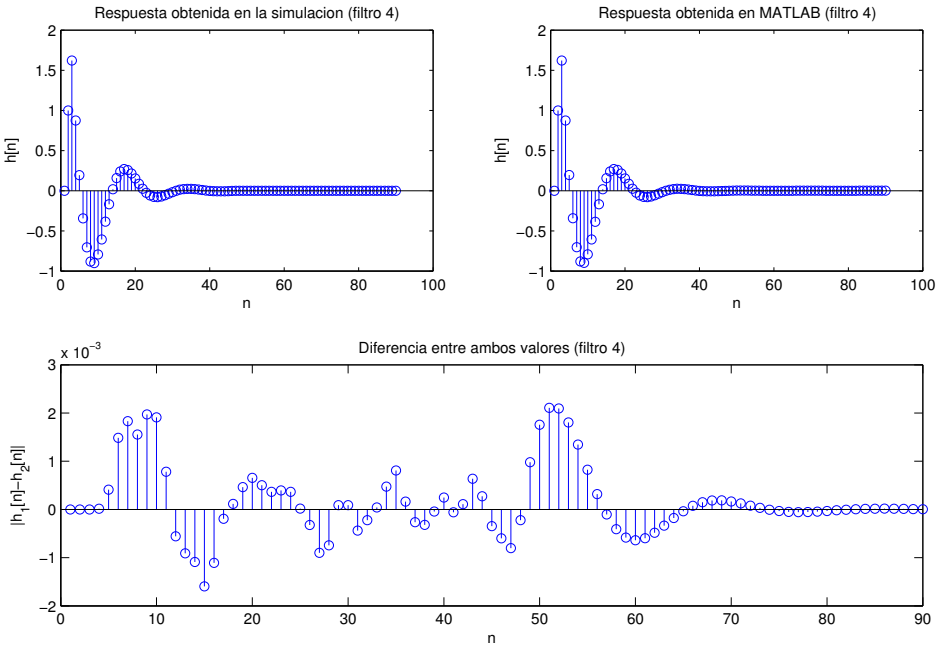

Figura 16: Comprobación del filtro 4

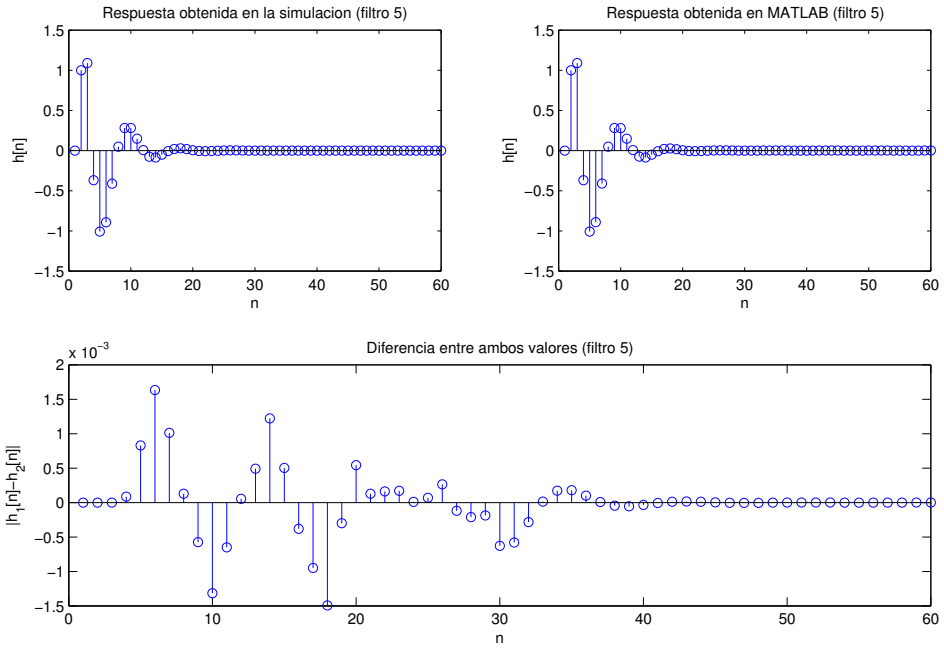

Figura 17: Comprobación del filtro 5

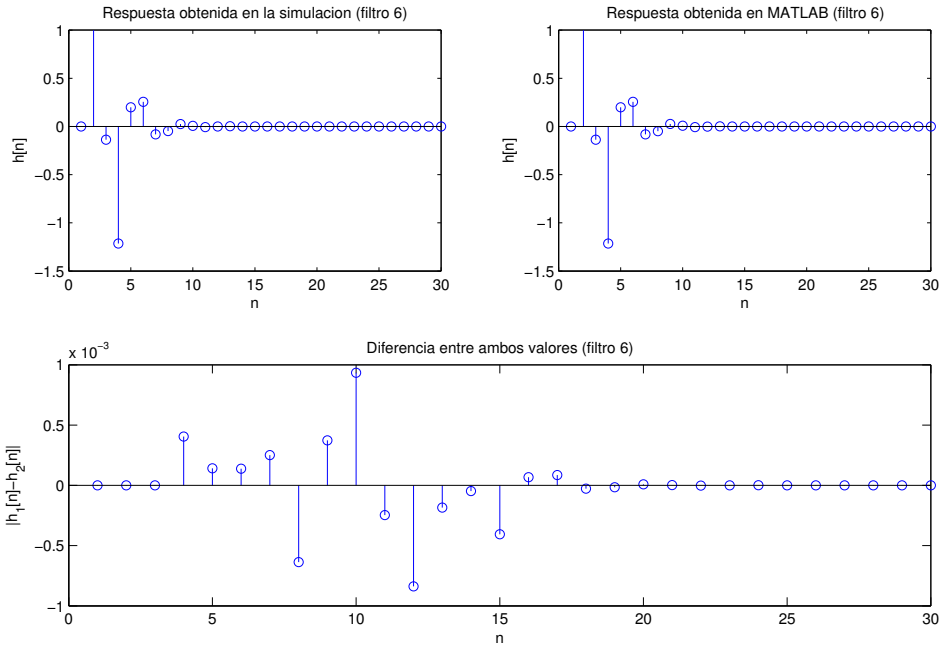

