Universidad Tecnológica Nacional

Técnicas Digitales IV

Trabajo Práctico 5: Diseño de filtros FIR en aritmética de punto fijo, implementación en FPGA

Autores:
Ignacio Rodríguez Grandi
Federico Elizondo
Matías Iván Bucca
Jonás Gabriel Ramírez Torres

Supervisor:
Pablo Cayuela
Sergio Olmedo

26 de junio de 2021

Índice

1. Objetivo **2**

2. Indroducción **2**
2.1. Punto Fijo 2
2.1.1. Concepto Punto Fijo 2
2.1.2. Punto Fijo Binario Natural 3
2.1.3. Notación $s\ p.r$ 3
2.1.4. Notación Qm.f 4
2.1.5. Operaciones en Punto Fijo 4
2.1.6. Algunos ejemplos de ajuste de rango 5
2.2. Filtro FIR (Finite Impulse Response) 6
2.2.1. Expresión matemática de los filtros FIR 6
2.2.2. Estructura básica 7

3. Implementación **7**
3.1. Cálculo de coeficientes 7
3.2. Generación señal de entrada 9
3.3. Implementación del filtro en VHDL 11
3.3.1. Circuito 11
3.3.2. Testbench 13

4. Comparación de resultados **15**
4.1. Resultados obtenidos en MATLAB 15
4.1.1. Señal filtrada 15
4.1.2. Espectro de la señal filtrada 15
4.2. Resultados obtenidos con la implementación del filtro 16
4.2.1. Señal filtrada 16
4.2.2. Espectro de la señal filtrada 17

5. Conclusión **17**

1. Objetivo

Diseñar e implementar un filtro FIR en fpga con las siguientes especificaciones:

- Rango de entrada: -1 a 1
- Cantidad de bits de entrada: 12
- Cantidad de bits de salida: 12
- Frecuencia de muestreo: 50 kH
- Frecuencia de corte: 1 kHz
- Tipo de respuesta: Pasa-bajos
- Cantidad de taps: 32
- SNR de salida: mínima 60 dB
- Atenuación en banda de stop: mínimo 32 dB
- Ganancia en banda de paso
- Mínima aritmética interna

2. Indroducción

2.1. Punto Fijo

2.1.1. Concepto Punto Fijo

Es una colección de N dígitos binarios posee 2^Nestados posibles. Podemos asociar cada estado con un número particular. Si entre dos estados consecutivos cualquiera la diferencia de dichos número es contante, **estamos trabajando en punto fijo.** (Figura 1).

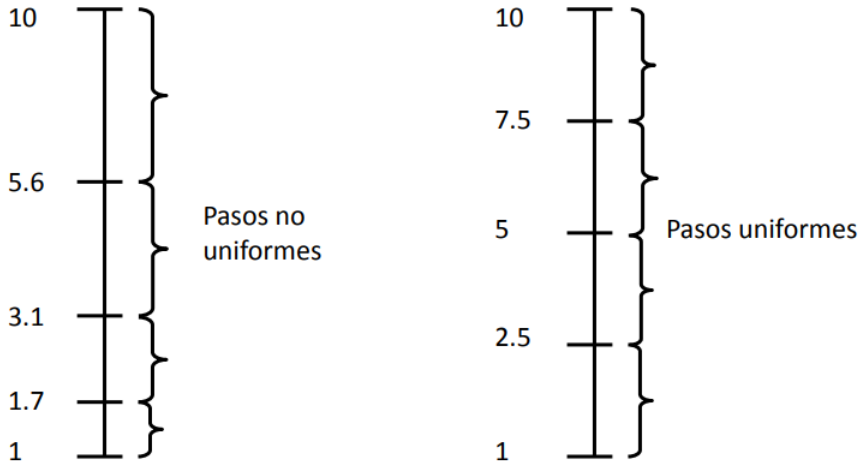

Figura 1: Escalas entre estados

2.1.2. Punto Fijo Binario Natural

Punto fijo binario consiste en definir la ponderación de cada bit en base 2.

