DCSE - Grupo especial

Trabajo con herramientas de diseño CAD-VHDL.

Memoria del proyecto

Madrid, Noviembre 2012

Autor:
Adrián Pérez Orozco

Índice general

Introducción

Descripción del proyecto

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.

El proyecto se ha estructurado en 5 hitos, correspondientes al diseño de cada uno de los subsistemas necesarios, de la forma siguiente:

Hito 1 Diseño y caracterización de filtros

Hito 2 Subsistema de ecualización

Hito 3 Reverberación

Hito 4 Integración global

Hito 5 Mejoras

Además, se propone un *hito 0* de introducción a VHDL, con el objetivo de familiarizarse con el lenguaje VHDL y las herramientas que se utilizarán para el desarrollo y simulación del sistema.

Para la realización del proyecto se utilizará la herramienta ModelSim. De forma auxiliar, se utilizará la herramienta MATLAB para obtener de forma más sencilla y eficiente información sobre las respuestas de los filtros tanto en el tiempo como en el dominio de la frecuencia.

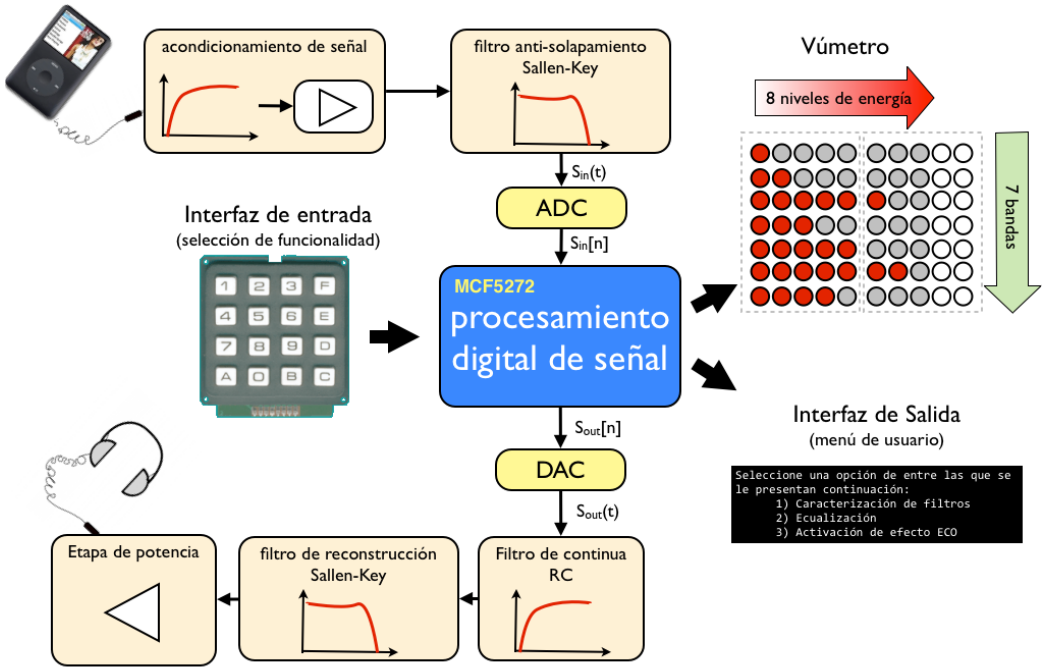

Figura 1: Descripcción del sistema completo

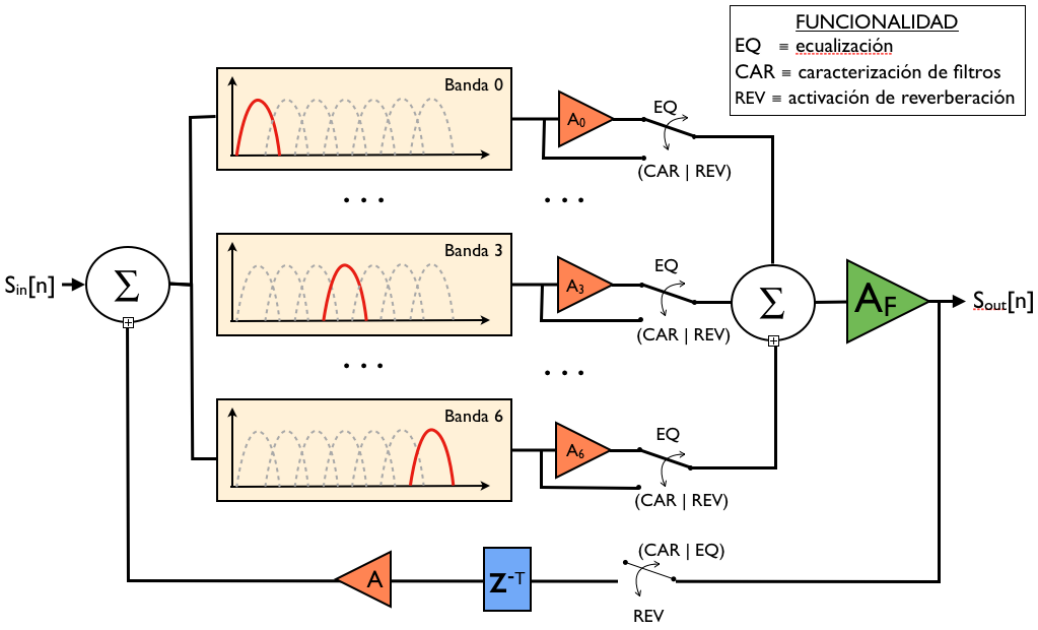

Figura 2: Descripcción del subsistema de procesado digital

Capítulo 1

Diseño y caracterización de filtros

En este capítulo vamos a detallar el proceso de diseño y caracterización de los filtros digitales, lo que corresponde al Hito 1 de nuestro proyecto.

1.1. Descripción de filtros para las distintas bandas de frecuencias

