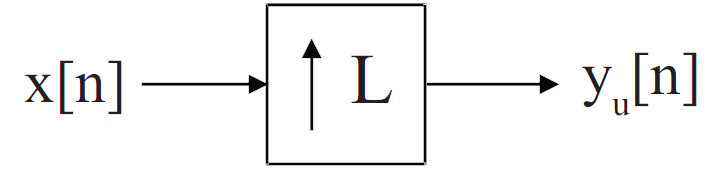
**- Explicación del filtro Interpolador:**

**UP-Sampler:**

UP-Sampler: Es un sistema que introduce muestras nulas entre las muestras consecutivas de la señal de entrada

- Un UP-Sampler con factor de UP-Sampling inserta L-1 muestras nulas entre las muestras consecutivas de la señal de entrada.



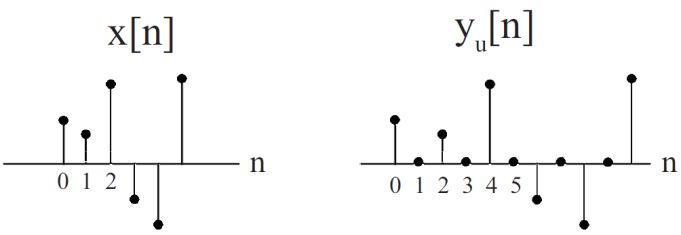
Donde son los múltiplos de L

- El proceso de insertar ceros genera grandes variaciones de la señal en el dominio del tiempo

🡪 Por lo que introduce componentes espectrales de alta frecuencia.

- No causa pérdida de información en la señal de entrada

- Ejemplo: UP-Sampler con L=2



=> Caracterización en el domino Z y de Fourier

La transformada Z de la salida del UP-Sampler es:

- Siendo los valores no nulos de 'n' aquellos que son múltiplos de L 🡪

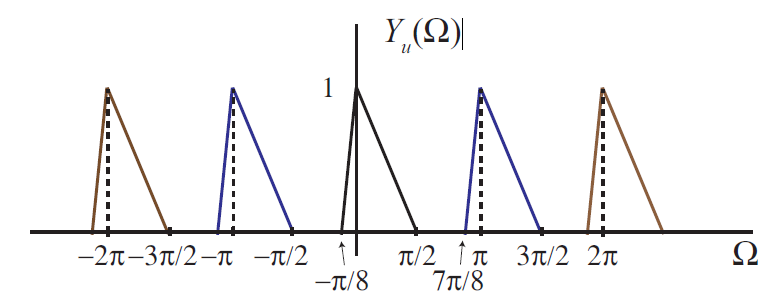
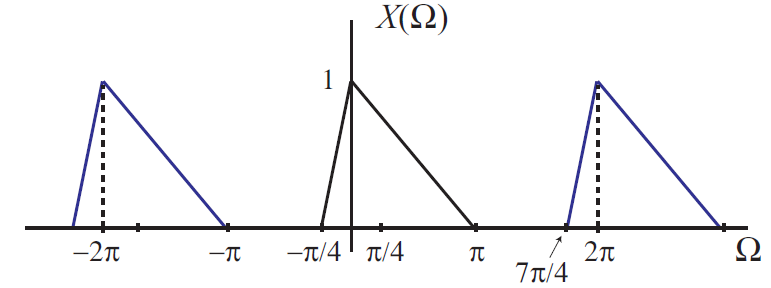
- La transformada de Fourier resulta:

- El espectro de la salida es una versión comprimida del espectro de la entrada

Siendo las señales en el dominio discreto de Fourier periódicas de periodo

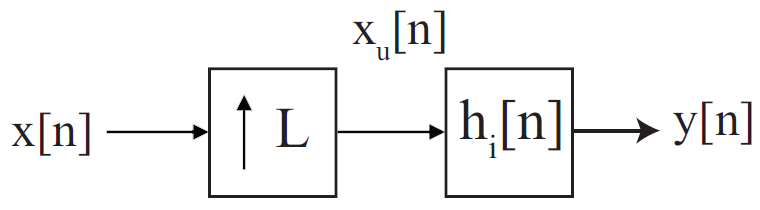
Al comprimir el espectro por un factor L, habrá L replicas del espectro entre

Se puede decir que aparecen L -1 imágenes del espectro centradas en múltiplos de



- Las imágenes son idénticas y contienen información redundante

- Sólo resulta de interés las componentes espectrales comprendidas en el intervalo

**Sistema Interpolador:**

Un Interpolador es aquel sistema formado por la interconexión en cascada de un UP-Sampler y un filtro

- UP-Sampler con factor L:

· Su señal de salida posee en el intervalo las componentes de la señal de entrada L veces repetidas:

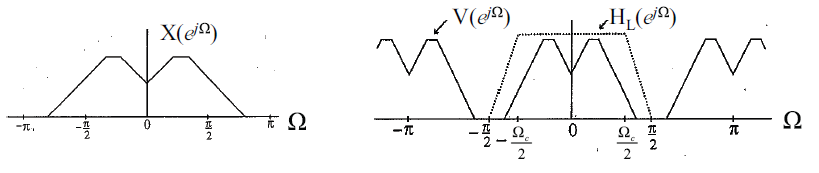
· Sólo importan las componentes no repetidas: intervalo

