UNIVERSIDAD DEL AZUAY
Facultad de Ciencia y Tecnología
Escuela de Ingeniería Electrónica

"Implementación de filtros digitales en controladores digitales de señal"

Tesis previa a la obtención del título de Ingeniero en Electrónica

Autor: Jose Andrés Cordero G.

Director: Ing. Leonel Pérez R.; Msc.

Cuenca, Ecuador
2006

Agradecimientos

Agradezco a todas las personas que aportaron con sus conocimientos y huellas en mi enseñanza universitaria, de manera especial a los profesores Leonel Pérez, Leopoldo Vásquez y Hugo Torres .
Quiero agradecer a mis padres, quienes me apoyaron en todos mis proyectos y a finalizar mi carrera universitaria.
Asimismo quiero dar las gracias a todas las personas que revisaron el desarrollo de esta tesis, a las que me dieron sugerencias, críticas y comentarios.

Resumen

El procesamiento digital de señales más conocida como DSP es una tecnología revolucionaria que esta cambiando varias capas de la ciencia e ingeniería como el campo médico, militar, comercial, industrial, espacial, con técnicas propias.
Aprender DSP envuelve varias habilidades como conocimientos matemáticos, electrónica, ingeniería de software, arquitectura de computadores, microprocesadores y el aprendizaje de una técnica especializada para aplicar en un área de interés, la técnica especializada tratada en esta tesis es el filtrado digital y su aplicación práctica en un sistema de adquisición con un microcontrolador especializado para tareas de DSP.
Esta tesis pretende contribuir al laboratorio de DSP dotando de material teórico y práctico, introduciendo los Filtros Digitales, específicamente los filtros digitales que se implementan por recursión y convolución, implementación de los filtros digitales usando un microcontrolador especializado para tareas de DSP y el análisis del sistema de filtrado usando la PC es otro acercamiento del presente trabajo.

Abstract

The digital signal processing better known as DSP is a revolutionary technology that is changing some fields of the science and engineering like the field medic, military, commercial, industrial, space with own techniques.
Learning DSP wrap some abilities like mathematical knowledges, electronic, software engineering, computer architecture, microprocessors and learning of a specialized technique to apply in an area of interest, the specialized technique treat in this thesis is the digital filtrate and the practice application in an acquisition system with a specialized microcontroller for DSP tasks.
This thesis pretend contribute to the DSP laboratory giving theorical and practice equipment, introducing Digital Filters, specifically that can be implemented by convolution and recurtion, implementation of the digital filters using a specialized microcontroller for DSP tasks and the analysis of filtrate system using the PC is another approach of the present work.

Índice de Contenido

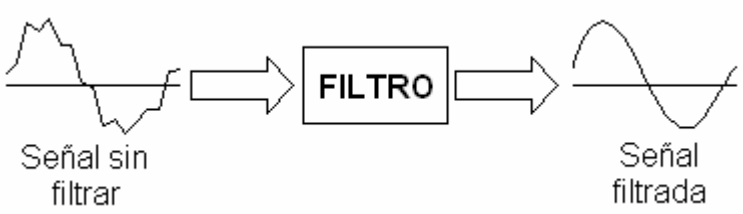
Autor: Cordero García, José Andrés
Trabajo de Graduación
Director: Ing. Leonel Pérez R.; Msc.
Julio 2006

Implementación de filtros digitales en controladores digitales de señal

Introducción

El procesamiento digital de señales es un tema vasto que esta siendo explotado en aplicaciones en la última década, tal es el caso de los filtros digitales que tienen gran precisión y estabilidad comparada con los filtros analógicos.
Existen dos tipos de procesamientos de señales: analógico y digital.
DSP es la intersección de diferentes áreas de estudio, *DSP* es la base de muchas áreas de tecnología, desde teléfonos móviles a módems y software multimedia de las PCs, por lo que cada área ha desarrollado su propia tecnología de *DSP* con técnicas especializadas.
DSP son las matemáticas, los algoritmos, y las técnicas usadas para manipular señales.
El DSP es una tecnología que tuvo sus orígenes cuando aparecieron las primeras computadoras digitales en la década de los años 1960-1970, una de las áreas pioneras en esa época fue el *radar & sonar*.
El filtrado de datos digitalizados, si no es la técnica más fundamental dentro del procesamiento de señales, ciertamente es la técnica más antigua.
La función básica de un filtro es la de dejar pasar un rango de frecuencias mientras se rechazan otras.

El siguiente bloque ilustra un filtro restaurando una señal:



El filtrado digital tiene muchas técnicas usadas en procesamiento digital de señales (*DSP*), en áreas como: comunicaciones, imagen, video digital, comunicaciones de voz, etc...
Para implementar los filtros digitales se necesita un procesador el cual puede ser de propósito general o especializado en el cual se carga un programa que realiza el procesado de los datos.
Por otro lado los filtros analógicos son implementados usando resistores, inductores, capacitores y amplificadores operacionales.

1.1 Introducción Señales y Sistemas

Señal

Una señal es una descripción de como un parámetro varía con respecto a otro parámetro.

Sistema

Un sistema es cualquier proceso que produce una señal de salida en respuesta a una señal de entrada. El bloque de abajo ilustra un sistema.

Notación de señales

Las señales continuas – usan paréntesis:

$$x(t)\ ,\ y(t)$$

Las señales discretas – usan corchetes y en vez de una *t* usan la *n* que indica el número de muestra:

$$x[n]\ ,\ y[n]$$

En algunos libros suele encontrarse también esta notación

$$x(n)\ ,\ y(n)$$

Para señales discretas en el tiempo se usan letras minúsculas

$$x[n]$$

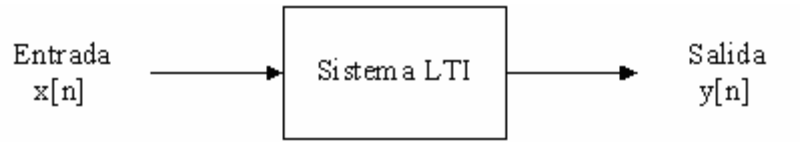
Para señales discretas en el dominio de la frecuencia se usan letras mayúsculas

$$X[f]$$

1.2 Sistemas Lineales e Invariantes en el Tiempo (LTI) discretos

Matemáticamente un sistema discreto es descrito como un operador **T** que toma secuencias x[n] (*llamada excitación*) y transforma esta en otra secuencia llamada y[n] (*llamada respuesta*). Así:

$$y[n] = \mathrm{T}(x[n])$$



Sistemas Lineales Discretos

El término lineal define una clase especial de sistema, donde la salida es la superposición, o suma, de las salidas individuales que tienen entradas individuales que han sido aplicadas al sistema separadamente.

Por ejemplo:
Si aplicamos una entrada $x_1[n]$ a un sistema, el resultado es una salida $y_1[n]$

$$x_1[n] \xrightarrow{\textit{resulta}} y_1[n]$$

Dando otra entrada diferente

$$x_2[n] \xrightarrow{\textit{resulta}} y_2[n]$$

Para que el sistema sea lineal, cuando la entrada sea la suma de

$$x_1[n] + x_2[n]$$

La salida debe ser la suma de las salidas individuales

$$x_1[n] + x_2[n] \xrightarrow{\textit{resulta}} y_1[n] + y_2[n]$$

Parte de esta descripción de linealidad es una característica de proporcionalidad, esto significa que si las entradas son escaladas por factores constantes C_n, entonces las salidas también serán escaladas por los mismos factores C_n

$$C_1 x_1[n] + C_2 x_2[n] \xrightarrow{\textit{resulta}} C_1 y_1[n] + C_2 y_2[n]$$

Propiedad de homogeneidad

Analizando sistemas lineales y no lineales con MATLAB

Ejercicio 1.2.1

Comprobar la linealidad del sistema:

$$y[n] = -\frac{x[n]}{2}$$

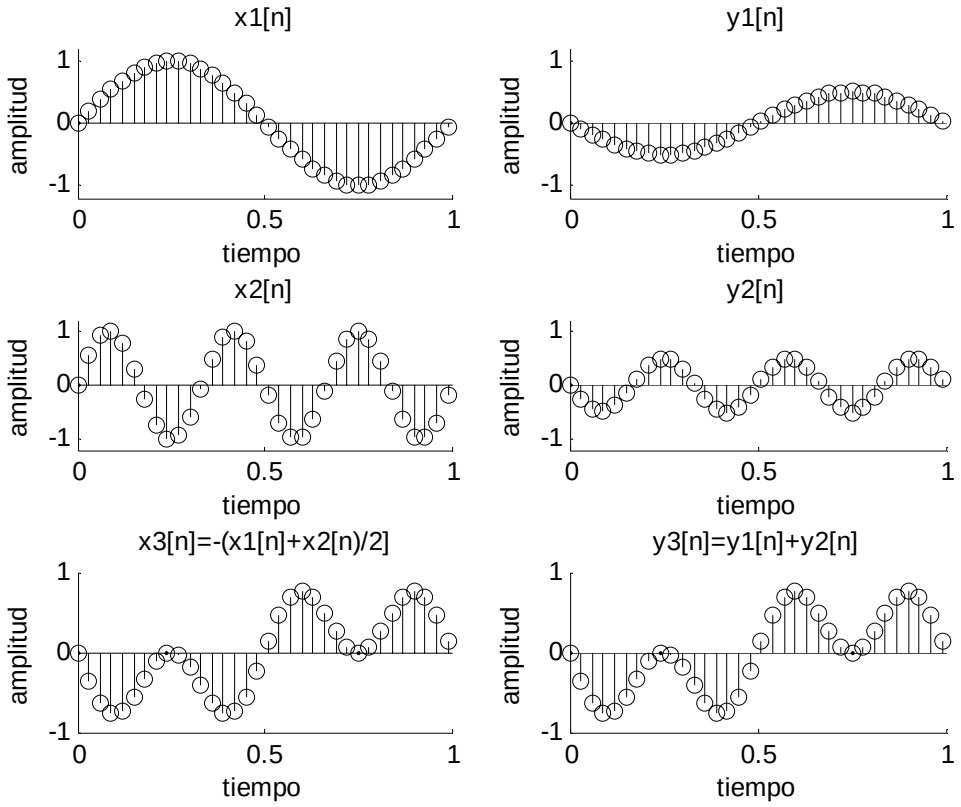
$$x[n] = \sin(2 * \pi * f * n)$$

$x_1[n]$ Una señal seno de 1-Hz muestreada a 32 muestras por ciclo.

$x_2[n]$ Una señal seno de 3-Hz muestreada a 32 muestras por ciclo.

Script MATLAB:

```
% n son las muestras n=inicio: muestras/ciclo: fin
% Definiendo y graficando la señal x1[n]
x1=sin(2*pi*1*n);
subplot(3,2,1);
stem(n,x1,'k')
title('x1[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2]);
y1=-1/2*x1;
subplot(3,2,2);
stem(n,y1,'k')
title('y1[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%x2[n] onda seno de 3-Hz muestreada a 32 muestras por ciclo
n=0:0.03:1;
x2=sin(2*pi*3*n);
subplot(3,2,3);
stem(n,x2,'k')
title('x2[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
y2=-1/2*x2;
subplot(3,2,4);
stem(n,y2,'k')
title('y2[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%x3[n]
subplot(3,2,5);
stem(n,-(x1+x2)/2,'k')
title('x3[n]=-(x1[n]+x2[n)/2]');
ylabel('amplitud');
xlabel('tiempo');
%y3[n]
subplot(3,2,6);
stem(n,y1+y2,'k')
title('y3[n]=y1[n]+y2[n]');
ylabel('amplitud');
xlabel('tiempo');
```



Observando las señales *x3*[*n*], *y3*[*n*] podemos afirmar que el sistema es lineal.

Ejercicio 1.2.2

Comprobar la no linealidad del sistema:

$$y[n] = x[n]^2$$

$$x[n] = \sin(2 * \pi * f * n)$$

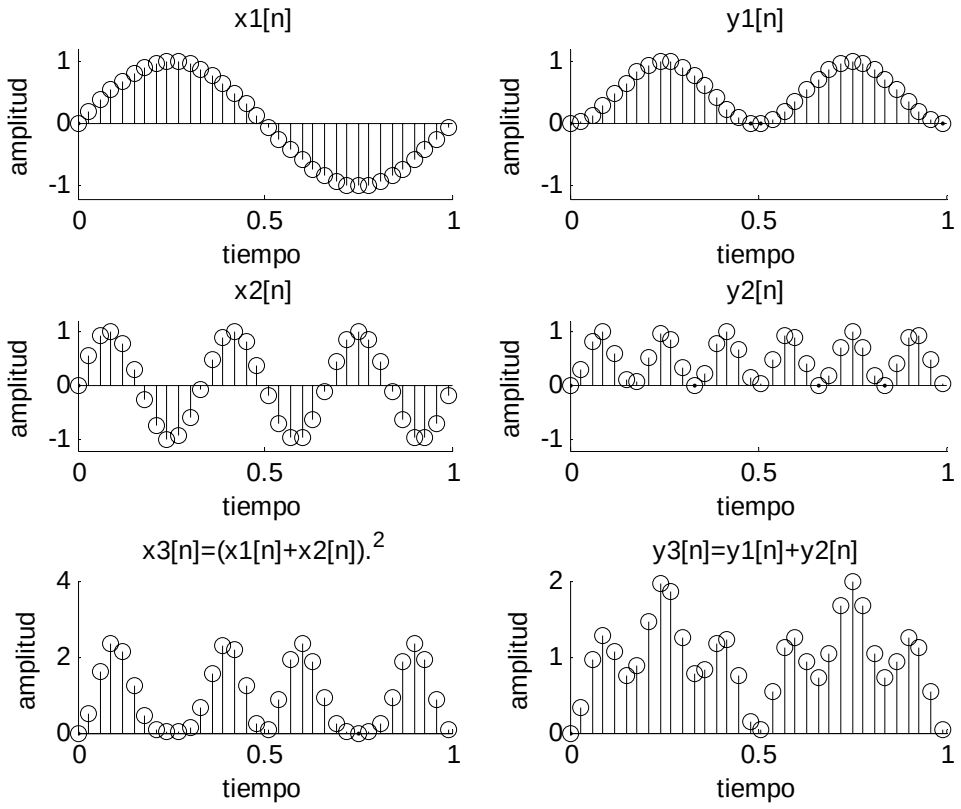
$x_1[n]$ Una señal seno de 1-Hz muestreada a 32 muestras por ciclo.

$x_2[n]$ Una señal seno de 3-Hz muestreada a 32 muestras por ciclo.

Script MATLAB:

```
%x1[n] onda seno de 1-Hz muestreada a 32 muestras por ciclo
%n=inicio:muestras/ciclo:fin;
n=0:0.03:1;
x1=sin(2*pi*1*n);
subplot(3,2,1);
stem(n,x1,'k')
title('x1[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
y1=x1.^2;
subplot(3,2,2);
stem(n,y1,'k')
title('y1[n]');
ylabel('amplitud');
xlabel('tiempo');
```

```
axis([0 1 -1.2 1.2])
%x2[n] onda seno de 3-Hz muestreada a 32 muestras por ciclo
n=0:0.03:1;
x2=sin(2*pi*3*n);
subplot(3,2,3);
stem(n,x2,'k')
title('x2[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
y2=x2.^2;
subplot(3,2,4);
stem(n,y2,'k')
title('y2[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%x3[n]
subplot(3,2,5);
stem(n,(x1+x2).^2,'k')
title('x3[n]=(x1[n]+x2[n]).^2');
ylabel('amplitud');
xlabel('tiempo');
%y3[n]
subplot(3,2,6);
stem(n,y1+y2,'k')
title('y3[n]=y1[n]+y2[n]');
ylabel('amplitud');
xlabel('tiempo');
```



Observando las señales *x3*[*n*], *y3*[*n*] se comprueba que el sistema no es lineal.

Sistemas Invariantes en el Tiempo

Un sistema invariante en el tiempo es uno donde un tiempo (*delay*) o cambio (*shift*) en la secuencia de entrada causa un equivalente tiempo de retraso o cambio en la secuencia de salida.

Ejemplo:
Un sistema nos da la salida $y[n]$ al aplicar una entrada $x[n]$

$$x[n] \quad \xrightarrow{resulta} \quad y\ [n]$$

Para que este sistema sea invariante en el tiempo, la entrada debe ser una versión con retraso de la señal original $x[n]$ así:

$$x[n]' = x[n+k] \quad \xrightarrow{resulta} \quad y\ [n]' = y[n+k]$$

Donde *k* representa el retraso (*delay*) o cambio (*shift*).

Analizando sistema invariante en el tiempo con MATLAB

Ejercicio 1.2.3
Asumiremos que nuestra entrada es $x[n]$, señal senoidal con Amplitud de A=1, frecuencia f=1-Hz y la versión de retraso es:

$$x[n+4] = x'[n]$$

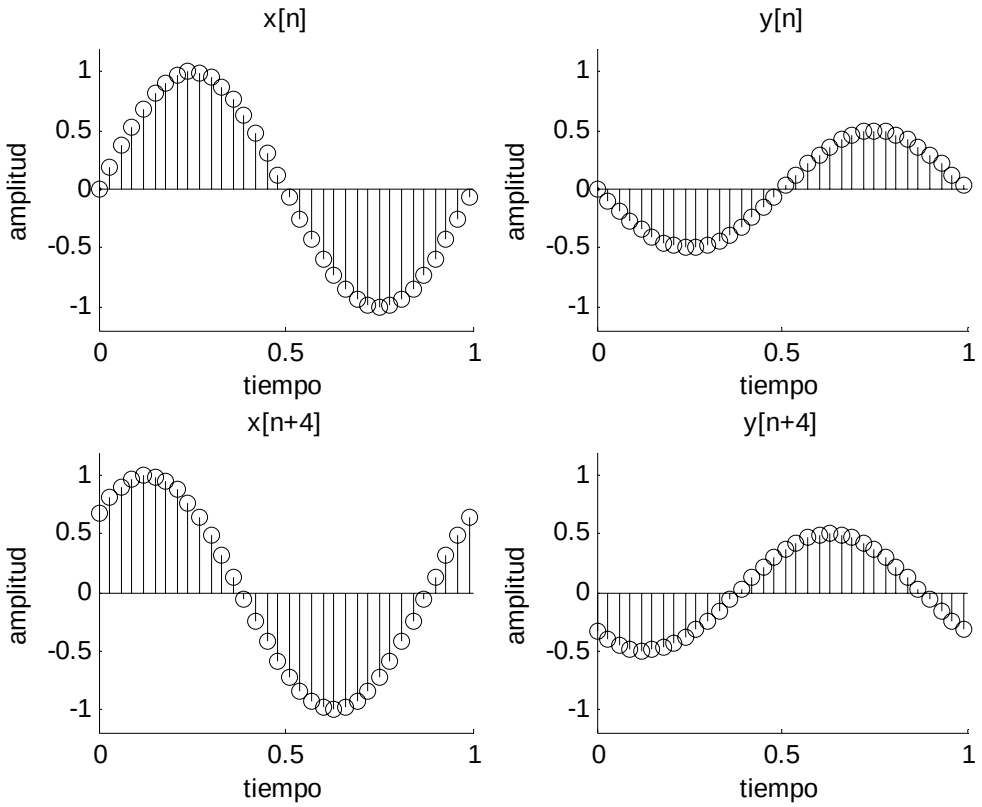
$$x[n] = A.\sin(2.\pi.f.n)$$

$$y[n] = -\frac{x[n]}{2}$$

Script MATLAB:

```
%x[n] onda seno de 1-Hz muestreada a 32 muestras por ciclo
%n=inicio:muestras/ciclo:fin;
n=0:0.03:1;
x=sin(2*pi*1*n);
subplot(2,2,1);
stem(n,x,'k')
title('x[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
y=-1/2*x;
subplot(2,2,2);
stem(n,y,'k')
title('y[n]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%x'[n]=x[n+4] versión con retraso de la original x[n]
n=0:0.03:1;
%4x0.03=0.12  0.03 es el muestreo de la señal
x2=sin(2*pi*1*(n+0.12));
subplot(2,2,3);
stem(n,x2,'k')
```

```
title('x[n+4]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
y2=-1/2*x2;
subplot(2,2,4);
stem(n,y2,'k')
title('y[n+4]');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
```



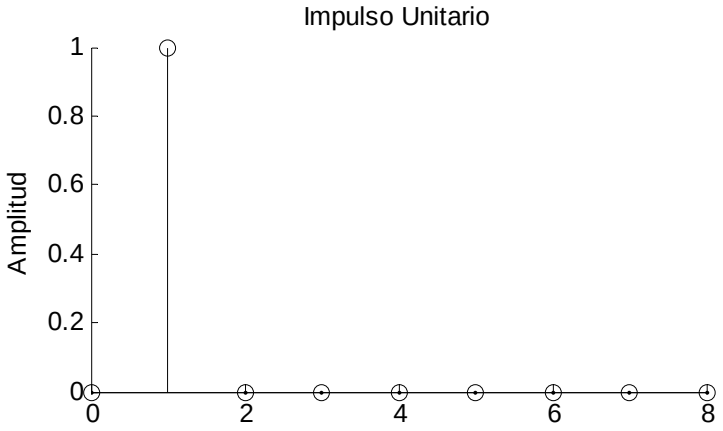
Secuencias fundamentales

En procesamiento digital de señales se usan varios tipos de secuencias elementales para propósitos de análisis, representación o para descripción de señales mas complicadas.

Impulso unitario o muestra unitaria

El impulso unitario (*unit sample*) denotado por $\delta[n]$ esta definido por:

$$\delta[n] = \begin{cases} 1 & n=0 \\ 0 & n\neq 0 \end{cases}$$

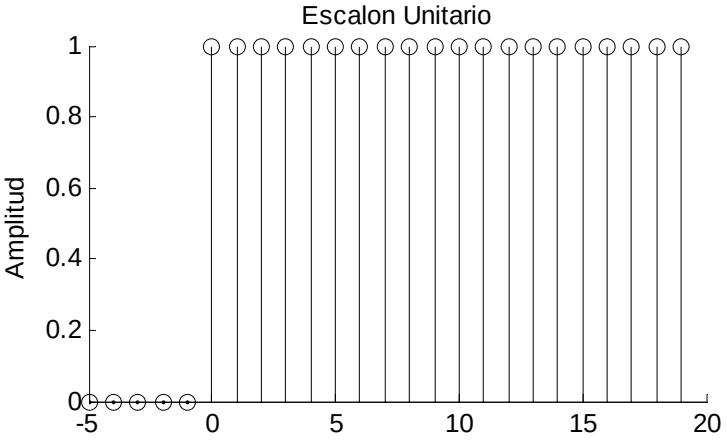


Es una de las señales discretas más simples.

$$\delta[n-k] = \begin{cases} 1 & n=k \\ 0 & n\neq k \end{cases}$$

Escalón Unitario

$$u[n] = \begin{cases} 1 & n\geq 0 \\ 0 & n<0 \end{cases}$$



El impulso unitario discreto $\delta[n]$ es la primera diferencia del escalón unitario discreto

$$\delta[n] = u[n] - u[n-1]$$

El escalón unitario (*step unit*) es la sumatoria del impulso unitario

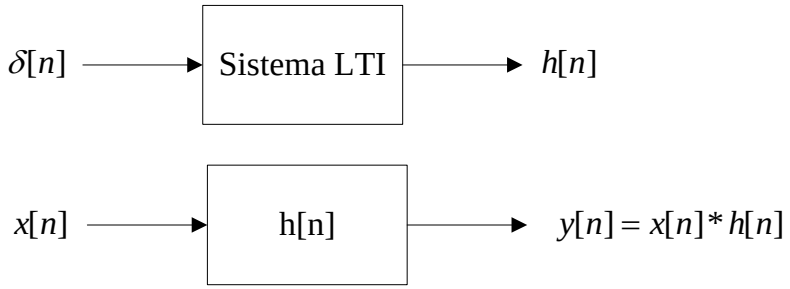
$$u[n] = \sum_{n=0}^{\infty} \delta[n]$$

1.3 Convolución

La convolución es una vía matemática para combinar dos señales y formar una tercera señal. Es la técnica más simple y más importante en *DSP*, la convolución se usa para implementar los filtros digitales FIR.
La convolución es usada para describir la relación entre las tres señales de interés: señal de entrada, señal de salida, respuesta al impulso.
La respuesta al impulso de un sistema LTI discreto esta dada por $h[n]$, la operación matemática de la convolución es:

$$y[n] = LTI(x[n]) = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

La respuesta al impulso es la salida de un sistema LTI, debido a una entrada de impulso aplicada en el tiempo n=0.
La respuesta al impulso $h[n]$ caracteriza por completo el comportamiento de cualquier sistema LTI en el dominio del tiempo.



(*) Representa convolución en una ecuación

Si el sistema que se esta considerando es un filtro, la respuesta al impulso recibe el nombre de, *filter kern*el, *convolución kernel*, o simplemente *kernel*.
En procesamiento de imágenes la respuesta al impulso se llama función puntual de despliegue (*point sprend funtion*).

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Ecuación Convolución

Sumatoria de convolución

Si consideramos el producto de una señal $x[n]$ y una secuencia de impulsos recorridos en el tiempo tenemos:

$$x[n] \cdot \delta[n-k] = x[k] \cdot \delta[n-k]$$

Donde *n* indica tiempo, *x*[*k*] representa el valor de la señal *x*[*n*] en el tiempo *k*

$$x[n] = \ldots + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \ldots$$

$$x[n] = \sum_{k=-\infty}^{\infty} x[k].\delta[n-k]$$

Sea ***H*** el operador que denota el sistema al cual se aplica la entrada x[n], siendo:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k].\delta[n-k]$$

$$y[n] = H\left\{ \sum_{k=-\infty}^{\infty} x[k].\delta[n-k] \right.$$

Aplicando la propiedad de linealidad:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] H\left\{ \ \delta[n-k] \ \right\}$$

$$H\left\{ \ \delta[n-k] \ \right\} = h_k[n]$$

Respuesta al impulso del sistema recorrido en el tiempo

La salida debido a un impulso recorrido en el tiempo es una versión corrida en el tiempo de la salida debido a un impulso

$$h_k[n] = h_0[n-k]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k]$$

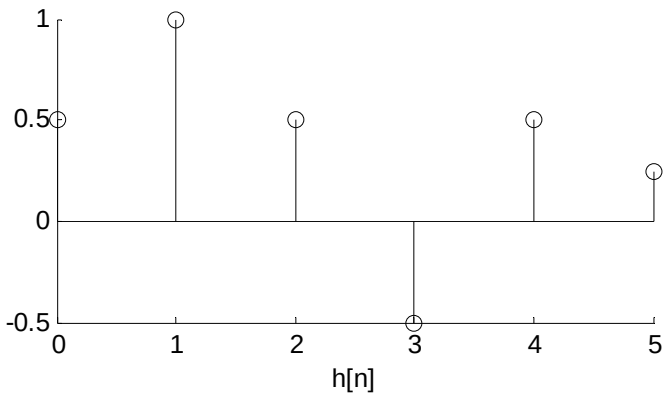
Sumatoria de convolución

Así el sistema LTI esta dado por la suma ponderada de la respuesta al impulso recorrido en el tiempo

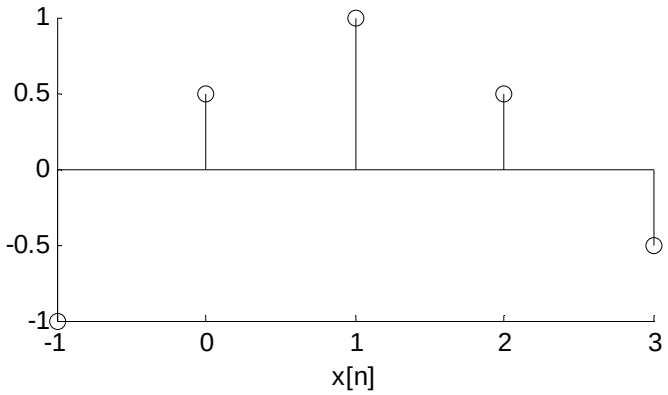
$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k]$$

Por Ejemplo:

La respuesta al impulso del sistema LTI es:



La entrada se presenta como una suma ponderada de impulsos recorridos en el tiempo



Aplicando la formula de la convolución $x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k]$

obtenemos:

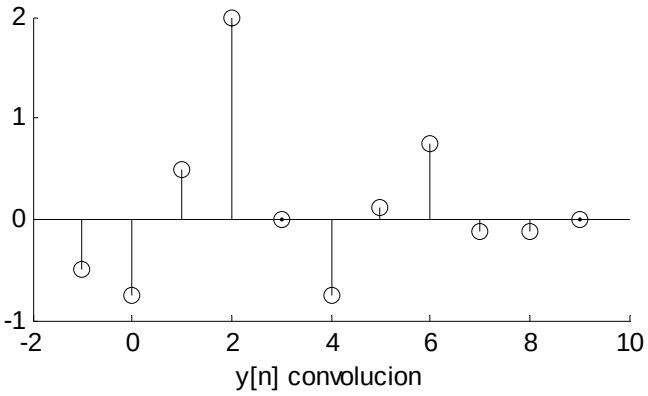
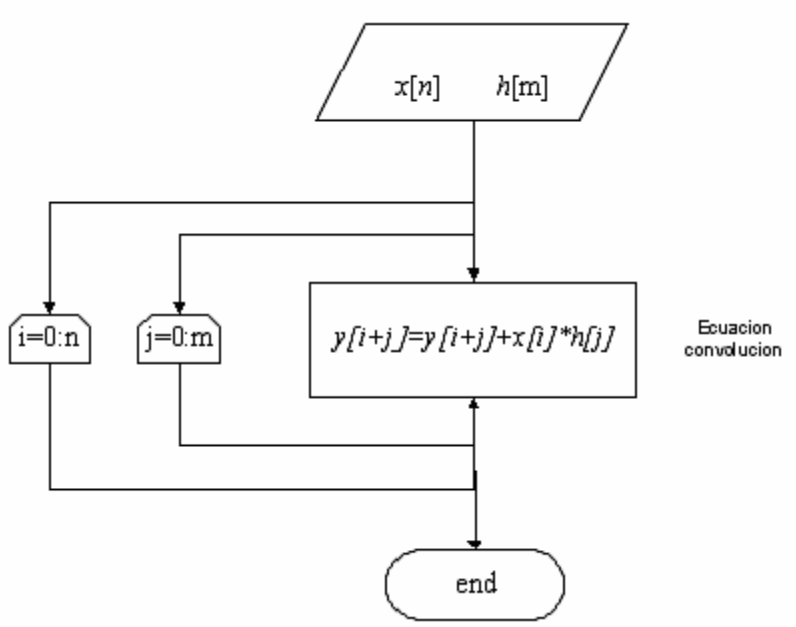


Diagrama Flujo Convolución (algoritmo Input-Side)



Analizando convolución con MATLAB

Ejercicio 1.3.1

Dada las siguientes secuencias:

$$x[n] = [3,11,7,\underset{\bullet}{0},-1,4,2] \quad -3 \le n \le 3$$

$$h[n] = [2,\underset{\bullet}{3},0,-5,2,1] \quad -1 \le n \le 4$$

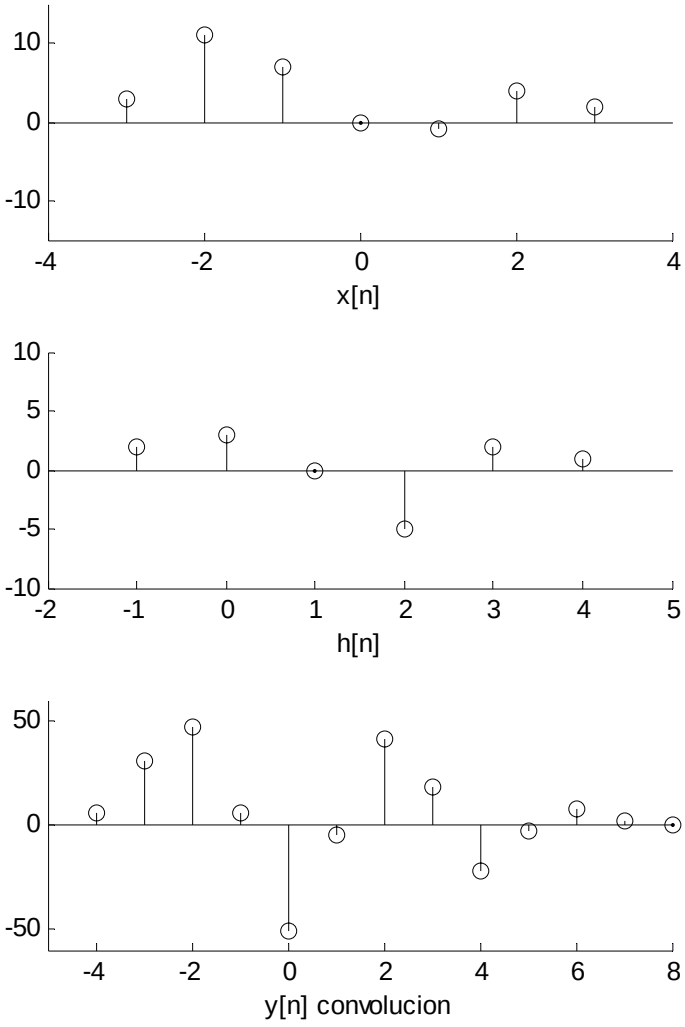
Determinar la convolución

$$y[n] = x[n] * h[n]$$

Script MATLAB:

```
x = [3 11 7 0 -1 4 2];
xm = length(x);
%desde h[n] 10
h = [2 3 0 -5 2 1];
hm = length(h);
%limpio array de salida
for i = 1:(xm + hm)
    y(i) = 0;
end
%matlab no permite crear vectores desde x(0)
for i = 1:xm
    for j = 1:hm
        %algoritmo de convolución
        y(i+j-1) = y(i+j-1) + x(i)*h(j)
    end
end
%plot de las señales y ya es la señal convuelta
subplot(3,1,1);
```

```
stem([-3:3],x,'k');
axis([-4 4 -15 15])
xlabel('x[n]');

subplot(3,1,2);
stem([-1:4],h,'k');
axis([-2 5 -10 10])
xlabel('h[n]');
subplot(3,1,3);
stem([-4:8],y,'k');
axis([-5 8 -60 60])
xlabel('y[n]');
```



Ejercicio 1.3.2

Matlab posee una función hecha para evaluar la convolución *conv(a,b)*, evaluando el ejercicio anterior con esta función sería:

Script MATLAB:

x = [3 11 7 0 -1 4 2];
h = [2 3 0 -5 2 1];
y=conv(x,h)

Resultado

y = 6 31 47 6 (-51) -5 41 18 -22 -3 8 2

Si comparamos en la ventana de comandos de Matlab la respuesta de los 2 ejercicios observaremos que es la misma.

Respuesta al impulso

La respuesta al impulso de un sistema puede ser determinada a través de una ecuación de diferencia, sustituyendo la entrada $x[n]$ por la función $\delta[n]$ y determinando la salida $y[n]$.

Ejemplo:
Determinar la respuesta al impulso. Asumir que un sistema discreto esta descrito por la ecuación de diferencia:

$$y[n] = b_1 y[n-1] + x[n]$$

Cambiando $x[n]$ por $\delta[n]$ tenemos

$$x[n] = \delta[n]$$
$$h[n] = b_1 h[n-1] + \delta[n]$$

Analizando *n*

n=0	$h[0] = 1$
n=1	$h[1] = b_1(1) + 0 = b_1$
n=2	$h[2] = b_1 b_1 + 0 = b_1^{\,2}$
n=3	$h[3] = b_1^{\,2} b_1 + 0 = b_1^{\,3}$
n=4	$h[4] = b_1^{\,3} b_1 + 0 = b_1^{\,4}$

De los resultados podemos ver que la forma general del impulso puede ser vista como:

$$h[n] = b_1^{\,n} \bullet u[n]$$

Cuando la respuesta al impulso del sistema es conocida, las características completas del sistema son conocidas.
La reacción del sistema a cualquier otra entrada puede ser determinada usando la *convolución.*

La salida del sistema puede entonces ser definida como

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Si la señal de entrada es

$$x[n] = a_1^{\,n} \bullet u[n]$$

$$h[n] = b_1^{\,n} \bullet u[n]$$

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

$$y[n] = \sum_{k=-\infty}^{\infty} a_1^{\,k} \bullet u[k] \bullet (b_1^{\,(n-k)} \bullet u[n-k])$$

$$y[n] = b_1^{\,n} \bullet \sum_{k=-\infty}^{n} \left(\frac{a_1}{b_1} \right)^k \bullet u[n])$$

Respuesta a la frecuencia

La respuesta a la frecuencia es una de las características más importantes de un sistema discreto, aunque no describe completamente al sistema como la ecuación de diferencia, la respuesta al impulso o la función de transferencia lo hacen, la respuesta a la frecuencia provee información importante sobre el estado de comportamiento ***estable*** del sistema.

Si empezamos considerando una señal senoidal analógica y muestreamos la señal usando un periodo de muestreo T_s:

$$x(t) = A \cdot \cos(\omega \cdot t) \ \textit{señal analógica}$$

$$x(n \cdot T_s) = A \cdot \cos(\omega \cdot n \cdot T_s) \ \textit{señal muestreada}$$

$$x(n \cdot T_s) = A \cdot \cos(\Omega \cdot n)$$

Donde Ω es la frecuencia digital

$$\Omega = \omega \cdot T_s = 2\pi \cdot f \cdot T_s = \frac{2\pi \cdot f}{f_s}$$

El rango de frecuencias para un sistema discreto no se extiende desde cero al infinito como es el caso de un sistema analógico, se debe recordar que el límite aceptable de frecuencias analógicas en un sistema discreto esta gobernado por el criterio de Nyquist y por lo tanto el rango esta definido por:

$$0 \leq f_d < f_s/2$$

o

$$0 \leq \Omega < \pi \quad \textit{rango frecuencia digital}$$

La frecuencia f_d esta en función de ambas frecuencias (*muestreo y digital*) del sistema

$$f_d = \frac{\Omega}{2\pi} \cdot f_s$$

Aunque una función seno o coseno es usada para determinar la respuesta a la frecuencia de un sistema, ambas funciones pueden ser descritas por exponenciales complejas así:

$$\cos[\Omega \cdot n] = \frac{e^{j \cdot \Omega \cdot n} + e^{-j \cdot \Omega \cdot n}}{2}$$

$$\sin[\Omega \cdot n] = \frac{e^{j \cdot \Omega \cdot n} - e^{-j \cdot \Omega \cdot n}}{2 \cdot j}$$

En efecto, todas las funciones periódicas pueden ser representadas por estas exponenciales complejas, incluso los valores constantes pueden ser representados por exponenciales de frecuencia cero. Por consiguiente cuando determinemos la respuesta a la frecuencia de un sistema discreto es común considerar el manejo de funciones exponenciales complejas, así:

$$x[n] = e^{j \cdot \Omega \cdot n}$$

La salida de un sistema discreto puede entonces hallarse convolucionando esta señal con la respuesta al impulso. La salida seria entonces así:

$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot e^{j \cdot \Omega (n-k)}$$

$$y[n] = e^{j \cdot \Omega \cdot n} \sum_{k=-\infty}^{\infty} h[k] \cdot e^{-j \cdot \Omega \cdot k}$$

O puede ser escrita así:

$$y[n] = x[n] * H(e^{j \cdot \Omega}) \ \textit{convolución}$$

Donde $H(e^{j \cdot \Omega})$ es definida como la respuesta a la frecuencia del sistema, si comparamos $H(e^{j \cdot \Omega})$ con la función de transferencia de un sistema discreto veremos una similitud

$$H(e^{j\cdot\Omega}) = \sum_{k=-\infty}^{\infty} h[k]\cdot e^{-j\cdot\Omega\cdot k}$$

$$H(z) = \sum_{k=-\infty}^{\infty} h[k]\cdot z^{-k} \quad \textit{función transferencia}$$

Entonces podemos definir la respuesta a la frecuencia en términos de la función de transferencia, simplemente reemplazando z por $e^{j\cdot\Omega}$, así:

$$H(e^{j\cdot\Omega}) = H(z)\Big|_{e^{j\cdot\Omega\cdot}}$$

Ejemplo:
Determinar la respuesta a la frecuencia del siguiente sistema:

$$y[n] = b_1 y[n-1] + x[n]$$

Primero debemos hallar la respuesta al impulso del sistema, cambiando $x[n]$ por $\delta[n]$ tenemos

$$x[n] = \delta[n]$$

$$h[n] = b_1 h[n-1] + \delta[n]$$

Analizando la respuesta al impulso con algunos valores de *n* tenemos

n=0	$h[0] = 1$
n=1	$h[1] = b_1(1) + 0 = b_1$
n=2	$h[2] = b_1 b_1 + 0 = b_1^{\,2}$
n=3	$h[3] = b_1^{\,2} b_1 + 0 = b_1^{\,3}$
n=4	$h[4] = b_1^{\,3} b_1 + 0 = b_1^{\,4}$

De los resultados podemos ver que la forma general del impulso puede ser escrita así:

$$h[n] = b_1^{\,n} \bullet u[n]$$

Empleando la *transformada-z* a $h[n]$ tenemos

$$H(z) = \frac{z}{(z-b_1)} \quad \textit{función transferencia}$$

$$H(z) = \frac{Y(z)}{X(z)}$$

Asumiendo que $b_1 = 0.8$ en el sistema y reemplazando z por $e^{j\cdot\Omega}$, tenemos

$$H(e^{j\cdot\Omega}) = \frac{e^{j\cdot\Omega}}{(e^{j\cdot\Omega} - 0.8)}$$

Usando la relación de Euler podemos escribir la ecuación de arriba así

$$H(e^{j\cdot\Omega}) = \frac{\cos(\Omega) + j\sin(\Omega)}{(\cos(\Omega) + j\sin(\Omega) - 0.8)} \qquad 0 \le \Omega < \pi \;\; \textit{rango frecuencia digital}$$

Evaluando entonces $H(e^{j\cdot\Omega})$, que es la respuesta a la frecuencia, tenemos

Respuesta a la frecuencia

frecuencia	***Magnitud***	***Fase(grados)***
0	*5*	*0°*
$\pi/4$	*1.4022*	*-52.48°*
$\pi/2$	*0.7809*	*-38.65°*
$3\pi/4$	*0.6007*	*-19.86°*
π	*0.5556*	*0°*

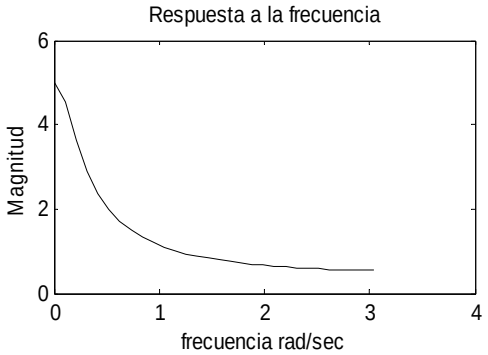
Nota: Para convertir de radianes a grados hay que multiplicar por $180/\pi$

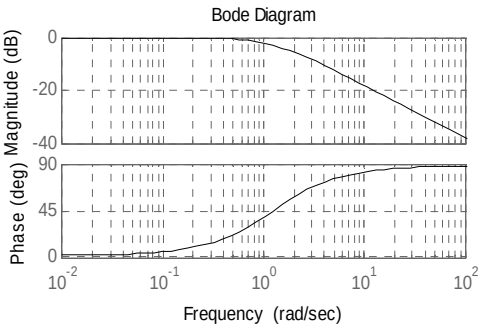
Analizando respuesta a la frecuencia con MATLAB

Ejercicio 1.3.3

Graficar la respuesta a la frecuencia de la función de transferencia

$$H(z) = \frac{z}{(z - 0.8)}$$





Script MATLAB:

```
n = [0 1];
d = [-0.8 1]
[H,w]=freqz(n,d,30)
subplot(2,1,1);
plot(w,abs(H))
title 'Respuesta a la frecuencia'
ylabel 'Magnitud'
xlabel 'frecuencia rad/sec'
%gráfica trazas de bode o logarítmicas
subplot(2,1,2);
bode(n,d);
grid
```

1.4 Ecuaciones de diferencia

Un sistema LTI discreto puede ser descrito por ecuaciones de diferenciales lineales de orden N con coeficientes constantes.

$$\sum_{k=0}^{N} a_k \bullet y[n-k] = \sum_{k=0}^{M} b_k \bullet x[n-k]$$

$$y[n] = \sum_{k=0}^{M} b_k \bullet x[n-k] - \sum_{k=1}^{N} a_k \bullet y[n-k]$$

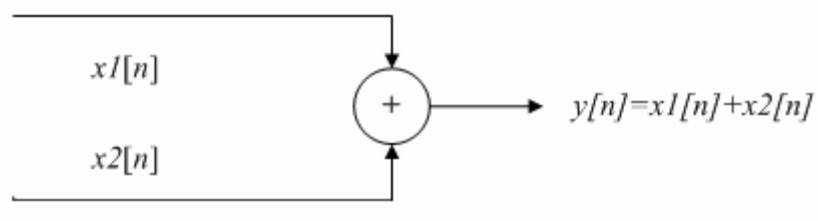
Se puede conocer completamente el sistema si se conocen los coeficientes a_k y b_k.
La salida $y[n]$ esta en función de valores presentes y pasados de la entrada $x[n]$ y de los valores pasados de la salida.
A este tipo de sistemas se les llama sistemas *recursivos*.

Representación de sistemas LTI discretos en diagramas de bloques

Un diagrama de bloques es una interconexión de operaciones elementales que actúan sobre la señal de entrada, describe además como se ordenan los cálculos u operaciones internas del sistema.

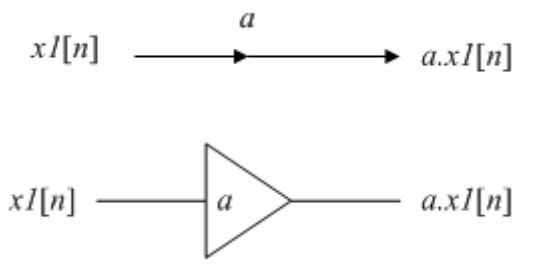
Diagramas básicos para interconectar sistemas

Suma



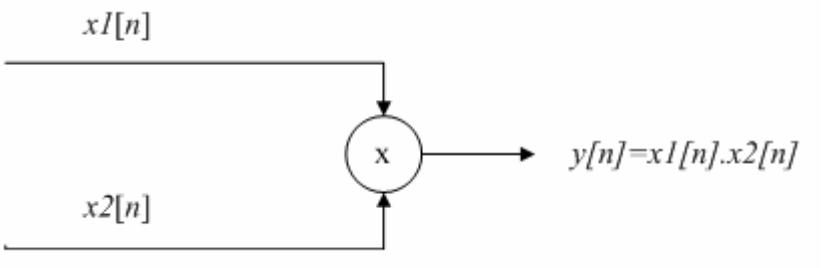
La figura muestra, un sistema sumador que realiza la suma de 2 señales para formar otra señal, la operación de suma es un sistema sin memoria.

Multiplicación escalar



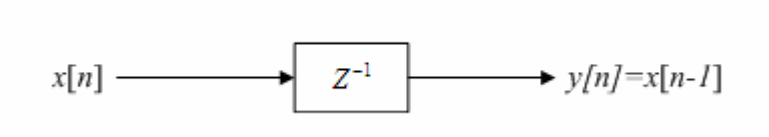
La figura muestra, cuando un escalar es aplicado a la señal, la multiplicación de un escalar es un sistema sin memoria.

Multiplicación



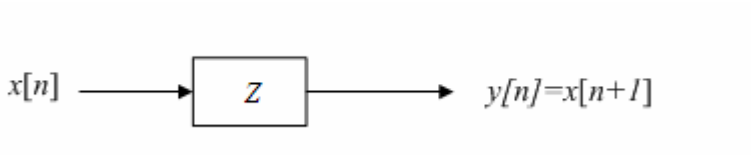
La figura muestra que la multiplicación de 2 señales forma otra señal, la operación de multiplicación es también un sistema sin memoria.

Elemento de retraso unitario



La figura representa un sistema especial que simplemente retrasa una señal cuando pasa a través de el, es un sistema con memoria.

Elemento de corrimiento unitario

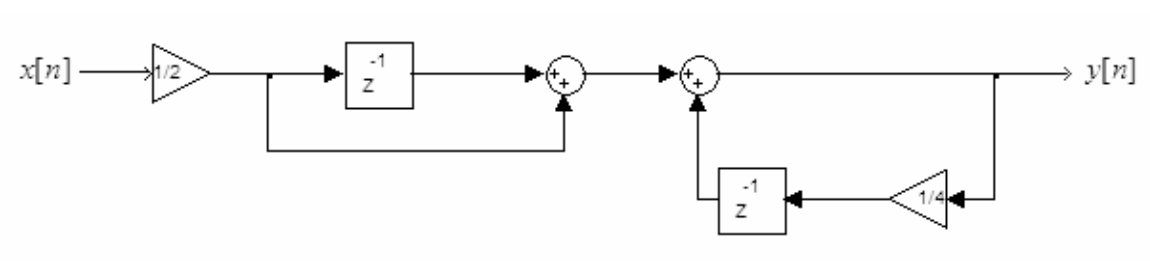


El sistema hace que la señal se adelante $x[n+1]$

Ejercicio 1.4.1

Construir un diagrama de bloques que represente el siguiente sistema

$$y[n] = \frac{1}{4}y[n-1] + \frac{1}{2}x[n] + \frac{1}{2}x[n-1]$$



1.5 Transformada discreta de Fourier

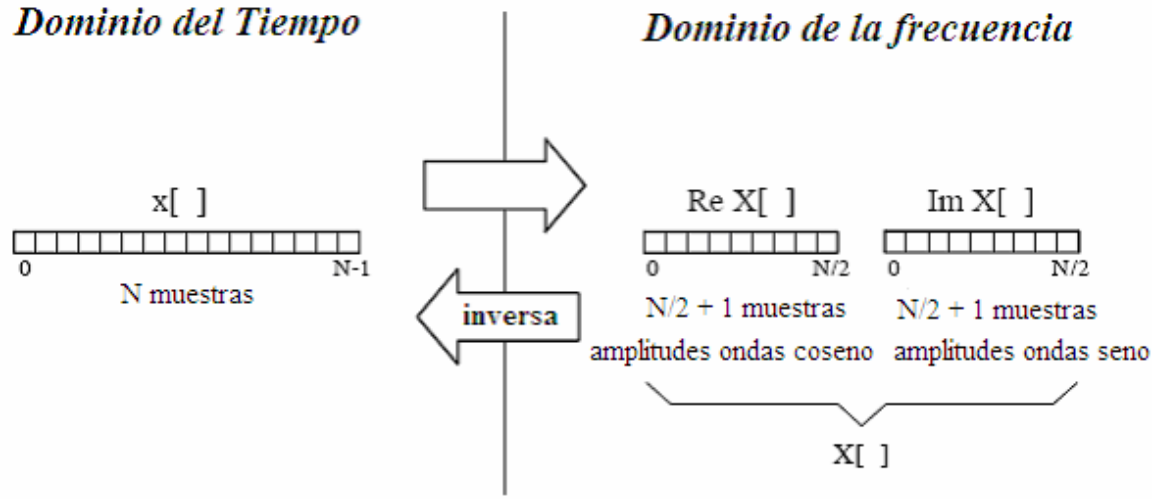
El análisis de Fourier es una familia de técnicas matemáticas, basadas en la descomposición de señales dentro de sinusoides.
La transformada discreta de Fourier ***DFT*** se usa para las señales digitalizadas con números reales.

Tipos de transformadas discretas de Fourier

DTFT (*transformada discreta de fourier en el tiempo*), se usa en señales no periódicas

DFT (*transformada discreta de fourier*), se usa en señales periódicas

Notación y formado de la DFT real



Como se observa en la figura anterior la DFT cambia una señal de entrada de *N* puntos, a dos señales de salida con *N/2+1* puntos.
El dominio de la frecuencia contiene exactamente la misma información que el dominio del tiempo, pero de diferente forma. En el dominio del tiempo el número de muestras se representa con la variable *N*, y puede ser cualquier entero con potencia 2 usualmente.

Notación

Letras minúsculas se usan para representar una señal en el dominio del tiempo

$$x[n] \quad y[n] \quad z[n]$$

Letras mayúsculas para señales en el dominio de la frecuencia

$$X[f] \quad Y[f] \quad Z[f]$$

$$X[] = \text{Re}\, X[] + \text{Im}\, X[]$$

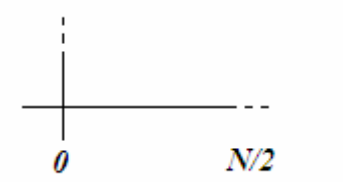
ReX[ ] es la parte real, representa las amplitudes de las ondas coseno.
ImX[ ] es la parte imaginaria, representa las amplitudes de las ondas seno.

La parte imaginaria *ImX*[ ] da lugar a la *DFT compleja*.

Variable independiente del dominio de la frecuencia

El eje horizontal en el dominio del tiempo puede ser representado de 4 maneras diferentes:

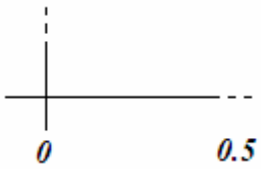
1) El eje horizontal es etiquetado de *0* a *N/2*



ReX[k] *ImX*[k] $\Rightarrow$ índice *k* es un entero

Método utilizado por los programadores.

2) El eje horizontal es etiquetado como una fracción de un ciclo de muestreo, esto significa que los valores a lo largo del eje horizontal siempre recorrerán de *0* a *0.5*, los datos discretos solo pueden contener frecuencias de *DC* de medio ciclo.



ReX[*f*] *ImX*[*f*] $\Rightarrow$ índice *f*

Se usa el índice *f* como notación

$$f = \frac{k}{N}$$

$$k = \frac{N}{2}$$

$$f = \frac{N/2}{N} = 1/2 = 0.5$$

3) Es similar al anterior a excepción que el eje horizontal es multiplicado por 2π , el índice usado es w (*frecuencia fundamental*).

$$ReX[w] \quad ImX[w] \quad \Rightarrow \text{ índices } w$$

$$w = 2 \bullet \pi \bullet f$$

4) Es etiquetado el eje horizontal en términos de las frecuencias analógicas usadas en una aplicación particular.

Por Ejemplo:
Si la frecuencia de muestreo es *10-khz*, el gráfico correrá de *0* a *5-khz.*

Ejercicio 1.5.1
Una señal coseno puede ser escrita con estas notaciones:

$$1)\ c[n] = \cos(2\pi k\, {}^{n}\!/_{N})$$
$$2)\ c[n] = \cos(2\pi f n)$$
$$3)\ c[n] = \cos(wn)$$

Síntesis, cálculo de la DFT inversa

$$X[f] \xrightarrow{inversa} x[n]$$

La ecuación de la síntesis es:

$$x[i] = \sum_{k=0}^{N-1} X[k] \bullet e^{j 2\pi k\, {}^{i}\!/_{N}}$$

$i=0,\ldots,N-1$
$k=0,\ldots,N/2$

$$x[i] = \sum_{k=0}^{N/2} \mathrm{Re}\,\overline{X}[k] \bullet \cos(2\pi k\ {}^{i}\!/_{N}) + \sum_{k=0}^{N/2} \mathrm{Im}\,\overline{X}[k] \bullet \sin(2\pi k\ {}^{i}\!/_{N})$$

En la ecuación $\mathrm{Re}\,\overline{X}[k]$ e $\mathrm{Im}\,\overline{X}[k]$ son matrices, porque las amplitudes se necesitan en la síntesis.
Normalizando quedaría:

$$\mathrm{Re}\,\overline{X}[k] = \frac{\mathrm{Re}\,X[k]}{N/2}$$

$$\mathrm{Im}\,\overline{X}[k] = -\frac{\mathrm{Im}\,X[k]}{N/2}$$

A excepción de

$$\mathrm{Re}\,\overline{X}[0] = \frac{\mathrm{Re}\,X[0]}{N}$$

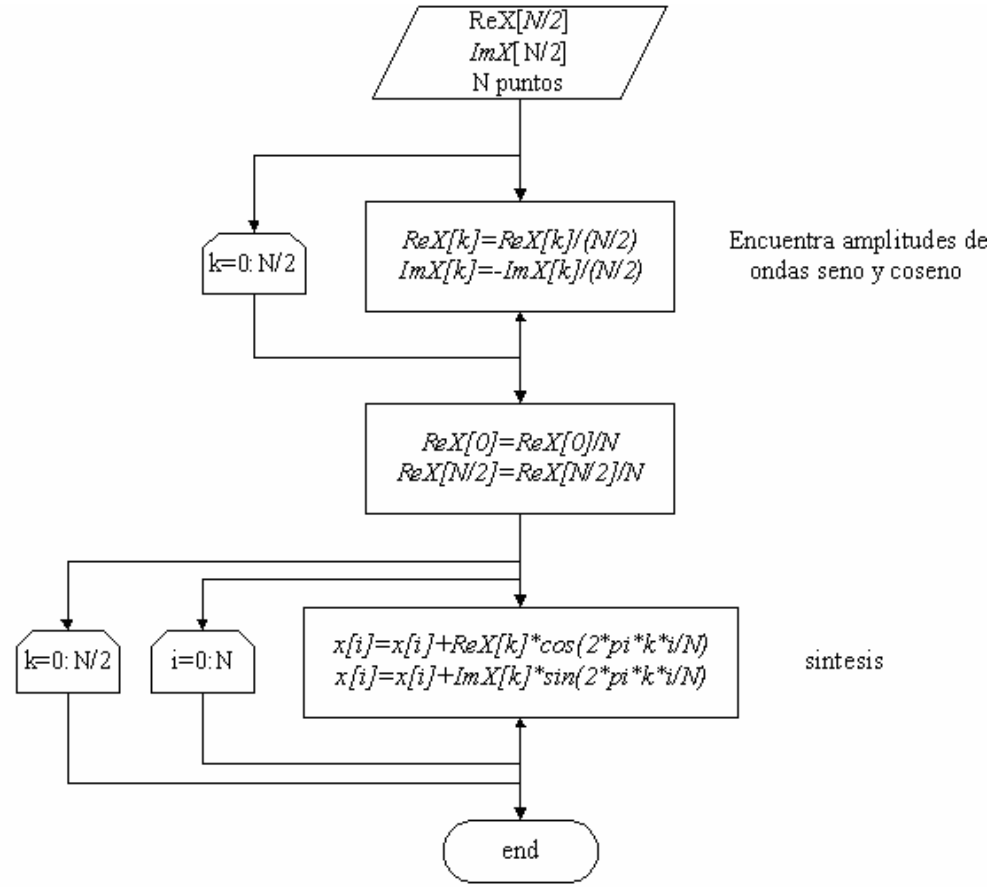
$$\mathrm{Re}\,\overline{X}[N/2] = \frac{\mathrm{Re}\,X[N/2]}{N}$$

N es número de muestras

- Para programar hay que seguir estos paso:

1) Dividir para *N/2*
2) Cambiar de signo (-) a $\mathrm{Im}\,\overline{X}[k]$
3) Dividir $\mathrm{Re}\,\overline{X}[0]$ y $\mathrm{Re}\,\overline{X}[N/2]$ para *N*

Diagrama de flujo DFT inversa



Análisis, cálculo de la DFT

$$x[n] \longrightarrow X[f]$$

La DFT puede ser calculada de varias formas, por ecuaciones simultaneas, por correlación, y por medio de la transformada rápida de Fourier *FFT*.

$$X[i] = \sum_{i=0}^{N-1} x[i] \bullet e^{-j2\pi k \, i/N}$$

De la relación de Euler tenemos:

$$e^{-j2\pi k \, i/N} = \cos(2\pi k \, i/N) - j \bullet \sin(2\pi k \, i/N)$$

DFT forma rectangular

$$X[i] = \sum_{i=0}^{N-1} x[i] \bullet \cos(2\pi k \, i/N) - j \sum_{i=0}^{N-1} x[i] \bullet \sin(2\pi k \, i/N)$$

i= índice muestras de entrada dominio del tiempo,i=0,1,…,N-1

$j = \sqrt{-1}$

N = número muestras de la secuencia de entrada y número de puntos de la frecuencia de salida de la DFT

$$\text{Re } X[i] = \sum_{i=0}^{N-1} x[i] \bullet \cos(2\pi k \, i/N)$$

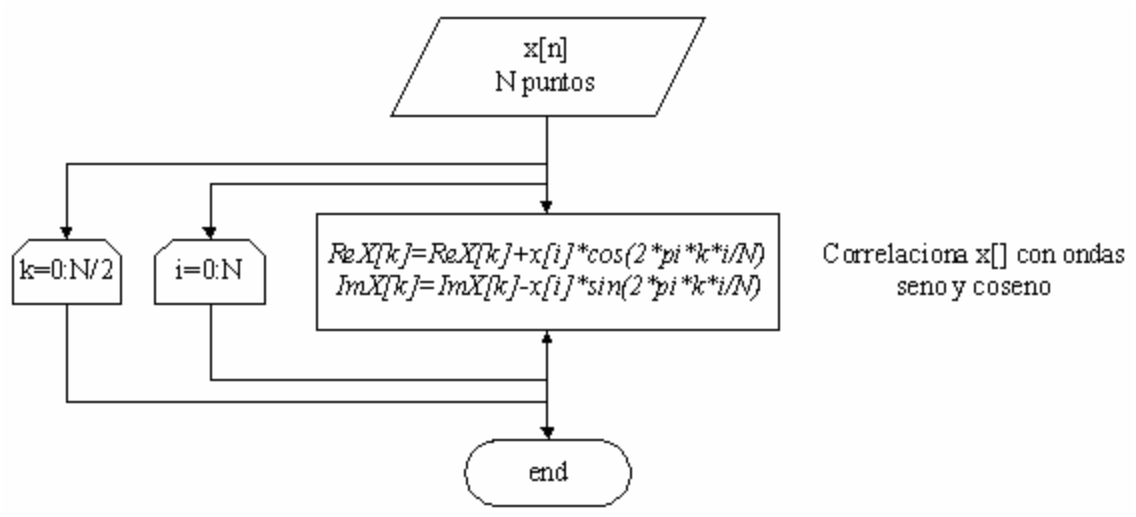
$$\text{Im } X[i] = -\sum_{i=0}^{N-1} x[i] \bullet \sin(2\pi k \, i/N)$$

$$X[i] = \text{Re } X[i] + \text{Im } X[i]$$

k=0,..,N/2

i=0,…,N-1

Diagrama de flujo de la DFT por correlación



Dualidad

Las ecuaciones de la síntesis y análisis son muy similares, mueven de un dominio a otro, los valores son multiplicados por las funciones base y los productos son sumados.
La única diferencia es el resultado:

Dominio del tiempo $\longrightarrow$ señal de *N* puntos

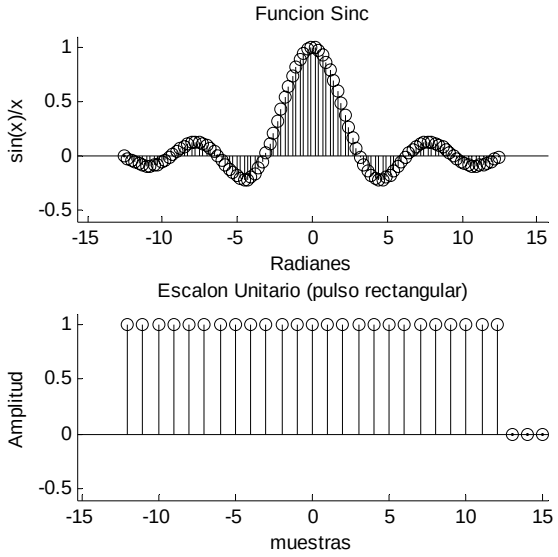
Dominio de la frecuencia $\longrightarrow$ 2 señales de *N/2 +1* puntos c/u.

Ejemplo:
Un punto del dominio del tiempo equivale a una sinusoide en el dominio de la frecuencia y vise-versa.

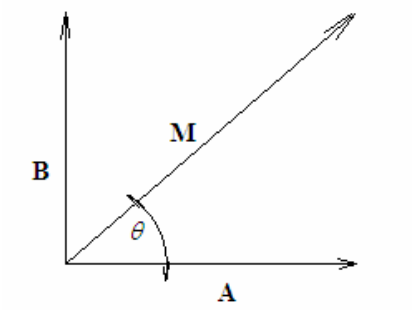
Función Sinc

Para muchas formas de onda del dominio del tiempo hay una forma de onda correspondiente en el dominio de la frecuencia.
Un pulso rectangular en el dominio del tiempo coincide con una función *sinc* en el dominio de la frecuencia.

$$Sinc(a) = \frac{\sin(\pi a)}{\pi a}$$



Notación Polar



Rectangular $\longrightarrow$ $A \bullet \cos(x) + B \bullet \sin(x) = M \bullet \cos(x + \theta)$

Polar $\longrightarrow$ $M = \left(A^2 + B^2\right)^{1/2}$ magnitud

Polar $\longrightarrow$ $\theta = \tan\left(B/A\right)$ ángulo fase

$$MagX[k] = \left(\mathrm{Re}\, X[k]^2 + \mathrm{Im}\, X[k]^2\right)^{1/2}$$

$$PhaseX[k] = \tan^{-1}\left(\frac{\text{Im}\,X[k]}{\text{Re}\,X[k]}\right)$$

$$\text{Re}\,X[k] = MagX[k] \bullet \cos(PhaseX[k])$$

$$\text{Im}\,X[k] = MagX[k] \bullet \sin(PhaseX[k])$$

DTFT

La transformada de tiempo discreto de Fourier es para señales no periódicas discretas.

$$\text{Re}\,X(w) = \sum_{n=-\infty}^{\infty} x[n] \bullet \cos(wn)$$

$$\text{Im}\,X(w) = \sum_{n=-\infty}^{\infty} x[n] \bullet \sin(wn)$$

$$x[n] = \frac{1}{\pi}\int_0^{\pi} \left(\text{Re}\,X(w) \bullet \cos(wn) - \text{Im}\,X(w) \bullet \sin(wn)\right) dw$$

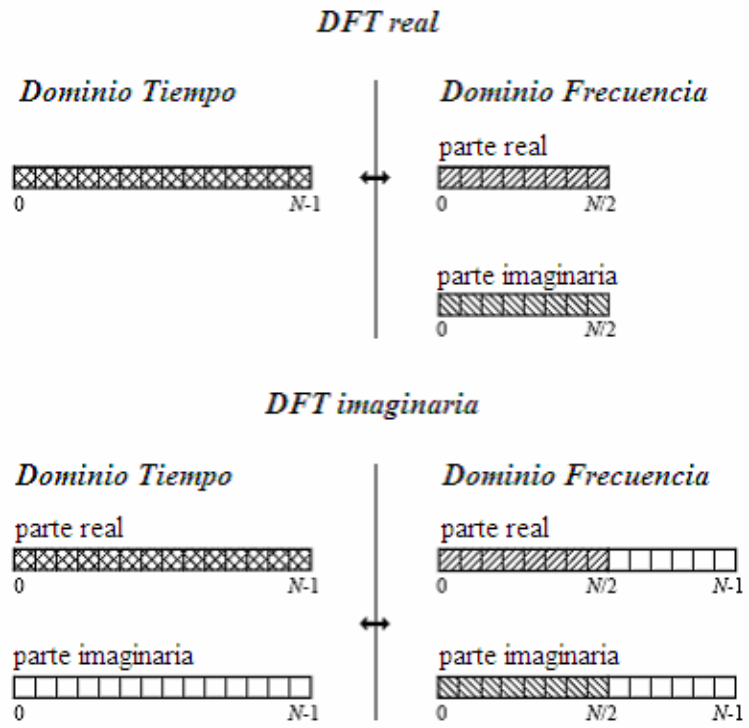
$$f = \frac{k}{N} \qquad k = \frac{N}{2} \qquad w = 2 \bullet \pi \bullet f$$

1.6 Transformada rápida de Fourier FFT

La *FFT* presentada por *Tukey y Cooley* (IBM) en 1965, es otra manera de calcular la *DFT*, la *FFT* requiere pocas líneas de código, es uno de los algoritmos más complicados en *DSP*.
La *FFT* es para el procesamiento digital de señales lo que el transistor es para la electrónica.
Existen varias rutinas de la *FFT* que pueden ser fácilmente implementadas, sin entender el trabajo interno de esta.
La *FFT* esta basada en la *DFT* compleja, que es una versión más sofisticada de la *DFT real*.

