UNIVERSIDAD AUSTRAL DE CHILE
Facultad de Ciencias de la Ingeniería
Escuela de Electricidad y Electrónica

Entorno integrado para la docencia en tratamiento digital de señales

Trabajo de Titulación para optar al
Título de Ingeniero Electrónico.

Profesor Patrocinante:
Sr. Néstor Fierro Morineaud.

Jaime Andrés Solar Bravo

Valdivia Chile 2004

ÍNDICE

Contenido **Página**

A mi madre que en todos estos años encontré su apoyo y comprensión con el cual pude terminar esta etapa tan importante de mi vida
Gracias.....

Jaime

Sr. Néstor Fierro Morineaud

Sr. Pedro Rey Clericus

Sr. Alejandro Villegas Macaya

Valdivia, Marzo 2004

RESUMEN

Hoy en día dentro del mundo de las telecomunicaciones se trata el tema del tratamiento digital de señales, ya sea en docencia como en sus aspectos prácticos que involucra.

El presente trabajo de titulación pretende desarrollar este tema, partiendo de los conceptos básicos, hasta poder implementar prácticas relacionadas con el tratamiento digital de señales, utilizando ya sea software (Matlab) y hardware (DSP) asociados.

En esta tesis se dan ha conocer los fundamentos más relevantes que están relacionados con el tratamiento digital de señales como lo son la definición de señales y sistemas, también se trata un tema muy importante como lo es el diseño de filtros digitales los cuales tienen un aspecto primordial a la hora de tocar este tema, referente a su consecución ya sea en software, como en hardware.

La importancia de los temas relacionados con el tratamiento digital de señales cada día adquiere una mayor importancia, debido a los constantes avances tecnológicos que nos permiten desarrollar nuevas experiencias para la mejor comprensión del tema a tratar en este trabajo.

SUMMARY

Today in day inside the world of the telecommunications it is the topic of the digital treatment of signs, either in academy like in their practical aspects that it involves.

The present titulación work seeks to develop this topic, leaving of the basic concepts, until being able to implement practical related with the digital treatment of signs, either using software (Matlab) and hardware (DSP) associates.

In this thesis they are given the is necessary to know the most excellent foundations that are related with the digital treatment of signs as they are it the definition of signs and systems, it is also a very important topic as it is it the design of digital filters which have a primordial aspect when playing this topic, with respect to their attainment either in software, like in hardware.

The importance of the topics related with the digital treatment of signs every day the acquires a bigger importance, due to the constants technological advances that allow us to develop new experiences for the best understanding in the topic to try in this work.

Objetivos.

Objetivos Generales:

- Entregar a los estudiantes un apoyo integral para el desarrollo profesional en el área de las telecomunicaciones, específicamente, para el tratamiento digital de señales.

- Desarrollar una plataforma de aprendizaje aplicando herramientas multimediales sobre la World Wibe Web, junto con un conjunto de aplicaciones Internet de apoyo para asignaturas relacionadas con el tratamiento de señales.

- Tratar de complementar el aprendizaje en ramos de tratamiento digital de señales, realizando experiencias de laboratorio los cuales, en algunos casos, nos permiten analizar y comprender de mejor manera, los fenómenos que se producen.

Objetivos Específicos:

- Familiarizar al usuario con los conceptos que están relacionados con el tratamiento digital de señales, como lo son los diferentes tipos de señales y sistemas que existen.

- Desarrollar los temas que nos ayudan al estudio tanto de los sistemas como de las señales, como lo son las transformadas de Fourier, Laplace y Z.

- Mostrar los pasos para la implementación de un filtro digital mediante Matlab, así como señalar las diferentes etapas para cargar los programas del filtro diseñado a través de Code Composer y luego llevarlo al DSP.

- Dar las herramientas necesarias para que el usuario sea capaz de modelar y diseñar un filtro de acuerdo a sus necesidades.

- Diseñar una página Web, para poder tener un acceso más libre a los diferentes tópicos que son desarrollados en este trabajo de titulación.

Introducción General

Día a día se va haciendo más necesario el tratar el tema de tratamiento digital de señales con todos sus temas involucrados.

Esta tesis parte con la introducción de los conceptos de tratamiento de señales, como lo son la de poder definir y entregar las principales características , tanto de los sistemas como de las señales, continuando con este desarrollo de este tema, se involucran las herramientas de análisis que son necesarias para estudiar las diversas señales que se puedan tener, como lo son las transformadas de Fourier, Laplace y Z, estas nos permiten comprender mediante un análisis minucioso todo lo referente a los sistemas de procesado de señales.

Posteriormente se desarrollara un capítulo relacionado con dos conceptos que son utilizados al momento de realizar el tratamiento sobre alguna señal, ya sea para muestrearla o multiplexarla.

Luego se expone un capítulo de diseño de filtros digitales, estos filtros han ido revolucionando el mundo del tratamiento digital de señales, por el hecho de poder lograr mediante un diseño adecuado un excelente rendimiento al momento de modelar y limitar una determinada señal, por tal motivo se deben tener presente todos lo tópicos relacionados con estos, como lo son las funciones de transferencia que los rigen y también mencionar los principales aspectos que son necesarios tener en cuenta a la hora de ponerse a diseñar algún tipo de filtro digital.

Finalmente en el último capítulo se expondrá como se pueden realizar experiencias de laboratorio utilizando herramientas de software como lo es el Matlab y Simulink, así como también utilizar el hardware disponible en nuestra universidad como lo es los DSP.

Cabe señalar que para poder dar un mayor acceso a los alumnos interesados en este tema del tratamiento digital de señales se procedió al diseño de una página Web, esta fue diseñada en Dreamweaver, con el único propósito de entregar los diversos aspectos que fueron involucrados en el desarrollo de este trabajo de titulación.

Señales y Sistemas

1. Introducción.

Los conceptos de señales y sistemas aparecen en una variedad muy amplia de campos, las ideas y técnicas asociadas con estos conceptos juegan un papel muy importante en áreas tan diversas de la ciencia y la tecnología como comunicaciones, diseño de circuitos, acústica, sistemas de generación y distribución de energía, procesamiento de voz, etc. Si bien la naturaleza física de las señales y sistemas que son relacionadas en estas disciplinas son tan diversas pueden ser bastantes diferentes, todos ellos tienen en común dos características básicas. Mientras que las señales son funciones de una o más variables independientes y contienen información acerca de la naturaleza o comportamiento de algún fenómeno, los sistemas responden a señales particulares produciendo otras señales. Los voltajes y corrientes como funciones del tiempo en un circuito eléctrico son ejemplos de señales, y el circuito es, en sí, un ejemplo de sistema, el cual, en este caso, responde a los voltajes y corrientes que se le aplican.

Para poder llevar a cabo las diferentes técnicas de análisis de señales y sistemas es necesario entregar un marco de referencia analítico que tenga por finalidad dar a conocer las ideas intuitivas de señales y sistemas.

En este primer capítulo, tiene como objetivo principal desarrollar estos dos conceptos, entregando sus definiciones y las principales propiedades que poseen tanto las señales como los sistemas.

SEÑALES Y SISTEMAS

1.1.-Señales.

Las señales pueden describir una variedad muy amplia de fenómenos físicos. Aunque las señales se pueden representar de muchas maneras, en todos los casos la información dentro de una señal está contenida en un patrón de variaciones de alguna forma.

Definimos una señal como una función matemática que depende de una o más variables independientes, y cuyo valor nos da información sobre el fenómeno físico al que está asociada.

Un ejemplo de representación de una señal es la de la voz la cual se representa de forma matemática por la presión acústica como una función del tiempo, y una imagen se representa como una función de brillantez con respecto a dos variables espaciales.

1.1.1.- Clasificación de las señales:

Podemos dividir las señales según varios criterios. Los más usuales son:

1. Por el número de variables independientes:

   Unidimensional $y(t) = 3t - 5$

   Multidimensional $f(x, y) = x^2 + 3xy + 5$

2. Por la variable independiente: Según si los valores que toma la variable pertenecen a un conjunto continuo(Variable continua), o si pertenecen a un conjunto finito(Variable discreta). Las señales se escribirán de la forma $x(t)$ y $x[n]$ respectivamente. Un ejemplo de ambas sería:

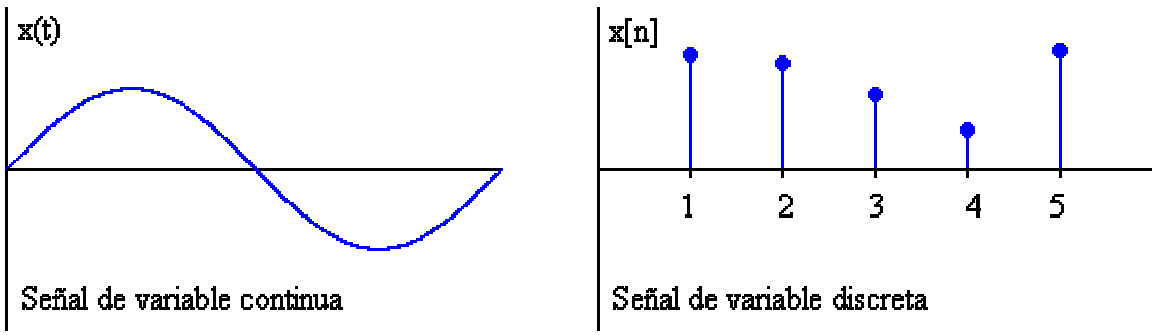


Figura 1.1

En aquellos puntos en los que la señal de variable discreta no tenga valores, no se considera que la señal sea nula, sino que no está definida. A este tipo se señales las llamaremos secuencias.
No se debe confundir la señal de variable continua con una señal continua. Por ejemplo la siguiente señal no es continua, pero si es de variable continua:

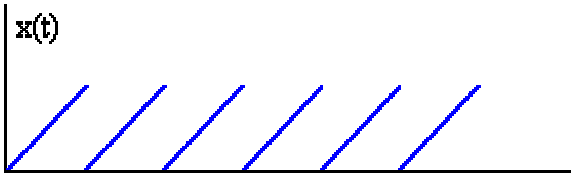


Figura 1.2

3. Por el rango de valores: La señal puede dividirse, al igual que la variable, en continua y discreta. Por ello existen señales continuas de variable continua(Sinusoide), continuas de variable discreta(Temperatura diaria a lo largo de un mes), discretas de variable continua (Parte entera) y discreta de variable discreta(Señal digital). Se puede pasar de una a otra mediante el proceso de muestreo:

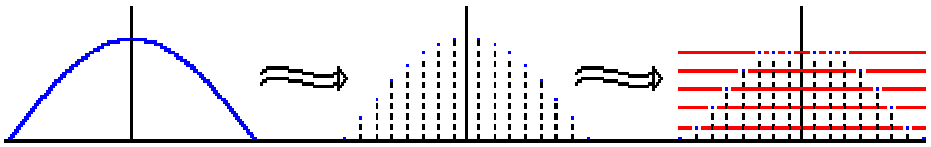


Figura 1.3

4. Por la determinación de la señal: Serán determinísticas aquellas señales cuyo valor para cada valor de la variable sea fijo(función matemática), y serán aleatorias aquellas cuyo valor es indeterminado(Temperatura en una fecha futura).

1.1.2.- Energía y potencia de una señal:

Sabemos que en una resistencia la potencia viene dada por:

$$p(t) = v(t)i(t) = \frac{v^2(t)}{r}$$

la energía por:

$$e(t) = \int_{t_1}^{t_2} p(t)dt = \int_{t_1}^{t_2} \frac{v^2(t)}{r} dt$$

y la potencia media por:

$$p_m = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2} p(t)dt$$

Definiremos entonces la energía y la potencia media de una señal como ($0 \le t_1 \le t_2$):

$$E = \int_{t_1}^{t_2} |x(t)|^2 \, dt$$

$$P_m = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2} |x(t)|^2 \, dt$$

y la energía y potencia totales de una señal como:

$$E_T = \lim_{t\to\infty}\int_{-t}^{t} |x(t)|^2 \, dt = \int_{-\infty}^{\infty} |x(t)|^2 \, dt$$

$$P_T = \lim_{t\to\infty}\left(\frac{1}{2t}\int_{-t}^{t} |x(t)|^2 \, dt\right)$$

Para el caso de una señal de variable discreta los definiremos como ($n_1 \le n \le n_2$):

$$E = \sum_{n_1}^{n_2} |x[n]|^2$$

$$P_m = \frac{1}{n_2 - n_1 + 1}\sum_{n_1}^{n_2} |x[n]|^2$$

y las totales como:

$$E_T = \lim_{n\to\infty}\sum_{-n}^{n} |x[n]|^2 = \sum_{-\infty}^{\infty} |x[n]|^2$$

$$P_m = \lim_{n\to\infty}\left(\frac{1}{2n+1}\sum_{-n}^{-n} |x[n]|^2\right)$$

Diremos que una señal es definida, bien en potencia, bien en energía, o bien en ambas, cuando la potencia, la energía, o ambas respectivamente no sean nulas ni infinitas. Así, pues, la señal formada por $x(t) = t$ no está definida ni en potencia ni en energía.

1.1.3.- Propiedades de las señales:

1. Simetría: Una señal es:

par si:	$x(t)=x(-t)$	$x[n]=x[-n]$
impar si:	$x(t)=-x(-t)$	$x[n]=-x[-n]$

Por ejemplo:

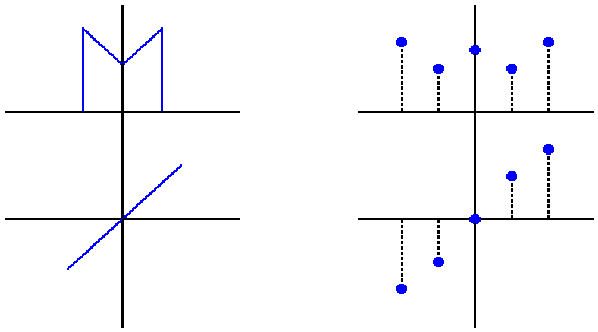


Figura 1.4

Toda señal impar, sea de variable discreta o continua, ha de valer cero en el origen, excepto si es discontinua en él.

Evidentemente no todas las señales son pares o impares, pero siempre vamos a poder descomponerlas en suma de una señal par y otra señal impar:

Sea $x(t)$ una señal. Entonces la podemos descomponer como $x(t)=x_p(t)+x_i(t)$, siendo:

$$x_p(t)=\tfrac{1}{2}[x(t)+x(-t)]$$
$$x_i(t)=\tfrac{1}{2}[x(t)-x(-t)]$$

e igualmente en señales de variable discreta.

2. Periodicidad: Una señal $x(t)$ es periódica si existe $T>0$ tal que $x(t)=x(t+T)$, donde T es el mínimo valor tal que se cumple la condición dada. Análogamente se dice que una señal $x[n]$ es periódica si existe $N>0$ tal que $x[n]=x[n+N]$, donde N es el mínimo valor tal que se cumple la condición dada.
3. Causalidad: Una señal $x(t)$ es causal si $x(t)=0\ \forall t<0$. Igualmente en variable discreta.
4. Ortogonalidad: Dos señales $x(t)$ e $y(t)$ se dice que son ortogonales en un intervalo $[t_1,t_2]$ si:

$$\int_{t_1}^{t_2} x(t)y(t)dt=0$$

Análogamente, y para dos señales $x[n]$ e $y[n]$ se dice que son ortogonales en un intervalo $\{n_1.\mathrm{K}\ ,n_2\}$ si:

$$\sum_{n_1}^{n_2} x[n]y[n] = 0$$

1.1.4.- Transformaciones en la variable independiente:

1. Desplazamiento en la variable: Consiste en restar a la variable una constante:

$$x(t-t_0)$$
$$x[n-n_0]$$

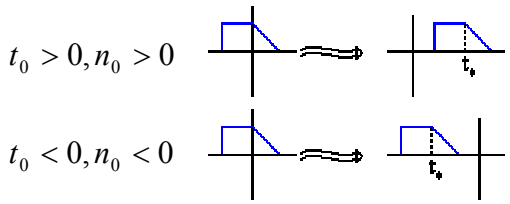


Figura 1.5

2. Reflexión: Consiste en invertir la señal respecto del origen de la variable:

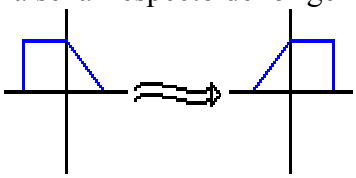


Figura 1.6

3. Escalado: Aquí hay que diferenciar el escalado en tiempo continuo y el escalado en tiempo discreto. Vamos a estudiar el escalado en tiempo continuo:

Consiste en multiplicar la variable por una constante:

$x(at)$

Aquí debemos tener en cuenta que la constante puede ser mayor o menor que la unidad:

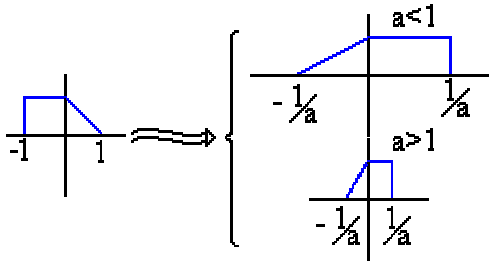


Figura 1.7

Es decir:
$a > 1$ disminuye la amplitud
$a < 1$ aumenta la amplitud

En tiempo discreto la operación es ligeramente distinta, pues consiste en cambiar la señal $y[n]$ de tal manera que(Suponiendo que $k > 1$):

$$y[kn] = \begin{cases} y[n/k] & n = \overset{\circ}{k} \\ 0 & n \neq \overset{\circ}{k} \end{cases}$$

Por ejemplo, para una señal dada, y $k = 3$

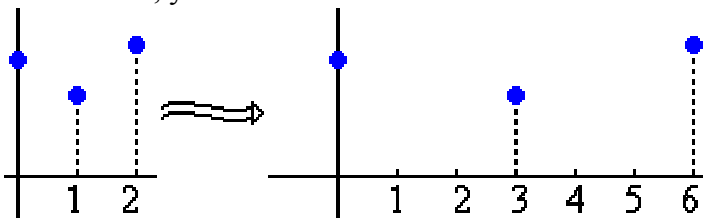
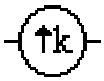
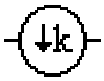


Figura 1.8

A esta operación de escalado aplicada a una señal de variable discreta se le llama interpolación, y consiste en intercalar $k-1$ ceros entre cada dos valores consecutivos. Se representa por el siguiente símbolo:



En el caso de que $k < 1$ la operación consiste en eliminar $k-1$ muestras entre cada k muestras separadas. A este proceso se le llama diezmado, y no tiene más aplicación práctica que recomponer una señal ya interpolada. Se representa por:



Evidentemente es posible conjugar las operaciones, pero siempre teniendo cuidado con el orden de actuación. Veamos un ejemplo de conjugación a partir de una señal dada:

$x(2t+1)$

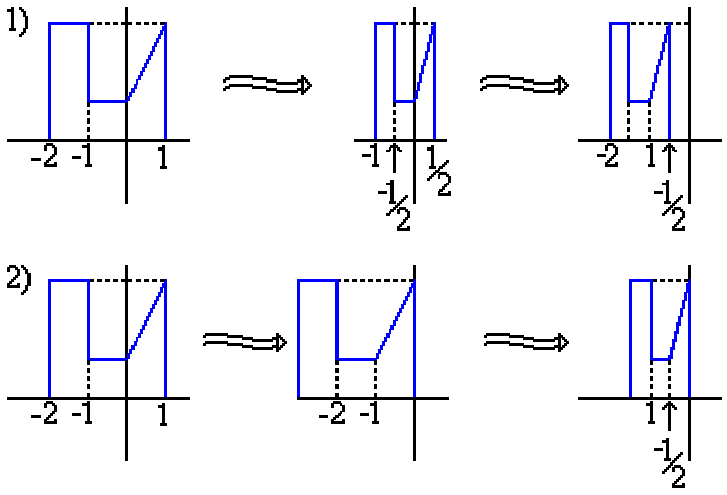


Figura 1.9

Evidentemente la primera conjugación está mal, ya que en realidad lo que está haciendo es $x(2(t+1))$, que es incorrecto. Lo correcto es, pues, desplazar primero y escalar después.