Figura 18: Comprobación del filtro 6

A.2. Aplicando atenuación

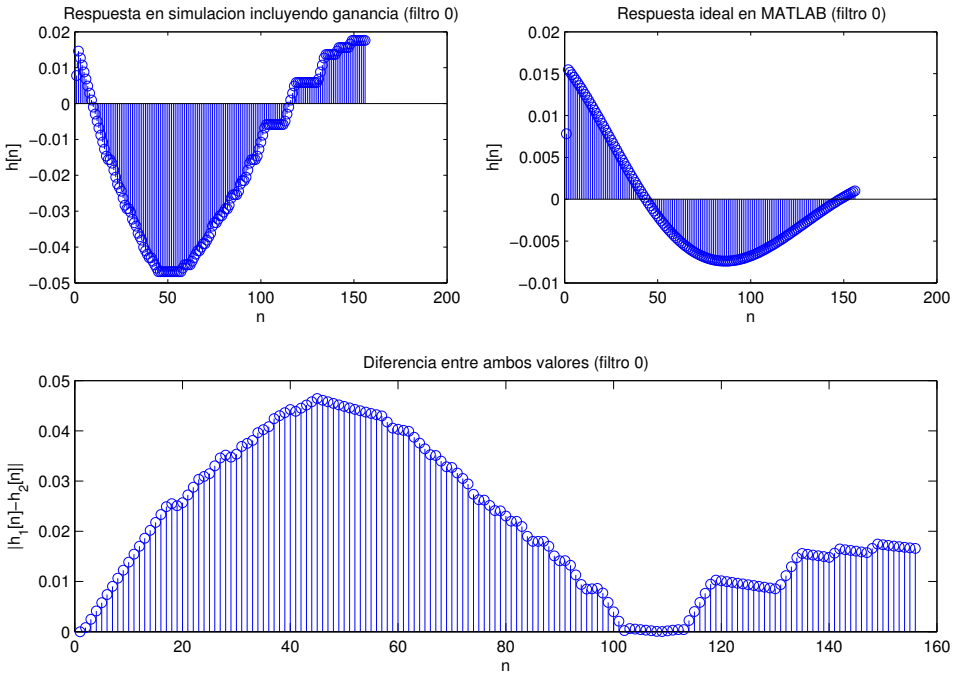

Figura 19: Comprobación del filtro 0 (con atenuación)