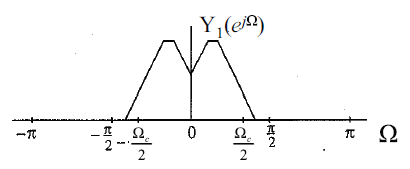
- Filtro Paso-Bajo :

· Elimina las componentes frecuenciales repetidas

: Es el factor de escala usado para normalizar la secuencia de salida

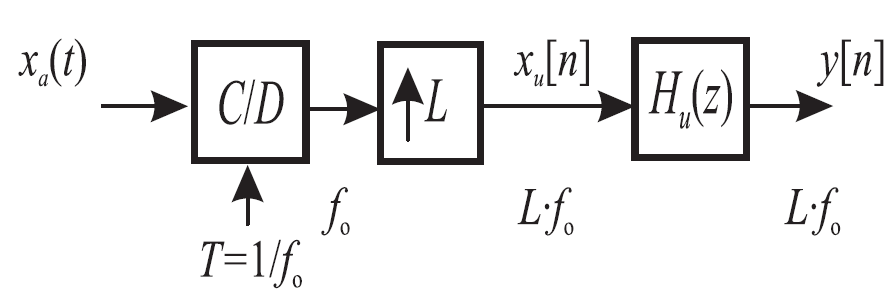
- Ejemplo gráfico:





- El filtro paso bajo rellena el valor de las muestras nulas introducidas por el UP-Sampler

- El interpolador se utiliza frecuentemente para incrementar la frecuencia de muestro de la señal dada.



**=> En nuestro caso:**

Tenemos un sistema interpolador de L = 6, para aumentar la frecuencia de muestreo de de la señal de la ROM de 8Ksamples a 48 Ksamples.

- El UP-Sampler hará que las componentes frecuenciales de la señal de 8k se conviertan en 48k

pero añadirá componentes en alta frecuencia que debemos eliminar con el filtro interpolado

- Debemos poner un filtro interpolador que elimine las componentes en alta

frecuencia que introduce el UP-Sampler.

En el dominio discreto, la frecuencia de corte del filtro interpolar debe ser como mínimo de:

- Estando, la señal dada en 48 Ksamples, haciendo el cambio discreto-analógico:

La frecuencia mínima de corte en el dominio continuo que tiene que tener nuestro filtro interpolador es:

Es decir, relación normalizada 4000/48000

- La interpolación "rellenará" los valores nulos (o duplicados) con los valores "auténticos" de la señal por lo que dicha señal estará preparada para ya para ser oída o procesada mediante el filtro en el caso que corresponda.

- En el caso de que la tengamos que filtrar, nuestro filtro está normalizado a 1800/48000 por lo que si la señal de entrada al filtro está en 48KSamples, la frecuencia de corte analógica es

- Lo que en el dominio discreto se traduce a un filtro de frecuencia de corte

Dicho filtro se usa tanto para filtrar la señal de 8K interpolada de la ROM a 48K como la de 48k

que viene directa del micrófono (Las señales se multiplexan a la entrada del filtro)

- Así pues, en un caso ideal, tendríamos en el interpolador el filtro normalizado 4000/48000 y

en el filtro en si del sistema el filtro paso bajo normalizado 1800/48000

**Esto es lo que debería hacerse.**

**Sin embargo**, debido a nuestra falta de tiempo para implementar 2 filtros con coeficientes diferentes y ya que se permite hacerlo, ambos filtros serán aquel con frecuencia de corte normalizada 1800/48000 🡪 Esto generará imperfecciones.

En el ejemplo que trataremos con la señal de componentes 200 y 2000 Hz, la segunda componente entrará dentro de la banda de transición del filtro real, por lo que al interpolarla eliminaremos parte de esta componente y al volverla a filtrar por el mismo filtro volverá a atenuarse aún mas, aproximando el resultado deseado.

- Así pues para el tratamiento de las señales:

- Caso ROM sin filtrar 🡪 Interpolamos usando el mismo filtro que el del sistema en só

- Caso ROM filtrado 🡪 Interpolamos y filtramos (ambos con el mismo tipo de filtro pero 2 instanciaciones diferentes lo que se traduce a una reducción aún mayor de las componentes de alta frecuencia.