1.1.5.- Ejemplos de señales:

- **Tiempo continuo:**

1. Exponencial compleja:

$$x(t) = ce^{\alpha t}, \text{ donde } c, \alpha \in C$$

En el caso de que $c, \alpha \in R$

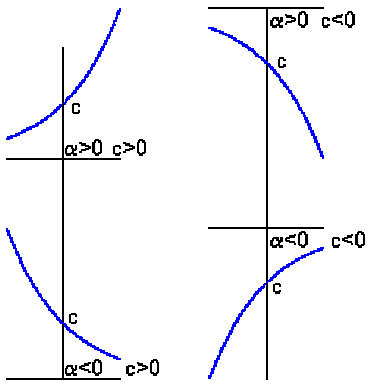
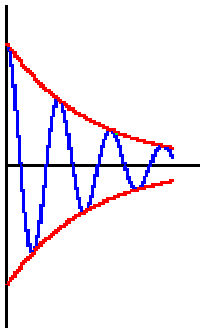


Figura 1.10

Si $\alpha = j\omega$ y $c \in R$

$$x(t) = ce^{j\omega t} = c(\cos(\omega t) + j\,\text{sen}(\omega t))$$

con lo que la función resultante es periódica



En el caso general:

$\alpha = \sigma + j\omega$

$c = c_0 e^{j\theta}$

$x(t) = c_0 e^{j\theta} e^{(\sigma + j\omega)t} = c_0 e^{\sigma t} e^{j(\theta + j\omega)} = c_0 e^{\sigma t}\left(\cos(\theta + \omega t) + j\,\mathrm{sen}(\theta + \omega t)\right)$

Podemos representar la parte real. La parte imaginaria es idéntica.

Figura 1.11

Relacionada con este tipo de señal están las exponenciales armónicamente relacionadas, que son todas aquellas exponenciales complejas de la forma:

$$\phi_k(t) = ce^{j(\omega_0 kt)} \quad k = \pm 1, \pm 2, \mathrm{K}$$

Evidentemente todas son periódicas de periodo T_0, y el periodo mínimo de cada una de ellas es $T_0 / |k|$

2. Impulso unidad o Delta de Dirac:

Para definir esta señal vamos a empezar por definir una señal $\delta_\Delta(t)$ como una señal cuadrada centrada en el origen, de anchura $2/\Delta$ y área unidad:

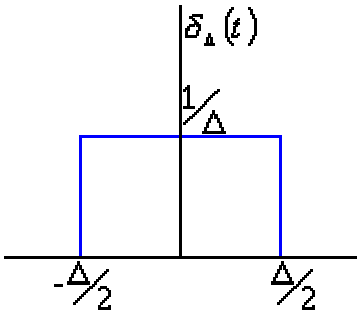


Figura 1.12

El límite de dicha función es la función delta de Dirac:

$$\lim_{\Delta \to 0} \delta_\Delta(t) = \delta(t)$$

Dicha función se representa por :

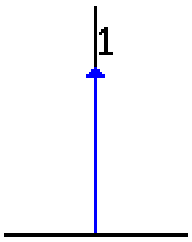


Figura 1.13

Donde el 1 indica que el área es la unidad, ya que la altura es infinita.

Podemos definirla como:

$$\delta(t)=\begin{cases}0 & t\neq 0\\ \infty & t=0\end{cases}$$

Algunas de sus propiedades son:

$\int_{-\infty}^{\infty}\delta(t)dt=1$

$$\int_{-\infty}^{t}\delta(t)dt=\begin{cases}0 & t<0\\ 1 & t>0\\ Indef. & t=0\end{cases}$$

$x(t)\cdot\delta(t)=x(0)\cdot\delta(t)$

$x(t)\cdot\delta(t-t_0)=x(t_0)\cdot\delta(t-t_0)$

$x(t-t_0)\cdot\delta(t)=x(-t_0)\cdot\delta(t)$

$x(t-t_0)\cdot\delta(t-t_0)=x(0)\cdot\delta(t-t_0)$

3. Escalón unidad:

Al igual que en la anterior, vamos a definir una función accesoria cuyo límite nos va a dar la función escalón unidad. Por tanto definimos la función $U_{\Delta}(t)$ como:

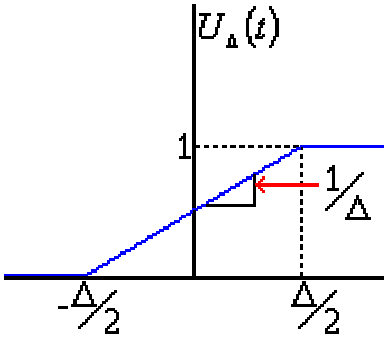


Figura 1.14

Su límite es la función escalón unidad:

$$\lim_{\Delta \to 0} U_{\Delta}(t) = U(t)$$

Sus propiedades son análogas a las de la delta de Dirac. Además, si la derivamos, obtenemos:

$$\frac{d(U_{\Delta})}{dt} = \left\{ \begin{array}{ll} 0 & t < -\Delta/2 \\ 0 & t > \Delta/2 \\ 1/\Delta & -\Delta/2 < t < \Delta/2 \end{array} \right\} = S_{\Delta}(t) \Rightarrow \frac{d(U)}{dt} = S(t)$$

Por tanto podemos definir:

$$U(t) = \int_{-\infty}^{t} S(\tau) \cdot d\tau$$

4. Pulso rectangular:

Esta señal se representa por:

$$x(t) = A\Pi\left(\frac{t - t_0}{\tau}\right)$$

Su representación gráfica es:

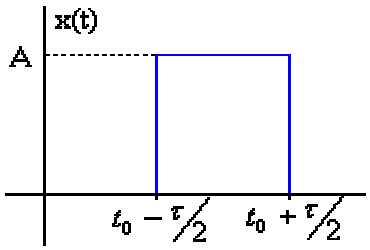


Figura 1.15

Es posible crear esta señal a partir de dos funciones escalones retardadas convenientemente:

$$x(t) = A\Pi\left(\frac{t - t_0}{\tau}\right) = A \cdot \left[U\left(t - t_0 + \tau/2\right) - U\left(t - t_0 - \tau/2\right)\right]$$

5. Señal triangular:

Esta señal se representa por:

$$x(t) = A\Delta\left(\frac{t - t_0}{2\tau}\right)$$

Su representación gráfica es:

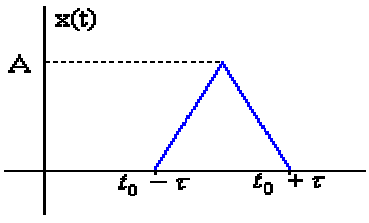


Figura 1.16

6. Función $sinc(t)$:
Se define como:

$$sinc(t) = \frac{\text{sen}(\pi t)}{\pi t}$$

Se puede verificar que es continua en $t = 0$. Asimismo $sinc(t)$ es nulo en todos los puntos en los que t sea entero, exceptuando a $t = 0$.

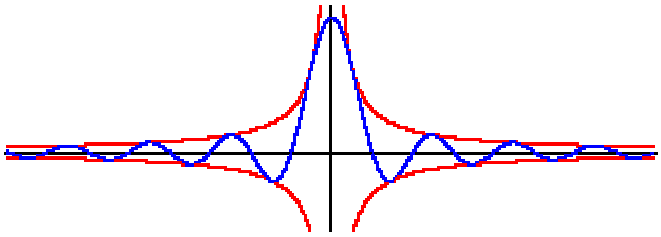


Figura 1.17

- **Tiempo discreto:**

1. Impulso delta de Kronecker:

Se define como:

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

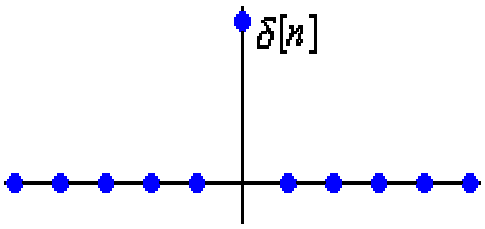


Figura 1.18

Algunas de sus propiedades más importantes son:

$$x[n]\delta[n] = x[0]\delta[n]$$

$$x[n]\delta[n - n_0] = x[n_0]\delta[n - n_0]$$

$$x[n - n_0]\delta[n] = x[-n_0]\delta[n]$$

$$x[n - n_0]\delta[n - n_0] = x[0]\delta[n - n_0]$$

$$\sum_{m=-\infty}^{n} \delta[m] = \begin{cases} 0 & m < 0 \\ 1 & m \geq 0 \end{cases}$$

2. Escalón unidad en tiempo discreto:

Se define como:

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

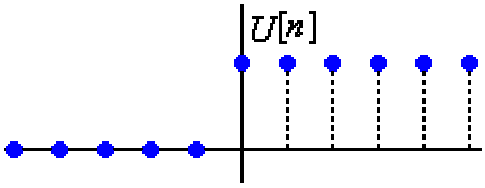


Figura 1.19

También se puede definir como:

$$U[n] = \sum_{m=-\infty}^{n} \delta[m]$$

3. Exponencial compleja en tiempo discreto:

Se define como:

$$x[n] = c\alpha^n \text{, donde } c, \alpha \in C$$

En el caso de que $c, \alpha \in R$:

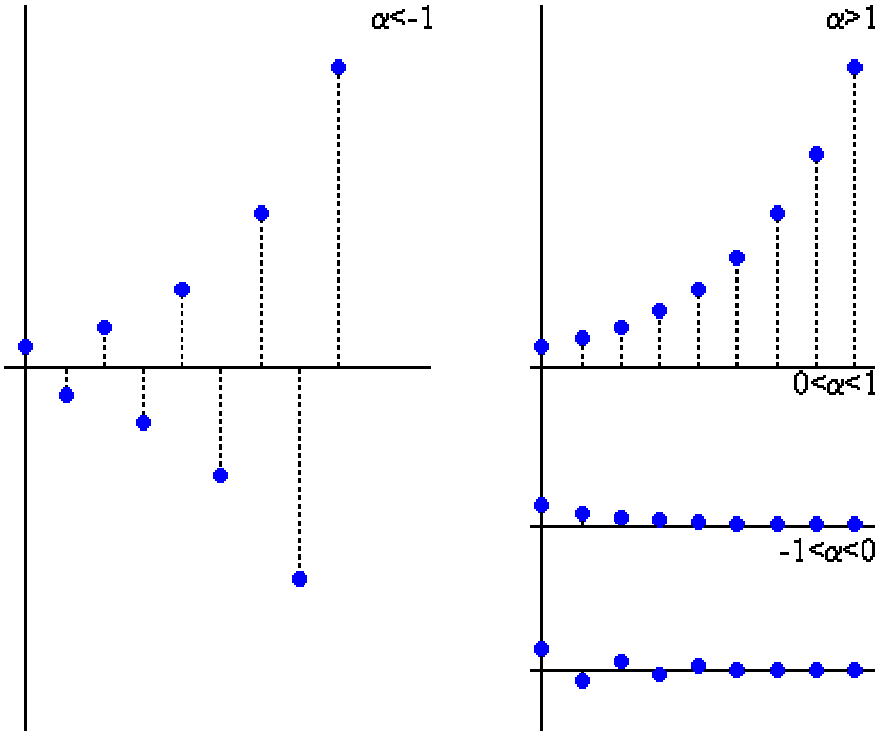


Figura 1.20

En el caso de que $c \in R$, $\alpha \in C$, escribimos:

$$x[n] = ce^{j\Omega_0 n}$$

Donde las unidades de Ω_0 son radianes, y le llamamos pulsación, o vulgarmente frecuencia. Por las formulas de Euler podemos escribir:

$$e^{j\Omega_0 n} = \cos(\Omega_0 n) + j \cdot \text{sen}(\Omega_0 n)$$

Dicha señal es periódica en la frecuencia, es decir, para valores de Ω_0 separados por un múltiplo entero de veces 2π, el comportamiento se repite, de tal manera la oscilación va creciendo con la frecuencia hasta llegar a π, y luego vuelve a disminuir hasta llegar a 2π, repitiéndose el proceso.

También puede ser periódica en la variable. Para ello se ha de cumplir que:

$$e^{j\Omega_0(n+N)} = e^{j\Omega_0 n} e^{j\Omega_0 N}$$

Si $\Omega_0 N = 2k\pi, \quad k \in Z$, entonces $e^{j\Omega_0 N} = 1$ y la señal es periódica.

Al igual que en variable continua, existen unas exponenciales armónicas relacionadas, de frecuencia múltiplo de la fundamental Ω_0, cuyo número no es infinito, ya que son periódicas en la frecuencia. Las representamos como:

$\Phi_k[n] = e^{jk\Omega_0 n}$

1.2.- Sistemas:

Un sistema se puede ver como cualquier tipo de proceso en el cual se produce una transformación de señales, entonces, un sistema tiene una señal de entrada y una señal de salida la cuál esta relacionada con la entrada a través de la transformación del sistema

Igualmente definimos un sistema como cualquier transformación realizada sobre una señal.

Figura 1.21

Un ejemplo serían las transformaciones que sobre la señal de un generador hace el circuito al que esta conectado.

Sistemas de tiempo continuo, es aquel en el que las señales de entrada de tiempo continuo son transformadas en señales de salida de tiempo continuo.

De forma similar, un sistema de tiempo discreto, esto es, uno que transforma entradas de tiempo discreto en salidas de tiempo discreto.

1.2.1.- Conexión de los sistemas:

1. En serie o cascada:

Consiste en conectar la señal de salida de un sistemas a la entrada de otro sistema:

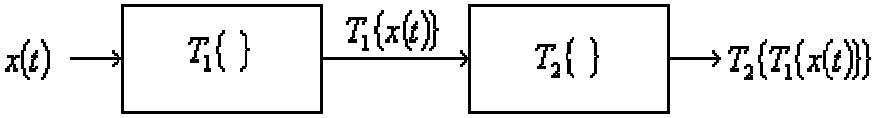


Figura 1.22

2. En paralelo:

Este tipo de conexión consiste en hacer dividir la señal y hacerla pasar por diferentes sistemas, para luego sumarlos:

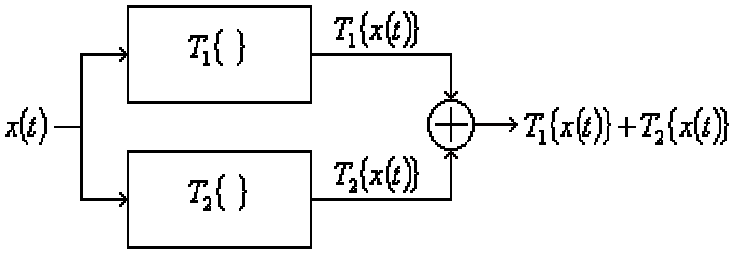


Figura 1.23

3. Con realimentación. Consiste en sumar parte de la señal de salida de un sistema a su propia entrada:

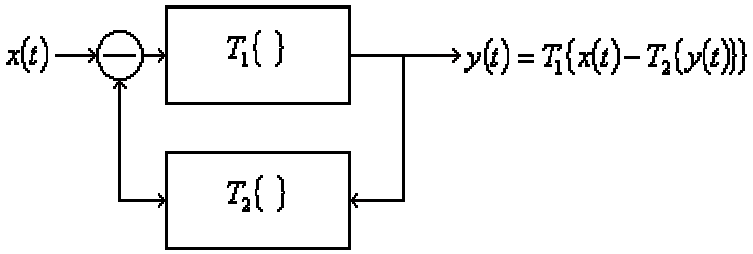


Figura 1.24

Por supuesto es posible usar combinaciones de los tres tipos de conexión. Un ejemplo sería:

$y[n] = x^2[n] - 3x[n] + 2x^2[n-1]$

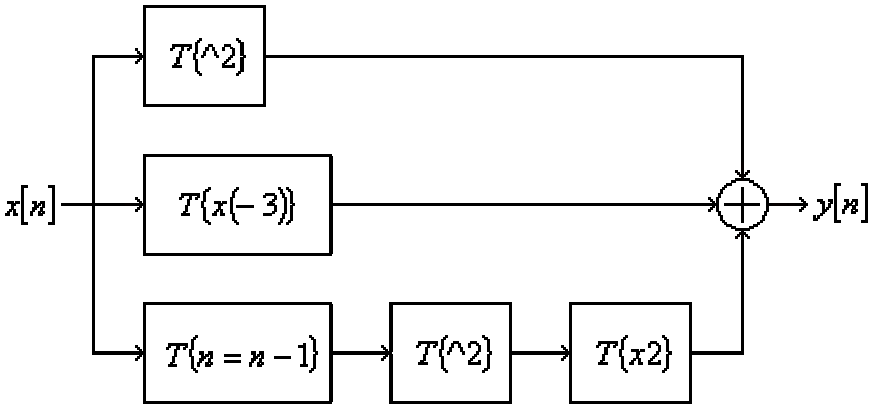


Figura 1.25

1.2.2.- Propiedades de los sistemas:

1. Linealidad:

Se dice que un sistema es lineal si cumple los principios de superposición y multiplicación por una constante. La manera de expresarlo es:

Un sistema $T\{f(x)\}$ es lineal si $T\{ax_1(t) + bx_2(t)\} = aT\{x_1(t)\} + bT\{x_2(t)\}$, para cualesquiera señales $x_1(t)$, $x_2(t)$ y para cualesquiera constantes a, b.

Por ejemplo:

$T\{\text{sen}(x)\}$	No es lineal
$T\{\text{Re}[x]\}$	No es lineal
$T\{x^2\}$	No es lineal
$T\{3x\}$	Si es lineal

Existe un tipo de sistemas, no lineales, consistentes en aplicar una transformación lineal y sumar una constante al resultado. Por ejemplo:

$$T\{3x+1\}$$

Sin embargo, estos sistemas si son lineales con respecto a los incrementos de la señal, y por ello se les llama sistemas incrementalmente lineales.

Por último hacer notar que todo sistema lineal verifica que ante una señal de entrada nula, la señal de salida es nula.

2. Invarianza:

Un sistema se dice invariante si un desplazamiento en la señal o secuencia de entrada produce el mismo desplazamiento en la señal o secuencia de entrada. Podemos decir que:

Un sistema $y(t)=T\{f(x)\}$ es invariante si $y(t-t_0)=T\{x(t-t_0)\}$, para cualquier señal $x(t)$.

Por ejemplo:

$y[n]=T\{x[n]\}=n\cdot x[n]$

$T\{x[n-n_0]\}=n\cdot x[n-n_0]$

$y[n-n_0]=(n-n_0)\cdot x[n-n_0]$

Son distintas, luego el sistema es variante.

$y(t)=T\{x(t)\}=\text{sen}(x(t))$

$T\{x(t-t_0)\}=\text{sen}(x(t-t_0))$

$y(t-t_0)=\text{sen}(x(t-t_0))$

Luego el sistema es invariante.

3. Causalidad:

Un sistema es causal si en un instante dado la señal de salida depende solamente del valor de la señal de entrada en dicho instante, o en instantes anteriores, nunca posteriores.

Por ejemplo:

$y(t) = T\{x(t)\} = 3x(t) - x^2(t)$ Es causal

$y[n] = T\{x[n]\} = 3x[n-1] + x[n] + x[n+2]$ No es causal

Los sistemas no causales con variable temporal no son realizables físicamente

4. Estabilidad:

Un sistema se dice estable si ante una señal de entrada acotada da una señal de salida acotada. En ingles se dice que el sistema es BIBO. Se puede expresar como:

Un sistema $y(t) = T\{f(x)\}$ es estable si $\forall |x(t)| \le k_1 \rightarrow |T\{x(t)\}| \le k_2$, para cualquier señal $x(t)$.