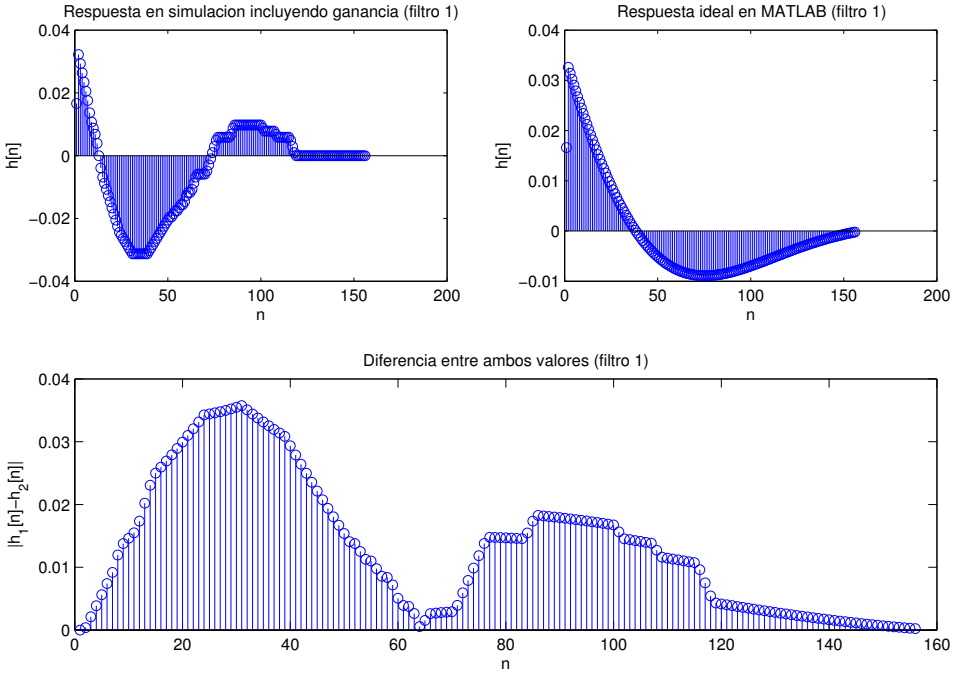

Figura 20: Comprobación del filtro 1 (con atenuación)

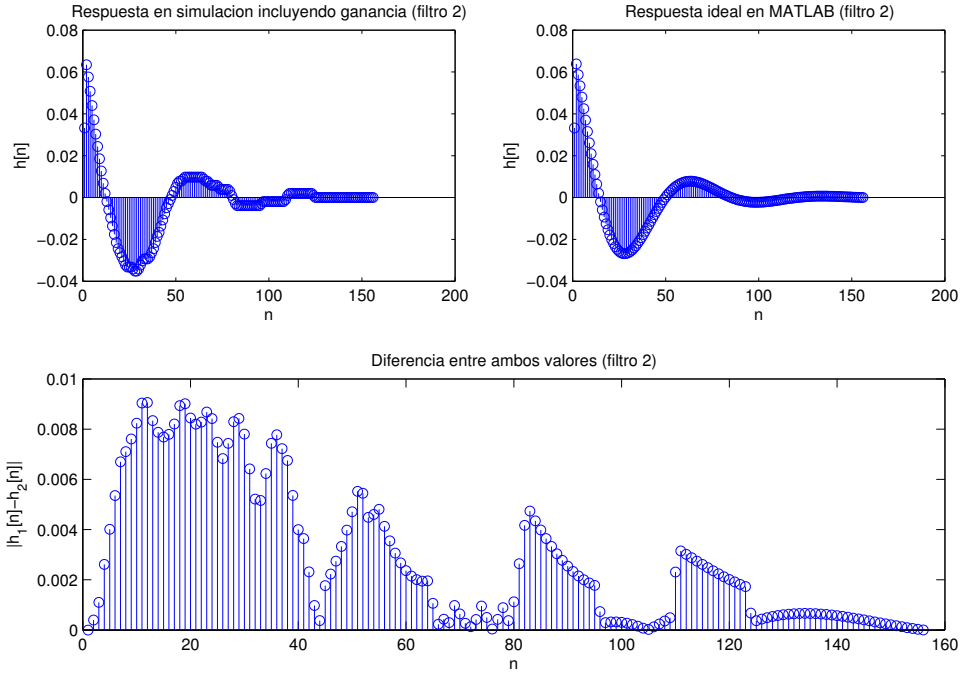

Figura 21: Comprobación del filtro 2 (con atenuación)