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 1.2, y los coeficientes para la realización de los filtros IIR en la tabla 1.2.

1.2. 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 1.1.

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

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 1.1: Descripción de las distintas bandas y sus filtros asociados

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 1.2: Descripción de los coeficientes de los filtros IIR

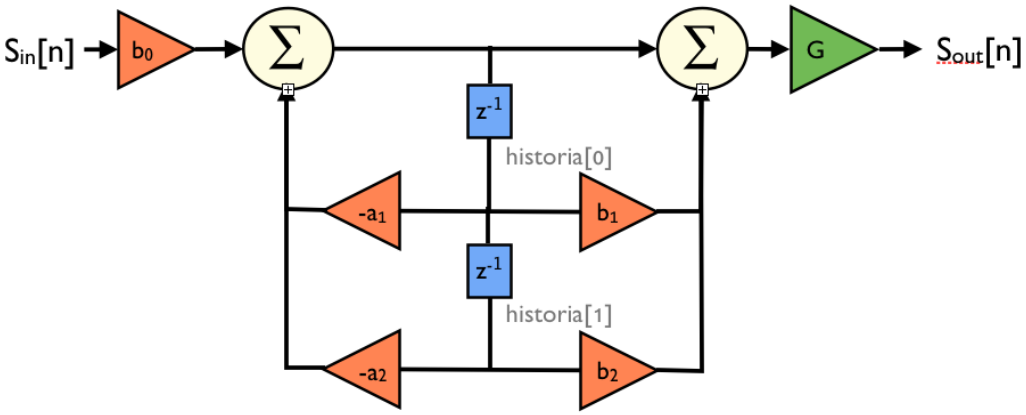

Figura 1.1: Diagrama de bloques de un filtro IIR de segundo orden

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 como fraccionarios para tener el coeficiente normalizado a 1024.

A continuación se muestra el código de la implementación de los filtros en VHDL. La implementación de los 7 filtros es idéntica, cambiando únicamente el valor de las constantes a_1, a_2 y G, ya que el resto de coeficientes son iguales en todos los casos.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity filterX is
port (
    sin : in std_logic_vector (15 downto 0);
    sout : out std_logic_vector (15 downto 0);
        clk : in bit);
end;

architecture filterarchX of filterX is
    signal historia0 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(0,16);
    signal historia1 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(0,16);
    constant b0 : std_logic_vector (15 downto 0)
```