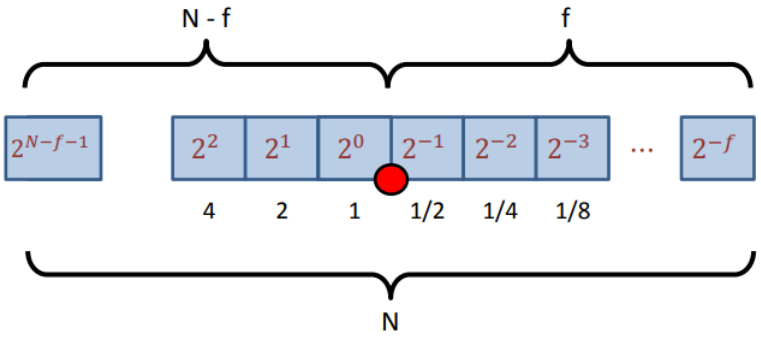

Figura 2: Ponderación de bits.

2.1.3. Notación *s p.r*

Cada sigla representa:

- s: Signo (S/U).
- p: Precisión → Cantidad de bits.
- r: Resolución → Bits de la parte fraccional.

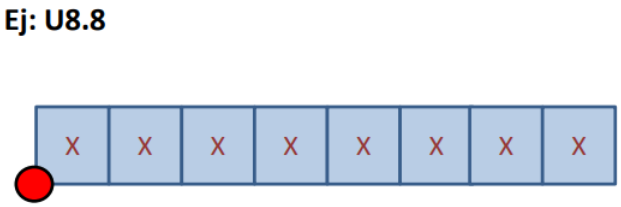

Figura 3: Ejemplo notación s p.r

2.1.4. Notación Qm.f

Cada sigla representa:

- m: Cantidad de bits de la porción entera.
- f: Cantidad de bits de la porción fraccional.
- Siempre con signo.

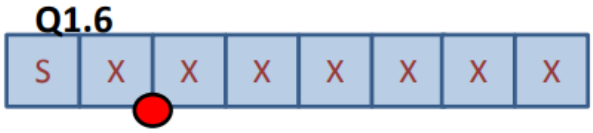

Figura 4: Ejemplo notación Qm.f

2.1.5. Operaciones en Punto Fijo

Suma - Resta:
$Sp1.r1 + Sp2.r2$
$S(p1 - r1 + r2 + 1).r2$ para $p1 - r1 > p2 - r2$ y $r2 > r1$
$S(p2 - r2 + r1 + 1).r1$ para $p2 - r2 > p1 - r1$ y $r1 > r2$

$S(p2+1).r2$ para $p2-r2 > p1-r1$ y $r2 > r1$
Multiplicación:
$Sp1.r1xSp2.r2 = S(p1+p2).(r1+r2)$

2.1.6. Algunos ejemplos de ajuste de rango

Overflow (Wrap):

- Truncado de bits de rango.
- Útil para operaciones en módulo N.
- Fácil de implementar.

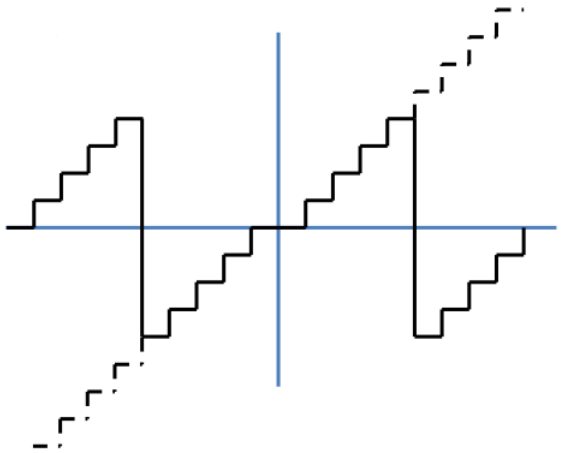

Figura 5: Overflow

Saturación:
Consiste el enclavar la salida a uno de los límites.