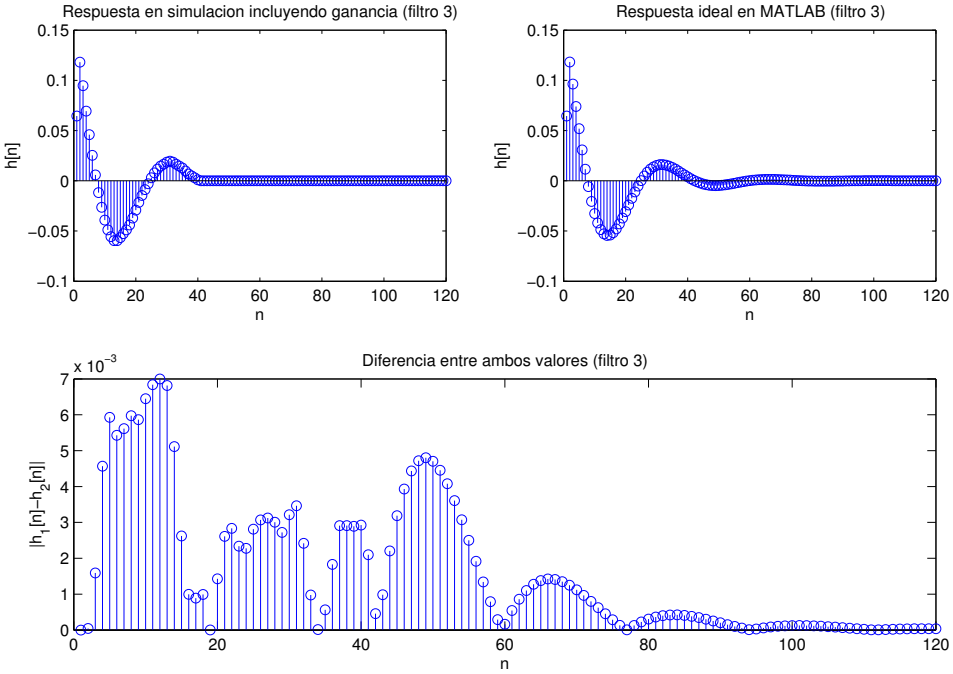

Figura 22: Comprobación del filtro 3 (con atenuación)

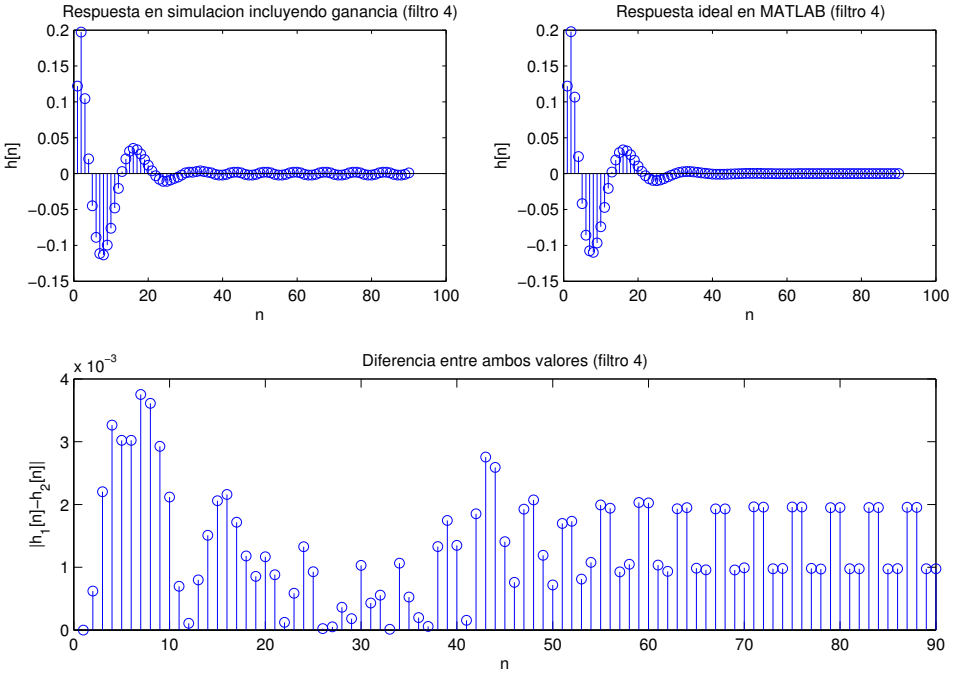

Figura 23: Comprobación del filtro 4 (con atenuación)