Comparación de la DFT compleja y real

La figura de abajo muestra el almacenamiento de datos de la *DFT real* e *imaginaria*.



Como trabaja la FFT ?

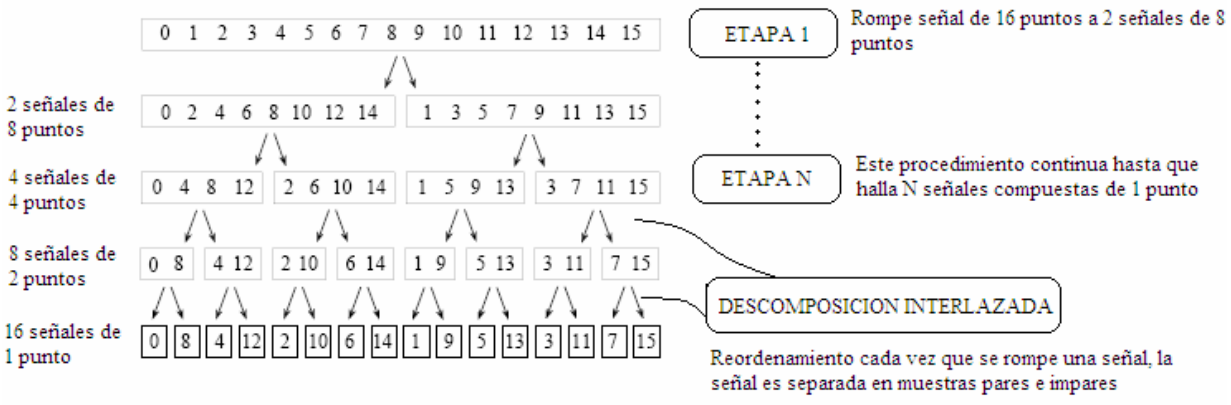
En notación compleja, tanto el dominio del tiempo, como el dominio de la frecuencia tienen una señal de *N* puntos complejos. Cada uno de estos puntos complejos esta compuesto de 2 números reales, una parte real y una parte imaginaria.

Por ejemplo:
Cuando hablamos sobre una muestra compleja *X*[42], se refiere a la combinación de *ReX*[42] e *ImX*[42] (*cada variable compleja sostiene 2 partes*).

La *FFT* opera descomponiendo en *N* puntos una señal del dominio del tiempo, a *N* señales del dominio de la frecuencia cada una compuesta de un simple punto.
El segundo paso es calcular los *N espectros de frecuencia* correspondientes a esas *N* señales del dominio del tiempo. Últimamente se sintetiza los *N* espectros a un solo espectro de frecuencias.

Ejemplo:

El gráfico de abajo ilustra la descomposición usada por la *FFT*



Hay $\log_2 N$ etapas requeridas en esta descomposición.

Por ejemplo:

Una señal de 16 puntos $\longrightarrow (2)^4$ requiere 4 etapas

Una señal de 512 puntos $\longrightarrow (2)^7$ requiere 7 etapas

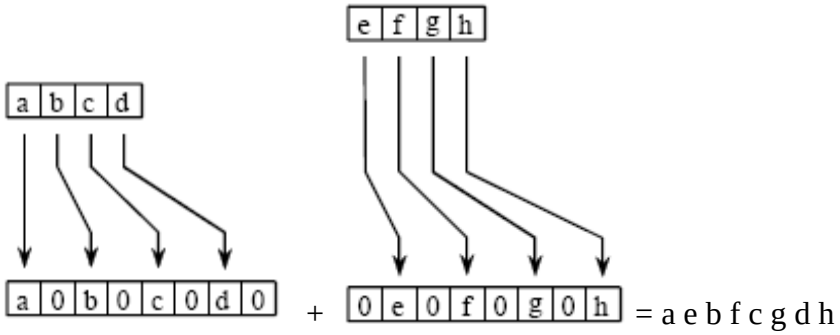
El modelo de reordenamiento de las muestras es así:

orden normal			orden reverso	
0	0000		0	0000
1	0001		8	1000
2	0010		4	0100
3	0011		12	1100
4	0100		2	0010
5	0101	bit reverso	10	1010
6	0110	⇨	6	0110
7	0111		14	1110
8	1000		1	0001
9	1001		9	1001

El próximo paso en el algoritmo de la *FFT* es encontrar el espectro de frecuencia de un punto de una señal del dominio del tiempo, el cual ya esta hecho porque el espectro de una señal de 1 punto es igual a este, cada señal de 1 punto es ahora un espectro de frecuencia.

El último paso en la *FFT* es combinar los *N* espectros de frecuencia en el exacto orden reverso que toma la descomposición en el dominio del tiempo.

Por ejemplo:
Si se consideran 2 señales en el dominio del tiempo [a b c d] y [e f g h]. Combinando cada señal con ceros *0* para hacer una señal de 8 puntos y luego sumando ambas señales obtenemos una señal de 8 puntos así:



Vemos que ambas señales son combinadas con ceros, en una señal los ceros están en las muestras pares y en la otra en las muestras impares.
Una señal es retrasada por una muestra, este retraso del dominio del tiempo equivale a multiplicar el espectro por una sinusoide.

Diagrama Flujo: modelo para combinar señales dominio frecuencia

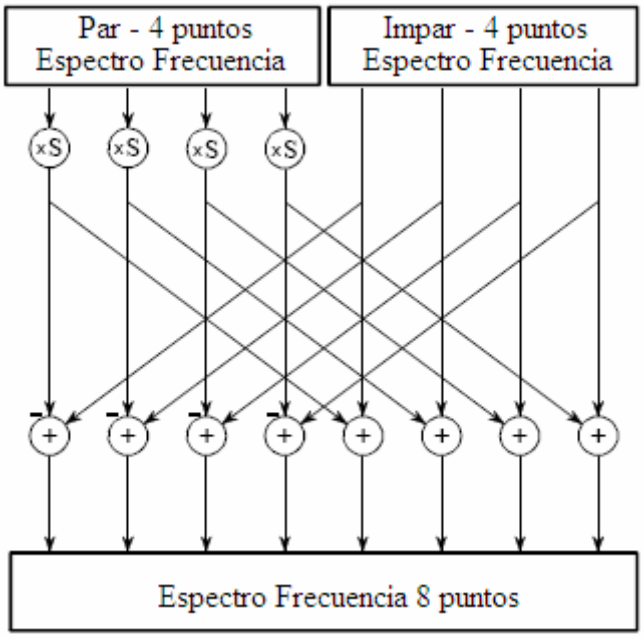
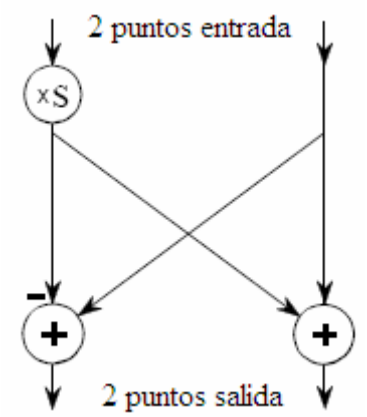


Diagrama Flujo: butterfly

Transforma 2 puntos complejos en otros 2 puntos complejos



xS corresponde a multiplicar el espectro por una sinusoide

Analizando la DFT y la DFT inversa con MATLAB

Ejercicio 1.6.1

Muestrear y efectuar una DFT de 8 puntos sobre una señal continua que contiene componentes de 1-khz y 2-khz, expresada como:

$$x(t) = \sin(2\pi * 1000 * t) + 0.5\sin(2\pi * 2000 * t + {}^{3\pi}\!/_{4})$$

El periodo de muestreo es *ts*

$$ts = {}^{1}\!/_{fs}$$

Se necesita una entrada de 8 puntos *N=8*, para efectuar la *DFT*

$$n \bullet ts = n\frac{1}{ts}$$

$$x[n] = \sin(2\pi * 1000 * n \cdot ts) + 0.5\sin(2\pi * 2000 * n \cdot ts + {}^{3\pi}\!/_{4})$$

Escogiendo una frecuencia de muestreo *fs=8-khz*, la DFT resulta indicada porque contendrá las amplitudes de la señal de entrada $x[n]$ en el análisis de la frecuencia

$$f_{analisis}(m) = {}^{m \cdot f_s}\!/_{N}$$

$$m \rightarrow 0 : N - 1$$

0-khz,1-khz,2-khz,…,7-khz

Nota: la función *dft_real* que se utiliza en el ejercicio esta escrita en C para más información léase la ayuda que viene en MATLAB para poder compilar y hacer funciones *mex*.

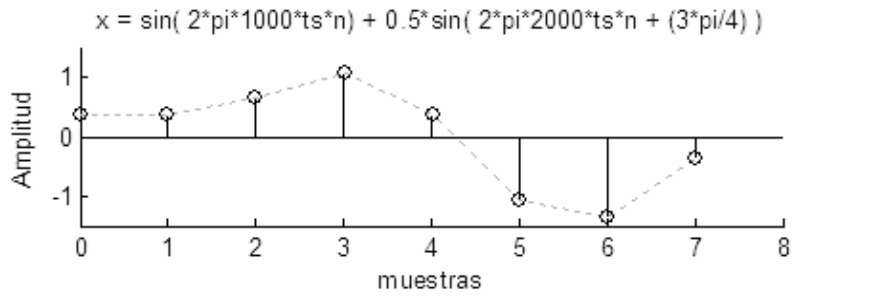
Código en C para MATLAB:

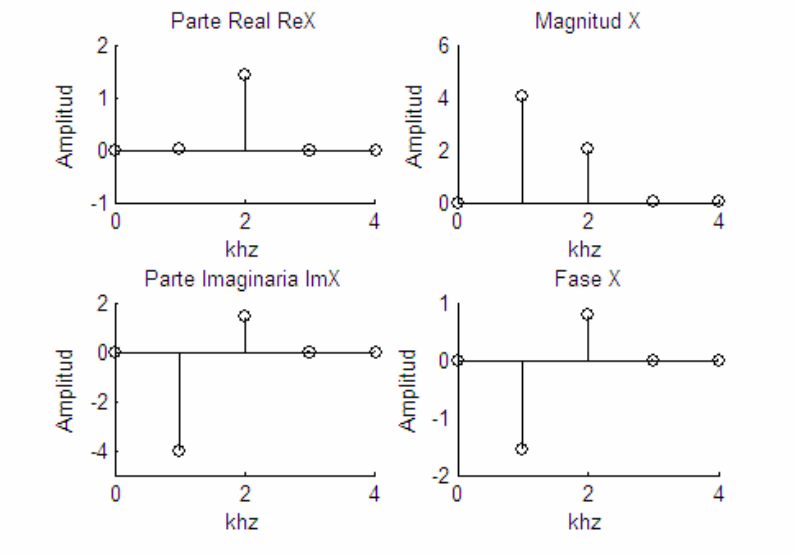
```
#include "math.h"
#include "mex.h"
#include "stdio.h"
#include "conio.h"

/*
 * Funcion en C
 */
#define pi 3.14159265

void dft_real(double x[],int n,double ReX[],double ImX[])
{/*dft*/
    int k,i;
    //limpio salida
    for(k = 0; k < n/2 + 1; k++)
    {
        ReX[k] = 0;
        ImX[k] = 0;
    }

    //correlaciona x con las ondas seno y coseno
    for(k = 0; k < n/2 + 1; k++)
    {
        for(i = 0 ;i < n; i++)
        {
            ReX[k]=ReX[k] + x[i]*cos(2*pi*k*i/n);
            ImX[k]=ImX[k] - x[i]*sin(2*pi*k*i/n);
        }
    }


}/*dft*/

/*
 *  GATEWAY = main()
 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{/*main*/
    double *x, *ReX, *ImX;
    int n;

    /*determino numero elementos matrices*/
    n = mxGetN(prhs[0]);

    /*salidas*/
    plhs[0]=mxCreateDoubleMatrix(1,n/2 + 1,mxREAL);
    plhs[1]=mxCreateDoubleMatrix(1,n/2 + 1,mxREAL);

    /*asigno punteros a las entradas y salidas*/
    x = mxGetPr(prhs[0]);
    ReX = mxGetPr(plhs[0]);
    ImX = mxGetPr(plhs[1]);

    dft_real(x,n,ReX,ImX);

}/*main*/
```

Script MATLAB:

```
%DFT de 8 puntos
%Ejemplo 1 (pag. 54 Understanding Digital Signal Processing
%Nota utiliza la función dft_real escrita en C para hacer los cálculos
```

```
% fs=8-khz
%definiendo muestras
n=0:7;
ts=(1/8000);
%definiendo x[i]
x = sin( 2*pi*1000*ts*n) + 0.5*sin( 2*pi*2000*ts*n + (3*pi/4) )
%graficar x[i]
subplot(3,2,1);
hold on
plot(n,x,'g:');
stem(n,x,'k');
axis([0 8 -1.5 1.5]);
title 'x = sin( 2*pi*1000*ts*n) + 0.5*sin( 2*pi*2000*ts*n + (3*pi/4) )'
ylabel('Amplitud');
xlabel('muestras');
hold off
%llamo función de C dft_real
[ReX,ImX]=dft_real(x)
%graficar ReX
nr=0:4;
subplot(3,2,3);
stem(nr,ReX,'k');
title 'Parte Real ReX'
ylabel('Amplitud');
xlabel('khz');
%graficar ImX
nr=0:4;
subplot(3,2,5);
stem(nr,ImX,'k');
title 'Parte Imaginaria ImX'
ylabel('Amplitud');
xlabel('khz');
axis([0 4 -5 2]);
%graficar magnitud dft
nr=0:4;
subplot(3,2,4);
mag=sqrt(ReX.^2+ImX.^2)
stem(nr,mag,'k');
title 'Magnitud X'
ylabel('Amplitud');
xlabel('khz');
%graficar fase dft
nr=0:4;
subplot(3,2,6);
%ojo en divisiones para 0 da error ,revisar el command window y
%arreglar manualmente o hacer un script que solucione el problema
phase = atan(ImX./ReX)
%revisando el command window vemos que da error en phase([4:5])
phase(4)=0;
phase(5)=0;
stem(nr, phase, 'k');
title 'Fase X'
ylabel('Amplitud');
xlabel('khz');
```





Ejercicio 1.6.2

Hallar la *DFT inversa* del ejercicio anterior.
Utilizar las respuestas de *ReX* e *ImX* del ejercicio anterior, la respuesta debe ser igual a x (señal entrada).

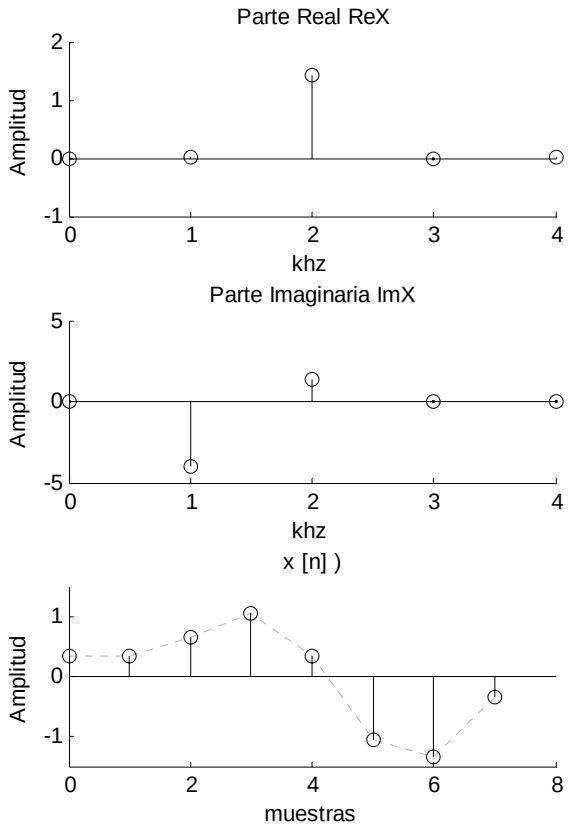
Código en C para MATLAB:

```
#include "math.h"
#include "mex.h"
#include "stdio.h"
#include "conio.h"

/*
 * Funcion en C
 */
#define pi 3.14159265

void inv_dft_real(double ReX[],double ImX[],int n ,double x[] )
{/*inv dft*/
    int k, i, N=n*2-2;
    //encuentra amplitudes ondas seno y coseno

    for(k = 0; k < n ; k++)
    {
        ReX[k] = ReX[k] / (N/2);
        ImX[k] = -ImX[k] / (N/2);
    }
     ReX[0] = ReX[0] / 2;
     ReX[n] = ReX[n] / 2;

     x[0]= ReX[0];
     x[1]= ReX[n] ;

     //limpia matriz salida
     for(k = 0; k < N + 1; k++)
    {
        x[k] = 0;
    }
    //síntesis
    for(k = 0; k < n ; k++)
    {
        for(i = 0 ;i <= N; i++)
        {
            x[i]=x[i] + ReX[k]*cos(2*pi*k*i/N);
            x[i]=x[i] + ImX[k]*sin(2*pi*k*i/N);
```

```
        }
    }

}/*inv dft*/

/*
 *  GATEWAY = main()
 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{/*main*/
    double *x, *ReX, *ImX;
    int n;

    /*determino numero elementos matrices*/
    n = mxGetN(prhs[0]);

    /*salidas*/
    plhs[0]=mxCreateDoubleMatrix(1,n*2-2 ,mxREAL);


    /*asigno punteros a las entradas y salidas*/
    x = mxGetPr(plhs[0]);
    ReX = mxGetPr(prhs[0]);
    ImX = mxGetPr(prhs[1]);

    inv_dft_real(ReX,ImX,n,x);

}/*main*/
```

Script MATLAB:

```
%DFT inversa del ejercicio anterior
n=0:7;
ts=(1/8000);
%ReX e ImX definidas en el ejercicio anterior revisar command window

%graficar ReX
nr=0:4;
subplot(3,1,1);
stem(nr,ReX,'k');
title 'Parte Real ReX'
ylabel('Amplitud');
xlabel('khz');

%graficar ReX
nr=0:4;
subplot(3,1,2);
stem(nr,ImX,'k');
title 'Parte Imaginaria ImX'
ylabel('Amplitud');
xlabel('khz');

%graficar x
xx = inv_dft_real(ReX,ImX);
subplot(3,1,3);
hold on
plot(n,xx,'g:');
stem(n,xx,'k');
axis([0 8 -1.5 1.5]);
title 'x [n] )'
ylabel('Amplitud');
xlabel('muestras');
hold off;
```



1.7 Transformada Laplace

La transformada de Laplace es una técnica muy usada para transformar ecuaciones diferenciales a ecuaciones algebraicas, que pueden ser más fáciles de manipular para obtener el resultado deseado. La transformada de Laplace cambia una señal del dominio del tiempo al dominio s, llamado también *plano-s*.

La *transformada de Laplace* de una función del tiempo $x(t)$ es $X(s)$ o $L\{x(t)\}$

$$X(s) = L\{x(t)\} = \int_{-\infty}^{\infty} (x(t) \cdot e^{-st}) dt$$

La variable compleja *s* es usualmente referida como una frecuencia compleja, con la forma $s = \sigma \pm j\omega$. Donde σ y ω son variables reales, referidas como frecuencia *Neper* y frecuencia radial respectivamente.

La *transformada inversa de Laplace* esta definida por:

$$x(t) = L^{-1}\{X(s)\} = \frac{1}{2\pi \cdot j}\int_C (X(s)\cdot e^{st})ds$$

C es el contorno de integración escogido

En la mayoría de los casos prácticos la evaluación directa de Laplace puede ser evitada usando algunos pares de transformadas. A continuación se muestra una tabla con algunas pares de transformadas de Laplace.

Pares de transformadas de Laplace

x(t)	**X(s)**
1	$\frac{1}{s}$
$u(t)$	$\frac{1}{s}$
$\delta(t)$	1
t	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^n+1}$
$\sin(\omega t)$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
e^{-at}	$\frac{1}{s+a}$
$e^{-at}\cdot\sin(\omega t)$	$\frac{\omega}{(s+a)^2+\omega^2}$
$e^{-at}\cdot\cos(\omega t)$	$\frac{s+a}{(s+a)^2+\omega^2}$

Por ejemplo:

Encontrar la transformada de Laplace de $x(t) = e^{-at}$

$$X(s) = \int_0^\infty e^{-at}e^{-st}dt$$

$$X(s) = \int_0^\infty e^{-(a+s)t}dt = \frac{1}{s+a}$$

Propiedades de la transformada de Laplace

Algunas propiedades de la transformada de Laplace son mostradas en la tabla de abajo, para obtener más transformadas de Laplace se utiliza conjuntamente con las pares de transformadas de Laplace, expuestas anteriormente.

Propiedades de la transformada de Laplace

Propiedad	Función del tiempo	Transformada
Homogeneidad	$a \cdot f(t)$	$a \cdot F(s)$
Aditividad	$f(t)+g(t)$	$F(s)+G(s)$
Linealidad	$a \cdot f(t)+b \cdot g(t)$	$a \cdot F(s)+b \cdot G(s)$
Primera derivada	$\frac{d}{dt} f(t)$	$s \cdot F(s)-f(0)$
Segunda derivada	$\frac{d^2}{dt^2} f(t)$	$s \cdot F(s)-s \cdot f(0)-\frac{d}{dt} f(0)$
Kth derivada	$\frac{d^k}{dt^k} f(t)$	$s^k \cdot F(s)=\sum_{n=0}^{k-1} s^{(k-1-n)} \cdot f^n(0)$
Integración	$\int_0^t f(t)dt$	$\frac{F(s)}{s}$
Desplazamiento Frecuencia	$e^{-at} \cdot f(t)$	$X(s+a)$
Desplazamiento Tiempo	$u_1(t-\mathrm{T}) f(t-T)$	$e^{-aT} \cdot F(s)$
Convolución	$f(t)=\int_0^t h(t-T)x(T)dT$	$Y(s)=H(s) \cdot X(s)$
Multiplicación	$f(t)*g(t)$	$\frac{1}{2\pi \cdot j} \int_{c-j\infty}^{c+j\infty} F(s-r)G(r)dr$

Analizando transformada de Laplace con MATLAB

Ejercicio 1.7.1

Matlab posee un comando llamado laplace(f), que computa la transformada de Laplace de una función, utilizar el comando para encontrar la transformada de Laplace de $x(t)=e^{-at}$

Script MATLAB:

```
%Encuentra la transformada de una función
syms a t ;
x = exp(-a*t);
X=laplace(x)
```

Si observamos en el command window de Matlab tendremos algo como esto

```
X =

1/(s+a)
```

Función de Transferencia

La función de transferencia $H(s)$ de un sistema es igual a la transformada de Laplace de la señal de salida dividida para la transformada de Laplace de la señal de entrada.

$$H(s) = \frac{Y(s)}{X(s)} = \frac{L[y(t)]}{L[x(t)]}$$

La función de transferencia también es igual a la transformada de Laplace de la respuesta al impulso de un sistema:

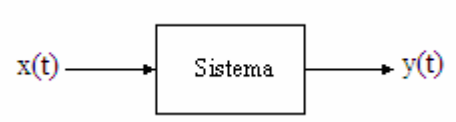
$$H(s) = L[h(t)]$$

La función de transferencia también puede ser puesta de esta forma:

$$H(s) = \frac{P(s)}{Q(s)}$$

Donde $P(s)$ y $Q(S)$ son los polinomios en *s*.

Por Ejemplo:
Un sistema lineal es mostrado en la figura de abajo:



Si relacionamos la entrada $x(t)$ y la salida $y(t)$ del sistema LTI de la figura de arriba, con una ecuación diferencial homogénea de coeficientes constantes

$$a_2 \frac{d^2y(t)}{dt^2} + a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_1 \frac{dx(t)}{dt} + b_0 x(t)$$

a_n y b_n son los coeficientes constantes, los cuales pueden ser positivos o negativos, cero, real o complejo.
Gracias a *Laplace*, la función exponencial compleja e^{st} es muy usada, esta función tiene la maravillosa propiedad de poder ser diferenciada cualquier número de veces sin destruir su forma original, que es:

$$\frac{d(e^{st})}{dt} = se^{st}, \frac{d^2(e^{st})}{dt^2} = s^2e^{st}, \ldots, \frac{d^n(e^{st})}{dt^n} = s^ne^{st}$$

Si dejamos que $x(t)$ y $y(t)$ sean funciones de e^{st}, $x(e^{st})$ y $y(e^{st})$ y usamos la propiedad descrita arriba tendremos

$$a_2s^2y(e^{st}) + a_1s^1y(e^{st}) + a_0s^0y(e^{st}) = b_1\ sx(e^{st}) + b_0x(e^{st})$$

O

$$(a_2s^2 + a_1s + a_0)y(e^{st}) = (b_1\ s + b_0)x(e^{st})$$

Aunque es más simple la ecuación, se puede simplificar más considerando la relación de $y(e^{st})$ sobre $x(e^{st})$ como la función de transferencia, la relación polinómica de la función de transferencia $H(s)$ quedaría así:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_1s + b_0}{a_2S^2 + a_1S + a_0}$$

La función $H(s)$ nos permite determinar la estabilidad, y la respuesta a la frecuencia de un sistema continuo.
Donde la salida $Y(s)$ esta dada por:

$$Y(s) = X(s)\frac{b_1s + b_0}{a_2S^2 + a_1S + a_0} = X(s)H(s)$$

Polos y Ceros sobre el plano-s

Como se vio previamente, la función de transferencia de un sistema LTI puede ser expresada con la relación polinómica en *s*

$$H(s) = \frac{P(s)}{Q(s)}$$

El numerador y denominador pueden ser factorizados a

$$H(s) = H_0\frac{(s - z_1)(s - z_2)...(s - z_n)}{(s - p_1)(s - p_2)...(s - p_n)}$$

Donde la raíces $z_1, z_2, \ldots, z_n$ del numerador son llamadas *ceros* (*zeros*) de la función de transferencia y las raíces $p_1, p_2, \ldots, p_n$ del denominador son llamadas *polos* de la función de transferencia. Los *polos* y *ceros* pueden ser colectivamente referidos

como *frecuencias críticas*. Cada factor $(s - z_i)$ es llamado factor cero y cada factor $(s - p_j)$ es llamado factor polo.

Un repetido *cero* apareciendo *n* veces es llamado *cero nth-orden*, así también un repetido polo apareciendo *n* veces es llamado *polo nth-orden*.

Por ejemplo:
Considere la función de transferencia:

$$H(s) = \frac{s^3 + 5s^2 + 8s + 4}{s^3 + 13s^2 + 59s + 87}$$

El numerador y el denominador pueden ser factorizados a

$$H(s) = \frac{(s+2)^2(s+1)}{(s+5+2j)(s+5-2j)(s+3)}$$

Examinando los polos y ceros tenemos:

Numerador	**Denominador**
s=-1 es un simple cero	*s=-5+2j es un simple polo*
s=-2 es un cero de segundo-orden	*s=-5-2j es un simple polo*
	s=-3 es un simple polo

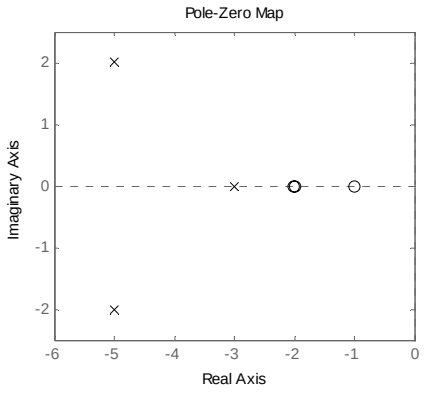
Los *polos* y *ceros* pueden ser representados gráficamente como localizaciones en un plano complejo como el de la figura mostrada a continuación, donde *0=zero* y *x=polo*.

Los *polos* pueden proporcionar indicaciones del comportamiento del sistema como se muestra en la tabla de abajo, además los *polos* y *ceros* poseen las siguientes propiedades que pueden ser usadas para ***apresurar*** el análisis de un sistema.

1.- Para la *H(s)* real, compleja o imaginaria, los polos y ceros ocurrirán en pares complejos conjugados que son simétricos sobre el eje σ.

2.- Para *H(s)* teniendo incluso simetría, los polos y ceros exhibirán simetría sobre $j\omega$.

2.- Para *H(s)* no negativa, cualquier cero sobre el eje $j\omega$, ocurrirá en pares



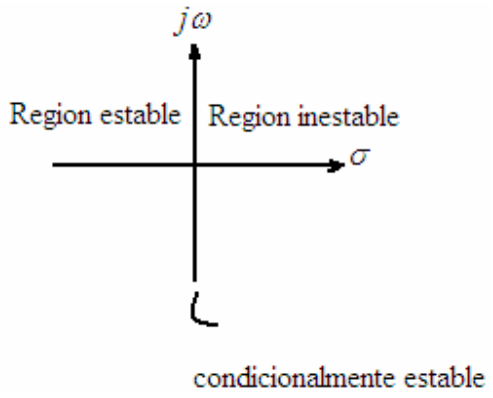
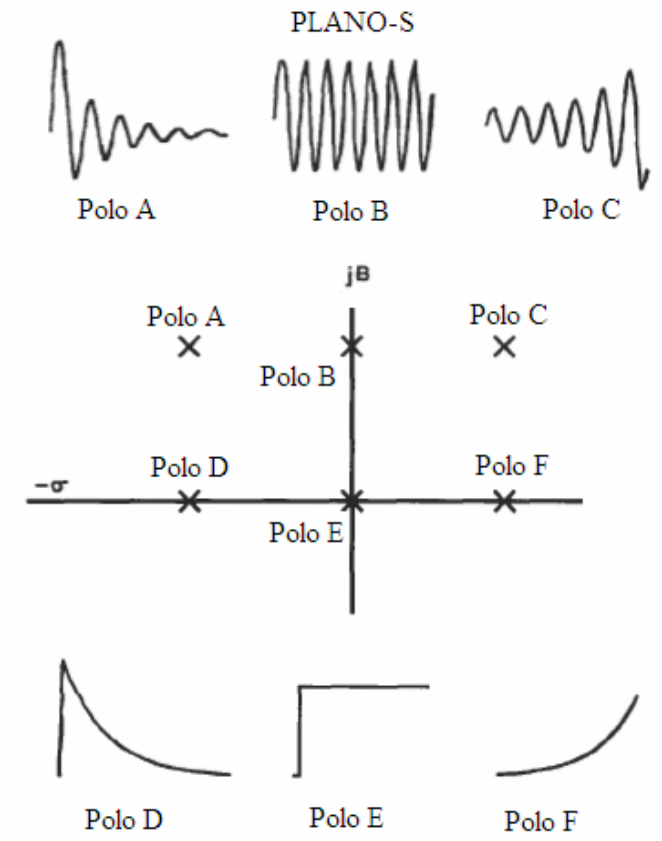


Tabla de impacto de la ubicación del polo en el comportamiento del sistema

Tipo de polo	Correspondencia natural Respuesta componente	Comportamiento Sistema
Simple, negativo	*Exponencial decayendo*	*Estable*
simple, positivo	*Exponencial divergente*	*Inestabilidad divergente*
Par real, negativo , no igual	*Exponencial decayendo*	*Overdamped(estable)*
Par real, negativo, igual	*Exponencial decayendo*	*Critically damped(estable)*
Par complejo conjugado, con partes reales negativas	*Decayendo sinusoide exponencialmente*	*Underdamped(estable)*
Par complejo conjugado, con partes reales cero	*sinusoide*	*Undamped(marginalmente estable)*
Par complejo conjugado, con partes reales positivas	*Sinusoide saturándose exponencialmente*	*Oscilatorio (inestabilidad)*



Analizando polos y ceros con MATLAB

Ejercicio 1.7.2

Matlab posee el comando *roots(a)* que encuentra las raíces de un polinomio, se puede utilizar para determinar los *polos* y *ceros* de una función de transferencia.

- Encontrar los polos y ceros de la siguiente transformada de Laplace:

$$X(s) = \frac{4s^2 + 6}{s^3 + s^2 - 2}$$

Script MATLAB:

```
%encuentra raíces de polos y ceros
n =[0 4 0 6];
d =[1 1 0 -2];
%ceros
z = roots(n)
%polos
p = roots(d)
```

En el command window de Matlab se puede observar el resultado

```
z =
        0 + 1.2247i
        0 - 1.2247i
p =
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
   1.0000
```

Tenemos un par de ceros complejos conjugados, un simple polo y un par de polos complejos conjugados.

- El comando *poly( )* usa los polos o ceros para hallar los coeficientes del polinomio.

Usando las respuestas anteriores para calcular los coeficientes se puede realizar así:

Script MATLAB:

```
n=poly(z)
d=poly(p)
```

Ejercicio 1.7.3

Considere la función de transferencia:

$$H(s) = \frac{s^3 + 5s^2 + 8s + 4}{s^3 + 13s^2 + 59s + 87}$$

Obtener y graficar los polos y ceros

Script MATLAB:

```
%graficando polos y ceros en el plano-s
clc
syms s
n=(s^3 + 5*s^2 + 8*s+4);
d=(s^3 + 13*s^2 + 59*s+87);
H=tf([1 5 8 4],[1 13 59 87])
%Computa plano-s con polos y ceros
pzmap(H);
%Calcula polos del sistema
pole( H )
%calcula zeros del sistema
zero(H)
```

Observando el command window tendremos:

```
Transfer function:
 s^3 + 5 s^2 + 8 s + 4
-----------------------------
s^3 + 13 s^2 + 59 s + 87

ans =
  -5.0000 + 2.0000i
  -5.0000 - 2.0000i
  -3.0000

ans =
   -2.0000
   -2.0000
   -1.0000
```

1.8 Transformada- z

La *transformada – z* juega el mismo rol en el análisis de señales discretas y sistemas LTI como la transformada de Laplace lo hace en el análisis de señales continuas en el tiempo y sistemas LTI. La transformada-z es una generalización de la *DTFT*, la transformada-z facilita el análisis de las ecuaciones diferenciales discretas y esta definida como

$$Z\{x[n]\} = X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n} \quad \textit{transformada-z directa}$$

$$z = r \cdot e^{jw}$$

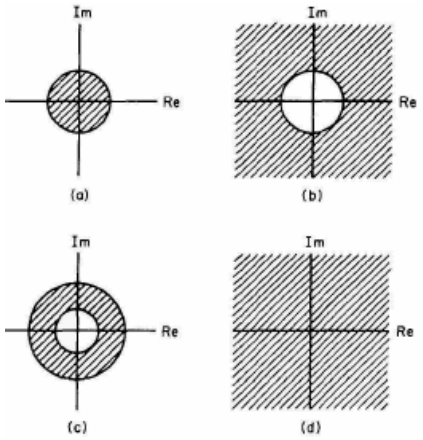
Donde *z* es la variable compleja, el conjunto de valores de *z* para los cuales *X*(*z*) existe se llama región de convergencia (*ROC*).
La *transformada-z inversa* de una función compleja *X*(*z*) , esta dada por

$$x[n] = Z^{-1}[X(z)] = \frac{1}{2\pi j} \oint_C X(z) z^{n-1} dz \quad \textit{transformada-z inversa}$$

Donde *C* denota el contorno encerrado en *ROC* de *X*(*z*) , tomado en sentido contrario a las agujas del reloj.

ROC

La región del plano-z para el cual una serie converge se llama región de convergencia (*ROC*). Para una secuencia dada *x*[*n*] si la serie $X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n}$ converge para un valor de $z = z_1$ entonces la serie convergirá para todos los valores de *z* para los cuales $|z| = |z_1|$. Recíprocamente, si la serie diverge para $z = z_2$, entonces la serie divergirá para todos los valores de *z* para los cuales $|z| = |z_2|$.
Porque la convergencia depende de la magnitud de *z*, la región de convergencia casi siempre ira encerrada por un circulo centrado en el origen del *plano-z*. Esto no quiere decir que la región de convergencia ira siempre con un círculo, a continuación se muestra un gráfico con algunas configuraciones de *ROC*.



Configuraciones de convergencia

Pares de la transformada-z

A continuación se muestra una tabla con los pares de la transformada-z más comunes:

***x*[*n*]**	***X*(*z*)**
$A \cdot \delta[n]$	A
$A \cdot u[n]$	$\frac{A \cdot z}{(z-1)}$
$A \cdot n \cdot u[n]$	$\frac{A \cdot z}{(z-1)^2}$
$A \cdot a^n \cdot u[n]$	$\frac{A \cdot z}{(z-a)}$
$A \cdot \cos(\Omega \cdot n) \cdot u[n]$	$\frac{A \cdot z \cdot [z - \cos(\Omega)]}{z^2 - 2 \cdot \cos(\Omega) \cdot z + 1}$

$A \cdot \sin(\Omega \cdot n) \cdot u[n]$	$\frac{A \cdot z \cdot \sin(\Omega)}{z^2 - 2 \cdot \cos(\Omega) \cdot z + 1}$
$A \cdot a^n \cdot \cos(\Omega \cdot n + \Phi) \cdot u[n]$	$\frac{A \cdot z \cdot [z \cdot \cos(\Phi) - a \cdot \cos(\Phi - \Omega)]]}{z^2 - 2 \cdot a \cdot \cos(\Omega) \cdot z + a^2}$

Propiedades de la transformada-z

En la siguiente tabla se muestran algunas propiedades muy usadas de la transformada-z

Dominio del Tiempo	*Dominio de la Frecuencia*
$h[n]$	$H(z)$
$\sum_{k=-\infty}^{\infty} x_1[k] \cdot x_2[n-k]$	$X_1(z) \cdot X_2(z)$
$x[n-k]$	$z^{-k} \cdot X(z)$
$n \cdot x[n]$	$-z \frac{dF(z)}{dz}$

Analizando transformada-z con MATLAB

Ejercicio 1.8.1

Matlab posee los comandos *ztrans(f)* e *iztrans(F)* que calculan la transformada-z directa e inversa respectivamente de una función.

Calcular las transformadas de $x[n] = \cos(\Omega \cdot n)$

Script MATLAB:

```
clc
syms n omega;
x=cos(n*omega);
%calcular la transformada z de x[n]
X=ztrans(x)
pretty(X);
%calcular la transformada z  inversa de X(z)
x=iztrans(X)
pretty(x);
```

Ejercicio 1.8.2

Considere la función de transferencia

$$H(z) = \frac{0.094(1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + z^{-4})}{1 + 0.4860z^{-2} + 0.0177z^{-4}}$$

Calcular los polos y los ceros de la función de transferencia y dibujar el plano-z

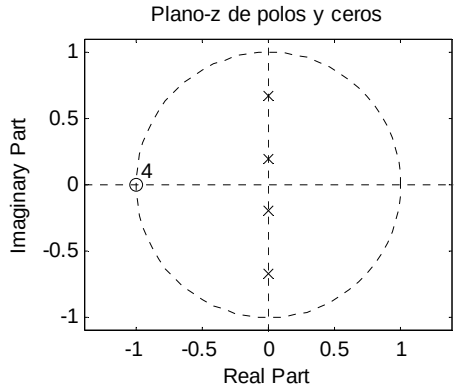
Script MATLAB:

```
clc %limpia command window
syms z;
n = 0.094*[1 4 6 4 1];
d= [1 0  0.4860 0 0.0177];
%función transferencia
H=tf(n,d,'variable','z^-1')
%calcula polos y ceros de la función transferencia
poles=pole(H)
zeros=zero(H)
%Gráfica plano z
zplane(n,d)
title 'Plano-z de polos y ceros'
```

Revisando el command window de Matlab tendremos:

```
Transfer function:
0.094 + 0.376 z^-1 + 0.564 z^-2 + 0.376 z^-3 + 0.094 z^-4
---------------------------------------------------------
              1 + 0.486 z^-2 + 0.0177 z^-4

poles =
        0 + 0.6681i
        0 - 0.6681i
        0 + 0.1991i
        0 - 0.1991i

zeros =
  -1.0002
  -1.0000 + 0.0002i
  -1.0000 - 0.0002i
  -0.9998
```

Se tiene un cero de orden 4 en z=-1, y 2 pares de polos conjugados



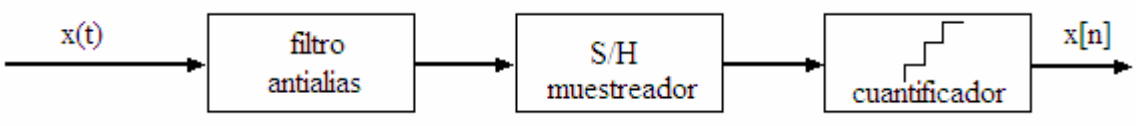
Introducción ADC-DAC

La conversión analógica - digital *ADC* y la conversión digital - analógica *DAC* son los procesos que permiten a las computadoras interactuar con las señales encontradas en la naturaleza, la ciencia y la ingeniería.

El proceso de convertir una señal analógica a una señal digital con tiempo discreto y amplitud discreta es efectuado en varios pasos:

Paso 1.- El espectro de frecuencia de la señal analógica debe ser limitado.
Paso 2.- La señal debe ser muestreada a una frecuencia apropiada.
Paso 3.- El valor muestreado debe ser cuantificado a un aceptable nivel de precisión.

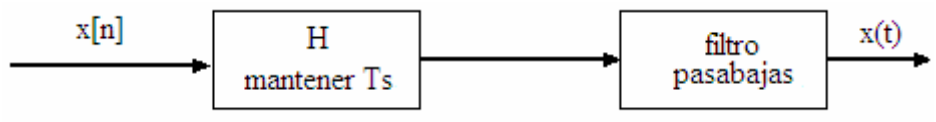
El gráfico de abajo muestra el proceso *ADC*



Cuando se desea convertir una señal digital de regreso a su forma analógica hay varios métodos, el método más simple requiere solo 2 pasos:

Paso 1.- El valor de salida de la señal digital debe mantenerse durante un periodo de muestreo.
Paso 2.- Pasar la señal a través de un filtro pasabajas.

El diagrama de abajo se muestra el proceso *DAC*



2.1 Muestreo y Cuantificación

Muestreo en el dominio del tiempo

Cuando una señal analógica *x*(*t*) es muestreada, las muestras son tomadas usualmente a intervalos de tiempo iguales, puede considerarse al muestreo como un interruptor (*switch*) que se abre y se cierra periódicamente cada T_s segundos.

$$T_s = 1/f_s \quad \textit{Periodo muestreo}$$

La forma de onda digitalizada resultante puede ser especificada así:

$$x_s(nT_s) = x(t)\big|_{t=nT_s}$$

Por Ejemplo:
Si tenemos una señal analógica *x*(*t*), la versión muestreada de la señal será $x_s(nT_s)$,así:

$$x(t) = 50 \cdot e^{-100 \cdot t} + \cos(200.t)$$

$$x(nT_s) = 50 \cdot e^{-100 \cdot nT_s} + \cos(200.nT_s)$$

Asumiendo que la señal es muestreada a una frecuencia de muestreo de 1-khz se pueden calcular los valores de $x_s(nT_s)$

$$T_s = 1/f_s = 1/1000 = 0.001\_seg$$

Los valores pueden guardarse como una secuencia de números, una vez realizado el muestreo la señal queda digitalizada, pudiendo ser representada ahora así:

$$x[n] = 50 \cdot e^{-100 \cdot n} + \cos(200.n)$$

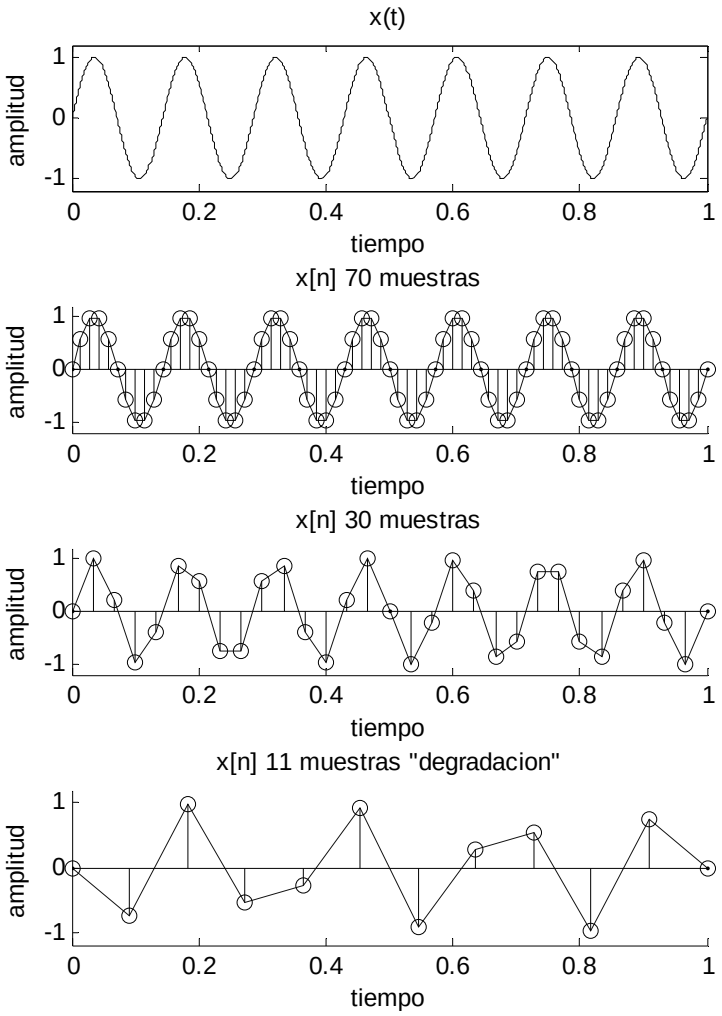
Analizando muestreo con MATLAB

Ejercicio 2.1.1
Utilizaremos una señal de entrada definida como $x(t) = \sin(2\pi \cdot f)$ para analizar con diferentes frecuencias de muestreo.

$$T_s = 1/f_s \qquad T_s = n \cdot T_s$$

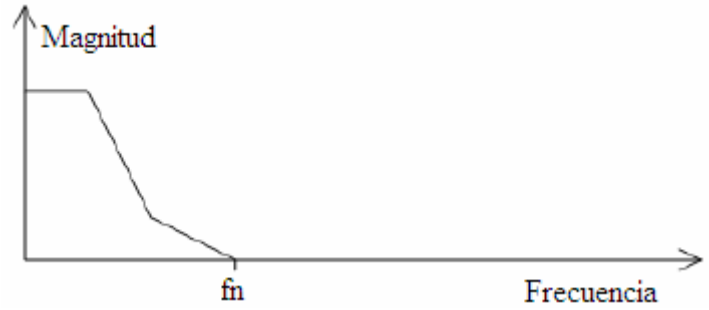
Script para MATLAB:

```
%Muestreo
%fs frecuencia muestreo probar con varias frecuencias
%fs 1-khz
fs = 1000;
Ts = 1 / fs;
nTs=0:Ts:1;
%señal de entrada x(t)
x=sin(2*pi*7*nTs);
%gráfico la señal de entrada
subplot(4,1,1);
plot(nTs,x,'k')
title('x(t)');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])

%************************
%analizando con otra fs
fs = 70;
Ts = 1 / fs;
nTs=0:Ts:1;
%señal muestreada x[n]
x=sin(2*pi*7*nTs);
subplot(4,1,2);
hold on
plot(nTs,x,'b')
stem(nTs,x,'k')
title('x[n] 70 muestras');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%************************
%analizando con otra fs
fs = 30;
Ts = 1 / fs;
nTs=0:Ts:1;
%señal muestreada x[n]
x=sin(2*pi*7*nTs);
subplot(4,1,3);
hold on
plot(nTs,x,'b')
stem(nTs,x,'k')
title('x[n] 30 muestras');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
%************************
%analizando con otra fs
fs = 11;
Ts = 1 / fs;
nTs=0:Ts:1;
%señal muestreada x[n]
x=sin(2*pi*7*nTs);
subplot(4,1,4);
hold on
plot(nTs,x,'b')
stem(nTs,x,'k')
title('x[n] 11 muestras "degradación"');
ylabel('amplitud');
xlabel('tiempo');
axis([0 1 -1.2 1.2])
```

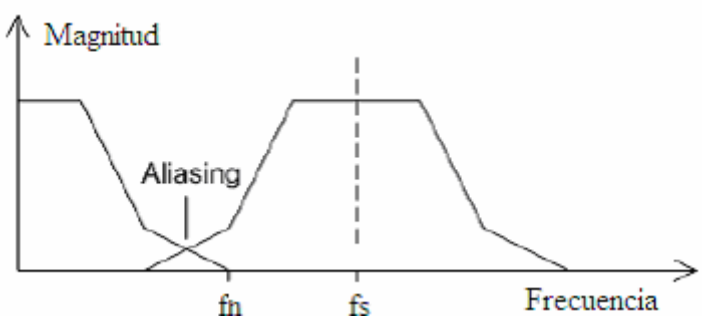


Muestreo en el dominio de la frecuencia

El espectro de frecuencias de una señal antes de muestrear se muestra en la figura de abajo



Si muestreamos esta señal a una frecuencia *fs* el espectro de la señal muestreada será como el de la siguiente figura



El espectro de la señal original (*antes de muestrear*) es replicado a través del espectro a intervalos *fs,* de esta replicación es posible que haya corrupción de las componentes de frecuencia de la señal original con las componentes de la señal replicada, a esa corrupción se le llama ***aliasing***.

Cuando una señal es digitalizada es muy importante capturar toda la información posible, presente en la señal analógica original sin generar *aliasing*.
La frecuencia de muestreo debe ser por lo menos el doble que la frecuencia más alta presente en la señal original, esta relación se le conoce como criterio de *Nyquist*, este criterio es un requisito para el muestreo.

$$f_s > 2 \cdot fn \quad \textit{Criterio Nyquist}$$

fn, frecuencia más alta señal entrada

Para asegurar que este requisito se cumple en todo momento, es necesario limitar el ancho de banda de la señal de entrada a la mitad de la *fs* (*frecuencia de muestreo*), esto se hace implementando un filtro antialiasing (*filtro analógico pasa bajas*).

Cuantificación de las muestras

Una vez que la señal analógica ha sido muestreada, esta es una señal discreta, los valores de la señal existen solamente en momentos particulares de tiempo, pero la amplitud de la señal sigue siendo continua. Entonces el próximo paso de la conversión analógica a digital *ADC* es cuantificar las amplitudes continuas de la señal a uno o varios valores de amplitud discretos.

El número de los posibles valores permitidos para la amplitud esta determinado por el tamaño de la variable escogida para guardar la información.

Por ejemplo:
Si escogemos un simple byte de memoria (*8 bits*) para guardar la información, entonces la amplitud puede tomar 2^8 o *256* valores diferentes.
Si la señal tiene una amplitud de *+1 Voltio* a *-1 Voltio*

$$\frac{2}{256} = 0.0078$$

Tenemos que cada amplitud puede estar separada por *0.0078 Voltios*, por otro lado si se escogiera 2 bytes de memoria para guardar cada muestra, la amplitud puede tomar *65.536* valores diferentes

$$\frac{2}{65536} = 0.000031$$

Cada valor de amplitud puede estar separado por *0.000031 Volt*s.

Mientras más larga sea la variable usada para guardar la muestra más podemos aproximar la representación digital a la señal analógica original.

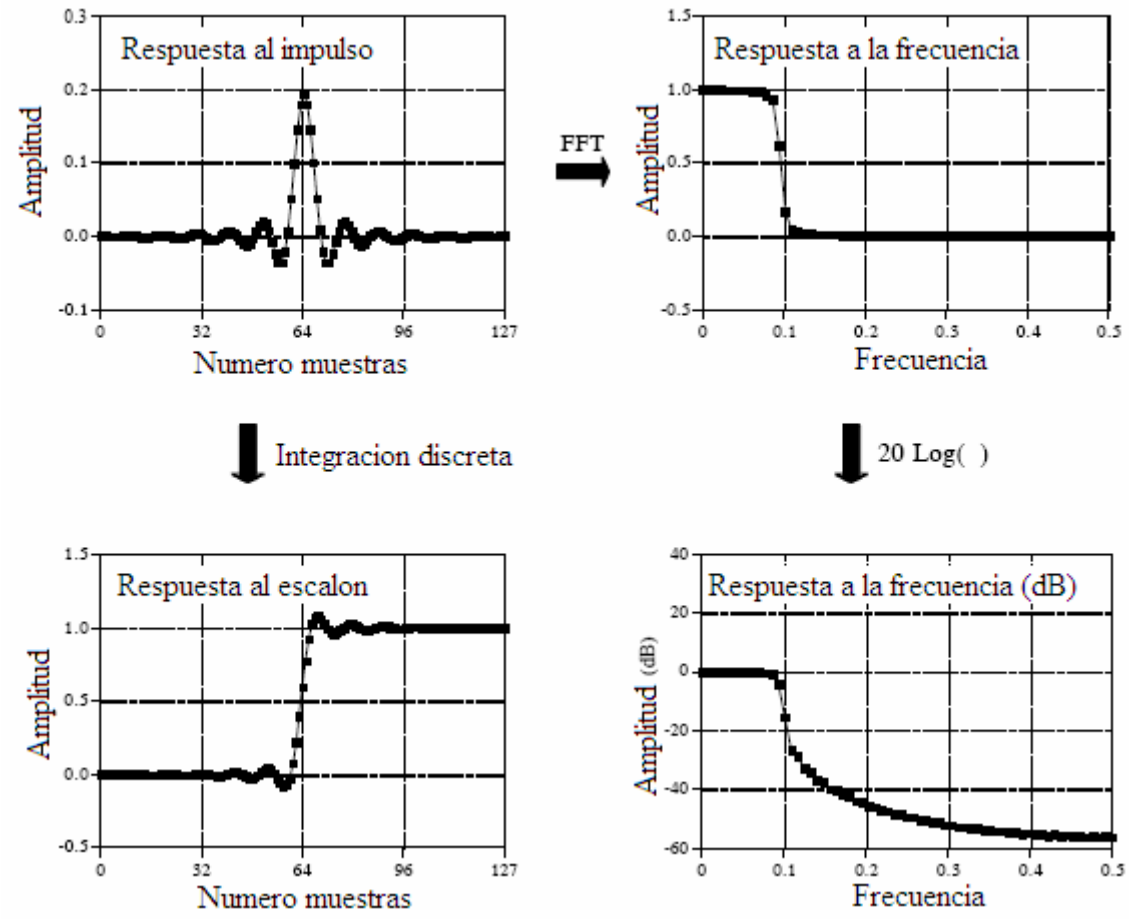
2.2 Introducción a los filtros digitales

Los filtros digitales son usados para: *separar* señales que se han combinado y para *restaurar* señales que se han distorsionado de alguna manera. Los filtros analógicos son usados para las mismas tareas pero los filtros digitales pueden lograr mejores resultados.

Ventajas	Desventajas
Fase lineal	Limitación de Velocidad
No cambian cualquiera que sea el entorno	Efectos de longitud finita de las palabras
Procesamiento de varias señales con un solo filtro	Tiempos de diseño y desarrollo
Posibilidad de almacenar datos	
Efectivo en aplicaciones de baja frecuencia	

El filtrado digital es una de las herramientas más poderosas del *DSP*, aparte de las ventajas obvias de eliminación virtual de errores en el filtrado asociado con los componentes pasivos que fluctúan sobre el tiempo y la temperatura. El filtrado digital es un concepto totalmente diferente del filtrado analógico.
Todo filtro lineal tiene una **respuesta al impulso**, una **respuesta al escalón**, y una **respuesta a la frecuencia**. Cada una de esas respuestas contiene información completa sobre el filtro, además describen como el filtro *reaccionará* a diferentes circunstancias, pero en diferente forma. Si una de las tres respuestas es especificada, las otras dos son estables y pueden ser calculadas directamente.
La respuesta al escalón puede ser calculada por integración discreta de la respuesta al impulso, la respuesta a la frecuencia puede ser calculada usando la *FFT* de la respuesta al impulso.
La respuesta a la frecuencia puede ser graficada en escala lineal y logarítmica, la escala lineal es mejor mostrando el *ripple* pasa banda y el *roll-off*, mientras que la escala logarítmica muestra la atenuación en la *stopband*.

En el siguiente gráfico se muestran las respuestas de un filtro lineal



La información puede ser representada de dos maneras: en el dominio del tiempo y en el dominio de la frecuencia. La respuesta al escalón describe como la información representada en el dominio del tiempo esta siendo modificada por el sistema, en contraste, la respuesta a la frecuencia muestra como la información en el dominio de la frecuencia esta cambiando.

Las distinciones de la respuesta al escalón y la respuesta a la frecuencia son críticas en el diseño de un filtro porque es *imposible* optimizar un filtro para ambas aplicaciones, un buen desempeño en el dominio del tiempo resulta en un pobre desempeño en el dominio de la frecuencia y viceversa.

Por ejemplo:
Si se esta diseñando un filtro para remover ruido de una señal *ECG*, la información esta representada en el dominio del tiempo, entonces la respuesta al impulso va a ser un parámetro importante y la respuesta a la frecuencia va a ser de poco interés.

Parámetros en el dominio del tiempo y la frecuencia

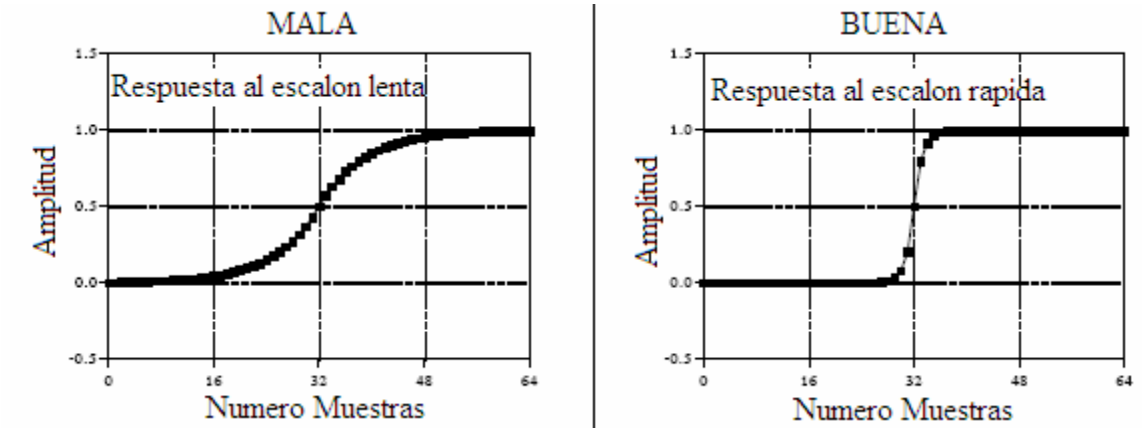
Parámetros en el dominio del tiempo

En el dominio del tiempo hay tres parámetros que son la clave para evaluar filtros en el dominio del tiempo.

Velocidad de transición (*rise time*)

La respuesta al escalón debe ser lo más corta posible para que haya una transición rápida, se usa para medir el desempeño de un filtro en el dominio del tiempo.

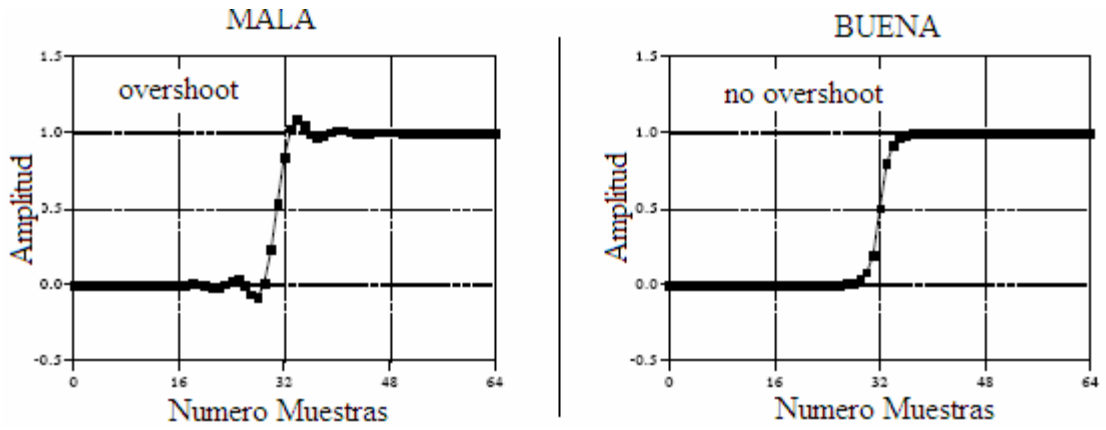
En la figura de abajo se muestra la velocidad de transición:



Overshoot

El overshoot en la respuesta al escalón debería eliminarse porque este cambia la amplitud de las muestras en la señal, esta es una distorsión básica de la información contenida en el dominio del tiempo, el overshoot varía según el filtro utilizado.

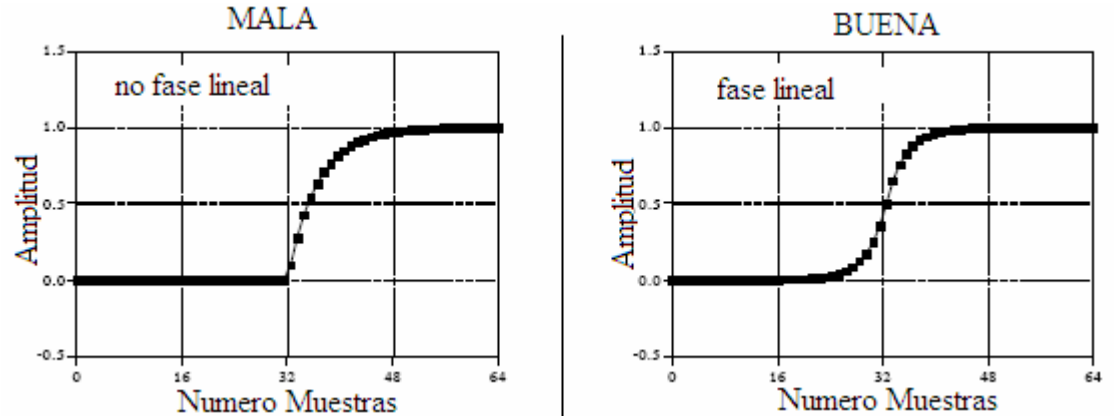
En la figura de abajo se muestra el overshoot:



Fase lineal

Se necesita simetría para hacer que los bordes de subida luzcan como los bordes de bajada, a esta simetría se le llama *fase lineal* (linear phase).
La fase lineal es el ***retraso constante*** que tiene un filtro.

En el gráfico de abajo se muestra la fase lineal que debe tener un filtro:

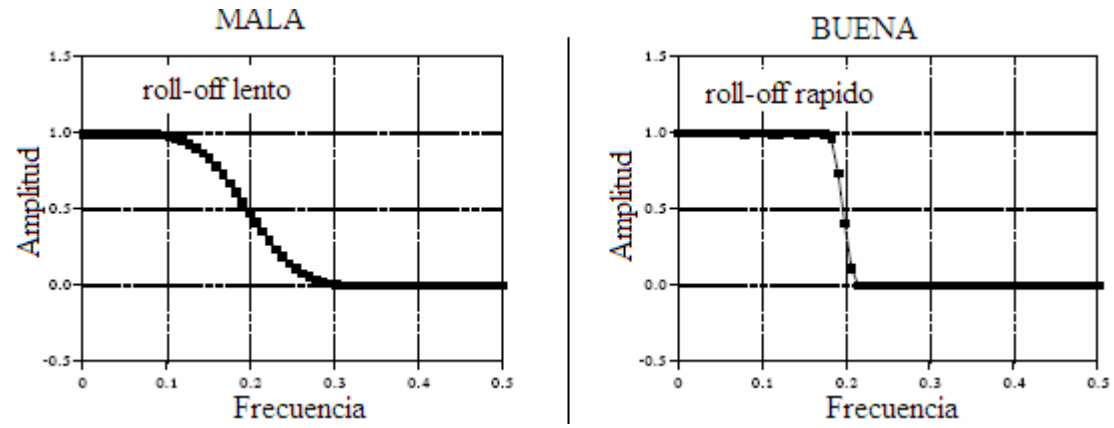


Parámetros en el dominio de la frecuencia

El propósito de los filtros es dejar pasar algunas frecuencias, mientras bloquea completamente otras frecuencias. **Pasa banda** (passband) banda de frecuencias que se permite pasar desde *DC* hasta f_c ,**Banda de atenuación** (stopband) contiene las frecuencias que son rechazadas, **Banda de transición** (transition band) esta entre las dos anteriores, la división entre la pasa banda y la banda de transición se llama **frecuencia de corte** f_c , en el diseño de filtros analógicos es usualmente definida la reducción de la amplitud a *0.707* (*-3 dB*) , en filtros digitales es menos estandarizado pero es común ver a un 99%, 90%, 70.7% *y* 50% del nivel de la amplitud definida que sea la frecuencia de corte.

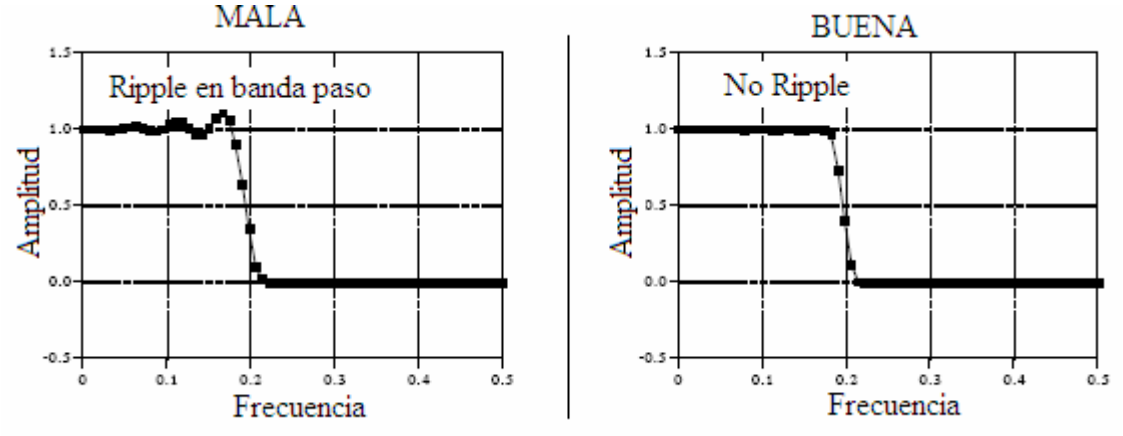
Roll – off

Un buen roll – off significa que la banda de transición es muy estrecha, en el gráfico de abajo se muestra el roll – off:



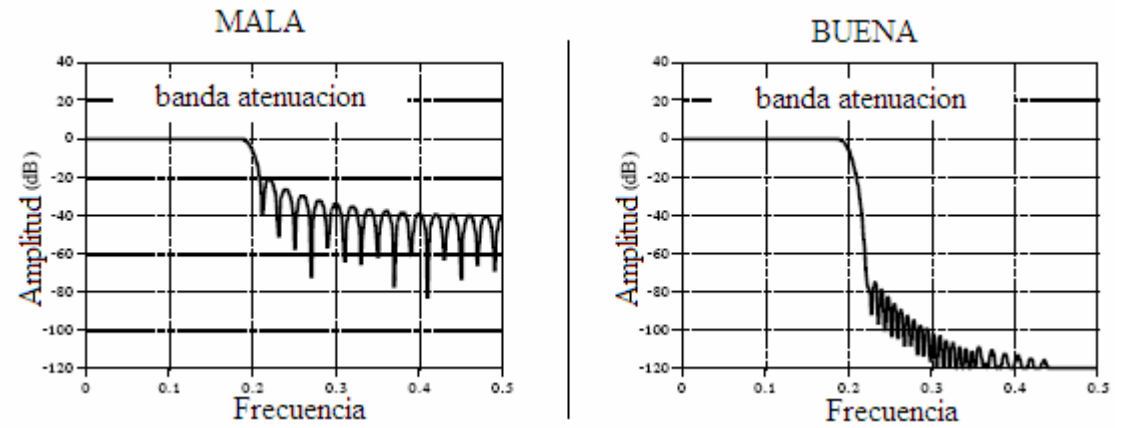
Ripple

Las frecuencias de la banda de paso oscilan, en el gráfico de abajo se muestra el ripple:



Banda Atenuación (*stopband attenuation*)

Banda de frecuencias que deben ser rechazadas, en el gráfico de abajo se muestra la atenuación:



Selectividad de los filtros

El propósito primario de los filtros es la de diferenciar entre las diferentes bandas de frecuencia y la selección de frecuencia que es el método mas común de clasificar a los filtros. Nombres como pasa-bajas, pasa-altas, pasa-banda, y elimina-banda son usados para categorizar filtros, además de ser nombres, describen completamente un filtro.