- Caso Microfono sin filtrar 🡪 La ponemos a la salida directamente

- Caso Microfono filtrado 🡪 Filtramos directamente la señal con el filtro 1800/48000

A la entrada del segundo filtro implementado multiplexaremos:

- La señal interpolada de la ROM

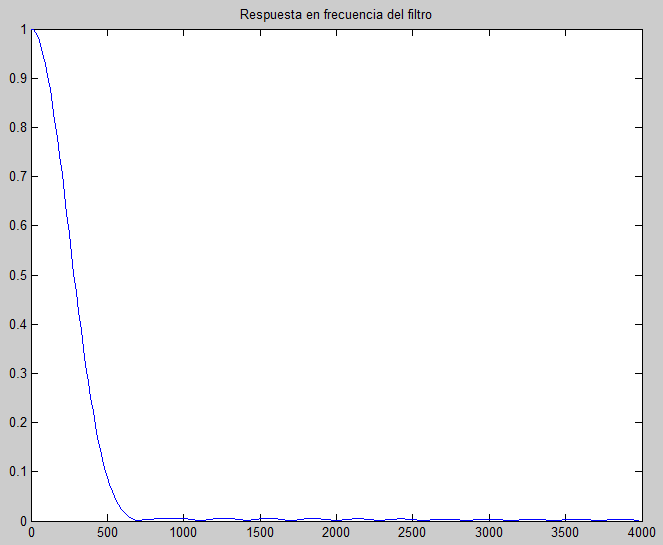
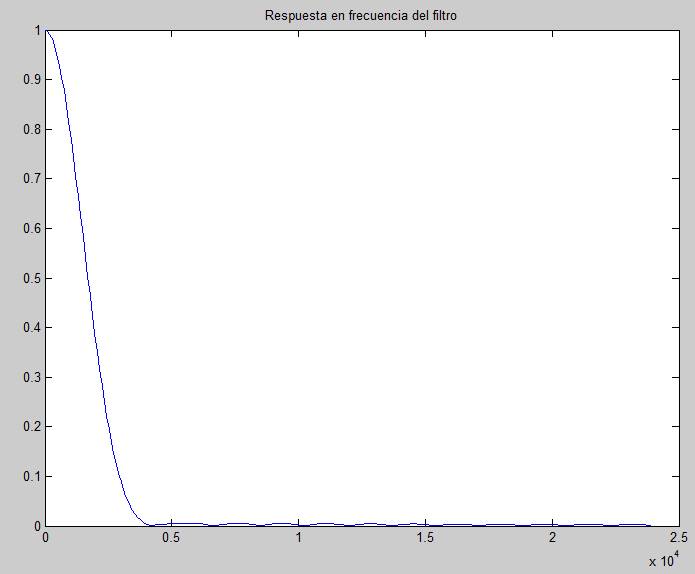
- La senal del micrófono

**X. Tratamiento en Matlab:**

**X.1 Cualitativamente:**

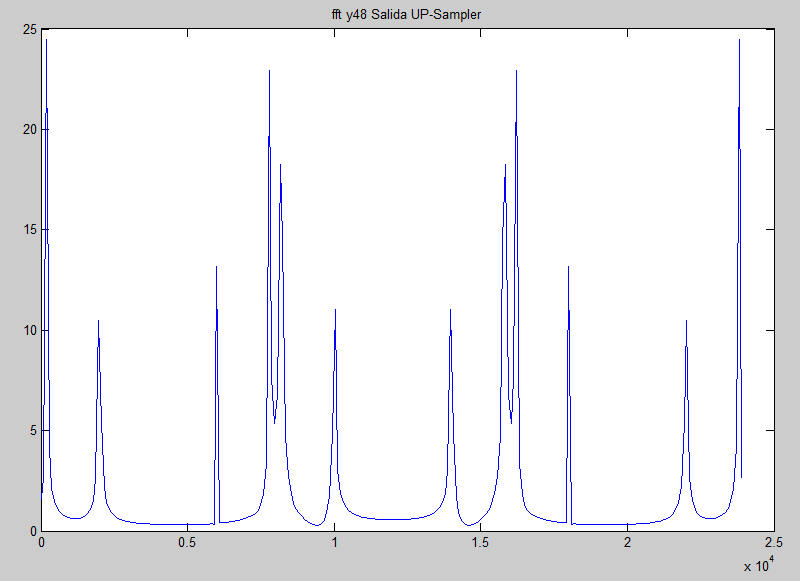
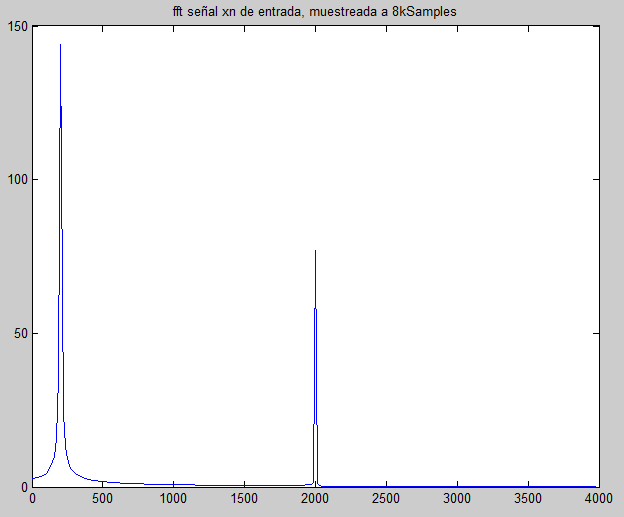
- La respuesta en frecuencia de nuestro **filtro** sobre una señal a 48Ksamples en el dominio contínuo de Fourier es la figura de la izquierda abajo