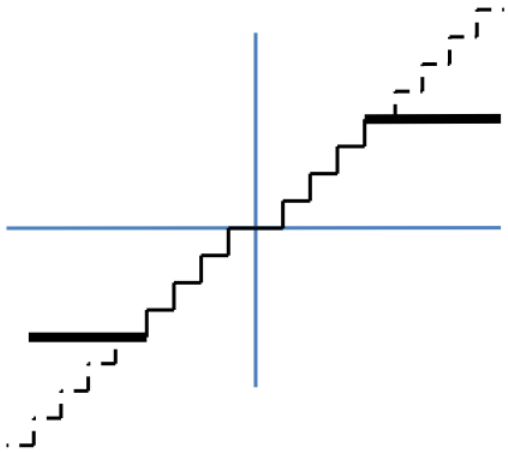

Figura 6: Saturación.

2.2. Filtro FIR (Finite Impulse Response)

Se trata de un tipo de filtros digitales cuya respuesta a una señal impulso como entrada tendrá un número finito de términos no nulos.

Los filtros FIR tienen la gran ventaja de que pueden diseñarse para ser de fase lineal, lo cual hace que presenten ciertas propiedades en la simetría de los coeficientes. Este tipo de filtros tiene especial interés en aplicaciones de audio. Además son siempre estables.

Por el contrario también tienen la desventaja de necesitar un orden mayor respecto a los filtros IIR para cumplir las mismas características. Esto se traduce en un mayor gasto computacional.

2.2.1. Expresión matemática de los filtros FIR

La salida puede expresarse como la convolución de la señal de entrada $x(n)$ con la respuesta al impulso $h(n)$:

$$y(n) = \sum_{k=0}^{N-1} h_k . x_{n-k}$$

Aplicando la transformada Z a la expresión anterior:

$$H(Z) = \sum_{k=0}^{N-1} h_k.z^{-k} = h_0 + h_1.z^{-1} + ... + h_{N-1}.z^{-(N-1)}$$

2.2.2. Estructura básica

La estructura básica de un FIR es:

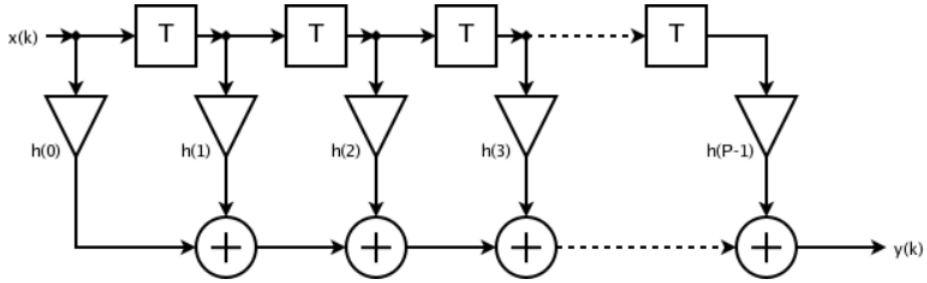

Figura 7: Estructura básica de un filtro FIR.

En la figura 7 los términos h(n) son los coeficientes y los T son retardos.
Pueden hacerse multitud de variaciones de esta estructura. Hacerlo como varios filtros en serie, en cascada, etc.

3. Implementación

3.1. Cálculo de coeficientes

Como primer paso, mediante un script en matlab. Se calculan los coeficientes del filtro para las especificaciones solicitadas. Los coeficientes están en un principio en punto flotante y luego se pasan a punto fijo.