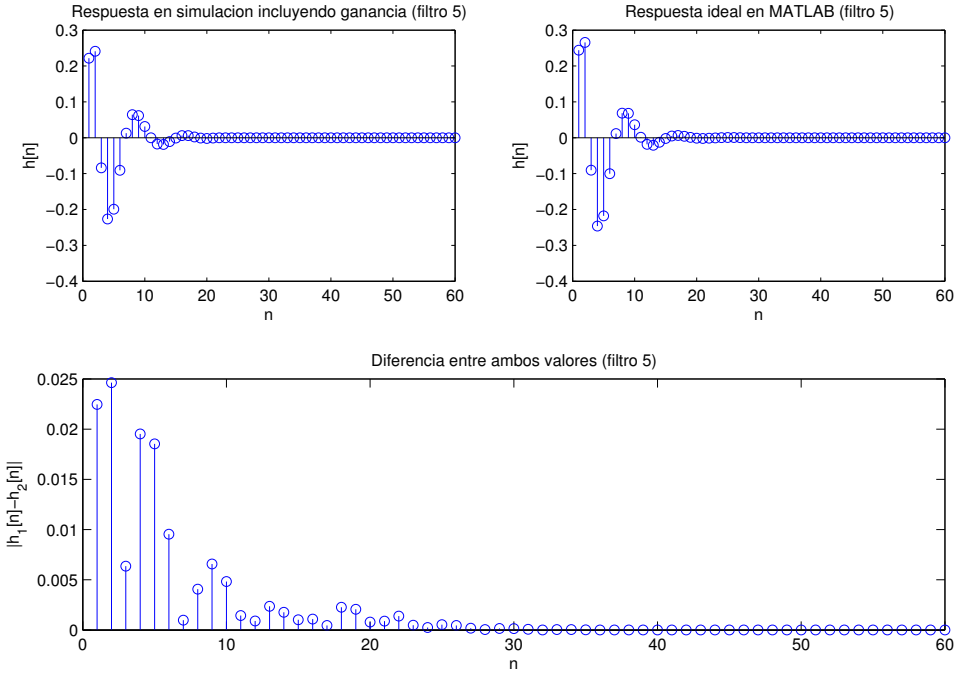

Figura 24: Comprobación del filtro 5 (con atenuación)

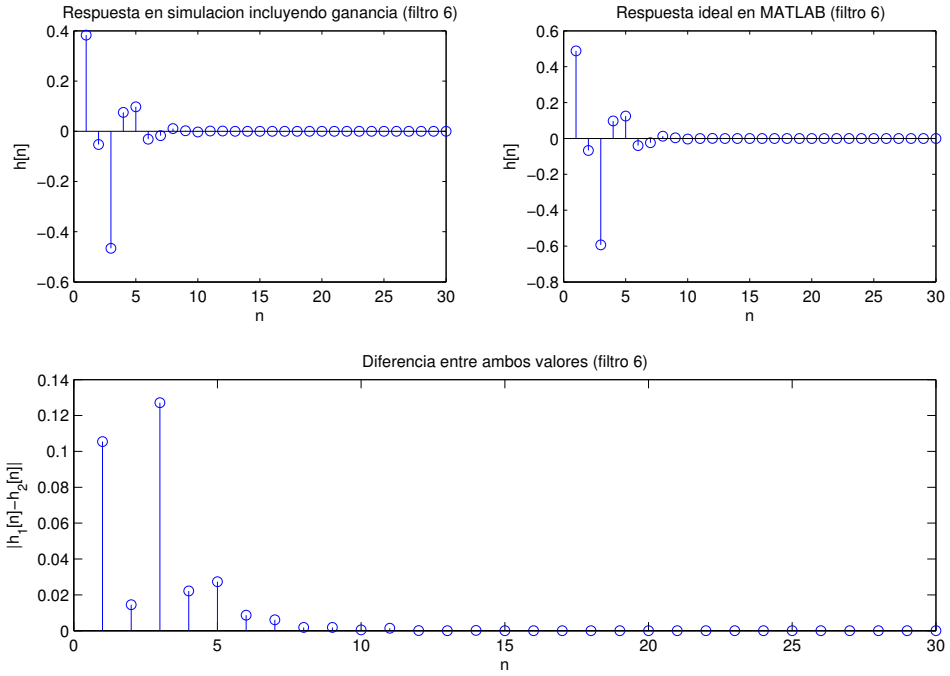

Figura 25: Comprobación del filtro 6 (con atenuación)