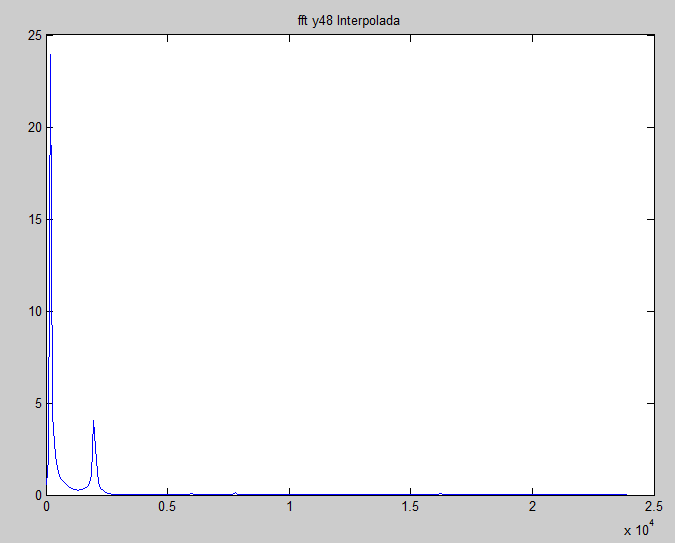
Si la señal que le introducimos está a 8KSamples, la respuesta que tendría en el filtro tendría la misma forma pero comprimida por un factor de 6 en el dominio de la frecuencia (fig. derecha)



- Ahora generamos la señal a guardar en la ROM a 8KSamples cuyas componentes frecuenciales son las de la figura de la izquierda abajo

- Al pasarlo por el **UP-Sampler**, en el dominio discreto, el espectro se convierte en 6 réplicas comprimidas por 6 de la señal de entrada, que al pasarlo a tiempo continuo se produce el mismo efecto. El espectro estará comprimido considerando la señal como de 8KSamples pero si consideramos su periodo de muestreo de 48 KSamples, no estará comprimido y la única diferencia serán las réplicas 🡪 Figura de la derecha





- Para realizar la interpolación completa, pasamos la señal de salida del UP-Sampler por el **filtro interpolador**, que en esta práctica tendrá los mismos coeficientes que el filtro del sistema.

Al ver su salida en la figura de abajo, vemos que efectivamente su espectro en 48KSamples es aproximadamente igual al de la señal de la señal en 8KSamples.

Por lo que la señal de salida del interpolador reproducida a 48KSamples debería sonar aproximado a la señal inicial de la ROM reproducida a 8KSamples pero distorsionada.

**Codigo:**

%% filtro

Wn=1800/48000; % valor normalizado de la frecuencia de corte

% Frecuencia de corte/Frecuencia de muestreo

% Para filtrar una señal muestrada a 8000 habra que

% interpolarla

Num = fir1(27,Wn,'low'); % Creamos los coeficientes del filtro

% Redondeamos al Hardware

duration = 1; % dura 1 sg.

amplitude = 1;

sampleFreq = 8000; % Frecuencia de muestreo de la señal que guardaremos en la ROM

xn= 0.6\*genera\_tono( 200,duration,amplitude,sampleFreq) +0.3\*genera\_tono( 2000,duration,amplitude,sampleFreq);

% Ya hemos generado la señal de un seg muestreada a 8000 Hz

figure();

% Respuesta en frecuencia del filtro

ver\_fft(Num,48e3,'Respuesta en frecuencia del filtro')

figure();

% Muestramos la respuesta en frecuencia de la señal de entrada y su sonido

ver\_fft(xn,8e3,'fft señal xn de entrada, muestreada a 8kSamples')

sound(xn,8e3);

figure();

% Interpolamos la señal de 8K:

% 1) Paso de 8kSamples a 48 kSamples

y48 = zeros(1,6\*length(xn)); % 48 KSamples, 6 muestras / cada muestra de 8KHz

for i=1:6

y48(1:6:end) = xn ;

end

ver\_fft(y48,48e3,'fft y48 Salida UP-Sampler');

% 2) Filtramos la señal con el filtro normalizado 18000/48000

% Muestramos la respuesta en frecuencia de la señal de salida y su sonido

figure();

y48 = filter(Num,1,y48);

ver\_fft(y48,48e3,'fft y48 Interpolada');

sound(y8,8e3);

**X.2 Cuantitativamente:**

Una vez que hemos visto el efecto que produce el interpolador y el filtro nos disponemos a calcular con Matlab la salida del sistema en los diferentes casos teniendo en cuenta los efectos que produce la cuantificación de los datos en un número limitado de bits.

Los datos están expresados en 16 bits, extendiendo el signo los expresamos en 18 bits que nos requiere el sistema.