Listing 1: Generación de coeficientes

```
% Ajustes para las especificaciones
f1    = 500;   % Frecuencia  1
f2    = 5000;  % Frecuencia  2
fc    = 1e3;   % Frecuencia de corte del filtro
fs    = 50e3;  % Frecuencia de muestreo
L     = 1000;  % Longitud de simulacion
```

```
NFFT = 1024; % Cantidad de puntos para los graficos
    en frecuencia
simulation_type = 0;
% Definicion de punto fijo
NB_I        = 12;    % Cantidad de bits de la entrada
NBF_I       = 11;    % Parte fraccional de la entrada

NB_COEFFS   = 12;    % Cantidad de bits de los
   coeficientes
NBF_COEFFS = 11;     % Parte fraccional de los
   coeficientes

NB_PP       = 12; % Cantidad de bits de producto
   parcial
NBF_PP      = 11; % Parte fraccional del producto
   parcial

NB_ACCUM    = 12+5; % Cantidad de bits del
   acumulador
NBF_ACCUM   = 11; % Parte fraccional del acumulador

NB_O        = 12;    % Cantidad de bits de la salida
NBF_O       = 11;    % Parte fraccional de la salida

NTAPS       = 32;    % Cantidad de taps

% Calculo de coeficientes
b = fir1(NTAPS-1,fc/(fs/2),'low'); % fir1(Orden del
    filtro, frecuencia digital (0 a 1), tipo de
   respuesta)

% Cuantizacion de los coeficientes a punto fijo
bfx=fi(b,1,NB_COEFFS,NBF_COEFFS);
```

A continuación se muestra la respuesta en frecuencia del filtro comparando los coeficientes en punto flotante y punto fijo.

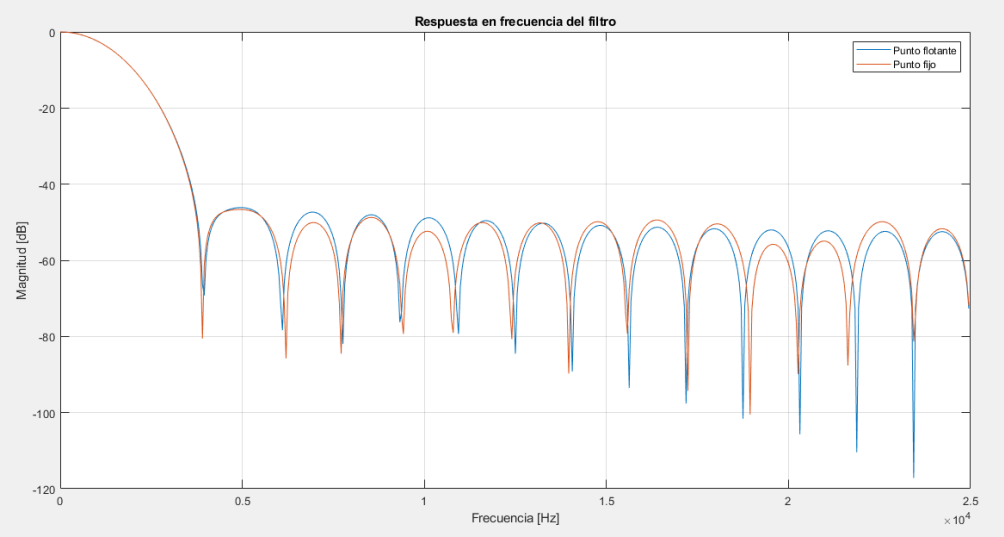

Figura 8: Comparación respuesta con coeficientes punto fijo y flotante.

Cómo siguiente paso, se guardan los 32 coeficientes en un archivo "$coeffs.vhd$".