B. Gráficas de evaluación de reverb

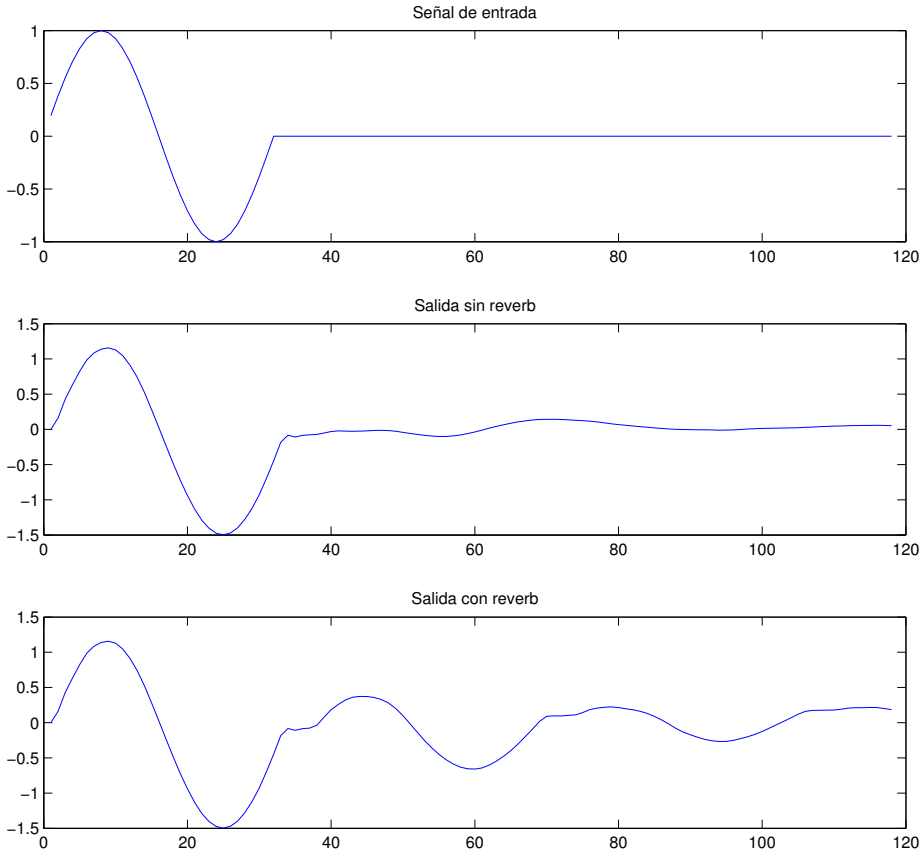

Figura 26: Comprobación de la función reverb

C. Gráficas de evaluación de ajuste de ganancia

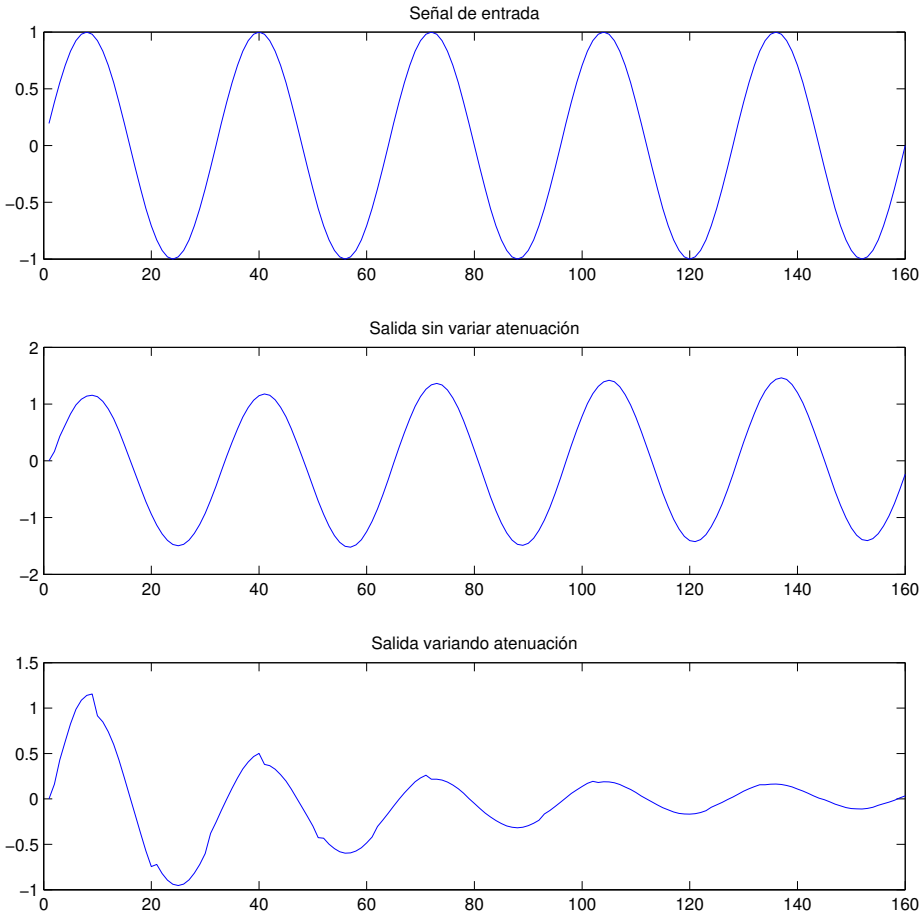

Figura 27: Comprobación del ajuste de ganancia

D. Gráficas de prueba de mejoras

D.1. Sumadores

En la gráfica que se presenta a continuación se pueden observar las siguientes señales:

Valid_rc Esta señal se pone a nivel alto cuando el resultado del sumador ripple carry es válido.

Valid_cb Esta señal se pone a nivel alto cuando el resultado del sumador carry bypass es válido.

sig1 Término 1 de la suma.

sig2 Término 2 de la suma.

sout_rc Resultado del sumador ripple carry.

sout_cb Resultado del sumador carry bypass.

sout_ts Resultado de la función + VHDL