Por ejemplo:

$y[n] = T\{x[n]\} = x^2[n]$ Es estable

$y(t) = T\{x(t)\} = e^{at}x(t)$ No es estable

$y(t) = T\{x(t)\} = \text{sen}(x(t))x(t)$ Es estable

$y[n] = T\{x[n]\} = \sum_{k=-\infty}^{n} x[k]$ No es estable

5. Invertibilidad:

Un sistema es invertible si podemos encontrar un sistema que al introducir la señal de salida del original nos devuelva la señal de entrada. Es decir:

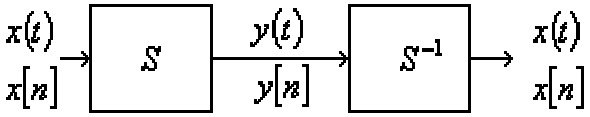


Figura 1.26

Por ejemplo:

$y(t) = T\{x(t)\} = 3x(t)$ $x(t) = T^{-1}\{y(t)\} = \frac{1}{3}y(t)$

$y(t) = T\{x(t)\} = x^2(t)$ $\nexists\, T^{-1}$

$y[n] = T\{x[n]\} = \sum_{k=-\infty}^{n} x[k]$ $x[n] = T^{-1}\{y[n]\} = y[n] - y[n-1]$

6. Memoria:

Se dice que un sistema no tiene memoria si la señal de salida depende únicamente del valor actual de la señal de entrada.

Por ejemplo:

$y(t) = T\{x(t)\} = t \cdot x(t)$ Sin memoria

$y(t) = T\{x(t)\} = x(t) + x(t-3)$ Con memoria

Evidentemente todo sistema sin memoria es causal.

1.3.- Resumen.

En este capítulo se dieron a conocer las herramientas básicas del procesado digital de señales definiendo básicamente lo que son las señales de tiempo continuo, como también las señales discretas, que ya han sido procesadas.

Aquí también se entregaron las principales características que poseen las señales de tiempo continuo y discreto, como son la linealidad, invarianza etc.

Por último se definió lo que es un sistema dando a conocer ya sea las conexiones que se pueden desarrollar con estos como así también las características que los rigen.

Transformadas de Fourier Laplace y Z

2. Introducción.

Para poder analizar y comprender el amplio tema del procesamiento digital de señales, es necesario tener conocimientos sobre ciertas herramientas que son utilizadas para el desarrollo de este tema, es por esta razón que este capítulo esta enfocado en dar a conocer algunas de estas herramientas matemáticas como lo son las denominadas: Transformadas de Fourier, Laplace y la transformada Z.

Estás técnicas de análisis que serán descritas en el siguiente capítulo son de gran valor para analizar y conocer las propiedades de las señales ya sea de tiempo continuo como en tiempo discreto.

2.1.- Transformada de Fourier.

Como bien se mencionaba en la introducción a este capítulo, aquí se tratará de entregar las herramientas que son necesarias para poder desarrollar el tema del tratamiento digital de señales, para comenzar con este estudio, se definirán las series y transformadas de Fourier en tiempo continuo, para seguir posteriormente con la transformada de Fourier en tiempo discreto.

La serie de Fourier puede usarse algunas veces para representar una función dentro de un intervalo. Si una función esta definida sobre toda la recta real, puede ser representada con una serie Fourier si es periódica. Si no es periódica, entonces no puede representarse con una serie Fourier para todo *x*. Aun en este caso es posible representar la función en términos de senos y cosenos, pero la serie de Fourier se convierte en una integral de Fourier. La motivación proviene de considerar formalmente las series de Fourier como funciones con período 2*T* y hacer tender *T* al infinito.

Suponiendo:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\frac{\pi}{T}x}$$

Y

$$c_n = \frac{1}{2T}\int_{-T}^{T} e^{-in\frac{\pi}{T}t} f(t)dt$$

Tomando

$$\omega_n = \frac{n\pi}{T} \quad \text{and} \quad \Delta\omega = \omega_n - \omega_{n-1} = \frac{\pi}{T}$$

y reemplazando la fórmula de la integral por los coeficientes de Fourier:

$T \to \infty$

$$\frac{1}{2\pi}\sum_{n=-\infty}^{\infty}\left(e^{i\omega_n x}\int_{-T}^{T} e^{-i\omega_n t} f(t)dt\right)\Delta\omega$$

La sumatoria se asemeja a una suma de Riemann de una integral definida, y en el límite $T \to \infty$ ($\Delta\omega \to 0$) tendríamos:

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\left(e^{i\omega x}\int_{-T}^{T} e^{-i\omega t} f(t)dt\right)d\omega \qquad x \in \mathbb{R}$$

Una función F(ω) se denomina la **transformada de Fourier** de *f*(*x*), si: Existe.

$$F(\omega) := \mathcal{F}\{f(t)\} := \int_{-\infty}^{\infty} e^{-i\omega t} f(t) dt$$

$$\mathcal{F}^{-1}\{F(\omega)\} := \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega x} F(\omega) d\omega$$

que esta definida en $\Re$ y toma valores complejos. Para que la transformada de Fourier de una señal x(t) exista (en forma ordinaria no como función generalizada), x debe satisfacer las siguientes propiedades denominadas condiciones de Dirichlet:

(1) x(t) es absolutamente integrable, esto es:

$$\int_{-\infty}^{\infty} |x(t)|\, dt < \infty.$$

(2) x(t) posee un número finito de discontinuidades en cualquier intervalo finito.

2.1.1.- Transformada Inversa de Fourier Sea x(t) una señal cuya transformada de Fourier es X(ω). La transformada inversa de Fourier es el proceso de obtener x(t) a través de X(ω) y se define como:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{j\omega t}\, d\omega.$$

La transformada inversa de Fourier se traduce a integrar la Función que $X(\omega)e^{j\omega t}$ esta definida de los reales a los complejos.

2.1.2.- Algunos pares de transformadas de Fourier

Señal	Transformada de Fourier
$\sum_{k=-\infty}^{+\infty} a_k e^{jk\omega_0 t}$	$2\pi \sum_{k=-\infty}^{+\infty} a_k \delta(\omega - k\omega_0)$
$e^{j\omega_0 t}$	$2\pi\delta(\omega - \omega_0)$
$\cos \omega_0 t$	$\pi[\delta(\omega - \omega_0) + \delta(\omega + \omega_0)]$
$\operatorname{sen} \omega_0 t$	$\frac{\pi}{j}[\delta(\omega - \omega_0) - \delta(\omega + \omega_0)]$
$x(t) = 1$	$2\pi\delta(\omega)$
$\frac{\operatorname{sen} Wt}{\pi t}$	$X(\omega) = \begin{cases} 1 & \lvert\omega\rvert < W \\ 2 & \lvert\omega\rvert > W \end{cases}$
$\delta(t)$	1
$u(t)$	$\frac{1}{j\omega} + \pi\delta(\omega)$
$\delta(t - t_0)$	$e^{-j\omega t_0}$
$e^{-at}u(t)$, $\operatorname{Re}\{a\} > 0$	$\frac{1}{a + j\omega}$
$te^{-at}u(t)$, $\operatorname{Re}\{a\} > 0$	$\frac{1}{(a + j\omega)^2}$
$\frac{t^{n-1}}{(n-1)!}e^{-at}u(t)$, $\operatorname{Re}\{a\} > 0$	$\frac{1}{(a + j\omega)^n}$

Tabla 2.1

2.1.3.- Algunas propiedades de la Transformada de Fourier.

Propiedad	Señal	Transformada de Fourier
	$x(t)$	$X(\omega)$
	$y(t)$	$Y(\omega)$
Linealidad	$ax(t)+by(t)$	$aX(\omega)+bY(\omega)$
Desplazamiento en tiempo	$x(t-t_0)$	$e^{-j\omega t_0}X(\omega)$
Desplazamiento en frecuencia	$e^{j\omega_0 t}x(t)$	$X(\omega-\omega_0)$
Escalamiento de tiempo y de frecuencia	$x(at)$	$\frac{1}{\lvert a\rvert}X\left(\frac{\omega}{a}\right)$
Inversión en el tiempo	$x(-t)$	$X(-\omega)$
Conjugación	$\overline{x(t)}$	$\overline{X(-\omega)}$
Convolución	$x(t)*y(t)$	$X(\omega)Y(\omega)$
Multiplicación	$x(t)y(t)$	$\frac{1}{2\pi}X(\omega)Y(\omega)$
Diferenciación en tiempo	$\frac{d}{dt}x(t)$	$j\omega X(\omega)$
Integración	$\int_{-\infty}^{t}x(t)\,dt$	$\frac{1}{j\omega}X(\omega)+\pi X(0)\delta(\omega)$
Diferenciación en frecuencia	$tx(t)$	$j\frac{d}{d\omega}X(\omega)$

Tabla 2.2

2.1.4.- La transformada de tiempo discreto de Fourier.

Para una señal muestreada la Transformada de Fourier **se define como**:

$$X(w) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-jwn}$$

donde ahora ω es una frecuencia normalizada a la de muestreo que varía entre $-\pi$ $+\pi$ (π corresponde a la mitad de la frecuencia de muestreo) Esta expresión recibe el nombre de **Transformada de Fourier en tiempo discreto (DTFT).**

La DTFT tiene el problema de no ser adecuada para el tratamiento por computador:

1. Porque es imposible, disponer de una señal de longitud finita.
2. Porque es poco operativo manejar una expresión como la ecuación anterior.

Es preferible a efectos de procesamiento digital, disponer de un conjunto finito de valores de la transformada. Por ello, se suele usar la Transformada Discreta de Fourier (DFT) definida como:

$$X(n) = \sum_{k=0}^{N-1} X(k) e^{-j2\pi kn/N} \quad n = 0,1,\ldots N-1$$

$$X(n) = \sum_{k=0}^{N-1} x(k)\, W^{nk} \qquad W = e^{-j2\pi/n}$$

aunque también se puede emplear, y es corrientemente utilizada, la siguiente nomenclatura x(n) muestras y x(k) coeficientes DFT

$$X(k) = \sum_{k=0}^{N-1} X(n) e^{-j2\pi kn/N} \quad k = 0,1,\ldots N-1$$

Esta ecuación representa un conjunto de N puntos de la TF en el intervalo -π +π En realidad cuando mediante Matlab calculamos la TF, se utiliza DFT con un elevado número de puntos para simular la continuidad.

Podemos interpretar los resultados de DFT de una secuencia xs(n) desde dos puntos de vista:

1.-Como los coeficientes espectrales (series de Fourier) de una señal periódica discreta cuyos muestreos coinciden con la secuencia xs(n)

2.-Como el espectro de una señal aperiódica discreta cuyos muestreos corresponden a la secuencia xs(n)

La Transformada Discreta de Fourier es una aproximación al espectro de la señal analógica original. Su magnitud se ve influenciada por intervalos de muestreo, mientras que su fase depende de los instantes de muestreo.

- **Transformada de Fourier en Tiempo Discreto de señales aperiódicas:**

La señal aperiódica, debe de ser muestreada durante un tiempo D. DFT produce los coeficientes espectrales correspondientes a la extensión periódica x(t) con periodo D. El espacio es fo=1/D. A fo se le denomina resolución espectral. Esta depende sólo de la duración. Si la señal está limitada en el tiempo, la forma de aumentar la duración es añadir ceros.

Nota: para el diseño es muy importante elegir adecuadamente los parámetros

Frecuencia de muestreo *fs=1/ts*

La frecuencia de muestreo se determina a partir del teorema de muestreo (si queremos detectar el espectro de una señal hasta una máxima frecuencia B, la frecuencia de muestreo deberá ser 2B)

Resolución de frecuencia *f0=1/D*

La duración del muestreo se elije para una determinada resolución en frecuencia.

Una regla de diseño puede ser: Si queremos los M primeros armónicos de una señal con un error máximo del 5%, el número de muestreos N=8M

2.2.- La Transformada de Laplace.

El análisis de Fourier es de gran importancia para el estudio de los problemas que involucran las señales como los sistemas lineales e invariantes en el tiempo. Esto se debe principalmente al hecho de que un gran número de señales se pueden representar mediante una combinación lineal de exponenciales complejas y esas exponenciales complejas son funciones características de los sistemas lineales invariantes en el tiempo.

En los siguientes párrafos se considerará esta generalización de la transformada de Fourier utilizando una clase amplia de señales exponenciales complejas. La transformada resultante es conocida como la transformada de Laplace.

Se define la Transformada de Laplace de la señal *x(t)* como:

$$\mathcal{L}(f)(s) = \int_0^\infty e^{-st} f(t)\ dt$$

Para los S $\in \Re$ para los cuales converge esta integral.

Según veremos esta integral converge para un considerable numero de funciones y la función está defin$\mathcal{L}(f)(s)$e semirrectas de la forma (a,+∞).

La cantidad compleja S= σ+$j\omega$. De esta forma se generaliza el concepto de frecuencia en la Transformada de Fourier.

Se hace notar que el límite inferior de la integral es 0, lo cual proporciona una misma Transformada para señales causales ya que *x(t)* y *x(t)u(t)* son iguales.

La Transformada de Laplace existe si la integral que la define es finita. Para ello se necesita que los valores de σ sean unos concretos, lo que define una región de convergencia de la Transformada de Laplace.

Con la Transformada de Laplace se generaliza el concepto de función de Transferencia de un sistema a aquellos cuyas condiciones iniciales son no nulas.

2.2.1.- La Transformada Inversa de Laplace. La utilización práctica de la transformada de Laplace requiere no sólo el cálculo de la misma a partir de una función dada, sino también el problema inverso, es decir, encontrar una función *f* conocida como su transformada de Laplace. Sea *f*(s) la Transformada de Laplace de una función *f*(t). La Transformada Inversa de Laplace (o Antitransformada) de *f*(s) se denota:

$$\mathbf{L^{-1}\{ f(s)\} = f(t)}$$

Método para hallar la Antitransformada de Laplace: Existen varios métodos para determinar la antitransformada de Laplace; a continuación se explicará el *Método de las Fracciones Parciales*.

Cualquier función racional de la forma P(s) / Q(s), donde P(s) y Q(s) son polinomios en los cuales el grado de P(s) es menor que el de Q(s), puede escribirse como una suma de fracciones parciales de la forma $\mathbf{A / (as + b)^r}$, donde A es una constante y r = 1,2,3 .... Al hallar las antitransformadas de cada fracción parcial, se halla L^{-1} { P(s)/ Q(s)}.

2.2.2.- Algunas Propiedades de la Transformada de Laplace.

1. Suma y Resta

Sean $F_1(s)$ y $F_2(s)$ las transformadas de Laplace de $f_1(t)$ y $f_2(t)$ respectivamente. Entonces:

$$L\ \{ f_1(t) + f_2(t) \} = F_1(s) + F_2(s)$$

2. Multiplicación por una constante

Sea k una constante y F(s) la transformada de Laplace de $f(t)$. Entonces:

$$L\ \{ kf(t)\} = kF(s)$$

3. Diferenciación

Sea F(s) la transformada de Laplace de $f(t)$, y $f(0)$ es el límite de $f(t)$ cuando t tiende a cero. La Transformada de Laplace de la derivada con respecto al tiempo de $f(t)$ es:

$$L\ \{ df(t)/dt\} = sF(s) - \lim_{t \to 0} f(t) = sF(s) - f(0)$$

En general, para las derivadas de orden superior de $f(t)$:

$$L\ \{ d^n f(t)/dt^n\} = s^n F(s) - s^{n-1} f f(0) - s^{n-2} f^{(1)}(0) - ..... - f^{(n-1)}(0).$$

4. Teorema del Valor Inicial

Si la Transformada de Laplace de f(t) es F(s), entonces:

$$\lim_{t \to 0} f(t) = \lim_{s \to \infty} s\,F(s)$$

si el límite existe.

2.2.3.- Algunos pares de la transformada de Laplace

$f(t)$	$F(s)$
impulso unitario $\delta(t)$	1
escalón unitario $1(t)$	$\frac{1}{s}$
t	$\frac{1}{s^2}$
e^{-at}	$\frac{1}{s+a}$
te^{-at}	$\frac{1}{(s+a)^2}$
$sen(\omega t)$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
$t^n \qquad (n=1,2,3,.....)$	$\frac{n!}{s^{n+1}}$
$\frac{1}{b-a}(e^{-at}-e^{-bt})$	$\frac{1}{(s+a)(s+b)}$
$\frac{1}{b-a}(be^{-bt}-ae^{-at})$	$\frac{s}{(s+a)(s+b)}$
$\frac{1}{ab}\left[1+\frac{1}{a-b}(be^{-at}-ae^{-bt})\right]$	$\frac{1}{s(s+a)(s+b)}$
$e^{-at}sen(\omega t)$	$\frac{\omega}{(s+a)^2+\omega^2}$
$e^{-at}\cos(\omega t)$	$\frac{s+a}{(s+a)^2+\omega^2}$
$\frac{1}{a^2}(at-1+e^{-at})$	$\frac{1}{s^2(s+a)}$
$\frac{\omega_n}{\sqrt{1-\zeta^2}}e^{-\zeta\omega_n t}sen(\omega_n\sqrt{1-\zeta^2}t)$	$\frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$
$\frac{-1}{\sqrt{1-\zeta^2}}e^{-\zeta\omega_n t}sen(\omega_n\sqrt{1-\zeta^2}t+\phi)$ $\phi=\tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}$	$\frac{s}{s^2+2\zeta\omega_n s+\omega_n^2}$
$1-\frac{1}{\sqrt{1-\zeta^2}}e^{-\zeta\omega_n t}sen(\omega_n\sqrt{1-\zeta^2}t+\phi)$ $\phi=\tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}$	$\frac{\omega_n^2}{s\left(s^2+2\zeta\omega_n s+\omega_n^2\right)}$

Tabla 2.3

2.3.- Transformada Z

La ***Transformada Zeta*** es un modelo matemático que se emplea entre otras aplicaciones en el estudio del Procesamiento de Señales Digitales, como son el análisis y proyecto de Circuitos Digitales, los Sistemas de Radar o Telecomunicaciones y especialmente los Sistemas de Control de Procesos por computadoras.

La transformada Z es un ejemplo más de Transformada, como lo son la Transformada de Fourier para el caso de tiempo discreto y las Transformada de Fourier y Laplace para el caso del tiempo continuo.

La importancia del modelo de la Transformada Z radica en que permite reducir Ecuaciones en Diferencias o ecuaciones recursivas con coeficientes constantes a Ecuaciones Algebraicas lineales.

La transformada Z es la contraparte en tiempo discreto de la transformada de Laplace en tiempo continuo.

En la práctica aparecen muchas señales de tiempo discreto mediante el muestreo de una señal de tiempo continuo x(t).

La transformada Z hace posible el análisis de ciertas señales discretas que no tienen transformada de Fourier en tiempo discreto; pudiéndose demostrar que la transformada Z se reduce, a la transformada de Fourier de tiempo discreto cuando la variable de transformación es unitaria ó sea cuando |Z| = 1.

La transformada Z de una señal de tiempo discreto $x[n]$ se define como:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n}$$

donde z es una variable compleja.

La transformada Z de una señal $x[n]$ se denota por:

$$X(z) \equiv Z\{x[n]\}$$

Mientras que la relación entre $x[n]$ y $X(z)$ se indica mediante:

$$x[n] \xleftrightarrow{z} X(z)$$

Desde un punto de vista matemático, la transformada *z* es simplemente una representación alternativa de la señal. De este modo el coeficiente de Z^{-n}, para una transformada determinada, es el valor de la señal en el instante *n*. Y por tanto, el exponente de *Z* contiene la información necesaria para identificar las muestras de la señal.

2.3.1.- Región de convergencia de la transformada z.

Como se puede observar, la transformada *Z* se puede expresar como una serie de potencias infinita y existe sólo para aquellos valores de *z* para los cuales converge la serie. De esta forma, se define la región de convergencia (ROC) de *X*(*z*) como el conjunto de todos los valores de *z* para los cuales *X*(*z*) adquiere valores finitos.

Siempre que se calcule la transformada *Z* de una secuencia, se debe también indicar su correspondiente ROC. En el ejemplo 1, *X*(*z*) toma valores finitos para todo *Z* excepto para el punto *z*=0, y por tanto la ROC se define como C-{0}.Ver figura.