Listing 2: Guardado de coeficientes en archivo.

```
% Escritura de archivo de coeficientes
fid=fopen('coeffs.vhd','w+');      % Abre archivo .vhd
for i=1:NTAPS                      % Imprime coeficientes
    tmp=bfx(i);
    fprintf(fid,['"' tmp.bin '",\n']);
end
fclose(fid)         % Cierra archivo
```

3.2. Generación señal de entrada

Se trata de la suma de dos señales sinusoidales de 500 hz y 5 khz. El resultado se ve en la figura 9. Los valores generados se guardan en un archivo "x.txt"

Listing 3: Generación señal de entrada

```
if (simulation_type==0)
  a = sin(2*pi* round(f1/fs*NFFT)/NFFT * (0:L-1)) + sin
     (2*pi* round(f2/fs*NFFT)/NFFT * (0:L-1)); % Se
     redondea la frecuencia para no tener leakage en la
      representacion espectral
else
  a = sin(2*pi* round(f1/fs*NFFT)/NFFT * (0:L-1));
end

a = a /max(a) * (1-2*2^-NBF_I); % Normalizacion de la
   senal de entrada
afx          = fi(a                    ,1,NB_I,NBF_I);
%Genero vectores de entrada para el testbench
fid=fopen('x.txt','w+');
for i=1:length(yfx)
    tmp=afx(i);
    fprintf(fid,[tmp.bin '\n']);
end
fclose(fid)
```

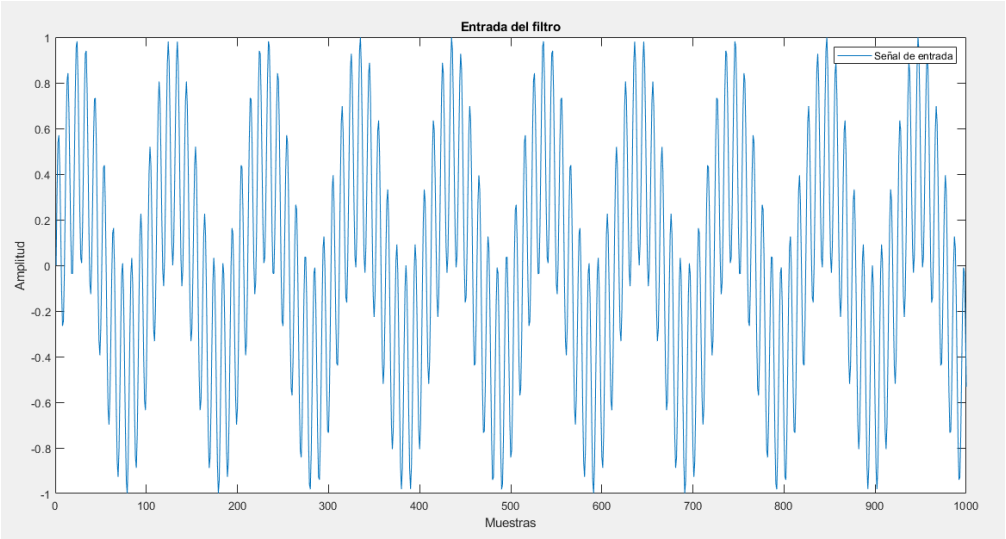

Figura 9: Señal de entrada generada

3.3. Implementación del filtro en VHDL

El filtro se puede simplificar como un registro de desplazamiento para los valores de entrada, los cuales son multiplicados por los coeficientes del filtro y sumados entre si para generar cada salida y[n]. Los valores de esta están truncados, tomando solo los 12 bits más significativos.

3.3.1. Circuito

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

entity fir is
port(
  clk_i : in std_logic;
  reset_i : in std_logic;
  x       : in signed(11 downto 0);
  y       : out signed(11 downto 0)
);
end entity;
architecture behav of fir is

  type fixed_t is array (0 to 31) of signed(11 downto 0);
  signal taps : fixed_t := (
        "000000000101",
        "000000000110",
        "000000001001",
        "000000001110",
        "000000010101",
        "000000011101",
        "000000101000",
        "000000110100",
        "000001000001",
        "000001001111",
        "000001011100",
        "000001101001",
        "000001110100",
        "000001111101",
        "000010000011",
        "000010000110",
        "000010000110",
        "000010000011",
        "000001111101",
```