- Tenemos unos coeficientes están normalizados a 1 y van cuantificados en 16 bits en C-2, por lo que el escalón cuántico es: utilizándose el bit de mayor peso como mínimo para el signo. Por lo que, obtenemos los **coeficientes cuantificados** haciendo:

Wn=1800/48000;

Num = fir1(27,Wn,'low');

Num =Num/max(abs(Num));

Num =round(Num\*2^15);

- Para las entradas utilizaremos directamente las grabadas en los 'datain.txt' que ya están cuantificados correctamente utilizando los mismos métodos que el filtro de la FPGA

**X.2.1 Caso filtrado del micrófono**

- En este caso simplemente creamos la señal de entrada muestreada a 48 KSamples, la codificamos y la pasamos por el filtro. Veamos ahora el caso sin cuantificación.

Wn=1800/48000;

Num = fir1(27,Wn,'low');

duration = 1;

amplitude = 1;

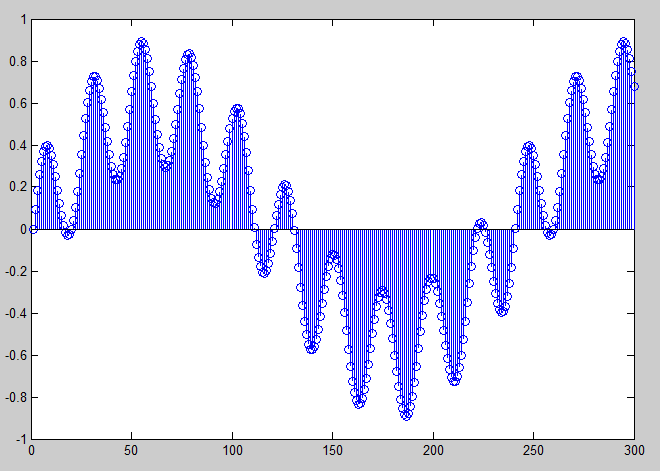
sampleFreq = 48000;

xn= 0.6\*genera\_tono( 200,duration,amplitude,sampleFreq) +0.3\*genera\_tono( 2000,duration,amplitude,sampleFreq);

Salida = conv(Num,xn);

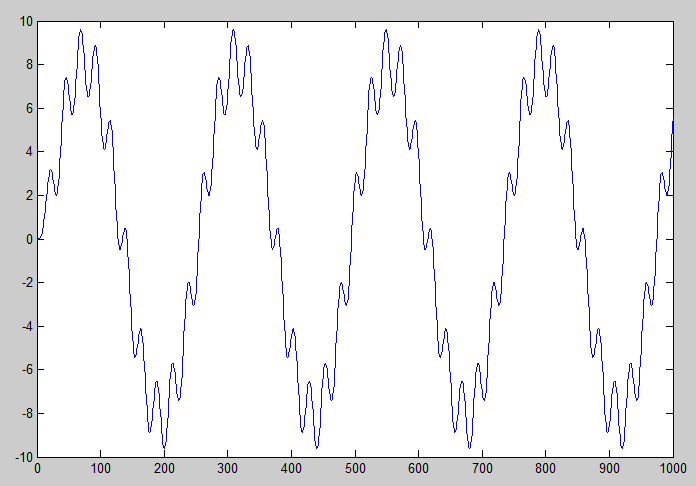
plot(Salida(1:1000));

- La representación de la **entrada** es:



- Vemos las componentes de 200 y 2000 Hz

- Así pues primero representamos la salida del filtro para dicha entrada, sin tener en cuenta redondeo ni errores por cuantificación, simplemente teórico.

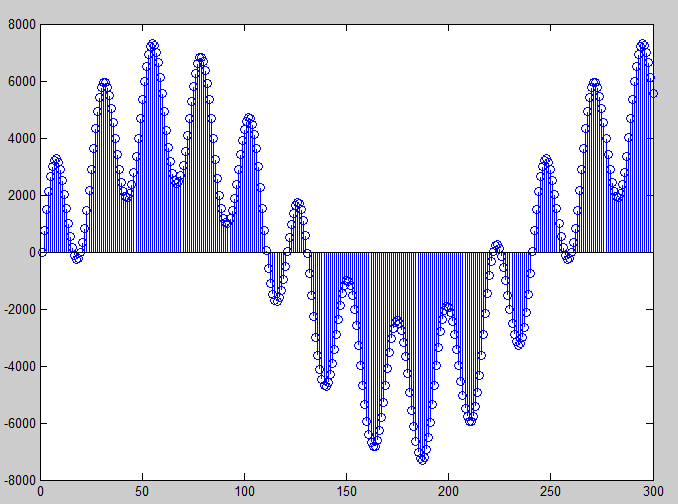


- Vemos que el filtro cumple su cometido y elimina parte de la señal de 2000 Hz pero no toda debido a que dicha señal se encuentra en la banda de transición del filtro real.

**=> Simulación de Filtro FPGA**

- Ahora veamos si nuestro filtro Hardware funciona como es debido, para ello hemos creado una señal con 'generatono' bastante mayor para que no haya mucho error por redondeo y la hemos pasado a un '.txt' con 'var2txt' para que hace directamente el redondeo.