```
                := conv_std_logic_vector(XXX,16);
    constant b1 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(XXX,16);
    constant b2 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(XXX,16);
    constant a1 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(XXX,16);
    constant a2 : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(XXX,16);
    constant gs : std_logic_vector (15 downto 0)
                := conv_std_logic_vector(XXX,16);

begin
    filter_proc: process(clk)
        variable sum1 : std_logic_vector (31 downto 0);
        variable sum2 : std_logic_vector (31 downto 0);
        variable mult : std_logic_vector (31 downto 0);

    begin
        if (clk'event and clk = '1') then
            sum1 := b0*sin - a1*historia0 - a2*historia1;
            sum2 := sum1 + b1*historia0 + b2*historia1;
            historia1 <= historia0;
            historia0 <= sum1(25 downto 10);
            mult := gs*sum2(25 downto 10);
            sout <= mult(25 downto 10);
        end if;
    end process;
end filterarchX;
```

1.3. Comprobación de la validez de los filtros

Finalmente, tras haber implementado los 7 filtros en VHDL, debemos comprobar mediante simulación que su funcionamiento es el correcto. Para ello, nos ayudaremos de la herramienta *MATLAB*, que nos permite fácilmente realizar filtros dados los coeficientes a y b de los que disponemos.

Para comprobar que el funcionamiento de nuestros filtros es el correcto, obtendremos sus respuestas al impulso colocando en la simulación a la entrada una función $\delta[n]$. Realizaremos la misma operación en MATLAB y comprobaremos que ambas respuestas sean iguales.

En las figuras 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 y 1.8 se pueden ver las comprobaciones realizadas. Para cada filtro, se ha representado: La respuesta al impulso obtenida en la simulación con ModelSim (arriba a la izquierda);

la respuesta al impulso obtenida con MATLAB (arriba a la derecha); y la diferencia en valor absoluto de ambas señales (debajo).

Para todas ellas, se puede apreciar que la diferencia es de un valor cientos de veces menor que el de la señal. Esto se debe al error de cuantificación introducido al limitar nuestro sistema a palabras de 16 bits con 10 bits fraccionarios. El error se va acumulando tras las diferentes iteraciones del filtro, y se empieza a compensar cuando la respuesta al impulso es negativa y el error se produce con el signo contrario.

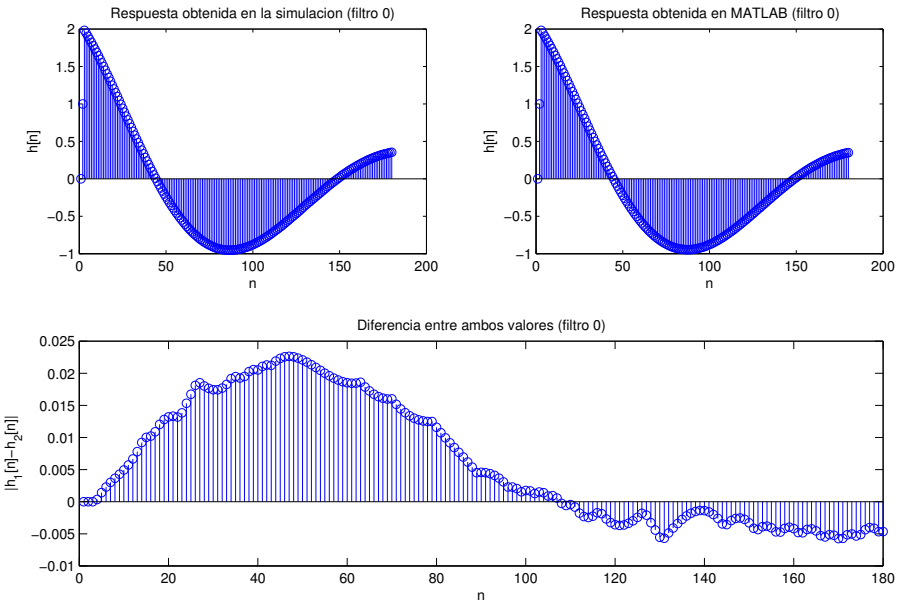

Figura 1.2: Comprobación del filtro 0

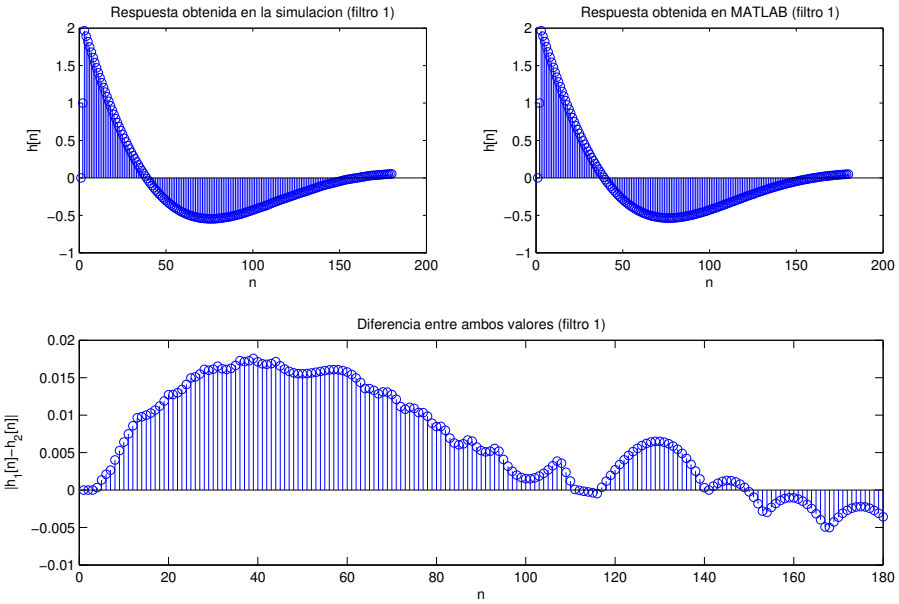

Figura 1.3: Comprobación del filtro 1

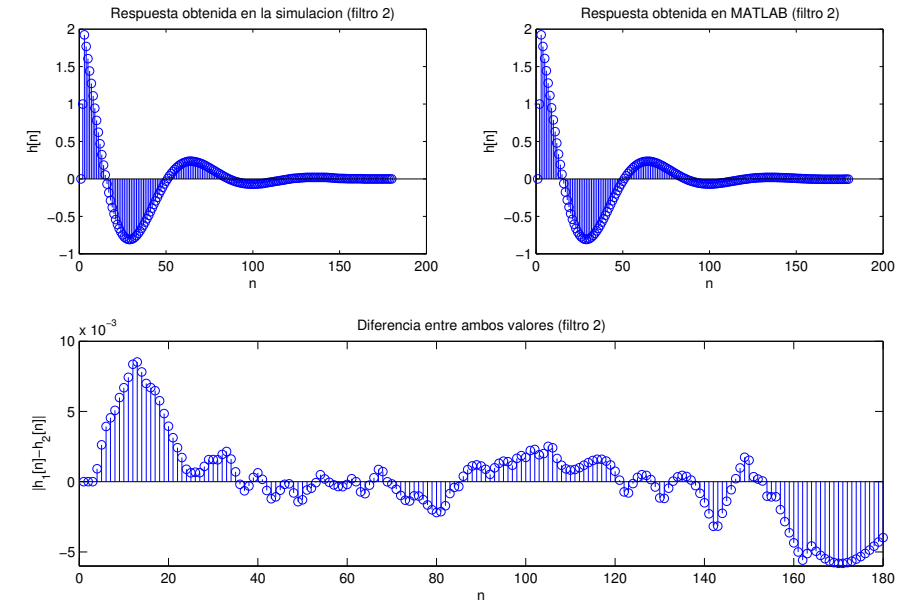

Figura 1.4: Comprobación del filtro 2

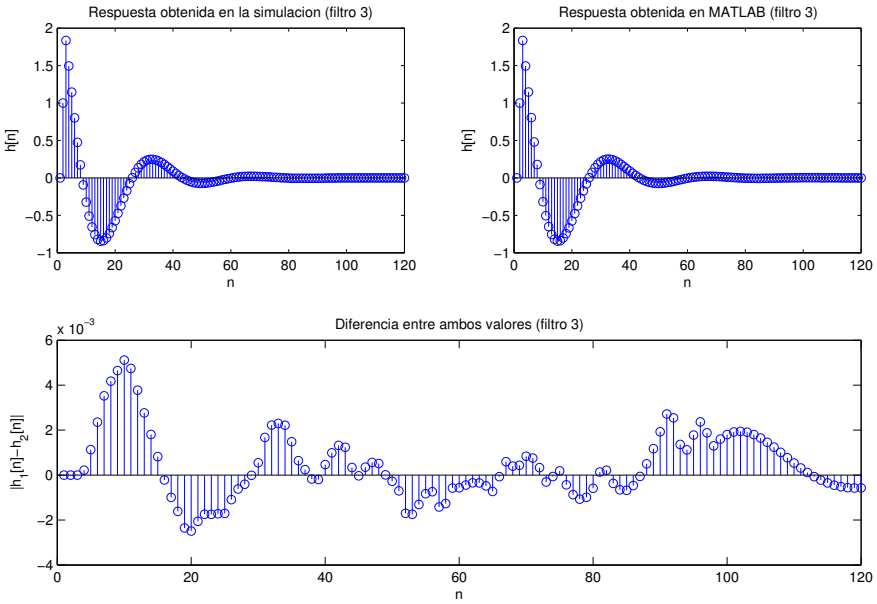

Figura 1.5: Comprobación del filtro 3

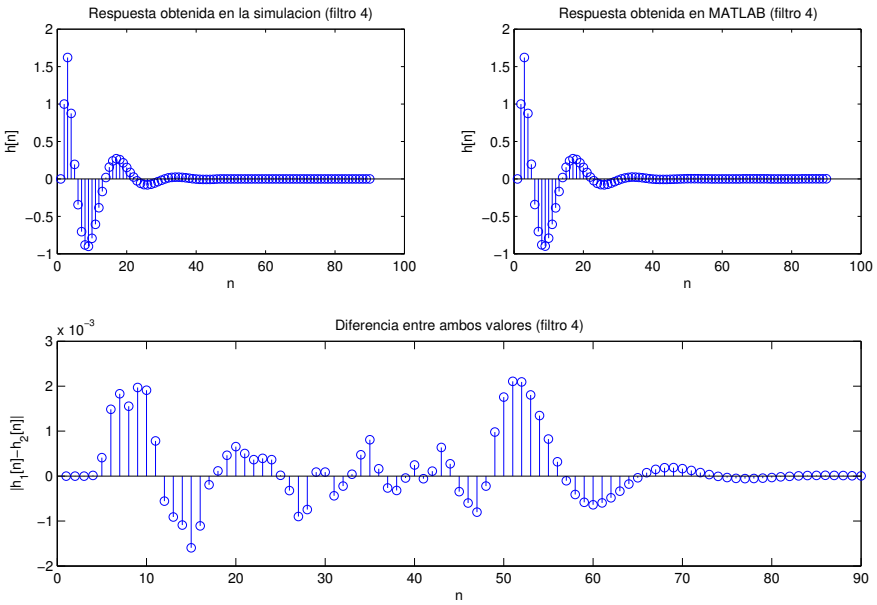

Figura 1.6: Comprobación del filtro 4

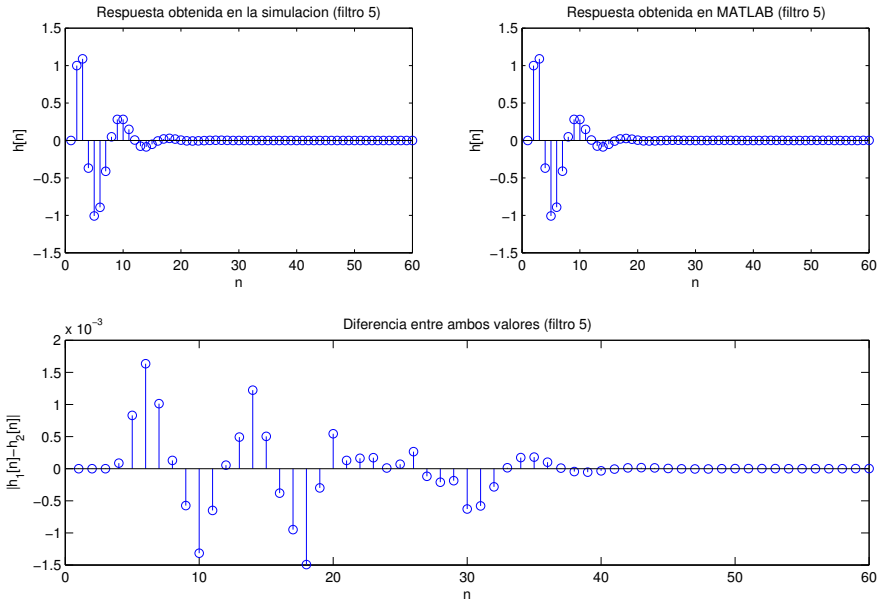

Figura 1.7: Comprobación del filtro 5