```
        "000001110100",
        "000001101001",
        "000001011100",
        "000001001111",
        "000001000001",
        "000000110100",
        "000000101000",
        "000000011101",
        "000000010101",
        "000000001110",
        "000000001001",
        "000000000110",
        "000000000101");

  signal delay : fixed_t;
begin
delay_proc : process(clk_i, reset_i)
begin
  if(reset_i='1') then
    for i in 0 to 31 loop
      delay(i) <= (others=>'0');
    end loop;
  elsif(rising_edge(clk_i)) then
    delay(0) <= x;
    for i in 1 to 31 loop
      delay(i) <= delay(i-1);
    end loop;
  end if;
end process;

calc_proc : process(clk_i, reset_i)
  variable acum : signed(23 downto 0);
begin
  if(reset_i='1') then
    y <= (others=>'0');
    acum := (others=>'0');
  elsif (rising_edge(clk_i)) then
    acum:=(others=>'0');
    for i in 0 to 31 loop
      acum := acum + taps(i)*delay(i);
    end loop;
    y <= acum(23 downto 12);
  end if;
end process;
end architecture;
```

3.3.2. Testbench

Este testbench toma los valores del archivo "x.txt", para ser utilizados como valores de entrada del filtro. Posteriormente guarda cada salida en un archivos llamado *output.txt*.