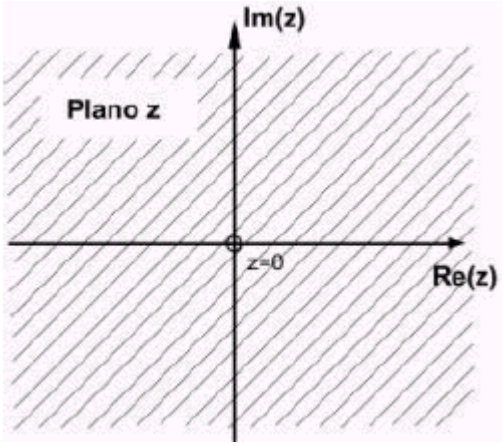


Figura 2.1

A continuación se citan algunas conclusiones acerca de la relación entre el tipo de señal bajo estudio, en el dominio del tiempo, y la ROC, en el dominio *Z:*

1) La ROC de una señal limitada por la derecha es el $(x[n] = 0 \; para \; n > N_1 > -\infty)$ interior de una circunferencia de radio r_1, pudiendo incluir o no el punto *Z*=0.

2) La ROC de una señal limitada por la izquierda $(x[n]=0 \; para \; n<N_2<\infty)$ es el exterior de una circunferencia de radio r_2, pudiendo incluir o no el punto Z=0.

3) Si $x[n]$ es una secuencia bilateral, esto es, una señal de duración infinita que no está limitada por la derecha ni por la izquierda, entonces la ROC es una región anular comprendida entre dos radios cualesquiera tales que $r_1 > r_2$.

4) Si $x[n]$ es una secuencia de duración finita entonces la ROC es todo el plano Z excepto quizás los puntos Z=0 y/o Z=∞ .

2.3.2.- Propiedades de la transformada Z.

En la tabla siguiente, se muestran de forma resumida las propiedades más importantes que cumple la transformada Z.

PROPIEDAD	Secuencia	Transformada z	ROC
Linealidad	$a_1x_1[n]+a_2x_2[n]$	$a_1X_1(z)+a_2X_2(z)$	$\{R_{x_1} \cap R_{x_2}\} \subset ROC$
Desplazamiento temporal	$x[n-n_o]$	$z^{-n_o}X(z)$	$ROC = R_x$
Escalado en frecuencia	$z_o^n x[n]$	$X(z/z_o)$	$ROC = \lvert z_o \rvert \cdot R_x$
Reflexión temporal	$x^*[-n]$	$X^*(1/z^*)$	$ROC = 1/R_x$
Diferenciación de $X(z)$	$n \cdot x[n]$	$-z\frac{d}{dz}X(z)$	$ROC = R_x$
Convolución de secuencias	$x_1[n]*x_2[n]$	$X_1(z)\cdot X_2(z)$	$\{R_{x_1} \cap R_{x_2}\} \subset ROC$
Teorema del valor inicial	$x[0]$	$\lim_{z\to\infty} X(z)$	

Tabla 2.4

Propiedad de la convolución.

De todas esas propiedades destaca por su importancia y utilidad la propiedad de la convolución.

Para calcular la convolución de dos señales usando la transformada z, se deberán seguir los siguientes pasos:

1. Calcular las transformadas *Z* de las señales. $X_1(z) = Z\{x_1[n]\} \quad X_2(z) = Z\{x_2[n]\}$
2. Multiplicar las dos transformadas *Z*. $X(z) = X_1(z) \cdot X_2(z)$
3. Encontrar la transformada *Z* inversa de *X*(*z*). $x[n] = Z^{-1}\{X(z)\}$

2.3.3.- Transformadas Z racionales.

Polos y ceros.

Los ceros de la transformada z, *X*(z), son los valores de *Z* para los cuales $X(z) = 0$.

Por el contrario, los polos de la transformada z son los valores de *z* para los cuales $X(z) = \infty$.

Si *X*(*z*) es una función racional de la forma:

$$X(z) = \frac{N(z)}{D(z)} = \frac{b_o + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_o + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}$$

Si $a_o \neq 0$ y $b_o \neq 0$, entonces se podrá factorizar los término $b_o z^{-M}$ y $a_o z^{-N}$ de la siguiente forma:

$$X(z) = \frac{N(z)}{D(z)} = \frac{b_o z^{-M}}{a_o z^{-N}} \cdot \frac{z^M + (b_1/b_o) z^{M-1} + \ldots + (b_M/b_o)}{z^N + (a_1/a_o) z^{N-1} + \ldots + (a_N/a_o)}$$

$$X(z) = \frac{N(z)}{D(z)} = \frac{b_o}{a_o} z^{-M+N} \cdot \frac{(z-z_1)\cdot(z-z_2)\cdot\ldots\cdot(z-z_M)}{(z-p_1)\cdot(z-p_2)\cdot\ldots\cdot(z-p_N)}$$

$$X(z) = G \cdot z^{N-M} \cdot \frac{\prod_{k=1}^{M} (z - z_k)}{\prod_{k=1}^{N} (z - p_k)}$$

A partir de esta última expresión, se puede representar gráficamente *X*(*z*) por un diagrama de polos y ceros en el plano complejo, en el cual se representan los polos por cruces (X) y los ceros por círculos (O). Obviamente, la ROC de la transformada *Z* no deberá contener ningún polo.

2.3.4.- La transformada Z inversa.

Una de las aplicaciones más importantes de la transformada Z es el análisis de sistemas lineales en tiempo discreto. A menudo para realizar este análisis es necesario calcular primero la transformada Z de las secuencias a tratar y después realizar todo el análisis en el dominio transformado. Por último, se hace necesario pasar de nuevo al dominio del tiempo y para ello se ha de calcular la transformada Z inversa.

Para las típicas clases de secuencias y de transformadas Z con las que trabajaremos en el análisis de sistemas de tiempo discreto lineales e invariantes con el tiempo, la transformada inversa se puede obtener simplemente por inspección de ciertas parejas de transformadas. En la siguiente tabla se presentan algunas de las parejas de transformadas más comunes que nos podemos encontrar. En general, las tablas de transformadas Z tienen una gran utilidad dentro del método de inspección. Si la tabla es amplia, puede ser factible expresar una determinada transformada Z como suma de términos, y obtener la transformada inversa de cada uno de ellos a través de la tabla.

	Secuencia	Transformada	ROC
1	$\delta[n]$	1	$Todo\ z$
2	$u[n]$	$\frac{1}{1-z^{-1}}$	$\|z\|>1$
3	$-u[-n-1]$	$\frac{1}{1-z^{-1}}$	$\|z\|<1$
4	$\delta[n-m]$	z^{-m}	$Todo\ z\ excepto\ 0\ (si\ m>0)$ $o\ \infty\ (si\ m<0)$
5	$a^n u[n]$	$\frac{1}{1-a\cdot z^{-1}}$	$\|z\|>\|a\|$
6	$-a^n u[-n-1]$	$\frac{1}{1-a\cdot z^{-1}}$	$\|z\|<\|a\|$
7	$n\,a^n u[n]$	$\frac{a\,z^{-1}}{(1-a\cdot z^{-1})^2}$	$\|z\|>\|a\|$
8	$-n\,a^n u[-n-1]$	$\frac{a\,z^{-1}}{(1-a\cdot z^{-1})^2}$	$\|z\|<\|a\|$
9	$[\cos\omega_o n]\cdot u[n]$	$\frac{1-[\cos\omega_o]z^{-1}}{1-[2\cdot\cos\omega_o]z^{-1}+z^{-2}}$	$\|z\|>1$
10	$[sen\,\omega_o n]\cdot u[n]$	$\frac{[sen\,\omega_o]z^{-1}}{1-[2\cdot cos\,\omega_o]z^{-1}+z^{-2}}$	$\|z\|>1$
11	$[r^n\cos\omega_o n]\cdot u[n]$	$\frac{1-[r\cdot\cos\omega_o]z^{-1}}{1-[2\cdot r\cdot\cos\omega_o]z^{-1}+r^2\cdot z^{-2}}$	$\|z\|>r$
12	$[r^n sen\,\omega_o n]\cdot u[n]$	$\frac{[r\cdot sen\,\omega_o]z^{-1}}{1-[2\cdot r\cdot cos\,\omega_o]z^{-1}+r^2\cdot z^{-2}}$	$\|z\|>r$
13	$\begin{cases} a^n & 0\le n\le N-1 \\ 0 & en\ el\ resto \end{cases}$	$\frac{1-a^N\cdot z^{-N}}{1-a\cdot z^{-1}}$	$\|z\|>0$

Tabla 2.5

2.3.5.- Análisis y caracterización de sistemas LTI mediante la transformada Z.

La transformada Z es una herramienta importante en el análisis y representación de sistemas LTI discretos. Esto es debido principalmente a la propiedad de convolución, y por la cual la convolución en el dominio del tiempo es equivalente a una multiplicación en el dominio de la transformada Z.

$$Y(z) = H(z) \cdot X(z),$$

donde $X(z)$, $Y(z)$, y $H(z)$ son las transformadas Z de la entrada, la salida y la respuesta al impulso del sistema, respectivamente. Además, a $H(z)$ se le conoce como la función de transferencia del sistema.

Para Z evaluada en la circunferencia unidad $(z = e^{j\omega})$, e reduce a la respuesta en frecuencia del sistema, siempre que la circunferencia unidad esté contenida dentro de la ROC de $H(z)$.

Muchas propiedades de un sistema están relacionadas directamente con las características de los polos, de los ceros y de la región de convergencia de la función del sistema. A continuación se van a mostrar algunas de estas relaciones mediante el estudio de algunas propiedades del sistema.

Causalidad.

Un sistema LTI causal tiene una respuesta al impulso $h[n]$ que es cero para $n<0$. En este caso la transformada Z se puede expresar como:

$$H(z) = \sum_{n=0}^{\infty} \left(\frac{h[n]}{z^{+n}} \right)$$

Donde la correspondiente ROC será el exterior de un círculo en el plano Z que incluirá también el infinito.

De este modo se puede decir que un sistema LTI discreto es causal si y sólo si la ROC de su función de transferencia del sistema es el exterior de un círculo, incluyendo el infinito.

Si $H(z)$ es racional (con M ceros y N polos) y causal, entonces la ROC debe estar fuera del polo más externo y debe incluir el infinito. Por tanto, el límite de $H(z)$ cuando $Z \rightarrow \infty$ debe ser finito.

Estabilidad.

La estabilidad de un sistema LTI discreto requiere que su respuesta al impulso sea absolutamente sumable.

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty$$

Por otro lado, $|H(z)|$ es finita sólo si la secuencia $h[n] \cdot r^{-n}$ es absolutamente sumable.

$$|H(z)| \leq \sum_{n=-\infty}^{\infty} |h[n] \cdot r^{-n}| < \infty$$

Comparando ambas expresiones se puede llegar a la siguiente conclusión: Si el sistema $h[n]$ es estable, entonces la ROC de $H(z)$ debe incluir a la circunferencia unidad (r=1).

Por tanto se puede decir que un sistema LTI es estable si y sólo si la ROC de la función de transferencia del sistema, $H(z)$ incluye la circunferencia unidad, $|z|$=1.

En el caso concreto de tener un sistema LTI con una función de transferencia racional, el sistema será estable y causal si y sólo si todos los polos de $H(z)$ caen dentro del círculo unidad, es decir, si todos tienen una magnitud menor que 1.

2.4.- Resumen.

El capítulo recién pasado estuvo abocado en poder entregar las diversas herramientas que son utilizadas para el análisis de los sistemas de procesado de señales, como lo son las transformadas de Fourier; Laplace y Z, mediante estas se pueden obtener las respuestas tanto en frecuencia como en el tiempo de acuerdo a las necesidades del estudio.

Aquí se dieron las principales propiedades de estas transformadas como así también se muestran las transformadas de las señales más utilizadas en el tratamiento digital de señales.

Interpolación y Diezmado

3. Introducción.

En este capítulo se ha planteado como objetivo el de poder analizar los efectos introducidos por un sistema de conversión analógico/digital real y los cambios en la velocidad de muestreo de una señal, a través del uso de las operaciones de diezmado e interpolación. Los cambios en las tasas de muestreo encuentran aplicación en numerosos campos, como son: comunicaciones, compresión de datos, procesado de audio, etc.

También se dará a conocer uno de los aspectos principales dentro del campo de las comunicaciones como lo es el teorema del muestreo, ya en este se basa como deben ser muestreadas las señales.

3.1.- Conceptos previos.

La mayoría de las señales de interés práctico en los sistemas de comunicación, tales como la voz, las señales biológicas, sísmicas, radar, etc., tienen una naturaleza analógica, ya que están definidas en un intervalo continuo de tiempo y de amplitudes. Así, las señales eléctricas en un televisor, en un teléfono o en un radio-transmisor tienen una naturaleza analógica, desde un punto de vista temporal. Para procesar estas señales utilizando medios digitales, es necesario diseñar sistemas que las conviertan a secuencias y que permitan devolverlas al formato analógico con el menor grado de distorsión posible. Para diseñar este sistema de representación de señales continuas en el tiempo sobre una base de tiempos discreta, debemos plantearnos una serie de condiciones:

- Debe ser invertible
- Debe preservar toda la información de interés presente en la señal.
- Debe ser físicamente realizable.

Una forma práctica de poder llevar a cabo este proceso, cumpliendo las condiciones anteriores, es utilizar el muestreo periódico. El muestreo periódico no es, en general, una operación invertible. Sin embargo, cuando se restringe el tipo de señales sobre las que se aplica, puede eliminarse esa ambigüedad. Otro punto de este trabajo se dedicara a estudiar algunas consideraciones del diseño real de conversores de señales analógicas a digitales y viceversa.

Por último, en muchos casos es necesario cambiar la velocidad de muestreo de la señal continua subyacente, operando en el dominio discreto, bien aumentando, bien disminuyendo la tasa de muestras. De ahí el interés del estudio de los métodos para aumentar y reducir la velocidad de muestreo de forma digital, llamados interpolación y diezmado, respectivamente.

3.1.1.- Conversión tiempo Continuo/Discreto

Conversores C/D ideales

El sistema de conversión C/D (tiempo Continuo - tiempo Discreto) ideal, mostrado en la figura, recibe una entrada definida en tiempo continuo y produce una señal discreta en el tiempo.
La señal de entrada y de salida del sistema se relacionan temporalmente a través de la expresión X[n] = Xc(nT), que en el dominio transformado da lugar a:

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\left(\frac{1}{T}(\omega - 2k\pi)\right)$$

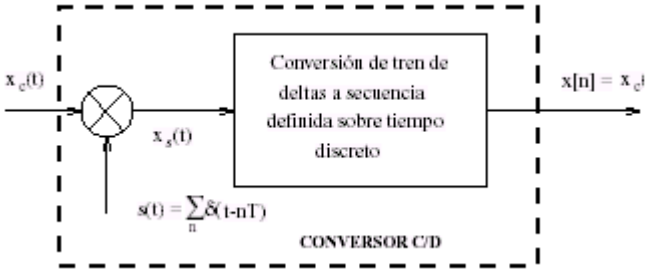


Figura 3.1

A partir de esta ecuación, podemos establecer que $X(e^{j\omega})$ es una superposición de versiones desplazadas y escaladas en frecuencia de $X_c(\Omega)$ que ha sido normalizada en frecuencia por el factor $\omega = \Omega T$. Esta normalización en frecuencia esta íntimamente ligada a la normalización temporal realizada en la transformación de Xs(t) a X[n] (el espaciado entre muestras consecutivas de Xs(t) es T y entre muestras consecutivas de X[n] es 1

3.1.2.- Conversión Tiempo Discreto/Continuo.

Si suponemos que nuestro muestreo se ha efectuado de acuerdo con el Teorema de Nyquist [1], el Conversor ideal Tiempo-Discreto/Tiempo-Continuo de la Figura, recupera la señal continua original Xr(t) = Xc(t).

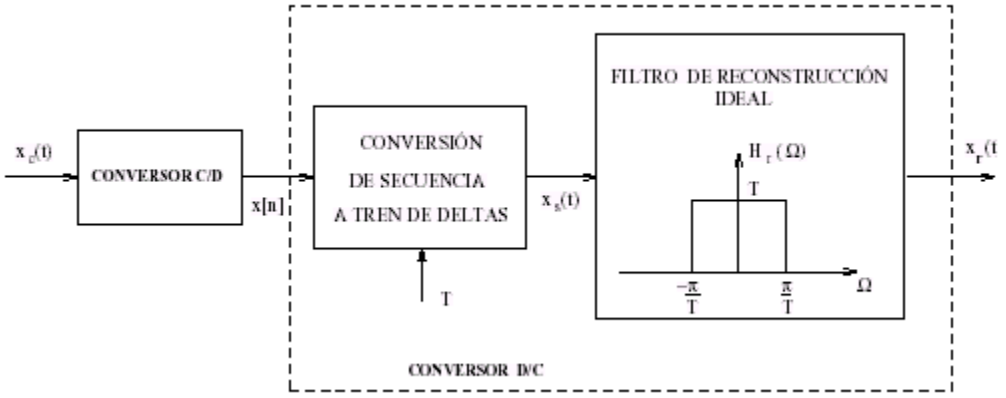


Figura 3.2

En este caso, la relación entre la señal continua de salida y la secuencia discreta de la que proviene es:

$$x_r(t) = \sum_{k=-\infty}^{+\infty} x[k] h_r(t - kT)$$

relación, que en el dominio frecuencial resulta en:

$$X_r(\Omega) = H_r(\Omega) X(e^{j\Omega T}) = \begin{cases} TX(e^{j\Omega T}), & |\Omega| \leq \pi/T \\ 0, & \text{en otro caso} \end{cases} = X_c(\Omega)$$

[1] *El teorema de Nyquist será definido en el siguiente apartado de este capítulo.*

3.2.- Teorema de Muestreo (Nyquist).

El proceso idealizado de muestreo de una señal y de la reconstrucción subsiguiente de la señal desde las muestras se representa en la siguiente figura:

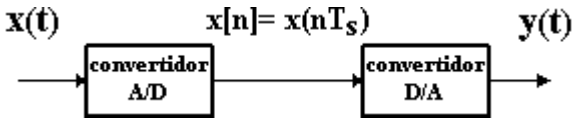


Figura 3.3

Esta figura muestra una señal continua X(t), que se muestrea usando un convertidor A/D para producir una sucesión de valores discretos X[n] =X(nTs), donde n es un entero que es el índice de muestreo y Ts es el período de muestreo. La frecuencia de muestreo es el valor fs = 1/Ts. El convertidor D/A ideal discreto permite transformar de nuevo los valores discretos e interpolar una curva suave entre ellos. El Teorema de Muestreo nos dice que si se elige una frecuencia de muestreo superior a dos veces la frecuencia mayor, f_{max}, presente en la señal de entrada, es decir $fs > 2*f_{max,}$ entonces la salida Y(t) en el sistema de la figura anterior, será igual a la entrada X(t) si se reconstruye adecuadamente la señal. Para obtener la frecuencia fmáx se puede representar la entrada como una suma de sinuoidales (Transformación de Fourier) y fmáx será la frecuencia asociada a la componente de mayor frecuencia con amplitud distinta de cero.

La mayoría de los computadores tienen un convertidor análogico-digital incorporado (A/D) y un convertidor digital- análogico (D/A) incluido en la tarjeta de sonido. Estos sistemas son las realizaciones físicas de los conceptos idealizados de convertidores A/D y D/A respectivamente.

3.3.- Diezmado e Interpolación.

Las operaciones de diezmado e interpolación modifican el número de muestras presentes en una señal, bien aumentando o bien disminuyendo la velocidad de muestreo de la señal continua subyacente.

Este tipo de operaciones son importantes en situaciones en las que, por ejemplo, existe un sobremuestreo que hace posible deshacerse de parte de las muestras sin perder información.