En las señales valid_rc y valid_cb se observa la diferencia entre los tiempos de retardo de ambos sumadores.

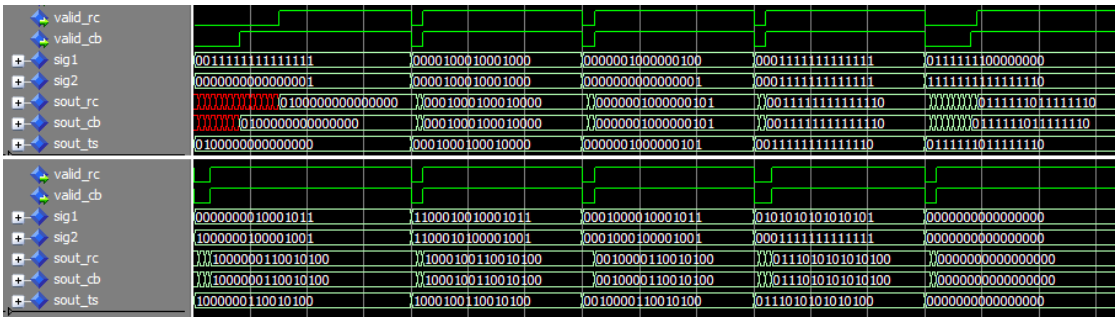

Figura 28: Comprobación de sumadores

D.2. Multiplicadores

En la gráfica que se presenta a continuación se pueden observar las siguientes señales:

Valid_ar Esta señal se pone a nivel alto cuando el resultado del multiplicador en array es válido.

Valid_cs Esta señal se pone a nivel alto cuando el resultado del multiplicador carry save es válido.

signal1 Factor de la multiplicación.

signal2 Factor de la multiplicación.

mult_cs Resultado del multiplicador carry save.

mult_ar Resultado del multiplicador en array.

mult_test Resultado de la función * VHDL

En las señales valid_ar y valid_cs se observa la diferencia entre los tiempos de retardo de ambos multiplicadores.