```
library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;
        use std.textio.all;
        use work.txt_util.all;
        entity fir_tb2 is
        end entity;
        architecture behav of fir_tb2 is
          signal clk_i   : std_logic :='0';
          signal reset_i : std_logic;
          signal x       : signed(11 downto 0);
          signal y       : signed(11 downto 0);
          file samples : TEXT open READ_MODE is "x.txt";
          file filtered : TEXT open WRITE_MODE is "output.txt";
        begin
        UUT : entity work.fir(behav) port map(clk_i,reset_i,x,y);
        clk_i   <= not(clk_i) after 10 ns;
        reset_i <= '0', '1' after 10 ns, '0' after 20 ns;
        process(clk_i)
          variable sample_line : LINE;
          variable x_int : string(12 downto 1);
        begin
          if(rising_edge(clk_i)) then
            readline(samples,sample_line);
            read(sample_line,x_int);
            x <= signed(to_std_logic_vector(x_int));
          end if;
        end process;
        process(clk_i)
          variable output_line : LINE;
          variable x_int : string(12 downto 1);
        begin
          if(falling_edge(clk_i)) then
            write(output_line,to_integer(y));
            writeline(filtered,output_line);
          end if;
        end process;
        end architecture;
```

En la figura 10 se observa a modo ilustrativo las señales vistas en la simulación.

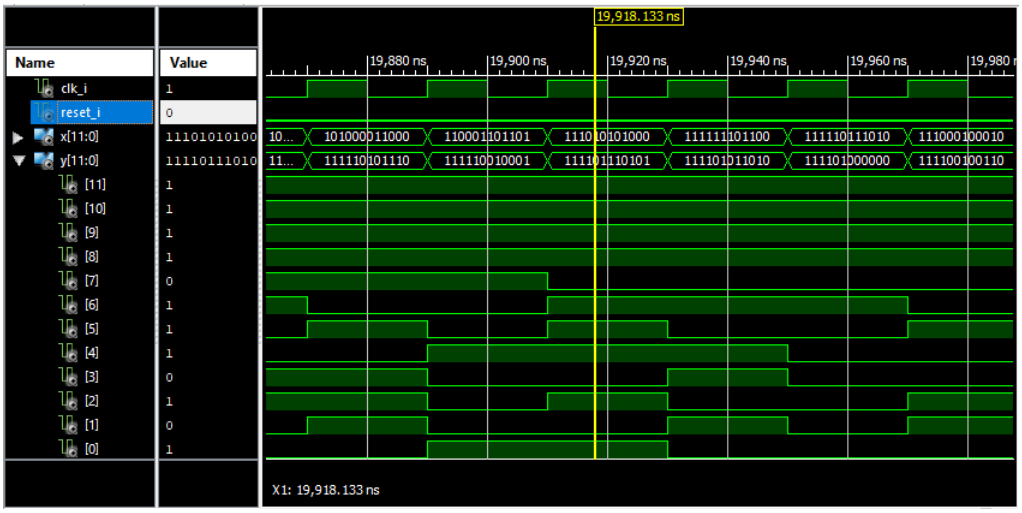

Figura 10: Testbench del filtro.

4. Comparación de resultados

4.1. Resultados obtenidos en MATLAB

4.1.1. Señal filtrada

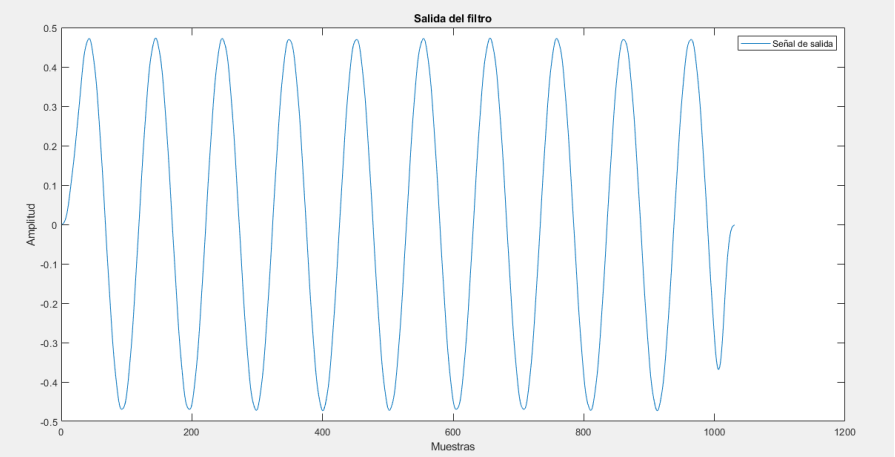

Figura 11: Señal filtrada en MATLAB.

4.1.2. Espectro de la señal filtrada

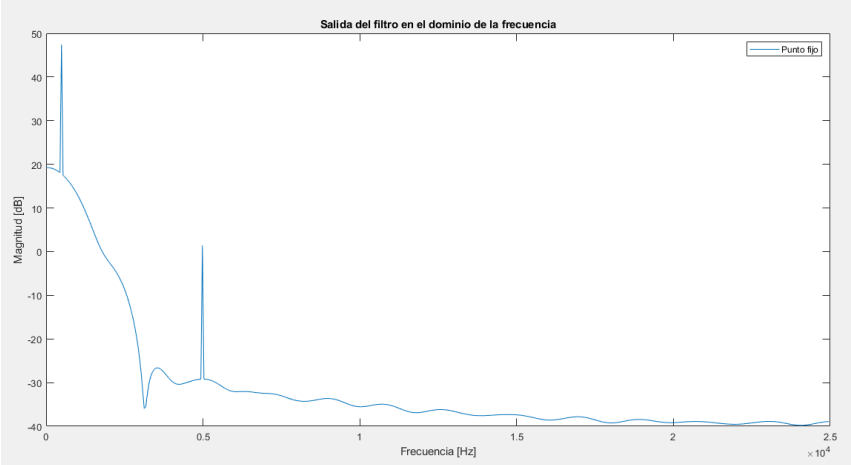

Figura 12: Espectro de la señal filtrada en MATLAB.

4.2. Resultados obtenidos con la implementación del filtro

4.2.1. Señal filtrada

Para este caso, mediante un script en *MATLAB*, se leen los valores del archivo *output.txt* y luego se grafican.

Listing 4: Lectura salida generada y ploteo