Aunque todo el procesado se haga de forma discreta, podemos hablar de velocidad de muestreo igualmente, ya que la señal discreta X[n] puede ser considerada como el resultado de haber muestreado una señal continua Xc(t), y el diezmar o interpolar X[n] equivale a modificar la velocidad de muestreo de la señal continua Xc(t) que subyace a lo largo de todo el proceso, aunque esta no aparezca nunca de forma explícita. Así cabe hablar por ejemplo de efectos de "aliasing" al diezmar una señal, aunque el proceso sea de naturaleza discreta.

Las operaciones de diezmado e interpolación pueden ser consideradas duales entre sí, como se aprecia en la figura:

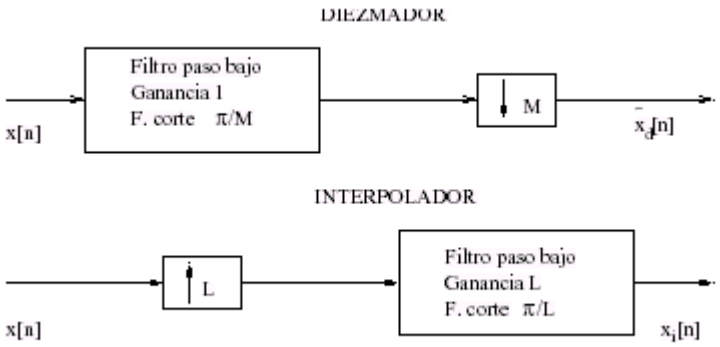


Figura 3.4

El filtro paso bajo presente en el diezmador tiene como misión evitar el posible "aliasing" causado por la reducción de la tasa de muestreo. Es decir, el descartar muestras puede no ser una operación invertible si la nueva velocidad de muestreo no satisface el criterio de Nyquist, o lo que es lo mismo, si en frecuencia se produce solapamiento de espectros.

Cabe observar que la operación de interpolación es invertible si el filtro paso bajo no se hace cero en el intervalo $\left(-\frac{\pi}{L}, \frac{\pi}{L}\right)$.

3.3.1.- Diezmado.

El concepto de diezmado, si bien en su etimología se refiere a una reducción en un factor de diez, se aplica de forma genérica al muestreo de una señal discreta por un factor entero, es decir, a preservar una de cada M muestras de la señal discreta. La ecuación que lo describe es muy sencilla:

$$x_d[n] = x[nM]$$

que indica que las muestras presentes en la señal diezmada se obtienen de las muestras de la señal original en los instantes nM.

Hay que indicar que en varios textos, la operación que acabamos de explicar se denomina compresión, dejándose el concepto de diezmado para el caso en el que además se incluye un filtro paso bajo. Con esta salvedad en mente, y abusando un poco de la notación, aquí denotaremos a veces la operación anterior como diezmado, aunque no exista tal filtro paso bajo.

A diferencia del caso continuo, el diezmado implica una aparente pérdida de información.

Es decir, cuando una señal continua sufre una compresión la señal resultante Y(t) = X(Mt) contiene todos los valores presentes en la original, aunque comprimidos en el tiempo. Sin embargo, al realizar esa compresión con una señal discreta, de cada M muestras desaparecen M-1. Dado que el proceso es análogo al muestreo de una señal continua, cabe preguntarse si se pueden aplicar las mismas ideas acerca de la recuperación de la información: límite de Nyquist, aliasing, etc. Ese es el caso, y a continuación vamos a ver como el estudio en frecuencia nos ilustra al respecto.

Parece existir un límite a la tasa de diezmado que se puede permitir sin perder información sobre las frecuencias presentes, y por tanto, sobre la señal en sí. Como ya es conocido, el espectro de la señal diezmada tiene la forma:

$$X_d(e^{j\omega}) = \frac{1}{M}\sum_{i=0}^{M-1} X(e^{j(\omega/M - 2\pi i/M)})$$

Cualquier frecuencia que este por encima de π / M va a producir un solapamiento de espectros, dado que al igual que en el muestreo de una señal continua, la señal muestreada consiste en la repetición periódica del espectro original (con una posterior expansión debido a

la compresión en el tiempo). Así, el teorema del muestreo para señales continuas se puede extender al caso discreto:

- Señal continua de máxima frecuencia Ω_M rad/s $\longrightarrow$ periodo de muestreo $T_s < \frac{\pi}{\Omega_M}$.
- Señal discreta de máxima frecuencia ω_M rad/s $\longrightarrow$ tasa de diezmado $M < \frac{\pi}{\omega_M}$.

En el caso de que no se verifique el teorema del muestreo se producirá aliasing, debido al solapamiento de las réplicas.

Así, como hemos visto anteriormente, la condición para evitar el aliasing es $M < \frac{\pi}{\omega_M}$ (M puede ser racional). Eso indica que en muchos casos tenemos exceso de información en las muestras presentes, dado que parte del espectro está a libre.

Al igual que ocurre en el caso continuo, en muchas ocasiones es preciso introducir un filtro paso bajo previo para evitar el aliasing introducido por el muestreo. Idealmente ese filtro ha de ser un filtro paso bajo con frecuencia de corte π/M, la máxima frecuencia que puede estar presente sin incurrir en aliasing. Ese filtro suaviza los cambios bruscos que no pueden ser seguidos con la nueva tasa de muestreo, de modo que se evite el que esas altas frecuencias se confundan posteriormente con otras frecuencias más bajas.

3.3.2.- Interpolación.

La operación de interpolación, si bien se realiza de forma discreta, puede ser dividida conceptualmente en dos fases:

- Reconstrucción de la señal continua de la cual se deriva la discreta muestreando.
- Muestreo de la señal continua a la nueva velocidad.

En el primero de los pasos habrá que utilizar un filtro paso bajo. Ese filtro paso bajo se puede realizar de forma discreta, aunque para ello hay que efectuar una expansión de la señal, es decir, insertar ceros para permitir la inclusión de las nuevas muestras, que serán las que conceptualmente se obtendrían en el segundo de los pasos anteriores. Por tanto, la operación de interpolación se compone de una operación de expansión seguida de un filtrado paso bajo.

El filtro interpolador ideal es un filtro paso bajo de frecuencia de corte y $\frac{\pi}{L}$ ara L. La respuesta en tiempo de ese filtro ideal corresponde a una sinc:

$$h[n] = sinc(\frac{n}{L}) \longrightarrow H(e^{j\omega}) = \begin{cases} L & 0 \leq |\omega| < \frac{\pi}{L} \\ 0 & \text{resto} \end{cases}$$

Con esa respuesta impulsional, la ecuación que relaciona la salida del interpolador Xi[n] con la entrada X[n] queda de la forma:

$$x_i[n] = \sum_k x[k] sinc(\frac{n - kL}{L})$$

En situaciones prácticas se usan filtros sencillos, como pueden ser interpoladores de orden cero y de orden uno.

La respuesta impulsional de un interpolador de orden cero es de la forma:

$$h[n] = \begin{cases} 1 & 0 \leq n < L \\ 0 & \text{resto} \end{cases}$$

Podemos realizar la interpolación de una señal mediante el uso de la Transformada Discreta de Fourier y su inversa. Esta interpolación se realiza matemáticamente una vez tenemos todos los puntos muestreados de la señal, es decir, se aplica en tiempo no real.

Deben seguirse los siguientes pasos:

- Se realiza la Transformada Discreta de Fourier de la señal de N puntos obtenida al muestrear. Para ello, puede emplearse la implementación del algoritmo para la FFT vista anteriormente.
- Se añaden ceros al espectro. Para ello tenemos que separar la secuencia de valores obtenida como resultado de la Transformada en dos mitades y añadir entre ellas los ceros que deseemos (tantos como nuevos puntos se quieran obtener).
- A la secuencia de valores obtenida una vez añadidos los ceros se le aplica la Inversa de la Transformada Discreta de Fourier obteniéndose, como resultado, la señal inicial pero con más puntos (tantos más como ceros introducidos). La Inversa de la Transformada puede realizarse mediante la IFFT.

Utilizando la función 'FFT' resulta muy sencillo aplicar este método de interpolación con el que además, como ya se comentó, si la señal original ha sido muestreada cumpliendo el teorema del muestreo, se asegura que la señal será reconstruida con exactitud. Sin embargo, el inconveniente que posee el método es que no puede aplicarse en tiempo real (no pueden obtenerse nuevos puntos a medida que van obteniéndose las muestras de la señal), sino que sólo puede realizarse la interpolación cuando se dispone de todas las muestras. Esto es lógico ya que para calcular la Transformada se necesitan todos los puntos.

3.4.- Resumen.

En este capítulo se procedió en dar a conocer los conceptos de interpolación y diezmado, ya que estos son utilizados en algunos casos para la consecución de muestreo de señales, ya sea para obtener una mejor calidad de onda.

Por eso se dio a entender de que manera son útiles, dando sus principales características y sus ecuaciones que los rigen, una de las principales aplicaciones es en el diseño de filtros digitales.

Diseño de Filtros Digitales

4. Introducción.

Hoy en día, se utilizan ampliamente en el mundo de las telecomunicaciones el filtrado, como puede ser para mejorar la calidad de recepción de alguna voz que se transmite en algún canal telefónico, o como también en dejar pasar hasta cierta frecuencia algún tipo de información, este capítulo estará abocado en describir los principales parámetros que están involucrados en el diseño de filtros digitales y entrar en detalle en los filtros FIR e IIR.

En el proceso de diseño de un filtro digital se determinan los coeficientes de un filtro FIR o IIR causal que aproxima de forma precisa las especificaciones de respuesta en frecuencia. La elección de uno de los dos tipos de filtro, FIR o IIR, dependerá de la naturaleza del problema y de las especificaciones de la respuesta en frecuencia deseada.

En la práctica, los filtros FIR se emplean en problemas de filtrado donde hay un requisito de fase lineal dentro de la banda de paso del filtro. Si no existe este requisito es posible escoger uno de tipo IIR o FIR. Como regla general los filtros IIR tiene lóbulos laterales menores en la banda de rechazo que un filtro FIR con el mismo tipo de diseño. Por ésta razón, si se puede tolerar alguna distorsión en la fase debido a su no-linealidad, o simplemente no es importante, se prefiere un filtro IIR. Esto se debe a que su implementación involucra menos parámetros, requiere menos memoria y tiene menor complejidad computacional.

Para la tarea en el diseño de filtros digitales se facilita enormemente por la disponibilidad de numerosos programas de software. Durante esta trabajo se utilizará MATLAB como herramienta de diseño. La implementación de los filtros se hará en el DSP.

4.1.- Conceptos Generales.

Un filtro digital es el Proceso computacional que genera una secuencia discreta a partir de otra, según una regla preestablecida**.**

Existen varios pasos que se pueden seguir para obtener el diseño de un filtro digital como son:

- Establecer las especificaciones del filtro para unas determinadas prestaciones. Estas son las mismas que las requeridas para el diseño de filtros analógicos (frecuencias de paso...)
- Determinar la función de transferencia que cumpla con las especificaciones dadas(determinar los coeficientes que aproxima de forma precisa las especificaciones de respuesta en frecuencia deseada)
- Realizar la función de transferencia en hardware o software

En el diseño de filtros selectivos en frecuencia. Las características deseadas del filtro se especifican en el dominio de la frecuencia en función de la respuesta del filtro en magnitud y fase. El proceso del diseño del filtro consiste bien en:

a) La selección de los coeficientes de la ecuación en diferencias, ó

b) La determinación de la respuesta impulsional h(n), de forma que se cumpla algún criterio sobre las características en el dominio del tiempo o de la frecuencia.

Etapas del diseño

- Especificación de las propiedades.
- Aproximación.
- Realización.

4.2.- Filtros FIR o Filtros IIR

Para comenzar este estudio se darán a conocer algunas de las características más relevantes de los dos tipos de filtros digitales que serán vistos en este capítulo, como los son FIR y los IIR.

- Los filtros IIR (respuesta infinita al impulso) producen distorsión de fase, es decir la fase no es lineal con la frecuencia.
- Los filtros FIR (respuesta finita al impulso) son de fase lineal (se emplean cuando se pide este requisito). En caso de que no se exija este requisito se puede emplear cualquiera de los dos tipos.
- El orden del filtro IIR es mucho menor que el FIR para una misma aplicación.
- IIR tiene lóbulos laterales menores en la banda de rechazo que uno FIR con el mismo número de parámetros. Por esta razón se puede tolerar alguna distorsión de fase, se prefiere un filtro IIR, principalmente porque su implementación involucra menos parámetros.
- Los filtros FIR son siempre estables.

Hoy día con el diseño asistido por computador la complejidad del filtro no es tan importante, y es el diseñador en cada caso particular conociendo las propiedades de cada uno de ellos el que debe de elegir el modelo que más se adapte a sus necesidades.
El principal problema en el diseño consiste en el cálculo de los coeficientes de la ecuación de diferencias que se adapten mejor a nuestras necesidades de respuesta en frecuencia.
Como se ha comentado es usual clasificar a los filtros en función de la banda de paso:

- Pasa baja
- Pasa alta
- Pasa banda
- Para banda o rechazo de banda

La respuesta en frecuencia de cualquier filtro digital es periódica en el dominio de la frecuencia repitiéndose cada Fs.
En la práctica no es posible diseñar filtros ideales, se presenta a continuación la plantilla de los diferentes tipos de filtros:

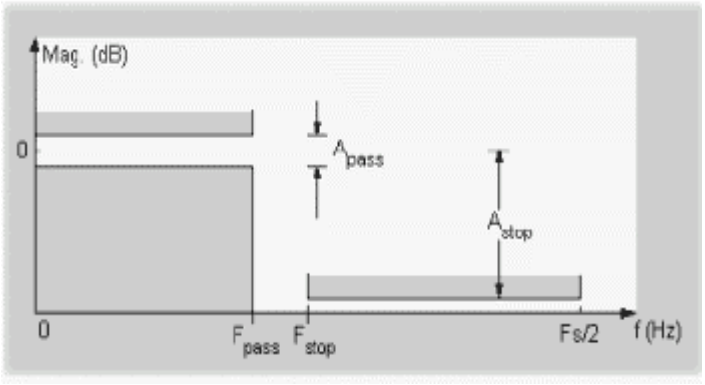


Figura 4.1 Filtro Pasa Bajo.

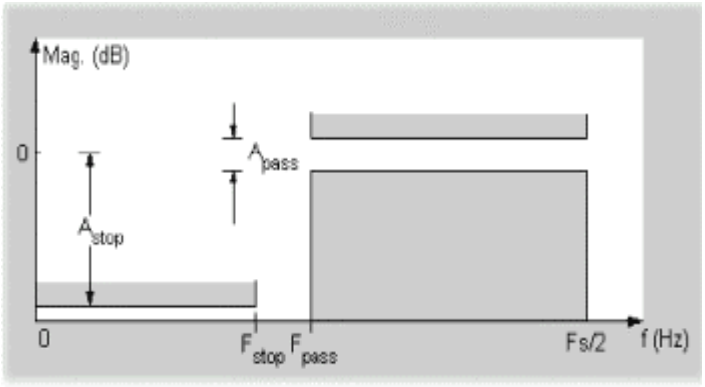


Figura 4.2 Filtro Pasa Alto.

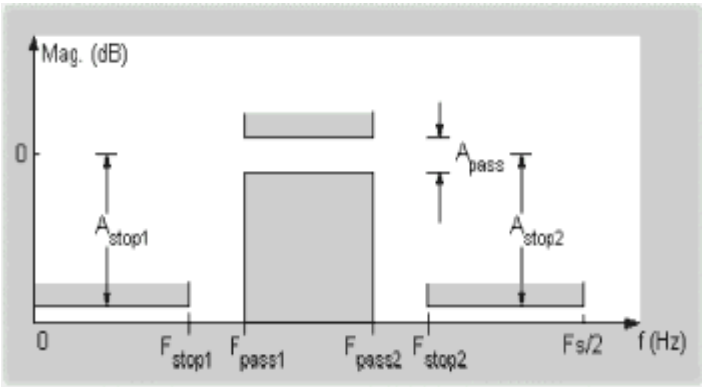


Figura 4.3 Filtro Pasa Banda.

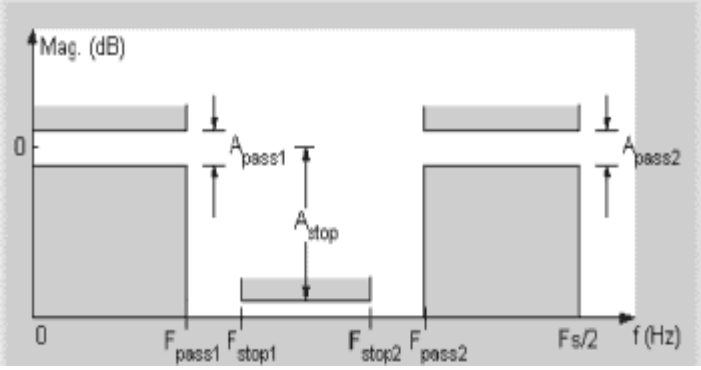


Figura 4.4 Filtro Rechaza Banda.

Nota: los métodos de diseño de filtros digitales se dividen básicamente en dos tipos:

- Basado en el diseño de filtros analógicos (simulación numérica de filtros analógicos, que es el origen histórico del Procesado Digital de la señal).
- Propio de los filtros digitales.

4.2.1.- Filtros FIR.

Un filtro FIR con longitud *M*, entrada *x(n)* y salida *y(n)* se describe por la ecuación de diferencias:

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k)$$

donde {*bk*} son los coeficientes del filtro.

Alternativamente, es posible representar la secuencia de salida como la convolución de la respuesta a impulso del sistema *h(n)* con la entrada.

$$y(n) = \sum_{k=0}^{M-1} h_d(k) x(n-k)$$

donde *x(n)* representa la entrada muestreada, $h_d(n)$ representa la respuesta a impulso ideal del filtro, e *y(n)* es la salida filtrada.

Es claro que las ecuaciones anteriores son iguales en su forma y, por lo tanto, se puede decir que $b_k = h_d(k)$, $k = 0, 1, 2,...,M-1$.

En este tipo de filtros no existe recursión, es decir, la salida sólo depende de la entrada y no de valores pasados de la salida.

La respuesta es por tanto una suma ponderada de valores pasados y presentes de la entrada, de ahí de que se denomine Media en Movimiento (Moving Average).

La función de transferencia tiene un denominador constante y sólo tiene ceros.

La respuesta es de duración finita ya que si la entrada se mantiene en cero durante M periodos consecutivos, la salida será también ceros.

4.2.2.- Filtros IIR.

Los filtros de Respuesta Infinita a Impulso, o filtros IIR, de la misma manera que los filtros FIR, son sistemas LTI (lineales e invariables en el tiempo) que recrean un amplio rango de diferentes respuestas en frecuencia.

Consideremos un sistema discreto caracterizado por la ecuación de coeficientes invariantes en el tiempo:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

Mediante la transformada *Z* se puede obtener la función de transferencia caracterizada por la ecuación anterior:

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

De esta caracterización se obtienen polos y ceros, los cuales dependen de la elección de los parámetros del sistema $\{bk\}$ y $\{ak\}$, y determinan las características de la respuesta en frecuencia del sistema.

Los filtros IIR se pueden clasificar en:
Filtros AR (autoregresivo).
La ecuación diferencia que describe este filtro AR es de la siguiente forma:

$$y[n]+A_1y[n-1]+A_2y[n-2]+...+A_N\,y[n-N]=x[n].$$

Lo que da lugar a una función de transferencia de la siguiente manera:

$$H(z) = \frac{1}{1+A_1z^{-1}+A_2z^{-2}+...A_Nz^{-N}}$$

La función de transferencia solo posee polos.
El filtro es recursivo ya que la salida depende no solo de la entrada actual sino además de valores pasados de la salida (filtros con realimentación)

El término autoregresivo tiene un sentido estadístico en que la salida $y[n]$ tiene una regresión hacia sus valores pasados.
La respuesta al impulso es de duración infinita, de ahí su nombre.