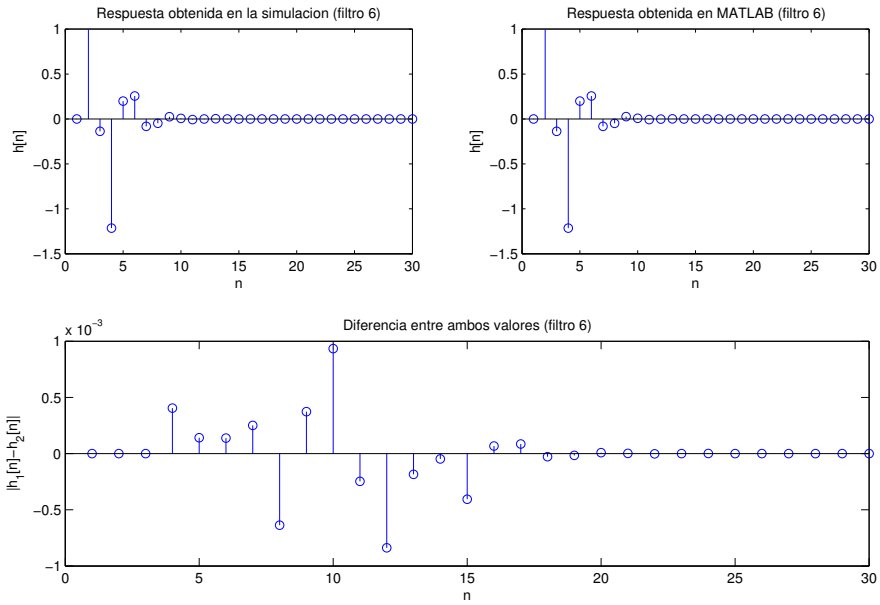

Figura 1.8: Comprobación del filtro 6

Appendices

.1. Descripción del programa de simulación de filtros

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;



entity test_filter is
    PORT (
                iout0 : out integer;
            iout1 : out integer;
            iout2 : out integer;
            iout3 : out integer;
            iout4 : out integer;
            iout5 : out integer;
            iout6 : out integer);
end;

architecture only of test_filter is

signal sout0 : std_logic_vector (15 downto 0);
signal sout1 : std_logic_vector (15 downto 0);
signal sout2 : std_logic_vector (15 downto 0);
signal sout3 : std_logic_vector (15 downto 0);
signal sout4 : std_logic_vector (15 downto 0);
signal sout5 : std_logic_vector (15 downto 0);
signal sout6 : std_logic_vector (15 downto 0);

signal out0 : signed (15 downto 0);
signal out1 : signed (15 downto 0);
signal out2 : signed (15 downto 0);
signal out3 : signed (15 downto 0);
signal out4 : signed (15 downto 0);
signal out5 : signed (15 downto 0);
signal out6 : signed (15 downto 0);

COMPONENT filter0
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter1
```

```
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter2
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter3
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter4
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter5
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

COMPONENT filter6