```
% Lectura salida generada por el filtro implementado en
    vhdl
fid=fopen('output.txt','r');
data= fscanf(fid,'%f')
fclose(fid)

figure();
plot(data)
legend('Senal de salida')
title('Salida del filtro implementado en vhdl')
ylabel('Amplitud');
xlabel('Muestras');
```

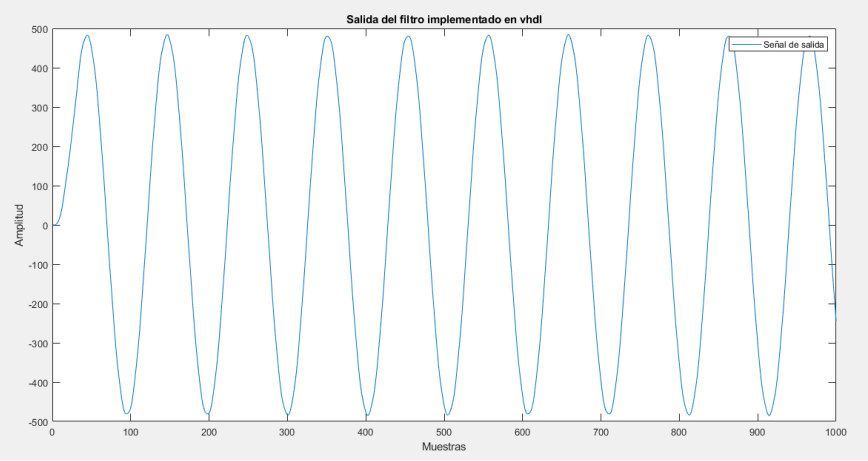

Figura 13: Señal filtrada con el filtro implementado en VHDL.

4.2.2. Espectro de la señal filtrada

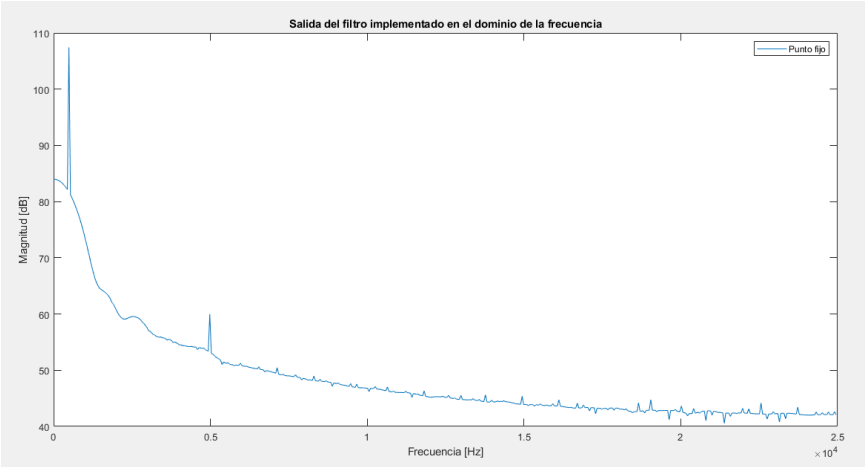

Figura 14: Espectro de la señal filtrada con el filtro implementado en VHDL.

5. Conclusión

En este trabajo se adquirieron conocimientos respecto al manejo de números punto fijo, el cual es de gran utilidad para realizar algoritmos destinados a *DSP* (Digital Signal Processor).

Se logró profundizar conocimientos respecto a los filtros FIR y se aprendió a realizar una implementación básica en VHDL.

Observando los resultados obtenidos, se puede decir que la salida del filtro implementado satisface los requerimientos especificados, trayendo como única desventaja la cantidad de componentes utilizados. Como era de esperarse, se puede apreciar que a mayor cantidad de "taps"del filtro, mayor será la cantidad de componentes a implementar.