Filtros ARMA (autoregresivo y media en movimiento).
Es el filtro más general. La ecuación que describe este filtro ARMA de órden N es de la siguiente forma:

$$y[n]+A_1y[n-1]+A_2y[n-2]+...+A_N\,y[n-N] = B_0x[n]+B_1x[n-1]+B_2x[n-2]+...+B_Mx[n-M]$$

Y cuya función de transferencia es:

$$H(z) = \frac{B_0+B_1z^{-1}+...+B_Mz^{-M}}{1+A_1z^{-1}+A_2z^{-2}+...A_Nz^{-N}}$$

Un filtro de este tipo se denota por ARMA(N,M), es decir, es autoregresivo de orden N y media en movimiento de orden M.

4.3.- Diseño de Filtros Digitales.

Como bien se menciono anteriormente Un filtro digital es un sistema discreto utilizado para extraer características desde el dominio de la frecuencia sobre señales muestreadas. La operación de filtrado se realiza por medio de cálculos directos con las señales muestreadas. Las ventajas que presentan los filtros digitales frente a los analógicos son las siguientes:

- Respuesta dinámica: El ancho de banda del filtro digital está limitado por la frecuencia de muestreo, mientras que en los filtros analógicos con componentes activos suelen estar restringidos por los amplificadores operacionales.
- Intervalo dinámico: En filtros analógicos aparecen derivas que limitan por abajo el rango y se saturan con la alimentación. En cambio en los filtros digitales es fijado por el número de bits que representa la secuencia, y el límite inferior por el ruido de cuantificación y los errores de redondeo.

- Conmutabilidad: Si los parámetros de un filtro se conservan en registros, los contenidos de dichos registros pueden ser modificados a voluntad. Además, estos filtros se pueden conmutar, pudiéndose multiplexar en el tiempo para procesar varias entradas a la vez.
- Adaptabilidad: Un filtro digital puede ser implementado en soporte físico *(hardware)* o mediante un programa de computador *(software*).
- Ausencia de problemas de componentes: Los parámetros de los filtros se representan por medio de números binarios y no derivan con el tiempo. Al no haber componentes, no hay problemas de tolerancia o deriva de componentes, y ningún otro problema asociado con un comportamiento no ideal de resistencias, condensadores, bobinas o amplificadores. Tampoco existen problemas de impedancia de entrada ni salida, ni efectos de adaptación de impedancias entre etapas.

Una distinción fundamental en los sistemas discretos dinámicos lineales e invariantes, y en particular en los filtros digitales, es la duración de la respuesta ante el impulso. Se habla de sistemas de respuesta de pulso finito o no recursivo *(FIR, finite impulse response)* y de sistemas de respuesta infinita o recursivo *(IIR, infinite impulse response*).

4.3.1.- Diseño de filtros no Recursivos FIR.

Los filtros no recursivos tienen ventajas muy interesantes que les hacen ser ampliamente utilizados en múltiples aplicaciones. La característica más destacable es su facilidad de diseño para conseguir una respuesta en frecuencias de fase lineal, esto es, la señal que pase a través de él no será distorsionada. Los FIR son por su propia constitución estables, no habiendo problemas en su diseño o fase de implementación.

Aunque el diseño de los FIR requiera de una gran cantidad de operaciones de sumas y multiplicaciones, tanto su estructura de programación como su implementación en soporte físico resulta fácil y escalable.

Su mayor desventaja está en que para iguales requisitos de especificaciones del filtro resulta con menor orden los IIR que los FIR, implicando programas más largos o circuitos mayores.

Básicamente hay dos métodos para el diseño de filtros no recursivos. El primero trata de definir la respuesta en frecuencia del filtro para luego determinar los coeficientes del filtro mediante la transformada inversa de Fourier; mientras que la segunda estrategia utiliza métodos de

optimización capaz de ir modificando los coeficientes del filtro para aproximarlo a la respuesta en frecuencia deseada.

Los filtros digitales suelen ser caracterizados en términos de rangos de frecuencia, tanto de la banda pasante como de la supresora. Los cuatro tipos básicos fueron nombrados anteriormente. Al ser éstos sistemas discretos, sus respuestas frecuenciales son periódicas con la frecuencia de Nyquist, ω_N, por lo que sólo se considerará el intervalo $[-\omega_N, \omega_N]$.

Con tal propósito se parte de una suposición más general, y además factible, consistente en la realización de un filtro con variación lineal del argumento respecto a la frecuencia, esto es, se tendrá que el desfase introducido será del tipo $\varphi = -\lambda\omega$, donde λ es una constante. El caso particular de desfase nulo será λ igual a cero. Luego si ante una señal de entrada cualquiera, ésta se separa en forma de sumas de sinusoides del tipo *sen(wnT),* cada una de ellas producirá una respuesta del tipo:

$$|G(\omega)|\operatorname{sen}[\omega(nT-\lambda)]$$

Por tanto cada armónico de la señal de entrada estará desfasada λ veces, de forma que se obtendrá en salida una versión no distorsionada de la señal de entrada en el rango de frecuencias dependientes de la banda pasante. A estos filtros se les llaman no dispersivos.

Partiendo de la respuesta en frecuencia de un filtro no recursivo de manera que éste tenga un orden m, y que *m* sea igual a *2N, m=2*N, se podrá poner que:

$$G(\omega)=\sum_{k=0}^{2N} g_k e^{-jk\omega T}$$
$$=e^{-jN\omega T}\left\{g_0 e^{jN\omega T}+g_1 e^{j(N-1)\omega T}+\ldots+g_N+\ldots+g_{2N}e^{-jN\omega T}\right\}$$

Si se impone las siguientes condiciones:

$$g_0 = g_{2N}$$
$$g_1 = g_{2N-1}$$
$$\vdots$$
$$g_{N-1} = g_{N+1}$$

observándose que el contenido de entre las llaves es real y que el desfase introducido por el filtro es -*N* ωT, siendo por tanto el argumento lineal con la frecuencia.
Desprendiéndose que esta característica nace de la condición de simetría par de los coeficientes *gi* alrededor de N.

4.3.2.- Ventanas

En algunos casos el orden del filtro es elevado, los rizados tanto en la banda pasante como en la supresora se mantienen, haciéndose mayores las oscilaciones en las zonas de transición entre las bandas. Además la atenuación en la banda no pasante no es cero y la transición entre las bandas no es abrupta. A este fenómeno se llama efecto de Gibbs. Así, por ejemplo, en la figura siguiente se muestra un filtro paso bajo de orden 51 con una frecuencia de corte normalizada de 0.4, evidenciando que aun siendo elevado el orden del filtro el efecto Gibbs se mantiene. Este fenómeno no desaparece con la longitud del filtro.

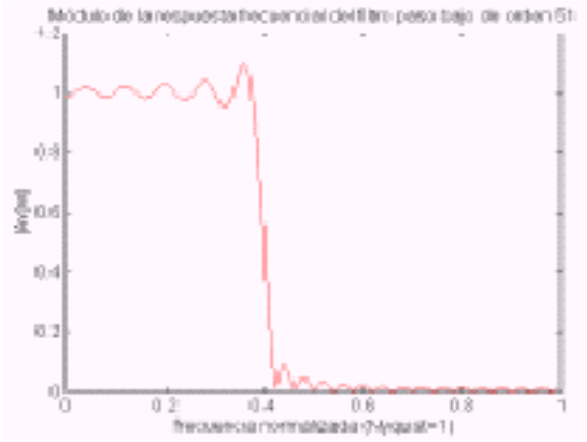


Figura 4.5

Sin embargo, la aplicación de ciertas funciones ventanas permiten aliviar este efecto no deseado. Si se quiere disminuir las oscilaciones, la respuesta del pulso infinito original debe ser multiplicado por una función ventana que no sea un pulso rectangular puro, de manera que la nueva secuencia de ponderación *gn'* quedará como:

$$g_n' = g_n w_n$$

siendo g_n definida como:

$$g_n = \frac{1}{2\pi}\int_{-\omega_N}^{\omega_N} G(\omega)e^{jn\omega T}d\omega = \frac{1}{2\pi}\int_{-\omega_C}^{\omega_C} e^{j(n-\lambda)\omega T}d\omega$$
$$= \frac{1}{\pi}\frac{\text{sen}[(n-\lambda)\omega_C T]}{(n-\lambda)T} = \frac{2\omega_c}{\omega_s}\frac{\text{sen}[(n-\lambda)\omega_C T]}{(n-\lambda)\omega_C T} \qquad n = 0,\pm1,\pm2,...$$

y w_n una función ventana definida por:

$$w_n = \begin{cases} 1 & |n| \le N \\ 0 & |n| > N \end{cases}$$

tal que N marca el número de la secuencia de transición entre la banda pasante y la supresora. Pero este tipo de ventana, obsérvese que es la aplicada en el caso del truncamiento, deriva en el efecto de Gibbs. Para evitarlo existen varios tipos de funciones ventanas, así por ejemplo se tiene:

$$w_n = \begin{cases} \alpha + (1-\alpha)\cos(\pi n / N) & |n| \le N \\ 0 & |n| > N \end{cases}$$

Donde para a = *0.5* es llamada la ventana de *Von Hann* y cuando a = *0.54* es la denominada ventana de *Hamming*.

La ventana de *Blackman* está definida por:

$$w_n = \begin{cases} 0.42 + 0.5\cos(\pi n / N) + 0.08\cos(2\pi n / N) & |n| \le N \\ 0 & |n| > N \end{cases}$$

Esta ventana reduce el rizado comparado con las dos anteriores, pero la transición entre bandas es muy suave. Un compromiso entre el rizado y la ruptura abrupta es particularmente fácil con la ventana de *Kaiser* definidas por:

$$w_n = \begin{cases} I_0(\beta)/I_0(\alpha) & |n| \le N \\ 0 & |n| > N \end{cases}$$

donde α es un parámetro y

$$\beta = \alpha\left\{1-(n/N)^2\right\}^{1/2}$$

Aquí $I_0(x)$ es una función de Bessel de orden cero definida por la serie:

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\frac{1}{k!}\left(\frac{x}{2}\right)^k\right)^2$$

A medida de tener mayor β se disminuye el rizado pero también disminuye la pendiente de transición entre la banda pasante y la supresora. En la siguiente figura se representa la respuesta frecuencial de un filtro FIR paso bajo con $\omega c / \omega s = 1/8$ y con orden de filtro de 23.

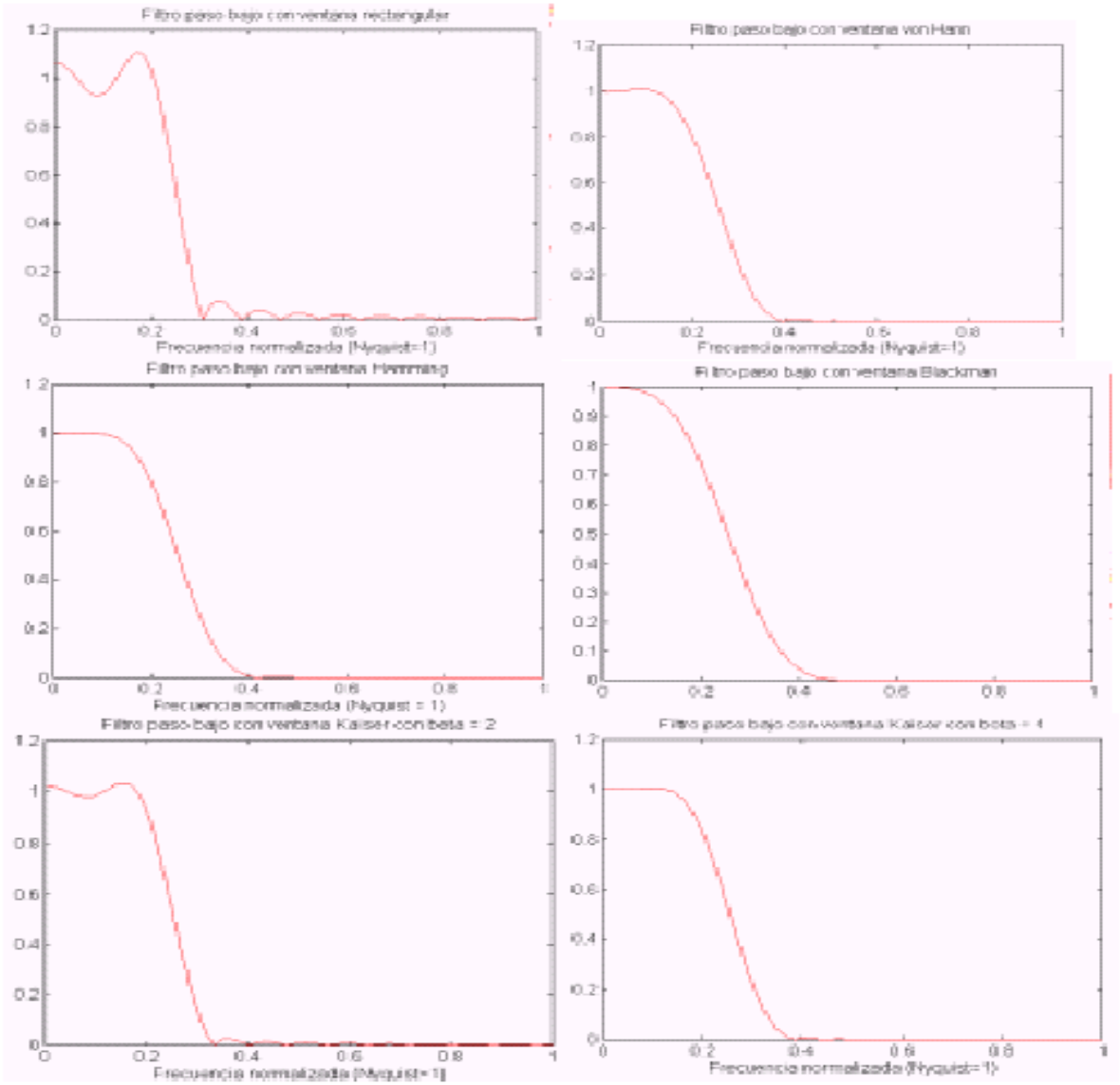


Figura 4.6

La comparativa demuestra que excepto la ventana de *von Hann* y de *Kaiser* para β igual a 2, todas las demás carecen de rizado tanto en la banda pasante como en la supresora, aunque se ve cómo la de *Kaiser* con β igual a 2 resulta la más abrupta en la transición.

Desprendiéndose que aunque en todas ellas aparece un menor rizado respecto a la ventana rectangular, existe un compromiso entre rizado y pendiente de transición. Por ello ningún filtro FIR con transformación de Fourier y aplicación de ventana es óptimo, pero resulta sencillo y económico.

El procedimiento de diseño con filtros FIR sigue los siguientes pasos:

- Se establece la respuesta de frecuencia deseada como filtro paso bajo
- Determinar la secuencia de ponderación por medio de:

$$g_n = \frac{2\omega_c}{\omega_s}\frac{\text{sen}\left[(n-\lambda)\omega_C T\right]}{(n-\lambda)\omega_C T} \qquad n = 0,\pm1,\pm2,...$$

- Se elige una función ventana y un ancho de ventana para satisfacer las especificaciones de rizado y amplitud de transición requeridas. Se asigna la ventana de acuerdo con la respuesta impulsional.
- Se desplaza la repuesta de pulso para hacerla causal.

Características importantes en el dominio de la frecuencia de algunas funciones ventanas.

Tipo de ventana	Ancho de transición del lóbulo principal	Pico de lóbulos principales (dB)
Rectangular	4π / M	-13
Bartlett	8π / M	-27
Hanning	8π / M	-32
Hamming	8π / M	-43
Blackman	12π / M	-58

Tabla 4.1

4.3.3.- Diseño de Filtros Recursivos IIR.

La ventaja de los filtros IIR respecto a los FIR es la de tener un menor orden del filtro para iguales especificaciones de diseño. Aunque la desventaja es la falta de desfase lineal introducido por el filtro, así como la necesidad de realizar estudios de estabilidad, pues ésta no está garantizada en el diseño.

Los filtros no recursivos pueden ser diseñados por varios métodos, siendo el más común el basado en las transformaciones bilineales. Este procedimiento requiere del conocimiento de la función de transferencia en el dominio *s* del filtro a diseñar. Los coeficientes del filtro en el dominio *s* son transformados a uno equivalente en el dominio z, los coeficientes de la discretización formarán el filtro IIR.

El origen de este proceder viene dado por la cantidad de experiencia acumulada en el diseño de filtros analógicos. Por tanto, todos los polinomios, tablas, métodos analíticos y gráficos para definir el filtro analógico, serán usados en el diseño de los filtros recursivos.

Si bien hay varios métodos de discretización, la mayoría de ellos tienen problemas de solapamiento en frecuencias, por realizar una relación entre el plano *s* a *z* de varias regiones del dominio *s* a una sola z. Sin embargo, la transformación bilineal realiza una transformación unívoca entre el dominio *s* a z. Esta transformación se define como:

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} = \frac{2}{T}\frac{z-1}{z+1}$$

Y su relación inversa es del tipo:

$$z = \frac{2/T+s}{2/T-s}$$

Al observar las ecuaciones anteriores se desprende que esta transformación es no lineal. De hecho, si se sustituye *s* por $j\omega_a$ y z por $e^{j\omega_d T}$ queda:

$$\omega_a = \frac{2}{T}tan\left(\frac{\omega_d T}{2}\right)$$

Siendo ω_a la frecuencia angular analógica y ω_d la frecuencia angular discreta. Por tanto si se requiere un filtro digital cuyas características en frecuencia estén definidas por $\omega_{d1}, \omega_{d2}, \ldots, \omega_{dk}$, deberá usarse un filtro analógico cuyas frecuencias sean:

$$\omega_{ai} = \frac{2}{T} tan\left(\frac{\omega_{di} T}{2}\right) \quad 1 \le i \le k$$

Siendo conocido como el *prewarping* del filtro analógico. El método de diseño de filtros IIR basados en transformaciones bilineales tiene el siguiente procedimiento:

- Definir las características del filtro digital $\omega_{d1}, \omega_{d2}, \ldots, \omega_{dk}$.
- Realizar la operación de *prewarping* de acuerdo con las ecuaciones obteniendo las frecuencias analógicas $\omega_{a1}, \omega_{a2}, \ldots, \omega_{ak}$.
- Diseñar el filtro analógico con las frecuencias definidas en el punto 2.
- Reemplazar *s* en el filtro analógico.

4.4.- Cuantificación de los coeficientes del filtro.

En la realización de filtros FIR e IIR en hardware o software en un procesador de propósito general, la precisión con la que se pueden especificarlos los coeficientes del filtro esta limitada por la longitud de la palabra del procesador o la longitud del registro donde se almacena los coeficientes. Como los coeficientes que se usan en la implementación de un filtro dado no son exactos, los polos y ceros de la función de transferencia en general serán distintos a los polos y ceros deseados, Consecuentemente, se obtiene un filtro con respuesta en frecuencia , diferente a la respuesta en frecuencia del filtro con coeficientes no cuantificados.

Por lo tanto la sensibilidad de las características de la respuesta en frecuencia del filtro a la cuantificación de los coeficientes se minimiza realizando un filtro que tenga un número grande de polos y ceros como una interconexión de secciones de filtros de segundo orden. Esto nos lleva a realizaciones en forma de cascada y paralelo para las cuales los bloques de construcción básicos son secciones de filtro de segundo orden.

4.5.- Efectos de redondeo en filtros digitales.

Los errores de cuantificación que ocurren en operaciones aritméticas realizadas en un filtro digital: la presencia de uno ovarios cuantificadores en la realización de un filtro digital resulta en un equipo no lineal con características que pueden ser significativamente diferentes al filtro lineal ideal. Por ejemplo, un filtro lineal recursivo puede exhibir oscilaciones indeseables a su salida, incluso en ausencia de señal de entrada.

Como resultado de las operaciones aritméticas de precisión finita realizadas en el filtro digital, algunos registros se pueden desbordar si el nivel de la señal de entrada se hace demasiado grande. El desbordamiento representa otra forma de distorsión no lineal indeseable en la señal deseada a la salida del filtro. Consecuentemente, se debe poner un cuidado especial en escalar la señal de entrada de forma apropiada, tanto para prevenir completamente el desbordamiento o, al menos, para minimizar su posibilidad de ocurrencia.