Hay dos conjuntos de especificaciones para definir completamente la respuesta de un filtro.

Especificaciones de frecuencia

Usadas para describir la banda de paso y la banda de atenuación, pueden estar en *Hz* o en *rad/seg*.

Especificaciones de característica de ganancia

Especificaciones de las características de ganancia de la pasa-banda y la elimina-banda de la respuesta de un filtro. La ganancia de un filtro simplemente es la relación del nivel de la señal de salida para el nivel de la señal de entrada. Si la ganancia es mas grande que 1, entonces la señal de salida es mas larga que la señal de entrada, mientras si la ganancia es menor a 1, la salida es menor a la entrada. En las aplicaciones de filtrado, la respuesta a la ganancia en la banda de atenuación es muy pequeña, por esta razón se convierte a decibeles (*dB*).
Por ejemplo:
Un filtro tiene una ganancia de *0.707* en la banda de paso y una ganancia de *0.0001* en la banda de atenuación. También podemos especificar en *dB* las ganancias convirtiéndolas así:

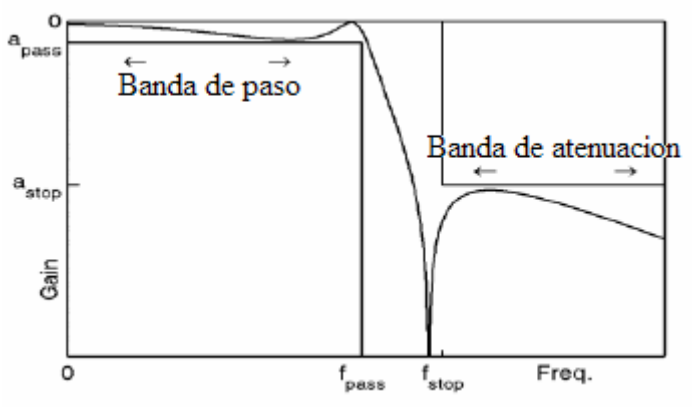
$$ganancia_{dB} = 20 \cdot \log(ganancia)$$

$$ganancia_{dB} = 20 \cdot \log(0.707) = -3dB \quad Banda\ de\ paso$$

$$ganancia_{dB} = 20 \cdot \log(0.0001) = -80dB \quad Banda\ de\ atenuación$$

Filtros pasa-bajas (*LP*)

Los filtros pasa-bajas son usados cuando es importante limitar el contenido de una señal a las frecuencias altas.



a_{pass} Es la ganancia de la banda de paso, se extiende desde *DC* hasta la f_{pass}

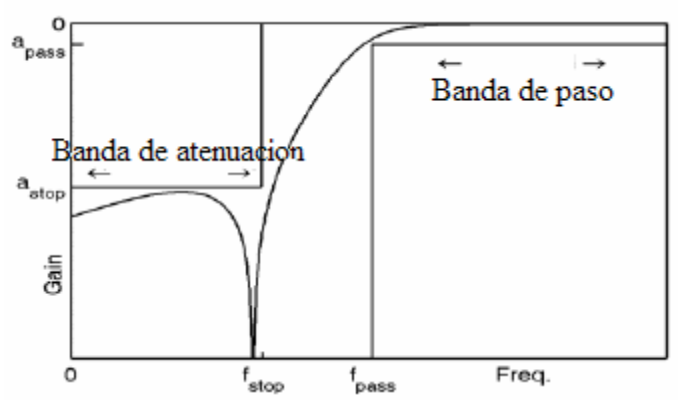
a_{stop} Es la ganancia de la banda de atenuación, se extiende desde la f_{stop} al infinito

f_{pass} O f_c frecuencia de corte

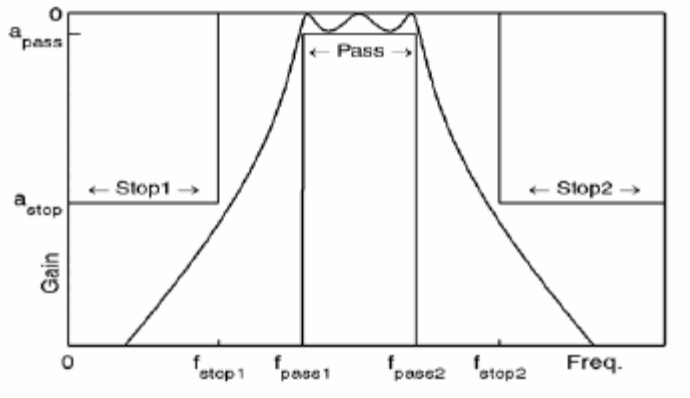
f_{stop} Frecuencia de eliminación

Filtros pasa-altas (*HP*)

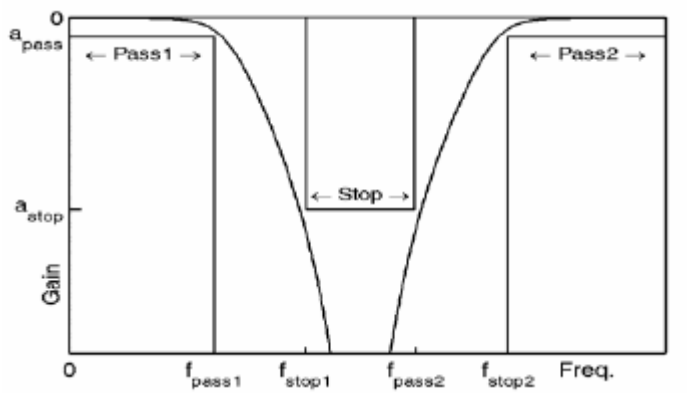
Los filtros pasa-altas son usados cuando es importante eliminar las frecuencias bajas de una señal.



Filtros pasa-banda (*BP*)



Filtros elimina-banda (*BS*)



Implementación de filtros digitales

Hay dos maneras de ***implementar filtros digitales***:

(1) La manera más sencilla de implementar un filtro digital es **convolucionando** la señal de entrada con el *kernel*, a este tipo de filtros se les llama *Finite Impulse Response* o filtros ***FIR***.

(2) Otra manera de implementar filtros digitales es por **recursión**, se usan muestras calculadas previamente de la salida y junto con las muestras de entrada se calcula la salida. Los filtros recursivos en vez de usar un *kernel* usan un conjunto de *coeficientes de recursión*. A este tipo de filtros que son implementados por recursión se les llama también *Infinite Impulse Response* o filtros ***IIR***.

2.3 Filtros digitales IIR (infinite impulse response)

Los filtros implementados por recursión o filtros *IIR* se ejecutan más rápido que los filtros implementados por convolución o *FIR* , pero tienen menos desempeño que los filtros *FIR* , porque para una respuesta a la frecuencia dada requieren de pocos delays, sumadores y multiplicadores, la desventaja es su fase no lineal.
Este tipo de filtros *teóricamente* pueden tener respuestas al impulso que continúan por siempre (tienden al infinito), es por eso que a estos filtros digitales se les llama filtros con respuesta infinita al impulso o *IIR*.
Hay una variedad de métodos para diseñar filtros digitales *IIR*, un método muy común es usar una función de aproximación de un filtro analógico que ya ha sido diseñada y simplemente se traslada de alguna manera esta aproximación (*Método de la respuesta al impulso invariante, Método de la transformación bilineal, etc…)* para usar en un sistema discreto. Las aproximaciones más conocidas son: Butterworth, Chebyshev, Cauer, Bessel.

La forma general de la función de transferencia de un filtro IIR es

$$H(Z) = \frac{\sum_{k=0}^{M} b_k \cdot z^{-k}}{1 + \sum_{k=1}^{N} a_k \cdot z^{-k}}$$

2.3.1 Funciones de aproximación

La realización de un filtro ideal es inalcanzable, lo mejor que se puede hacer es una aproximación, hay varias aproximaciones que se pueden usar, cada aproximación tiene relativas ventajas y desventajas.
Los aproximantes más conocidos son: Butterworth, Chebyshev, Bessel, Cauer.

Butterworth

Aproximante Butterworth normalizado
Esta aproximación tiene una transición uniforme (*no ripple*) a través de la banda de paso y la banda de transición, la fase también es uniforme lo cual es importante cuando se requiere baja distorsión de fase y selectividad moderada.

Magnitud

$$\left| H_{B,n}\left[j(w/w_0) \right] \right| = \frac{1}{\sqrt{1 + \varepsilon^2 \cdot (w/w_o)^{2 \cdot n}}}$$

w_0 Es la frecuencia pasa-banda
n Es el orden de la función de aproximación
ε Es el factor de ajuste de la ganancia pasa-banda
B Indica un filtro Butterworth

Ganancia

$$\varepsilon = \sqrt{10^{-0.1 \cdot a_{pass}} - 1}$$

La respuesta a la ganancia decrece en un factor de $-20 \cdot n$ *dB* por década al cambio de frecuencia.

Orden
El orden de un filtro Butterworth depende de las especificaciones dadas (*frecuencias y ganancias*).

$$n_B = \frac{\log\left[(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)\right]}{2\cdot\log(w_{stop}/w_{pass})}$$

Función de transferencia

(n par) $$H_{B,n}(S) = \frac{\prod_m (B_{2m})}{\prod_m (S^2 + B_{1m}\cdot S + B_{2m})}$$

m=0,1,…,(n/2)-1

(n impar) $$H_{B,n}(S) = \frac{R\cdot\prod_m (B_{2m})}{(S+R)\cdot\prod_m (S^2 + B_{1m}\cdot S + B_{2m})}$$

m=0,1,…,(n-1)/2-1

Radio

$$R = \varepsilon^{-1/n}$$

Ángulos

(n par) $$\theta_m = \frac{\pi\cdot(2\cdot m + n + 1)}{2\cdot n} \qquad m=0,1,\ldots,(n/2)-1$$

(n impar) $$\theta_m = \frac{\pi\cdot(2\cdot m + n + 1)}{2\cdot n} \qquad m=0,1,\ldots,(n-1)/2-1$$

Localización de los polos

$$S = \sigma \pm j\omega$$

$$\sigma_m = R\cdot\cos(\theta_m)$$

$$\omega_m = R\cdot\sin(\theta_m)$$

Si es ***n impar*** tendrá un polo real en

$$\sigma_r = -R$$

Coeficientes de la función de transferencia

$$B_{1m} = -2\cdot\sigma_m$$
$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

Ejercicio 2.3.1

Determinar el orden del filtro, los polos, los coeficientes y la función de transferencia para las siguientes especificaciones:

$$a_{pass} = -3dB, a_{stop} = -18dB, w_{pass} = 1 rad/seg, w_{stop} = 2 rad/seg$$

Primero hay que calcular las constantes, para esto usaremos matlab.

Script Matlab:

```
clear all
a_pass = -1;
a_stop = -18;
w_pass = 1;
w_stop = 2;
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
```

$$\varepsilon = \sqrt{10^{-0.1 \cdot a_{pass}} - 1} = 0.5088$$

```
%Calcula orden
nb = (log((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))/...
(2*log(w_stop/w_pass))
```

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(w_{stop}/w_{pass})} = 3.9529$$

$$n_B = 4 \qquad \textit{Orden filtro Butterworth}$$

```
%calcula radio
R = E^(-1/nb)
```

$$R = \varepsilon^{-1/n} = 1.1840$$

```
%calcula ángulos
mm= input('n es [0]par [1]impar = ','s');
mm = str2num(mm);
if mm == 0
    disp 'par';
    m = n/2 -1
end
if mm == 1
    disp 'impar';
    m = (n - 1)/2 -1
end
for i=1:m+1
theta(i)=(sym('pi') *(2*(i-1) + n + 1))/(2*n);
tt(i)=(pi*(2*(i-1) + n + 1))/(2*n);
end
theta
```

(n par) $\theta_m = \dfrac{\pi \cdot (2 \cdot m + n + 1)}{2 \cdot n}$ *m=0,1,…,(n/2)-1*

$$\theta_0 = \frac{\pi \cdot (2 \cdot 0 + 4 + 1)}{2 \cdot 4} = \frac{5\pi}{8}$$

$$\theta_1 = \frac{\pi \cdot (2 \cdot 1 + 4 + 1)}{2 \cdot 4} = \frac{7\pi}{8}$$

```
%calcula los polos
for i=1:m+1
polos(i)=R*cos(tt(i))+j*R*sin(tt(i));
end
polos
```

$$S = \sigma \pm j\omega$$

$$\sigma_m = R \cdot \cos(\theta_m)$$

$$\omega_m = R \cdot \sin(\theta_m)$$

$$S_0 = -0.4531 \pm j1.0939$$

$$S_1 = -1.0939 \pm j0.4531$$

```
%calcula coeficientes
for i=1:m+1
B1m(i)=-2*R*cos(tt(i));
B2m(i)=(R*cos(tt(i))).^2+(R*sin(tt(i))).^2;
end
B1m
B2m
```

$$B_{1m} = -2 \cdot \sigma_m$$

$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

Matlab nos entrega los coeficientes para sustituir en la función de transferencia directamente.

```
B1m =

    0.9062    2.1878

B2m =

   1.4019    1.4019
```

(n par) $$H_{B,n}(S)=\frac{\prod_{m}(B_{2m})}{\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})}$$

$$m=0,1,\ldots,(n/2)-1$$

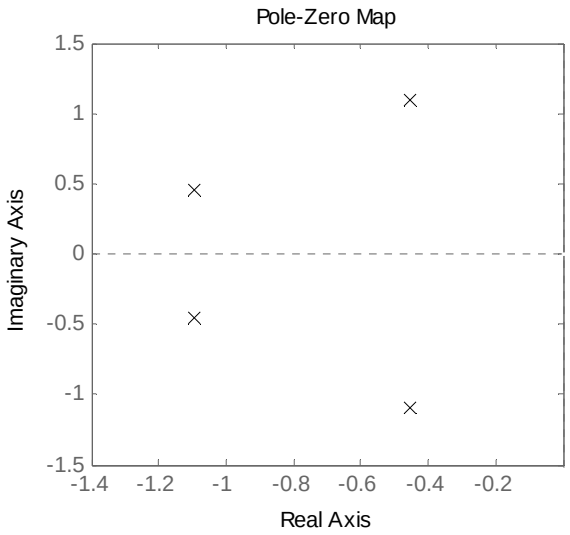
$$H_{B,4}(S)=\frac{1.4019}{(S^2+0.9062\cdot S+1.4019)}\cdot\frac{1.4019}{(S^2+2.1878\cdot S+1.4019)}$$

$$H_{B,4}(S)=\frac{1.4019^2}{(S^2+0.9062\cdot S+1.4019)\cdot(S^2+2.1878\cdot S+1.4019)}$$

Ahora se tiene que expandir la función de transferencia de arriba para obtener el plano-s en Matlab.

```
syms s
%expandimos función de transferencia para graficar polos y ceros
expand((s^2+2.1878*s+1.4019)*(s^2+0.9062*s+1.4019))
num= [0 0 0 0 (1.4019)^2];
den= [1 1547/500 119659609/25000000 21687393/5000000 196532361/100000000];
H=tf(num,den)
pzmap(H)
```

```
Transfer function:
                                  1.965
   -----------------------------------------------
   s^4 + 3.094 s^3 + 4.786 s^2 + 4.337 s + 1.965
```



Filtros LP, HP, BP, SB

Desnormalización

Anteriormente se vio el aproximante Butterworth normalizado, las desnormalización de la función de aproximación sirve para producir filtros pasa-bajas, pasa-altas, pasa-banda y elimina-banda en la frecuencia deseada.

La Desnormalización tiene tres pasos:

(1) Determinar el orden Especificaciones desnormalizadas	**(2)** Determinar la función de Aproximación normalizada	**(3)** Desnormalizar la aproximación a *LP, HP, BP, BS*

Desnormalización LP del aproximante Butterworth

Usada para generar un filtro pasa-bajas.
(1) El primer paso para desnormalizar es determinar el orden de la función de aproximación usando las especificaciones desnormalizadas.

$$\Omega_{rL} = \frac{w_{stop}}{w_{pass}} = \frac{f_{stop}}{f_{pass}}$$

Redefiniendo términos tenemos:

$$n_B = \frac{\log\left[(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)\right]}{2\cdot\log(\Omega_{rL})}$$

(2) El segundo paso para desnormalizar es determinar la función de aproximación normalizada.

(3) Reemplazar

$$S_L = \frac{s}{w_0}$$

$$w_0 = w_{pass}$$

$$w_0 = f_{pass\cdot}\cdot\frac{1rad/seg}{0.159154943092\cdot Hz} \quad \textbf{o} \quad f_{stop}\cdot\frac{1rad/seg}{0.318309886184\cdot Hz}$$

Manejo de los factores

- Tenemos la función de transferencia normalizada

$$H(S) = \left. \frac{A_1 \cdot S + A_2}{B_1 \cdot S + B_2} \right|_{S = s/w_0}$$

$$H(S) = \frac{A_1 \cdot \frac{s}{w_0} + A_2}{B_1 \cdot \frac{s}{w_0} + B_2} = \frac{A_1 \cdot s + A_2 \cdot w_0}{B_1 \cdot s + B_2 \cdot w_0} = \frac{a_1 \cdot s + a_2}{b_1 \cdot s + b_2}$$

Generalizando los resultados tenemos

$$a_1 = A_1$$
$$a_2 = A_2 \cdot w_0$$

$$b_1 = B_1$$
$$b_2 = B_2 \cdot w_0$$

- En otro caso tendremos

$$H(S) = \left. \frac{A_0 \cdot S^2 + A_1 \cdot S + A_2}{B_0 \cdot S^2 + B_1 \cdot S + B_2} \right|_{S = s/w_0}$$

$$H(S) = \frac{A_0 \cdot (\frac{s}{w_0})^2 + A_1 \cdot (\frac{s}{w_0}) + A_2}{B_0 \cdot (\frac{s}{w_0})^2 + B_1 \cdot (\frac{s}{w_0}) + B_2}$$

$$H(s) = \frac{a_0 \cdot s^2 + a_1 \cdot s + a_2}{b_0 \cdot s^2 + b_1 \cdot s + b_2}$$

Generalizando los resultados tenemos

$$a_0 = A_0$$
$$a_1 = A_1 \cdot w_0$$

$$a_2 = A_2 \cdot w_0^{\,2}$$

$$b_0 = B_0$$
$$b_1 = B_1 \cdot w_0$$
$$b_2 = B_2 \cdot w_0^{\,2}$$

Ejercicio 2.3.2

Determinar la función de transferencia para un filtro Butterworth pasa-bajas, para las siguientes especificaciones:

$$a_{pass} = -0.5dB\,,\, a_{stop} = -21dB$$

$$f_{pass} = 1khz\,,\, f_{stop} = 2khz$$

Script Matlab:

```
clear all
a_pass = -0.5;
a_stop = -21;
f_pass = 1000;
f_stop = 2000;
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
%calcula omega
omega=f_stop/f_pass;
%Calcula orden
nb = (log((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))/...
(2*log(omega))
```

$$\Omega_{rL} = \frac{w_{stop}}{w_{pass}} = \frac{f_{stop}}{f_{pass}} = \frac{2.000}{1000} = 2$$

$$\varepsilon = \sqrt{10^{-0.1 \cdot a_{pass}} - 1} = 0.3493$$

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(\Omega_{rL})} = 4.9997$$

$$n_B = 5 \qquad orden$$

```
n = input('Orden > nb =','s');
n = str2num(n);

%calcula radio
R = E^(-1/n)
```

$$R = \varepsilon^{-1/n} = 1.2341$$

```
%calcula ángulos
mm= input('n es [0]par [1]impar = ','s');
mm = str2num(mm);
if mm == 0
    disp 'par';
    m = n/2 -1
end
if mm == 1
    disp 'impar';
    m = (n - 1)/2 -1
end
for i=1:m+1
theta(i)=(sym('pi') *(2*(i-1) + n + 1))/(2*n);
tt(i)=(pi*(2*(i-1) + n + 1))/(2*n);
end
theta
```

(n impar) $\theta_m = \dfrac{\pi \cdot (2 \cdot m + n + 1)}{2 \cdot n}$ *m=0,1,…,(n-1)/2-1*

$$\theta_0 = \frac{3\pi}{5}$$

$$\theta_1 = \frac{4\pi}{5}$$

```
%calcula los polos
for i=1:m+1
polos(i)=R*cos(tt(i))+j*R*sin(tt(i));
end
polos
```

$$S = \sigma_m \pm j\omega_m$$
$$\sigma_m = R \cdot \cos(\theta_m)$$
$$\omega_m = R \cdot \sin(\theta_m)$$

$$S_0 = -0.381 \pm j1.1737$$
$$S_1 = -0.9984 \pm j0.7254$$

```
%calcula coeficientes
for i=1:m+1
B1m(i)=-2*R*cos(tt(i));
B2m(i)=(R*cos(tt(i))).^2+(R*sin(tt(i))).^2;
end
B1m
B2m
```

```
B1m =
    0.7627    1.9968

B2m =
   1.5231    1.5231
```

(n impar) $H_{B,n}(S) = \dfrac{R \cdot \prod_m (B_{2m})}{(S+R) \cdot \prod_m (S^2 + B_{1m} \cdot S + B_{2m})}$

m=0,1,…,(n-1)/2-1

La función de transferencia normalizada quedaría así:

$$H_{B,5}(S) = \frac{1.2341}{(S+1.2341)} \cdot \frac{1.5231}{(S^2+0.7627 \cdot S+1.5231)} \cdot \frac{1.5231}{(S^2+1.9968 \cdot S+1.5231)}$$

Normalizando la frecuencia

$$w_0 = w_{pass} = 1000Hz \cdot \frac{1\,rad/seg}{0.159154943092Hz} = 6283.19\,rad/seg$$

Desnormalizando la función de transferencia

$$S_L = \frac{s}{w_0}$$

$$w_0 = w_{pass} = 6283.19\,rad/seg$$

(1) **(2)** **(3)**

$$H_{B,5}(S) = \frac{1.2341}{(\frac{s}{w_0}+1.2341)} \cdot \frac{1.5231}{((\frac{s}{w_0})^2+0.7627 \cdot (\frac{s}{w_0})+1.5231)} \cdot \frac{1.5231}{((\frac{s}{w_0})^2+1.9968 \cdot (\frac{s}{w_0})+1.5231)}$$

Aplicando directamente el manejo de factores para no desarrollar todo, solo debemos calcular los coeficientes.

(1)

$a_2 = A_2 \cdot w_0 = 1.2341 * 6283.19 = 7754$

$b_1 = B_1 = 1$

$b_2 = B_2 \cdot w_0 = 1.2341 * 6283.19 = 7754$

(2)

$a_2 = A_2 \cdot w_0{}^2 = 1.5231 * (6283.19)^2 = 6.013 \cdot 10^7$

$b_0 = B_0 = 1$

$b_1 = B_1 \cdot w_0 = 0.7627 * 6283.19 = 4792$

$b_2 = B_2 \cdot w_0{}^2 = 1.5231 * (6283.19)^2 = 6.013 \cdot 10^7$

(3)

$a_2 = A_2 \cdot w_0{}^2 = 1.5231 * (6283.19)^2 = 6.013 \cdot 10^7$

$b_0 = B_0 = 1$

$b_1 = B_1 \cdot w_0 = 1.9968 * 6283.19 = 1.254 \cdot 10^4$

$$b_2 = B_2 \cdot w_0^{\ 2} = 1.5231 * (6283.19)^2 = 6.013 \cdot 10^7$$

El filtro Butterworth LP quedaría expresado así:

$$H_{B,5}(s) = \frac{a_2}{(b_1 \cdot s + b_2)} \cdot \frac{a_2}{(b_0 \cdot s^2 + b_1 \cdot s + b_2)} \cdot \frac{a_2}{(b_0 \cdot s^2 + b_1 \cdot s + b_2)}$$

$$H_{B,5}(s) = \frac{7754}{(s + 7754)} \cdot \frac{6.013 \cdot 10^7}{(s^2 + 1.254 \cdot 10^4 \cdot s + 6.013 \cdot 10^7)} \cdot \frac{6.013 \cdot 10^7}{(s^2 + 4792 \cdot s + 6.013 \cdot 10^7)}$$

Desnormalización HP del aproximante Butterworth

Usada para generar un filtro pasa-altas.

$$\Omega_{rH} = \frac{w_{pass}}{w_{stop}} = \frac{f_{pass}}{f_{stop}}$$

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(\Omega_{rH})}$$

$$S_H = \frac{w_0}{s}$$

$$w_0 = w_{pass}$$

$$w_0 = f_{pass} \cdot \frac{1 rad/seg}{0.159154943092 \cdot Hz} \ \textbf{o} \ f_{stop} \cdot \frac{1 rad/seg}{0.318309886184 \cdot Hz}$$

Manejo de los factores

- $$H(S) = \left. \frac{A_1 \cdot S + A_2}{B_1 \cdot S + B_2} \right|_{S = w_0/s}$$

$$H(S) = \frac{A_1 \cdot \frac{w_0}{s} + A_2}{B_1 \cdot \frac{w_0}{s} + B_2} = \frac{A_2}{B_2} \cdot \frac{s + (A_1/A_2) \cdot w_0}{s + (B_1/B_2) \cdot w_0} = \frac{A_2}{B_2} \cdot \frac{a_1 \cdot s + a_2}{b_1 \cdot s + b_2}$$

Generalizando los resultados tenemos

$$a_1 = 1$$
$$a_2 = \frac{A_1}{A_2} \cdot w_0$$

$$b_1 = 1$$
$$b_2 = \frac{B_1}{B_2} \cdot w_0$$

-

$$H(S) = \left. \frac{A_0 \cdot S^2 + A_1 \cdot S + A_2}{B_0 \cdot S^2 + B_1 \cdot S + B_2} \right|_{S = w_0 / s}$$

$$H(S) = \frac{A_0 \cdot (\frac{w_0}{s})^2 + A_1 \cdot (\frac{w_0}{s}) + A_2}{B_0 \cdot (\frac{w_0}{s})^2 + B_1 \cdot (\frac{w_0}{s}) + B_2}$$

Simplificando

$$H(s) = \frac{A_2}{B_2} \cdot \frac{s^2 + (A_1 / A_2) \cdot w_0 \cdot s + (A_0 / A_2) \cdot {w_0}^2}{s^2 + (B_1 / B_2) \cdot w_0 \cdot s + (B_0 / B_2) \cdot {w_0}^2}$$

$$H(s) = \frac{A_2}{B_2} \cdot \frac{a_0 \cdot s^2 + a_1 \cdot s + a_2}{b_0 \cdot s^2 + b_1 \cdot s + b_2}$$

Generalizando los resultados tenemos

$$a_0 = 1$$
$$a_1 = \frac{A_1}{A_2} \cdot w_0$$
$$a_2 = \frac{A_0}{A_2} \cdot {w_0}^2$$
$$b_0 = 1$$
$$b_1 = \frac{B_1}{B_2} \cdot w_0$$

$$b_2 = \frac{B_0}{B_2} \cdot {w_0}^2$$

Desnormalización BP del aproximante Butterworth

Usada para generar una función de aproximación para un filtro pasa-banda.

$$\Omega_{rP} = \frac{w_{stop2} - w_{stop1}}{w_{pass2} - w_{pass1}} = \frac{f_{stop2} - f_{stop1}}{f_{pass2} - f_{pass1}}$$

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(\Omega_{rP})}$$

$$S_P = \frac{s^2 + {w_0}^2}{BW \cdot s}$$

$$w_0 = \sqrt{w_{pass1} \cdot w_{pass2}}$$

$$BW = w_{pass2} - w_{pass1}$$

$$w = f \cdot \frac{1 rad/seg}{0.159154943092 \cdot Hz} \text{ o } f \cdot 2 \cdot \pi \, rad/seg$$

Las frecuencias de la banda de atenuación y la banda de paso deben ser simétricas a ambos lados de w_0, para comprobar la simetría se usa esta ecuación:

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si la ecuación no satisface, el lado que sea mayor debe ser reducido, para esto debe ir incrementando w_{stop1} o decrementando w_{stop2}.

NOTA: La función de aproximación desnormalizada resultante resulta ser el doble que la función normalizada.

Manejo de los factores

- $$H(S) = \left. \frac{A_1 \cdot S + A_2}{B_1 \cdot S + B_2} \right|_{S=\frac{s^2+{w_0}^2}{BW \cdot s}}$$

$$H(S) = \frac{A_1 \cdot \frac{s^2 + {w_0}^2}{BW \cdot s} + A_2}{B_1 \cdot \frac{s^2 + {w_0}^2}{BW \cdot s} + B_2}$$

Simplificando tenemos

$$H(s) = \frac{A_1 \cdot s^2 + A_2 \cdot BW \cdot s + A_1 \cdot {w_0}^2}{B_1 \cdot s^2 + B_2 \cdot BW \cdot s + B_1 \cdot {w_0}^2} = \frac{a_0 \cdot s^2 + a_1 \cdot s + a_2}{b_0 \cdot s^2 + b_1 \cdot s + b_2}$$

Generalizando los resultados tenemos

$$a_0 = A_1$$
$$a_1 = A_2 \cdot BW$$
$$a_2 = A_1 \cdot {w_0}^2$$

$$b_0 = B_1$$
$$b_1 = B_2 \cdot BW$$
$$b_2 = B_1 \cdot {w_0}^2$$

- $$H(S) = \left. \frac{A_2}{B_0 \cdot S^2 + B_1 \cdot S + B_2} \right|_{\frac{s^2+{w_0}^2}{BW \cdot s}}$$

Hallamos los polos de la función de transferencia denotados por *p*

$$H(S) = \left. \frac{A_2}{(S + p) \cdot (S + p^*)} \right|_{\frac{s^2+{w_0}^2}{BW \cdot s}}$$

Simplificando

$$H(s)=\frac{A_2}{\left(\frac{s^2+w_0{}^2+p\cdot BW\cdot s}{BW\cdot s}\right)\cdot\left(\frac{s^2+w_0{}^2+p^*\cdot BW\cdot s}{BW\cdot s}\right)}$$

$$H(s)=\frac{A_2\cdot BW^2\cdot s^2}{\left(s^2+w_0{}^2+p\cdot BW\cdot s\right)\cdot\left(s^2+w_0{}^2+p^*\cdot BW\cdot s\right)}$$

$$H(s)=\frac{A_2\cdot BW^2\cdot s^2}{\left(s^2+b_1\cdot s+b_2\right)\cdot\left(s^2+b_4\cdot s+b_5\right)}$$

Ejercicio 2.3.3

Determinar la función de transferencia para un filtro Butterworth pasa-banda, para las siguientes especificaciones:

$$a_{pass}=-1dB\,,a_{stop}=-21dB$$
$$f_{pass}1=300hz\,,\ f_{stop1}=50hz$$
$$f_{pass2}=3khz\,,\ f_{stop2}=9khz$$

Comprobación de simetría

$$\frac{w_{pass1}}{w_{stop1}}=\frac{w_{stop2}}{w_{pass2}}$$

$$\frac{300\cdot 2\pi}{50\cdot 2\pi}=\frac{9000\cdot 2\pi}{3000\cdot 2\pi}$$

$6=3$ *No simetría*

Cambiando $f_{stop1}=100hz$

$$\frac{300\cdot 2\pi}{100\cdot 2\pi}=\frac{9000\cdot 2\pi}{3000\cdot 2\pi}$$

$3=3$ *Simetría*

Script Matlab:

```
%ingresa especificaciones
clear all
a_pass = -1;
a_stop = -21;
f_pass1 = 300;
f_stop1 = 100;
f_pass2 = 3000;
f_stop2 = 9000;
%calcula omega
omega=(f_stop2-f_stop1)/(f_pass2-f_pass1);
```

$$\Omega_{rP} = \frac{w_{stop2} - w_{stop1}}{w_{pass2} - w_{pass1}} = \frac{f_{stop2} - f_{stop1}}{f_{pass2} - f_{pass1}} = 3.2963$$

```
%Calcula orden
nb = (log((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))/...
(2*log(omega))
```

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(\Omega_{rP})} = 2.59$$

$$n_B = 3$$

```
n = input('Orden > nb =','s');
n = str2num(n);
%normaliza frecuencias
wpass1 = f_pass1*2*pi
wpass2 = f_pass2*2*pi
%calcula wo
wo = sqrt(wpass1*wpass2)
```

$$w_0 = \sqrt{w_{pass1} \cdot w_{pass2}} = 5960.8 \, rad/seg$$

```
%calcula BW
BW = wpass2 - wpass1
```

$$BW = w_{pass2} - w_{pass1} = 16965$$

```
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
```

$$\varepsilon = \sqrt{10^{-0.1 \cdot a_{pass}} - 1} = 0.5088$$

```
%calcula radio
R = E^(-1/n)
```

$$R = \varepsilon^{-1/n} = 1.2526$$

```
%calcula ángulos
mm= input('n es [0]par [1]impar = ','s');
mm = str2num(mm);
if mm == 0
    disp 'par';
```

```
    m = n/2 -1
end
if mm == 1
    disp 'impar';
    m = (n - 1)/2 -1
end
for i=1:m+1
theta(i)=(sym('pi') *(2*(i-1) + n + 1))/(2*n);
tt(i)=(pi*(2*(i-1) + n + 1))/(2*n);
end
theta
```

(n impar) $\theta_m = \dfrac{\pi \cdot (2 \cdot m + n + 1)}{2 \cdot n}$ *m=0,1,…,(n-1)/2-1*

$$\theta_0 = \frac{2\pi}{3}$$

```
%calcula los polos
for i=1:m+1
polos(i)=R*cos(tt(i))+j*R*sin(tt(i));
end
polos
```

$$S = \sigma \pm j\omega$$

$$\sigma_m = R \cdot \cos(\theta_m)$$

$$\omega_m = R \cdot \sin(\theta_m)$$

$$S_0 = -0.6263 + j1.0848$$

$$\sigma_0 = -0.6263 \qquad \omega_0 = 1.0848$$

```
%calcula coeficientes
for i=1:m+1
B1m(i)=-2*R*cos(tt(i));
B2m(i)=(R*cos(tt(i))).^2+(R*sin(tt(i))).^2;
end
B1m
B2m
```

$$B_{1m} = -2 \cdot \sigma_m$$

$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

```
B1m =
    1.2526

B2m =
    1.5689
```

(n impar) $$H_{B,n}(S)=\frac{R\cdot\prod_{m}(B_{2m})}{(S+R)\cdot\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})}$$ *m=0,1,…,(n-1)/2-1*

$$H_{B,3}(S)=\frac{1.2526}{(S+1.2526)}\cdot\frac{1.5689}{(S^2+1.2526+1.5689)}$$

Hallando los polos del factor de segundo orden tenemos:

$$H_{B,3}(S)=\frac{1.2526}{(S+1.2526)}\cdot\frac{1.5689}{(S+0.6263+1.0847j)(S+0.6263-1.0847j)}$$

$$H_{B,3}(S)=\underset{(1)}{\frac{1.2526}{(S+1.2526)}}\cdot\underset{(2)}{\frac{1.5689}{(S+0.6263\pm1.0847j)^2}}$$

El ultimo paso para desnormalizar es reemplazar $S_P=\frac{s^2+w_0{}^2}{BW\cdot s}$, empleando manejo de factores

(1)

$$a_1=A_2\cdot BW=1.2526*BW\cdot s$$

$$b_0=B_1=1$$
$$b_1=B_2\cdot BW=1.2526\cdot BW\cdot s=21250\cdot s$$
$$b_2=B_1\cdot w_0{}^2=1\cdot(5969.8)^2=3.553\cdot10^7$$

$$\frac{1.2526}{(S+1.2526)}=\frac{1.2526\cdot(16965)\cdot s}{s^2+21250\cdot s+3.553\cdot10^7}$$

(2)

$$\frac{1.5689}{\left(\frac{s^2+w_0{}^2}{BW\cdot s}+0.6263+1.0847j\right)\left(\frac{s^2+w_0{}^2}{BW\cdot s}+0.6263-1.0847j\right)}$$

$$\frac{1.5689\cdot BW^2\cdot s^2}{(s^2+w_0{}^2+0.6263\cdot BW\cdot s+1.0847j\cdot BW\cdot s)\cdot(s^2+w_0{}^2+0.6263\cdot BW\cdot s-1.0847j\cdot BW\cdot s)}$$

$$\frac{1.5689\cdot BW^2\cdot s^2}{(s^2+w_0{}^2+0.6263\cdot BW\cdot s+1.0847j\cdot BW\cdot s)\cdot(s^2+w_0{}^2+0.6263\cdot BW\cdot s-1.0847j\cdot BW\cdot s)}$$

$$\frac{1.5689\cdot BW^2\cdot s^2}{\left(s^2+10625\cdot s+18402j\cdot s+3.553\cdot 10^7\right)\cdot\left(s^2+10625\cdot s-18402j\cdot s+3.553\cdot 10^7\right)}$$

$$\frac{1.5689\cdot BW^2\cdot s^2}{\left(s^2+s\cdot(10625+18402j)+3.553\cdot 10^7\right)\cdot\left(s^2+s\cdot(10625-18402j)+3.553\cdot 10^7\right)}$$

Hallamos nuevamente los polos, usando un comando ya visto de Matlab *roots( )*

```
roots([1 10625+18402*j 3.553*10^7])
ans =
  1.0e+004 *
  -0.9909 - 1.9835i
  -0.0716 + 0.1433i

roots([1 10625-18402*j 3.553*10^7])

ans =
  1.0e+004 *
  -0.9909 + 1.9835i
  -0.0716 - 0.1433i
```

$$\frac{1.5689\cdot BW^2\cdot s^2}{(s+9909+19835j)\cdot(s+716-1433j)\cdot(s+9909-19835j)\cdot(s+716+1433j)}$$

$$\frac{1.5689\cdot BW^2\cdot s^2}{(s+9909\pm 19835j)^2\cdot(s+716\pm 1433j)^2}$$

La función de transferencia para el filtro Butterworth pasa-bajas sería la siguiente:

$$H_{B,6}(S)=\underbrace{\frac{1.2526\cdot(16965)\cdot s}{s^2+21250\cdot s+3.553\cdot 10^7}}_{\textbf{(1)}}\cdot\underbrace{\frac{1.5689\cdot BW^2\cdot s^2}{(s+9909\pm 19835j)^2\cdot(s+716\pm 1433j)^2}}_{\textbf{(2)}}$$

$$H_{B,6}(S)=\frac{1.2526\cdot(16965)^3\cdot 1.5689\cdot s^3}{s^2+21250\cdot s+3.553\cdot 10^7\cdot(s+9909\pm 19835j)^2\cdot(s+716\pm 1433j)^2}$$

Desarrollando

$(s+9909\pm 19835j)^2$

$=s^2+716\cdot s+1433j\cdot s+716\cdot s+716^2+1433j\cdot 716-1433j\cdot s-1433j\cdot 716+1433^2$
$=s^2+1432\cdot s+2566145$
$(s+716\pm 1433j)^2$
$=s^2+198185+491615506$

$$H_{B,6}(S) = \frac{1.2526 \cdot (16965)^3 \cdot 1.5689 \cdot s^3}{\left(s^2 + 21250 \cdot s + 3.553 \cdot 10^7\right) \cdot \left(s^2 + 1432 \cdot s + 2566145\right) \cdot \left(s^2 + 198185 + 491615506\right)}$$

Desnormalización SB del aproximante Butterworth

Usada para generar una función de aproximación para un filtro elimina-banda, es un caso muy similar al BP (*pasa-banda*).

$$\Omega_{rS} = \frac{w_{pass2} - w_{pass1}}{w_{stop2} - w_{stop1}} = \frac{f_{pass2} - f_{pass1}}{f_{stop2} - f_{stop1}}$$

Debe satisfacer la simetría en esta ecuación

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si no satisface la ecuación se debe incrementar w_{pass1} o decrementar w_{pass2} para satisfacer la ecuación de arriba.

$$n_B = \frac{\log\left[(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)\right]}{2 \cdot \log(\Omega_{rS})}$$

$$S_S = \frac{BW \cdot s}{s^2 + {w_0}^2}$$

$$w_0 = \sqrt{w_{pass1} \cdot w_{pass2}}$$

$$BW = w_{pass2} - w_{pass1}$$

$$w = f \cdot \frac{1 rad/seg}{0.159154943092 \cdot Hz} \quad \mathbf{o} \quad f \cdot 2 \cdot \pi \, rad/seg$$

Manejo de los factores

- $$H(S) = \left.\frac{A_1 \cdot S + A_2}{B_1 \cdot S + B_2}\right|_{S = \frac{BW \cdot s}{s^2 + {w_0}^2}}$$

$$H(S)=\frac{A_1\cdot\frac{BW\cdot s}{s^2+{w_0}^2}+A_2}{B_1\cdot\frac{BW\cdot s}{s^2+{w_0}^2}+B_2}$$

Simplificando tenemos

$$H(s)=\frac{A_2}{B_2}\cdot\frac{s^2+(A_1/A_2)\cdot BW\cdot s+{w_0}^2}{s^2+(B_1/B_2)\cdot BW\cdot s+{w_0}^2}=\frac{a_0\cdot s^2+a_1\cdot s+a_2}{b_0\cdot s^2+b_1\cdot s+b_2}$$

Generalizando los resultados tenemos

$$a_0=1$$
$$a_1=(A_1/A_2)\cdot BW$$
$$a_2={w_0}^2$$

$$b_0=1$$
$$b_1=(B_1/B_2)\cdot BW$$
$$b_2={w_0}^2$$

-

$$H(S)=\left.\frac{A_2}{B_0\cdot S^2+B_1\cdot S+B_2}\right|_{S=\frac{BW\cdot s}{s^2+{w_0}^2}}$$

Hallamos los polos de la función de transferencia denotados por *p*

$$H(S)=\left.\frac{A_2}{(S+p)\cdot(S+p^*)}\right|_{S=\frac{BW\cdot s}{s^2+{w_0}^2}}$$

Simplificando

$$H(s)=\frac{A_2}{p\cdot p^*}\cdot\frac{\left(s^2+{w_0}^2\right)^2}{\left(s^2+\frac{BW}{p}\cdot s+{w_0}^2\right)\cdot\left(s^2+\frac{BW}{p^*}\cdot s+{w_0}^2\right)}$$

$$H(s)=\frac{A_2}{B_2}\cdot\frac{\left(s^2+{w_0}^2\right)^2}{\left(s^2+b_1\cdot s+b_2\right)\cdot\left(s^2+b_4\cdot s+b_5\right)}$$

Chebyshev

Aproximante Chebyshev normalizado

La función de aproximación Chebyshev también tiene solo polos como el aproximante Butterworth, el aproximante Chebyshev tiene ripple en la banda de paso del filtro, tiene una transición más rápida que el aproximante Butterworth, por esta razón se necesitan filtros de menor orden.
La fase no es muy lineal, si se desea una baja distorsión este filtro no es el indicado.

Magnitud

$$\left|H_{C,n}\left[j(w/w_0)\right]\right|=\frac{1}{\sqrt{1+\varepsilon^2\cdot {C_n}^2\cdot(w/w_o)}}$$

w_0 Es la frecuencia pasa-banda

C_n Es el polinomio de Chebyshev

n Es el orden de la función de aproximación

ε Es el factor de ajuste de la ganancia pasa-banda

C Indica un filtro Chebyshev

Ganancia

$$\varepsilon=\sqrt{10^{-0.1\cdot a_{pass}}-1}$$

Orden

El orden de un filtro Chebyshev depende de las especificaciones dadas.

$$n_C=\frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}}-1)/(10^{-0.1\cdot a_{pass}}-1)}\right]}{\cosh^{-1}(w_{stop}/w_{pass})}$$

Función de transferencia

(n par) $$H_{C,n}(S)=\frac{G\cdot\prod_m(B_{2m})}{\prod_m(S^2+B_{1m}\cdot S+B_{2m})}$$

$m=0,1,\ldots,(n/2)-1$

(n impar) $$H_{C,n}(S) = \frac{\sinh(D)\cdot\prod_{m}(B_{2m})}{(S+\sinh(D))\cdot\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})}$$

m=0,1,…,(n-1)/2-1

Variables

$$D = \frac{\sinh^{-1}(\varepsilon^{-1})}{n}$$

$$G = 10^{0.05\cdot a_{pass}}$$

Ángulos

(n par) $$\phi_m = \frac{\pi\cdot(2\cdot m+1)}{2\cdot n}$$ *m=0,1,…,(n/2)-1*

(n impar) $$\phi_m = \frac{\pi\cdot(2\cdot m+1)}{2\cdot n}$$ *m=0,1,…,(n-1)/2-1*

Polos

$$S = \sigma \pm j\omega$$

$$\sigma_m = -\sinh(D)\cdot\sin(\phi_m)$$

$$\omega_m = \cosh(D)\cdot\cos(\phi_m)$$

Si ***n impar*** habrá un polo real en

$$\sigma_r = -\sinh(D)$$

Coeficientes de la función de transferencia

$$B_{1m} = -2\cdot\sigma_m$$

$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

Ejercicio 2.3.4

Determinar el orden, los coeficientes y la función de transferencia para un filtro Chebyshev, con las siguientes especificaciones:

$$a_{pass} = -1dB\,, a_{stop} = -22dB\,, w_{pass} = 1\,{}^{rad}\!/_{seg}\,, w_{stop} = 2\,{}^{rad}\!/_{seg}$$

Utilizaremos un script en Matlab más desarrollado que el presentado en Butterworth

Script Matlab:

```
%ingresa especificaciones
disp('*****************************');
disp('Calculos para filtro Chebyshev');
disp('*****************************');
clear all
a_pass = -1;%dB
a_stop = -22;
w_pass = 1;%rad/seg
w_stop = 2;
%Calcula orden
nc = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(w_stop/w_pass))
%redondea al entero mas cercano
n = round(nc)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
```

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(w_{stop}/w_{pass})} = 2.9599$$

$$n_C = 3$$

```
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
```

$$\varepsilon = \sqrt{10^{-0.1\cdot a_{pass}} - 1} = 0.5088$$

```
%calcula variables
D = asinh(E^(-1))/n
G = 10^(0.05*a_pass)
sinh_D = sinh(D)
```

$$D = \frac{\sinh^{-1}(\varepsilon^{-1})}{n} = 0.4760$$

$$G = 10^{0.05\cdot a_{pass}} = 0.8913$$

$$\sinh(D) = 0.4942$$

```
%calcula angulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
```

```
end
phi
```

(n impar) $\phi_m = \dfrac{\pi \cdot (2 \cdot m + 1)}{2 \cdot n}$ *m=0,1,…,(n-1)/2-1*

$$\phi_0 = \frac{1\pi}{6}$$

```
%calcula polos
for i=1:(n-m)/2
    polos(i) = -sinh(D)*sin(phi_s(i)) + j*cosh(D)*cos(phi_s(i));
    %Calculo independiente de sigma y omega
    sigma(i)= -sinh(D)*sin(phi_s(i));
    omega(i)= cosh(D)*cos(phi_s(i));
end
%si n impar hay un polo real
if m == 1
   polo_real = -sinh(D)
end
polos
disp('Representacion independiente');
sigma
omega
```

$$S = \sigma \pm j\omega$$
$$\sigma_m = -\sinh(D) \cdot \sin(\phi_m)$$
$$\omega_m = \cosh(D) \cdot \cos(\phi_m)$$

$$\sigma_0 = -0.2471$$
$$\omega_0 = 0.9660$$

n impar hay un polo real en

$$\sigma_r = -\sinh(D) = -0.4942$$

```
%calculo coeficientes función transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
end
B1m
B2m
```

$$B_{1m} = -2 \cdot \sigma_m$$

$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

$$B_{10} = 0.4942$$
$$B_{20} = 0.9942$$

La función de transferencia normalizada es

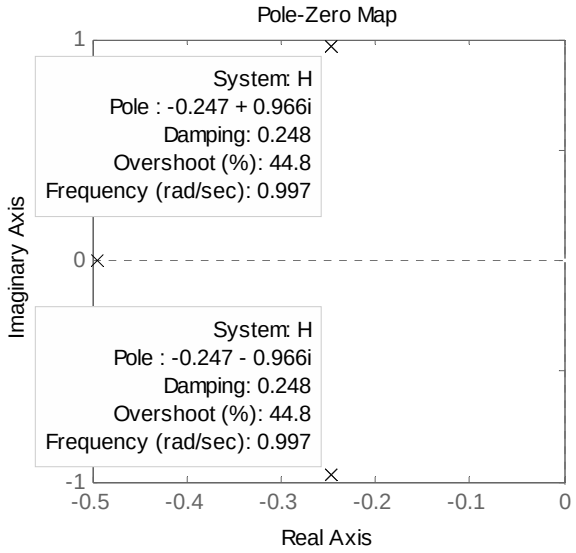
(n impar)

$$H_{C,n}(S) = \frac{\sinh(D)\cdot\prod_{m}(B_{2m})}{\left(S+\sinh(D)\right)\cdot\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})}$$

$$m=0,1,\ldots,(n-1)/2-1$$

$$H_{C,3}(S) = \frac{0.4942\cdot 0.9942}{\left(S+0.4942\right)\cdot(S^2+0.4942\cdot S+0.9942)}$$

```
%Datos reemplazar con función transferencia que calculo
%expandir función de transferencia para graficar polos y ceros
syms s
expand((s + 0.4942)*(s^2+0.4942*s+0.9942))
num = [0 0 0 0.4942*0.9942];
den = [1 2471/2500 30960841/25000000 12283341/25000000];
H=tf(num,den)
pzmap(H)
```



Filtros LP, HP, BP, SB

Desnormalización LP del aproximante Chebyshev

Usada para generar un filtro pasa-bajas.

$$\Omega_{rL} = \frac{w_{stop}}{w_{pass}} = \frac{f_{stop}}{f_{pass}}$$

Redefiniendo términos tenemos:

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rL})}$$

$$S_L = \frac{s}{w_0}$$

$$w_0 = w_{pass}$$

$$w = f \cdot 2\pi\, rad/seg$$

Desnormalización HP del aproximante Chebyshev

Usada para generar un filtro pasa-altas.

$$\Omega_{rH} = \frac{w_{pass}}{w_{stop}} = \frac{f_{pass}}{f_{stop}}$$

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rH})}$$

$$S_H = \frac{w_0}{s}$$

$$w_0 = w_{pass}$$

$$w = f \cdot 2\pi\, rad/seg$$

Ejercicio 2.3.5

Determinar la función de transferencia para un filtro Chebyshev pasa-altas (*HP*), con las siguientes especificaciones:

$$a_{pass} = -1.5dB\,,\, a_{stop} = -40dB\,,\, f_{pass} = 2khz\,,\, f_{stop} = 800Hz$$

Utilizaremos un script en Matlab más desarrollado que el presentado en Butterworth, para automatizar y agilizar el cálculo.

Script Matlab:

```
%filtro Chebyshev HP
%ingresa especificaciones
disp('****************************');
disp('      filtro Chebyshev HP');
```

```
disp('******************************');
clear all
a_pass = -1.5;%dB
a_stop = -40;
f_pass = 2000;%Hz
f_stop = 800;
w_pass = 2*pi*f_pass;%rad/seg
w_stop = 2*pi*f_stop;
%Calcula Omega
Omega = w_pass / w_stop
```

$$\Omega_{rH} = \frac{w_{pass}}{w_{stop}} = \frac{f_{pass}}{f_{stop}} = 2.5$$

```
%calcula Wo
wo = w_pass
```

$$w_0 = w_{pass} = 12566$$

```
%Calcula orden
nc = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(Omega))
%redondea al entero mas cercano
n = round(nc)
```

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rH})} = 3.6641$$

$$n_C = 4$$

```
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
```

$$\varepsilon = \sqrt{10^{-0.1\cdot a_{pass}} - 1} = 0.6423$$

```
%calcula variables
D = asinh(E^(-1))/n
G = 10^(0.05*a_pass)
sinh_D = sinh(D)
```

$$D = \frac{\sinh^{-1}(\varepsilon^{-1})}{n} = 0.3065$$

$$G = 10^{0.05\cdot a_{pass}} = 0.8414$$

$$\sinh(D) = 0.3113$$

```
%calcula ángulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
phi
```

(n par) $\phi_m = \dfrac{\pi \cdot (2 \cdot m + 1)}{2 \cdot n}$ *m=0,1,…,(n/2)-1*

$$\phi_0 = \frac{\pi}{8}$$

$$\phi_1 = \frac{3\pi}{8}$$

```
%calcula polos
for i=1:(n-m)/2
    polos(i) = -sinh(D)*sin(phi_s(i)) + j*cosh(D)*cos(phi_s(i));
    %Calculo independiente de sigma y omega
    sigma(i)= -sinh(D)*sin(phi_s(i));
    omega(i)= cosh(D)*cos(phi_s(i));
end
%si n impar hay un polo real
if m == 1
   polo_real = -sinh(D)
end
polos
disp('Representacion independiente');
sigma
omega
```

$$S = \sigma \pm j\omega$$
$$\sigma_m = -\sinh(D) \cdot \sin(\phi_m)$$
$$\omega_m = \cosh(D) \cdot \cos(\phi_m)$$

$$\sigma_0 = -0.1191 \qquad \omega_0 = 0.9676$$
$$\sigma_1 = -0.2876 \qquad \omega_1 = 0.4008$$

```
%calculo coeficientes función transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
end
B1m
B2m
```

$$B_{1m} = -2 \cdot \sigma_m$$
$$B_{2m} = \sigma_m{}^2 + \omega_m{}^2$$

$$B_{10} = 0.2383 \qquad B_{20} = 0.9505$$
$$B_{11} = 0.5752 \qquad B_{22} = 0.2435$$

Función de transferencia normalizada

(n par) $$H_{C,n}(S)=\frac{G\cdot\prod_{m}(B_{2m})}{\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})} \qquad m=0,1,\ldots,(n/2)-1$$

$$H_{C,4}(S)=\underbrace{(0.8414)}_{(1)}\cdot\underbrace{\frac{0.9505}{(S^2+0.2383\cdot S+0.9505)}}_{(2)}\cdot\underbrace{\frac{0.2435}{(S^2+0.5752\cdot S+0.2435)}}_{(3)}$$

(1)

$$(0.8414)$$

(2)

$$\frac{0.9505}{(S^2+0.2383\cdot S+0.9505)}=\frac{0.9505}{\left(\left(\frac{w_0}{s}\right)^2+0.2383\cdot\left(\frac{w_0}{s}\right)+0.9505\right)}$$

$$=\frac{0.9505\cdot s^2}{(w_0{}^2+0.2383\cdot w_0\cdot s+0.9505\cdot s^2)}=\frac{0.9505\cdot s^2}{(157904356+29945.5\cdot s+0.9505\cdot s^2)}$$

$$=\frac{s^2}{(s^2+3150.4\cdot s+1.6613\cdot 10^8)}$$

(3)

$$\frac{0.2435}{(S^2+0.5752\cdot S+0.2435)}=\frac{0.2435}{\left(\left(\frac{w_0}{s}\right)^2+0.5752\cdot\left(\frac{w_0}{s}\right)+0.2435\right)}$$

$$=\frac{0.2435\cdot s^2}{(w_0{}^2+0.5752\cdot w_0\cdot s+0.2435\cdot s^2)}=\frac{0.2435\cdot s^2}{(157904356+7228\cdot s+0.2435\cdot s^2)}$$

$$=\frac{s^2}{(s^2+29684\cdot s+6.4848\cdot 10^8)}$$

Función de transferencia desnormalizada

$$H_{C,4}(s)=\underbrace{(0.8414)}_{(1)}\cdot\underbrace{\frac{s^2}{(s^2+3150.4\cdot s+1.6613\cdot 10^8)}}_{(2)}\cdot\underbrace{\frac{s^2}{(s^2+29684\cdot s+6.4848\cdot 10^8)}}_{(3)}$$

$$H_{C,4}(s) = \frac{(0.8414)\cdot s^4}{\left(s^2 + 3150.4\cdot s + 1.6613\cdot 10^8\right)\cdot\left(s^2 + 29684\cdot s + 6.4848\cdot 10^8\right)}$$

Desnormalización BP del aproximante Chebyshev

Usada para generar una función de aproximación para un filtro pasa-banda.

$$\Omega_{rP} = \frac{w_{stop2} - w_{stop1}}{w_{pass2} - w_{pass1}} = \frac{f_{stop2} - f_{stop1}}{f_{pass2} - f_{pass1}}$$

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rP})}$$

$$S_P = \frac{s^2 + {w_0}^2}{BW\cdot s}$$

$$w_0 = \sqrt{w_{pass1}\cdot w_{pass2}}$$
$$BW = w_{pass2} - w_{pass1}$$
$$w = f\cdot 2\pi\, rad/seg$$

Las frecuencias de la banda de atenuación y la banda de paso deben ser simétricas a ambos lados de w_0, para comprobar la simetría se usa esta ecuación:

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si la ecuación no satisface, el lado que sea mayor debe ser reducido, para esto debe ir incrementando w_{stop1} o decrementando w_{stop2}.

Desnormalización SB del aproximante Chebyshev

Usada para generar una función de aproximación para un filtro elimina-banda.

$$\Omega_{rS} = \frac{w_{pass2} - w_{pass1}}{w_{stop2} - w_{stop1}} = \frac{f_{pass2} - f_{pass1}}{f_{stop2} - f_{stop1}}$$

Debe satisfacer la simetría en esta ecuación

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si no satisface la ecuación se debe incrementar w_{pass1} o decrementar w_{pass2} para satisfacer la ecuación de arriba.

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rS})}$$

$$S_S = \frac{BW \cdot s}{s^2 + w_0^{\ 2}}$$

$$w_0 = \sqrt{w_{pass1} \cdot w_{pass2}}$$
$$BW = w_{pass2} - w_{pass1}$$
$$w = f \cdot 2\pi\, rad/seg$$

Ejercicio 2.3.6

Determinar la función de transferencia para un filtro Chebyshev elimina-banda, con las siguientes especificaciones:

$$a_{pass} = -1dB\,,\, a_{stop} = -35dB$$
$$w_{pass}1 = 3000\, rad/seg\,,\, w_{stop1} = 6000\, rad/seg$$
$$w_{pass2} = 24000\, rad/seg\,,\, w_{stop2} = 12000\, rad/seg$$

Script Matlab:

```
%filtro Chebyshev STOPBAND
%ingresa especificaciones
disp('******************************');
disp('      filtro Chebyshev SB');
disp('******************************');
clear all
a_pass = -1;%dB
a_stop = -35;
w_pass1 = 3000;%rad/seg
w_stop1 = 6000;
w_pass2 = 24000;
w_stop2 = 12000;
%Verficamos simetría
s1=w_pass1/w_stop1;
s2=w_stop2/w_pass2;
if s1==s2
    disp('Si simetria');
else
```

```
    disp('No simetría, realice ajuste');
end
```

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

$$\frac{1}{2} = \frac{1}{2}$$

```
%Calcula Omega
Omega = (w_pass2-w_pass1) / (w_stop2-w_stop1)
```

$$\Omega_{rS} = \frac{w_{pass2} - w_{pass1}}{w_{stop2} - w_{stop1}} = 3.5$$

```
%Calcula orden
nc = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(Omega))
%redondea al entero mas cercano
n = round(nc)
```

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rS})} = 2.8044 = 3$$

```
%revisa si n es par o
impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula wo
wo = sqrt(w_pass1*w_pass2)
```

$$w_0 = \sqrt{w_{pass1} \cdot w_{pass2}} = 8485.3 \, rad/sg$$

```
%calcula BW
BW = w_pass2 - w_pass1
```

$$BW = w_{pass2} - w_{pass1} = 21000$$

```
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
```

$$\varepsilon = \sqrt{10^{-0.1\cdot a_{pass}} - 1} = 0.5088$$

```
%calcula variables
D = asinh(E^(-1))/n
```

```
G = 10^(0.05*a_pass)
sinh_D = sinh(D)
```

$$D = \frac{\sinh^{-1}(\varepsilon^{-1})}{n} = 0.4760$$

$$G = 10^{0.05 \cdot a_{pass}} = 0.8913$$

$$\sinh(D) = 0.4942$$

```
%calcula angulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
phi
```

(n impar) $\phi_m = \frac{\pi \cdot (2 \cdot m + 1)}{2 \cdot n}$ *m=0,1,…,(n-1)/2-1*

$$\phi_0 = \frac{\pi}{6}$$

```
%calcula polos
for i=1:(n-m)/2
    polos(i) = -sinh(D)*sin(phi_s(i)) + j*cosh(D)*cos(phi_s(i));
    %Calculo independiente de sigma y omega
    sigma(i)= -sinh(D)*sin(phi_s(i));
    omega(i)= cosh(D)*cos(phi_s(i));
end
%si n impar hay un polo real
if m == 1
   polo_real = -sinh(D)
end
polos
disp('Representacion independiente');
sigma
omega
```

$$S = \sigma \pm j\omega$$

$$\sigma_m = -\sinh(D) \cdot \sin(\phi_m)$$

$$\omega_m = \cosh(D) \cdot \cos(\phi_m)$$

$$\text{Polo real } \sigma_r = -0.4942$$

$$\sigma_0 = -0.2471 \qquad \omega_0 = 0.9660$$

```
%calculo coeficientes función transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
end
B1m
B2m
```

$$B_{1m} = -2 \cdot \sigma_m$$

$$B_{2m} = \sigma_m^{\,2} + \omega_m^{\,2}$$

$$B_{10} = 0.4942 \qquad B_{20} = 0.9942$$

Función transferencia normalizada

(n impar)

$$H_{C,n}(S) = \frac{\sinh(D)\cdot\prod_{m}(B_{2m})}{(S+\sinh(D))\cdot\prod_{m}(S^2+B_{1m}\cdot S+B_{2m})}$$

m=0,1,…,(n-1)/2-1

$$H_{C,3}(S) = \frac{0.4942}{(S+0.4942)}\frac{0.9942}{(S^2+0.4942\cdot S+0.9942)}$$

```
Roots([1 0.4942 0.9942])  %Hallamos raíces
```

Factorizamos

$$H_{C,3}(S) = \underset{\textbf{(1)}}{\frac{0.4942}{(S+0.4942)}}\underset{\textbf{(2)}}{\frac{0.9942}{(S+0.2471\pm 0.9660\cdot j)^2}}$$

Sustituimos

$$S_S = \frac{BW\cdot s}{s^2+{w_0}^2}$$

(1)

$$\frac{0.4942}{(S+0.4942)} = \frac{0.4942}{\left(\left(\frac{BW\cdot s}{s^2+{w_0}^2}\right)+0.4942\right)} = \frac{0.4942\cdot\left(s^2+{w_0}^2\right)}{\left(BW\cdot s+0.4942\cdot\left(s^2+{w_0}^2\right)\right)}$$

$$= \frac{0.4942\cdot\left(s^2+{w_0}^2\right)}{\left(21000\cdot s+0.4942\cdot\left(s^2+{w_0}^2\right)\right)} = \frac{\left(s^2+7.20\cdot 10^7\right)}{\left(s^2+42493\cdot s+7.20\cdot 10^7\right)}$$

(2)

$$\frac{0.9942}{(S+0.2471\pm 0.9660\cdot j)^2} = \frac{0.9942}{\left(\left(\frac{BW\cdot s}{s^2+{w_0}^2}\right)+0.2471+0.9660\cdot j\right)\cdot\left(\left(\frac{BW\cdot s}{s^2+{w_0}^2}\right)+0.2471-0.9660\cdot j\right)}$$

$$= \frac{0.9942 \cdot \left(s^2 + w_0{}^2\right)^2}{\left(BW \cdot s + 0.2471 \cdot \left(s^2 + w_0{}^2\right) + 0.9660 \cdot j \cdot \left(s^2 + w_0{}^2\right)\right) \cdot \left(BW \cdot s + 0.2471 \cdot \left(s^2 + w_0{}^2\right) - 0.9660 \cdot j \cdot \left(s^2 + w_0{}^2\right)\right)} =$$

$$= \frac{0.9942 \cdot \left(s^2 + w_0{}^2\right)^2}{\left(\left(s^2 + w_0{}^2\right) \cdot \left(0.2471 + 0.9660 \cdot j\right) + BW \cdot s\right) \cdot \left(\left(s^2 + w_0{}^2\right) \cdot \left(0.2471 - 0.9660 \cdot j\right) + BW \cdot s\right)} =$$

$$= \frac{0.9942}{0.9942} \cdot \frac{\left(s^2 + w_0{}^2\right)^2}{\left(s^2 + w_0{}^2 \cdot + \frac{BW \cdot s}{\left(0.2471 + 0.9660 \cdot j\right)}\right) \cdot \left(s^2 + w_0{}^2 \cdot + \frac{BW \cdot s}{\left(0.2471 - 0.9660 \cdot j\right)}\right)} =$$

$$= \frac{\left(s^2 + w_0{}^2\right)^2}{\left(s^2 + 7.20 \cdot 10^7 \cdot + 5219.3 \cdot s + 20404 \cdot j \cdot s\right) \cdot \left(s^2 + 7.20 \cdot 10^7 \cdot + 5219.3 \cdot s - 20404 \cdot j \cdot s\right)} =$$

$$= \frac{\left(s^2 + w_0{}^2\right)^2}{\left(s^2 + \left(5219.3 + 20404 \cdot j\right) \cdot s + 7.20 \cdot 10^7\right) \cdot \left(s^2 + \left(5219.3 - 20404 \cdot j\right) \cdot s + 7.20 \cdot 10^7\right)}$$

Hallamos las raíces, para factorizar usando el comando *roots( )*

$$= \frac{\left(s^2 + w_0{}^2\right)^2}{\left(s + 4632 \pm 23369 \cdot j\right)^2 \cdot \left(s + 588 \pm 2965 \cdot j\right)^2}$$

(a) **(b)**

Desarrollamos los polinomio **(a)** y **(b)** en matlab así:

```
expand((sym('s')+4632-23369*j)*(sym('s')+4632+23369*j))

ans =
s^2+9264*s+567565585

expand((sym('s')+588-2965*j)*(sym('s')+588+2965*j))

ans =
s^2+1176*s+9136969
```

$$= \frac{\left(s^2 + w_0{}^2\right)^2}{\left(s^2 + 9264 \cdot s + 567565585\right) \cdot \left(s^2 + 1176 \cdot s + 9136969\right)}$$

Usando los resultados (1) y (2) para obtener la función de transferencia de un filtro Chebyshev elimina-banda.

$$H_{C,6}(s) = \frac{\left(s^2 + 7.20 \cdot 10^7\right)}{\left(s^2 + 42493 \cdot s + 7.20 \cdot 10^7\right)} \cdot \frac{\left(s^2 + 7.20 \cdot 10^7\right)^2}{\left(s^2 + 9264 \cdot s + 567565585\right) \cdot \left(s^2 + 1176 \cdot s + 9136969\right)}$$

$$H_{C,6}(s) = \frac{\left(s^2 + 7.20 \cdot 10^7\right)^3}{\left(s^2 + 42493 \cdot s + 7.20 \cdot 10^7\right) \cdot \left(s^2 + 9264 \cdot s + 567565585\right) \cdot \left(s^2 + 1176 \cdot s + 9136969\right)}$$

Inverso de Chebyshev

Aproximante Inverso de Chebyshev normalizado

Llamada también Chebyshev Tipo II, es una aproximación racional con polos y ceros en la función de transferencia. Esta aproximación tiene una respuesta en la banda de paso muy uniforme y plana como la aproximación de Butterworth. Tiene ripple en la banda de atenuación causada por los ceros de la función de transferencia.
El aproximante Inverso de Chebyshev tiene mejores características de transición que un Butterworth y mejor respuesta de fase que un Chebyshev, esta aproximación tiene características muy buenas pero es más trabajosa al momento de diseñar.