- Teniendo en cuenta los errores por cuantificación y redondeo, el código para emular en Matlab el filtro es: Entrada

Wn=1800/48000;

Num = fir1(27,Wn,'low');

Num =Num/max(abs(Num));

Num =round(Num\*2^15);

xn = txt2var('L\_in.txt');

yn = txt2var('L\_out.txt');

Salida = conv(Num,xn)/ 2^15;

Salida = floor (Salida);

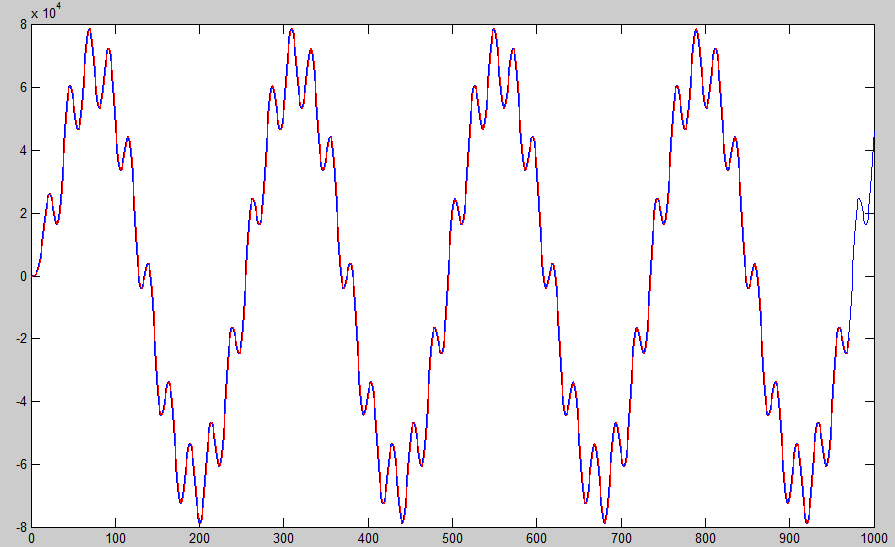
figure();

plot(Salida(1:1000));

hold on;

plot(yn(30:1000),'r');

- Este código representa la señal obtenida por el filtro de la FPGA en azul y la obtenida por Matlab utilizando los mismos procedimientos en rojo pero desplazada ya que el sistólico no tiene muestras validas hasta que el primer resultado recorra toda la estructura.



- Vemos que ambas señales coinciden a la perfección.

**X.2.2 Caso micrófono sin filtrar y filtrado**

- En este caso la señal guardada en la ROM está en 8KSamples por lo que tenemos que pasarla siempre a 48KSamples para que sea reproducible ya que el reproductor lo hace a esa velocidad. A continuación utilizaremos la misma señal de siempre ideal y sin contar errores por redondeo y al final pondremos el caso práctico:

Wn=1800/48000;

Num = fir1(27,Wn,'low');

duration = 1;

amplitude = 1;

sampleFreq = 8000;

xn= 0.6\*genera\_tono( 200,duration,amplitude,sampleFreq) +0.3\*genera\_tono( 2000,duration,amplitude,sampleFreq);

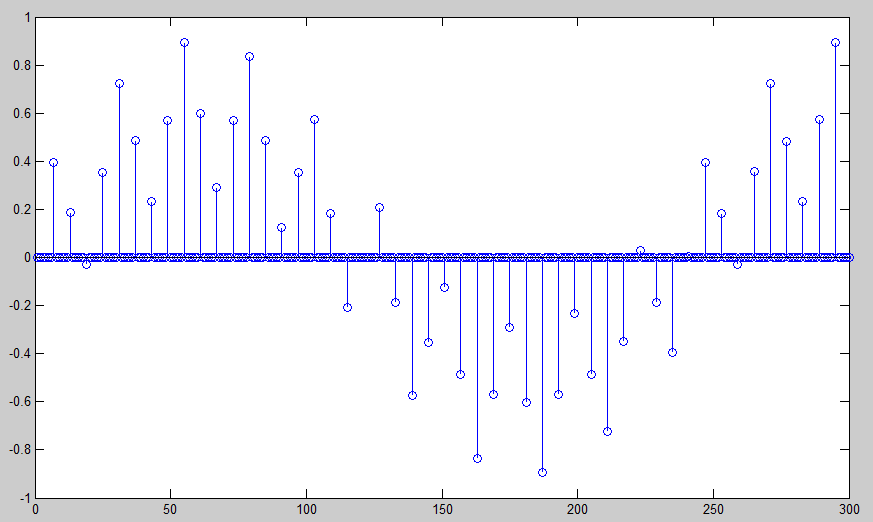
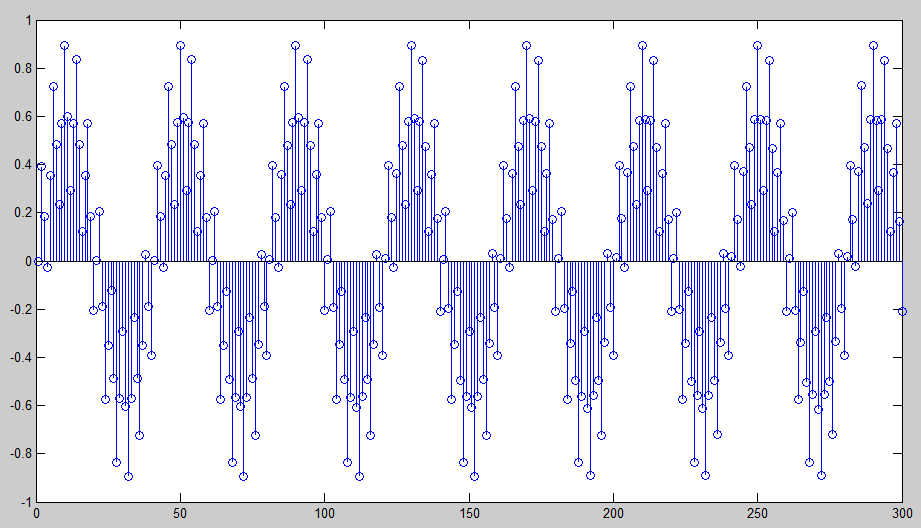
y48 = zeros(1,6\*length(xn)); % 48 KSamples, 6 muestras / cada muestra de 8KHz