Los efectos no lineales debido a la aritmética de precisión finita hace extremadamente difícil analizar de forma precisa las prestaciones de un filtro digital. Para llevar a cabo un análisis de los efectos de cuantificación, se adoptan caracterizaciones estadísticas de los errores de cuantificación, que produce un modelo lineal para el filtro. Así somos capaces de cuantificar los efectos de los errores de cuantificación en la implementación de filtros digitales.

4.6.- Resumen.

En el presente capítulo se han descrito las características más importantes en el diseño de los filtros ya sean FIR e IIR.

Se dieron a conocer especificaciones tanto de sus ecuaciones de transferencia que los rigen como de su implementación digital.

Como regla general los filtros FIR se utilizan comúnmente en situaciones en donde existe la necesidad de un filtro de fase lineal. Estos requerimientos son muy utilizados en el mundo de las telecomunicaciones, en algunos casos es imprescindible no distorsionar las señales para su buena recepción o transmisión.

Los filtros IIR se utilizan generalmente en aplicaciones donde es tolerable alguna distorsión de fase.

También dentro de este capítulo se señalan los pasos que se deben seguir para poder configurar un filtro digital que cumpla todas las exigencias del medio en donde será utilizado.

Experiencias de Laboratorio

5. Introducción

El siguiente capítulo esta diseñado para comenzar a desarrollar las diferentes prácticas de laboratorio, ya sea en simulación(Simulink), como también en forma práctica(DSP), que son complementarios a los diferentes tópicos que han sido involucrados en este trabajo de titulación.
Para poder llevar a cabo esto, se darán diversos ejemplos para cada capítulo, dando a conocer en forma detallada los aspectos más relevantes de cada experiencia, como así también mencionar los parámetros a tener presente a la hora de realizar los laboratorios.
Cabe mencionar que el grado de dificultad de estas experiencias irán en forma creciente, ya que se pretende que las personas al comenzar su estudio en tratamiento digital de señales comiencen en los aspectos más básicos para poder comprender de mejor manera los fenómenos que se producen.

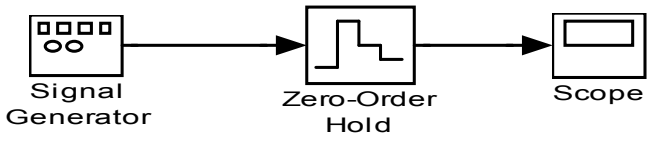
5.1.-Laboratorios para capítulo N°1: Señales y Sistemas.

Para comenzar a adentrarnos en las prácticas de laboratorio, partiremos desarrollando algunas simulaciones en Simulink para el capítulo primero que esta abocado a la descripción de los diferentes tipos de señales y sistemas que se pueden encontrar en el mundo de las telecomunicaciones.

Experiencia N°1:

Esta primera experiencia esta orientada a observar lo que ocurre al tener algún tipo de señal y pasarla a través de un bloque de muestreo, que nos permita visualizar mediante un osciloscopio, como la señal cambia su forma al realizarse lo antes mencionado, para realizar esto es necesario, seguir los siguientes pasos:

1. Abrir el software Matlab.
2. Posteriormente pinchar con el ratón la librería correspondiente a Simulink.
3. Una vez abierta pinchar para crear un nuevo archivo, en el cuál se trabajará, en este será necesario, arrastrar desde la librería de Simulink→Source→Signal Generator, este deberá ser arrastrado al archivo que esta abierto mediante el ratón.
4. Luego será necesario sacar desde la librería Simulink→Discrete→Zero Order Hold, el cual será unido con la fuente sacada anteriormente.
5. Para poder observar lo que ocurre mediante esta señal, será necesario utilizar el osciloscopio, el cuál se saca de Simulink→Sink→Scope, con lo cual es diagrama queda de la siguiente manera:



Parámetros a utilizar:

En Signal Generator:

- Wave Form: Sine
- Amplitude: 1
- Frequency: 1 hertz

En Zero Order Hold:

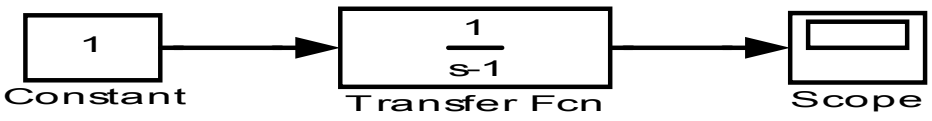
- Sample Time: 1

En el osciloscopio se deberá pinchar los ajustes de auto regulación para una mejor visualización. Aquí se puede decir que al observar la figura del osciloscopio se da la muestra que sé esta realizando de la señal de entrada que en este caso es una señal senoidal, para ver diferentes formas de onda, se puede variar el generador de onda como así también el tiempo de muestra.

Experiencia N°2.

La idea de esta segunda parte es la de poder observar la respuesta al impulso, aplicándole una función de transferencia que esta relacionada con la transformada de Laplace, por lo tanto los pasos para realizar esta experiencia son los siguientes:

1. Al igual que en la primera parte se debe tener abierto un nuevo archivo de Simulink, y arrastrar desde Simulink→Sources→Constant.
2. Luego sacar desde Simulink→Continuous→Transfer Fcn, la cual debe ser conectada con el impulso antes mencionado.
3. Por último se debe utilizar un osciloscopio el cuál se saca de Simulink→Sink→Scope, con lo cual es diagrama queda de la siguiente manera:



Parámetros a utilizar:

Para generar el impulso:

- Constant Value: 1

Para Transfer Fcn:

- Numerator:[1]
- Denominator:[1 1]

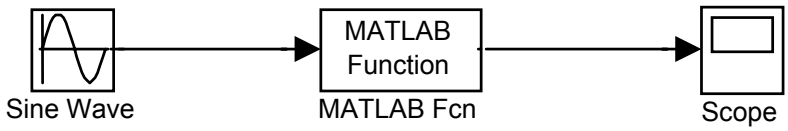
También se debe utilizar el osciloscopio, el cuál se debe ajustar a una escala adecuada para visualizar una forma de onda parecida a una respuesta exponencial. De acuerdo a esto se puede decir que a medida que se varíen los parámetros de la función de transferencia irán variando los valores de la salida pero manteniéndose la forma de la onda.

Experiencia N°3

En esta oportunidad se llevará a cabo la suma de dos tipos de ondas, las cuales serán de tipo senoidal con una exponencial y observar su comportamiento, los pasos a seguir son los siguientes:

1. Abrir un archivo en Simulink, poner en este de la librería Simulink→Source→Sine Wave.
2. De Simulink→Functions & Tables→Matlab Fcn, esta función nos permite ingresar cualquier tipo de función que este definida en Matlab.
3. Por último de Simulink→Sink→Scope, para permitir visualización.

La manera en que debe quedar el diagrama es el siguiente:



Parámetros a utilizar:

Sine wave:

- Amplitude: 1
- Frequency: 1 Rad/sec

Matlab Fcn:

- Matlab Function: exp(u)
- Output width: -1

El osciloscopio solo es necesario ajustar el parámetro autoregulación.

Aquí se debe visualizar una señal parecida a una senoidal, pero distorsionada en su parte inferior, por la composición de la señal exponencial que esta asociada con esta.

Para pode complementar este capítulo realice otras experiencias utilizando las herramientas de Simulink.

5.2.- Laboratorios para capítulo N°2: Transformadas de Fourier, Laplace y Z.

En este apartado serán tratados las diferentes herramientas que son utilizadas en el estudio de las señales como lo son las transformadas de Fourier, Laplace y Z.

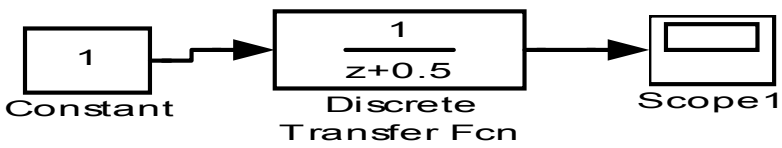
Igual que en el caso anterior, aquí será utilizado Matlab y la librería Simulink, con lo cual se podrá observar como responden las funciones al realizar alguna de las transformadas.

Experiencia N°1

Como bien es sabido una de las herramientas más utilizadas en el tratamiento de señales son las transformadas, estas suelen ser complicadas en el aspecto de su cálculo, aquí es donde se puede sacar el mayor provecho del software Matlab, ya que ofrece una plataforma amigable, tanto en su programación como así también en su visualización.

Estas experiencias están orientadas a observar, la respuesta en el tiempo que se presenta al tener algún tipo de señal de entrada para realizar esto es necesario, seguir los siguientes pasos:

1. Abrir el software Matlab.
2. Posteriormente pinchar con el ratón la librería correspondiente a Simulink.
3. Una vez abierta pinchar para crear un nuevo archivo, en el cuál se trabajará, en este será necesario, arrastrar desde la librería de Simulink→Source→constant, este deberá ser arrastrado al archivo que esta abierto mediante el ratón.
4. Luego será necesario sacar desde la librería Simulink→Discrete→Discrete transfer Fcn, el cual será unido con la fuente sacada anteriormente.
5. Para poder observar lo que ocurre mediante esta señal, será necesario utilizar el osciloscopio, el cuál se saca de Simulink→Sink→Scope, con lo cual es diagrama queda de la siguiente manera:



Parámetros a utilizar:

Constant:

- Constant value: 1

En Discrete Transfer Fcn:

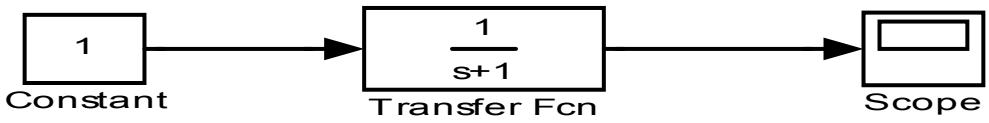
- Numerator: 1
- Denominator: [1 0.5]
- Sample time: 1

En el osciloscopio se deberá pinchar los ajustes de auto regulación para una mejor visualización. Aquí se puede decir que al observar la figura del osciloscopio muestra la respuesta en el tiepmo de la función mediante la transformada Z, hay que tener presente que esta respuesta es la unión de las transformadas, tanto de la función en función de Z y la fuente que en este caso es una señal impulso.

Experiencia N°2.

La idea de esta segunda parte es la de poder observar la respuesta al impulso, aplicándole una función de transferencia que esta relacionada con la transformada de Laplace, por lo tanto los pasos para realizar esta experiencia son los siguientes:

1. Al igual que en la primera parte se debe tener abierto un nuevo archivo de Simulink, y arrastrar desde Simulink→Sources→Constant.
4. Luego sacar desde Simulink→Continuous→Transfer Fcn, la cual debe ser conectada con el impulso antes mencionado.
5. Por último se debe utilizar un osciloscopio el cuál se saca de Simulink→Sink→Scope, con lo cual es diagrama queda de la siguiente manera:



Parámetros a utilizar:

Para generar el impulso:

- Constant Value: 1

Para Transfer Fcn:

- Numerator:[1]
- Denominator:[1 1]

También se debe utilizar el osciloscopio, el cuál se debe ajustar a una escala adecuada para visualizar una forma de onda parecida a una respuesta exponencial. De acuerdo a esto se puede decir que a medida que se varíen los parámetros de la función de transferencia irán variando los valores de la salida pero manteniéndose la forma de la onda.

Para poder complementar estas experiencias se puede utilizar el Matlab, dando una herramienta muy adecuada, el siguiente programa es utilizado para ir corroborando las respuestas de cualquier tipo de función, aquí se han usado funciones sencillas por tener una respuesta en la cual se puedan identificar claramente el tipo de respuesta en el tiempo que ofrece, pudiendo realizar cualquier tipo de simulación no importando el grado de dificultad que ofrezca las funciones a analizar.

Programa:

```
%Respuesta a:
ze=[0 1];
po=[1 0.5];
imp=[1 zeros(1,20)];
esc=ones(1,21);

ejes=[0 21 -2 2];
axis(ejes);
n=0:20;
y=filter(ze,po,imp);
yy=filter(ze,po,esc);
figure(1)
title('respuesta a IMPULSO');
stem(n,y)
grid
figure(2)
title('respuesta a ESCALON');
stem(n,yy)
grid
```

Con este programa se pueden comprobar las experiencias anteriores. Dando idénticos resultados.

También dentro de Matlab se ofrece la posibilidad de trabajar con cualquier transformada, para el caso de la transformada de Fourier, se utiliza el siguiente comando:
» help fourier, apareciendo las siguientes sentencias
--- help for sym/fourier.m ---

FOURIER Fourier integral transform.
F = FOURIER(f) is the Fourier transform of the sym scalar f
with default independent variable x. The default return is
a function of w.
If f = f(w), then FOURIER returns a function of t: F = F(t).
By definition, F(w) = int(f(x)*exp(-i*w*x),x,-inf,inf), where
the integration above proceeds with respect to x (the symbolic
variable in f as determined by FINDSYM).
F = FOURIER(f,v) makes F a function of the sym v instead of
the default w:
FOURIER(f,v) <=> F(v) = int(f(x)*exp(-i*v*x),x,-inf,inf).
FOURIER(f,u,v) makes f a function of u instead of the
default x. The integration is then with respect to u.
FOURIER(f,u,v) <=> F(v) = int(f(u)*exp(-i*v*u),u,-inf,inf).
Examples:
syms t v w x
fourier(1/t) returns i*pi*(Heaviside(-w)-Heaviside(w))
fourier(exp(-x^2),x,t) returns pi^(1/2)*exp(-1/4*t^2)
See also IFOURIER, LAPLACE, ZTRANS.

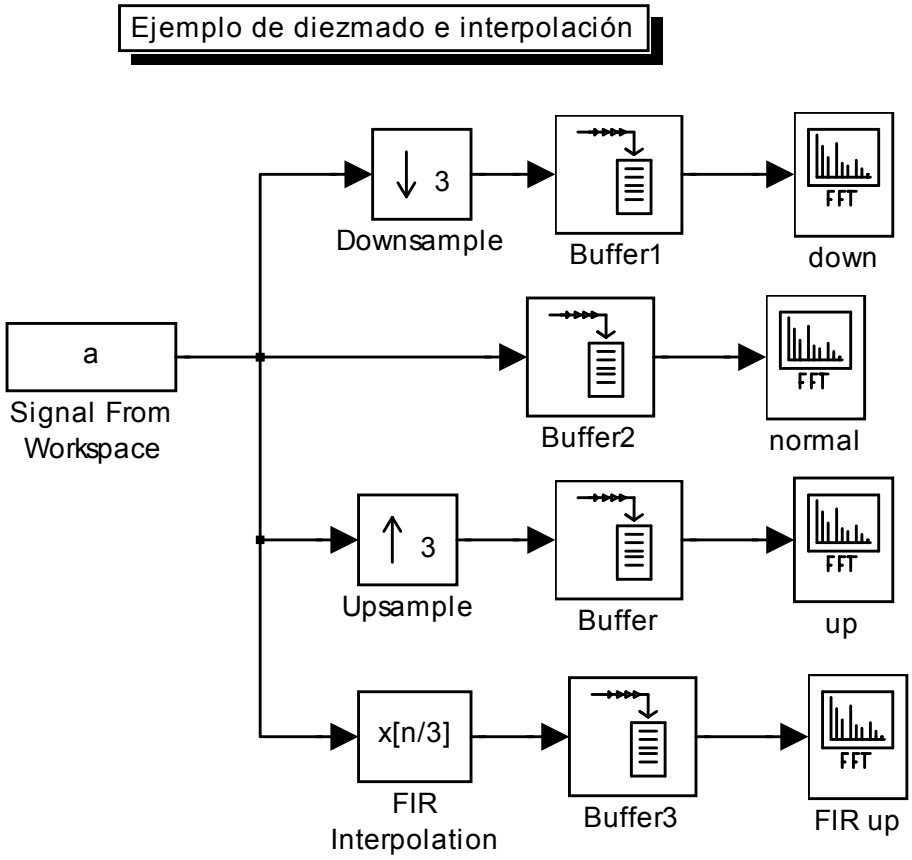
Aquí solamente es necesario escribir la función de la cual se quiere obtener ya sea su transformada de Fourier, Laplace o Z, dependiendo de las necesidades del problema.

5.3.- Laboratorios para el capítulo 3: Interpolación y Diezmado.

Experiencia N°1:

Para poder comprender mejor estos términos, realice las siguientes simulaciones de Simulink, siguiendo los siguientes pasos:

1.- Cree el siguiente diagrama:



Nota : Definir la señal de entrada "a" en el entorno MATLAB

Parámetros a utilizar:

Para Downsample:

- Downsample factor N=3
- Sample offset = 0
- Input sample time = 1

Para Upsample:

- Upsample factor N = 3
- Sample offset = 0
- Input sample time = 1

Para Fir interpolation:

- Fir filter coefficients = fir1(15,1/3)
- Interpolation factor = 3
- Input sample time = 1.

Para Buffer 1:

- Buffer size = 85
- Input sample time = 3

Para Buffer 2:

- Buffer size = 256
- Input sample time = 1

Para Buffer :

- Buffer size = 768
- Input sample time = 1/3

Para Buffer 3 :

- Buffer size = 768
- Input sample time = 1/3

Para FFT down:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 128
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Para FFT normal:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 256
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Para FFT up:

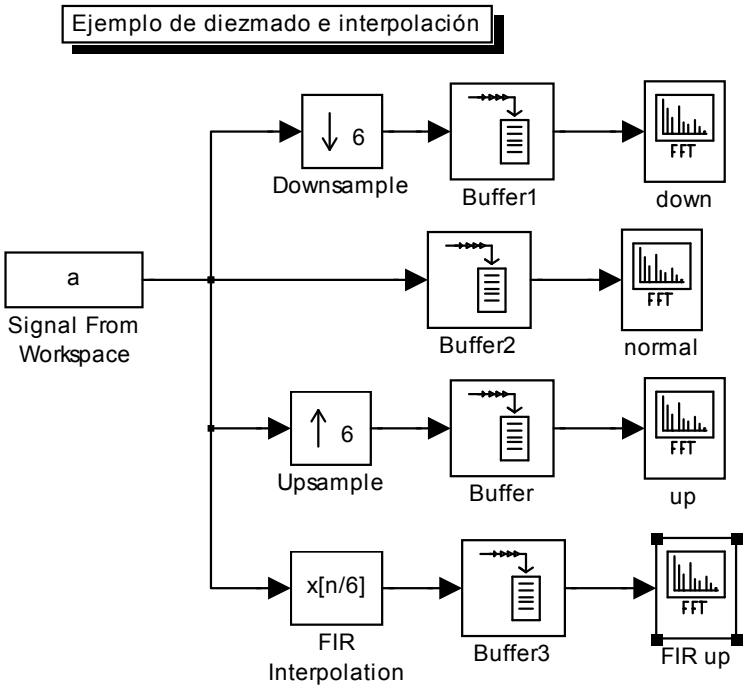
- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 1024
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Para FFT FIR up:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 1024
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Experiencia N°2:

Construya el siguiente diagrama: Aquí se aumentaran los parámetros y poder observar la diferencia entre estos dos modelos.



Nota : Definir la señal de entrada "a" en el entorno MATLAB

Parámetros a utilizar:

Para Downsample:

- Downsample factor N=6
- Sample offset = 0
- Input sample time = 1

Para Upsample:

- Upsample factor N = 6
- Sample offset = 0
- Input sample time = 1

Para Fir interpolation:

- Fir filter coefficients = fir1(30,1/6)
- Interpolation factor =6
- Input sample time = 1.