Magnitud

$$\left|H_{I,n}\left[j(w/w_0)\right]\right| = \frac{\sqrt{\varepsilon_i{}^2 \cdot C_n{}^2 \cdot (w_0 / w)}}{\sqrt{1 + \varepsilon_i{}^2 \cdot C_n{}^2 \cdot (w_0 / w)}}$$

w_0 Es la frecuencia pasa-banda

C_n Es el polinomio de Chebyshev

n Es el orden de la función de aproximación

ε_i Es el factor de ajuste de la ganancia pasa-banda

I Indica un filtro Inverso de Chebyshev

Ganancia

$$\varepsilon_i = \sqrt{10^{-0.1 \cdot a_{stop}} - 1}$$

Orden

El orden de un filtro Inverso de Chebyshev se calcula de la misma manera como de un filtro Chebyshev se tratase:

$$n_I = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(w_{stop} / w_{pass})}$$

Función de transferencia

(n par) $$H_{I,n}(S) = \frac{\prod_m (B_{2m}) \cdot \prod_m (S^2 + A_{1m} \cdot S + A_{2m})}{\prod_m (A_{2m}) \cdot \prod_m (S^2 + B_{1m} \cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n/2)-1$$

(n impar) $$H_{I,n}(S) = \frac{[\sinh(D_i)]^{-1}}{\left(S + [\sinh(D_i)]^{-1}\right)} \cdot \frac{\prod_m (B_{2m}) \cdot \prod_m (S^2 + A_{1m} \cdot S + A_{2m})}{\prod_m (A_{2m}) \cdot \prod_m (S^2 + B_{1m} \cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n-1)/2-1$$

Variables

$$D_i = \frac{\sinh^{-1}(\varepsilon_i^{-1})}{n}$$

Ángulos

(n par) $$\phi_m = \frac{\pi \cdot (2 \cdot m + 1)}{2 \cdot n} \qquad m=0,1,\ldots,(n/2)-1$$

(n impar) $$\phi_m = \frac{\pi \cdot (2 \cdot m + 1)}{2 \cdot n} \qquad m=0,1,\ldots,(n-1)/2-1$$

Polos

$$\sigma'_m = -\sinh(D_i) \cdot \sin(\phi_m)$$

$$\omega'_m = \cosh(D_i) \cdot \cos(\phi_m)$$

$$S = \sigma \pm j\omega$$

$$\sigma_m = \frac{\sigma'_m}{\sigma'^2_m + \omega'^2_m}$$

$$\omega_m = \frac{-\omega'_m}{\sigma'^2_m + \omega'^2_m}$$

Si ***n impar*** habrá un polo real en

$$\sigma_r = -[\sinh(D_i)]^{-1}$$

Ceros

$$\sigma_{Zm} = 0.0$$

$$\omega_{Zm} = \sec(\phi_m)$$

Coeficientes de la función de transferencia

$$B_{1m} = -2 \cdot \sigma_m \qquad B_{2m} = {\sigma_m}^2 + {\omega_m}^2$$

$$A_{1m} = -2 \cdot \sigma_{Zm} = 0.0 \qquad A_{2m} = {\omega_{Zm}}^2$$

Ejercicio 2.3.7

Determinar el orden, los coeficientes y la función de transferencia para un filtro Inverso de Chebyshev, con las siguientes especificaciones:

$$a_{pass} = -1dB\,, a_{stop} = -22dB\,, w_{pass} = 1 {rad}/{seg}\,, w_{stop} = 2 {rad}/{seg}$$

Script Matlab:

```
%filtro Inverso Chebyshev normalizado
%ingresa especificaciones
disp('************************************');
disp('Calculos para filtro Inverso Chebyshev');
disp('************************************');
clear all
a_pass = -1;%dB
a_stop = -22;
w_pass = 1;%rad/seg
w_stop = 2;
%Calcula orden
ni = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(w_stop/w_pass))
%redondea al entero mas cercano
n = round(ni)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula ganancia
Ei =1/(sqrt(10^(-0.1*a_stop)-1))
%calcula variables
Di = asinh(Ei^(-1))/n
sinh_Dii = (sinh(Di))^(-1)
%calcula ángulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
phi
%calcula polos
for i=1:(n-m)/2
    sigmap(i)= -sinh(Di)*sin(phi_s(i));
    omegap(i)= cosh(Di)*cos(phi_s(i));
    sigma(i)= sigmap(i)/((sigmap(i))^2+(omegap(i))^2);
```

```
    omega(i)= -omegap(i)/((sigmap(i))^2+(omegap(i))^2);
end
%si n impar hay un polo real
if m == 1
   polo_real = -(sinh(Di))^(-1)
end
disp('Polos');
sigmap
sigma
omegap
omega
%calcula ceros
for i=1:(n-m)/2
    sigmaz(i)=0.0;
    omegaz(i)= sec(phi_s(i));
end
disp('Ceros');
sigmaz
omegaz
%calculo coeficientes función transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
    A1m(i) = 0.0;
    A2m(i) = sigmaz(i)^2 + omegaz(i)^2;
End
```

(n impar)

$$H_{I,n}(S) = \frac{\left[\sinh(D_i)\right]^{-1}}{\left(S + \left[\sinh(D_i)\right]^{-1}\right)} \cdot \frac{\prod_m (B_{2m}) \cdot \prod_m (S^2 + A_{1m} \cdot S + A_{2m})}{\prod_m (A_{2m}) \cdot \prod_m (S^2 + B_{1m} \cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n-1)/2-1$$

$$H_{I,3}(S) = \frac{0.7728}{(S + 0.7728)} \cdot \frac{0.4125}{1.3333} \cdot \frac{\left(S^2 + 0 \cdot S + 1.333\right)}{\left(S^2 + 0.5337 \cdot S + 0.4125\right)}$$

Esta función de transferencia esta normalizada a $w_{stop} = 1 \, rad/seg$ en lugar de $w_{pass} = 1 \, rad/seg$, esto significa que w_{pass} debería ser igual a $w_{pass} = 0.5 \, rad/seg$.

Para corregir este **problema** debemos cambiar $S = \frac{S}{2}$.

$$H_{I,3}(S) = \frac{0.7728}{(S + 1.5456)} \cdot 0.3094 \cdot \frac{\left(S^2 + 5.3333\right)}{\left(S^2 + 1.0674 \cdot S + 1.6500\right)}$$

Filtros LP, HP, BP, SB

Desnormalización LP del aproximante Inverso de Chebyshev

Usada para generar un filtro pasa-bajas.

$$\Omega_{rL} = \frac{w_{stop}}{w_{pass}} = \frac{f_{stop}}{f_{pass}}$$

Redefiniendo términos tenemos:

$$n_I = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rL})}$$

$$S_L = \frac{s}{w_0}$$

$$w_0 = w_{stop}$$

$$w = f \cdot 2\pi\, rad/seg$$

Ejercicio 2.3.8

Determinar la función de transferencia para un filtro Inverso de Chebyshev pasa-bajas (*LP*), con las siguientes especificaciones:

$$a_{pass} = -0.25dB,\ a_{stop} = -38dB,\ w_{pass} = 600\, rad/seg,\ w_{stop} = 1000\, rad/seg$$

Script Matlab:

```
%ingresa especificaciones
disp('****************************************');
disp('Calculos para filtro Inverso Chebyshev LP');
disp('****************************************');
clear all
a_pass = -0.25;%dB
a_stop = -38;
w_pass = 600;%rad/seg
w_stop = 1000;
%calcula Omega
Omega = w_stop / w_pass
%Calcula orden
ni = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(Omega))
%redondea al entero mas cercano
n = round(ni)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
```

```
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula ganancia
Ei =1/(sqrt(10^(-0.1*a_stop)-1))
%calcula variables
Di = asinh(Ei^(-1))/n
sinh_Dii = (sinh(Di))^(-1)
%calcula angulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
phi
%calcula polos
for i=1:(n-m)/2
    sigmap(i)= -sinh(Di)*sin(phi_s(i));
    omegap(i)= cosh(Di)*cos(phi_s(i));
    sigma(i)= sigmap(i)/((sigmap(i))^2+(omegap(i))^2);
    omega(i)= -omegap(i)/((sigmap(i))^2+(omegap(i))^2);
end
%si n impar hay un polo real
if m == 1
   polo_real = -(sinh(Di))^(-1)
end
disp('Polos');
sigmap
sigma
omegap
omega
%calcula ceros
for i=1:(n-m)/2
    sigmaz(i)=0.0;
    omegaz(i)= sec(phi_s(i));
end
disp('Ceros');
sigmaz
omegaz
%calculo coeficientes funcion transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
    A1m(i) = 0.0;
    A2m(i) = sigmaz(i)^2 + omegaz(i)^2;
end
B1m
B2m
A1m
A2m
```

(n par) $$H_{I,n}(S) = \frac{\prod_{m}(B_{2m}) \cdot \prod_{m}(S^2 + A_{1m} \cdot S + A_{2m})}{\prod_{m}(A_{2m}) \cdot \prod_{m}(S^2 + B_{1m} \cdot S + B_{2m})}$$

m=0,1,…,(n/2)-1

$$H_{I,6}(S) = \frac{0.5455 \cdot (S^2 + 1.0718)}{1.0718 \cdot (S^2 + 0.2679 \cdot S + 0.5455)} \cdot \frac{0.7142 \cdot (S^2 + 2)}{2 \cdot (S^2 + 0.9583 \cdot S + 0.7142)} \cdot \frac{1.0340 \cdot (S^2 + 1.0718)}{14.9282 \cdot (S^2 + 1.8952 \cdot S + 1.0340)}$$

$$H_{I,6}(S) = \frac{0.0126 \cdot (S^2 + 1.0718)}{(S^2 + 0.2679 \cdot S + 0.5455)} \cdot \frac{(S^2 + 2)}{(S^2 + 0.9583 \cdot S + 0.7142)} \cdot \frac{(S^2 + 1.0718)}{(S^2 + 1.8952 \cdot S + 1.0340)}$$

Desnormalizando

$$H_{I,6}(S) = \frac{0.0126 \cdot (S^2 + 1.0718 \cdot 10^6)}{(S^2 + 267.9 \cdot S + 545.5 \cdot 10^3)} \cdot \frac{(S^2 + 2 \cdot 10^6)}{(S^2 + 958.3 \cdot S + 714.2 \cdot 10^3)} \cdot \frac{(S^2 + 1.0718 \cdot 10^6)}{(S^2 + 1895.2 \cdot S + 1.0340 \cdot 10^6)}$$

Desnormalización HP del aproximante Inverso de Chebyshev

Usada para generar un filtro pasa-altas.

$$\Omega_{rH} = \frac{w_{pass}}{w_{stop}} = \frac{f_{pass}}{f_{stop}}$$

$$n_I = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rH})}$$

$$S_H = \frac{w_0}{s}$$

$$w_0 = w_{stop}$$

$$w = f \cdot 2\pi \, rad/seg$$

Desnormalización BP del aproximante Inverso de Chebyshev

Usada para generar una función de aproximación para un filtro pasa-banda.

$$\Omega_{rP} = \frac{w_{stop2} - w_{stop1}}{w_{pass2} - w_{pass1}} = \frac{f_{stop2} - f_{stop1}}{f_{pass2} - f_{pass1}}$$

$$n_I = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rP})}$$

$$S_P = \frac{s^2 + {w_0}^2}{BW \cdot s}$$

$$w_0 = \sqrt{w_{stop1} \cdot w_{stop2}}$$
$$BW = w_{stop2} - w_{stop1}$$
$$w = f \cdot 2\pi \, rad/seg$$

Las frecuencias de la banda de atenuación y la banda de paso deben ser simétricas a ambos lados de w_0, para comprobar la simetría se usa esta ecuación:

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si la ecuación no satisface, el lado que sea mayor debe ser reducido, para esto debe ir incrementando w_{stop1} o decrementando w_{stop2}.

Ejercicio 2.3.9

Determinar la función de transferencia para un filtro Inverso de Chebyshev pasa-banda, con las siguientes especificaciones:

$$a_{pass} = -0.5dB \, , a_{stop} = -33dB$$
$$f_{pass1} = 100 \, rad/seg \, , f_{stop1} = 50 \, rad/seg$$
$$f_{pass2} = 200 \, rad/seg \, , f_{stop2} = 400 \, rad/seg$$

Script Matlab:

```
disp('****************************************');
disp('Calculos para filtro Inverso Chebyshev PB');
disp('****************************************');
%ingresa especificaciones
clear all
a_pass = -0.5;%dB
a_stop = -33;
w_pass1 = 2*pi*100;%rad/seg
w_stop1 = 2*pi*50;
w_pass2 = 2*pi*200;
w_stop2 = 2*pi*400;
%Verficamos simetría
s1=w_pass1/w_stop1;
s2=w_stop2/w_pass2;
if s1==s2
    disp('Si simetria');
else
    disp('No simetria');
end
%Calcula Omega
Omega = (w_stop2-w_stop1) / (w_pass2-w_pass1)
%Calcula orden
ni = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(Omega))
%redondea al entero mas cercano
n = round(ni)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
```

```
else
    disp('n impar');
    m = 1;
end
%calcula wo
wo = sqrt(w_stop1*w_stop2)
%calcula BW
BW = w_stop2 - w_stop1
%calcula ganancia
Ei =1/(sqrt(10^(-0.1*a_stop)-1))
%calcula variables
Di = asinh(Ei^(-1))/n
sinh_Dii = (sinh(Di))^(-1)
%calcula ángulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
phi
%calcula polos
for i=1:(n-m)/2
    sigmap(i)= -sinh(Di)*sin(phi_s(i));
    omegap(i)= cosh(Di)*cos(phi_s(i));
    sigma(i)= sigmap(i)/((sigmap(i))^2+(omegap(i))^2);
    omega(i)= -omegap(i)/((sigmap(i))^2+(omegap(i))^2);
end
%si n impar hay un polo real
if m == 1
   polo_real = -(sinh(Di))^(-1)
end
disp('Polos');
sigmap
sigma
omegap
omega
%calcula ceros
for i=1:(n-m)/2
    sigmaz(i)=0.0;
    omegaz(i)= sec(phi_s(i));
end
disp('Ceros');
sigmaz
omegaz
%calculo coeficientes funcion transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
    A1m(i) = 0.0;
    A2m(i) = sigmaz(i)^2 + omegaz(i)^2;
end
B1m
B2m
A1m
A2m
```

(n impar)

$$H_{I,n}(S) = \frac{\left[\sinh(D_i)\right]^{-1}}{\left(S + \left[\sinh(D_i)\right]^{-1}\right)} \cdot \frac{\prod_m (B_{2m}) \cdot \prod_m (S^2 + A_{1m} \cdot S + A_{2m})}{\prod_m (A_{2m}) \cdot \prod_m (S^2 + B_{1m} \cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n-1)/2-1$$

$$H_{I,3}(S) = \frac{0.4710}{(S + 0.4710)} \cdot \frac{0.1902 \cdot (S^2 + 1.3333)}{1.3333 \cdot (S^2 + 0.4038 \cdot S + 0.1902)}$$

$$H_{I,3}(S) = \frac{0.4710 \cdot 0.1427 \cdot (S \pm 1.1547 \cdot j)^2}{(S + 0.4710) \cdot (S + 0.2019 \pm 0.3866 \cdot j)^2}$$

Desnormalizando

$$H_{I,6}(S) = \frac{0.1427 \cdot (1035.8 \cdot S) \cdot (S \pm 2819.4 \cdot j)^2 \cdot (S \pm 280 \cdot j)^2}{(S^2 + 1035.8 \cdot S + 7.8957 \cdot 10^5) \cdot (S + 319.8 \pm 1389.7 \cdot j)^2 \cdot (S + 124.2 \pm 539.6 \cdot j)^2}$$

$$H_{I,6}(S) = \frac{0.1427 \cdot (1035.8 \cdot S) \cdot (S^2 + 7.949 \cdot 10^6) \cdot (S^2 + 78400)}{(S^2 + 1035.8 \cdot S + 7.8957 \cdot 10^5) \cdot (S^2 + 639.6 \cdot S + 2.0335 \cdot 10^6) \cdot (S^2 + 248.4 \cdot S + 3.0659 \cdot 10^5)}$$

Desnormalización SB del aproximante Inverso de Chebyshev

Usada para generar una función de aproximación para un filtro elimina-banda.

$$\Omega_{rS} = \frac{w_{pass2} - w_{pass1}}{w_{stop2} - w_{stop1}} = \frac{f_{pass2} - f_{pass1}}{f_{stop2} - f_{stop1}}$$

Debe satisfacer la simetría en esta ecuación

$$\frac{w_{pass1}}{w_{stop1}} = \frac{w_{stop2}}{w_{pass2}}$$

Si no satisface la ecuación se debe incrementar w_{pass1} o decrementar w_{pass2} para satisfacer la ecuación de arriba.

$$n_C = \frac{\cosh^{-1}\left[\sqrt{(10^{-0.1 \cdot a_{stop}} - 1)/(10^{-0.1 \cdot a_{pass}} - 1)}\right]}{\cosh^{-1}(\Omega_{rS})}$$

$$S_S = \frac{BW \cdot s}{s^2 + {w_0}^2}$$

$$w_0 = \sqrt{w_{stop1} \cdot w_{stop2}}$$

$$BW = w_{stop2} - w_{stop1}$$

$$w = f \cdot 2\pi \, rad/seg$$

2.3.2 Métodos de transformación

Hay diferentes métodos de transformación de las características de un filtro analógico a un digital, no hay un equivalente perfecto digital para un filtro analógico en todas las frecuencias, pero se pueden combinar las características importantes del filtro. Hay varios métodos de transformación: respuesta al impulso invariante, respuesta al escalón invariante y la transformación bilineal.

Método de la respuesta al impulso invariante

Este método de transformación esta basado en crear un filtro digital con una respuesta al impulso que es una versión muestreada de la respuesta al impulso de un filtro analógico.

Primero se comienza con una función de transferencia de un filtro analógico $H(s)$ y usando la transformada inversa de Laplace se determina la respuesta al impulso de un sistema continuo $h(t)$, a continuación se muestrea esa respuesta para determinar la respuesta al impulso de un sistema discreto $h(n \cdot T)$, entonces se toma la transformada-z de $h(n \cdot T)$ y se encuentra la función de transferencia de un sistema discreto $H(z)$.

Por ejemplo:
Se desea convertir la función de transferencia de un sistema continuo a una función de transferencia de un sistema discreto usando el método de la respuesta al impulso invariante.

$$H(s) = \frac{12}{(s+2)\cdot(s+5)}$$

Se debe expandir en fracciones parciales para facilitar el uso de la transformada inversa de Laplace. Matlab posee un comando para expansión en fracciones parciales encontrando los polos, los residuos y los términos.

```
[r,p,k] = residue(num,den)
```

$$\frac{B(s)}{A(s)} = \frac{r(1)}{s+p(1)} + \ldots + \frac{r(n)}{s+p(n)} + k(s)$$

Expandimos las fracciones

```
y=expand((sym('s')+2)*( sym('s')+5));
```

$$H(s) = \frac{12}{s^2 + 7\cdot s + 10}$$

Expandimos en fracciones parciales

```
num=[0 0 12];
den=[1 7 10];
[r,p,k]=residue(num,den)
```

$$H(s)=\frac{4}{s+2}-\frac{4}{s+5}$$

Transformada de Laplace para hallar *h(t)*

```
ilaplace(4/(sym('s')+2))
```

$$\frac{4}{s+2}=4\cdot e^{-2\cdot t}$$

```
ilaplace(-4/(sym('s')+5))
```

$$-\frac{4}{s+5}=-4\cdot e^{-5\cdot t}$$

$$h(t)=(4\cdot e^{-2\cdot t}-4\cdot e^{-5\cdot t})\cdot u(t)$$

Se muestrea a un intervalo T , para hallar la respuesta al impulso del sistema discreto

$$h(nT)=(4\cdot e^{-2\cdot nT}-4\cdot e^{-5\cdot nT})\cdot u(nT)$$

Se usa la trasformada-z

$$H(z)=\frac{4}{1-e^{-2T}\cdot z^{-1}}-\frac{4}{1-e^{-5T}\cdot z^{-1}}$$

$$H(z)=\frac{4\left(e^{-2T}-e^{-5T}\right)\cdot z^{-1}}{\left(1-e^{-2T}\cdot z^{-1}\right)\cdot\left(1-e^{-5T}\cdot z^{-1}\right)}$$

Ejercicio 2.3.10

Determinar el impulso invariante de un filtro digital basado en la aproximación de Butterworth de orden 2. Calcular para *T=1* y *T=0.1*

$$H_{B,2}(s)=\frac{1}{s^2+1.4142\cdot s+1}$$

Matlab posee un comando llamado `[numz,denz]=impinvar(num,den,fs)`

Script Matlab:

```
%transformación respuesta al impulso invariante
num = [0 0 1];
den = [1 1.4142 1];
%T=1
[numz,denz]=impinvar(num,den,1/1)
```

$$H(z)=\frac{0.4530\cdot z^{-1}}{1-0.7497\cdot z^{-1}+0.2431\cdot z^{-2}}$$

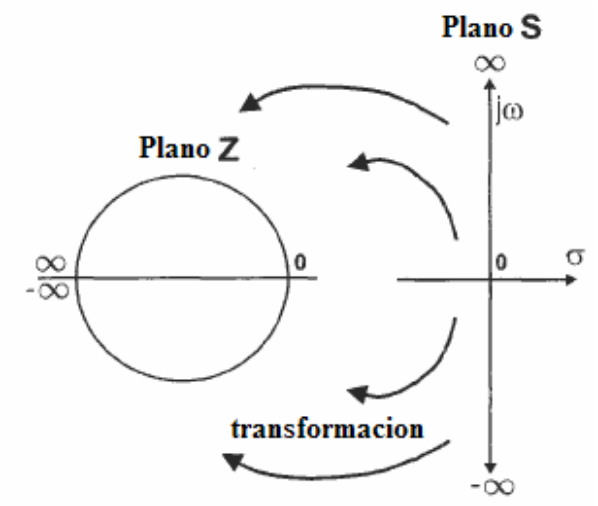
```
%T=0.1
[numz,denz]=impinvar(num,den,1/0.1)
```

$$H(z)=\frac{0.0093\cdot z^{-1}}{1-1.8588\cdot z^{-1}+0.8681\cdot z^{-2}}$$

Método de la transformación bilineal

El método de la transformación bilineal es una técnica popular para el diseño de filtros digitales IIR por ofrecer varias ventajas sobre otras técnicas.
La transformación bilineal es usada para convertir la respuesta a la frecuencia analógica a una respuesta en el dominio digital. La ventaja de la transformación bilineal es que cualquier respuesta (*LP,HP,PB,SB*) puede ser convertida. El dominio digital es también conocido como el dominio – z.

En el gráfico de abajo se puede observar la transformación del *plano s* al *plano z*.



La relación entre S y Z puede ser descrita por:

$$s=\frac{2}{T}\cdot\frac{z-1}{z+1}$$

Donde T es el periodo de muestreo.
Las ecuaciones de abajo son importantes para el trazado entre el dominio analógico y digital (*distorsión de frecuencias*).

$$w=\frac{2}{T}\cdot\tan\left(\frac{\Omega}{2}\right)$$

$$\Omega=\frac{2}{T}\cdot\tan^{-1}\left(\frac{w\cdot T}{2}\right)$$

El trazado de frecuencias analógicas a frecuencias digitales es bastante ***lineal para bajas frecuencias***, pero para ***altas frecuencias no es muy lineal*** y se produce una distorsión de las altas frecuencias.

Por ejemplo:
Se desea diseñar un filtro pasa bajas usando el aproximante de Butterworth, con las siguientes características:

$$a_{pass} = -1dB\,,\, a_{stop} = -20dB\,,\, f_{pass} = 1-khz\,,\, f_{stop} = 5-khz\,,\, f_s = 20-khz$$

El primer paso es calcular las frecuencias digitales (normalizar), el cual puede estar representado en 2 formas:

$$(1)\Omega = 2\cdot\pi\cdot\frac{f_d}{f_s}$$

$$(2)\, f_d = f_s\cdot\frac{\Omega}{2\cdot\pi}$$

Usando la primera forma tendremos:

$$\Omega_{pass} = 2\cdot\pi\cdot\frac{f_d}{f_s} = 2\cdot\pi\cdot\frac{1000}{20000} = 0.1\cdot\pi$$

$$\Omega_{stop} = 2\cdot\pi\cdot\frac{f_d}{f_s} = 2\cdot\pi\cdot\frac{5000}{20000} = 0.5\cdot\pi$$

Hallamos el equivalente de frecuencias analógicas necesarias para diseñar el filtro analógico, usando la ecuación de trazado entre el dominio analógico y digital, este *distorsion*a las frecuencias.

$$w = \frac{2}{T}\cdot\tan\left(\frac{\Omega}{2}\right)$$

$$w_{pass} = \frac{2}{T}\cdot\tan\left(\frac{\Omega_{pass}}{2}\right) = 2\cdot 20000\cdot\cdot\tan\left(\frac{0.1\cdot\pi}{2}\right) = 6335.4\,{}^{rad}\!/_{seg}$$

$$w_{stop} = \frac{2}{T}\cdot\tan\left(\frac{\Omega_{stop}}{2}\right) = 2\cdot 20000\cdot\cdot\tan\left(\frac{0.5\cdot\pi}{2}\right) = 40000\,{}^{rad}\!/_{seg}$$

Abajo se puede observar la distorsión de las altas frecuencias

$\frac{6335.4}{2 \cdot \pi} = 1008 - Hz$ *bastante lineal en bajas frecuencias, ligera distorsión (1000-Hz)*

$\frac{40000}{2 \cdot \pi} = 6366.2 - Hz$ *no lineal en altas frecuencias, distorsión (5000-Hz)*

Ejercicio 2.3.11

Diseñar un filtro digital IIR pasa bajas usando la aproximación de Butterworth, usando la transformación bilineal, con las siguientes características.

$$a_{pass} = -1dB\,,\, a_{stop} = -20dB\,,\, f_{pass} = 1 - khz\,,\, f_{stop} = 5 - khz\,,\, f_s = 20 - khz$$

Normalizamos frecuencias digitales

$$\Omega = 2 \cdot \pi \cdot \frac{f_d}{f_s}$$

$$\Omega_{pass} = 2 \cdot \pi \cdot \frac{f_d}{f_s} = 2 \cdot \pi \cdot \frac{1000}{20000} = 0.1 \cdot \pi$$

$$\Omega_{stop} = 2 \cdot \pi \cdot \frac{f_d}{f_s} = 2 \cdot \pi \cdot \frac{5000}{20000} = 0.5 \cdot \pi$$

Distorsionamos frecuencias digitales para hallar equivalente analógico

$$w = \frac{2}{T} \cdot \tan\left(\frac{\Omega}{2}\right)$$

$$w_{pass} = \frac{2}{T} \cdot \tan\left(\frac{\Omega_{pass}}{2}\right) = 2 \cdot 20000 \cdot \cdot \tan\left(\frac{0.1 \cdot \pi}{2}\right) = 6335.4\, rad/seg$$

$$w_{stop} = \frac{2}{T} \cdot \tan\left(\frac{\Omega_{stop}}{2}\right) = 2 \cdot 20000 \cdot \cdot \tan\left(\frac{0.5 \cdot \pi}{2}\right) = 40000\, rad/seg$$

Desarrollamos la función de aproximación Butterworth

Script Matlab:

```
disp('*******************************');
disp('Calculos para filtro Butterworth');
disp('*******************************');
%ingresa especificaciones
clear all
a_pass = -1;%dB
a_stop = -20;
w_pass = 6335.4;%rad/seg
w_stop = 40000;
%calcula omega
Omega=w_stop/w_pass
%Calcula orden
nb = (log((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))/...
(2*log(Omega))
%redondea al entero mas cercano
n = round(nb)
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula radio
R = E^(-1/n)
%calcula ángulos
for i=1:(n-m)/2
theta(i)=(sym('pi') *(2*(i-1) + n + 1))/(2*n);
tt(i)=(pi*(2*(i-1) + n + 1))/(2*n);
end
theta
%calcula los polos
for i=1:(n-m)/2
polos(i)=R*cos(tt(i))+j*R*sin(tt(i));
end
polos
%calcula coeficientes
for i=1:(n-m)/2
B1m(i)=-2*R*cos(tt(i));
B2m(i)=(R*cos(tt(i))).^2+(R*sin(tt(i))).^2;
end
B1m
B2m
```

(n par) $$H_{B,n}(S) = \frac{\prod_{m}(B_{2m})}{\prod_{m}(S^2 + B_{1m}\cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n/2)-1$$

$$H_{B,2}(S) = \frac{1.9652}{\left(S^2 + 1.9825\cdot S + 1.9652\right)}$$

Desnormalizamos la aproximación a LP

$$S_L = \frac{s}{w_0} \qquad w_o = w_{pass}$$

$$H_{B,2}(s)=\frac{1.9652}{\left(\left(\frac{s}{w_0}\right)^2+1.9825\cdot\left(\frac{s}{w_0}\right)+1.9652\right)}$$

$$H_{B,2}(s)=\frac{7.8879\cdot 10^7}{\left(s^2+1.256\cdot 10^4\cdot s+7.8879\cdot 10^7\right)}$$

Usamos la transformación bilineal

$$s=\frac{2}{T}\cdot\frac{z-1}{z+1}$$

$$H(z)=\frac{7.8879\cdot 10^7}{\left(\left(2\cdot f_s\cdot\frac{z-1}{z+1}\right)^2+1.256\cdot 10^4\cdot\left(2\cdot f_s\cdot\frac{z-1}{z+1}\right)+7.8879\cdot 10^7\right)}$$

$$H(z)=\frac{7.8879\cdot 10^7\cdot\left(z^2+2\cdot z+1\right)}{\left(1.60\cdot 10^9\cdot\left(z^2-2\cdot z+1\right)+502400000\cdot\left(z^2-1\right)+7.8879\cdot 10^7\cdot\left(z^2+2\cdot z+1\right)\right)}$$

$$H(z)=\frac{7.8879\cdot 10^7\cdot\left(z^2+2\cdot z+1\right)}{2.1813\cdot 10^9\cdot z^2-3.0422\cdot 10^9\cdot z+1.1765\cdot 10^9}$$

$$H(z)=\frac{0.0362\cdot\left(z^2+2\cdot z+1\right)}{\cdot z^2-1.3947\cdot z+0.5394}$$

$$H(z)=\frac{0.0362\cdot\left(1+2\cdot z^{-1}+z^{-2}\right)}{\cdot 1-1.3947\cdot z^{-1}+0.5394\cdot z^{-2}}$$

Matlab tiene un comando para hallar la función de transferencia de un sistema digital con el método bilineal *[numd,dend]=bilinear(num,den,fs)*

```
%transformación bilineal
num = [0 0 7.8879*10^7];
den = [1 1.256*10^4 7.8879*10^7];
[numd,dend]=bilinear(num,den,20000)
```

$$H(z)=\frac{0.0362+0.0723\cdot z^{-1}+0.0362\cdot z^{-2}}{\cdot 1-1.3947\cdot z^{-1}+0.5394\cdot z^{-2}}$$

La respuesta a la frecuencia del filtro digital IIR puede ser determinada así:

$$H(e^{j\Omega})=\frac{0.0362\cdot\left(1+2\cdot e^{-j\Omega}+e^{-2\cdot j\Omega}\right)}{\cdot 1-1.3947\cdot e^{-j\Omega}+0.5394\cdot e^{-2\cdot j\Omega}}$$

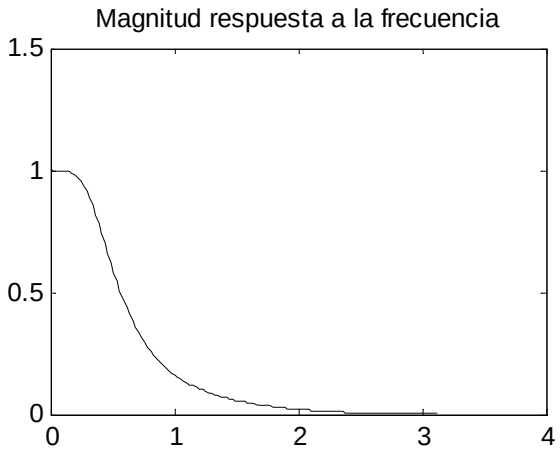
Usando la relación de Euler podemos reescribir la ecuación de arriba así:

$$H(e^{j\Omega}) = \frac{0.0362 \cdot (1 + 2 \cdot \cos(\Omega) + \cos(2 \cdot \Omega) - j \cdot 2 \cdot \sin(\Omega) - j \cdot \sin(2 \cdot \Omega))}{\cdot 1 - 1.3947 \cdot (\cos(\Omega) - j \cdot \sin(\Omega)) + 0.5394 \cdot (\cos(2 \cdot \Omega) - j \cdot \sin(2 \cdot \Omega))}$$

Dando valores a Ω entre $(0 - \pi)$, hallamos la magnitud y fase de la respuesta a la frecuencia.

En Matlab podemos hallar directamente y graficar así:

```
numd = [0.0362 0.0723 0.0362];
dend = [1 1.3947 0.5394];
[Hz,w]=freqz(numd,dend,128) %evalúa 128 puntos
plot(w,abs(Hz));
title('Magnitud respuesta a la frecuencia');
```



Ejercicio 2.3.12

Diseñar un filtro pasa bajas digital IIR usando el aproximante Chebyshev, con las siguientes características:

$$a_{pass} = -1dB\,,\, a_{stop} = -60dB\,,\, f_{pass} = 10 - khz\,,\, f_{stop} = 20 - khz\,,\, f_s = 50 - khz$$

Script Matlab:

```
clc;%limpia command window
disp('*****************************');
disp('      filtro Chebyshev LP'     );
disp('*****************************');
%ingresa especificaciones
a_pass = -1;%dB
a_stop = -60;
f_pass = 10000;%Hz
f_stop = 20000;
```

```
f_sampler = 50000;%frecuencia muestreo
%Normaliza frecuencias analógicas a digitales
Omega_pass = 2*pi*f_pass/f_sampler;
Omega_stop = 2*pi*f_stop/f_sampler;
%Calcula frecuencias analógicas equivalentes
w_pass = 2*f_sampler*tan(Omega_pass/2);%rad/seg
w_stop = 2*f_sampler*tan(Omega_stop/2);
%Calcula Omega
Omega=w_stop/w_pass;
%calcula Wo
wo = w_pass
%Calcula orden
nc = (acosh(sqrt((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))) /...
(acosh(Omega));
%redondea al entero mas cercano
n = round(nc)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula ganancia
E = sqrt(10^(-0.1*a_pass)-1);
%calcula variables
D = asinh(E^(-1))/n
G = 10^(0.05*a_pass)
sinh_D = sinh(D)
%calcula angulos
for i=1:(n-m)/2
    phi(i) = (sym('pi')*(2*(i-1)+1))/(2*n);
    phi_s(i)=(pi*(2*(i-1)+1))/(2*n);
end
%calcula polos
for i=1:(n-m)/2
    polos(i) = -sinh(D)*sin(phi_s(i)) + j*cosh(D)*cos(phi_s(i));
    %Calculo independiente de sigma y omega
    sigma(i)= -sinh(D)*sin(phi_s(i));
    omega(i)= cosh(D)*cos(phi_s(i));
end
%calculo coeficientes funcion transferencia
for i=1:(n-m)/2
    B1m(i) = -2*sigma(i);
    B2m(i) = sigma(i)^2 + omega(i)^2;
end
B1m
B2m
```

(n par) $$H_{C,n}(S) = \frac{G \cdot \prod_{m} (B_{2m})}{\prod_{m} (S^2 + B_{1m} \cdot S + B_{2m})}$$

m=0,1,…,(n/2)-1

$$H_{C,4}(S) = 0.8913 \cdot \frac{0.9865}{S^2 + 0.2791 \cdot S + 0.9865} \cdot \frac{0.2714}{S^2 + 0.6737 \cdot S + 0.2714}$$

Desnormalizando

$$H_{C,4}(s) = 0.8913 \cdot \frac{0.9865}{s^2 + 0.2791 \cdot s + 0.9865} \cdot \frac{0.2714}{s^2 + 0.6737 \cdot s + 0.2714}$$

$$H_{C,4}(s) = \frac{6.8453 \cdot 10^{18}}{\left(s^2 + 2.0276 \cdot 10^4 \cdot s + 5.2074 \cdot 10^9\right) \cdot \left(s^2 + 4.8950 \cdot 10^4 \cdot s + 1.4748 \cdot 10^9\right)}$$

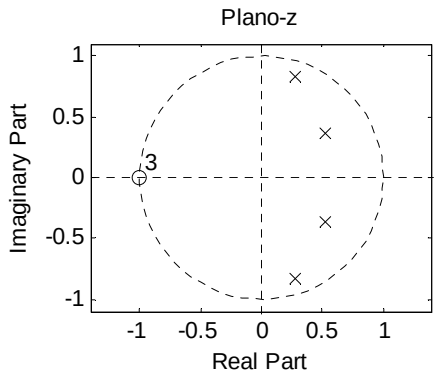
$$H_{C,4}(s) = \frac{6.8453 \cdot 10^{18}}{s^4 + 6.923 \cdot 10^4 \cdot s^3 + 7.675 \cdot 10^9 \cdot s^2 + 2.848 \cdot 10^{14} \cdot s + 7.68 \cdot 10^{18}}$$

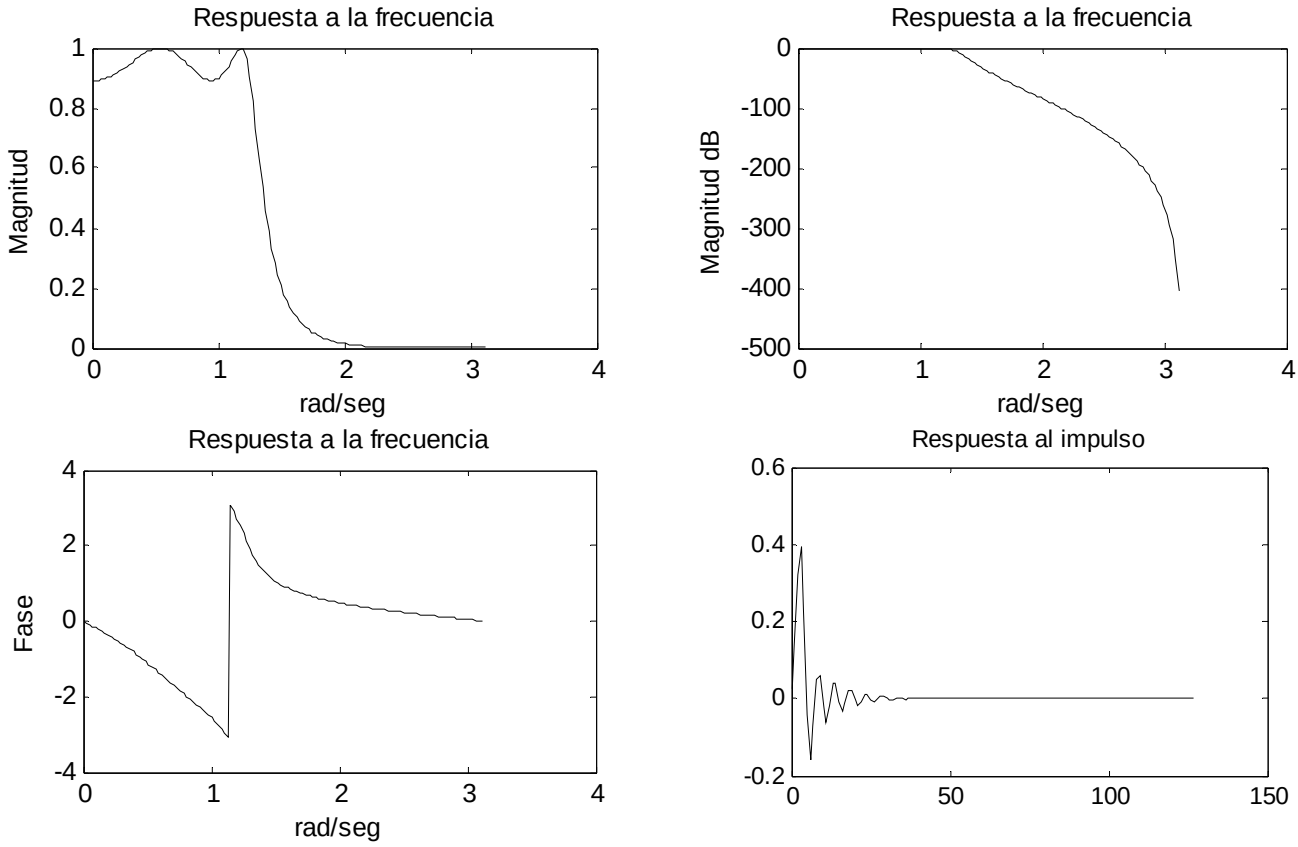
Usando la transformada bilineal

```
%funciones transferencia para n=4
num_1 = [0 0 G];
den_1 = [0 0 1];
a = tf(num_1,den_1);

num_2 = [0 0 B2m(1)*wo^2];
den_2 = [1 B1m(1)*wo B2m(1)*wo^2];
b = tf(num_2,den_2);

num_3 = [0 0 B2m(2)*wo^2];
den_3 = [1 B1m(2)*wo B2m(2)*wo^2];
c = tf(num_3,den_3);
Hs = a*b*c
%transformada bilineal
[num,den] = tfdata(Hs,'v');%toma numerador y denominador
TF
[numd,dend] = bilinear(num,den,f_sampler);
Hz=tf(numd,dend,'variable','z^-1')
```

$$H_{C,4}(z) = \frac{0.02426 + 0.09704 \cdot z^{-1} + 0.1456 \cdot z^{-2} + 0.09704 \cdot z^{-3} + 0.02426 \cdot z^{-4}}{1 - 1.598 \cdot z^{-1} + 1.746 \cdot z^{-2} - 1.02 \cdot z^{-3} + 0.3074 \cdot z^{-4}}$$

```
%grafica respuesta a la frecuencia
[num,den] = tfdata(Hz,'v');
[H,w]=freqz(num,den,128);
plot(w,abs(H))
ylabel('Magnitud');
xlabel('rad/seg')
title('Respuesta a la frecuencia');

figure(5);
plot(w,20*log(abs(H)))
ylabel('Magnitud dB');
xlabel('rad/seg')
title('Respuesta a la frecuencia');
%gráfica fase
figure(2);
plot(w,angle(H))
title('Respuesta a la frecuencia');
ylabel('Fase');
xlabel('rad/seg')
%plano z
figure(3);
zplane(num,den)
title('Plano-z');
%Respuesta al impulso
[h,t]=impz(num,den,128);
figure(4);
plot(t,h);
title('Respuesta al impulso');
```





2.4 Filtros digitales FIR (finite impulse response)

Los filtros digitales con *respuesta finita al impulso* reciben el nombre de filtros *FIR*, este tipo de filtros tienen algunas ventajas sobre los filtros *IIR*, los filtros *FIR* son siempre estables, realizables (*más fáciles de diseñar)* y tienen fase lineal o lo que es igual a un *retraso constante* bajo condiciones especificas, sin embargo la ***mayor*** desventaja de los filtros *FIR* es que el número de coeficientes necesarios para implementar un filtro específico es mucho mayor que en un filtro digital *IIR*.
Un método estándar y popular de diseño de filtros digitales es la técnica de optimización de Parks – Mclellan, otro método muy común de diseño de filtros digitales *FIR* es las series de Fourier usando funciones de ventana (*window*).

La forma general de la función de transferencia de un filtro FIR es

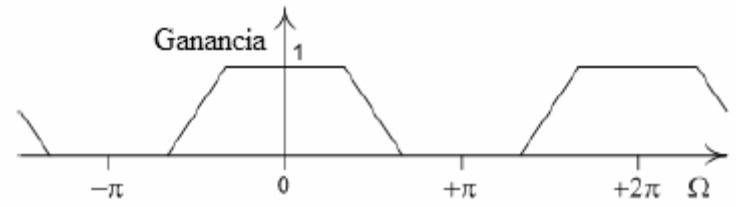
$$H(Z) = \sum_{k=0}^{M} h(k) \cdot z^{-k}$$

2.4.1 Método de las series de Fourier para diseñar filtros FIR

Coeficientes de la respuesta a la frecuencia y respuesta al impulso

En el proceso de diseño de un filtro, se empieza el diseño con las características de respuesta a la frecuencia; la banda de frecuencias criticas y las ganancias en cada banda son determinadas con las especificaciones dadas.
La respuesta a la frecuencia de un filtro digital es periódica en el dominio de la frecuencia con un periodo igual a la frecuencia de muestreo.

En el gráfico de abajo se muestra la naturaleza periódica de la respuesta a la frecuencia.



Puesto que la respuesta es periódica, esta puede ser descrita por las series de Fourier de la forma:

$$H(e^{j\cdot\Omega}) = \sum_{k=-\infty}^{\infty} h(k)\cdot e^{-j\cdot k\cdot\Omega}$$

En donde la frecuencia compleja exponencial puede tomar todos los valores posibles de frecuencia.
Los coeficientes dentro de la sumación son los coeficientes de la respuesta al impulso que describen al filtro digital *FIR*. El procedimiento para determinar los coeficientes de la respuesta al impulso a partir de la respuesta a la frecuencia es directo, el resultado final de la derivación se muestra a continuación:

$$h(n) = \frac{1}{2\cdot\pi}\cdot \int_{\Omega_0-\pi}^{\Omega_0+\pi} H(e^{j\cdot\Omega})\cdot e^{j\cdot k\cdot\Omega}\cdot d\Omega \qquad n=0,\ \pm 1,\ \pm 2,\ldots$$

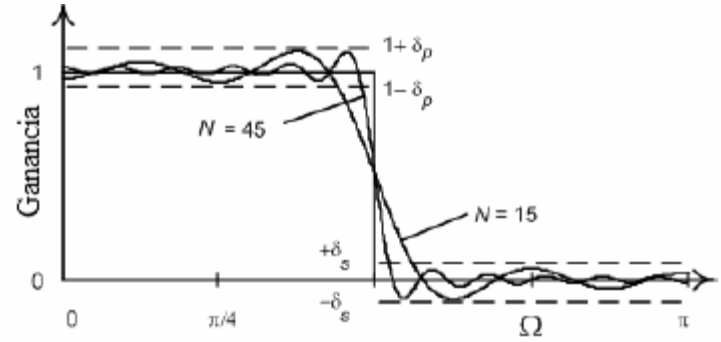
La integración debe incluir solamente un periodo completo de la respuesta a la frecuencia, pero *no es posible* implementar un numero de coeficientes infinitos.
El número de coeficientes que se usen depende de cómo se desee aproximar al ideal y cuantos coeficientes podamos guardar.
Se puede asumir que los índices están limitados en un rango de $-M \le n \le +M$, el cual limita el número de coeficientes usados o guardados a $N = 2\cdot M + 1$. Al hacer esto se esta poniendo a cero (0) los otros coeficientes.

La respuesta a la frecuencia puede ser determinada usando:

$$H(e^{j\cdot\Omega}) = \sum_{n=-M}^{M} h(n)\cdot e^{-j\cdot n\cdot\Omega}$$

La figura de abajo muestra el efecto de limitar el número de coeficientes, graficando la respuesta a la frecuencia usando un número finito de coeficientes, con *n=45* y *n=15*, los errores dentro de la banda de paso y la banda de atenuación están especificados como δ_p y δ_s respectivamente.

La respuesta a la frecuencia puede oscilar dentro de esos *límites de error*.



A medida que se incremente el número de coeficientes en la aproximación del filtro *FIR* se puede observar que el *ripple* se concentra en la banda de paso, este *ripple* **no puede** ser eliminado, incluso aumentando el número de coeficientes este se concentra en la transición, a este efecto se le conoce como ***fenómeno de Gibb's*** y resulta cuando una discontinuidad es modelada con una serie, sin embargo hay métodos que pueden usarse para reducir este efecto.
Se puede trasladar esas especificaciones a decibeles o viceversa usando las ecuaciones de abajo:

$$a_{pass} = 20\cdot\log(1-\delta_p)$$

$$a_{stop} = 20\cdot\log(1-\delta_s)$$

$$\delta_p = 1-10^{0.05\cdot a_{pass}}$$

$$\delta_s = 10^{0.05\cdot a_{stop}}$$

La respuesta de la fase de un filtro *FIR* puede ser una función lineal de la frecuencia bajo ciertas condiciones.

Por ejemplo:
La respuesta a la frecuencia de un filtro *FIR* dentro de la banda de paso es:

$$H_{banda-paso}(e^{j\cdot\Omega}) = 1\cdot e^{-j\cdot n\cdot\Omega} = 1 < -\tau\cdot\Omega$$

Se esta especificando una ganancia unitaria, mientras que el ángulo de fase cambia linealmente con la frecuencia.
Las condiciones que permiten a la fase lineal cambiar son los coeficientes de la respuesta al impulso que pueden ser *simétricos* o *anti-simétricos*.
Son simétricos si satisface:

$$h(n) = h(-n)$$

Y si satisface la ecuación de abajo son anti-simétricos

$$h(n) = -h(-n)$$

Donde $\tau = \frac{N-1}{2}$ N es el número de coeficientes o longitud del filtro.

Orden filtros FIR

Los filtros analógicos y los filtros digitales *IIR* usan el orden de un filtro para medir el ***tamaño*** de un filtro. El orden se refiere al término más alto en la ecuación polinomial usada para describir el filtro.

Los filtros digitales *FIR* generalmente usan el número de los coeficientes de la respuesta al impulso (***kernel***) requeridos para describir el ***tamaño*** del filtro. Esto se debe a que los filtros *FIR* son implementados usando la *convolución* donde el numero de coeficientes (*los coeficientes son substituidos dentro de la ecuación de diferencias*) afecta directamente la longitud del procesamiento.

Para calcular la longitud estimada de un filtro *FIR* podemos usar estas relaciones:

Window (ventana)	***longitud***
`Rectangular`	$N = 1 + (\text{int})\left[0.97\cdot\frac{f_s - (f_{stop} - f_{pass})}{f_{stop} - f_{pass}}\right]$
`Bartlett`	$N = 1 + (\text{int})\left[4.15\cdot\frac{f_s - (f_{stop} - f_{pass})}{f_{stop} - f_{pass}}\right]$
`Von Hann`	$N = 1 + (\text{int})\left[3.33\cdot\frac{f_s - (f_{stop} - f_{pass})}{f_{stop} - f_{pass}}\right]$
`Hamming`	$N = 1 + (\text{int})\left[3.34\cdot\frac{f_s - (f_{stop} - f_{pass})}{f_{stop} - f_{pass}}\right]$
`Blackman`	$N = 1 + (\text{int})\left[6.0\cdot\frac{f_s - (f_{stop} - f_{pass})}{f_{stop} - f_{pass}}\right]$

Características de los filtros FIR

Cuatro tipos de filtros digitales *FIR* pueden ser diseñados, considerando los coeficientes simétricos y anti - simétricos combinados con la longitud del filtro que puede ser par o impar.

Filtros FIR tipo I

Tienen longitud impar y coeficientes simétricos, tienen una respuesta a la frecuencia simétrica par en $\Omega = 0$ y $\Omega = \pi$, esta respuesta simétrica par permite a la frecuencia tomar cualquier valor en esas frecuencias criticas.
Con este tipo de filtros *FIR* es posible implementar filtros *LP*, *HP*, *PB* y *SB.*

Filtros FIR tipo II

Tienen coeficientes simétricos y longitud par, la respuesta a la frecuencia es impar en $\Omega = 0$ y par en $\Omega = \pi$, esta condición hace que la respuesta en $\Omega = \pi$ sea cero.
Este tipo de filtros *FIR* no son recomendados para filtros *HP* y *SB.*

Filtros FIR tipo III

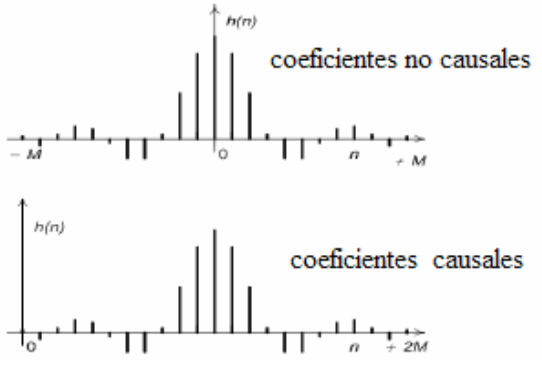
Los coeficientes son anti – simétricos y longitud impar, la respuesta a la frecuencia es impar en $\Omega = 0$ y $\Omega = \pi$, la respuesta a la frecuencia es cero en ambas frecuencias criticas.
Este tipo de filtros *FIR* son usados para filtros *PB.*

Filtros FIR tipo IV

Tienen longitud par y coeficientes anti - simétricos, la respuesta a la frecuencia es impar en $\Omega = 0$ y par en $\Omega = \pi$, la condición de simétrica impar hace que este filtro sea malo para implementar con filtros *LP* y *SB.*
Los coeficientes del filtro derivados de

$$h(n) = \frac{1}{2 \cdot \pi} \cdot \int_{\Omega_0 - \pi}^{\Omega_0 + \pi} H(e^{j \cdot \Omega}) \cdot e^{j \cdot k \cdot \Omega} \cdot d\Omega \qquad n=0, \pm 1, \pm 2, \ldots$$

no producen un filtro *causal*, esto significa que el sistema no puede ser implementado en tiempo real (*Real time*).
El problema puede ser resuelto cambiando todos los valores de los coeficientes a la derecha sobre el eje del tiempo, entonces solo los valores positivos de *n* producirán coeficientes.
En el siguiente gráfico se muestra esta acción de cambio:



La desventaja de esta acción es que incrementa el tiempo de retardo (*delay*) entre la entrada del sistema y la salida por *M* periodos de muestreo.
Los coeficientes causales pueden ser determinados de los coeficientes no causales haciendo ajustes en los índices, los índices de los coeficientes no causales toman valores de $-M$ a $+M$ y los índices de los coeficientes causales toman valores de *0* a *M*.

$$h_{CAUSAL}(n+M) = h_{NOCAUSAl}(n) \qquad n=0, \pm 1, \ldots, \pm M$$

Coeficientes FIR ideales

Se puede calcular los coeficientes ideales para varios tipos de filtros usando la ecuación, los tipos de filtros FIR que se verán serán del **Tipo I**.

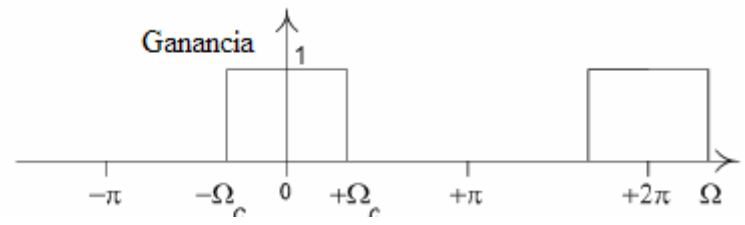
$$h(n) = \frac{1}{2\cdot\pi}\cdot\int_{\Omega_0-\pi}^{\Omega_0+\pi} H(e^{j\cdot\Omega})\cdot e^{j\cdot k\cdot\Omega}\cdot d\Omega \qquad n=0, \pm 1, \pm 2,\ldots$$

En cada caso figura la descripción ideal de filtros LP, HP, PB y SB con ecuaciones por determinar los coeficientes basados sobre los parámetros dados de un filtro particular.
La respuesta a la frecuencia en la banda de paso de cada filtro esta definida por:

$$H_{banda-paso}(e^{j\cdot\Omega}) = 1\cdot e^{-j\cdot n\cdot\Omega} = 1 < -\tau\cdot\Omega$$

Y nos permite calcular los coeficientes causales directamente.

Filtro LP
La figura de abajo muestra las especificaciones para un filtro pasa bajas



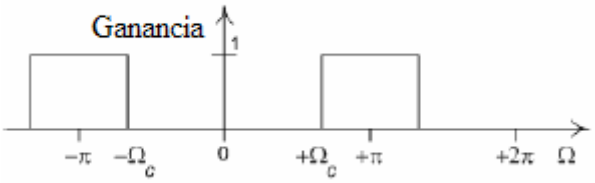
Coeficientes del filtro LP

$$h_{LP}(n) = \frac{1}{2\cdot\pi}\cdot\int_{-\Omega_c}^{+\Omega_c} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega \qquad n=0, 1, \ldots, 2M$$

$$\tau = M \qquad \Omega_C = \frac{w_{stop} + w_{pass}}{2 \cdot fs}$$

$$h_{LP}(n) = \begin{cases} \dfrac{\sin\left[(n-\tau)\cdot\Omega_c\right]}{(n-\tau)\cdot\pi}, n \neq \tau \\ \dfrac{\Omega_c}{\pi}, n = \tau \end{cases} \qquad n=0, 1, \ldots, 2M$$

Filtro HP
La figura de abajo muestra las especificaciones para un filtro pasa altas



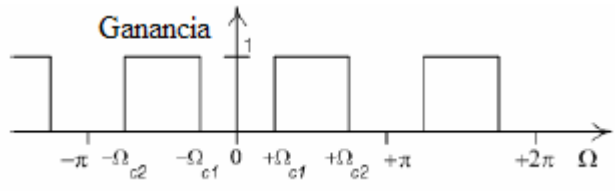
Coeficientes del filtro HP

$$h_{HP}(n) = \frac{1}{2\cdot\pi}\cdot\left[\int_{-\pi}^{-\Omega_c} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega + \int_{+\Omega_c}^{+\pi} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega\right] \qquad n=0, 1, \ldots, 2M$$

$$h_{HP}(n) = \begin{cases} \dfrac{\sin\left[(n-\tau)\cdot\pi\right] - \sin\left[(n-\tau)\cdot\Omega_c\right]}{(n-\tau)\cdot\pi}, n \neq \tau \\ \dfrac{\pi-\Omega_c}{\pi}, n = \tau \end{cases} \qquad n=0, 1, \ldots, 2M$$

$$\tau = M$$

Filtro PB
La figura de abajo muestra las especificaciones para un filtro pasa banda



Coeficientes del filtro PB

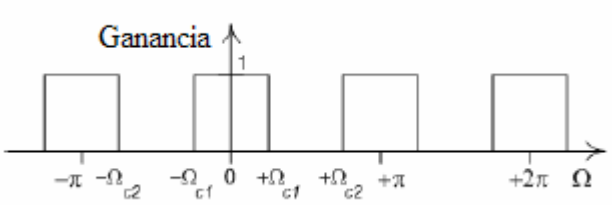
$$\tau = M$$

$$h_{PB}(n) = \frac{1}{2\cdot\pi}\cdot\left[\int_{-\Omega_{C2}}^{-\Omega_{C1}} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega + \int_{+\Omega_{C2}}^{+\Omega_{C1}} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega\right] \quad n=0,\ 1,\ \ldots,2M$$

$$h_{PB}(n) = \begin{cases} \dfrac{\sin\left[(n-\tau)\cdot\Omega_{C2}\right]-\sin\left[(n-\tau)\cdot\Omega_{C1}\right]}{(n-\tau)\cdot\pi}, n\neq\tau \\ \dfrac{\Omega_{C2}-\Omega_{C1}}{\pi}, n=\tau \end{cases} \quad n=0,\ 1,\ \ldots,2M$$

Filtro SB

La figura de abajo muestra las especificaciones para un filtro elimina banda



Coeficientes del filtro SB

$$\tau = M$$

$$h_{SB}(n) = \frac{1}{2\cdot\pi}\cdot\left[\int_{-\pi}^{-\Omega_{C2}} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega + \int_{+\Omega_{C1}}^{+\Omega_{C1}} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega + \int_{+\Omega_{C2}}^{+\pi} e^{j\cdot(n-\tau)\cdot\Omega}\cdot d\Omega\right]$$

$n=0,\ 1,\ \ldots,2M$

$$h_{SB}(n) = \begin{cases} \dfrac{\sin\left[(n-\tau)\cdot\pi\right]-\sin\left[(n-\tau)\cdot\Omega_{C2}\right]+\sin\left[(n-\tau)\cdot\Omega_{C1}\right]}{(n-\tau)\cdot\pi}, n\neq\tau \\ \dfrac{\pi-\Omega_{C2}+\Omega_{C1}}{\pi}, n=\tau \end{cases}$$

$n=0,\ 1,\ \ldots,2M$

Ejercicio 2.4.1

Determinar los coeficientes ideales de la respuesta al impulso para un filtro pasa bajas LP, de 21 puntos de extensión o longitud, que satisfaga las siguientes características.

$$w_{pass} = 2 \cdot \pi \cdot 3000 \, rad/seg \, , w_{stop} = 2 \cdot \pi \cdot 4000 \, rad/seg \, , f_s = 20 - khz$$

Primero se debe hallar la Ω_C (*frecuencia de corte*) para el filtro ideal.

$$\Omega_C = \frac{w_{stop} + w_{pass}}{2 \cdot fs}$$

$$\Omega_C = \frac{2 \cdot \pi \cdot 3000 + 2 \cdot \pi \cdot 4000}{2 \cdot 20000} = 1.0996 \, rad/seg$$

$$\tau = \frac{N-1}{2} = \frac{21-1}{2} = 10$$

Los coeficientes ideales causales se hallan con la formula:

$$h_{LP}(n) = \begin{cases} \dfrac{\sin\left[(n-\tau)\cdot\Omega_c\right]}{(n-\tau)\cdot\pi}, n \neq \tau \\ \dfrac{\Omega_c}{\pi}, n = \tau \end{cases} \qquad n=0, 1, \ldots, 2M$$

Script Matlab:

```
%Ingreso datos
clc;
w_pass = 2*pi*3000;%rad/seg
w_stop = 2*pi*4000;
f_sampler = 20000;%Hz
N = 21;%longitud o extensión filtro
%Calcula frecuencia corte wc
Omega_c = (w_stop + w_pass)/(2*f_sampler)
%calcula tau
T = (N-1)/2
M = T;
%calcula coeficientes causales ideales
for i=1:(2*T)+1
    if i == M+1
       h_LP(i) = Omega_c / pi;
    else
       h_LP(i) =(sin(((i-1)-T)*Omega_c))/(((i-1)-T)*pi);
    end
end
disp 'Coeficientes calculados';
disp '**********************';
for i=1:(2*T)+1
    fprintf('h_LP[%d]= %3.5f\n',i-1,h_LP(i))
end
```

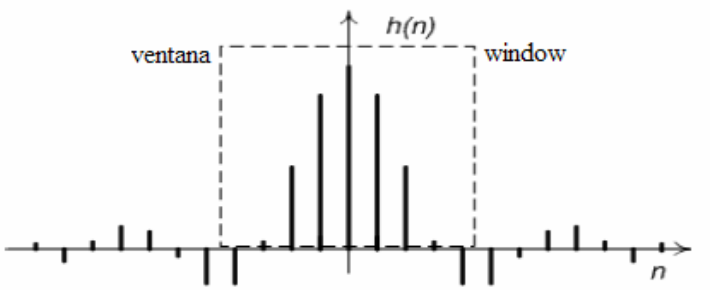
Los coeficientes causales calculados con el script son:

```
Coeficientes calculados
**********************
```

```
h_LP[0]= -0.03183
h_LP[1]= -0.01606
h_LP[2]= 0.02339
h_LP[3]= 0.04491
h_LP[4]= 0.01639
h_LP[5]= -0.04502
h_LP[6]= -0.07568
h_LP[7]= -0.01660
h_LP[8]= 0.12876
h_LP[9]= 0.28362
h_LP[10]= 0.35000
h_LP[11]= 0.28362
h_LP[12]= 0.12876
h_LP[13]= -0.01660
h_LP[14]= -0.07568
h_LP[15]= -0.04502
h_LP[16]= 0.01639
h_LP[17]= 0.04491
h_LP[18]= 0.02339
h_LP[19]= -0.01606
h_LP[20]= -0.03183
```

Técnicas Windowing (*ventanas*)

No es posible incluir un número infinito de coeficientes necesarios para implementar un filtro ideal, se tiene que reducir el número de coeficientes usados basándonos en el contenido del diseño que se tenga.
En el punto anterior se truncaban todos los coeficientes no causales mas halla de $\pm M$ y manteníamos el resto.
El procedimiento de *windowing* puede ser comparado a pegar una ventana de $N = 2 \cdot M + 1$ sobre todos los coeficientes ideales, en la figura de abajo se muestra este proceso.



Todos los coeficientes dentro de la ventana son retenidos o guardados y los coeficientes que están fuera de la ventana son descartados.

Al hacer este proceso se produce una **función rectangular *"ventana o window"***, en la cual todos los coeficientes de la ventana con los índices dentro del rango de la ventana tienen el valor de *1* y todos los otros coeficientes tienen un valor de *0*.
El *kernel* de los filtros es determinado coeficiente por coeficiente multiplicando los coeficientes ideales y los coeficientes de la ventana, así:

$$\textbf{\textit{Kernel}} \qquad h(n) = h_{IDEAL}(n) \cdot w(n) \qquad n=0,1,\ldots, \pm M$$

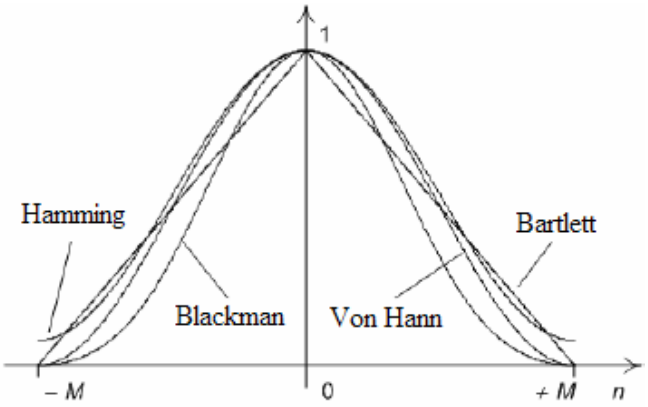
Los coeficientes de la ventana pueden ser definidos, así:

$$w_{rect}(n) = w_{rect}(-n) = 1 \qquad n=0,1,\ldots,M$$

Si embargo la truncación abrupta de los coeficientes del filtro tiene un efecto adverso en la resultante respuesta a la frecuencia del filtro. Es por eso que hay otras técnicas o funciones de *windowing* o *ventas* que reducen fácilmente los coeficientes a cero usando una truncación mas lisa, las ventanas mas conocidas son:

- Bartlett window o ventana triangular
- Rectangular window
- Von Hann window
- Hamming window
- Blackman window
- Kaiser window

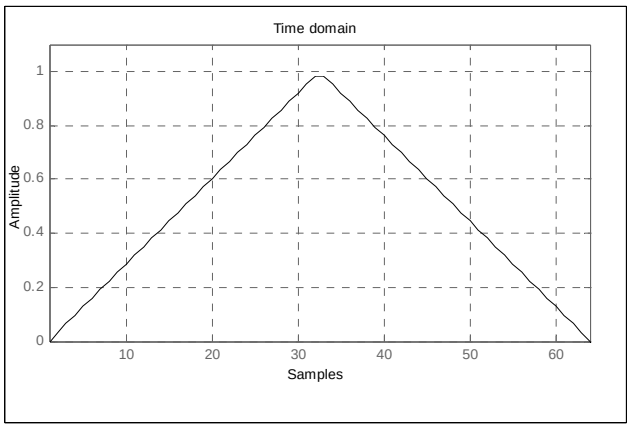
En el gráfico de abajo se puede observar la truncación de Bartllet, Hamming, Blackman y Von Hann.



Bartlett window

Llamada también ventana triangular, la truncación es más lisa que en la ventana rectangular.

En la figura de abajo se puede observar esta ventana.