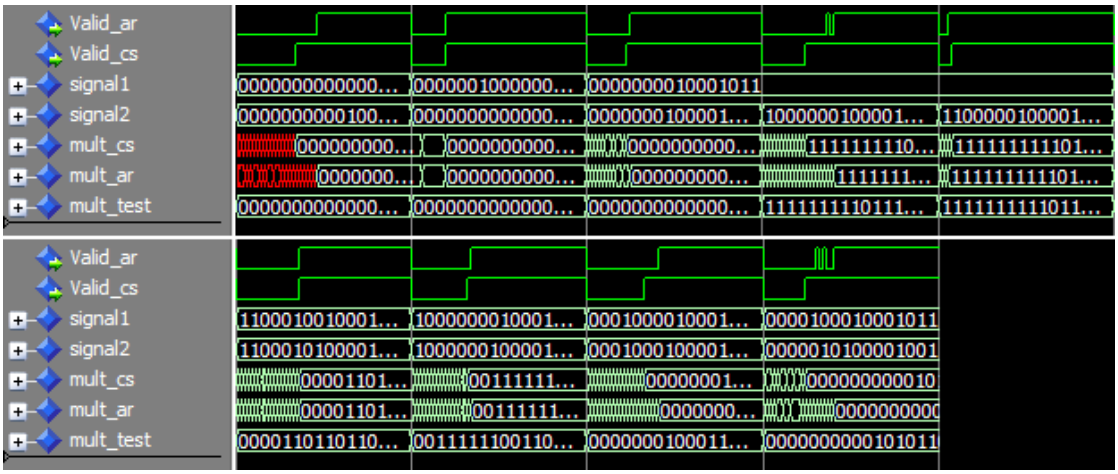

Figura 29: Comprobación de multiplicadores

E. Relación de ficheros VHDL

Las siguientes tablas muestran una lista de los archivos .vhd implementados junto a su descripción. Se han mantenido tanto las versiones con mejoras (sumadores y multiplicadores reales) como sin mejoras, con el fin de poder comparar su funcionamiento o entender mejor su funcionalidad.

Sistema global

Archivo VHD	Descripción
equalizer_system.vhd	Sistema global integrado
equalizer_system_m.vhd	Sistema global integrado con mejoras

Subsistemas

Archivo VHD	Descripción
delay.vhd	Módulo de retardo
filter_bench_m.vhd	Banco de filtros (con mejoras)
filter_bench.vhd	Banco de filtros (sin mejoras)
filter_generic_m.vhd	Filtro genérico (con mejoras)
filter_generic.vhd	Filtro genérico (sin mejoras)
reverb_m.vhd	Subsistema de reverb (con mejoras)
reverb.vhd	Subsistema de reverb (sin mejoras)
vumetro_it.vhd	Vúmetro individual
vumetro.vhd	Vúmetro de 7 señales

Mejoras

Archivo VHD	Descripción
adder_16b.vhd	Sumador de 16 bits (incluye las dos arquitecturas)
multiplier_16x16b.vhd	Multiplicador de 16 bits (incluye las dos arquitecturas)
am_block.vhd	Bloque del multiplicador en array
carry_bypass_block.vhd	Bloque de sumador carry bypass
csm_block.vhd	Bloque de multiplicador carry save
csm_lastblock.vhd	Último bloque de multiplicador carry save
FA_m.vhd	Full adder modificado (señales P y G)
FA.vhd	Full adder

Testbench

Archivo VHD	Descripción
tadder_16b.vhd	Pruebas del sumador (ambas arquitecturas)
tdelay.vhd	Pruebas del retardador
tequalizer_system.vhd	Pruebas del sistema global (con y sin mejoras)
tfilter_bench.vhd	Pruebas del banco de filtros (con y sin mejoras)
tfilter_generic.vhd	Pruebas del filtro genérico (con y sin mejoras)
tfilter_impz.vhd	Testbench para obtener respuestas al impulso a fichero
tmultiplier_16x16b.vhd	Pruebas del multiplicador (ambas arquitecturas)
treverb_m.vhd	Pruebas del módulo de reverb
tsystem_adjust.vhd	Pruebas de la funcionalidad de ajustar ganancia
tsystem_reverb.vhd	Pruebas de la funcionalidad de reverb (a nivel global)
tvumetro_it.vhd	Pruebas del vúmetro individual

Módulos auxiliares

Archivo VHD	Descripción
filehandler.vhd	Módulo para leer y escribir señales en ficheros
txt_util.vhd	Funciones útiles para tratar con señales de ficheros

F. Relación de scripts de Matlab

La siguiente tabla muestra una lista de los scripts Matlab utilizados en la comprobación de los resultados.

Archivo .m	Descripción
check_adjust.m	Comprobar funcionamiento de ajuste de ganancia
check_reverb.m	Comprobar funcionamiento de reverb
compare_filters.m	Comparar la respuestas al impulso (sin atenuación)
compare_filters_att.m	Comparar la respuestas al impulso (con atenuación)
gen_signal.m	Generar señales de estímulo
RdVHDL.m	Función de lectura de archivos VHDL
WrVHDL.m	Función de escritura de archivos VHDL