for i=1:6

y48(1:6:end) = xn ;

end

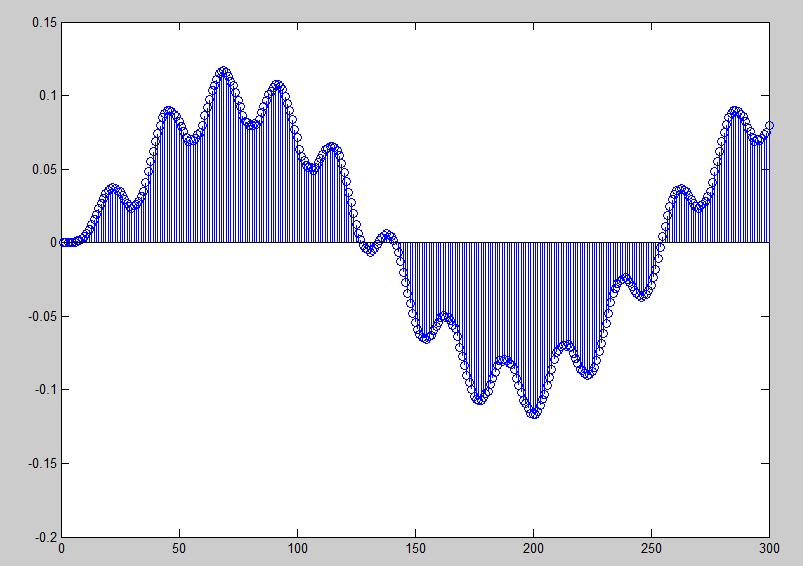
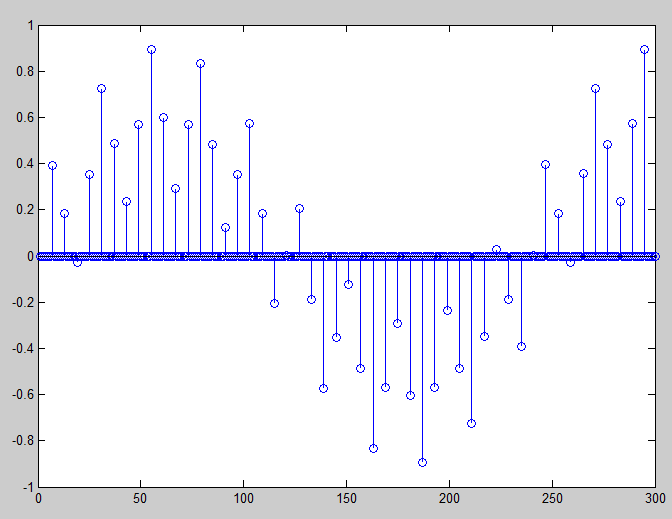
- La señal de la ROM en 8K y en 48K a la salida del UP-Sampler es:



- Vemos que el UP-Sampler mete 5 ceros por cada muestra.

- Ahora pasamos la señal de salida del UP-Sampler por el filtro interpolador y podemos apreciar como el interpolador rellena los valores nulos:

Salida = conv(Num,y48);