Script Matlab:

Matlab puede generar esta ventana con los comandos:

```
N =64;%extencion o puntos de la ventana
wvtool(bartlett(N))
```

Los coeficientes de esta ventana pueden ser calculados así:

$$M = \frac{N-1}{2}$$

$$w_{bart}(n) = w_{bart}(-n) = \frac{M-n}{M} \quad n=0,1,\ldots,M$$

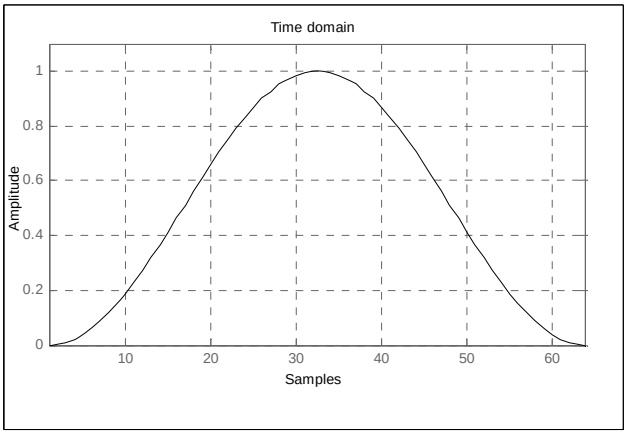
Matlab tiene un comando para calcular los coeficientes de esta ventana

```
w = bartlett(n)
```

NOTA: El vector *w* de coeficientes es simétrico (*-5,-4,-3,-2,-1, 0,1,2,3,4,5*)

Von Hann window

La figura de abajo muestra una ventana Von Hann (llamada también Hanning window).



Script Matlab:

```
N =64;%extencion o puntos de la ventana
wvtool(hann(N))
```

Los coeficientes de esta ventana pueden ser calculados así:

$$M = \frac{N-1}{2}$$

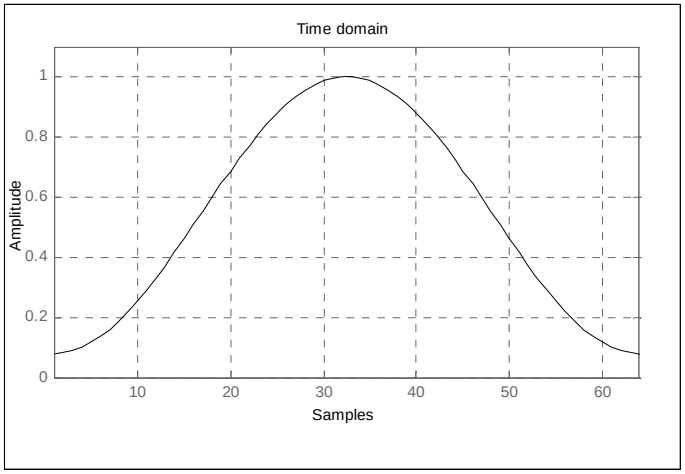
$$w_{hann}(n) = w_{hann}(-n) = 0.5\left[1-\cos\left(\frac{\pi\cdot(M-n)}{M}\right)\right] \qquad n=0,1,\ldots,M$$

Matlab tiene un comando para calcular los coeficientes de esta ventana

```
w = hann(n)
```

Hamming window

La figura de abajo muestra una ventana Hamming.



Script Matlab:

```
N =64;%extencion o puntos de la ventana
wvtool(hamming(N))
```

Los coeficientes de esta ventana pueden ser calculados así:

$$M = \frac{N-1}{2}$$

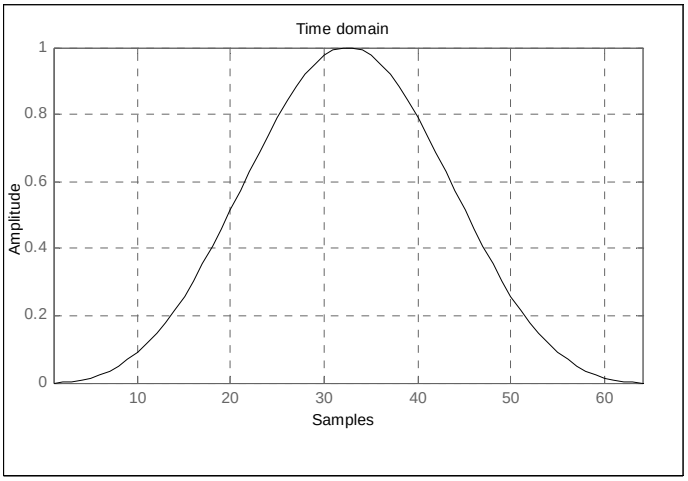
$$w_{hamm}(n) = w_{hamm}(-n) = 0.54 - 0.46 \cdot \cos\left(\frac{\pi \cdot (M-n)}{M}\right) \qquad n=0,1,\ldots,M$$

Matlab tiene un comando para calcular los coeficientes de esta ventana

```
w = hamming(n)
```

Blackman window

La figura de abajo muestra una ventana blackman.



Script Matlab:

```
N =64;%extencion o puntos de la ventana
wvtool(blackman(N))
```

Los coeficientes de esta ventana pueden ser calculados así:

$$M = \frac{N-1}{2}$$

$$w_{blck}(n) = w_{blck}(-n) = 0.42 - 0.5 \cdot \cos\left(\frac{\pi \cdot (M-n)}{M}\right) + 0.08 \cdot \cos\left(\frac{2 \cdot \pi \cdot (M-n)}{M}\right)$$

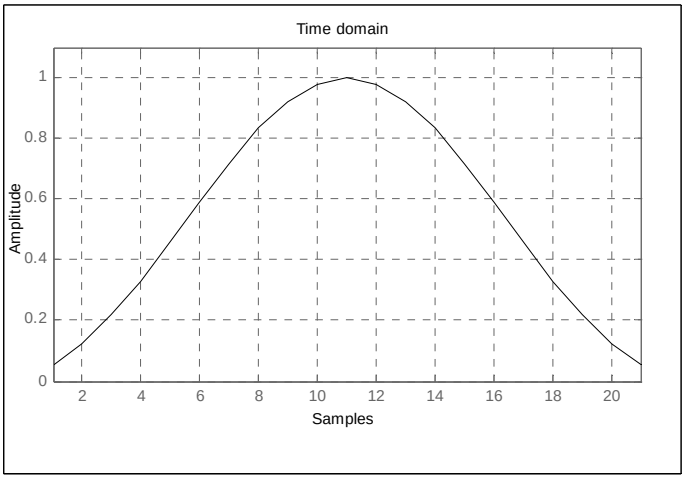
$$n=0,1,\ldots,M$$

Matlab tiene un comando para calcular los coeficientes de esta ventana

```
w =blackman(n)
```

Kaiser window

La figura de abajo muestra una ventana kaiser de longitud 21 con $\beta = 4.53$



Los coeficientes pueden ser calculados así:

$$w_{kais}(n) = w_{kais}(-n) = \frac{I_0 \cdot \left[\beta \cdot \sqrt{1-\left(\frac{2 \cdot n}{M}\right)^2}\right]}{I_0(\beta)} \qquad n=0,1,\ldots,M$$

I_0 es la función de Bessel de orden cero modificada

β Es una constante (3 - 9), controla la característica de transición entre las bandas de transición y atenuación.

$$\beta = \begin{cases} 0.1102 \cdot (A - 8.7), A > 50 \\ 0.5842 \cdot (A - 21)^{0.4}) + 0.07886 \cdot (A - 21), 21 \le A \le 50 \\ 0, A < 21 \end{cases}$$

A representa la longitud de la banda de errores ($\delta_p - \delta_s$) expresada como atenuación

$$A = -20 \cdot \log_{10}(\min(\delta_p, \delta_s))$$

min. quiere decir que se escoge el valor mas bajo.
La longitud estimada de la ventana puede ser hallada así:

$$N = \begin{cases} \frac{A - 7.95}{2.285 \cdot \Delta\Omega}, A > 21 \\ \frac{5.794}{\Delta\Omega}, A < 21 \end{cases}$$

$\Delta\Omega$ representa la banda de transición normalizada

$$\Delta\Omega = \frac{\left|w_{stop} - w_{pass}\right|}{f_s}$$

Matlab tiene un comando que nos facilita la obtención de los coeficientes de la venta

```
w = kaiser(n,beta)
```

Ejercicio 2.4.2

Determinar los coeficientes de la respuesta al impulso para un filtro pasa bajas FIR usando la ventana Hamming de longitud 21 y para las siguientes características:

$$w_{pass} = 2 \cdot \pi \cdot 3000 \, {rad}/{seg}\,,\; w_{stop} = 2 \cdot \pi \cdot 4000 \, {rad}/{seg}\,,\; f_s = 20 - khz$$

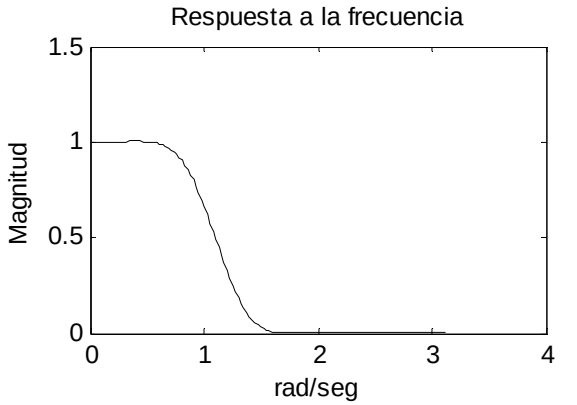
Script Matlab:

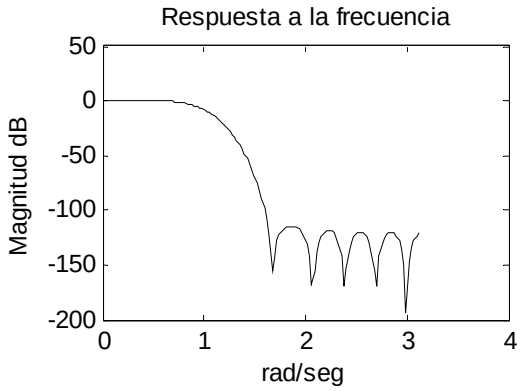
```
%Ingreso datos
clc;
clear all;
w_pass = 2*pi*3000;%rad/seg
w_stop = 2*pi*4000;
f_sampler = 20000;%Hz
N = 21;%longitud o extencion filtro
%Calcula frecuencia corte wc
Omega_c = (w_stop + w_pass)/(2*f_sampler)
%calcula tau
T = (N-1)/2
M = T;
%calcula coeficientes ideales causales
for i=1:(2*T)+1
    if i == M+1
       h_LP(i) = Omega_c / pi;
    else
       h_LP(i) =(sin(((i-1)-
T)*Omega_c))/(((i-1)-T)*pi);
    end
end
disp 'Coeficientes ideales calculados';
disp '************************';
for i=1:(2*T)+1
    fprintf('h_LP[%d]= %3.5f\n',i-
1,h_LP(i))
end
%calcula coeficientes de ventana Hamming
disp 'Coeficientes ventana Hamming';
disp '************************';
w_hamm = hamming(N)
%Calculo coeficientes
%h(n)=h_ideal(n)*w(n)
%h(n)=h_LP(n)*w_hamm(n)
for i=1:(2*T)+1
   h(i)=h_LP(i)*w_hamm(i);
end
disp 'Coeficientes calculados FIR';
disp '************************';
for i=1:(2*T)+1
    fprintf('h[%d]= %3.5f\n',i-1,h(i))
end

Hz=tf(h,1,'variable','z^-1')
%gráfica respuesta a la frecuencia
```

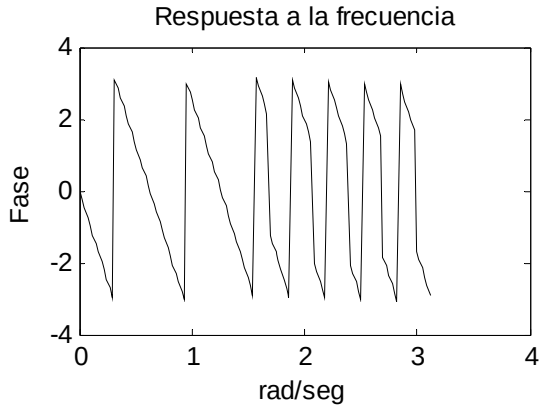
```
[num,den] = tfdata(Hz,'v');
[H,w]=freqz(num,den,128);
plot(w,abs(H),'-k')
ylabel('Magnitud');
xlabel('rad/seg')
title('Respuesta a la frecuencia');
figure(5);
plot(w,20*log(abs(H)),'-k')
ylabel('Magnitud dB');
xlabel('rad/seg')
title('Respuesta a la frecuencia');
%gráfica fase
figure(2);
plot(w,angle(H),'-k')
title('Respuesta a la frecuencia');
ylabel('Fase');
xlabel('rad/seg')
%plano z
figure(3);
zplane(num,den)
title('Plano-z');
%Respuesta al impulso
[h,t]=impz(num,den,128);
figure(4);
plot(t,h,'-k');
title('Respuesta al impulso');
%gráfica respuesta al escalón
[h,t]=stepz(num,den,128);
figure(6);
plot(t,h,'-k');
title('Respuesta al escalon');
```

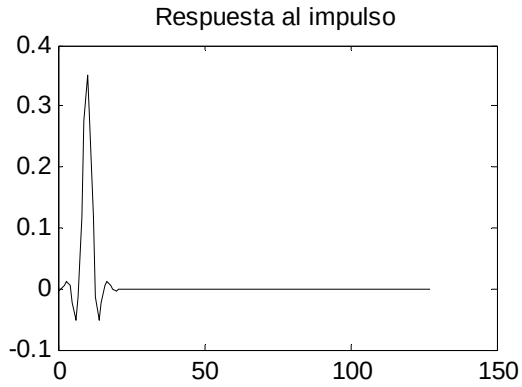
Los coeficientes que obtenemos son los que se muestran abajo y pueden verse en la command window de Matlab:

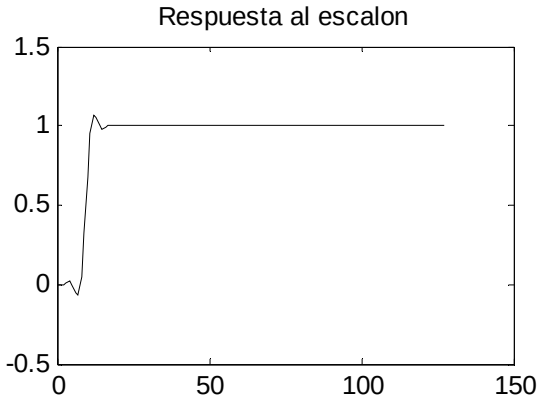
h_ideal (h_LP)	w_hamm	H(z) kernel
h_LP[0]= -0.03183	0.0800	h[0]= -0.00255
h_LP[1]= -0.01606	0.1025	h[1]= -0.00165
h_LP[2]= 0.02339	0.1679	h[2]= 0.00393
h_LP[3]= 0.04491	0.2696	h[3]= 0.01211
h_LP[4]= 0.01639	0.3979	h[4]= 0.00652
h_LP[5]= -0.04502	0.5400	h[5]= -0.02431
h_LP[6]= -0.07568	0.6821	h[6]= -0.05163
h_LP[7]= -0.01660	0.8104	h[7]= -0.01345
h_LP[8]= 0.12876	0.9121	h[8]= 0.11745
h_LP[9]= 0.28362	0.9775	h[9]= 0.27723
h_LP[10]= 0.35000	1.0000	h[10]= 0.35000
h_LP[11]= 0.28362	0.9775	h[11]= 0.27723
h_LP[12]= 0.12876	0.9121	h[12]= 0.11745
h_LP[13]= -0.01660	0.8104	h[13]= -0.01345
h_LP[14]= -0.07568	0.6821	h[14]= -0.05163
h_LP[15]= -0.04502	0.5400	h[15]= -0.02431
h_LP[16]= 0.01639	0.3979	h[16]= 0.00652
h_LP[17]= 0.04491	0.2696	h[17]= 0.01211
h_LP[18]= 0.02339	0.1679	h[18]= 0.00393
h_LP[19]= -0.01606	0.1025	h[19]= -0.00165
h_LP[20]= -0.03183	0.0800	h[20]= -0.00255











Ejercicio 2.4.3

Determinar los coeficientes de la respuesta al impulso para un filtro FIR pasa banda (*BP*) usando la ventana Kaiser que satisfaga las siguientes características:

$$a_{pass1} = -0.5dB\,,\, a_{stop1} = a_{stop2} = -50dB$$
$$f_{pass1} = 4 - khz\,,\, f_{stop1} = 2 - khz$$
$$f_{pass2} = 5 - khz\,,\, f_{stop2} = 8 - khz$$
$$f_{sampler} = 20 - khz$$

Calculamos los errores en la banda de paso y atenuación ($\delta_p - \delta_s$)

$$\delta_p = 1 - 10^{0.05 \cdot a_{pass}} = 1 - 10^{0.05 \cdot -0.5} = 0.055939$$
$$\delta_s = 10^{0.05 \cdot a_{stop}} = 10^{0.05 \cdot -50} = 0.0031623$$

A continuación debemos hallar A, β, $\Delta\Omega$ y N

- $$\beta = \begin{cases} 0.1102 \cdot (A - 8.7), A > 50 \\ 0.5842 \cdot (A - 21)^{0.4}) + 0.07886 \cdot (A - 21), 21 \le A \le 50 \\ 0, A < 21 \end{cases}$$
- $$A = -20 \cdot \log_{10}(\min(\delta_p, \delta_s))$$
- $$N = \begin{cases} \dfrac{A - 7.95}{2.285 \cdot \Delta\Omega}, A > 21 \\ \dfrac{5.794}{\Delta\Omega}, A < 21 \end{cases}$$
- $$\Delta\Omega = \frac{\left| w_{stop} - w_{pass} \right|}{f_s}$$

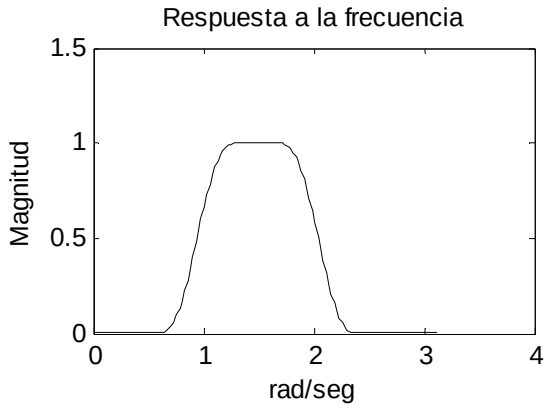
Script Matlab:

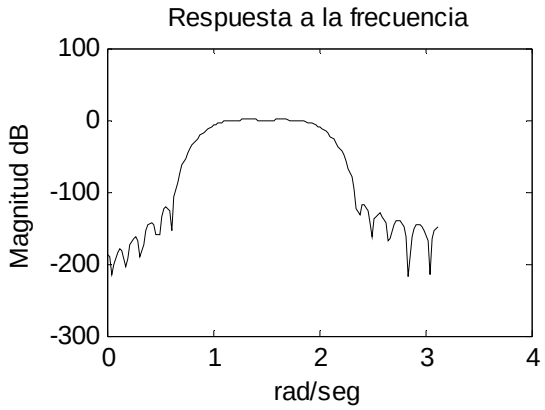
```
%ingreso Datos
clc;
clear all;
f_pass1 = 4000;%Hz
f_pass2 = 5000;
f_stop1 = 2000;
f_stop2 = 8000;
f_sampler = 20000;
a_pass1 = -0.5;
a_stop1 = -50;
%Calculo errores bandas paso y atenuación dp ds
dp = 1-10^(0.05*a_pass1)
ds = 10^(0.05*a_stop1)
%Calculo A
if dp < ds
    mind = dp;
else
    mind = ds;
end
A = -20*log10(mind)
%calcula B
if A > 50
    B = 0.1102*(A-8.7)
elseif A < 21
    B = 0
else
    B = 0.5842*(A-21)^0.4+0.07886*(A-21)
end
%calcula DOmega
DOmega1 = 2*pi*(f_stop1 - f_pass1)/f_sampler
DOmega2 = 2*pi*(f_stop2 - f_pass2)/f_sampler
if DOmega1 < DOmega2
    DOmega = abs(DOmega1);
else
```

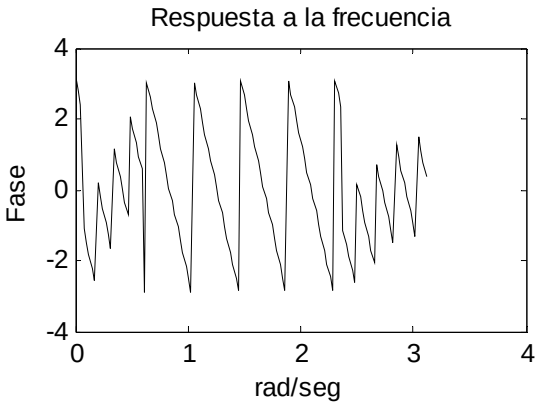
```
    DOmega = abs(DOmega2);
end
%calcula N
if A > 21
    N = (A-7.95)/(2.285*DOmega)
else
    N = 5.794/DOmega
end
% N sale 29.2 redondeando para que sea impar quedaría 31
%!!!!!!!!
N = 31
%!!!!!!!!
%Calcula frecuencias corte
Omega_C1 = 2*pi*(f_stop1 + f_pass1)/(2*f_sampler)
Omega_C2 = 2*pi*(f_stop2 + f_pass2)/(2*f_sampler)
%calcula tau
T = (N-1)/2
M = T;
%calcula coeficientes ideales
for i=1:(2*T)+1
    if i == M+1
       h_BP(i) = (Omega_C2 - Omega_C1) / pi;
    else
       h_BP(i) =((sin(((i-1)-T)*Omega_C2))-(sin(((i-1)-T)*Omega_C1))) /
(((i-1)-T)*pi);
    end
end
disp 'Coeficientes ideales calculados';
disp '************************';
for i=1:(2*T)+1
    fprintf('h_BP[%d]= %3.5f\n',i-1,h_BP(i))
end
%calcula coeficientes de ventana kaiser
disp 'Coeficientes ventana Hamming';
disp '************************';
w_kais = kaiser(N,B)
%Calculo coeficientes
%h(n)=h_ideal(n)*w(n)
%h(n)=h_LP(n)*w_hamm(n)
for i=1:(2*T)+1
   h(i)=h_BP(i)*w_kais(i);
end
disp 'Coeficientes calculados FIR';
disp '************************';
for i=1:(2*T)+1
    fprintf('h[%d]= %3.7f\n',i-1,h(i))
end

Hz=tf(h,1,'variable','z^-1')
%gráfica respuesta a la frecuencia
[num,den] = tfdata(Hz,'v');
[H,w]=freqz(num,den,128);
plot(w,abs(H),'-k')
ylabel('Magnitud');
xlabel('rad/seg')
title('Respuesta a la frecuencia');
figure(5);
plot(w,20*log(abs(H)),'-k')
ylabel('Magnitud dB');
xlabel('rad/seg')
title('Respuesta a la frecuencia');
%gráfica fase
figure(2);
plot(w,angle(H),'-k')
title('Respuesta a la frecuencia');
ylabel('Fase');
xlabel('rad/seg')
%plano z
figure(3);
zplane(num,den)
title('Plano-z');
%Respuesta al impulso
[h,t]=impz(num,den,128);
figure(4);
plot(t,h,'-k');
title('Respuesta al impulso');
%gráfica respuesta al escalon
```

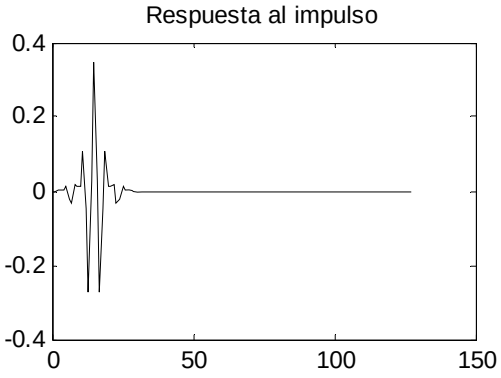
```
[h,t]=stepz(num,den,128);
figure(6);
plot(t,h,'-k');
title('Respuesta al escalon');
```

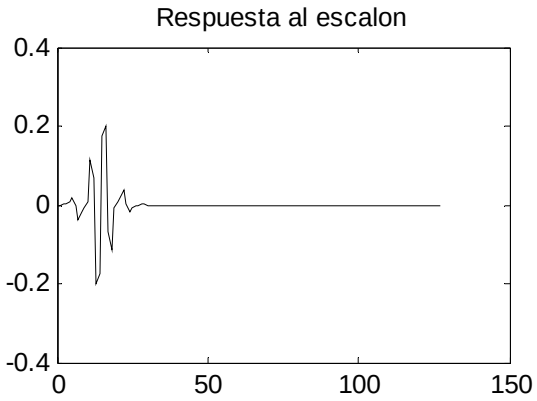
h_ideal (h_BP)	w_kais	H(z) kernel
h_BP[0]= -0.03623	0.0555	h[0]= -0.0020120
h_BP[1]= -0.02039	0.0989	h[1]= -0.0020162
h_BP[2]= 0.03175	0.1528	h[2]= 0.0048506
h_BP[3]= 0.00964	0.2168	h[3]= 0.0020888
h_BP[4]= 0.01027	0.2898	h[4]= 0.0029774
h_BP[5]= 0.03183	0.3703	h[5]= 0.0117872
h_BP[6]= -0.04467	0.4561	h[6]= -0.0203739
h_BP[7]= -0.06123	0.5446	h[7]= -0.0333459
h_BP[8]= 0.03086	0.6329	h[8]= 0.0195331
h_BP[9]= 0.01479	0.7180	h[9]= 0.0106179
h_BP[10]= 0.01865	0.7965	h[10]= 0.0148523
h_BP[11]= 0.12246	0.8656	h[11]= 0.1060046
h_BP[12]= -0.04939	0.9226	h[12]= -0.0455616
h_BP[13]= -0.28012	0.9650	h[13]= -0.2703138
h_BP[14]= 0.02610	0.9912	h[14]= 0.0258672
h_BP[15]= 0.35000	1.0000	h[15]= 0.3500000
h_BP[16]= 0.02610	0.9912	h[16]= 0.0258672
h_BP[17]= -0.28012	0.9650	h[17]= -0.2703138
h_BP[18]= -0.04939	0.9226	h[18]= -0.0455616
h_BP[19]= 0.12246	0.8656	h[19]= 0.1060046
h_BP[20]= 0.01865	0.7965	h[20]= 0.0148523
h_BP[21]= 0.01479	0.7180	h[21]= 0.0106179
h_BP[22]= 0.03086	0.6329	h[22]= 0.0195331
h_BP[23]= -0.06123	0.5446	h[23]= -0.0333459
h_BP[24]= -0.04467	0.4561	h[24]= -0.0203739
h_BP[25]= 0.03183	0.3703	h[25]= 0.0117872
h_BP[26]= 0.01027	0.2898	h[26]= 0.0029774
h_BP[27]= 0.00964	0.2168	h[27]= 0.0020888
h_BP[28]= 0.03175	0.1528	h[28]= 0.0048506
h_BP[29]= -0.02039	0.0989	h[29]= -0.0020162
h_BP[30]= -0.03623	0.0555	h[30]= -0.0020120






Respuesta a la frecuencia
Fase
rad/seg
4
2
0
-2
-4
0
1
2
3
4


Respuesta al impulso
0.4
0.2
0
-0.2
-0.4
0
50
100
150


Respuesta al escalon
0.4
0.2
0
-0.2
-0.4
0
50
100
150

Introducción a los DSPs

DSPs significa procesador digital de señales, son los encargados de llevar a cabo todas las operaciones matemáticas, los *DSPs* son microprocesadores especialmente diseñados para manejar tareas de *DSP* , como algoritmos de filtrado, análisis de Fourier, etc.
La fabricante de chips *MICROCHIP* fabrica ***DSC*** (Digital Signal Controllers) llamados ***dsPIC*** que combinan el alto desempeño de un microcontrolador de 16 bits con la potencia de cálculo de un *DSPs*.
Los *dsPIC* tienen el "*núcleo*" de un microcontrolador de 16 bits, con periféricos robustos y la capacidad de manejar interrupciones rápidamente y la "*mente*" de un *DSPs* que maneja las tareas de cálculo de alto nivel, creando una solución óptima en un solo chip, los *dsPIC* pueden realizar hasta 30 MIPS (30 Millones de operaciones por segundo) a gran velocidad, son muy eficientes para programar en *lenguaje C* y tienen memoria *FLASH*, memoria *EEPROM* y memoria de datos *SRAM*.
La diferencia principal entre un *microprocesador* y un *microcontrolador* es que microprocesador es un sistema abierto al que se le puede mejorar algunas características deseadas, mientras que el microcontrolador es un sistema cerrado con prestaciones limitadas que no pueden ser variadas respectivamente.
Existen en el mercado otros tipos de *DSPs* de otras marcas como: Motorota, Intel, Sharp, Texas Instruments, etc.

3.1 DSC dsPIC (Controlador digital de señales)

Para la implementación de los filtros digitales se ha escogido un *dsPIC* de la familia *dsPIC30F*, el *dsPIC30F4011* por razones de precio, disponibilidad y conocimiento de la tecnología de microcontroladores *PICmicro*, la arquitectura de los dsPIC es una arquitectura Harvard modificada mientras que la arquitectura de un microcontrolador *PICmicro* es arquitectura Harvard.

Características generales del dsPIC n

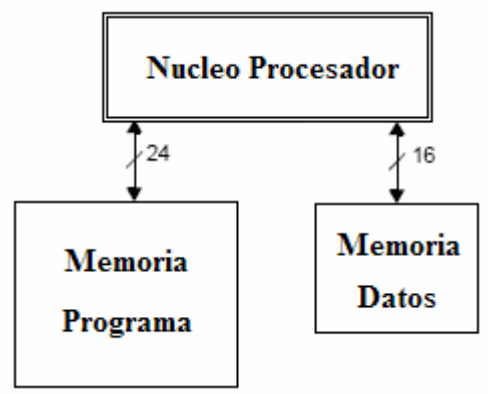
Un controlador digital de señal *dsPIC* posee al igual que un *PICmicro* todos los componentes de un computador los cuales no se pueden variar, entre las características más notorias de este *dsPIC* son:

- Arquitectura Harvard modificada
- 84 instrucciones básicas
- Instrucciones de 24-bits, datos 16-bits
- Memoria Flash de programa 48 kb
- Memoria SRAM 2 kb, memoria EEPROM 1 Kb
- 30 MIPs (oscilador 4-10 Mhz, PLL de 4x, 8x, 16x)
- 30 fuentes de interrupción
- Núcleo (*core*) con DSP engine
- Todas las instrucciones de DSP son de 1 ciclo
- SPI, I2C, CAN, UART, 3 TIMERS, 6 PWM, ADC 10 bits(500 Ksps)
- Modulo y direccionamiento de bit Reverso

Para un conocimiento más detallado del controlador digital de señales dsPIC véase el manual de referencia ***DS70135C, DS70067C*** y ***DS70046C*** de MICROCHIP.

Arquitectura dsPIC

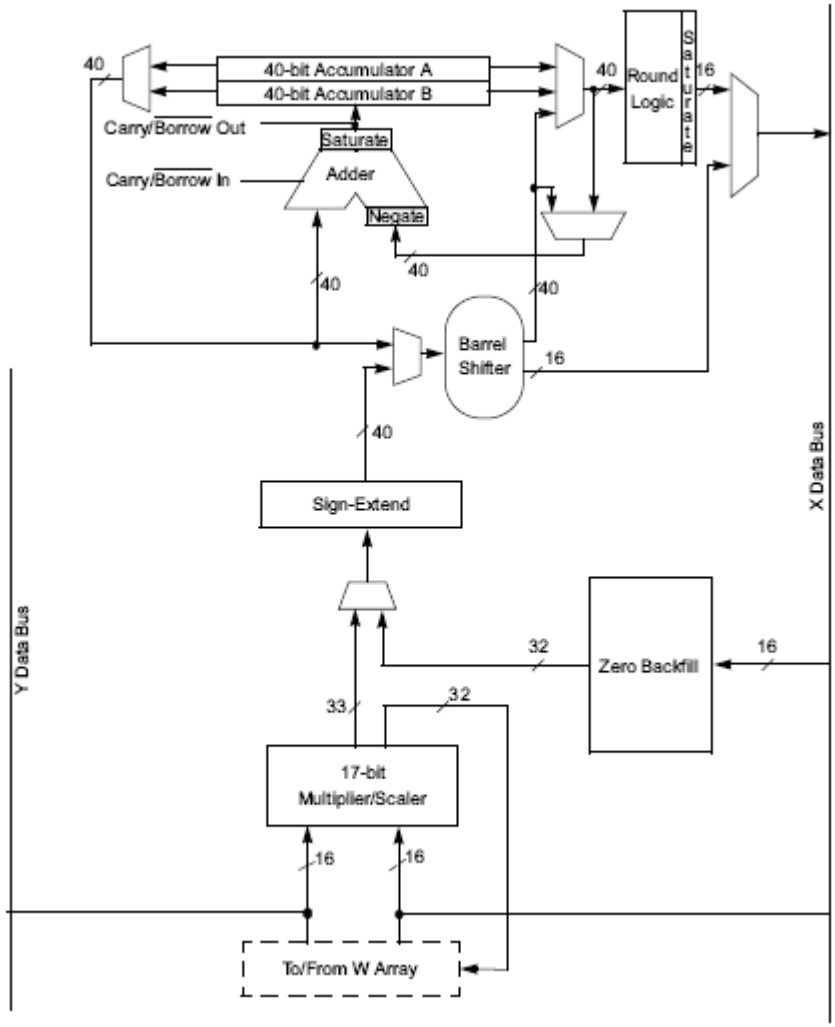
El *dsPIC* tiene una arquitectura Hardvard modificada con buses *separados* para la memoria de datos y de programa como se muestra en la siguiente figura:



Las instrucciones del *core o núcleo* son de 24-bits, con un *program counter PC* de 23-bits, así el *PC* puede direccional instrucciones en un espacio de memoria superior a los *4 Mb*.
El registro de trabajo es un array de *16x16 bits*, este puede actuar como registro de datos, dirección o offset. La memoria de datos es de 64 Kb y un ancho de datos de 16 bits y esta dividida en dos bloques de memoria de datos *X* y *Y*, cada bloque tiene su propia unidad de generación de direcciones (AGU), la mayoría de instrucciones pueden operar solamente en la memoria de datos *X AGU* proporcionando así una apariencia simple unificada en la memoria de datos.
El *multiply accumulate* ***MAC*** es una clase de instrucción de doble fuente para DSP que opera en ambos *AGUs* en *X y en* Y. Posee un modulo de direccionamiento para buffers circulares que soporta ambos espacios de direcciones *X* y *Y*, los buffers circulares son usados en la mayoría de algoritmos de DSP.
El *AGU X* soporta direccionamiento de bit reverso, el cual es usado por el algoritmo radix-2 FFT.
La operación de *C=A+B* es ejecutada en un ciclo, un ***DSP engine*** a sido incluido para dar más capacidad aritmética al *cor*e las instrucciones de DSP han sido diseñadas para un óptimo rendimiento en tiempo real.
Este dsPIC soporta divisiones de tipo: 16/16 signed, 32/16 signed, 32/16 unsigned, 16/16 unsigned.

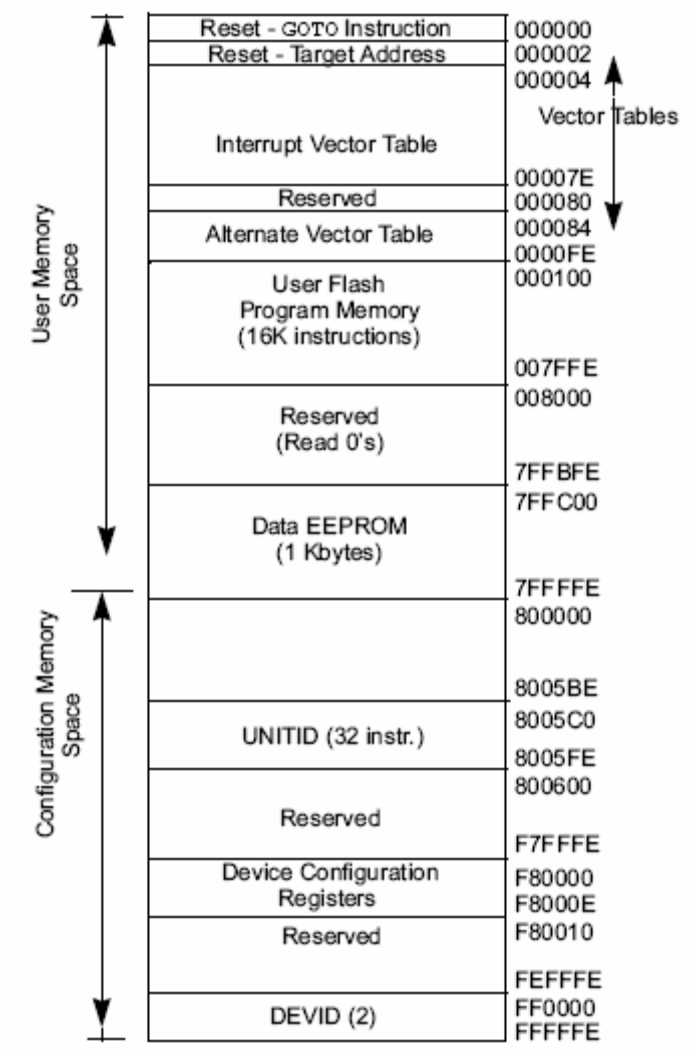
DSP engine

El DSP engine es un bloque de hardware que alimenta con datos al registro de trabajo **W,** pero este contiene sus propios registros especializados.
El DSP engine consiste de un multiplicador de 17x17 bits de alta velocidad, un barril de cambio y de un sumador/substractor de 40-bits.
El DSP engine tiene varias opciones que pueden ser seleccionadas a través de bits en el Registro CPU Core Configuration *CORCON*.
Abajo se muestra un diagrama del DSP engine



Organización de memoria

La capacidad de direccionamiento de la memoria de programa es de hasta 4 Mb de instrucciones, es direccionado por un *PC* de 23-bits. La tabla de abajo muestra una distribución de la memoria de programa:



AGU – unidad generadora de direcciones

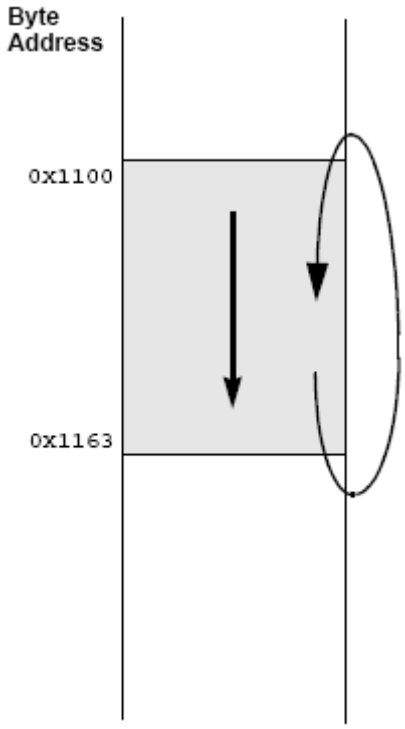
El dsPIC30F contiene 2 AGUs (X, Y) para generar direcciones de la memoria de datos. Ambos AGUs pueden generar cualquier dirección efectiva ***EA*** dentro de un rango de 64Kb, si existiera algún error en la dirección se generara una interrupción. La AGU soporta 3 tipos de direccionamiento de datos:

- Direccionamiento lineal
- Direccionamiento circular
- Direccionamiento de bit reverso

Los dos primeros son aplicables para la memoria de datos y programa, el último es solo aplicable para la memoria de datos.

Direccionamiento circular

El direccionamiento circular proporciona un automatizado soporte de buffers de datos circulares por medio de hardware, evitando así la necesidad de hacer por software el manejo de buffers de datos circulares, típicamente usado en algoritmos que usan realimentación. En el gráfico de abajo se muestra un direccionamiento circular



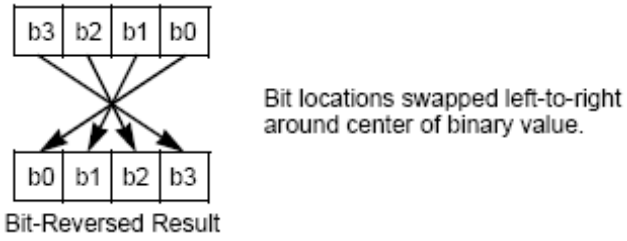
Por ejemplo: Para configurar en modo incremental el modulo

```
mov  #0x1100,W0
mov  W0,XMODSRT;start address
mov  #0x1163,W0
mov  W0,XMODEND;end address
mov  #0x8001,W0
mov  W0,MODCON;Enable W1, X AGU for
module
mov  #0000,W0;W0 holds buffer fill value
mov  #1100,W1;point W1 to buffer
DO  #49,FILL;Fill the 50 buffer location
FILL:
mov  W0,[W1++];fill the next location
```

Direccionamiento de bit reverso

Este direccionamiento simplifica el reordenamiento de los datos para los algoritmos *radix-2 FFT* , el direccionamiento de bit reverso se lleva acabo creando una imagen espejo "*mirror image*" de un puntero dirección cambiando los bits localizados alrededor del centro del valor binario.

En la figura de abajo se muestra la secuencia de bit reverso



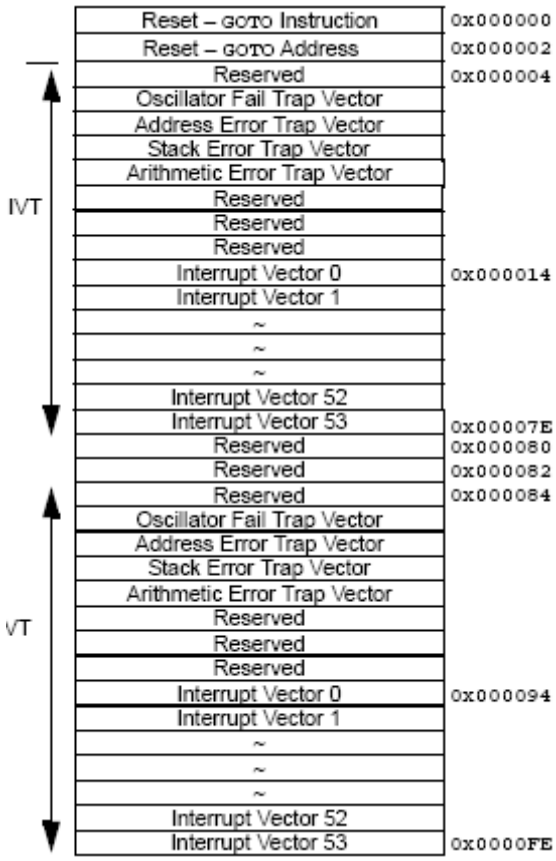
Ejemplo:
Leer una serie de 16 datos y escribir a una nueva localización en orden de bit reverso.

```
; Set XB for 16-word buffer, enable bit reverse addressing
MOV #0x8008,W0
MOV W0,XBREV
; Setup MODCON to use W1 for bit reverse addressing
MOV #0x01FF,W0
MOV W0,MODCON
; W0 points to input data buffer
MOV #Input_Buf,W0
; W1 points to bit reversed data
MOV #Bit_Rev_Buf,W1
; Re-order the data from Input_Buf into Bit_Rev_Buf
REPEAT #15
MOV [W0++],[W1++]
```

Interrupciones

El dsPIC 30F4011 tiene 30 fuentes de interrupción y 4 excepciones del procesador (*TRAPS*), las cuales deben ser arbitradas en base a un esquema de prioridad, hay 7 niveles de prioridad seleccionables por el usuario, cada interrupción tiene su propio vector.

En la figura de abajo se muestra la tabla de vectores de interrupción:



Los registros que controlan los niveles de prioridad en la interrupción son *IPC0:10*, los niveles de prioridad son de 0:7, 0 deshabilita la fuente de interrupción y 7 es el nivel más alto de interrupción.
Los registros *IEC0:2* controlan la habilitación de las interrupciones, los registros IFS0:2 nos permiten ver que interrupción fue generada.
Una mejora notable de los *dsPIC* comparada con los *PICmicro* de la familia 16 y 18 es que cada interrupción tiene su propio vector, esto da una ventaja en ahorro de tiempo y reducción en el servicio de la interrupción ya que no hay que estar preguntando que fuente genero la interrupción, en los *PICmicro* de la familia 18 existían solo 2 vectores de interrupción uno de prioridad alta y otro de prioridad baja, cuando ocurría una interrupción era necesario preguntar que fuente genero la interrupción.

En lenguaje C *MPLAB C30* el vector de interrupción del ADC se define así:

```
void __attribute__((__interrupt__)) _ADCInterrupt( void )
{
        //rutina
        //limpiar flag
}
```

Para saber como se define los vectores para los demás periféricos y módulos revisar el manual del compilador C *MPLAB C30 (**DS51284E**)*.

Oscilador

El sistema tiene las siguientes características:

- Varias fuentes de reloj internas y externas
- Multiplicador de frecuencia PLL (4x, 8x, 16x)
- Cambio entre varias fuentes de reloj
- Postscaler divide frecuencia para 4, 16, 64

El rango de frecuencia de entrada es de 4-10 Mhz.

Por ejemplo:
Si tenemos una frecuencia de 7.3728 Mhz a la entrada y tenemos PLL con 16x activo, tendremos:

$$Fout = F\,\text{int} \cdot PLL$$

$$Fout = 7.3728Mhz \cdot 16x = 120Mhz$$

La frecuencia más alta de trabajo es 120-Mhz

Ciclos de maquina Fcy
Un ciclo maquina es el tiempo mínimo que demora el microprocesador en realizar una acción.

$$F_{CY} = \frac{Fosc}{4} = \frac{frecuencia\_oscilador * PLL}{Postscaler * 4}$$

$$F_{CY} = \frac{Fosc}{4} = \frac{7372800 * 16}{4} \approx 30MIPS$$

Memoria EEPROM

La memoria de datos EEPROM es RD/WR durante el funcionamiento normal sobre el rango de VDD, hay 4 registros que controlan su funcionamiento:

- NVMCOM
- NVMADR
- NVMADRU
- NVMKEY

El dsPIC 30F4011 tiene una memoria EEPROM de 1K.

Puertos I/O

Todos los puertos del dsPIC tienen entradas Schmitt Trigger para mejorar la inmunidad al ruido.

Puertos Paralelos I/O (PIO)

Todos los pines de los puertos tienen tres registros asociados con la operación del puerto.
El registro *TRISx* determinada cuando el pin es entrada o salida. Los registros *LATx* y *PORTx* sirven para leer o escribir en los pines.
Un ciclo de instrucción es requerido entre el cambio dirección de un puerto o al leer o escribir al puerto, la instrucción *NOP* puede ser usada.

Por ejemplo:

```
MOV     0xFF00,W0       ;configura PORTB <15:8> como entradas
MOV     W0,TRISB
NOP                     ;delay un ciclo
BTSS    PORTB,#3        ;próxima instrucción
```

El registro *ADPCFG* define cuando un pin es analógico o digital, *1* configura como digital y *0* configura como analógico.

La configuración de los pines del *dsPIC* comparada con la de *PICmicro* es casi la misma la única diferencia es el registro *ADPCFG*.

Módulos

TIMERs

Hay 5 TIMERs de 16 bits, los mismos que están diseñados como *TIMER0, TIMER1,…, TIMER*5.
Cada *TIMER* tiene los siguientes registros:

- TMRx registro de 16 bits (contador)
- PRx periodo
- TxCON registro de control

Cada TIMER tiene asociado bits de control de interrupciones:

- TxIE habilita interrupción
- TxIF flag de estado de la interrupción
- TxIP controla prioridad de interrupción

Todos los TIMERs tienen la misma circuiteria funcional, pero pueden variar algunas características entre los diferentes TIMERs.
Cada TIMER puede operar en los siguientes modos:

- Como timer síncrono
- Como contador síncrono
- Como contador asíncrono
- Como gated timer
- Hay 3 tipos de TIMERs: tipo A, tipo B y tipo C.

La configuración de los TIMERs de los *dsPIC,* si se compara con la de los *PICmicro* es prácticamente igual.

Input Capture

Este modulo es usado para capturar un valor de tiempo en base a uno o dos bases de tiempo, es muy usado en aplicaciones de medición de frecuencia o medición de un impulso.
El modulo Input Capture puede operar en varios modos, el registro de configuración es el *ICxCON*.

Output Compare

Este modulo tiene la habilidad de comparar los valores de una base de tiempo seleccionada con los valores de uno o dos registros de comparación. Se puede generar un simple pulso o un tren de impulsos, los canales de comparación están diseñados como *OC1, OC2*, etc…
Cada canal *Output Compare* tiene los siguientes registros:

- OCxCON control de canal
- OCxR registro de datos
- OCxRS registro secundario de datos

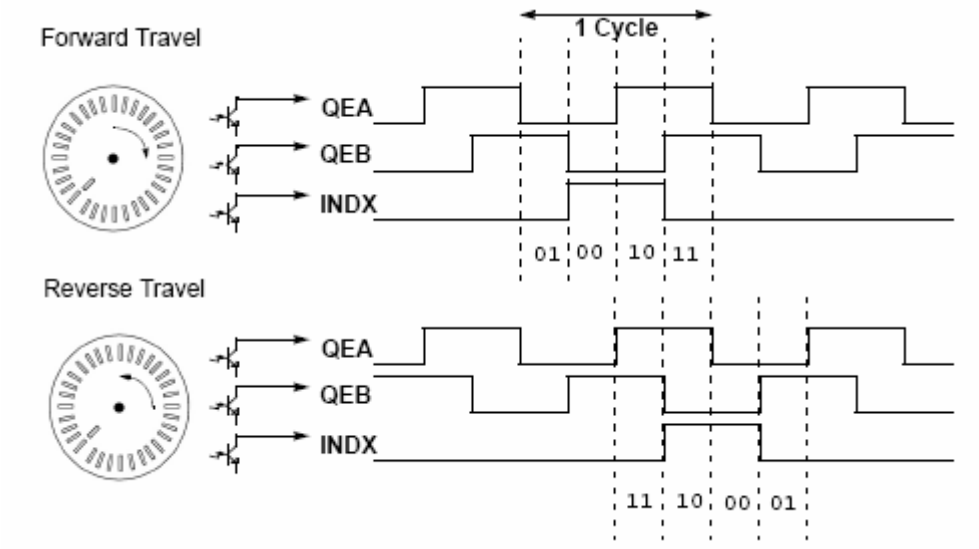
Cada modulo *Output Compare* puede operar como:

- Modo de comparación simple
- Modo de comparación dual
- Modo PWM

Este modulo puede ser utilizado para generar una señal de salida analógica, existe una nota de aplicación que nos enseña como hacer esto, la nota de aplicación se llama "**Using PWM to Generate Analog Output " (DS00538C)**.

Quadrature Encoger Interface QEI

Este modulo es usado en detección de velocidad y posición en sistemas de rotación. El QEI proporciona la interfase para encoders incrementales, para obtener el dato de la posición mecánica.
En el gráfico de abajo se muestra las señales de interfase de un encoder



Características del QEI:

- Tres entradas para dos señales QEA y QEB, y un índice de pulso INDX como en el gráfico de arriba.
- Contador de estado de dirección
- Modo de medición de posición
- Filtro digital programable para eliminación de ruido en la entrada
- Contador/Timer alternativo
- Interrupciones QEI

Todas estas características son seleccionadas configurando los siguientes registros:

- QEICON
- DFLTCON
- POSCNT
- MAXCNT
- ADPCFG

ADC 10 bits

El conversor A/D permite la conversión de una señal analógica a un número digital de 10 bits.
El ADC tiene las siguientes características:

- Conversión por aproximación sucesiva ***SAR***
- Pines de referencia de voltaje externos
- Ciclo máximo de muestreo 500ksps.

El modulo A/D tiene 6 registros de configuración:

- ADCON1 registros de control
- ADCON2
- ADCON3
- ADCHS selector entradas
- ADPCFG configuración de puertos
- ADCSSL analiza entradas

Frecuencia de conversión A/D

El convertidor A/D tiene un ciclo máximo en el cual la conversión es completada, un reloj analógico T_{AD} controla el tiempo de conversión. La conversión A/D requiere de 12 periodos T_{AD}.
El reloj del periodo de conversión A/D es seleccionado por software usando un contador de 6-bits.

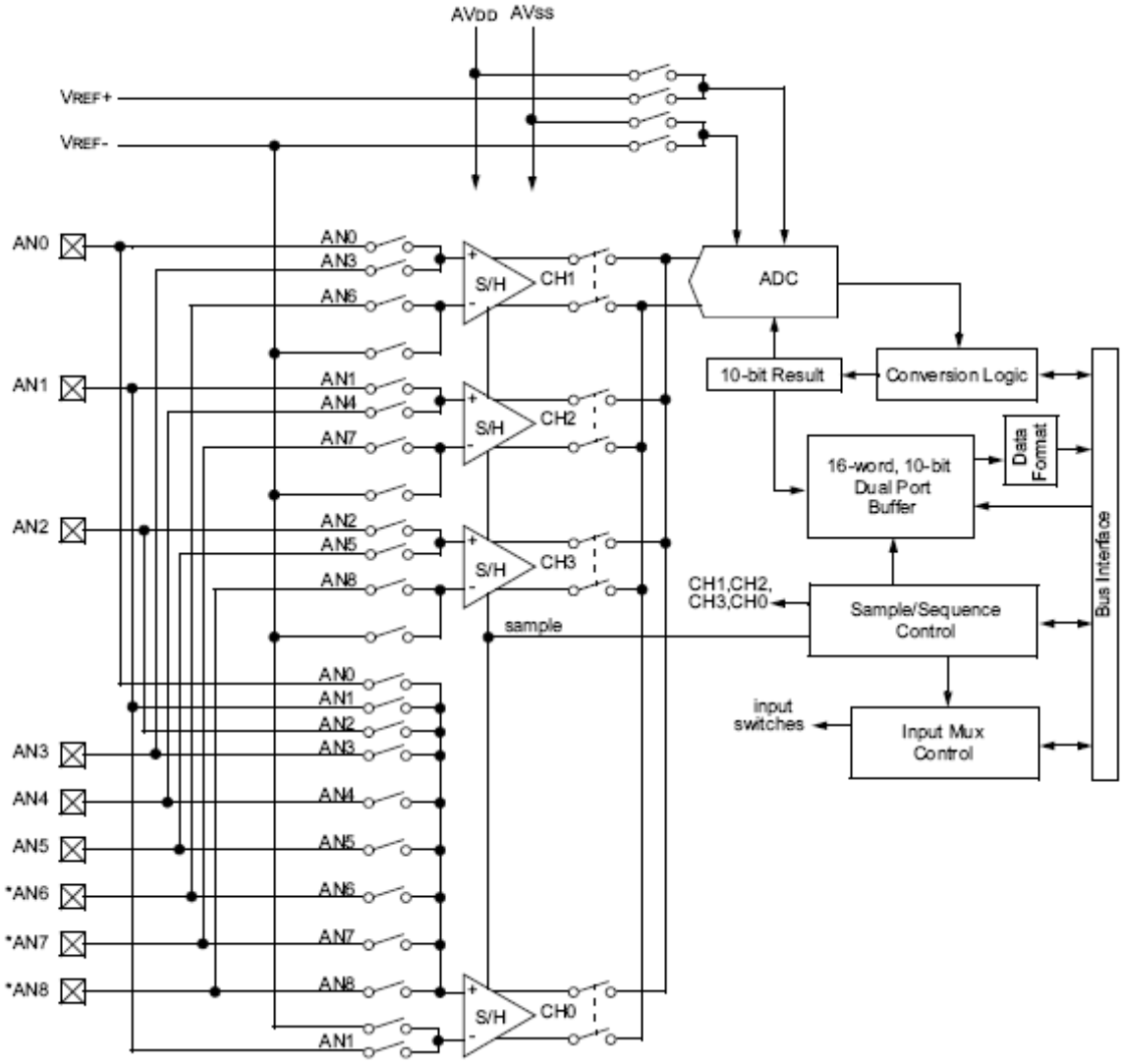
A/D periodo de conversión

$$T_{AD} = \frac{T_{CY}\,(ADCS + 1)}{2}$$

$$ADCS = \frac{2 \cdot T_{AD}}{T_{CY}} - 1$$

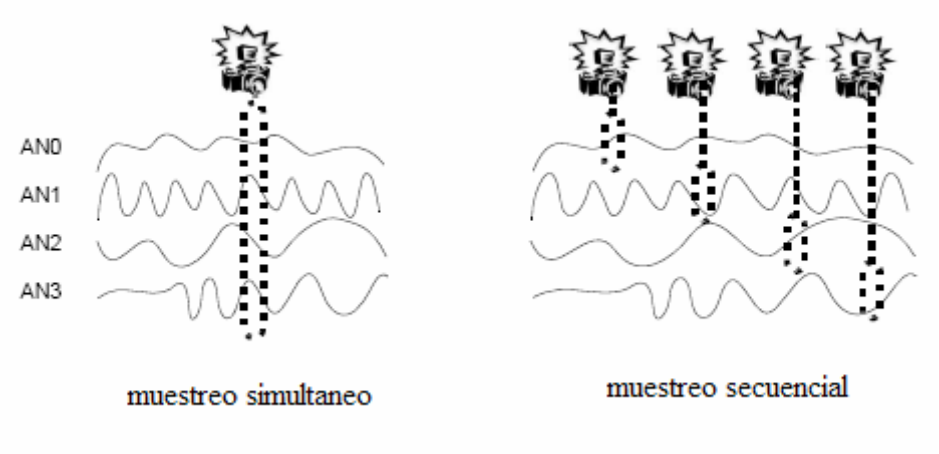
Para una correcta conversión del A/D se debe asegurar mínimo un T_{AD} de *153.85 nsec*.

En la figura de abajo se muestra un bloque del ADC:



El *ADC* del *dsPIC* es mucho mas configurable que el de los *PICmicro*, el resultado de la conversión puede ser visto en 4 tipos de formatos, tiene 4 S/H con entradas diferenciales, las cuales pueden estar referidas a *Vref-* o a un pin, para muestrear a 500Ksps es necesario usar 2 S/H.

Se puede hacer muestreo secuencial o simultaneo, en la figura de abajo se muestra como es un muestreo secuencial y simultaneo.



El ADC puede ser configurado de tal forma que el hardware realiza todo el proceso de adquisición evitando hacer por software, lo único que se hace por software es la lectura de los buffer del ADC *ADCBUF0:F*.

Por ejemplo:
Configurar el ADC para muestrear una señal en el canal *AN0* usando un solo canal *CH0* a *2,6 Khz* automáticamente, usando el *TMR3* como fuente de trigger y que genere una interrupción cada 16 muestras.

El código de abajo inicializa el ADC el delay de muestreo es definido por el *TMR3* cada 16 muestras se genera una interrupción, las 16 muestras deben ser leídas en la rutina de servicio de interrupción.

```
#define FOSC    7372800
#define PLL     16
#define FCY     FOSC*PLL/4
#define SAMPLINGRATE    2600
#define SAMPCOUNT       (FCY/SAMPLINGRATE)+1

void Init_ADC(void)
{
        // ADC configuration
        ADCON1bits.ADON = 0;
        ADCON1bits.ADSIDL = 0;
        ADCON1bits.FORM = 0;
        ADCON1bits.SSRC = 2;
        ADCON1bits.SIMSAM = 0;
        ADCON1bits.ASAM = 1;
        ADCON2bits.VCFG = 0;
        ADCON2bits.CHPS = 0;
        ADCON2bits.SMPI = 15;
        ADCON3bits.ADCS = 9;   // Tad  min 165 ns
        ADCON3bits.SAMC = 1;
        ADCHS = 0b0000000000100000;
        ADCSSL = 0x0000;
        ADPCFG = 0xFFFF;
        ADPCFGbits.PCFG0 = 0;

        // timer 3 configuration
        TMR3 = 0x0000;
        PR3 = SAMPCOUNT;
        IFS0bits.T3IF = 0;
        IEC0bits.T3IE = 0;
```

```
        IFS0bits.ADIF = 0;
        IEC0bits.ADIE = 1;
        ADCON1bits.ADON = 1;
        T3CONbits.TON = 1;
}
```

En el manual de la familia *dsPIC30F* vienen más ejemplos de configuración del ADC.

Set de instrucciones ASM

El set de instrucciones es muy parecido al de los *PICmicro* manteniendo así una fácil migración, son 84 instrucciones.
La mayoría de instrucciones requieren solo de una localización de memoria.