        port (
                sin : in std_logic_vector (15 downto 0);
                sout : out std_logic_vector (15 downto 0);
                clk : in bit);
END COMPONENT ;

SIGNAL clk   : bit := '0';
SIGNAL sin  : std_logic_vector (15 downto 0) := "0000000000000000";

begin
out0 <= signed(sout0);
out1 <= signed(sout1);
```

```
out2 <= signed(sout2);
out3 <= signed(sout3);
out4 <= signed(sout4);
out5 <= signed(sout5);
out6 <= signed(sout6);

iout0 <= to_integer(out0);
iout1 <= to_integer(out1);
iout2 <= to_integer(out2);
iout3 <= to_integer(out3);
iout4 <= to_integer(out4);
iout5 <= to_integer(out5);
iout6 <= to_integer(out6);

dut0 : filter0
   PORT MAP (
   sin => sin,
   clk => clk,
   sout => sout0);

dut1 : filter1
   PORT MAP (
   sin => sin,
   clk => clk,
   sout => sout1);

dut2 : filter2
   PORT MAP (
   sin => sin,
   clk => clk,
   sout => sout2);

dut3 : filter3
   PORT MAP (
   sin => sin,
   clk => clk,
   sout => sout3);

dut4 : filter4
   PORT MAP (
     sin => sin,
     clk => clk,
     sout => sout4);

dut5 : filter5
   PORT MAP (
   sin => sin,
```

```
   clk => clk,
   sout => sout5);

dut6 : filter6
   PORT MAP (
   sin => sin,
   clk => clk,
   sout => sout6);

clock : PROCESS
   begin
   wait for 10 ns; clk  <= not clk;
end PROCESS clock;

stimulus : PROCESS
   begin
   sin <= "0000000000000000";
   wait for 5 ns; sin  <= "0000010000000000";
   wait for 10 ns; sin  <= "0000000000000000";
   wait;
end PROCESS stimulus;

end only;
```