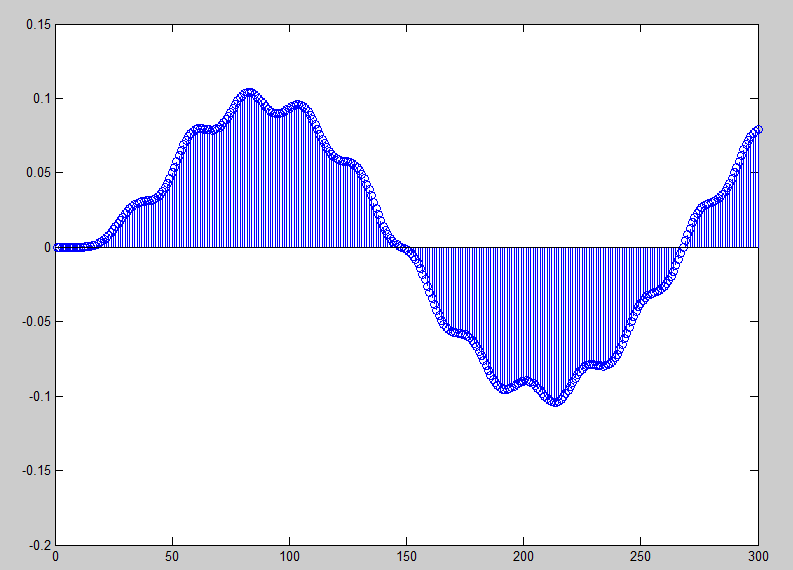


- Así pues, la gráfica de la derecha muestra es la señal de 48KSamples obtenida a partir de la de 8 KSamples de la ROM y ya está preparada para ser reproducida por el dispositivo.

- Vemos que la señal es aproximadamente idéntica a la de la señal del micrófono de 48 KSamples

- En el caso de que nos pidan que la filtremos, debemos volver a pasarla, obteniendo la señal:

Salida2 = conv(Num,Salida);



- Vemos que al volver a pasarla por el filtro vuelve a disminuir la componente de alta frecuencia como esperábamos

**=> Simulación de Filtro FPGA**

- Ahora veamos si nuestro filtro Hardware funciona como es debido, para ello hemos emulado la señal guardada en la ROM de la FPGA y la hemos tratado como debería

- Teniendo en cuenta los errores por cuantificación y redondeo además de usar Sample&Hold en la interpolación, el código para emular en Matlab el filtro en el caso de que la señal interpolada no tenga que ser filtrada es:

Wn=1800/48000;

Num = fir1(27,Wn,'low');

Num =Num/max(abs(Num));

Numr = Num;

Num =round(Num\*2^15);

% Esta vez creamos la señal nosotros:

duration = 1; % dura 1 sg.

amplitude = 1;

sampleFreq = 8000; % Frecuencia de muestreo de la señal ROM

xn=0.6\*genera\_tono( 200,duration,amplitude,sampleFreq) +0.3\*genera\_tono( 2000,duration,amplitude,sampleFreq);

xn =round(xn\*2^10); % Mismo escalado que en la ROM.coe

y48 = xn(1)\*rectwin(6)';

for i=2:length(xn)

y48 =[y48 xn(i)\*rectwin(6)'];

end;

Salida = conv(Num,y48)/ 2^15;

Salida = floor (Salida); %Salida Matlab con cuantificaion

yn = txt2var('resul.txt'); %Salida filtro

Salida2 = conv(Numr,y48); %Salida Matlab sin cuantificacion

figure();

hold on;

plot(yn(1:1000),'r');

hold on;

plot(Salida2(1:1000),'k');

hold on;

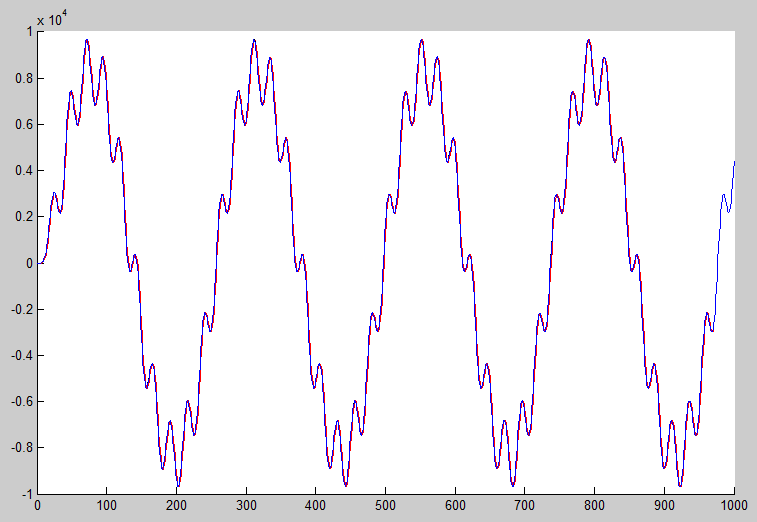
plot(Salida(1:1000));

- Este código representa:

- La señal obtenida por el filtro de la FPGA en azul

- La señal obtenida por Matlab utilizando los mismos procedimientos en rojo

- La señal obtenida por Matlab sin errores de cuantificación en negro



- Vemos que las 3 señales coinciden por lo que el sistema realiza la función deseada y los errores por cuantificación son despreciables

- En el caso de que nos pidan que **filtremos la salida de la ROM** simplemente en Matlab utilizamos el mismo código pero poniendo:

Salida = conv(Num,y48)/ 2^15;

Salida = conv(Num,Salida)/ 2^15;

Salida = floor (Salida);