El set de instrucciones esta dividido en 5 categorías:

- Operaciones orientadas a bytes
- Operaciones orientadas a bits
- Operaciones literales
- Operaciones DSP
- Operaciones de control

Instrucciones matemáticas

Assembly Syntax		Description	Words	Cycles
ADD	f {,WREG}[1]	Destination = f + WREG	1	1
ADD	#lit10,Wn	Wn = lit10 + Wn	1	1
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	1	1
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1
ADDC	f {,WREG}[1]	Destination = f + WREG + (C)	1	1
ADDC	#lit10,Wn	Wn = lit10 + Wn + (C)	1	1
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	1	1
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1
DAW.B	Wn	Wn = decimal adjust Wn	1	1
DEC	f {,WREG}[1]	Destination = f – 1	1	1
DEC	Ws,Wd	Wd = Ws – 1	1	1
DEC2	f {,WREG}[1]	Destination = f – 2	1	1
DEC2	Ws,Wd	Wd = Ws – 2	1	1
DIV.S	Wm, Wn	Signed 16/16-bit integer divide	1	18[2]
DIV.SD	Wm, Wn	Signed 32/16-bit integer divide	1	18[2]
DIV.U	Wm, Wn	Unsigned 16/16-bit integer divide	1	18[2]
DIV.UD	Wm, Wn	Unsigned 32/16-bit integer divide	1	18[2]
DIVF	Wm, Wn	Signed 16/16-bit fractional divide	1	18[2]
INC	f {,WREG}[1]	Destination = f + 1	1	1
INC	Ws,Wd	Wd = Ws + 1	1	1
INC2	f {,WREG}[1]	Destination = f + 2	1	1
INC2	Ws,Wd	Wd = Ws + 2	1	1
MUL	f	W3:W2 = f * WREG	1	1
MUL.SS	Wb,Ws,Wnd	{Wnd + 1,Wnd} = sign(Wb) * sign(Ws)	1	1
MUL.SU	Wb,#lit5,Wnd	{Wnd + 1,Wnd} = sign(Wb) * unsign(lit5)	1	1
MUL.SU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = sign(Wb) * unsign(Ws)	1	1
MUL.US	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsign(Wb) * sign(Ws)	1	1
MUL.UU	Wb,#lit5,Wnd	{Wnd + 1,Wnd} = unsign(Wb) * unsign(lit5)	1	1
MUL.UU	Wb,Ws,Wnd	{Wnd + 1,Wnd} = unsign(Wb) * unsign(Ws)	1	1
SE	Ws,Wnd	Wnd = sign-extended Ws	1	1
SUB	f {,WREG}[1]	Destination = f – WREG	1	1
SUB	#lit10,Wn	Wn = Wn – lit10	1	1
SUB	Wb,#lit5,Wd	Wd = Wb – lit5	1	1
SUB	Wb,Ws,Wd	Wd = Wb – Ws	1	1
SUBB	f {,WREG}[1]	Destination = f – WREG – $(\overline{C})$	1	1
SUBB	#lit10,Wn	Wn = Wn – lit10 – $(\overline{C})$	1	1
SUBB	Wb,#lit5,Wd	Wd = Wb – lit5 – $(\overline{C})$	1	1
SUBB	Wb,Ws,Wd	Wd = Wb – Ws – $(\overline{C})$	1	1
SUBBR	f {,WREG}[1]	Destination = WREG – f – $(\overline{C})$	1	1
SUBBR	Wb,#lit5,Wd	Wd = lit5 – Wb – $(\overline{C})$	1	1
SUBBR	Wb,Ws,Wd	Wd = Ws – Wb – $(\overline{C})$	1	1
SUBR	f {,WREG}[1]	Destination = WREG – f	1	1
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	1	1
SUBR	Wb,Ws,Wd	Wd = Ws – Wb	1	1
ZE	Ws,Wnd	Wnd = zero-extended Ws	1	1

Instrucciones lógicas

Assembly Syntax		Description	Words	Cycles
AND	f {,WREG}(see Note)	Destination = f .AND. WREG	1	1
AND	#lit10,Wn	Wn = lit10 .AND. Wn	1	1
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	1	1
AND	Wb,Ws,Wd	Wd = Wb .AND. Ws	1	1
CLR	f	f = 0x0000	1	1
CLR	WREG	WREG = 0x0000	1	1
CLR	Wd	Wd = 0x0000	1	1
COM	f {,WREG}(see Note)	Destination = $\overline{f}$	1	1
COM	Ws,Wd	Wd = $\overline{Ws}$	1	1
IOR	f {,WREG}(see Note)	Destination = f .IOR. WREG	1	1
IOR	#lit10,Wn	Wn = lit10 .IOR. Wn	1	1
IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	1	1
IOR	Wb,Ws,Wd	Wd = Wb .IOR. Ws	1	1
NEG	f {,WREG}(see Note)	Destination = $\overline{f}$ + 1	1	1
NEG	Ws,Wd	Wd = $\overline{Ws}$ + 1	1	1
SETM	f	f = 0xFFFF	1	1
SETM	WREG	WREG = 0xFFFF	1	1
SETM	Wd	Wd = 0xFFFF	1	1
XOR	f {,WREG}(see Note)	Destination = f .XOR. WREG	1	1
XOR	#lit10,Wn	Wn = lit10 .XOR. Wn	1	1
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1
XOR	Wb,Ws,Wd	Wd = Wb .XOR. Ws	1	1

Instrucciones de rotación/cambio

Assembly Syntax		Description	Words	Cycles
ASR	f {,WREG}(see Note)	Destination = arithmetic right shift f	1	1
ASR	Ws,Wd	Wd = arithmetic right shift Ws	1	1
ASR	Wb,#lit4,Wnd	Wnd = arithmetic right shift Wb by lit4	1	1

ASR	Wb,Wns,Wnd	Wnd = arithmetic right shift Wb by Wns	1	1
LSR	f {,WREG}(see Note)	Destination = logical right shift f	1	1
LSR	Ws,Wd	Wd = logical right shift Ws	1	1
LSR	Wb,#lit4,Wnd	Wnd = logical right shift Wb by lit4	1	1
LSR	Wb,Wns,Wnd	Wnd = logical right shift Wb by Wns	1	1
RLC	f {,WREG}(see Note)	Destination = rotate left through Carry f	1	1
RLC	Ws,Wd	Wd = rotate left through Carry Ws	1	1
RLNC	f {,WREG}(see Note)	Destination = rotate left (no Carry) f	1	1
RLNC	Ws,Wd	Wd = rotate left (no Carry) Ws	1	1
RRC	f {,WREG}(see Note)	Destination = rotate right through Carry f	1	1
RRC	Ws,Wd	Wd = rotate right through Carry Ws	1	1
RRNC	f {,WREG}(see Note)	Destination = rotate right (no Carry) f	1	1
RRNC	Ws,Wd	Wd = rotate right (no Carry) Ws	1	1
SL	f {,WREG}(see Note)	Destination = left shift f	1	1
SL	Ws,Wd	Wd = left shift Ws	1	1
SL	Wb,#lit4,Wnd	Wnd = left shift Wb by lit4	1	1
SL	Wb,Wns,Wnd	Wnd = left shift Wb by Wns	1	1

Instrucciones de bits

Assembly Syntax		Description	Words	Cycles
BCLR	f,#bit4	Bit clear f	1	1
BCLR	Ws,#bit4	Bit clear Ws	1	1
BSET	f,#bit4	Bit set f	1	1
BSET	Ws,#bit4	Bit set Ws	1	1
BSW.C	Ws,Wb	Write C bit to Ws<Wb>	1	1
BSW.Z	Ws,Wb	Write $\overline{Z}$ bit to Ws<Wb>	1	1
BTG	f,#bit4	Bit toggle f	1	1
BTG	Ws,#bit4	Bit toggle Ws	1	1
BTST	f,#bit4	Bit test f	1	1
BTST.C	Ws,#bit4	Bit test Ws to C	1	1
BTST.Z	Ws,#bit4	Bit test Ws to Z	1	1
BTST.C	Ws,Wb	Bit test Ws<Wb> to C	1	1
BTST.Z	Ws,Wb	Bit test Ws<Wb> to Z	1	1
BTSTS	f,#bit4	Bit test f then set f	1	1
BTSTS.C	Ws,#bit4	Bit test Ws to C then set Ws	1	1
BTSTS.Z	Ws,#bit4	Bit test Ws to Z then set Ws	1	1
FBCL	Ws,Wnd	Find bit change from left (MSb) side	1	1
FF1L	Ws,Wnd	Find first one from left (MSb) side	1	1
FF1R	Ws,Wnd	Find first one from right (LSb) side	1	1

Instrucciones de control de flujo

Assembly Syntax		Description	Words	Cycles(see Note)
BTSC	f,#bit4	Bit test f, skip if clear	1	1 (2 or 3)
BTSC	Ws,#bit4	Bit test Ws, skip if clear	1	1 (2 or 3)
BTSS	f,#bit4	Bit test f, skip if set	1	1 (2 or 3)
BTSS	Ws,#bit4	Bit test Ws, skip if set	1	1 (2 or 3)
CP	f	Compare (f – WREG)	1	1
CP	Wb,#lit5	Compare (Wb – lit5)	1	1
CP	Wb,Ws	Compare (Wb – Ws)	1	1
CP0	f	Compare (f – 0x0000)	1	1
CP0	Ws	Compare (Ws – 0x0000)	1	1
CPB	f	Compare with Borrow (f – WREG – $\overline{C}$)	1	1
CPB	Wb,#lit5	Compare with Borrow (Wb – lit5 – $\overline{C}$)	1	1
CPB	Wb,Ws	Compare with Borrow (Wb – Ws – $\overline{C}$)	1	1
CPSEQ	Wb, Wn	Compare (Wb – Wn), skip if =	1	1 (2 or 3)
CPSGT	Wb, Wn	Compare (Wb – Wn), skip if >	1	1 (2 or 3)
CPSLT	Wb, Wn	Compare (Wb – Wn), skip if <	1	1 (2 or 3)
CPSNE	Wb, Wn	Compare (Wb – Wn), skip if ≠	1	1 (2 or 3)

Assembly Syntax		Description	Words	Cycles
BRA	Expr	Branch unconditionally	1	2
BRA	Wn	Computed branch	1	2
BRA	C,Expr	Branch if Carry (no Borrow)	1	1 (2)(1)
BRA	GE,Expr	Branch if greater than or equal	1	1 (2)(1)
BRA	GEU,Expr	Branch if unsigned greater than or equal	1	1 (2)(1)
BRA	GT,Expr	Branch if greater than	1	1 (2)(1)
BRA	GTU,Expr	Branch if unsigned greater than	1	1 (2)(1)
BRA	LE,Expr	Branch if less than or equal	1	1 (2)(1)
BRA	LEU,Expr	Branch if unsigned less than or equal	1	1 (2)(1)
BRA	LT,Expr	Branch if less than	1	1 (2)(1)
BRA	LTU,Expr	Branch if unsigned less than	1	1 (2)(1)
BRA	N,Expr	Branch if Negative	1	1 (2)(1)
BRA	NC,Expr	Branch if not Carry (Borrow)	1	1 (2)(1)
BRA	NN,Expr	Branch if not Negative	1	1 (2)(1)
BRA	NOV,Expr	Branch if not Overflow	1	1 (2)(1)
BRA	NZ,Expr	Branch if not Zero	1	1 (2)(1)
BRA	OA,Expr	Branch if Accumulator A Overflow	1	1 (2)(1)
BRA	OB,Expr	Branch if Accumulator B Overflow	1	1 (2)(1)
BRA	OV,Expr	Branch if Overflow	1	1 (2)(1)
BRA	SA,Expr	Branch if Accumulator A Saturate	1	1 (2)(1)
BRA	SB,Expr	Branch if Accumulator B Saturate	1	1 (2)(1)
BRA	Z,Expr	Branch if Zero	1	1 (2)(1)
CALL	Expr	Call subroutine	2	2
CALL	Wn	Call indirect subroutine	1	2

DO	#lit14,Expr	Do code through PC + Expr, (lit14 + 1) times	2	2
DO	Wn,Expr	Do code through PC+Expr, (Wn + 1) times	2	2
GOTO	Expr	Go to address	2	2
GOTO	Wn	Go to address indirectly	1	2
RCALL	Expr	Relative call	1	2
RCALL	Wn	Computed call	1	2
REPEAT	#lit14	Repeat next instruction (lit14 + 1) times	1	1
REPEAT	Wn	Repeat next instruction (Wn + 1) times	1	1
RETFIE		Return from interrupt enable	1	3 (2)[(2)]
RETLW	#lit10,Wn	Return with lit10 in Wn	1	3 (2)[(2)]
RETURN		Return from subroutine	1	3 (2)[(2)]

Instrucciones de control y snack

Assembly Syntax		Description	Words	Cycles
LNK	#lit14	Link Frame Pointer	1	1
POP	f	POP TOS to f	1	1
POP	Wd	POP TOS to Wd	1	1
POP.D	Wnd	Double POP from TOS to Wnd:Wnd + 1	1	2
POP.S		POP shadow registers	1	1
PUSH	f	PUSH f to TOS	1	1
PUSH	Ws	PUSH Ws to TOS	1	1
PUSH.D	Wns	PUSH double Wns:Wns + 1 to TOS	1	2
PUSH.S		PUSH shadow registers	1	1
ULNK		Unlink Frame Pointer	1	1

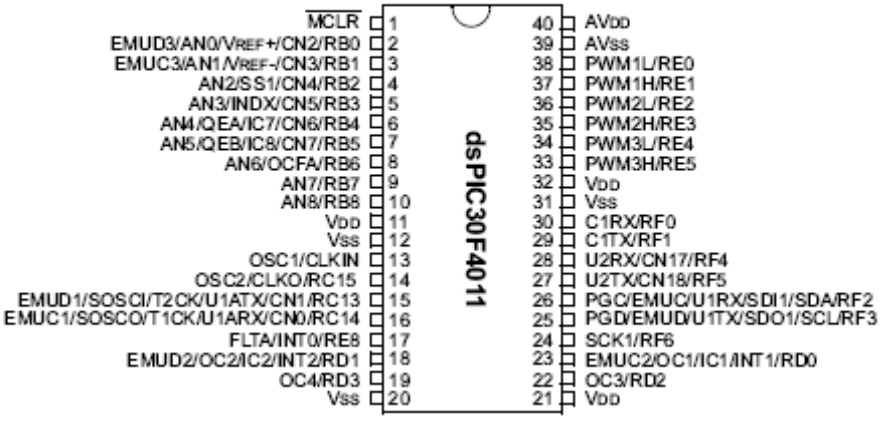
Assembly Syntax		Description	Words	Cycles
CLRWDT		Clear Watchdog Timer	1	1
DISI	#lit14	Disable interrupts for (lit14 + 1) instruction cycles	1	1
NOP		No operation	1	1
NOPR		No operation	1	1
PWRSAV	#lit1	Enter Power-saving mode lit1	1	1
Reset		Software device Reset	1	1

Instrucciones de DSP

Assembly Syntax		Description	Words	Cycles
ADD	Acc	Add accumulators	1	1
ADD	Ws,#Slit4,Acc	16-bit signed add to Acc	1	1
CLR	Acc,Wx,Wxd,Wy,Wyd,AWB	Clear Acc	1	1
ED	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance (no accumulate)	1	1
EDAC	Wm*Wm,Acc,Wx,Wy,Wxd	Euclidean distance	1	1
LAC	Ws,#Slit4,Acc	Load Acc	1	1
MAC	Wm*Wn,Acc,Wx,Wxd,Wy, Wyd,AWB	Multiply and accumulate	1	1
MAC	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square and accumulate	1	1
MOVSAC	Acc,Wx,Wxd,Wy,Wyd,AWB	Move Wx to Wxd and Wy to Wyd	1	1
MPY	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	Multiply Wn by Wm to Acc	1	1
MPY	Wm*Wm,Acc,Wx,Wxd,Wy,Wyd	Square to Acc	1	1
MPY.N	Wm*Wn,Acc,Wx,Wxd,Wy,Wyd	-(Multiply Wn by Wm) to Acc	1	1
MSC	Wm*Wn,Acc,Wx,Wxd,Wy, Wyd,AWB	Multiply and subtract from Acc	1	1
NEG	Acc	Negate Acc	1	1
SAC	Acc,#Slit4,Wd	Store Acc	1	1
SAC.R	Acc,#Slit4,Wd	Store rounded Acc	1	1
SFTAC	Acc,#Slit6	Arithmetic shift Acc by Slit6	1	1
SFTAC	Acc,Wn	Arithmetic shift Acc by (Wn)	1	1
SUB	Acc	Subtract accumulators	1	1

Diagrama de Pines del dsPIC 30F4011

40-Pin PDIP



4.1 Construcción de un KIT de entrenamiento básico para DSP

Para desarrollar la parte práctica de esta tesis se a decidido hacer un KIT de entrenamiento básico para DSP, el cual esta basado en los KIT's que ofrece MICROCHIP en su pagina web pero adaptado a nuestras necesidades y con componentes fáciles de encontrar en las tiendas locales de electrónica haciéndolo más sencillo y asequible.

El KIT de entrenamiento es de carácter didáctico y entre sus aplicaciones principales están:

- Adaptación y control de diferentes tipos de sensores
- Manejo de motores (*DC - PAP*) sistemas rotacionales con PWM y QEI.
- Filtrado digital de señales de voz (***10 Hz – 4 kHz speech***).
- Para Robótica
- Para laboratorio de DSP
- Para laboratorio de Instrumentación

4.1.1 Características generales

La tarjeta tiene las siguientes características:

- Tarjeta de bajo coste y dimensiones reducidas
- Alimentación a 12 VAC mediante transformador de red o batería de plomo recargable de 12VDC.
- Estabilización en la alimentación con diodo y filtro.
- Zócalo de 40 pines para el *dsPIC 30F4011* o un compatible, zócalo para un *PICmicro PIC18F2550* programado con un **firmware** para comunicación USB 2.0 con la *PC* y manejo de LCD's gráficas o matriciales.
- Jack 3.5mm de entrada de señales con filtro antialiasing de 4Khz y offset de 2.5V.
- Jack 3.5mm de salida de señales analógicas generadas con PWM con un filtro antialiasing de 4Khz.
- Conector para un sensor de temperatura LM335 con jumper para seleccionar o no un filtro RC.
- Conector para canales de *PWM*

- 6 Leds
- 3 pulsantes
- Jumpers para selección de comunicación con **firmware (***SPI* o *RS232*)
- Jumpers para selección de *UART1* o *UART2*.
- Conectores para expansión del sistema
- Un conector para reprogramación *ICSP* del *dsPIC* sin desmontarlo de la tarjeta
- Conector USB y DB9 para comunicaciones con PC
- 1 Conector 20 pines para conectar pantallas LCD con jumpers para seleccionar el uso de una pantalla gráfica o matricial.
- Switch para apagar el sistema
- 1 Pulsante que resetea todo el sistema
- 1 Zócalo para una memoria serial I2C

4.1.2 Listado de Materiales

Abajo se muestra la lista de materiales para armar la tarjeta ***dsPIC-4ks.***

Referencia	Descripción	Valor
C1	Capacitor Polarizado	10uF
C2	Capacitor Polarizado	10uF
C3	Capacitor Polarizado	10uF
C4	Capacitor Polarizado	10uF
C5	Capacitor Polarizado	10uF
C6	Capacitor	0.01uF
C7	Capacitor	0.0022uF
C8	Capacitor	0.01uF
C9	Capacitor Polarizado	10uF
C10	Capacitor	0.01uF
C11	Capacitor	0.0022uF
C12	Capacitor	22pf
C13	Capacitor	22pf
C14	Capacitor	22pf
C15	Capacitor	22pf
C16	Capacitor	47uf
C17	Capacitor	10uF
C18	Capacitor	10uF
C19	Capacitor Polarizado	47uf
C20	Capacitor	0.1uF
C21	Capacitor	0.1uF
D1	Diodo	1N4007
J1	Jumper	
J2	Jumper	
J3	Conector DB9 para PCB	
J4	Jumper	
J5	Jumper	
J6	Jack 3.5 mm	
J7	Jumper	
J8	Jumper	
J9	Jack 3.5 mm	

J10	Jumper	
J11	Jumper	
J12	Jumper	
J13	Jumper	
J14	Jumper	
J15	Jumper	
J16	Peineta, 2 pines	
J17	Conector ICD 20 pines	
J18	Jumper	
J19	Jack fuente	
JP1	Conector, 3-Pines	
JP2	Conector, 20 pines	
JP3	Peineta, 4 pines	
JP4	Peineta, 5-Pin	
JP5	Conector, 16 pines	
JP6	Conector, 16 pines	
JP7	Receptáculo USB tipo B	
LED1	Led 3mm	
LED2	Led 3mm	
LED3	Led 3mm	
LED4	Led 3mm	
LED5	Led 3mm	
LED6	Led 3mm	
LED7	Led 3mm	
Q1	Transistor NPN propósito general	
R1	Resistencia	4.7k
R2	Resistencia	4.7k
R3	Resistencia	4.7k
R4	Resistencia	470
R5	Resistencia	2k
R6	Resistencia	390ohm
R7	Resistencia	390ohm
R8	Resistencia	390ohm
R9	Resistencia	390ohm
R10	Resistencia	390ohm
R11	Resistencia	390ohm
R12	Resistencia	1K
R13	Resistencia	10K
R14	Resistencia	10K
R15	Resistencia	10K
R16	Resistencia	10k
R17	Resistencia	10K
R18	Resistencia	10K
R19	Resistencia	2.2K
R20	Resistencia	10k
R21	Resistencia	10K
R22	Resistencia	10K
R23	Resistencia	2K
R24	Resistencia	300K
R25	Potentiometro	50K
R26	Resistencia	10K
R27	Resistencia	10K

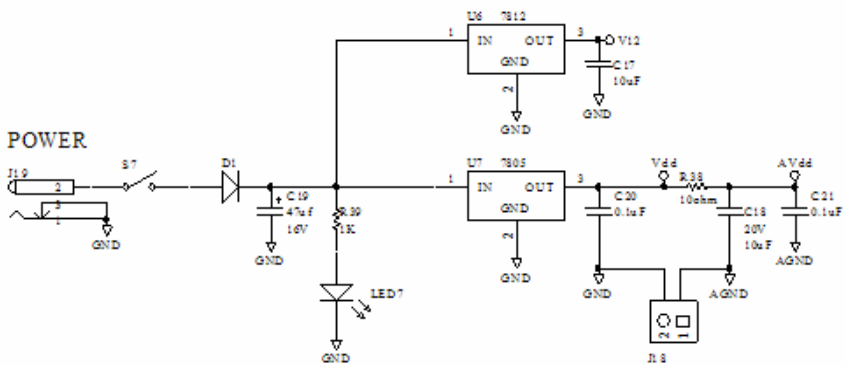
R28	Resistencia	2.2K
R29	Resistencia	10K
R30	Resistencia	10K
R31	Resistencia	10K
R32	Resistencia	4.7k
R33	Resistencia	100ohm
R34	Resistencia	100ohm
R35	Resistencia	10ohm
R36	Resistencia	1K
R37	Potentiometro	10K
R38	Resistencia	10ohm
R39	Resistencia	1K
S1	Pulsante	
S2	Pulsante	
S3	Pulsante	
S4	Pulsante	
S5	DIP Switch	
S6	Pulsante	
S7	Switch	
U1	MAX 232	
U2	512Kbit Serial I²C Memoria EEPROM	
U3	LM324	
U4	DSC dsPIC30F4011	
U5	PICmicro PIC18F2550	
U6	Regulador 7812	
U7	Regulador 7805	
Y1	Cristal	7.3728Mhz
Y2	Cristal	20Mhz

4.1.3 Descripción funcional

Descripción del hardware de la tarjeta ***dsPIC-4ks.***

Fuente Alimentación

La tarjeta ***dsPIC-4ks*** posee una fuente integrada con voltajes regulados a *5V* y *12V*, el esquema de la fuente se muestra en la figura de abajo:

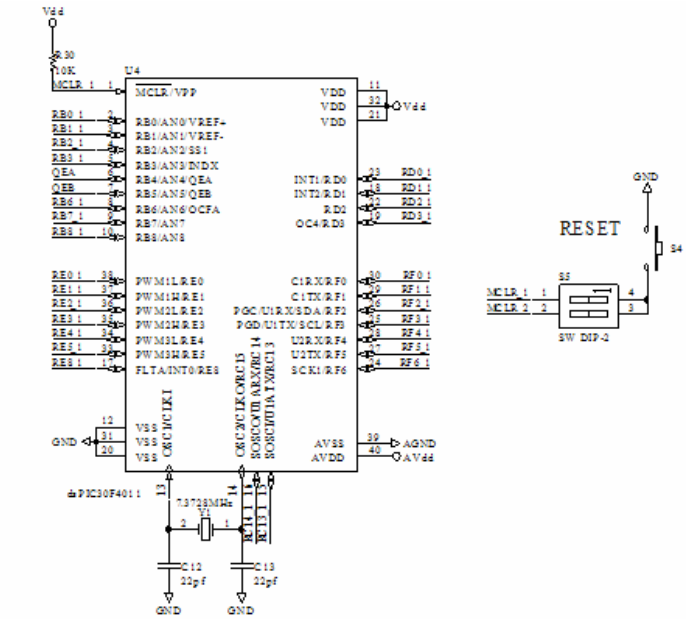


Mediante el switch ***S7*** se enciende y se apaga toda la tarjeta, el jumper ***J18*** une el retorno de la tierra analógica.

El conector ***J19*** es la entrada para la fuente de alimentación +15VAC (transformador), el ***Led 7*** indica cuando esta con tensión la tarjeta.

Controlador digital de señales dsPIC

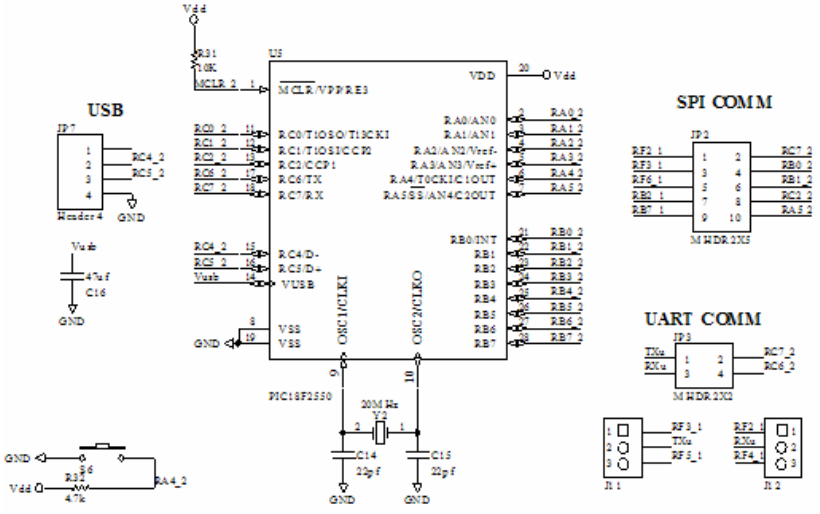
El corazón de la tarjeta es un controlador digital *dsPIC 30F4011*, según el programa grabado se podrá utilizar en diferentes aplicaciones, en la figura de abajo se muestra un esquema del dsPIC.



El dipswitch ***S5*** selecciona el reseteo del *dsPIC* y del *PICmicro*, que puede ser ambos al mismo tiempo, el pulsante ***S4*** resetea según la selección de ***S5***.
El sistema funciona a *120Mhz* que se obtiene usando un multiplicador de frecuencia a partir de los *7.3728Mhz*.

Firmware USB

Se utiliza un PICmicro PIC18F2550 para la comunicación USB de la tarjeta con la PC, el PICmicro tiene cargado un firmware por default (*PICDEM USB*) para funcionar entre puente entre la PC y dsPIC, se pueden mandar máximo 12000 caracteres por segundo.

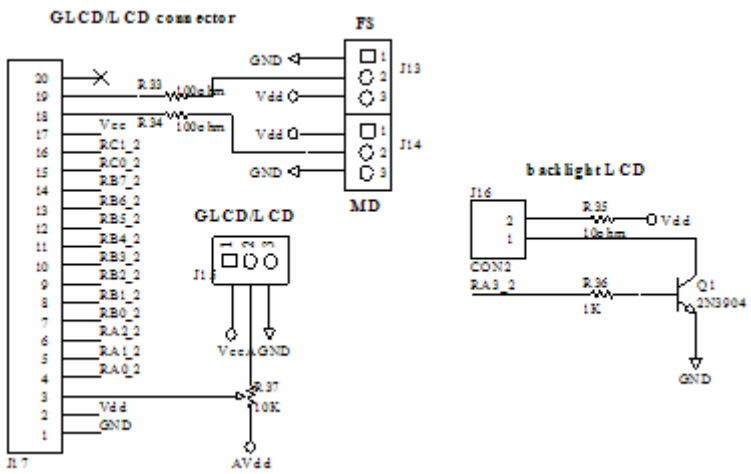


En ***JP2*** se ubican los jumpers para comunicación SPI entre el firmware y el dsPIC, ***JP3*** sirve para comunicar el firmware con el dsPIC mediante comunicación RS232, ***J11*** y ***J12*** sirven para seleccionar la UART del dsPIC que se va a utilizar para comunicarse con el firmware.
El pulsante ***S6*** cuando se mantiene pulsado durante un reset del sistema entra en modo de Bootloader esto quiere decir que puede ser sustituido el firmware con uno nuevo, para mas información sobre el Bootloader USB buscar en la pagina web www.microchip.com

Nunca pueden estar jumpers en ***JP2*** y ***JP3*** al mismo tiempo

Conector pantallas LCD

Conector de 20 pines compatible con pantallas LCD que tengan el controlador Hitachi y para pantallas LCD gráficas con controlador Toshiba T6963C.

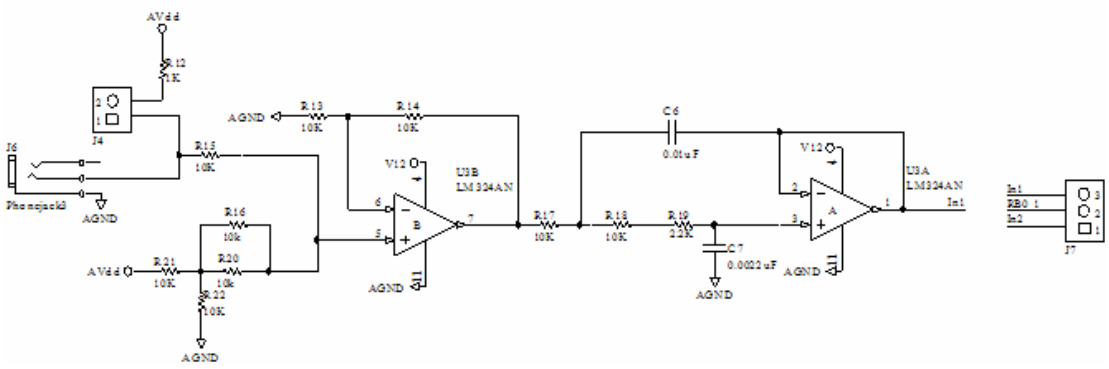


El jumper ***J15*** selecciona el LCD a utilizar (matricial o gráfico), ***J13*** y ***J14*** sirven para configurar el tamaño de fuente en la LCD gráfica, en ***J16*** se conecta el back Light del LCD si este tiene.

Revisar el manual de cada LCD para ver la disposición de pines.
En la figura de abajo se muestra un esquema del conector, para que funcionen las pantallas LCD, debe estar instalado el respectivo controlador en el firmware.

Jack de entrada para señales analógicas

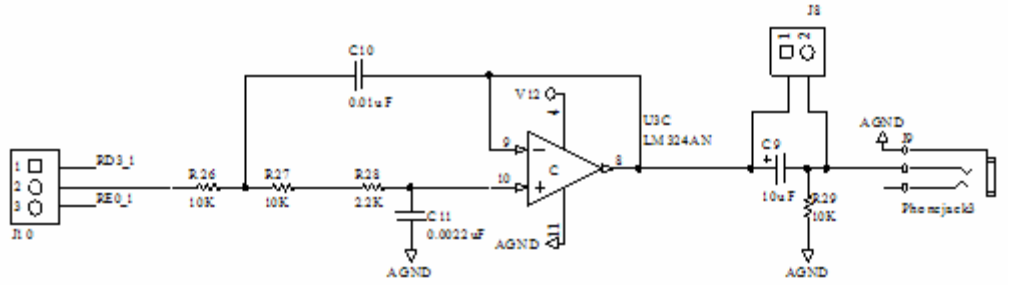
Este jack sirve para la entrada de señales analógicas de máximo *2.5 Vpp*, esta diseñado para acoplar las señales de entrada de la tarjeta de sonido del PC, las señales de la tarjeta de sonido del *PC* son de máximo 1.5Vpp. Este canal tiene un offset de 2.5V y un filtro antialiasing de 4Khz de tipo Chebyshev. El esquema es mostrado en la siguiente figura:



Cuando se conecte un micrófono el jumper ***J4*** deberá estar puenteado, el jumper ***J7*** selecciona la entrada de AN0.

Jack de salida de señales analógicas generadas con PWM

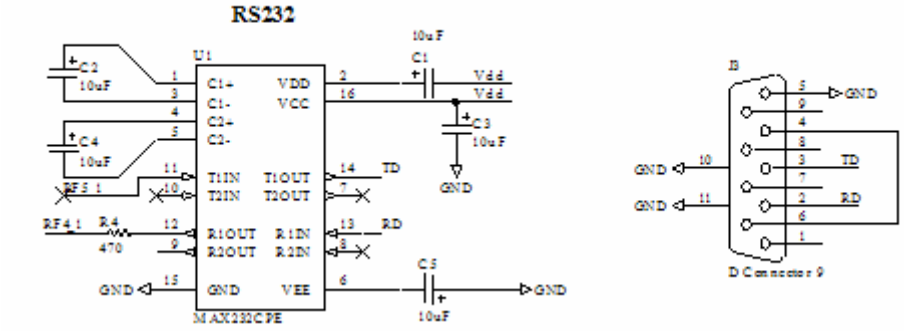
Este jack sirve para la salida de señales analógicas generadas con PWM, tiene un filtro antialiasing de 4Khz de tipo Chebyshev, la señal de salida no debe ser conectada directamente a la línea de entrada de la tarjeta de sonido del PC debe ser acoplada previamente. El esquema se muestra abajo:



J10 selecciona la fuente de PWM, ***J8*** activa o desactiva canal para acoplar a unos parlantes.

Conector RS232

La tarjeta ***dsPIC-4ks*** posee un puerto RS232 que funciona sin firmware, se maneja directamente desde el dsPIC, en el esquema de abajo se muestra este puerto:

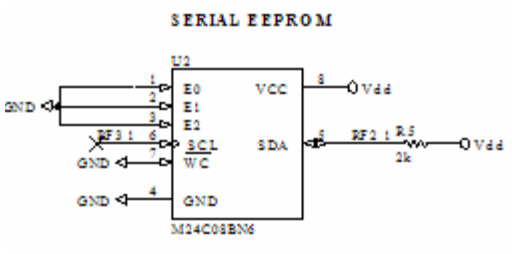


El CI MAX232 acopla las señales para la comunicación con el PC.

Memoria serial EEPROM

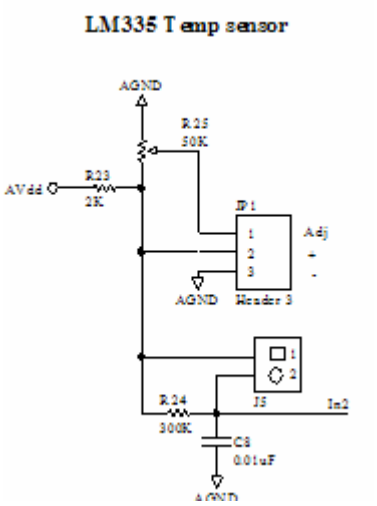
La tarjeta ***dsPIC-4ks*** posee un zócalo para una memoria serial EEPROM –I2C de las series 24XXX de Microchip o compatible, con el uso de una memoria se puede aumentar la velocidad de muestreo cuando se interfase la tarjeta con la PC.

En el esquema de abajo se muestra la memoria:



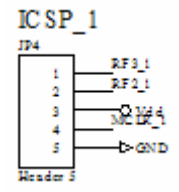
Conector para sensor de temperatura

El conector ***JP1*** sirve para conectar un sensor de temperatura LM335, en el esquema de abajo se ve el acoplamiento para el sensor, el potenciómetro ***R25*** sirve para ajustar el sensor, el jumper ***J5*** activa o desactiva un filtro RC a la entrada del sensor de temperatura.



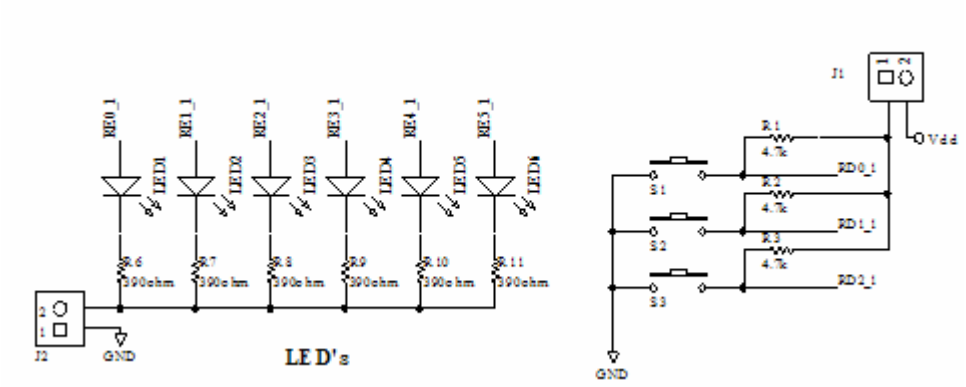
Conector para reprogramación del dsPIC

El conector ***JP4*** sirve para reprogramar el dsPIC sin desmontarlo de la tarjeta (usando in circuit serial programming ICSP), en la figura de abajo se muestra el conector:



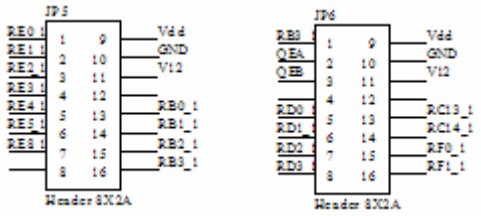
LEDs y Pulsantes

La tarjeta posee también los periféricos clásicos utilizados en microcontroladores los " leds y los pulsantes " para desarrollar aplicaciones, en el esquema de abajo se muestran estos periféricos:



Conectores de expansión

La tarjeta ***dsPIC-4ks*** posee 2 conectores (***JP5, JP6***) para expandir el sistema, permitiendo conectar nuevos periféricos, en el esquema de abajo se muestran los conectores de expansión:



4.2 Introducción a las herramientas de desarrollo

El proceso de desarrollo de una aplicación con microcontroladores puede ser dividido en tres etapas:

- Escribir código
- Debugging del código
- Programación del microcontrolador

Para esto se necesita de una herramienta que sirvan para cada función, la herramienta principal que soporta estas funciones es el ambiente integrado de desarrollo *MPLAB IDE*.

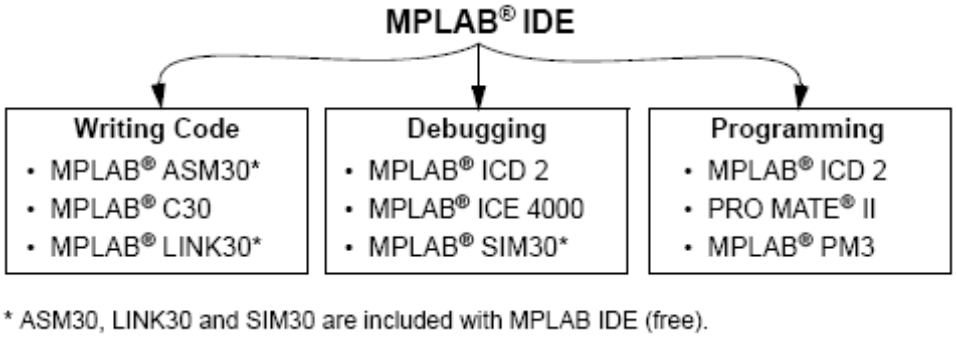
Para el desarrollo de aplicaciones para la PC se ha optado por usar *MATLAB*, *MATLAB* es un entorno de computación y desarrollo de aplicaciones integrado orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos, existen otros CADs para utilizar en *DS*P como son:

Labview, *Octave* y *Scilab*

Otro software de propósito general para procesamiento de señales es *dsPICworks*, en el se pueden generar señales y se pueden probar algoritmos como la FFT o un filtro digital.

4.2.1 MPLAB IDE

El *MPLAB IDE* permite crear un proyecto de inicio a fin en el mismo ambiente o con la misma aplicación, ya que no se necesita de un editor, un ensamblador, un compilador o un programador. La figura de abajo muestra los procesos que maneja el ambiente integrado.



MPLAB IDE esta disponible gratuitamente en la página de Microchip.

4.2.2 MPLAB C30

En *DSP* la mayoría de código que se escribe es escrito en "C", debido al fácil mantenimiento de funciones, la portabilidad y por ser mas fácil de entender que el lenguaje ensamblador. Esto no quiere decir que lenguaje ensamblador sea obsoleto, este se utiliza en rutinas donde se requiera optimizar y mejorar el rendimiento del código, el compilador de ensamblador que viene con *MPLAB IDE* se llama *ASM30*.

El compilador de *C* para los *dsPIC* es el *MPLAB C30*, es un poderoso compilador que tiene sus propias librerías de funciones como son los algoritmos de la FFT y filtros digitales que pueden ser muy engorrosos en lenguaje ensamblador. En la página web de Microchip se puede bajar la versión para estudiantes de este compilador.

4.2.3 MPLAB C18

Los microcontroladores *PICmicro* de la familia *PIC18* vienen con una arquitectura optimizada para ser programados en lenguaje C, el compilador de C para los *PICmicro* es el *MPLAB C18*, este compilador también posee sus propias librerías, se puede también bajar una versión para estudiantes de la página web de Microchip.

4.2.4 MATLAB

Matlab es el nombre abreviado de " *MATrix LABoratory*", es un programa para realizar cálculos numéricos con matrices y vectores. Matlab resulto ser un programa

muy atractivo para desarrollar esta tesis por tener varias características como son: el poder hacer gráficos fácilmente, por tener su propio lenguaje de programación, por poder compilar funciones en C y poder utilizarlas desde el mismo programa, por permitir el manejo de puertos de la PC y por su ya gran número de funciones para DSP disponibles.
Para ciertas operaciones resulta ser bastante rápido pero para otras es muy lento, sin embargo es bastante fácil de utilizar.

4.2.5 dsPIC WORKS

dsPICworks es un software de propósito general para procesamiento de señales, este combina la funcionalidad y sofisticación para llevar a cabo tareas complejas sin tener que recurrir al arduo trabajo de la matemática compleja, este software permite interfasar con el *MPLAB IDE,* puede conseguirse gratuitamente en la pagina web de Microchip.

4.2.6 dsPIC Filter Design Lite

dsPIC FD Lite es un software intuitivo para diseñar filtros digitales, este nos genera ya el código para implementar en los dsPIC los filtros ahorrándonos mucho tiempo de diseño, la desventaja es que tiene un límite de orden para generar los filtros. Este software no es gratuito y puede obtenerse en la página web de Microchip.
Este presenta de una manera gráfica la respuesta a la frecuencia, la respuesta al escalón, la fase, la respuesta al impulso de un filtro, también pueden obtenerse los coeficientes de la función de transferencia de algunos tipos de filtros digitales.

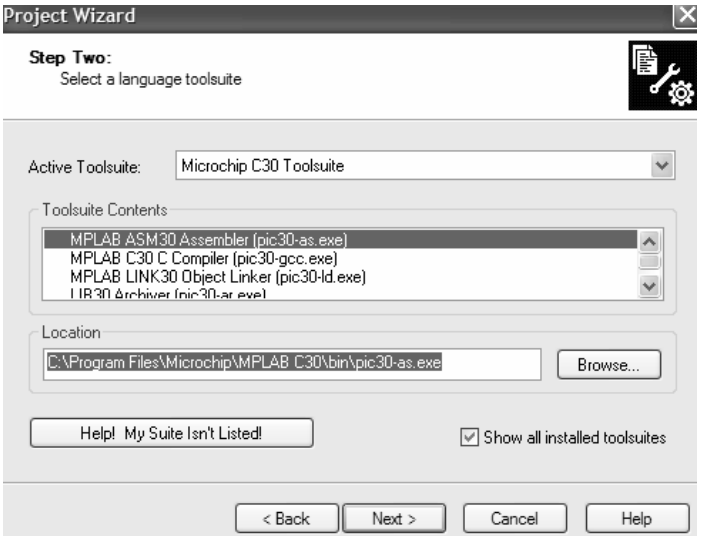
4.3 Creando un proyecto y simulando un dsPIC con MPLAB

Antes de empezar se asume que se tiene instalado en el PC el ambiente de desarrollo *MPLAB IDE* y el compilador *MPLAB C30*.

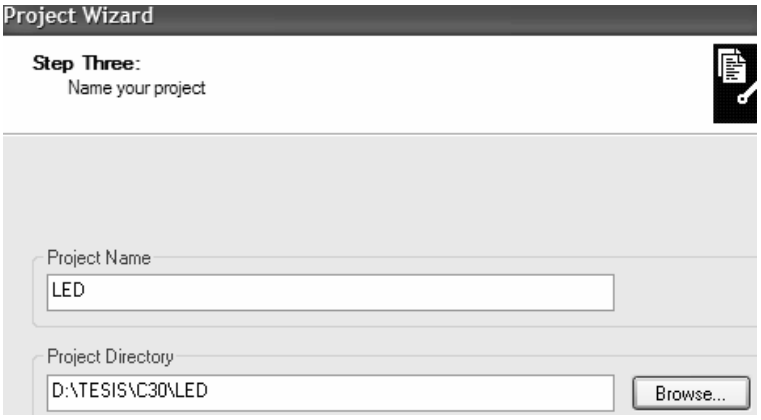
Lo primero que se tiene que hacer es crear una carpeta, por ejemplo llamaremos a una carpeta “ LED” y en ella haremos una aplicación para un dsPIC que encienda y apague un led.

Abrimos *MPLAB IDE* y creamos un nuevo proyecto para el dsPIC30F4011 usando el Project Wizard

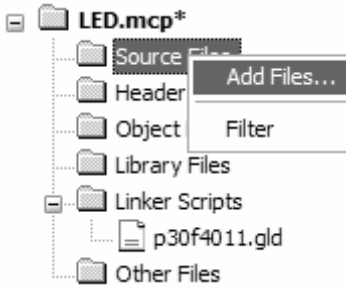
A continuación se debe seleccionar al compilador MPLAB C30 para escribir el código en C, si se desea escribir una aplicación usando ensamblador se debe escoger el compilador ASM30

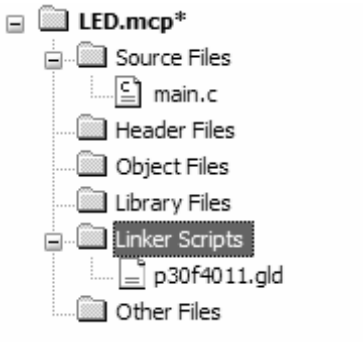


A continuación hay que poner un nombre al proyecto y dar una carpeta en la cual trabajaremos, en este caso usar la carpeta LED en donde quiera que se haya creado

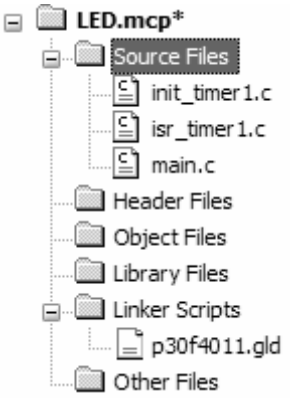


En las siguientes ventanas que salgan dar un clic en next, lo siguiente es crear un archivo con extensión ".c" y añadirlo al proyecto, así como añadir el archivo linker para el dsPIC especificado el cual se encuentra en el directorio donde se instalo la herramienta MPLAB C30.





Una vez realizado estos pasos intuitivos y fáciles se puede empezar a desarrollar una aplicación. A continuación se hará una aplicación que nos encienda y apague un led cada cierto tiempo usando el TIMER1 como delay.
Ya tenemos el archivo *main*.c en el cual estará el programa principal, pero además para facilitarnos la vida, para poder rehusar una configuración de un periférico en otro proyecto solo copiando el archivo crearemos otros archivos con los siguientes nombres *init_timer1*.c que inicializara al TIMER1 según la configuración que se encuentre en el, y otro archivo con el nombre *isr_timer1.c* en el cual se halla el vector de la interrupción del TIMER1 aquí es donde hacemos la rutina de mantenimiento para el TIMER1 en este caso se usara para hacer encender o apagar un led.



El código que debe contener cada archivo se muestra abajo, todas las configuraciones deben ser siempre realizadas usando el manual perteneciente al periférico en este caso usando el manual del dsPIC 30F4011.

```
******
Main.c
******

#include <p30f4011.h>
```

```
//---------------------------------------------------------------------
// main()
//---------------------------------------------------------------------
int main(void)
{
        // PORTE configuration for leds
        ADPCFG = 0xffff;
        TRISE = 0x0000;
        LATE = 0x0000;
        Init_Timer1();         // Initialized Timer
        T1CONbits.TON = 1;     //start T1
        while(1)
        {

        }
        return 0;
}
```

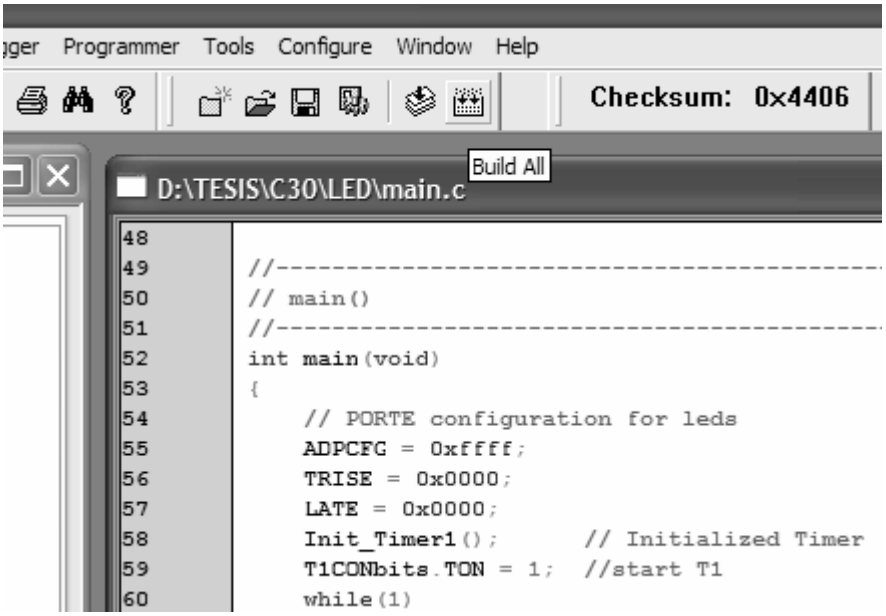
****
init_timer1.c
****

```
#include <p30f4011.h>

/*---------------------------------------------------------------------
  Function Name: Init_Timer1
  Description:   Initialize Timer1 for 0.5 second intervals
  Inputs:        None
  Returns:       None
---------------------------------------------------------------------*/
void Init_Timer1(void)
{
        //T1CON
        T1CON = 0b0000000000110000;
        //INTCON2
        INTCON2 = 0x0000;
        //PR1
        PR1 = 0xFFFF;
        //IEC0
        IEC0bits.T1IE = 1;
        //IPC0
        IPC0bits.T1IP = 7;
        //TMR1
        TMR1 = 0;
}
```

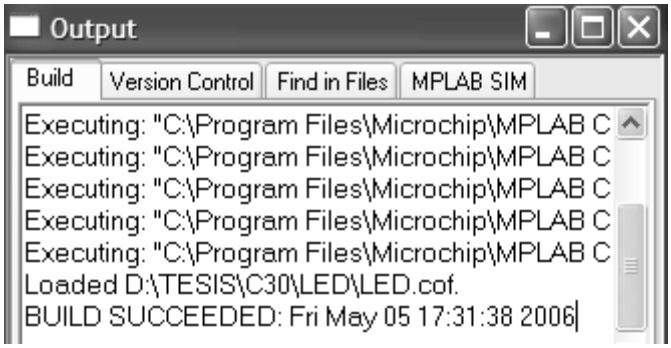
****
isr_timer1.c
****

```
#include <p30f4011.h>

/*---------------------------------------------------------------------
  Function Name: _T1Interrupt
  Description:   Timer1 Interrupt Handler
  Inputs:        None
  Returns:       None
---------------------------------------------------------------------*/
void __attribute__((__interrupt__)) _T1Interrupt( void )
{
        //led ON/OFF
        if (LATEbits.LATE1 == 0)
                LATEbits.LATE1 = 1;
        else
                LATEbits.LATE1 = 0;
        /* clear interrupt flag */
        IFS0bits.T1IF = 0;
}
```

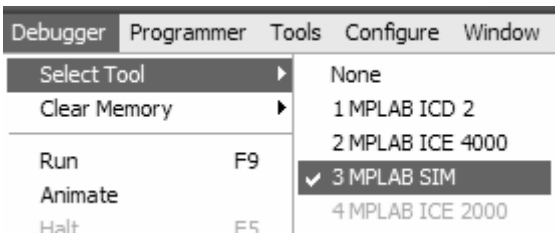
Una vez escrito el código de la aplicación es necesario compilarla haciendo un clic en *build all*



Si todo esta bien al compilar debe salir una ventana como la que se muestra abajo

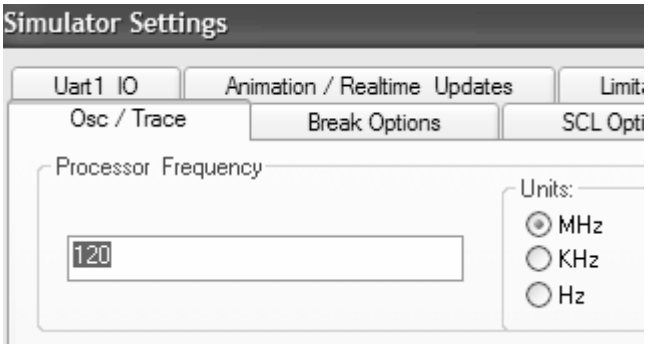


El siguiente paso es en la cadena de diseño de aplicaciones para dsPIC es el debugging del código para el cual tenemos varias herramientas de las cuales solo utilizaremos *MPLAB SIM*

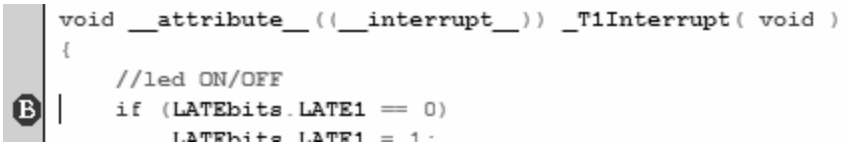


Una vez seleccionada la herramienta de debugging MPLAB SIM sale una nueva barra de herramientas que nos ayudara a correr el código

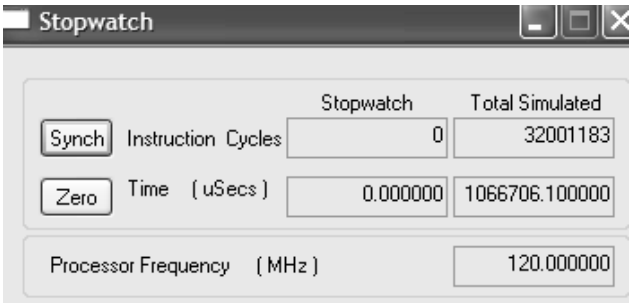
A continuación configuraremos el debugger con una frecuencia de 120 MHz para el procesador



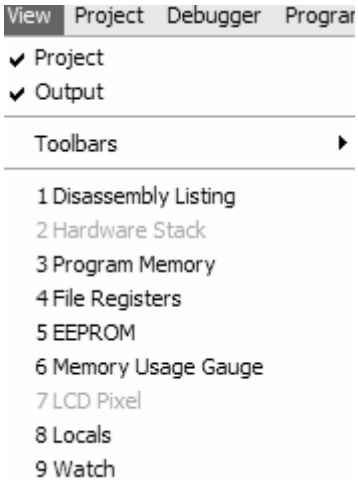
Los break points se ponen haciendo doble clic y sirven para cuando se esta simulando se detenga en ese punto la simulación, pondremos un break point en una línea del archivo *isr_timer1.c* para saber cada cuanto tiempo hay un desbordamiento del TIMER1

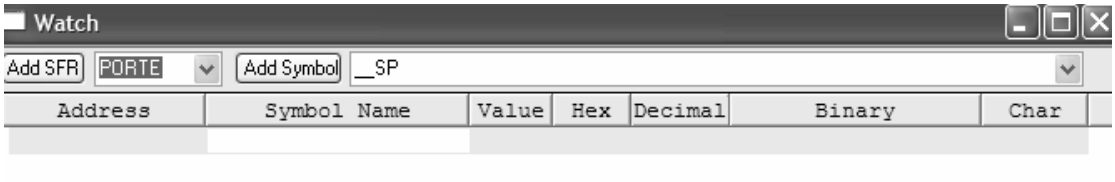


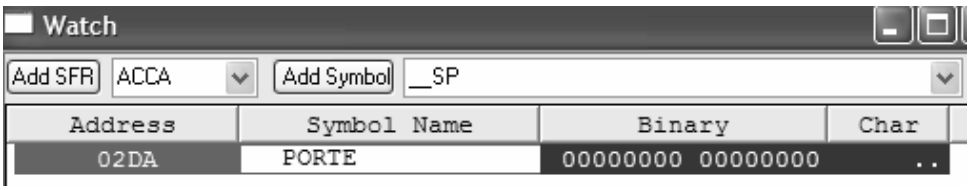
Para poder medir el tiempo de la simulación hasta el break point haremos visible la ventana StopWatch



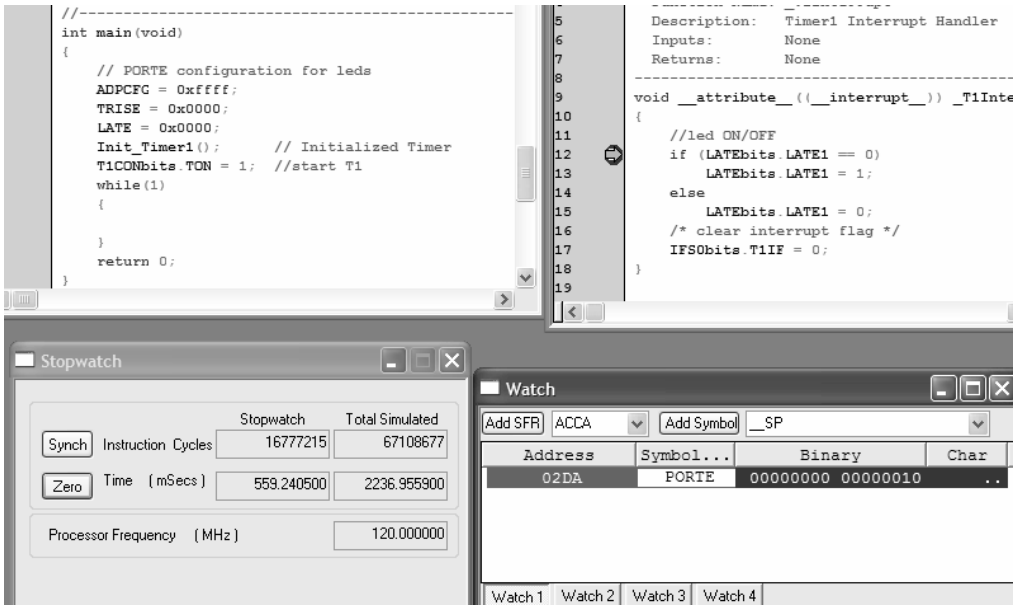
Para poder ver si se enciende el led o se apaga debemos ver el registro PORTE, esto se hace en la ventana Watch y añadiendo el registro

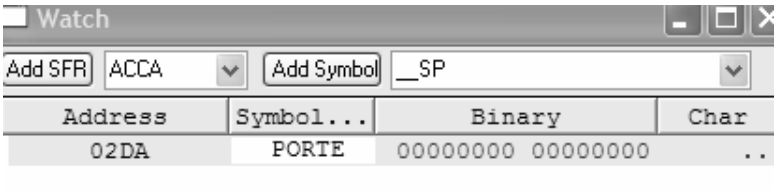






Para empezar la simulación pulsaremos F9 o , el programa iniciará corriendo la aplicación y se parara en el breakpoint cada 559 milisegundos se puede observar que cada vez que entra en la rutina de servicio de la interrupción se pregunta si el led esta encendido o apagado, si esta encendido lo apaga o viceversa, el cambio del led se puede ver en el registro PORTE





4.4 Generación de señales y adquisición con MATLAB

Con un amplio rango de herramientas para modelar sistemas, análisis, simulación y procesamiento de prototipos, Matlab es ideal para desarrollar proyectos avanzados, una razón mas para utilizar Matlab, se usara Matlab para generar señales a través de la línea de salida de la tarjeta de sonido del PC que nos ayudará a investigar las características de los filtros que diseñemos, también se usara para adquisición de datos a través de el puerto serial RS232, la tarjeta ***dsPIC-4ks*** tiene un conector para el puerto serial RS232 y otro para USB, cuando se utiliza el conector USB la comunicación es la misma ya que se emula un puerto serial.

Generación de señales de sonido

La generación de señales es muy sencilla, se realiza con una función llamada *sound()*, esta función convierte a sonido el contenido de un vector con una frecuencia de muestreo establecida al speaker de la PC (*line-out*), no se recomienda generar señales inferiores a los *100 Hz* en los parlantes del PC ya que pueden dañarse, abajo se muestra un script para generar una onda seno de 500 *Hz* durante *5 segundos*.

Script Matlab:

```
f = 500; %frecuencia de la señal
t = 5; % segundos
fs = 8000;frecuencia de muestreo de la señal de salida
Ts = 1 / fs;
nTs=0:Ts:t;
%señal x(t)
x=sin(2*pi*f*nTs);
%wavwrite(x,'sin20hz.wav');% esta línea genera un archivo wav de la señal
sound(x)
```

El script de arriba puede ser usado como plantilla para generar diferentes tipos de ondas.

Adquisición de datos usando puerto RS232

Matlab posee también una función llamada *serial()* para manejar el puerto RS232 del PC, cuando se use la comunicación USB de la tarjeta *dsPIC-4ks* es necesario ver que número de puerto es asignado y cambiar el código (*por ejemplo COM2*), ya que el firmware emula como si fuese comunicación RS232.

El dsPIC de la tarjeta *dsPIC-4ks* debe estar grabado con el firmware *adq_rs232.hex* para poder probar la comunicación entre el PC y la tarjeta, si esta todo bien después de ejecutar el script de abajo debe encenderse un led de la tarjeta *dsPIC-4ks* y en el command window de Matlab deberá salir “Test OK”.

Script Matlab:

```
%Prueba comunicación entre PC y Tarjeta dsPIC-4ks
clear all;
clc;
disp 'Test de comunicacion Tarjeta dsPIC-4ks'
%USB CDC emulación interfase
s = serial('COM4','BaudRate',230400); %crea serial port
s.InputBufferSize = 32768; %configura buffer entrada
s.ReadAsyncMode = 'continuous';
if (s.Status == 'closed')   %verifica si esta abierto serial port
    fopen(s);   %abre serial port
end
fprintf(s,'a'); %manda dato a Tarjeta
  while ( s.BytesAvailable < 0)
  end
  %lee dato de la Tarjeta
  input = fscanf(s,'%c',1);
  if (input == 'a')
     disp 'test OK'
  end

if (s.Status == 'open')%verifica si esta abierto serial port
    fclose(s);   %sierra serial port
end
delete (s);
```

5.1 Implementación de los filtros digitales

Al momento de implementar los filtros digitales surgen problemas como:

Qué tipo de filtro digital utilizar *FIR* o *IIR* ?

El sistema operará en tiempo real (*Real-Time*) o no (*Non real-Time*) ?

Que precisión usar *Floating-point* o *Fixed-point* ?

Filtros Digitales IIR

Ventajas	*Desventajas*
Alta selectividad para un orden particular	Fase no lineal
Respuesta a la magnitud precisa	Posible Inestabilidad debida a feedback
Corto procesamiento	Coeficientes difíciles de calcular

Filtros Digitales FIR

Ventajas	*Desventajas*
Fase lineal, constante delay	Orden filtro mucho mayor que en un IIR
Siempre estables	Largo procesamiento
Coeficientes fáciles de calcular	Requerimiento memoria grande

Usos
Transmisión datos
Sistemas de audio alta calidad

Sistema real-time (*RT*)

En un sistema RT las muestras de entrada deben ser procesadas para proveer una señal de salida antes de las próximas muestras de entrada, el tiempo de procesamiento

debe ser preciso y constante, una alta frecuencia de muestreo resta el tiempo disponible para poder procesar.

Sistema non real-time (*nRT*)
En este tipo de sistemas las muestras pueden ser guardadas y ser procesadas después, el procesamiento puede ser arduo, se usa representación floating-point para los coeficientes y las señales debido a que la velocidad no es un factor crítico.

Si se decide implementar en tiempo real el sistema de filtrado se deberá tomar en cuenta el periodo de muestreo de la señal y el tiempo que tarda en procesar el algoritmo el procesador.

Una de las principales desiciones es en que parte del sistema hacer fixed-point o floating-point ? En señales, coeficientes o resultados intermedios del procesamiento.

Truncación
Los efectos de presición se producen por la truncación obligatoria de los coeficientes del filtro y las señales de entrada y salida, esto puede hacer que difieran las especificaciones del filtro diseñado, debido a que la truncación mueve de su posición original a los polos y ceros.

$$h_{trunc}(n) = Round\left(\frac{h(n)\cdot(2^{B-1}-1)}{h(10)}\right)\cdot\frac{h(10)}{(2^{B-1}-1)}$$

La fórmula de arriba nos muestra el procedimiento para determinar los valores truncados, si se reduce el numero de bits B el error en los coeficientes aumenta.
En la tabla de abajo se comparan unos coeficientes truncados

Coefs	*Original*	*16-bit*	*12-bit*	*8-bit*
h(10)	0.348822	0.348822	0.348822	0.348822
h(9), h(11)	0.004010	0.004013	0.004090	0.002747
h(8), h(12)	-0.268081	-0.268076	-0.268049	-0.269170
h(7), h(13)	-0.008972	-0.008974	-0.009032	-0.008240
h(6), h(14)	0.102739	0.102740	0.102755	0.101625
h(5), h(15)	0.008322	0.008325	0.008350	0.008240
h(4), h(16)	0.012282	0.012285	0.012269	0.010987
h(3), h(17)	-0.004333	-0.004333	-0.004260	-0.005493
h(2), h(18)	-0.033368	-0.033363	-0.033400	-0.032960
h(1), h(19)	0.001195	0.001192	0.001193	0.000000
h(0), h(20)	0.012527	0.012530	0.012610	0.013733

La relación señal ruido *SNR* para un sistema digital es directamente proporcional al numero de bits usados en la cuantización, la relación es

$$SNR_{dB} = 6{,}02\cdot B - 1{,}2$$

Por ejemplo:
Las variables del filtro (entrada, salida y coeficientes) pueden estar cuantizadas en 16 bits, al hacer una multiplicación se necesitarán 32 bits y luego el resultado debe ser truncado de nuevo a 16 bits.

5.2 Implementación de un filtro digital IIR

Para implementar los filtros digitales diseñaremos utilizando el material presentado en el *Capítulo 2* y veremos un software alternativo de diseño que nos ahorrará bastante tiempo, la desventaja del software es que si se necesita un orden grande de el filtro digital no lo podremos hacer con este software, razón por la que se ha utilizado también Matlab para ahorrarnos tiempo en la matemática para diseñar los filtros.

5.2.1 Diseño de un filtro digital *IIR*

Se diseñara un filtro digital IIR pasa bajas utilizando el aproximante Butterworth con el método de la Transformación Bilineal, para las siguientes características:

$$a_{pass} = -1dB\,,a_{stop} = -21dB\,,\ f_{pass} = 300 - Hz\,,\ f_{stop} = 600 - Hz\,,\ f_s = 3 - khz$$

Recordar las especificaciones de la tarjeta dsPIC-4Ks para poder elegir las características del filtro, la frecuencia de corte de la tarjeta es 4Khz, la frecuencia máxima de muestreo de la tarjeta es 500Ksps, la cantidad máxima de caracteres por segundo que la tarjeta puede enviar al PC son 12000 caracteres por segundo, el ADC es de 10 bits.

El Script de abajo nos ayudará a calcular, los coeficientes para la función de transferencia y las respuestas del filtro.

```
clc;
clear all;
digits(16)%variable presicion
disp('*******************************');
disp('Calculos para filtro Butterworth');
disp('  Usando Tansformacion Bilineal ');
disp('*******************************');
%ingresa especificaciones
a_pass = -1;%dB
a_stop = -21;
f_pass = 300;%Hz
f_stop = 600;
f_sampler = 3000;
%Calcula frecuencias digitales
omega_pass = 2*sym('pi')*f_pass/f_sampler
omega_stop = 2*sym('pi')*f_stop/f_sampler
%Distorsiona para hallar frecuencias analógicas equivalentes
w_pass = 2*f_sampler*tan((2*pi*f_pass/f_sampler)/2)
w_stop = 2*f_sampler*tan((2*pi*f_stop/f_sampler)/2)
%calcula Omega
Omega=w_stop/w_pass
%Calcula orden
nb = (log((10^(-0.1*a_stop)-1)/(10^(-0.1*a_pass)-1)))/...
(2*log(Omega))
%redondea al entero mas cercano
n = round(nb)
%calcula ganancia
```

```
E = sqrt(10^(-0.1*a_pass)-1)
%revisa si n es par o impar
if rem(n,2) == 0
    disp('n par');
    m = 0;
else
    disp('n impar');
    m = 1;
end
%calcula radio
R = E^(-1/n)
%calcula ángulos
for i=1:(n-m)/2
theta(i)=(sym('pi') *(2*(i-1) + n + 1))/(2*n);
tt(i)=(pi*(2*(i-1) + n + 1))/(2*n);
end
theta
%calcula los polos
for i=1:(n-m)/2
polos(i)=R*cos(tt(i))+j*R*sin(tt(i));
end
polos
%calcula coeficientes
for i=1:(n-m)/2
B1m(i)=-2*R*cos(tt(i));
B2m(i)=(R*cos(tt(i))).^2+(R*sin(tt(i))).^2;
end
B1m
B2m
```

(n par) $$H_{B,n}(S) = \frac{\prod_{m}(B_{2m})}{\prod_{m}(S^2 + B_{1m}\cdot S + B_{2m})}$$

$$m=0,1,\ldots,(n/2)-1$$

$$H_{B,4}(S) = \frac{1{,}4019}{\left(S^2 + 0{,}9062\cdot S + 1{,}4019\right)}\cdot\frac{1{,}4019}{\left(S^2 + 2{,}1878\cdot S + 1{,}4019\right)}$$

Desnormalizamos la aproximación a pasa bajas LP, se puede expandir o se puede trabajar independientemente los términos de la función de transferencia, lo recomendable es trabajar independientemente, en este ejemplo expandiremos para mostrar como se hace en Matlab.

$$\Omega_{rL} = \frac{w_{stop}}{w_{pass}} = \frac{f_{stop}}{f_{pass}}$$

$$n_B = \frac{\log\left[(10^{-0.1\cdot a_{stop}} - 1)/(10^{-0.1\cdot a_{pass}} - 1)\right]}{2\cdot\log(\Omega_{rL})}$$

$$S_L = \frac{s}{w_0}$$

$$w_0 = w_{pass}$$

```
Wo = w_pass
%Varia según el orden
num_1 = [0 0 B2m(1)*Wo^2];
den_1 = [1 B1m(1)*Wo B2m(1)*Wo^2];
a = tf(num_1,den_1);
num_2 = [0 0 B2m(2)*Wo^2];
den_2 = [1 B1m(2)*Wo B2m(2)*Wo^2];
b = tf(num_2,den_2);
Hs = a*b
```

$$H_{B,4}(s) = \frac{2{,}830 \cdot 10^{13}}{\left(s^4 + 6032 \cdot s^3 + 1{,}819 \cdot 10^7 \cdot s^2 + 3{,}214 \cdot 10^{10} \cdot s + 2{,}830 \cdot 10^{13}\right)}$$

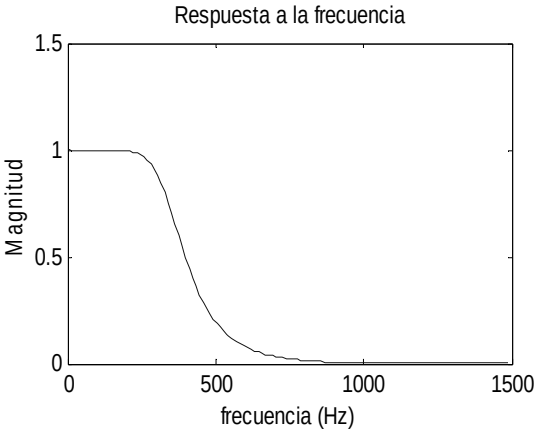
Hallamos la transformada bilineal ya expandida

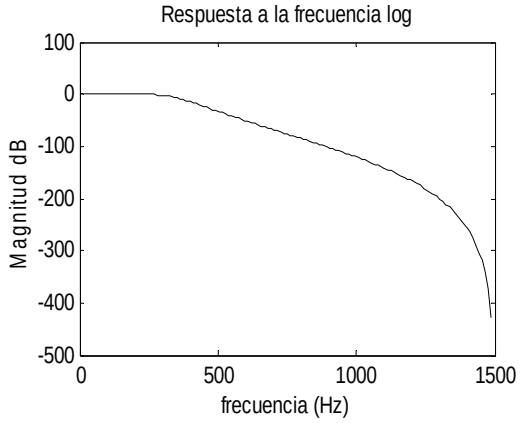
```
%Calculo directamente transformada bilineal
[num,den] = tfdata(Hs,'v');
[numd,dend] = bilinear(num,den,f_sampler);
Hz = tf(numd,dend,'variable','z^-1')
```

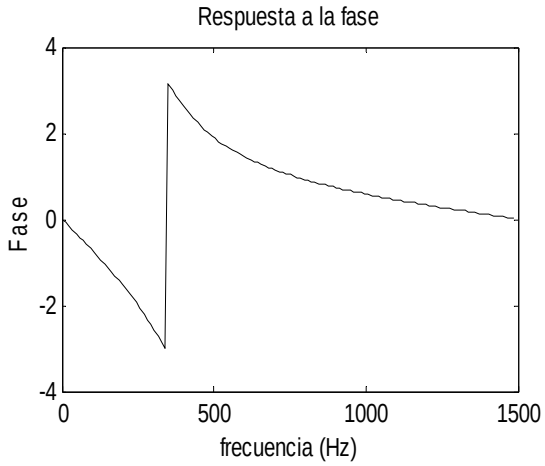
$$H_{B,4}(z) = \frac{0{,}008169 + 0{,}03268 \cdot z^{-1} + 0{,}04901 \cdot z^{-2} + 0{,}03268 \cdot z^{-3} + 0{,}008169 \cdot z^{-4}}{1 - 2{,}098 \cdot z^{-1} + 1{,}91 \cdot z^{-2} - 0{,}8203 \cdot z^{-3} + 0{,}1392 \cdot z^{-4}}$$

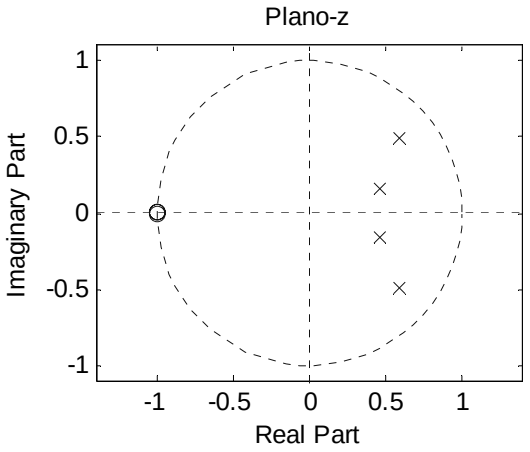
Graficamos las respuestas de la función de transferencia digital

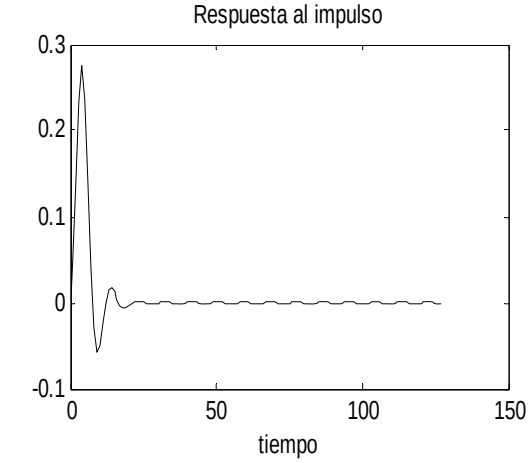
```
%gráfica respuesta a la frecuencia
[num,den] = tfdata(Hz,'v');
[H,w]=freqz(num,den,128);
plot(w*f_sampler/(2*pi),abs(H))
ylabel('Magnitud');
xlabel('frecuencia (Hz)')
title('Respuesta a la frecuencia');
%gráfica respuesta frecuencia log
figure(2);
plot(w*f_sampler/(2*pi),20*log(abs(H)),'-k')
ylabel('Magnitud dB');
xlabel('frecuencia (Hz)')
title('Respuesta a la frecuencia log');
%gráfica fase
figure(3);
plot(w*f_sampler/(2*pi),angle(H),'-k')
title('Respuesta a la fase');
ylabel('Fase');
xlabel('frecuencia (Hz)')
%plano z
figure(4);
zplane(num,den)
title('Plano-z');
%Respuesta al impulso
[h,t]=impz(num,den,128);
figure(5);
plot(t,h,'-k');
title('Respuesta al impulso');
xlabel('tiempo')
%gráfica respuesta al escalón
[h,t]=stepz(num,den,128);
figure(6);
plot(t,h,'-k');
title('Respuesta al escalón ');
xlabel('tiempo')
```

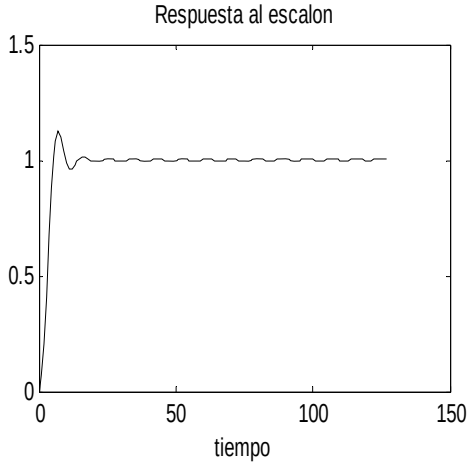






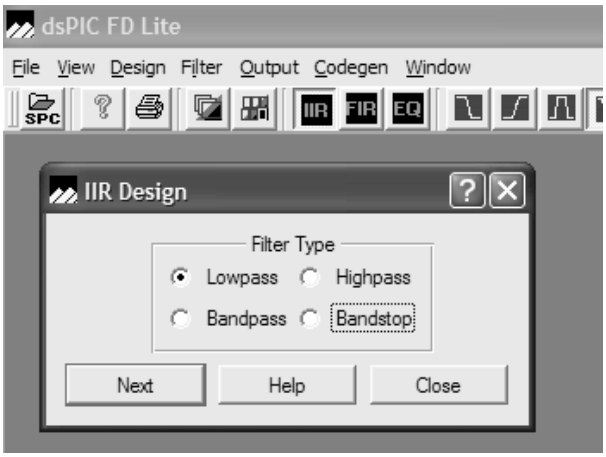


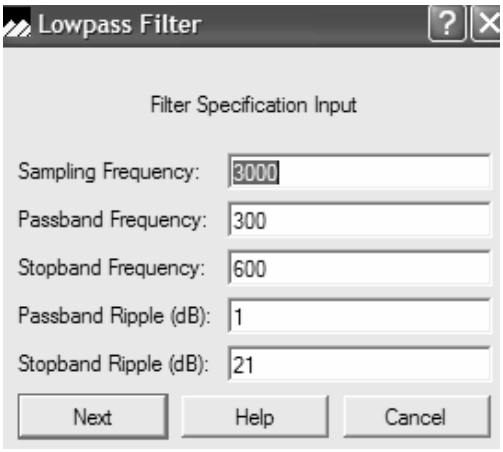


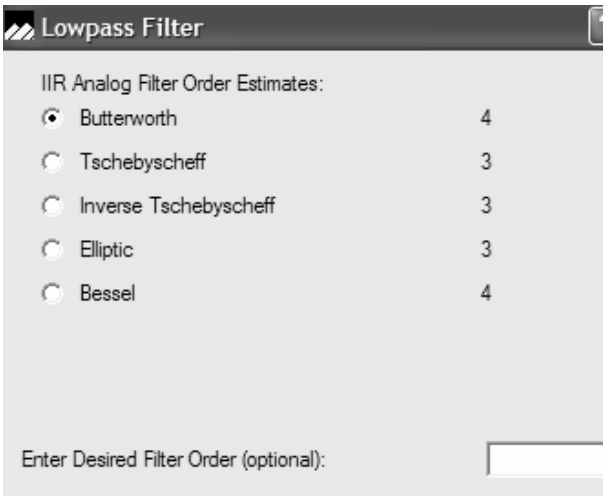


5.2.2 Diseño con dsPIC fd Lite

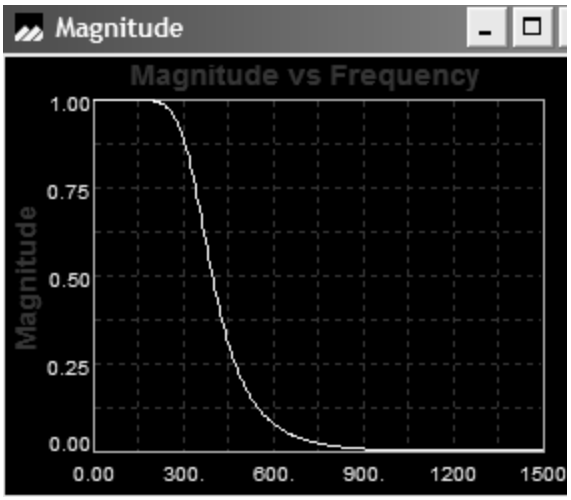
Ahora presentaremos un método alternativo, sencillo e intuitivo de diseño usando el software dsPIC fd Lite, seleccionaremos diseñar un filtro digital IIR pasa bajas y pondremos las especificaciones del filtro diseñado anteriormente.

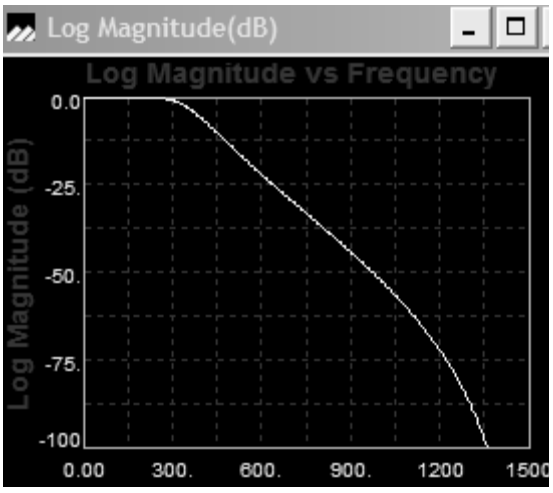


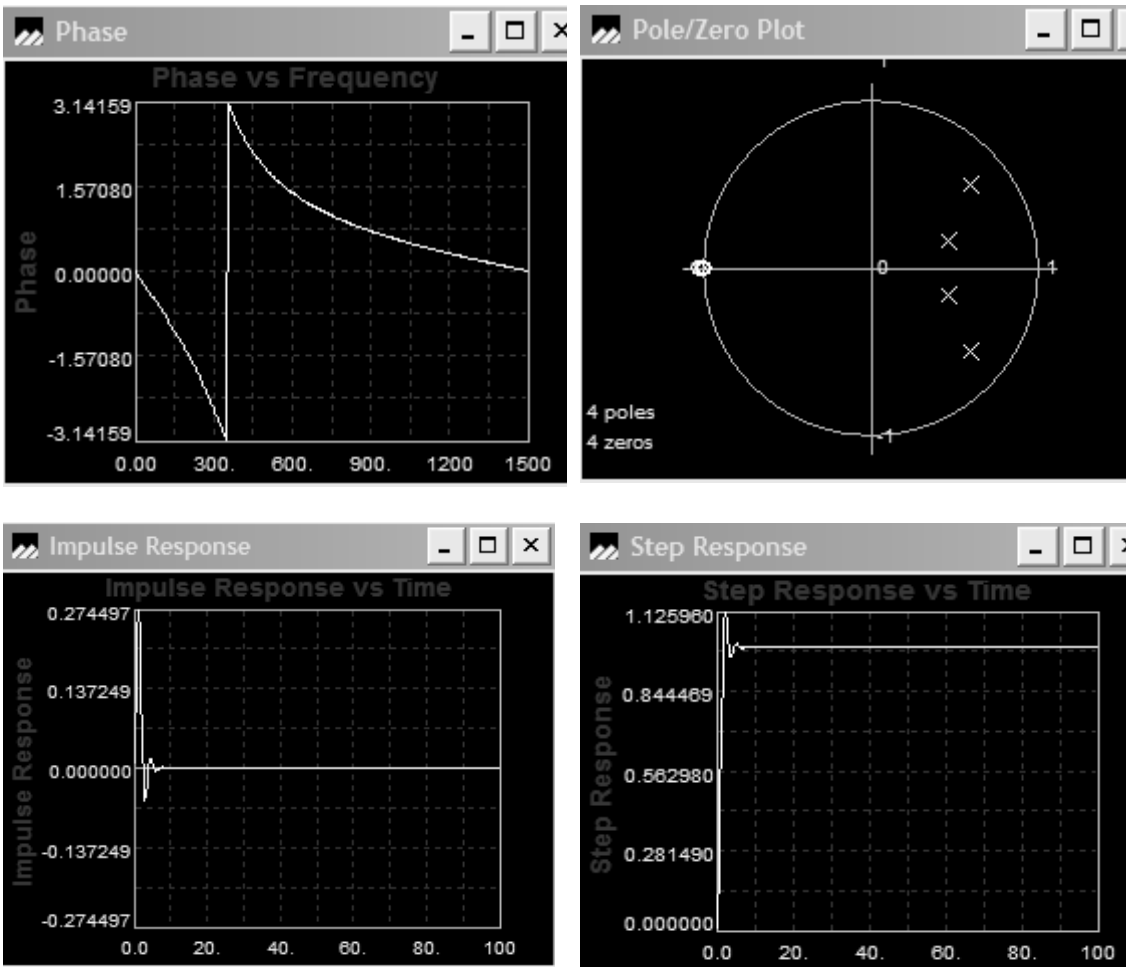




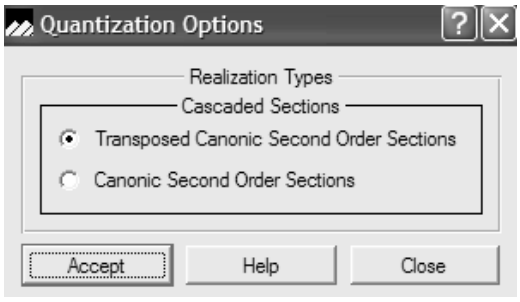
Una vez cargadas las especificaciones nos saldrán las respuestas del filtro digital diseñado, como se observa abajo son igual a las diseñadas con Matlab

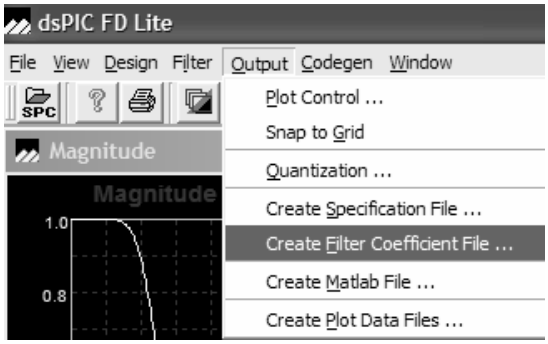






Para obtener los coeficientes del filtro con este software





El archivo creado contendrá algunas líneas de las cuales solo sirven las ultimas líneas, en la gráfica de abajo se indica cuales son los coeficientes

```
FILTER COEFFICIENT FILE
IIR DESIGN
FILTER TYPE                 LOW PASS
ANALOG FILTER TYPE          BUTTERWORTH
PASSBAND RIPPLE IN -dB        -.1000E+01
STOPBAND RIPPLE IN -dB        -.2100E+02
PASSBAND CUTOFF FREQUENCY   0.300000E+03 HERTZ
STOPBAND CUTOFF FREQUENCY   0.600000E+03 HERTZ
SAMPLING FREQUENCY          0.300000E+04 HERTZ
FILTER DESIGN METHOD: BILINEAR TRANSFORMATION
FILTER ORDER            4    4h
NUMBER OF SECTIONS      2    2h
NO. OF QUANTIZED BITS  16   10h
QUANTIZATION TYPE - FRACTIONAL FIXED POINT
COEFFICIENTS SCALED FOR CASCADE FORM I
         1        1     /* shift count for overall gain */
     16384     4000     /* overall gain                 */
         0        0     /* shift count for section  1 values */
      2608      A30     /* section  1 coefficient B0 */
      5217     1461     /* section  1 coefficient B1 */
      2608      A30     /* section  1 coefficient B2 */
     30038     7556     /* section  1 coefficient -A1*/
     -7706 FFFFE1E6     /* section  1 coefficient -A2*/
         1        1     /* shift count for section  2 values */
      1681      691     /* section  2 coefficient B0 */
      3362      D22     /* section  2 coefficient B1 */
      1681      691     /* section  2 coefficient B2 */
     19354     4B9A     /* section  2 coefficient -A1*/
     -9695 FFFFDA21     /* section  2 coefficient -A2*/
 0.7958984375000000E-01 3FB4600000000000 0.79589844E-01 /* section  1 B0 */
 0.1592102050781250E+00 3FC4610000000000 0.15921021E+00 /* section  1 B1 */
 0.7958984375000000E-01 3FB4600000000000 0.79589844E-01 /* section  1 B2 */
 0.9166870117187500E+00 BFED558000000000 0.91668701E+00 /* section  1 -A1*/
 -.2351684570312500E+00 3FCE1A0000000000 -.23516846E+00 /* section  1 -A2*/
 0.5130004882812500E-01 3FAA440000000000 0.10260010E+00 /* section  2 B0 */
 0.1026000976562500E+00 3FBA440000000000 0.20520020E+00 /* section  2 B1 */
 0.5130004882812500E-01 3FAA440000000000 0.10260010E+00 /* section  2 B2 */
 0.5906372070312500E+00 BFE2E68000000000 0.11812744E+01 /* section  2 -A1*/
 -.2958679199218750E+00 3FD2EF8000000000 -.59173584E+00 /* section  2 -A2*/
```

coeficientes

A0 siempre es 1

Si analizamos los coeficientes obtenidos con el dsPIC fd Lite podemos darnos cuenta que no concuerdan con los calculados, pero sin embargo las gráficas estaban iguales, la razón es que nosotros expandimos la función de transferencia, en cambio el software halla cada termino independiente, volviendo con Matlab se puede hallar la transformación bilineal de cada termino independientemente modificando un poco el script anterior:

```
%Calculo directamente transformada bilineal
[num,den] = tfdata(a,'v');
[numd,dend] = bilinear(num,den,f_sampler);
vpa(numd,6)
vpa(dend,6)
Hz = tf(numd,dend,'variable','z^-1')
```

```
Transfer function:
0.1026 + 0.2052 z^-1 + 0.1026 z^-2
-----------------------------------------
   1 - 1.181 z^-1 + 0.5917 z^-2
```

```
%Calculo directamente transformada
bilineal
[num,den] = tfdata(b,'v');
[numd,dend] =
bilinear(num,den,f_sampler);
vpa(numd,65)
vpa(dend,65)
```

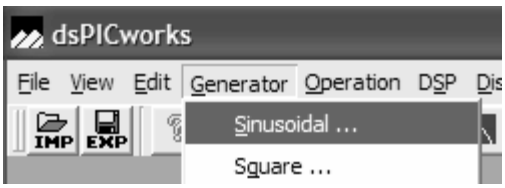
```
Hz = tf(numd,dend,'variable','z^-1')
```

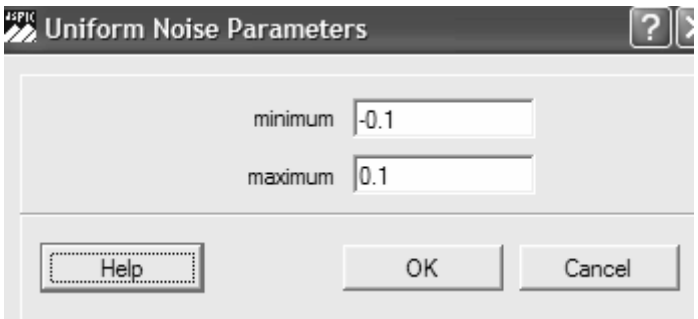
```
Transfer function:
0.07962 + 0.1592 z^-1 + 0.07962 z^-2
-------------------------------------------
   1 - 0.9167 z^-1 + 0.2352 z^-2
```

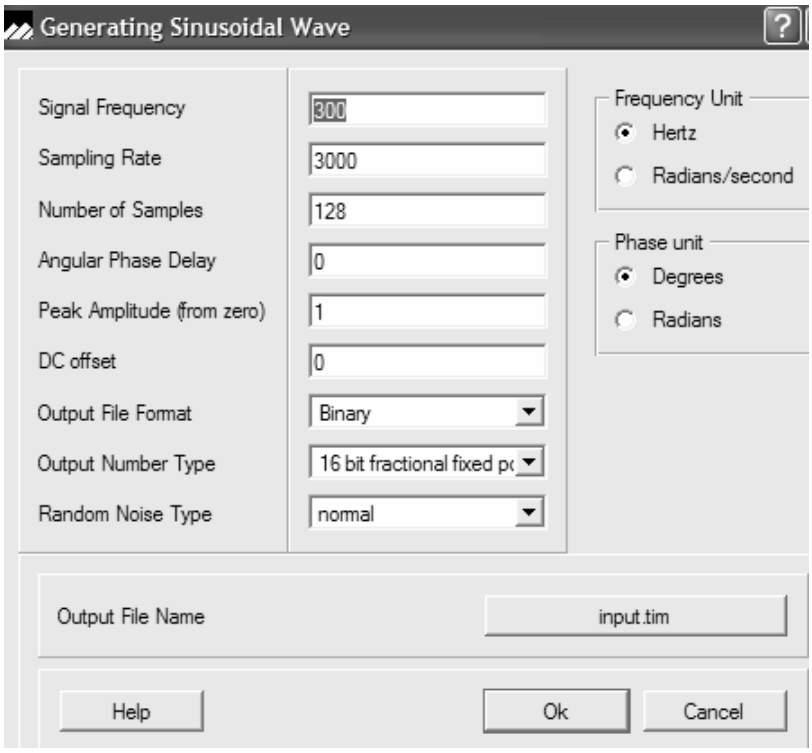
$$H_{B,4}(z) = \frac{0{,}07962 + 0{,}1592 \cdot z^{-1} + 0{,}079621 \cdot z^{-2}}{1 - 0{,}09167 \cdot z^{-1} + 0{,}2352 \cdot z^{-2}} \cdot \frac{0{,}1026 + 0{,}2052 \cdot z^{-1} + 0{,}1026 \cdot z^{-2}}{1 - 1{,}811 \cdot z^{-1} + 0{,}5917 \cdot z^{-2}}$$

5.2.3 Simulación del filtro con *dsPICworks*

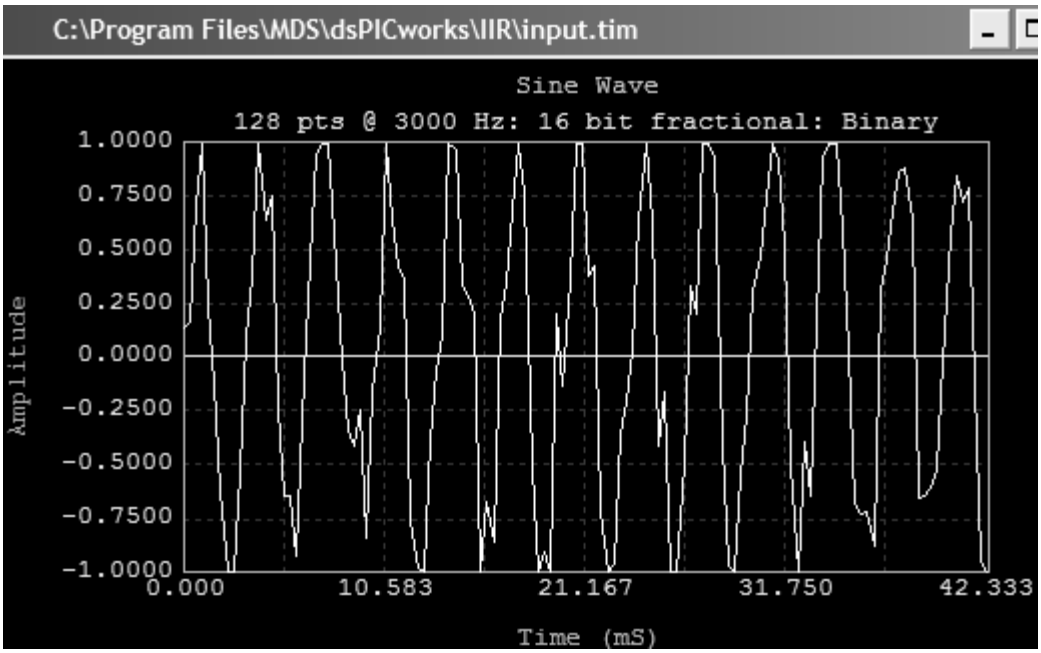
Una vez obtenidos los coeficientes del la función de transferencia del filtro, es el momento de correr datos a través del filtro y examinar si el filtro esta funcionando para esto se utilizara la herramienta *dsPICworks*, este software tiene la habilidad de generar formas de onda y la de importar formas de onda como archivos *wav*, también permite probar operaciones matemáticas y de DSP.
En el menú de dsPICworks seleccionaremos *Generador / Sinusoidal* para generar una onda seno con ruido



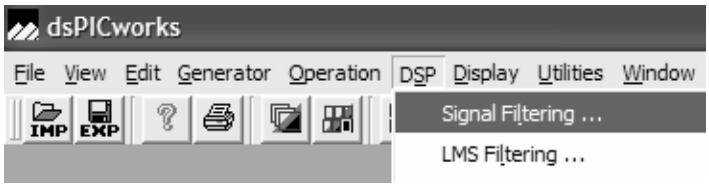


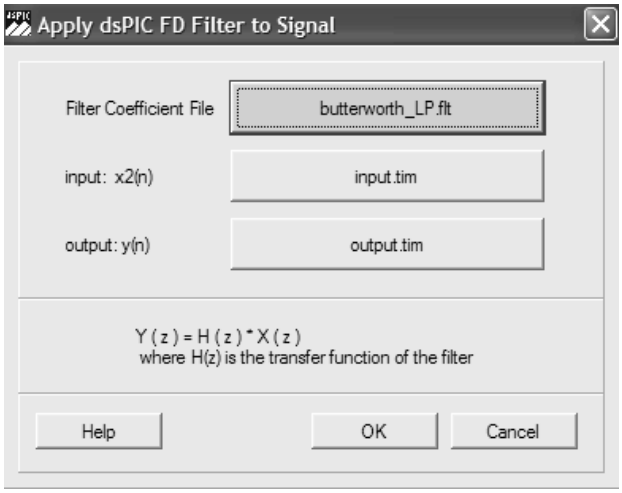


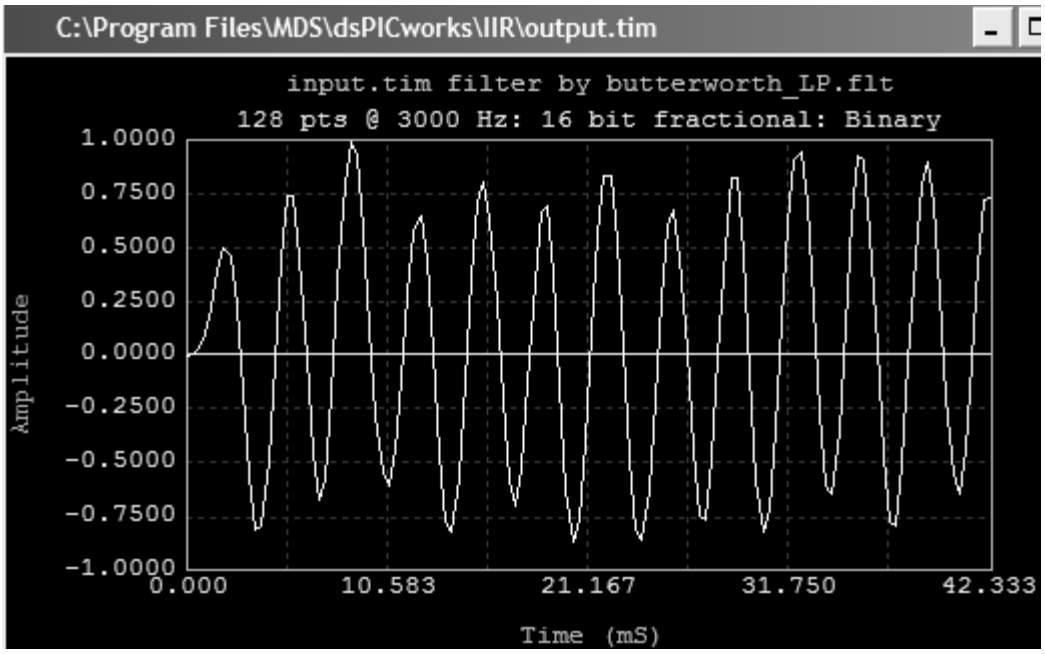
La onda seno debe lucir como la de abajo



En el menú seleccionaremos *DSP / Signal Filtering*, Signal Filtering nos permitirá filtrar una señal en el dominio del tiempo usando los coeficientes generados por ***dsPIC fd Lite***, dsPIC fd Lite genera un archivo *.FLT* el cual contiene los coeficientes del filtro, este método no sirve para probar los coeficientes calculados (*usuario),* aunque es posible implementar modificando el archivo *.FLT* lo cual es muy engorroso, mas adelante se vera un método para probar en Matlab los coeficientes calculados.







La gráfica de arriba muestra la señal filtrada, si se compara con la gráfica anterior se puede comprobar que el filtro esta funcionando perfectamente.

5.2.4 Simulación del filtro con *MATLAB*

En Matlab se puede analizar los coeficientes del filtro calculados utilizando la función ***filter()*** esta función sirve para probar filtros digitales FIR e IIR.

```
y = filter (B, A, x)
```

```
x es la señal de entrada(vector),
y es la señal de salida,
A coeficientes (denominador)
B  coeficientes (numerador)
```

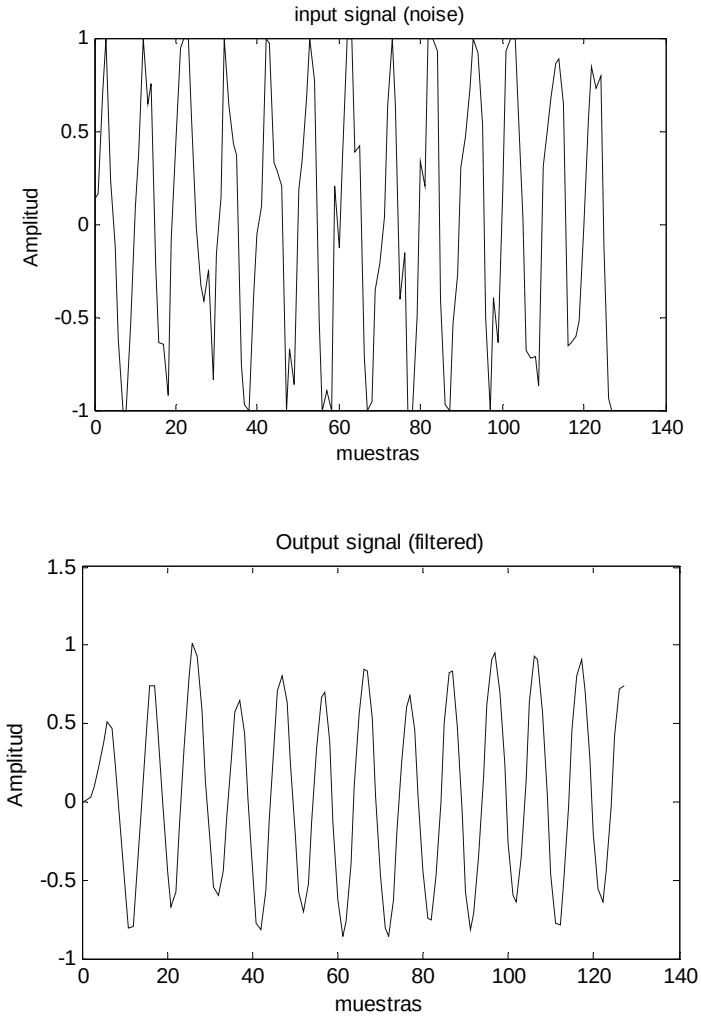
$$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \ldots + a(na+1)z^{-na}} X(z)$$

Utilizaremos la misma señal de entrada que se utilizo en dsPICworks, es necesario que la señal sea guardada con extensión *wav*.

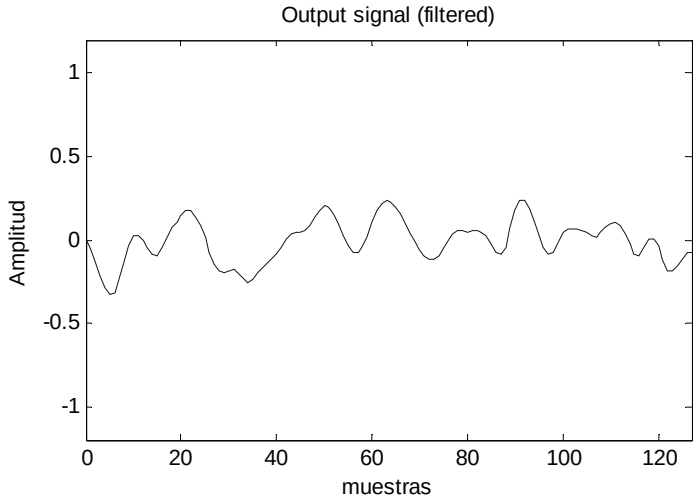
Script Matlab:

```
... código
```

```
% lee una onda desde un archivo wav
y = wavread('input.wav'); % input.wav onda generada en dsPICworks
%gráfica señal de entrada
figure(7);
plot([0:length(y)-1],y)
title('input signal (noise)');
xlabel('muestras');
ylabel('Amplitud');
% Analiza coeficientes del filtro
Y = filter(num,den,y);
% gráfica senal de salida
figure(8);
plot([0:length(Y)-1],Y)
title('Output signal (filtered)');
xlabel('muestras');
ylabel('Amplitud');
```



Si entra una señal superior a los *300 Hz* el filtro empieza a atenuar, en la gráfica de abajo se simulo la misma señal de entrada de pero con una frecuencia de 1Khz.

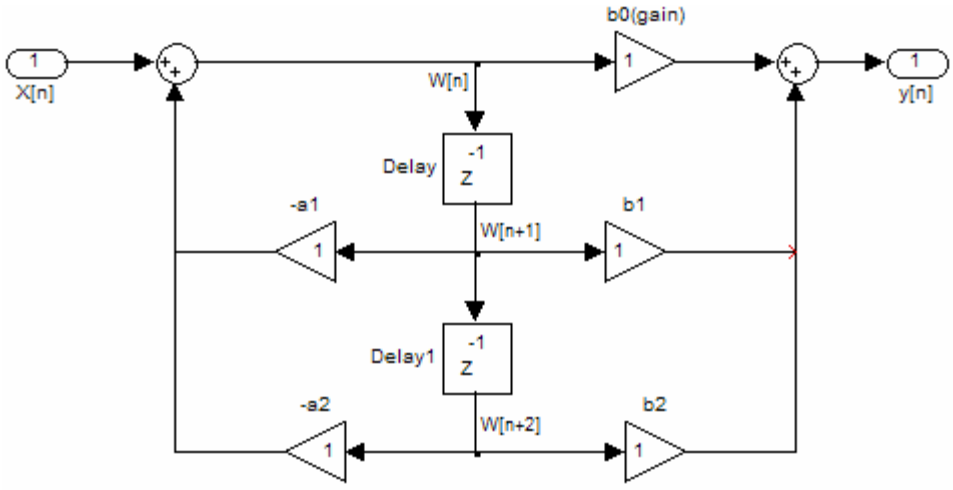


Las 3 gráficas anteriores nos dan una clara pauta del comportamiento del filtro cuando pasan los datos por este funcionando este como restaurador de la señal y como

rechazador de señales, si comparamos estas gráficas con las obtenidas con dsPICworks observaremos que el resultado es el mismo.

5.2.5 Código en "C" para implementar filtros digitales *IIR*

Para implementar los filtros digitales se recomienda trabajar con coeficientes cuadráticos, en la figura de abajo se muestra una estructura cuadrática (*forma directa II*)



Se puede generar la función de transferencia determinando las expresiones para las señales intermedias w[n], la señal de entrada x[n] y la señal de salida y[n]

$$w[n] = x[n] - a1 \cdot w[n-1] - a2 \cdot w[n-2]$$

$$y[n] = b0 \cdot w[n] + b1 \cdot w[n-1] + b2 \cdot w[n-2]$$

Las ecuaciones de arriba también pueden estar en términos del dominio – z

$$W[z] = X[z] - a1 \cdot z^{-1} \cdot W[z] - a2 \cdot z^{-2} \cdot W[z]$$

$$Y[z] = b0 \cdot W[z] + b1 \cdot z^{-1} \cdot W[z] + b2 \cdot z^{-2} \cdot W[z]$$

Combinando las 2 ecuaciones de arriba se obtiene la función de transferencia del filtro

$$H[z] = \frac{Y[z]}{X[z]} = \frac{b0 + b1 \cdot z^{-1} + b2 \cdot z^{-2}}{1 + a1 \cdot z^{-1} + a2 \cdot z^{-2}}$$

Para desarrollar los filtros digitales IIR, se usan combinaciones de términos cuadráticos como el término de arriba. Después de determinar el diagrama de bloques del sistema para un filtro, se usa este como ***guía*** para implementar el filtro, para cada término cuadrático se calcula una señal intermedia $w[n]$ y una señal de salida $y[n]$.

Reescribiendo…

$$m1 = w[n-1]$$
$$m2 = w[n-2]$$

$$w[n] = x[n] - a1 \cdot m1 - a2 \cdot m2$$
$$y[n] = b0 \cdot w[n] + b1 \cdot m1 + b2 \cdot m2$$

m1 y ***m2*** reflejan los estados de memoria del término cuadrático

```
o = x * gain;
for(j = 0; j < numb_quads ;j++)
{
      jj = j*3;
      w = o - m1[j] * a[jj+1] - m2[j] * a[jj+2];
      o = w + m1[j] * b[jj+1] + m2[j] * b[jj+2];
      m2[j] = m1[j];
      m1[j] = w;
}
```

El código de arriba puede ser lento por la calculación de bastante índice, la velocidad puede incrementarse utilizando punteros en los coeficientes y valores de memoria. Los coeficientes deben guardarse en un array "C" para ser utilizados secuencialmente

```
C[0] = gain             C[5] = a1 (quad 2)
C[1] = a1 (quad 1)      C[6] = a2 (quad 2)
C[2] = a2 (quad 1)      C[7] = b1 (quad 2)
C[3] = b1 (quad 1)      C[8] = b2 (quad 2)
C[4] = b2 (quad 1)      ...
```

Los estados de memoria se deben guardar en un array con un número igual al doble de términos cuadráticos "quad", usando la primera mitad ***m1*** y la segunda mitad ***m2***, el valor inicial es *0*.

```
M[0] = m1 (quad 1)      M[n] = m2 (quad 1)
M[1] = m1 (quad 2)      M[n+1] = m2 (quad 2)
M[2] = m1 (quad 3)      M[n+2] = m2 (quad 3)
```

n es el número de términos cuadráticos

La función en “C” para utilizar el filtro digital IIR quedaría así:

```
/*================================================
IIR Filter function Original by Les Thede
Arguments:  x - ptr señal entrada
            numb_quads – numero términos cuadráticos
            N – numero de valores en el array
================================================*/
void IIR_Filter(int *x, int numb_quads, int N)
{
      int i,j,k;
      int *input;             /* ptrs to in arrays */
      float w,temp;           // intermed & output values
      /* Make copies of input pointers */
      input = x;
      /* Start loop for number of data input values N*/
      for(k=0; k<N; k++)
      {//1
            //stage(1) input - output1 stage(2) ouput1 - output2
            temp = *input++;
            /* Start loop for number of quad factors */
            for(j = 0; j < numb_quads ;j++)
            {//2
                  i = j * 3;
                  w = temp - an[i+1]*m1[j] - an[i+2]*m2[j];
                  temp =  bn[i]*w + bn[i+1]*m1[j] + bn[i+2]*m2[j];
                  m2[j] = m1[j];
                  m1[j] = w;
            }//2
            /* Convert output to int and store */
            y[k] = (int)temp;
      }//1
}//end IIR_Filter
```

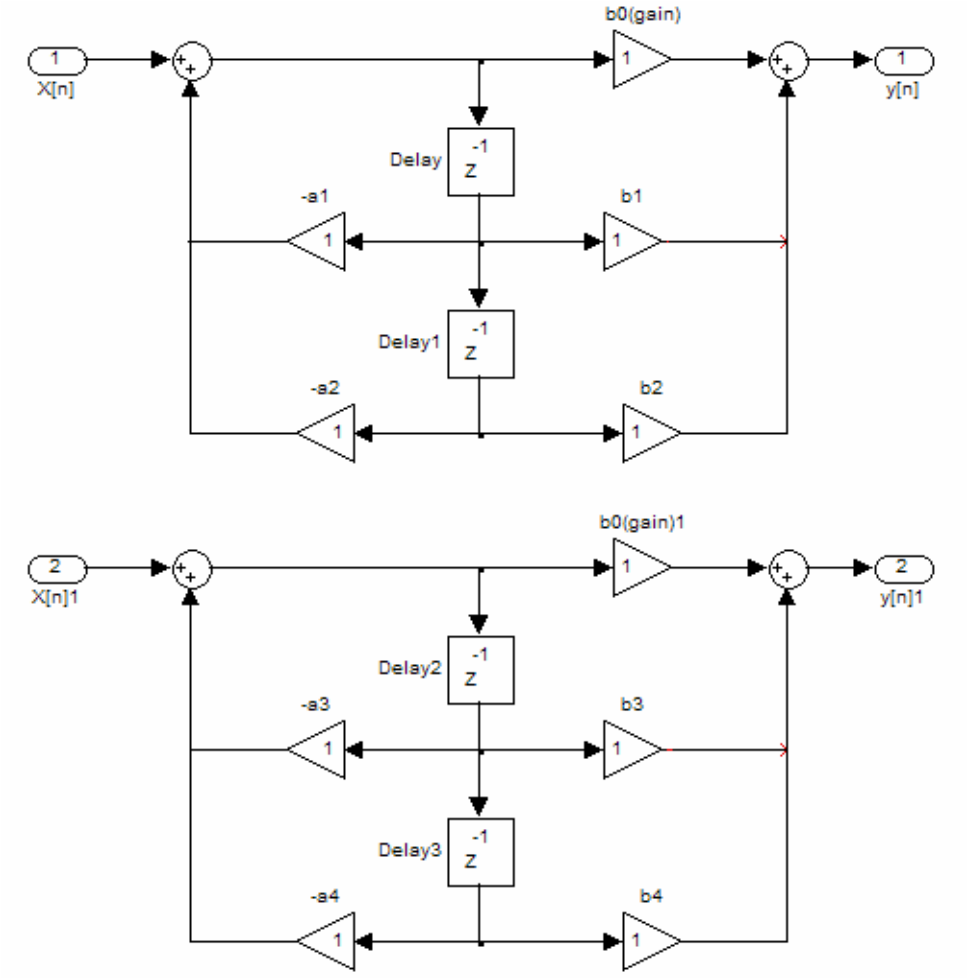
```
//==================================
// IIR_Filter global variables
// A coefficients
float an[6] = {

      };
// B coefficients
float bn[6] = {

      };
// memory
float m1[2];
float m2[2];
// output sigal - filtered
int y[128];
//==================================
```

El algoritmo de arriba no esta optimizado, como se explico anteriormente puede ser mejorado usando punteros, el algoritmo puede ser utilizado en aplicaciones *Non real-Time* o *Real-Time*.

5.2.6 Código en "C" para el filtro IIR diseñado

Usando la función de transferencia del filtro diseñado anteriormente obtendremos el diagrama de bloques del sistema y desarrollaremos el código.

$$H_{B,4}(z) = \frac{0{,}07962 + 0{,}1592 \cdot z^{-1} + 0{,}079621 \cdot z^{-2}}{1 - 0{,}09167 \cdot z^{-1} + 0{,}2352 \cdot z^{-2}} \cdot \frac{0{,}1026 + 0{,}2052 \cdot z^{-1} + 0{,}1026 \cdot z^{-2}}{1 - 1{,}811 \cdot z^{-1} + 0{,}5917 \cdot z^{-2}}$$



Obteniendo las señales intermedia y de salida tenemos:

$$w[n] = a0 \cdot x[n] + a1 \cdot w[n-1] + a2 \cdot w[n-2] \rightarrow a0 = siempre\_es\_1$$

$$y[n] = b0 \cdot w[n] + b1 \cdot w[n-1] + b2 \cdot w[n-2]$$

$$m1 = w[n-1]$$
$$m2 = w[n-2]$$

$$w[n] = a0 \cdot x[n] + a1 \cdot m1 + a2 \cdot m2$$

$$y[n] = b0 \cdot w[n] + b1 \cdot m1 + b2 \cdot m2$$

Código “C” filtro digital IIR

El filtro tiene 2 términos cuadráticos numb_quads = 2

```
//---------------------------------------------------------------------
--
// Variable declarations
//---------------------------------------------------------------------
--

//==================================
// IIR_Filter global variables
// A coefficients obtained with matlab script
float an[6] = {
      1.0000000, -1.1813329,  0.5917454,
      1.0000000, -0.9167003,  0.2351755
      };
// B coefficients
float bn[6] = {
      0.1026031, 0.20520624, 0.10260312,
      0.0796188, 0.15923760, 0.07961880
      };
// memory need clear
float m1[2];
float m2[2];
// output sigal - filtered
int y[128];
//==================================


//---------------------------------------------------------------------
--
// Function Prototypes
//---------------------------------------------------------------------
--
void IIR_Filter(int *x,int numb_quads, int N);


/*================================================
IIR Filter function
Arguments:  x - ptr senal entrada
                  numb_quads – numero términos cuadráticos
                  N – numero de valores en el array
================================================*/
void IIR_Filter(int *x, int numb_quads, int N)
{
      int i,j,k;
      int *input;             /* ptrs to in arrays */
      float w,temp;           // intermed & output values
      /* Make copies of input pointers */
      input = x;
      /* Start loop for number of data input values N*/
      for(k=0; k<N; k++)
      {//1
            //stage(1) input - output1 stage(2) ouput1 - output2
            temp = *input++;
            /* Start loop for number of quad factors */
            for(j = 0; j < numb_quads ;j++)
```

```
            {//2
                  i = j * 3;
                  w = temp - an[i+1]*m1[j] - an[i+2]*m2[j];
                  temp =  bn[i]*w + bn[i+1]*m1[j] + bn[i+2]*m2[j];
                  m2[j] = m1[j];
                  m1[j] = w;
            }//2
            /* Convert output to int and store */
            y[k] = (int)temp;
      }//1
}//end IIR_Filter
```

5.2.7 Comparación de desempeño entre un *PICmicro* y un *dsPIC*

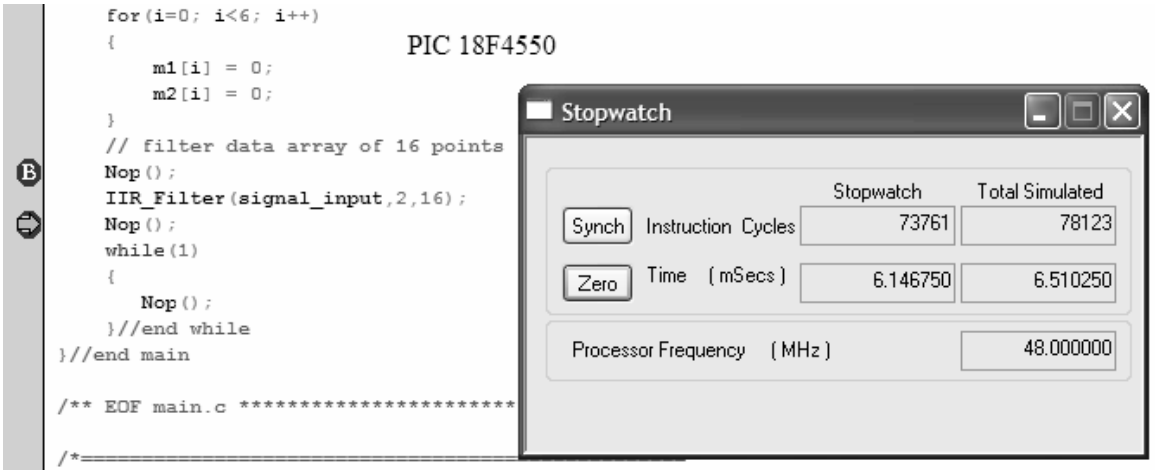
La comparación de desempeño se ha realizado simulando el algoritmo *IIR_Filter();* en un PIC18F4550 y en un dsPIC30F4011, para la señal de entrada se utilizo un array de tipo *int* con *16 muestras*.

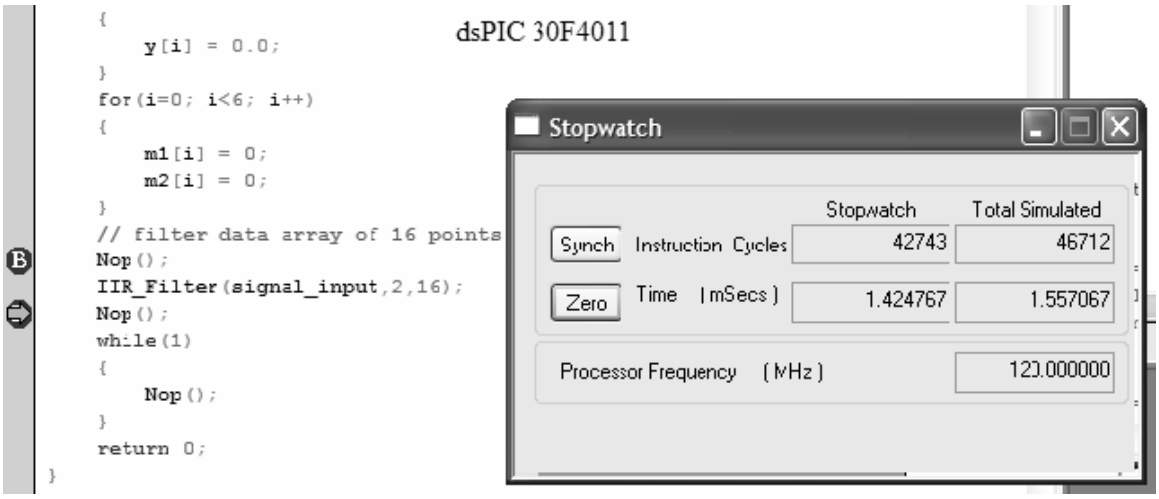
Se tomó el array de entrada que se utilizo en Matlab para probar el filtro, al array de entrada se le sumo 2.5 (*offset*), se le divido (*ADC 10bit*s) para /(5/1024) y se redondeo el resultado, todo esto para simular como llega el dato a la tarjeta *dsPIC-4ks*, esto sirve de gran ayuda para comprobar los resultados del algoritmo.

SIMULACION	Algoritmo *IIR_Filter()*	
	PICmicro PIC18F4550	*DSC dsPIC 30F4011*
Instrucciones	73761	42743
Frecuencia Procesador	48 Mhz	120 Mhz
Tiempo	6,14 milisegundos	1,42 milisegundos
# muestras	16	16
Términos cuadráticos	2	2

En el cuadro de arriba se puede observar la comparación de desempeño de ambos procesadores utilizando el algoritmo *IIR_Filter* , el número de instrucciones nos da la pauta de diferencia de arquitecturas.

En las siguientes gráficas se muestra *el STOPWATCH* de cada simulación





El algoritmo *IIR_Filter_opt()* es el algoritmo optimizado, utilizando punteros simula el circular buffer, en la tabla de abajo se muestran los resultados de simulación del algoritmo optimizado.

SIMULACION **dsPIC30F4011**	Algoritmo *IIR_Filter_opt()*	*IIR_Filter()*
Instrucciones	42415	42743
Frecuencia Procesador	120 Mhz	120 Mhz
Tiempo	1,413833 milisegundos	1,424767 milisegundos
# muestras	16	16
Términos cuadráticos	2	2

Código “C” filtro digital IIR optimizado

```
/*================================================
IIR Filter function using pointers
Arguments:  x - ptr señal entrada
            Y - ptr señal salida
            M - ptr to memory array
            C - ptr to coefs array
            numb_quads – numero términos cuadráticos
            N – numero de valores en el array
================================================*/
void IIR_Filter_opt(int *x,int *Y,float *M,float *C,int numb_quads,
int N)
{

      int *input,*y,                /* ptrs to in/out arrays */
      i,j;                          /* loop counters */
      float *c,*m1,*m2,             /* ptrs to coef/memory arrays*/
      w,temp;                       /* intermed & output values */
      /* Make copies of input and output pointers */
      input = x;
      y = Y;
      /* Start loop for number of data input values N*/
      for(i=0; i<N; i++)
      {//1
            m1 = M;
            m2 = M + numb_quads;
            c = C;
            //stage(1) input - output1 stage(2) ouput1 - output2
            temp = *input++;
            /* Start loop for number of quad factors */
            for(j = 0; j < numb_quads ;j++)
            {//2
                  w = temp - *m1 * *c++;
                  w -= *m2 * *c++;
                  temp = w * *c++;
                  temp += *m1 * *c++;
                  temp += *m2 * *c++;
                  *m2++ = *m1;
                  *m1++ = w;
            }//2
            /* Convert output to int and store */
            *y++ = (int)temp;
      }//1
}//end IIR_Filter



//==================================
// IIR_Filter global variables
// circular buffer Filter coefficients
float coef[]={
      //stage 1
      -1.1813329, //A1
      0.5917454,  //A2
      0.1026031,  //B0
      0.20520624, //B1
      0.10260312, //B2
      //stage 2
      -0.9167003, //A1
      0.2351755,  //A2
      0.0796188,  //B0
```

```
      0.15923760, //B1
      0.07961880  //B2
      };

// memory
float Mem[4];
// output sigal - filtered
int out[128];
//================================
```

5.3 Implementación de un filtro digital FIR

5.3.1 Diseño de un filtro digital *FIR*

Calcular un filtro digital FIR pasa bajas usando la ventana Kaiser de 7 puntos para las siguientes características:

$$a_{pass} = -1dB\,, a_{stop} = -21dB\,, f_{pass} = 300 - Hz\,, f_{stop} = 600 - Hz\,, f_s = 3 - khz$$

Script Matlab:

El siguiente script nos calculará los coeficientes del filtro

```
clc;
digits(1);
clear all;
disp('************************************');
disp(' Filtro FIR LP usando ventana kaiser');
disp('************************************');
%ingresa especificaciones
f_pass = 300;%Hz
f_stop = 600;
f_sampler = 3000;
w_pass = 2*pi*f_pass;%rad/seg
w_stop = 2*pi*f_stop;
a_pass1 = -1;
a_stop1 = -21;
%Calculo errores bandas paso y atenuación dp ds
dp = 1-10^(0.05*a_pass1)
ds = 10^(0.05*a_stop1)
%Calculo A
if dp < ds
    A = -20*log10(dp);
else
    A = -20*log10(ds);
end
%calcula Beta
if A > 50
    Beta = 0.1102*(A-8.7)
elseif A < 21
    Beta = 0.5 %default minimun
else
    Beta = 0.5842*((A-21)^(0.4)) + 0.07886*(A-21)
end
%calcula banda transición
DOmega = (w_stop - w_pass)/f_sampler;
%calcula longitud filtro estimada
%calcula N
if A > 21
    N = (A-7.95)/(2.285*DOmega);
else
    N = 5.794/DOmega;
end
N = round(N);
%Calcula frecuencia corte wc
```

```
Omega_c = (w_stop + w_pass)/(2*f_sampler)
%calcula tau
T = (N-1)/2
M = T;
%calcula coeficientes ideales causales
for i=1:(2*T)+1
    if i == M+1
        h_LP(i) = Omega_c / pi;
    else
        h_LP(i) =(sin(((i-1)-T)*Omega_c))/(((i-1)-T)*pi);
    end
End
%calcula coeficientes de ventana kaiser
w_kais = kaiser(N,Beta);
%calcula coeficientes "KERNEL" del filtro
%h(n)=h_ideal(n)*w(n)
for i=1:(2*T)+1
    h(i)=h_LP(i)*w_kais(i);
end
%truncacion 16 bits
B = 16;
for i=1:(2*T)+1
    ht(i) = (ceil(h(i)*(2^(B-1)-1))/h(T))* (h(T)/(2^(B-1)-1));
end
disp 'Coeficientes truncados 16 bits';
disp '******************************';
for i=1:(2*T)+1
    fprintf('h[%d]= %3.8f\n',i-1,ht(i))
end
%función de transferencia
Hz=tf(ht,1,'variable','z^-1')
%gráfica respuesta a la frecuencia
[num,den] = tfdata(Hz,'v');
[H,w]=freqz(num,den,128);
plot(w/(2*pi)*f_sampler,abs(H),'-k')
ylabel('Magnitud');
xlabel('frecuencia (Hz)')
title('Respuesta a la frecuencia');
figure(2);
plot(w/(2*pi)*f_sampler,20*log(abs(H)),'-k')
ylabel('Magnitud dB');
xlabel('frecuencia (Hz)')
title('Respuesta a la frecuencia log');
%gráfica fase
figure(3);
plot(w/(2*pi)*f_sampler,angle(H),'-k')
title('Respuesta a la fase');
ylabel('Fase');
xlabel('frecuencia (Hz)')
%Respuesta al impulso
[h,t]=impz(num,den,128);
figure(4);
plot(t,h,'-k');
title('Respuesta al impulso');
%gráfica respuesta al escalón
[h,t]=stepz(num,den,128);
figure(5);
plot(t,h,'-k');
title('Respuesta al escalon');
```

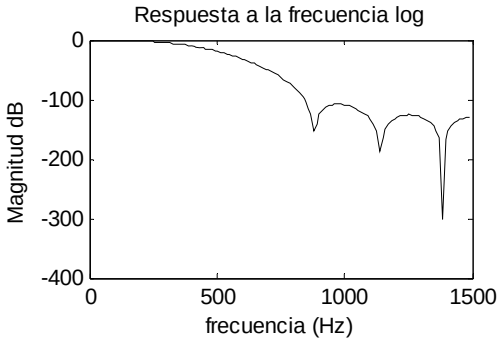
Si se reduce el tamaño de la ventana estimada puede ocurrir un offset DC en la salida del filtro. En el ejemplo de arriba el tamaño óptimo de la ventana es 9 pero por motivos de tiempo de procesamiento se ha escogido 7.

Al analizar la respuesta a la frecuencia en la escala logarítmica entre los filtros IIR y los filtros FIR podemos darnos cuenta de la eficiencia de los filtros FIR para atenuar las frecuencias no deseadas, pero esta eficiencia se ve muy opacada el momento de implementar en *Real – Time* un filtro FIR, ya que necesitan mucho calculo para realizar la convolución, una manera de contrarrestar el tiempo de procesado del algoritmo para un filtro FIR es utilizar un filtro de menor orden, al hacer un filtro de

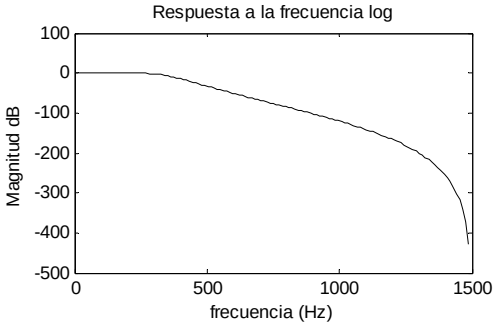
menor orden la característica de respuesta a la frecuencia ya varia y no atenuara tajantemente.

En las siguientes gráficas se comparan las respuestas de frecuencia de filtros FIR diseñados y el filtro IIR diseñado anteriormente

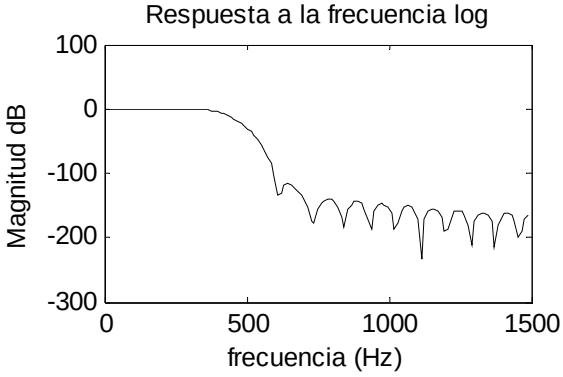
Filtro FIR orden 13



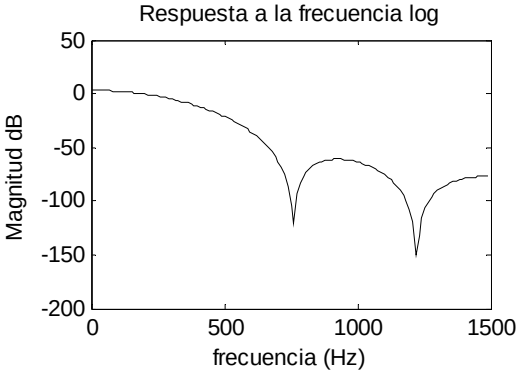
Filtro **IIR** orden 4



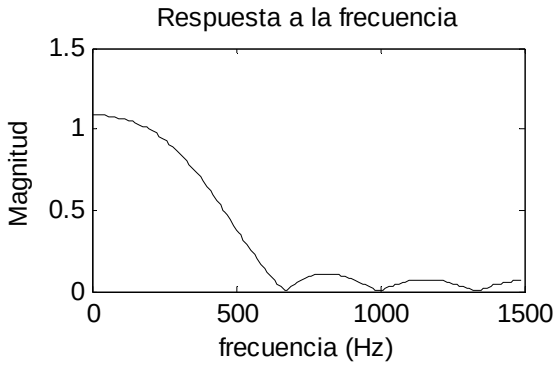
Filtro FIR orden 33

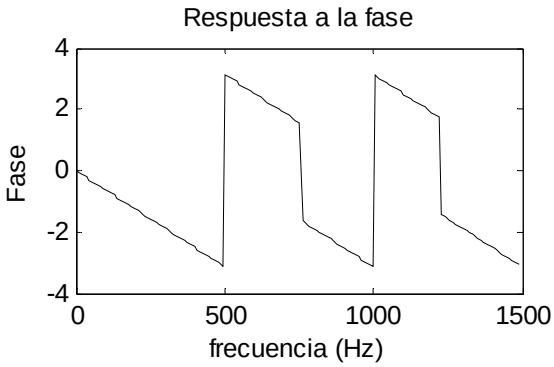


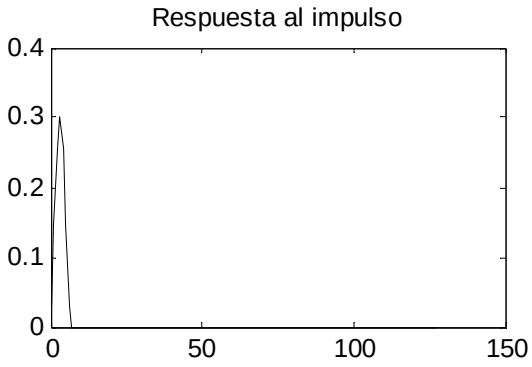
Filtro **FIR** orden 7(diseñado)

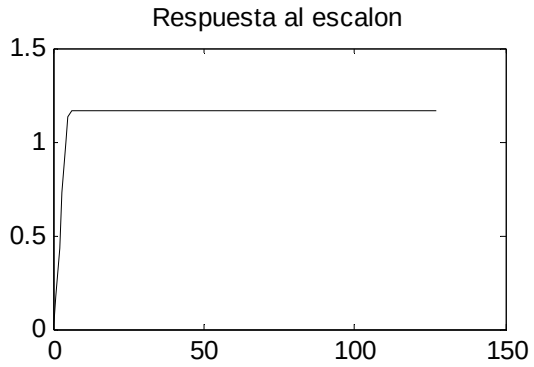


Características del filtro diseñado










```
Coeficientes truncados 16 bits
******************************
       h[0]= 0.03085421
       h[1]= 0.14734336
       h[2]= 0.25580615
       h[3]= 0.30002747
       h[4]= 0.25580615
       h[5]= 0.14734336
       h[6]= 0.03085421
```

Si se diseña el filtro utilizando el software *dsPIC FD Lite* puede observarse que los coeficientes diseñados son más suavizados, como se muestra a continuación

```
0.2770996093750000E-01,
0.1278991699218750E+00,
0.2176208496093750E+00,
0.2535095214843750E+00,
0.2176208496093750E+00,
0.1278991699218750E+00,
0.2770996093750000E-01
```

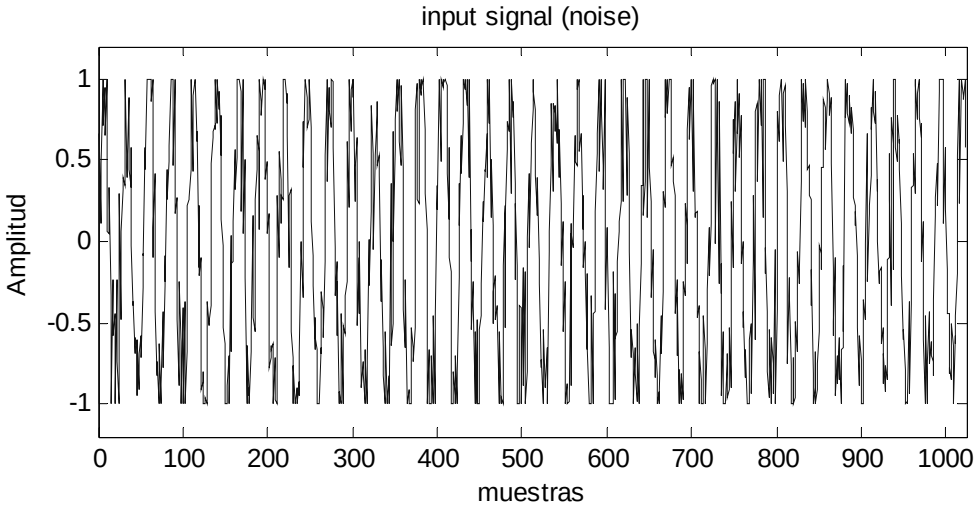
Esta diferencia se debe a que el filtro diseñado utilizando el script nos aconsejaba una ventana de mínimo 9 puntos, pero tuvimos que reducirle a 7 puntos, pero si diseñamos el filtro con 9 puntos los coeficientes son bastante más parejos.

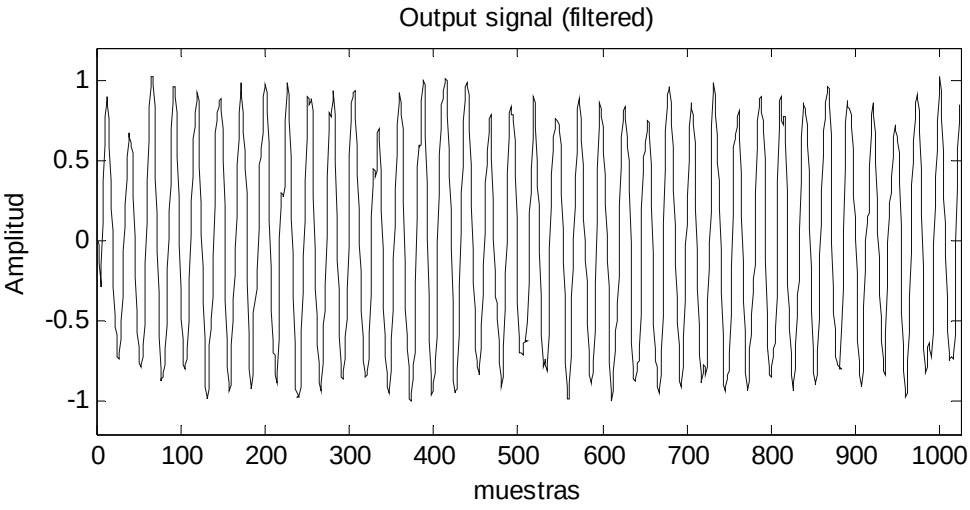
En conclusión a las pruebas realizadas el software *dsPIC FD Lite* esta utilizando algún algoritmo para suavizar los coeficientes calculados? …

5.3.2 Simulación del filtro con *MATLAB*

Código …

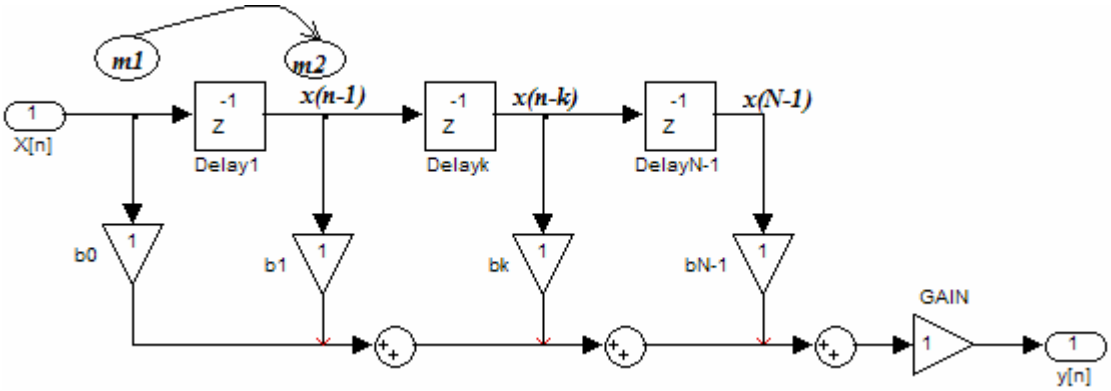
```
% lee una onda desde un archivo wav
y = wavread('input.wav'); % input.wav onda generada en dsPICworks
%gráfica señal de entrada
figure(7);
plot([0:length(y)-1],y)
title('input signal (noise)');
xlabel('muestras');
ylabel('Amplitud');
% Analiza coeficientes del filtro
Y = filter(num,den,y);
% gráfica señal de salida
figure(8);
plot([0:length(Y)-1],Y)
title('Output signal (filtered)');
xlabel('muestras');
ylabel('Amplitud');
```





5.3.3 Código en "C" para implementar filtros digitales *FIR*

En el siguiente diagrama se muestra el sistema de un filtro digital FIR



Como se puede apreciar en el diagrama el filtro FIR no tiene retroalimentación (*feedback*), la configuración del diagrama representa una convolución envolvente de *N* coeficientes y esta descrita por la siguiente ecuación

$$y[n] = g \cdot \sum_{k=0}^{N-1} x[n-k] * b[k]$$

Implementación *Real – Time*

Aunque el algoritmo luce muy censillo para implementar, el procedimiento puede ser complicado de alguna forma por el efecto de guardar ***N-1*** estados de memoria ***m*** (valores de muestras pasadas), además dependiendo del tamaño de la ventana los cálculos para procesar una muestra será mucho más largos que en un filtro IIR. Para implementar es necesario refrescar el estado de memoria después de cada convolución, esto se puede hacer dentro del mismo *loop* si se desarrolla la convolución en sentido reverso, una manera simple de hacer esto es poner en orden reverso los coeficientes y los estados de memoria y guardarlos en arreglos.

$$Coeficientes \;\; C = \{bn-1,....,b2,b1,b0\}$$

$$Memoria \;\; M = \{xn-1,....,x2,x1,x0\}$$

El segmento de código mostrado abajo muestra la implementación básica

```
m[N-1] = x;
o = a[0] * m[0];
for(k = 1; k <= N ;k++)
{
      m[k-1] = m[k];
      o += a[k] * m[k];
}
o *= gain;
```

Analizando el código como se hizo en el filtro IIR el uso de muchos índices en los cálculos es ineficiente y quita velocidad, por lo que se recomienda utilizar punteros para implementar el algoritmo de filtrado.

El código en **"C"** para implementar filtros FIR quedaría así:

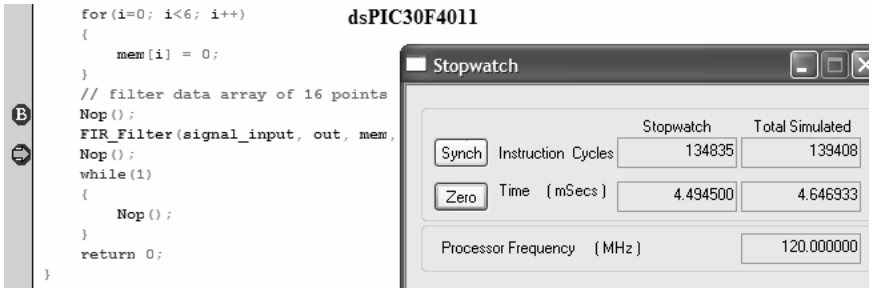
```
/*================================================
FIR Filter function using pointers Original by Les Thede
Arguments:  x - ptr input array
            Y - ptr output array
            M - ptr to memory array
            C - ptr to coefs array
            numb_coef – number of coefficients
            N – number of samples at input array
================================================*/

void FIR_Filter(int *X,int *Y,float *M,float *C,int numb_coef,int N)
{
      unsigned int res;
      int n;
      int *input,*y,          /* ptrs to in/out arrays */
      i,j;                    /* loop counters */
      float *c,*m1,*m2,       /* ptrs to coef/memory array */
      temp;                         /* output value */
      // Make copies of input and output pointers
      input = X;
      y = Y;
      /* Start loop for number of data values */
      for(i = 0; i < N ;i++)
      {//1
            // make present input as last
            M[numb_coef-1] = *input++;
            // Make copy of pointers and start loop
            c = C;
            m1 = m2 = M;
            // compute the first sample m[0]*b[0]
            temp = *m1++ * *c++;
            /* Use convolution method for computation */
            for(j = 1; j < numb_coef ;j++)
            {//2
                  *m2++ = *m1;
                  temp += (*m1++) * (*c++);
            }//2
            /* convert to int and store */
            *y++ = (int)(temp);
      }//1
}


//==================================
// FIR_Filter global variables
// B coefficients numb_coef = ;
float bn[] = {

      };
// memory
float Mem[13];
// output sigal - filtered
int out[128];
```

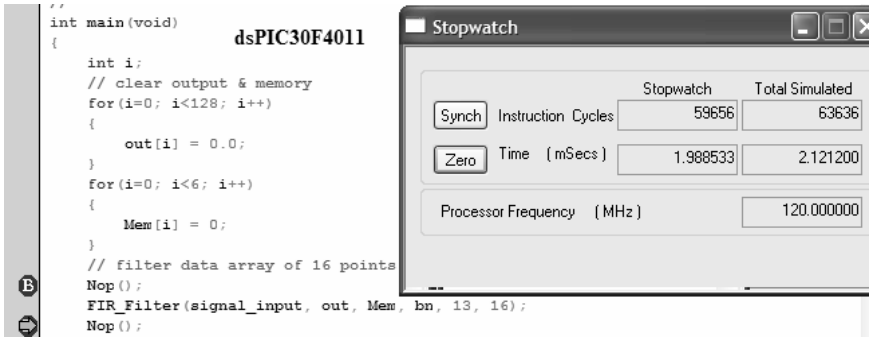
5.3.4 Comparación de desempeño

La comparación de desempeño se ha realizado sobre la función *FIR_Filter ( )* en un *dsPIC* variando el tamaño o orden del kernel (ventana).

SIMULACION **dsPIC30F4011**	Algoritmo *FIR_Filter()*	
Instrucciones	59656	134835
Frecuencia Procesador	120 Mhz	120 Mhz
Tiempo	1,988 milisegundos	4,494 milisegundos
# muestras	16	16
Tamaño Kernel	13	33

En las gráficas de abajo se muestran los puntos de simulación



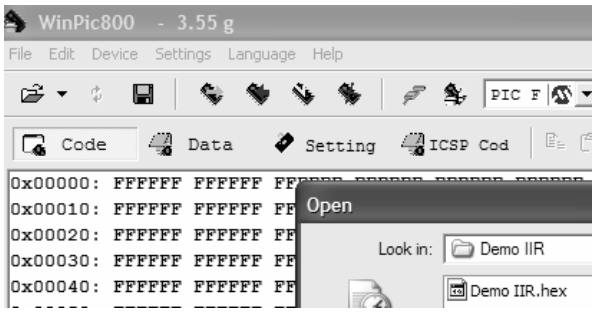


La simulación nos da una clara referencia del tiempo de procesamiento de los filtros digitales FIR.

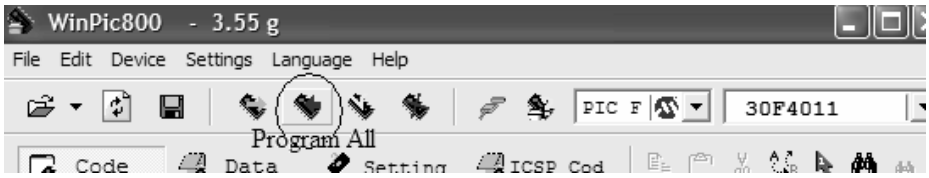
5.4 Programación del dsPIC

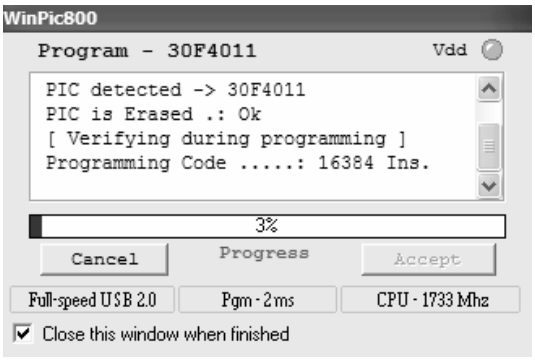
Para programar los dsPIC se utiliza el MPLAB IC2 que tiene un precio elevado o utilizando un programador de terceros que puede conseguirse por unos $50, el programador que se utilizara será el *GTP – USB Lite*, este viene con el software *WinPIC800* el cual nos permite grabar el procesador con alguna aplicación compilada de extensión *.hex*.

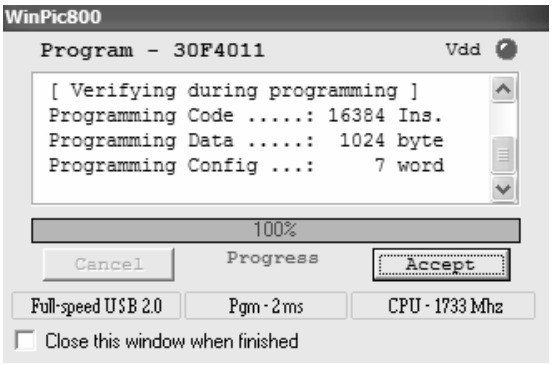
Se ha puesto a disposición 2 firmwares para poder probar los filtros digitales con Matlab, la una aplicación tiene un filtro digital IIR *Demo IIR.hex* y la otra tiene un filtro digital FIR *Demo FIR.hex* estos deben ser utilizados solo con la tarjeta *dsPIC-4ks* y son los filtros diseñados en este capitulo pero ya implementados en el dsPIC. Para programar el dsPIC abrimos el programa *WinPIC800* y cargamos la aplicación a grabarse



El siguiente paso es hacer clic en *Program All* para grabar la aplicación en el dsPIC, el programador detecta automáticamente que procesador se va a programar.



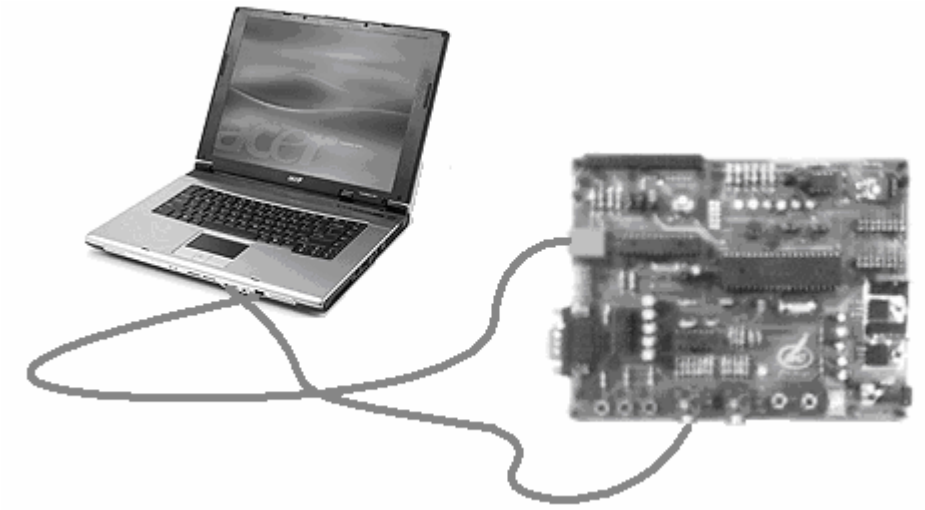




Esta es la última etapa en el diseño de una aplicación con microcontroladores, solo 2 pasos se necesitan para grabar una aplicación ya diseñada en un procesador, el programa es muy intuitivo en el manejo como todo programa en Windows.

5.5 Análisis y adquisición con Matlab de los filtros digitales

Para analizar los filtros digitales hay que grabar al dsPIC con los firmwares *Demo IIR.hex* y *Demo FIR.hex* el script en Matlab será el mismo para los 2.



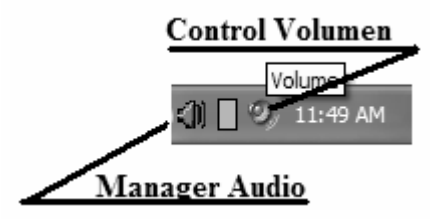
Hasta ahora los filtros solo se han simulado en la PC, en esta parte analizaremos los filtros implementados en el procesador de la tarjeta *dsPIC-4ks*, para esto generaremos una señal desde la tarjeta de sonido y la tarjeta *dsPIC-4ks* muestreara la señal, procesara (*filtrara*) esas muestras y las mandara al PC para analizar el comportamiento del filtro digital.

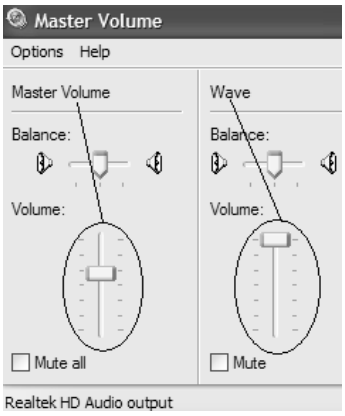
Configuración de la tarjeta de sonido de la PC

Antes de empezar a generar los sonidos a través de la tarjeta de sonido de la PC es necesario configurar algunos parámetros de esta como las ganancias y la desactivación de los efectos de sonido.

Para desactivar los efectos de sonido ir al manager de audio de la tarjeta de sonido y resetear todo, el ecualizador debe quedar plano (*flat*)

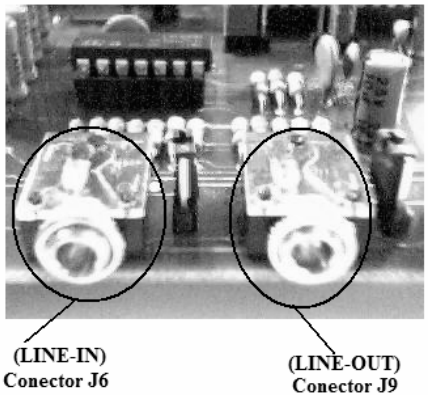
Para ajustar la ganancia ir a los controles de volumen y ajustar las barras del master volumen y wave como se muestran en la siguiente figura

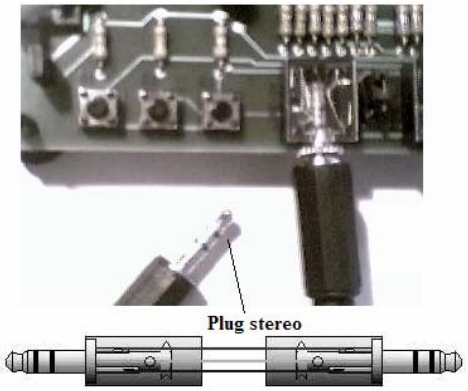


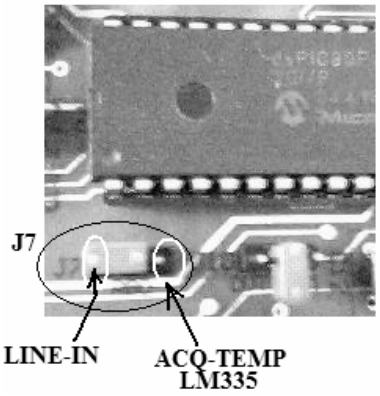


Conexión de la PC con la tarjeta *dsPIC – 4ks*

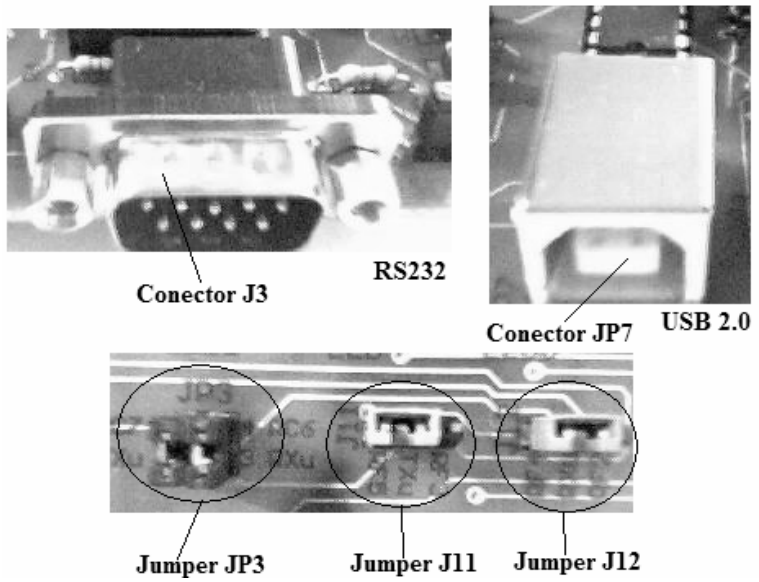
Para adquirir las señales generadas por la tarjeta de sonido con la tarjeta dsPIC – 4ks conectar un cable stereo en el conector J6 (LINE-IN) y revisar la posición del jumper J7 (*RB0*)







Para la comunicación de la PC con la tarjeta dsPIC – 4ks conectar un cable DB9 al conector J3 o conectar un cable USB en el conector JP7, si se utiliza el conector USB para comunicar la PC con la tarjeta revisar los jumpers (JP3, J11 y J12), estos jumpers seleccionan el puerto UART que se comunica con el firmware.



Script Matlab "ACQ_dsPIC4ks_board.m"

El siguiente script creara un menú para seleccionar el tipo de onda a generar a través de la tarjeta de sonido de la PC y se comunicara con la tarjeta para adquirir los valores del procesamiento.

```
%DSP LABORATORY - UDA
%test ok communications OK
clear all;
clc;
limit =60;
disp '   ***************************************'
disp '   *         dsPIC-4ks DSP Board           *'
disp '   *        UNIVERSIDAD DEL AZUAY          *'
disp '   *                              DSP Lab. *'
disp '   ***************************************'
disp 'Menu waveform generator:'
disp ' '
disp '      play WAV archive [1]         sine wave [2]'
disp '            sine wave + white noise [3] '
disp '____________________'
in = input('Select: ','s');
disp '____________________'
if in == '1'
    %play wav from archive
    onda = input('Data wav to play(.wav): ','s');
    out = wavread(onda); % input.wav dsPICworks maybe
elseif in == '2'
    onda = input('Sine wave frequency(Hz): ','s');
    f = str2num(onda);
    t=0:1/3000:2; %2 seg max play time
    out = sin(2*pi*f*t);
else
    onda = input('sine wave + wn frequency(Hz): ','s');
    f = str2num(onda);
    t=0:1/3000:2; %2 seg max play time
    y = sin(2*pi*f*t);
    randn('state',0);
    out = y + 0.5*randn(size(t));
end
reply = input('Acquisition Time (sg):','s');
if isempty(reply)
    reply = '0';
end
for i=limit;
  acq(i) = 0;
end
%USB CDC emulation interface
```

```
s = serial('COM4','BaudRate',230400); %crea serial port
s.InputBufferSize = 36000; %configura buffer entrada
s.ReadAsyncMode = 'continuous';
if (s.Status == 'closed')   %verifica si esta abierto serial
port
    fopen(s);   %abre serial port
end

disp('Wait please . . .');
n = str2num(reply);
t = timer('TimerFcn','stat=false;',...
                'StartDelay',n, 'Period', 0.001);
i = 1;
fprintf(s,'a');
start(t)
stat=true;
sound(out);
pause(0.005);
%get(t,'Running') %timer status
%tic %eval your code
%sound(out);
disp('Acquiring data . . .');

%*********your code************
while(stat==true)
  while ( s.BytesAvailable <= 4)
  end
  %read data
  input = fscanf(s,'%c',4);
  acq(i) = str2num(input);
  %acq(i) = str2num(read_usb(s,4));
  i = i + 1;
end
fprintf(s,'s');
while (s.BytesAvailable >= 4)
  input = fscanf(s,'%c',4);
  acq(i) = str2num(input);
  i = i + 1;
end

%monitor for your data
disp('Processing data . . .');
figure(2)
  plot(1:length(acq),(acq*(4.96/1024))-2.5);
  axis([0 length(acq) -2.5 2.5]);
  title('input signal fs to 3khz');
  ylabel('Amplitud(Volts)');
  xlabel('samples');
  drawnow

%plot(1:length(acq),acq)
%********end your code**********
%toc %eval your code
%get(t,'Running') %timer status
delete(t);
if (s.Status == 'open')%verifica si esta abierto serial port
    fclose(s);   %cierra serial port
end
delete (s);
```

Al hacer correr este script nos saldrá un menú en el command window de Matlab, como se muestra a continuación:

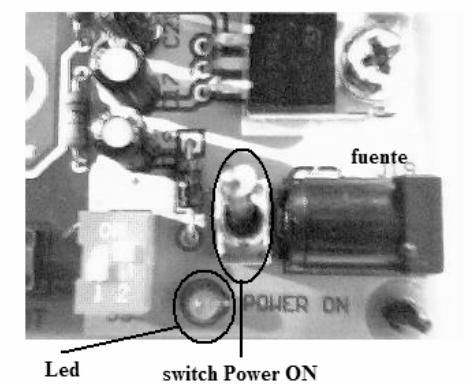
Es un menú muy intuitivo que nos permitirá usar desde ondas guardadas en archivos wav hasta generar ondas con ruido de diferente frecuencia.

```
*****************************************
*          dsPIC-4ks DSP Board          *
*         UNIVERSIDAD DEL AZUAY         *
*                                DSP Lab. *
```

```
   ****************************************
Menu waveform generator:

      play WAV archive [1]         sine wave [2]
            sine wave + white noise [3]

____________________
Select:
```

Análisis de las señales filtradas y no filtradas

Una vez grabado el dsPIC 30F4011 con los firmwares *Demo IIR.hex* o *Demo FIR.hex* encender la tarjeta dsPIC – 4ks, el led rojo deberá estar encendido



Abrir Matlab, hacer correr el script ACQ_dsPIC4ks_board.m y seleccionar los parámetros como se indica en la gráfica de abajo

Command Window

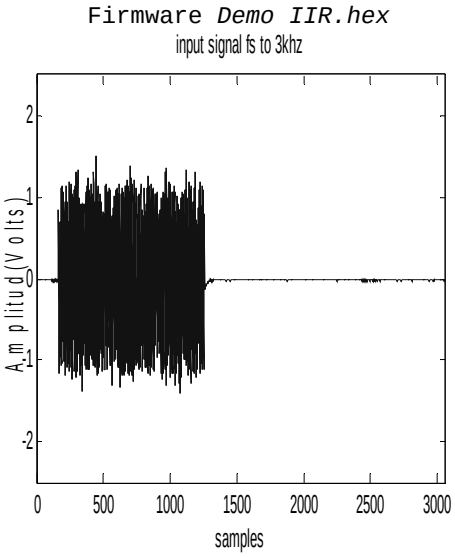
```
   ****************************************
   *          dsPIC-4ks DSP Board          *
   *         UNIVERSIDAD DEL AZUAY         *
   *                              DSP Lab. *
   ****************************************
Menu waveform generator:

      play WAV archive [1]         sine wave [2]
            sine wave + white noise [3]

____________________
Select: 1
____________________
Data way to play(.wav): input_300.wav
Acquisition Time (sg):1
Wait please . . .
Acquiring data . . .
Processing data . . .
>>
```

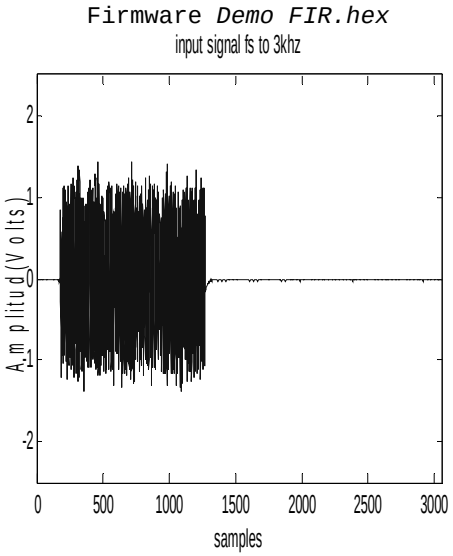
Seleccionar 1 para reproducir un archivo wav que contiene una onda se de 300 Hz con ruido

onda seno 300Hz con ruido

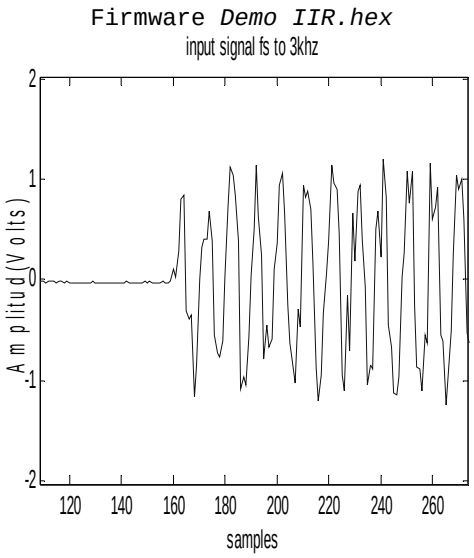
tiempo de adquisicion

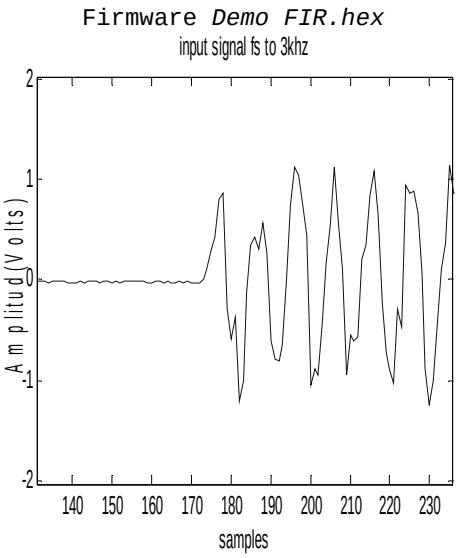
Las gráficas de abajo son las señales adquiridas a través de la tarjeta dsPIC – 4ks



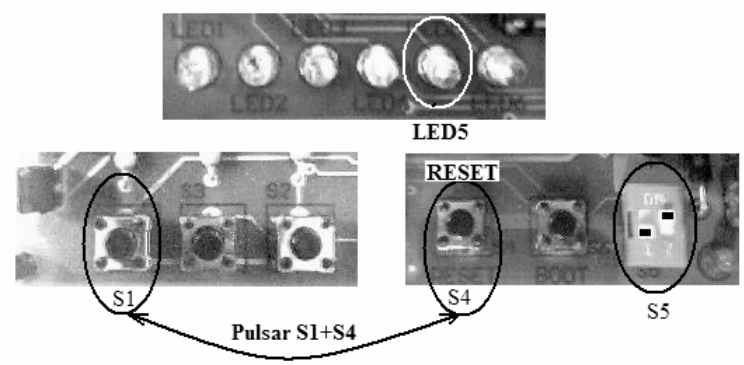


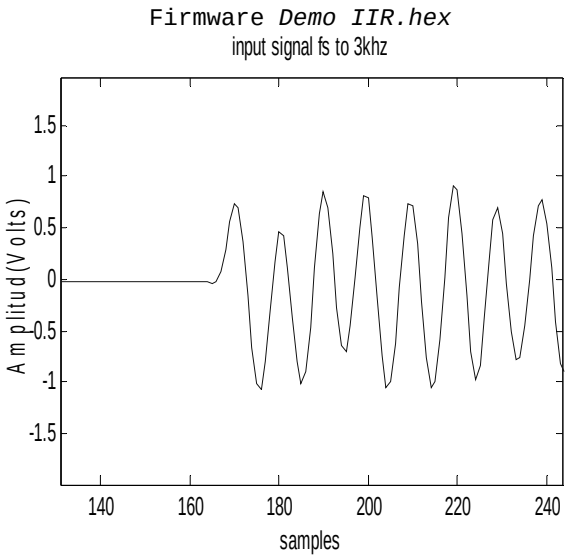
Haciendo un zoom en las gráficas veremos mejor las señales adquiridas

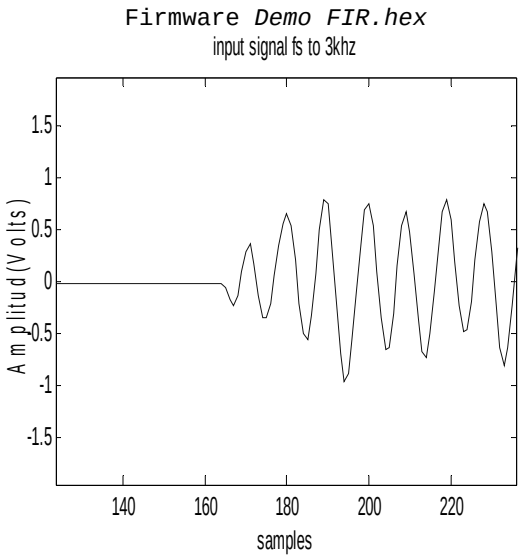




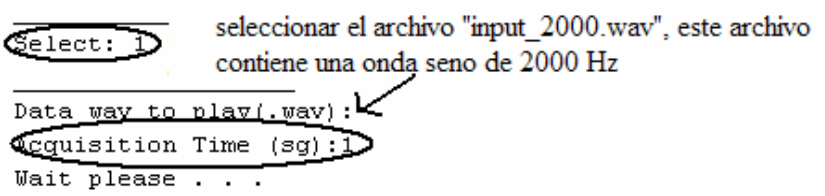
Como se puede observar las señales siguen con ruido, esto se debe a que no seleccionamos el filtro digital, para poder restaurar las señales usando el filtro digital es necesario ajustar el switch *S5* (ver gráfico) y pulsar *S1 + S4* (*RESET*) si todo esto es realizado el *LED5* deberá encenderse

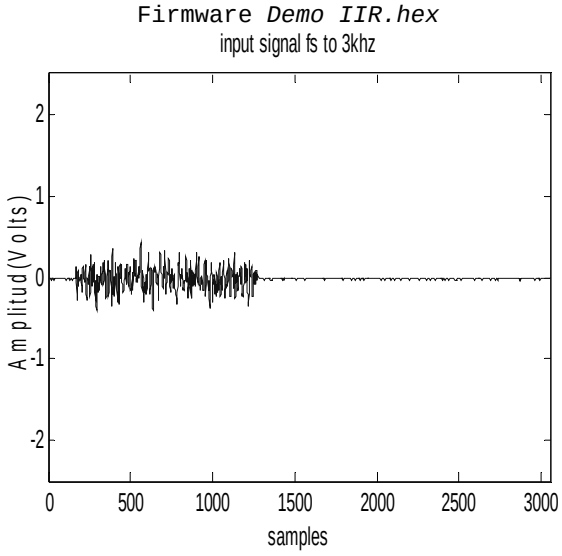


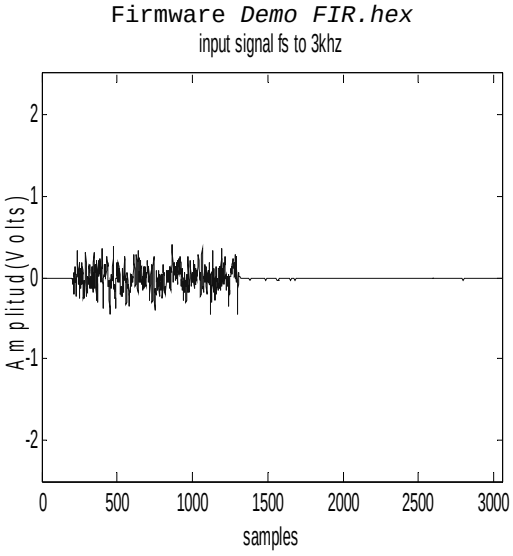




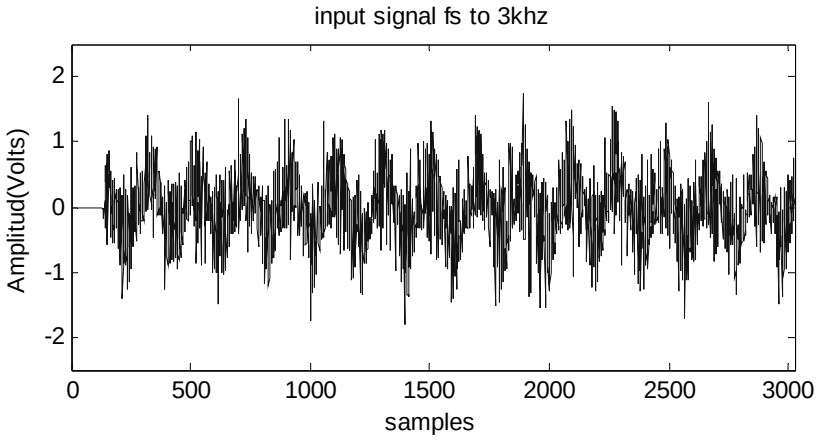
Para ver si rechaza y atenúa las señales superiores a 600Hz



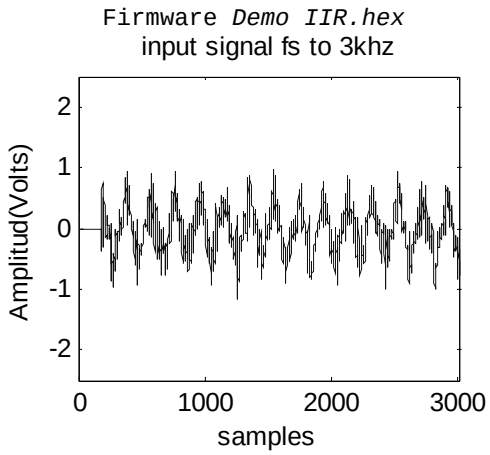




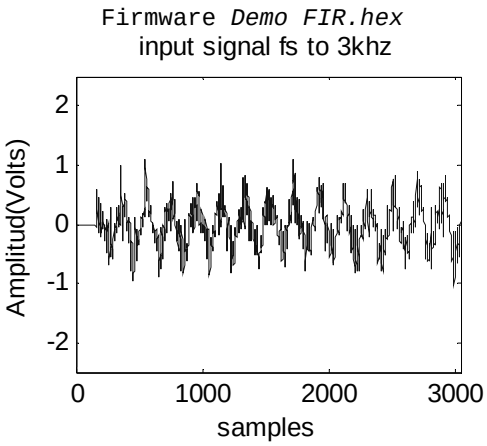
Onda seno con white-noise, señal sin filtrar



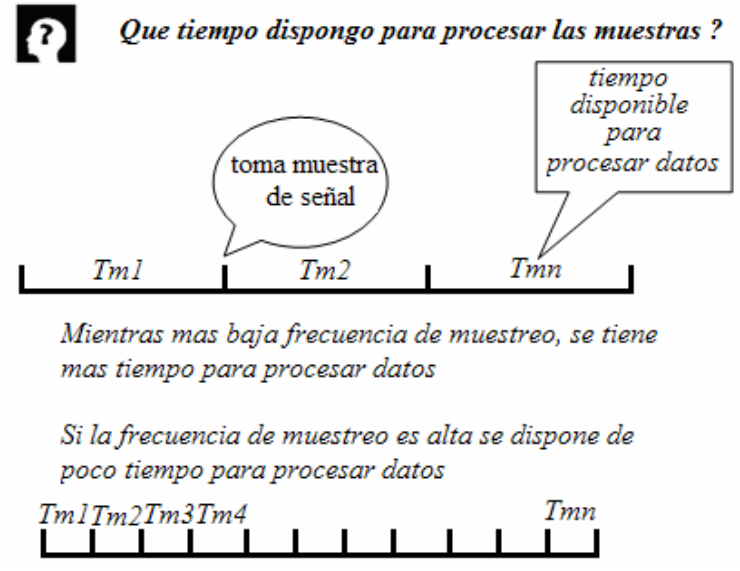
Firmware *Demo IIR.hex*



Firmware *Demo FIR.hex*



Como se puede ver en las gráficas los filtros están funcionando como restauradores y como rechazadores de señales, si se desea una mejor atenuación deberá seleccionarse un orden mayor del filtro pero se verá afectado el tiempo de procesamiento.



La velocidad de muestreo que se seleccione es un factor muy importante como se puede observar en la gráfica de arriba, si se muestrea a una frecuencia alta se dispone de poco tiempo para poder procesar la señal, si se utiliza un filtro de mayor orden el tiempo de procesamiento crecerá y el periodo de muestreo se vera afectado. El número de caracteres por segundo que se puedan enviar al PC influirá en el muestreo que se deba usar, es necesario recordar que el puerto serial RS232 puede transmitir un cierto número de caracteres por segundo y no esta disponible todo el acho de banda. La fórmula para calcular los caracteres por segundo que se pueden transmitir es

$$Ch/sg = \frac{baud}{11}$$

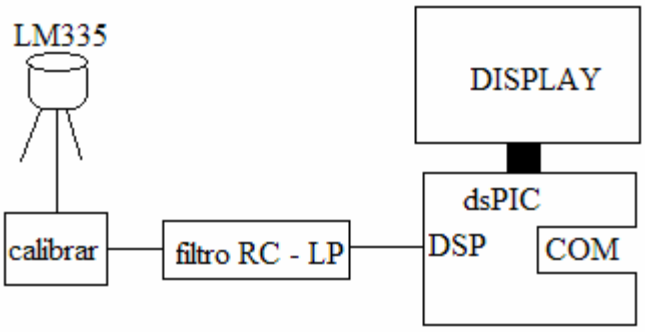
En nuestro caso se utilizo 230400 bps

$$Ch/sg = \frac{230400}{11} \approx 20945$$

Se pueden transmitir máximo 20945 caracteres por segundo pero se debe dividir para 4 porque el ADC es de 10 bits y cada dato es convertido a decimal y mandado como ASCII (Matlab solo reconoce ASCII).
Si se hace uso de la memoria EEPROM I2C de la tarjeta, la velocidad de muestreo puede aumentarse ya que los valores muestreados se iran guardando en esta y luego serán procesados sin importar el orden de este.

5.6 Implementación práctica de un filtro digital en la adquisición de una señal de un sensor analógico de temperatura

Utilizando la tarjeta dsPIC – 4ks implementaremos un filtro digital en un sistema Real Time para adquirir datos del sensor analógico de temperatura *LM335*, se utilizará una pantalla gráfica para visualizar los cambios de temperatura.
En el gráfico de abajo se muestra la adquisición y el procesamiento del sensor

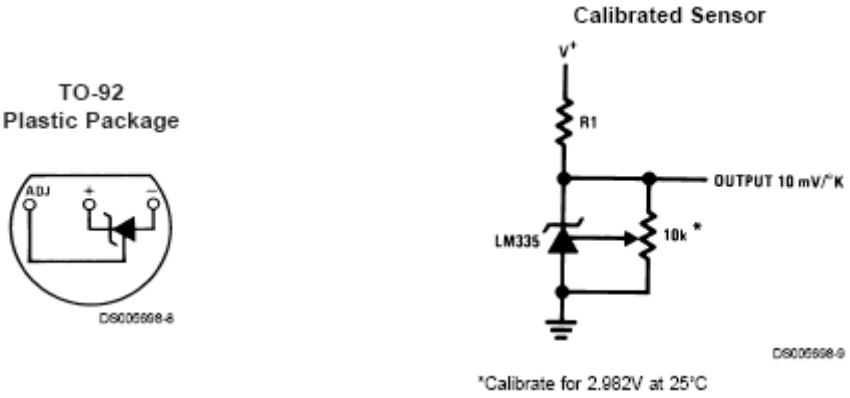


5.6.1 Descripción técnica del sensor analógico de temperatura LM335

El LM335 es un sensor de precisión fácil de calibrar, opera como un diodo zener que cuando esta ajustado produce entre sus extremos un voltaje proporcional a la temperatura.

La salida es de *0 Voltios* a *0* 0^0K (273^0C) y de *2,73 Voltios* a 0^0C, con incremento de $+0{,}10mV$ por cada ^0C. Tiene una precisión de $\pm 1^0C$ y un rango de medida de -40^0C hasta 100^0C, opera a una corriente de $400\mu A$ a $5mA$.

En el gráfico de abajo se puede observar el diagrama de conexiones del LM335



La calibración es muy sencilla, usando el potenciómetro *R25* se corrige la impresición del sensor sobre el rango de temperatura.

La salida del sensor (calibrado o no) puede ser expresada como

$$Vout_T = Vout_{To} \cdot \frac{T}{To}$$

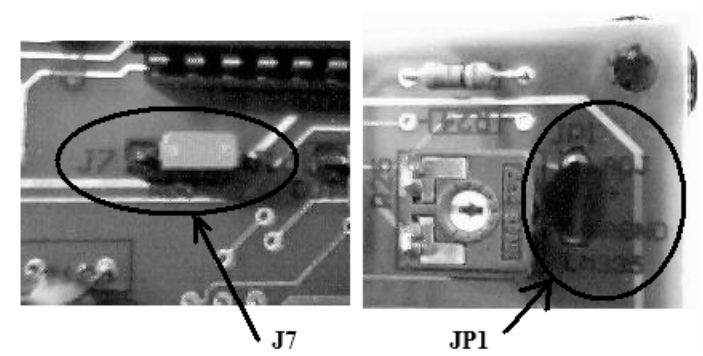
Donde *T* es la temperatura desconocida y *To* es una temperatura de referencia ambas expresadas en grados Kelvin. Nominalmente la salida esta calibrada a $10 {}^{mV}\!/_{{}^0K}$.

Para convertir de grados kelvin a grados centígrados se usa la siguiente relación

$$^0C = ^0K - 273.15$$

5.6.2 Tarjeta dsPIC – 4ks y el sensor de temperatura LM335

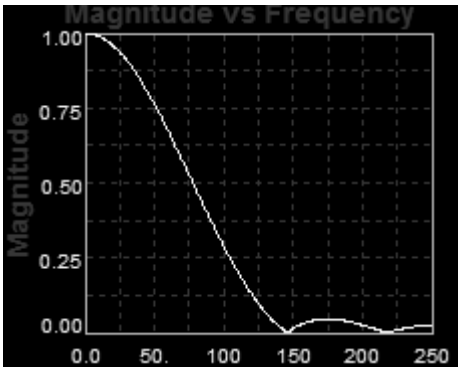
Colocar el sensor de temperatura en el conector *JP1*, el jumper *J5* selecciona o no el filtro analógico pasa bajas para la entrada del filtro y por ultimo cambiar de posición el jumper *J7* seleccionando la entrada del sensor de temperatura.

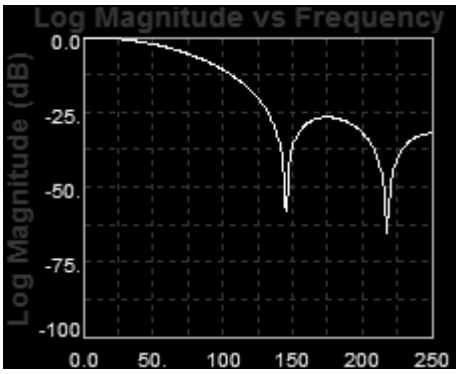


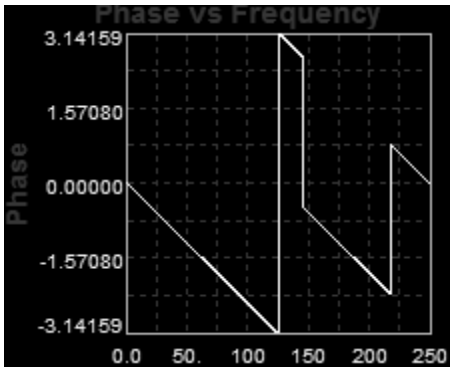
5.6.3 dsPIC FD Lite

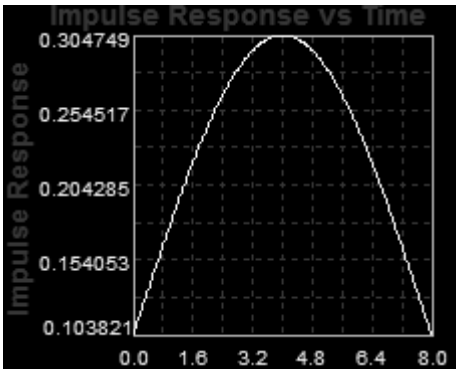
Diseñar un filtro FIR pasa bajas usando dsPIC FD Lite con una ventana rectangular de 5 puntos de longitud, para las siguientes características

$$a_{pass} = -1dB\,,\, a_{stop} = -21dB\,,\, f_{pass} = 60 - Hz\,,\, f_{stop} = 120 - Hz\,,\, f_s = 500 - Hz$$









Los coeficientes generados se muestran en la tabla de abajo

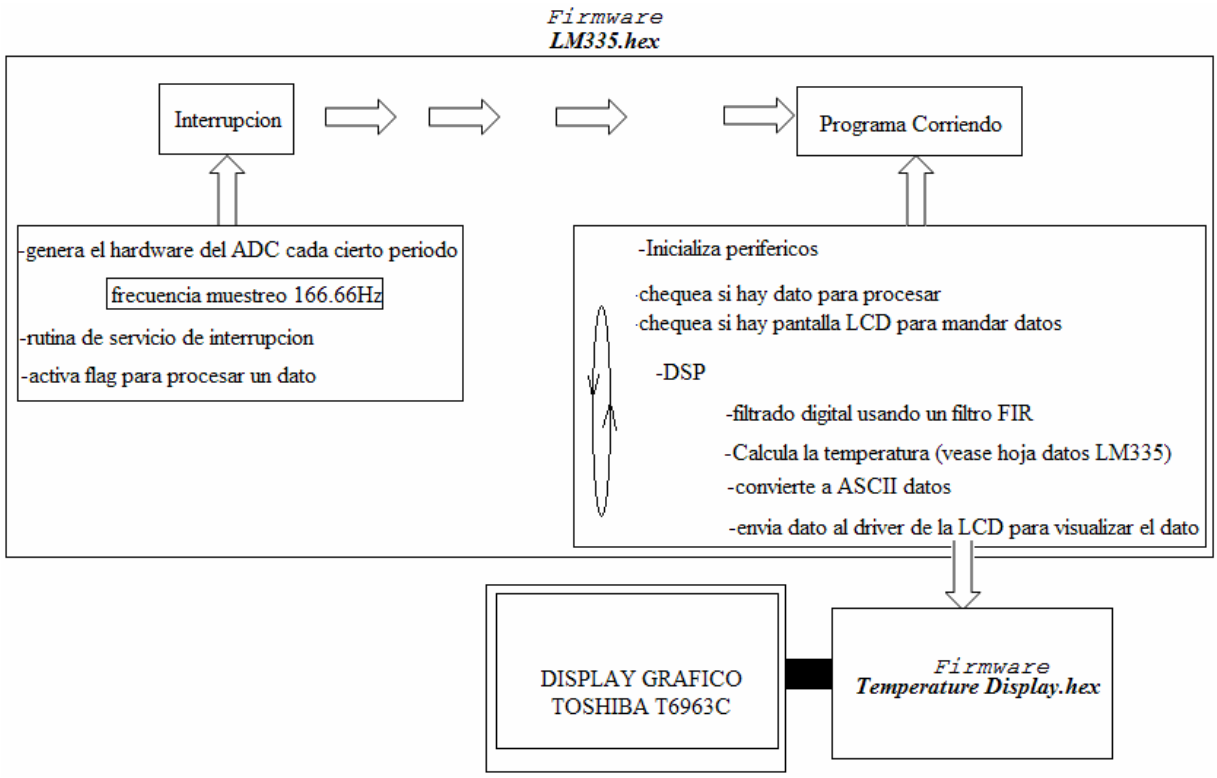
```
FILTER COEFFICIENT FILE
FIR DESIGN
SAMPLING FREQUENCY          0.500000E+03 HERTZ
   5                  /* number of taps in decimal     */
   5                  /* number of taps in hexadecimal */
16                    /* number of bits in quantized coefficients
  10                  /* number of bits in quantized coefficients
 0                    /* shift count in decimal  */
   0  0.100000000E+01 /* shift count (hex), gain multiplier */
       3402      D4A  /* coefficient of tap      0 */
       7989     1F35  /* coefficient of tap      1 */
       9986     2702  /* coefficient of tap      2 */
       7989     1F35  /* coefficient of tap      3 */
       3402      D4A  /* coefficient of tap      4 */
0.103820800781250E+00 3FBA940000000000/* coefficient of tap
0.243804931640625E+00 3FCF350000000000/* coefficient of tap
0.304748535156250E+00 3FD3810000000000/* coefficient of tap
0.243804931640625E+00 3FCF350000000000/* coefficient of tap
0.103820800781250E+00 3FBA940000000000/* coefficient of tap
```

coeficientes

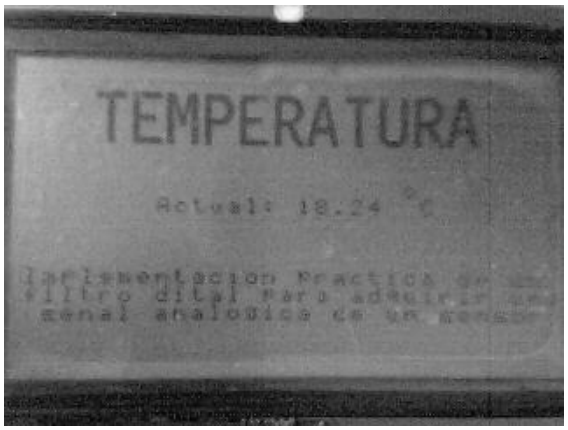
5.6.4 Diagrama de la aplicación

La aplicación adquirirá la señal analógica del sensor de temperatura, procesara las muestras y después mandara al display gráfico para visualizar las muestras en la pantalla.
Se eligió un filtro digital FIR porque es el más óptimo para tareas comunes como la reducción de ruido aleatorio, además la aplicación no necesitara separar frecuencias.

En el gráfico de abajo se muestra como trabaja el firmware *LM335.hex*



En la gráfica de abajo se muestra la aplicación, visualizando la temperatura



Conclusiones

El objetivo de esta tesis ha sido introducir los filtros digitales, específicamente la clase de filtros digitales que se implementan por recursión y los que se implementan por convolución.
En los filtros digitales implementados por recursión llamados IIR, el diseño parte de un aproximante digital, luego se realizan desnormalizaciones y luego se utiliza la transformada bilineal para obtener la función de transferencia del sistema digital y los coeficientes del filtro.
Se estudio los filtros digitales implementados por convolución de TIPO I, tienen longitud impar y coeficientes simétricos, esta clase de filtros son llamados FIR, el diseño de estos consiste en hallar los coeficientes ideales y luego multiplicar estos coeficientes por una ventana para reducir el número de coeficientes, al resultado de este producto se le llama *kernel* éste se convoluciona con la señal de entrada y se obtiene la señal filtrada.
El otro objetivo de esta tesis fue la de introducir la nueva tecnología de microcontroladores especializados para tareas de DSP de Microchip y utilizarla de plataforma para implementar los filtros digitales, desarrollando un kit de entrenamiento básico para DSP con conexión a la PC para análisis de los sistemas montados, el uso de MATLAB con la PC dio una ventaja potencial con gran flexibilidad, se utilizo para diseño, simulación, generación de señales a través de la tarjeta de sonido de la PC y adquisición de datos.
Pero queda aun por probar en laboratorio otros programas como Labview, octave o Scilab.

Para lograr estos objetivos la tesis se ha estructurado en 5 partes:
1) *Introducción*
En la primera parte se ha hecho una revisión de conceptos fundamentales en sistemas lineales como son: la convolución, la respuesta al impulso, la transformada de Laplace, la transformada – z , etc. que son la base para comprender un sistema de filtrado.

2) *Filtros Digitales*
En la segunda parte se ha hecho una introducción a los parámetros del dominio del tiempo y la frecuencia, los tipos de implementación de los filtros digitales y el diseño de filtros digitales IIR y FIR dentro de los cuales se encuentran algunos tipos de filtros.

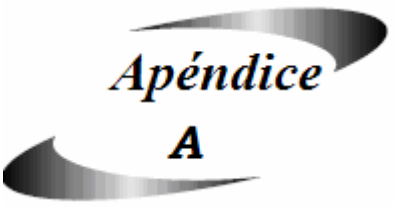
3) */ 4) DSC / Diseño y herramientas de desarrollo*
En estas partes se hizo una breve introducción a la arquitectura del controlador digital que servirá de base para implementar los filtros digitales, a las herramientas que se utilizan para desarrollar aplicaciones y el diseño del hardware para experimentar aplicaciones de DSP.

5) *Implementación FD*
Esta parte es totalmente experimental, se diseñaron 2 filtros digitales IIR e FIR se simularon los coeficientes para ver como se comportan los filtros, se explico la manera de implementarlos en el dsPIC y se hizo uso de un PC para generar señales, para analizar los datos muestreados y probar los filtros.

Este trabajo marca una base fundamental para la futura experimentación de aplicaciones de DSP en temas como procesamiento de voz y procesamiento de imágenes. Se a dado una pauta de cómo se diseñan proyectos con microcontroladores y como debe responder un ingeniero en estos tiempos, el trabajo con aplicaciones de DSP y microcontroladores demanda una rápida respuesta por parte del diseñador en un campo muy competente, para obtener una rápida respuesta es necesario estar actualizado al día en tendencias para desarrollo como el uso de CAD's e IDE's que nos facilitan el desarrollo de una aplicación.

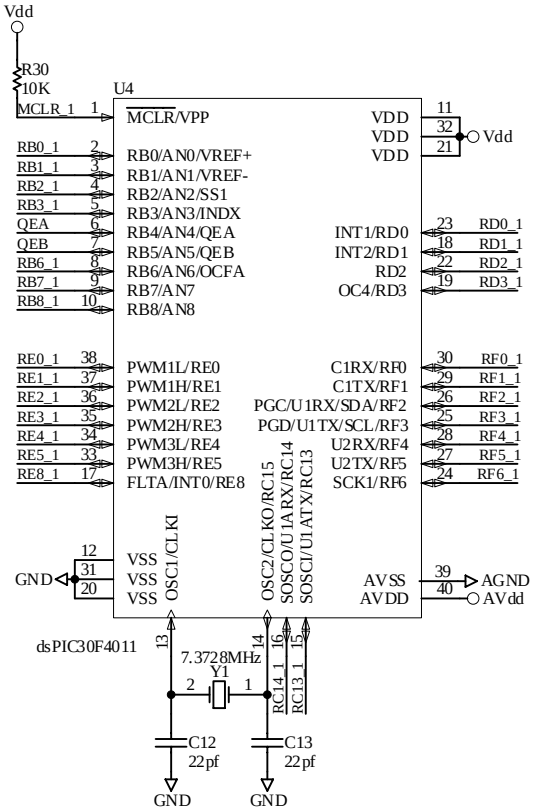
Matlab nos permite apresurar y simular los filtros rápidamente tecleando unas pocas líneas de código, esta ventaja debe ser aprovechada por los diseñadores, ya que en una aplicación concreta hay que ver las características de los filtros y seleccionar el filtro que más se acondicione a nuestras necesidades, pero además de eso será necesario simularlo para ver como se comporta cuando pasan los datos a través de este y si no estamos satisfechos con las pruebas realizadas será necesario probar con otro filtro rápidamente.

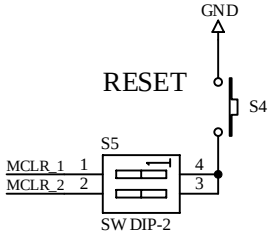
La utilización de la nueva tecnología de controladores digitales de Microchip es muy sencilla teniendo conocimiento previo de la familia de microcontroladores PICmicro, el diseño de aplicaciones en lenguaje C es muy parecida a hacer aplicaciones para la PC.



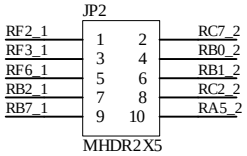
Esquemas electrónicos de la tarjeta de entrenamiento para DSP

dsPIC – 4ks

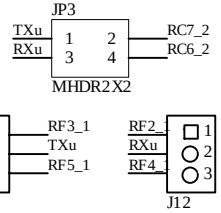




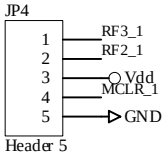
SPI COMM

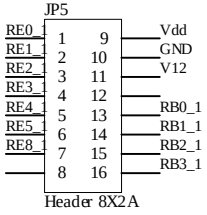


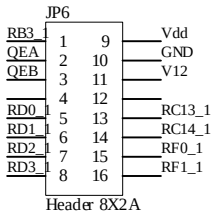
UART COMM



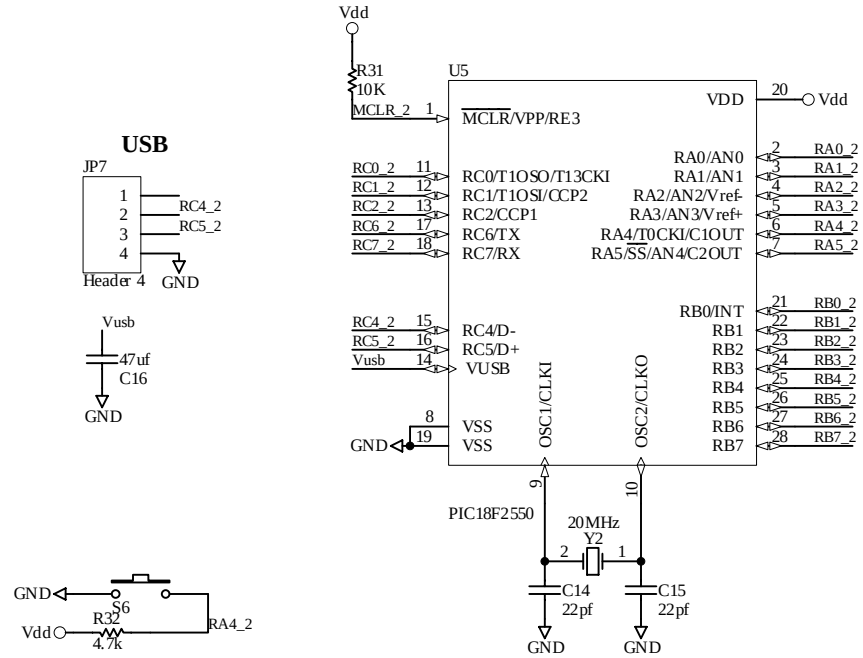
ICSP_1





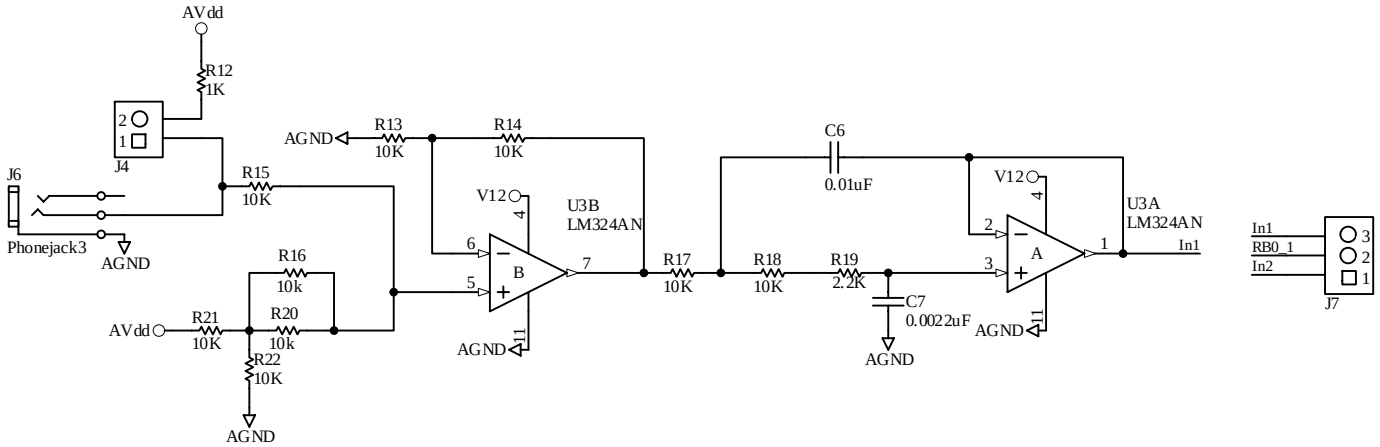


Titulo: ***DSP Core***			
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks		*Universidad del Azuay*
Fecha: 22/06/2006	Hora: 12:56:20	Hoja: de	*Facultad de Ciencia y Tecnologia*
Diseñador: *Jose Andres Cordero Garcia*			*Escuela Ingenieria Electronica*

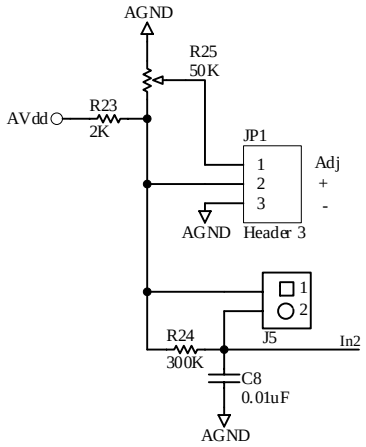


Titulo: ***Firmware USB***			
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks		*Universidad del Azuay*
			Facultad de Ciencia y Tecnologia
Fecha: 22/06/2006	Hora: 12:57:46	Hoja: de	*Escuela Ingenieria Electronica*
Diseñador: *Jose Andres Cordero Garcia*			

SOUND CARD INPUT

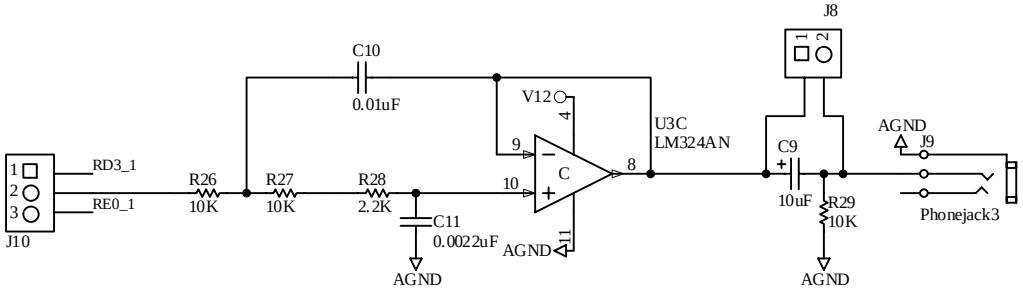


LM335 Temp sensor

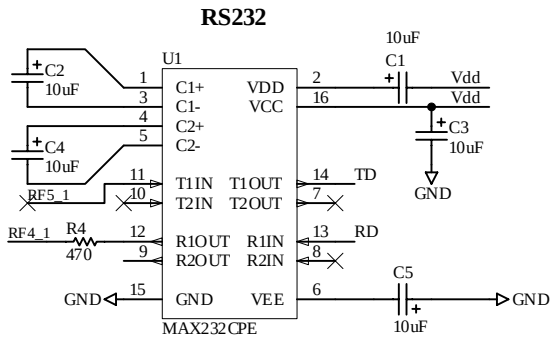


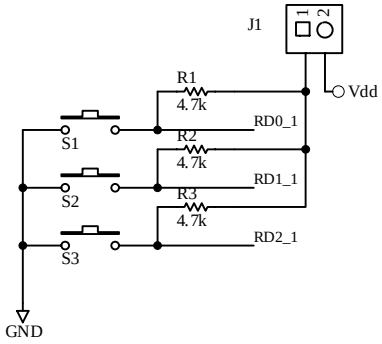
Titulo: ***Sound Card / Mic Input***		
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks	*Universidad del Azuay*
Fecha: 22/06/2006 Hora: 13:28:32	Hoja: de	*Facultad de Ciencia y Tecnologia*
Diseñador: *Jose Andres Cordero Garcia*		*Escuela Ingenieria Electronica*

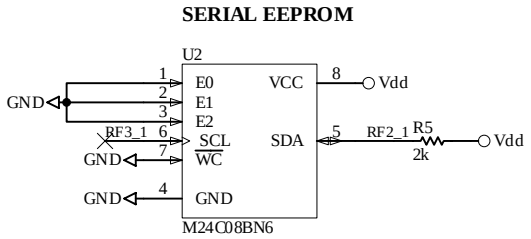
SPEAKER OUTPUT

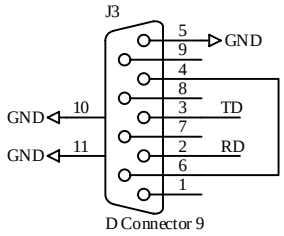


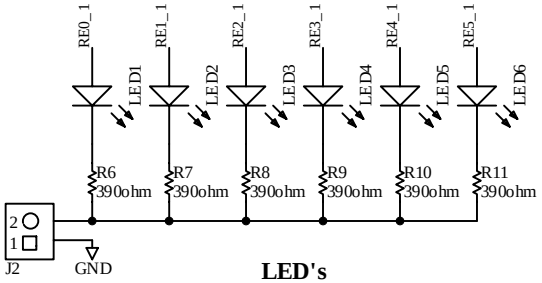
Titulo: ***Speaker OUT***		
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks	*Universidad del Azuay*
Fecha: 27/06/2006 Hora: 11:27:38 Hoja: de		*Facultad de Ciencia y Tecnologia*
Diseñador: *Jose Andres Cordero Garcia*		*Escuela Ingenieria Electronica*



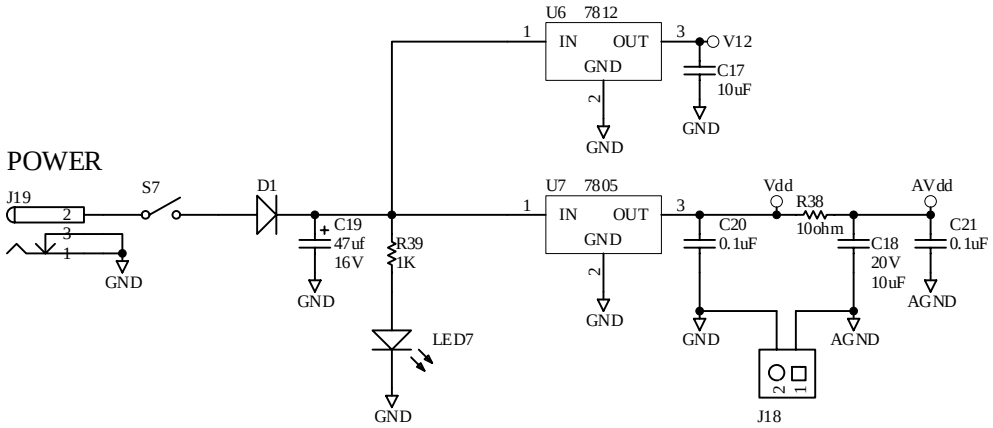








Titulo: ***Peripherals***			
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks		*Universidad del Azuay*
Fecha: 22/06/2006	Hora: 13:28:00	Hoja: de	*Facultad de Ciencia y Tecnologia*
Diseñador: *Jose Andres Cordero Garcia*			*Escuela Ingenieria Electronica*



Titulo: ***Power DC***			
Tamaño: A4	Proyecto: Tarjeta dsPIC-4ks		*Universidad del Azuay*
			Facultad de Ciencia y Tecnologia
Fecha: 22/06/2006	Hora: 14:18:35	Hoja: de	*Escuela Ingenieria Electronica*
Diseñador: *Jose Andres Cordero Garcia*			

Bibliografía

LYONS, Richard G. "*Understanding Digital Signal Processing*". Prentice Hall, 2001. 517p.

HAYES, Monson H. "*Schaum's Outlines of Digital Signal Processing*". McGraw Hill, 1999. 436p.

HAYKIN y Veen, Nan. "*Señales y Sistemas*"

HUNT, Brian R., LIPSMAN, Ronald L. "*A Guide to MATLAB for Beginners and Experienced Users*". Cambridge University Press, 2001. 346p.

MADISSETI, Vijay K. y WILLIAMS, Douglas B. "*Digital Signal Processing Handbook*". Chapman & Hall/CRCnetBase. 1999. 1689p.

OPPENHEIM, Alan V. y Schafer, Ronald W. "*Discrete-Time Signal Processing*". Prentice Hall, S.A. 796p.

OPPENHEIM, Alan V., WILLSKY, Alan S., NAWAB , S. Hamid Nawab. "*Señales y Sistemas*". 2ª ed. Prentice Hall, 1997. 956p.

PROAKIS, John G. y Manolakis, Dimitris, G. "*Digital Signal Processing – Principles, Algorithms and Applications*". 3ª ed. Prentice Hall, 1996. 1033p.

SMITH, Steven W. "*The Scientist and Engineer's Guide to Digital Signal Processing*". 2ª ed. California Technical Publishing, 1999. 650p.

THEDE, Les. "*Practical Analog and Digital Filter Design*". Artech House, 2004. 267p.

WINDER, Steve. "*Analog and Digital Filter Design*". Newnes, 2002. 457p.

Sitios de interés

www.dspguide.com
www.bores.com
www.dspguru.com
dspvillage.ti.com
www.microchip.com
www.dsptutor.freeuk.com
www.dsp.rice.edu
www.dsprelated.com
www-ece.rice.edu/~jdw/241/241.html
www.tecnum.com/asignaturas/tratamiento%20digital/TEMA8
www.faqs.org/docs/sp
www.winfilter.20m.com
www.digitalfilterdesign.com
en.wikipedia.org/wiki/Filter_design
ocw.mit.edu/index.html
www.motorola-dsp.com
www.k9spud.com/traxmod
www.filter-solutions.com