Para Buffer 1:

- Buffer size = 42
- Input sample time = 3

Para Buffer 2:

- Buffer size = 256
- Input sample time = 1

Para Buffer :

- Buffer size = 1536
- Input sample time = 1/6

Para Buffer 3 :

- Buffer size = 1536
- Input sample time = 1/6

Para FFT down:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 64
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Para FFT normal:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 256
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')

- Sample time of vector elements: 1

Para FFT up:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 2048
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Para FFT FIR up:

- Frequency: rad/sec
- Frequency range: whole
- Amplitude scaling: Magnitude.
- FFT lenght: 2048
- Y axis label: Magnitude
- Figure position: get(0,'DefaultFigurePosition')
- Sample time of vector elements: 1

Es de esperar que después de observar y analizar las diferentes formas de estos bloques se pueda tener una visión más clara de lo que son estos dos conceptos de interpolación y diezmado.

5.4.- Laboratorios para el capítulo 4: Diseño e implementación de filtros digitales.

En esta parte de las experiencias de laboratorio, se procederán a realizar de dos maneras las prácticas, la primera es de manera antigua, es decir, utilizando diversas herramientas de software, para realizar la posterior carga en el DSP, y luego utilizar el software CODE COMPOSER.

Experiencia N°1:

Diseño de un filtro digital rechaza banda entre los 2 y 6 Khz.

El primer paso a realizar en este tipo de diseño es crear un programa a través de Matlab, donde se especifique el tipo de diseño que se requiere, en este caso correspondiente a un filtro rechaza banda entre 2 y 6 Khz, este programa también tendrá la facultad de calcular los coeficientes correspondientes a este filtro, ya que estos serán utilizados más adelante.

Programa:

```
*FDSP.M
*DISEÑO DE UN FILTRO FIR Y GENERACION DE COEFICIENTES
*PARA DSK TMS320C31
clear;

N=24;
Wn=[0.1 0.3];
fs=39062;
* FILTRO RECHAZA BANDA ENTRE 2 y 6 KHZ.

h = FIR1(N,Wn,'stop',boxcar(N+1));
[H,W] = FREQZ(h,1,1024);
Hlog=10*log10(abs(H)/max(abs(H)));
zoom on
plot(W*fs/(2*pi),Hlog);
axis([ 0 fs/2 -50 2 ]);
grid;
title('Respuesta en frecuencia rechaza banda');
zoom on

!DEL COEFICIENTES.ASM
fid=fopen('coeficientes.asm','wt');
fprintf(fid,'%s\n\n','FIR_coef');

for i=1:length(h),
fprintf(fid,'%s','       .float   ');
fprintf(fid,'%15.14e ;\n',h(i));
end;
fprintf(fid,'%s\n\n','END_coef');
fclose(fid);
```

Este archivo será denominado diseño1.m , luego se hace correr a través de Matlab este programa entregando como resultado la siguiente respuesta en frecuencia:

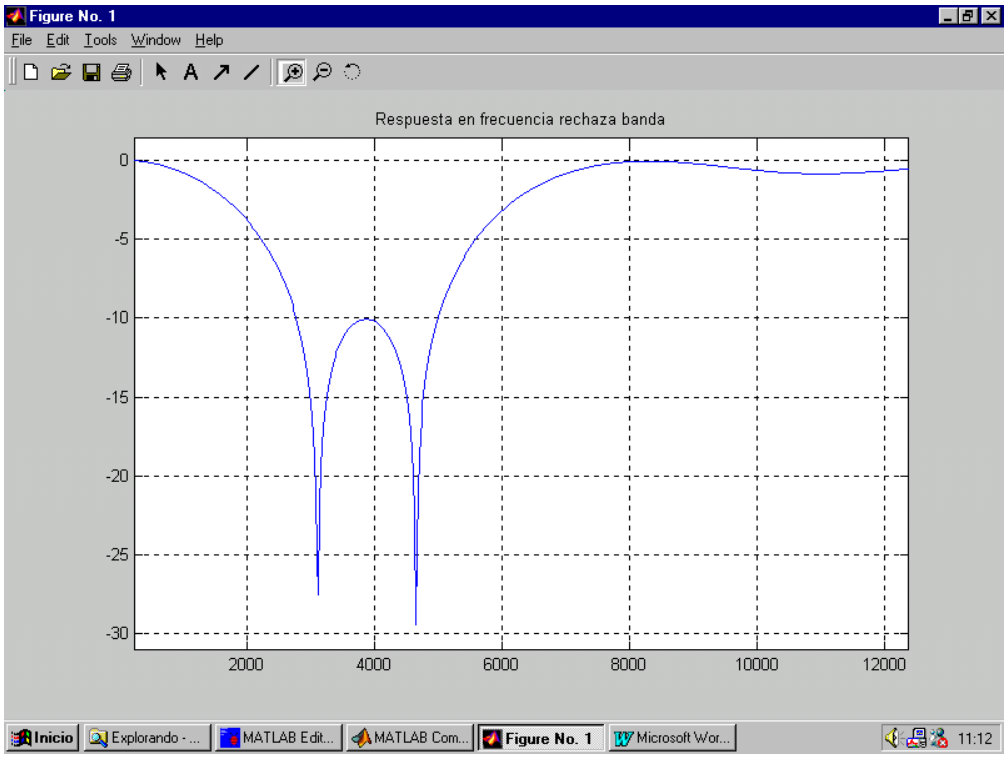


Figura 5.1

Al momento de correr este programa, se crea otro archivo denominado coeficientes.asm, en la carpeta de Matlab-Work, donde se crean los coeficientes que son referidos en este diseño.

Los coeficientes en este caso son:

```
FIR_coef
    .float   -1.15041811064660e-003 ;
    .float   -3.73751789689946e-003 ;
    .float   -7.39300075587439e-003 ;
    .float   -1.06170634712944e-002 ;
    .float   -1.08954089862425e-002 ;
    .float   -5.29819849145117e-003 ;
    .float   8.51688376307791e-003 ;
    .float   3.12179373915080e-002 ;
    .float   6.10562809852827e-002 ;
    .float   9.38323434642752e-002 ;
    .float   1.23704687898762e-001 ;
    .float   1.44661853522353e-001 ;
    .float   1.52203241374299e-001 ;
    .float   1.44661853522353e-001 ;
    .float   1.23704687898762e-001 ;
```

```
      .float   9.38323434642752e-002 ;
      .float   6.10562809852827e-002 ;
      .float   3.12179373915080e-002 ;
      .float   8.51688376307791e-003 ;
      .float   -5.29819849145117e-003 ;
      .float   -1.08954089862425e-002 ;
      .float   -1.06170634712944e-002 ;
      .float   -7.39300075587439e-003 ;
      .float   -3.73751789689946e-003 ;
      .float   -1.15041811064660e-003 ;
END_coef
```

Los coeficientes serán utilizados más adelante, ahora se debe crear un programa que será compilado en la tarjeta C31, el programa es el siguiente:

Programa:

```
Laboratorio DSP 02/02/2004
;NOTA: hacerlo en DOS con dsk3a y luego correrlos con dsk3d.
;-------------------------------------------------------
; Jaime Solar
;laboratorio1.ASM
;-------------------------------------------------------
; Define constants used by program    ;
TA       .set     8               ; AIC timing register values
TB       .set     20              ;
RA       .set     8               ;
RB       .set     20              ;
GIE      .set     0x2000          ; This bit in ST turns on interrupts
;        .include "C3XMMRS.ASM"      ;
         .start   "AICTEST",0x809802 ; Start assembling here
         .sect    "AICTEST"          ;
N         .set     32              ; Up to N taps data/coef storage
------------------------------------------------------------------------

DMA_ctrl  .set 0x808000;  DMA cntl
DMA_srce  .set 0x808004;  DMA srce address
DMA_dest  .set 0x808006;  DMA dest address
DMA_xfr   .set 0x808008;  DMA xfer counter
T0_ctrl   .set 0x808020;  TIM0 gl control
T0_count  .set 0x808024;  TIM0 count
T0_prd    .set 0x808028;  TIM0 prd
T1_ctrl   .set 0x808030;  TIM1 gl control
T1_count  .set 0x808034;  TIM1 count
T1_prd    .set 0x808038;  TIM1 prd
S0_gctrl  .set 0x808040;  SP 0 global control
S0_xctrl  .set 0x808042;  SP 0 FSX/DX/CLKX port ctl
S0_rctrl  .set 0x808043;  SP 0 FSR/DR/CLKR port ctl
```

```
S0_tctrl  .set 0x808044;  SP 0 R/X timer control
S0_tcount .set 0x808045;  SP 0 R/X timer counter
S0_tprd   .set 0x808046;  SP 0 R/X timer period
S0_xdata  .set 0x808048;  SP 0 Data transmit
S0_rdata  .set 0x80804C;  SP 0 Data receive
S1_gctrl  .set 0x808050;  SP 1 global control
S1_xctrl  .set 0x808052;  SP 1 FSX/DX/CLKX port ctl
S1_rctrl  .set 0x808053;  SP 1 FSR/DR/CLKR port ctl
S1_tctrl  .set 0x808054;  SP 1 R/X timer control
S1_tcount .set 0x808055;  SP 1 R/X timer counter
S1_tprd   .set 0x808056;  SP 1 R/X timer period
S1_xdata  .set 0x808058;  SP 1 Data transmit
S1_rdata  .set 0x80805C;  SP 1 Data receive
e_buscon  .set 0x808060;  Exp bus control
p_buscon  .set 0x808064;  Pri bus control


JJUMP    .set    0x809ff4  ;<- Base address
JXWRIT   .set    0x809ff5
JXREAD   .set    0x809ff6
JXCTXT   .set    0x809ff7
JXRUNF   .set    0x809ff8
JXSTEP   .set    0x809ff9
JXHALT   .set    0x809ffA
JW_HOST  .set    0x809ffB
JR_HOST  .set    0x809ffC
JSPARE   .set    0x809ffd

ADC_recv     .set      $-2
          .loop     N-2
          .float    0.0
          .endloop


;- - - - - - - - - - - - - - - - - - - - - - - - -
; FIR filter coefficients
;- - - - - - - - - - - - - - - - - - - - - - - - -

          .include "coeficientes.asm" ; es la línea donde llama a los coeficientes del diseño del filtro

;---------------------------------------------------------
; If more coeficients than buffers, generate an error
;---------------------------------------------------------
SZ          .set     END_coef-FIR_coef; Size of filter
          .if      N<SZ
          error -> The buffer length is less than coeficient length
          .endif
;-------------------------------------
SIZE        .word    SZ                  ; Size of filter
ADC_first   .word    ADC_recv        ;
```

```
ADC_end     .word    ADC_recv+SZ ;
ADC_last    .word    ADC_recv       ;
FIR_coefx   .word    FIR_coef       ;
;-------------------------------------
; Define some constant storage data
;-------------------------------------
A_REG       .word  (TA<<9)+(RA<<2)+0 ; A registers
B_REG       .word  (TB<<9)+(RB<<2)+2 ; B registers
C_REG       .word  00000011b        ; control
S0_gctrl_val .word  0x0E970300       ; Serial port control register values
S0_xctrl_val .word  0x00000111       ;
S0_rctrl_val .word  0x00000111       ;
;*************************************************
; Begin main code loop here
;*************************************************
main     or    GIE,ST         ; Turn on INTS
         ldi   0xF4,IE        ; Enable XINT/RINT/INT2
         b     main            ; Do it again!
;------------------------------
DAC2     push  ST              ; DAC Interrupt service routine
         push  R0            ; Save what is to be used
         pushf R0            ;
         push  R2            ;
         pushf R2            ;
         push  AR0           ;
         push  AR1           ;
         ldi   @ADC_last,AR1   ; Load pointers and circular access regsiters
         ldi   @FIR_coefx,AR0  ;
         ldi   @SIZE,BK        ;

FIR      mpyf3 *AR0++,*AR1++(1)%,R0
         ldf   0.0,R2
         ldi   @SIZE,RC
         subi  2,RC
         rptb  FIR2
         mpyf3 *AR0++,*AR1++(1)%,R0
FIR2  ||  addf3 R0,R2,R2
         addf  R2,R0

         fix   R0,R0          ;
         andn  3,R0           ;
         sti   R0,@S0_xdata    ; Output the new DAC value
         pop   AR1           ; Restore what was used
         pop   AR0           ;
         popf  R2            ;
         pop   R2            ;
         popf  R0            ;
         pop   R0            ;
         pop   ST            ;
         reti              ;
```

```
;------------------------------
ADC2    push  ST            ; Save what is to be used
        push  R3            ;
        pushf R3            ;
        push  AR0           ;
        ldi   @S0_rdata,R3  ; Get data from serial port
        lsh   16,R3         ; sign extend 16 lsb's
        ash   -16,R3        ;
        ldi   @ADC_last,AR0 ;
        float R3,R3         ; Convert to float and store for FIR data
        stf   R3,*AR0++     ;
        cmpi  @ADC_end,AR0  ; If at end of buffer, wrap back to begin
        ldige @ADC_first,AR0 ;
        sti   AR0,@ADC_last ;
        pop   AR0           ; Restore what was used
        popf  R3            ;
        pop   R3            ;
        pop   ST            ;
        reti                ;
;*****************************************************;
; The startup stub is used during initialization only ;
; and can be safely overwritten by the stack or data  ;
;*****************************************************;
        .entry  ST_STUB     ; Debugger starts here
ST_STUB ldp   T0_ctrl       ; Use kernel data page and stack
        ldi   @stack,SP
        ldi   0,R0          ; Halt TIM0 & TIM1
        sti   R0,@T0_ctrl   ;
        sti   R0,@T0_count  ; Set counts to 0
        ldi   1,R0          ; Set periods to 1
        sti   R0,@T0_prd    ;
        ldi   0x2C1,R0      ; Restart both timers
        sti   R0,@T0_ctrl   ;
        ;--------------------
        ldi   @S0_xctrl_val,R0;
        sti   R0,@S0_xctrl  ; transmit control
        ldi   @S0_rctrl_val,R0;
        sti   R0,@S0_rctrl  ; receive control
        ldi   0,R0          ;
        sti   R0,@S0_xdata  ; DXR data value
        ldi   @S0_gctrl_val,R0; Setup serial port
        sti   R0,@S0_gctrl  ; global control
;=====================================================;
; This section of code initializes the AIC            ;
;=====================================================;
AIC_INIT  LDI   0x10,IE     ; Enable only XINT interrupt
        andn  0x34,IF       ;
        ldi   0,R0          ;
        sti   R0,@S0_xdata  ;
        RPTS  0x040         ;
```

```
        LDI   2,IOF          ; XF0=0 resets AIC
        rpts  0x40           ;
        LDI   6,IOF          ; XF0=1 runs AIC
        ;---------------------
        ldi   @C_REG,R0      ; Setup control register
        call  prog_AIC       ;
        ldi   0xfffc  ,R0    ; Program the AIC to be real slow
        call  prog_AIC       ;
        ldi   0xfffc|2,R0    ;
        call  prog_AIC       ;
        ldi   @B_REG,R0      ; Bump up the Fs to final rate
        call  prog_AIC       ; (smallest divisor should be last)
        ldi   @A_REG,R0      ;
        call  prog_AIC       ;
        b     main           ; the DRR before going to the main loop
;--------------------------------
prog_AIC  ldi   @S0_xdata,R1    ; Use original DXR data during 2 ndy
        sti   R1,@S0_xdata    ;
        idle
        ldi   @S0_xdata,R1    ; Use original DXR data during 2 ndy
        or    3,R1            ; Request 2 ndy XMIT
        sti   R1,@S0_xdata    ;
        idle                  ;
        sti   R0,@S0_xdata    ; Send register value
        idle                  ;
        andn  3,R1            ;
        sti   R1,@S0_xdata    ; Leave with original safe value in DXR
        ;---------------------
        ldi   @S0_rdata,R0    ; Fix the receiver underrun by reading
        rets                  ;
stack     .word   $           ; Put stack here
;************************************************;
; Install the XINT/RINT ISR handler directly into    ;
; the vector RAM location it will be used for        ;
;************************************************;
        .start   "SP0VECTS",0x809FC5
        .sect    "SP0VECTS"
        B        DAC2        ; XINT0
        B        ADC2        ; RINT0
```

Como se puede observar dentro de este programa se llama a los coeficientes que son creados en el primer programa.

Cabe hacer notar que ahora estos dos programas ya sea el Laboratorio1.asm y coeficientes.asm, deben estar en la carpeta donde se encuentren los archivos ejecutables DSK3a y DSK3d, para realizar la carga en el DSP, de estos programas. Estos archivos están en la carpeta /DSP/DSK3_2000/DSK

Ahora se debe trabajar en ambiente DOS del computador como se explica:

Entrar en DOS aparecerá:

C:\windows>cd..

C:\>cd DSP

C:\DSP>cd dsk3_2000

C:\DSP>dsk3_2000>cd dsk3

C:\DSP>dsk3_2000>dsk3>dsk3a laboratorio1.asm; aquí se compila el programa.

C:\DSP>dsk3_2000>dsk3>dsk3d laboratorio1.dsk; aquí se carga el programa en la tarjeta, aquí se abrirá una ventana donde se tendrá que colocar el comando LOAD laboratorio1 y luego presionar F5 (Run), mediante un osciloscopio se puede observar la respuesta del filtro, tal como se tenía en la figura de la simulación mediante Matlab.

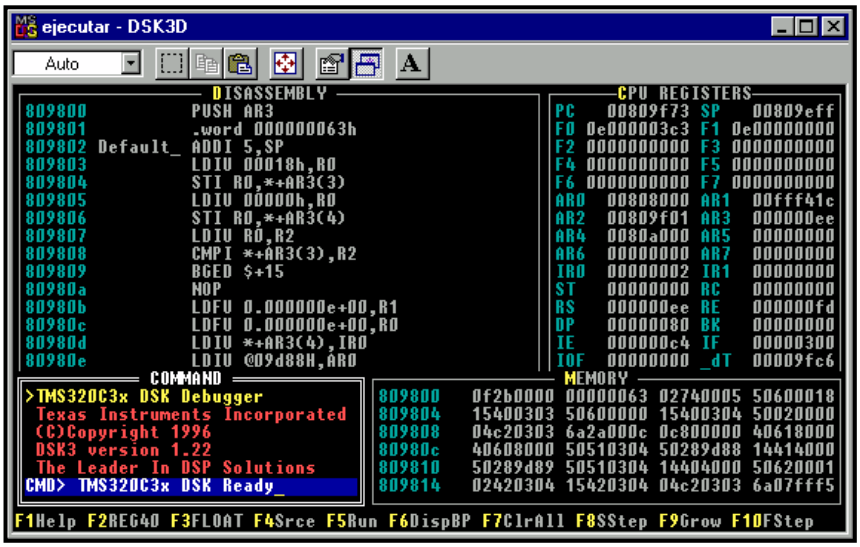


Figura 5.2

Esta es la presentación que aparece al trabajar con los comandos DSK3D para la compilación y carga de los programas en el DSP.

Experiencia N°2 : Diseño de un filtro digital pasa bajo de 3 Khz.

Aquí se trabajara de igual manera que en la práctica anterior, cambiando solamente los programas, es decir el programa para este filtro es el siguiente:

```
*FDSP.M
*DISEÑO DE UN FILTRO FIR Y GENERACION DE COEFICIENTES
*PARA DSK TMS320C31
clear;

N=24;
Wn=[0.153];
fs=39062;
* FILTRO PASA BAJO HASTA 3 KHZ.

h = FIR1(N,Wn,'DC-1',kaiser(N+1,4));
[H,W] = FREQZ(h,1,1024);
Hlog=10*log10(abs(H)/max(abs(H)));
zoom on

plot(W*fs/(2*pi),Hlog);
axis([ 0 fs/2 -50 2 ]);
grid;
title('Respuesta en Frecuencia Filtro Pasa Bajo');
zoom on

!del coeficientes2.asm
fid=fopen('coeficientes2.asm','wt');
fprintf(fid,'%s\n\n','FIR_coef');

for i=1:length(h),
fprintf(fid,'%s','       .float   ');
fprintf(fid,'%15.14e ;\n',h(i));
end;

fprintf(fid,'%s\n\n','END_coef');
fclose(fid);
```

El nombre de este archivo es diseño2.m, al igual que el anterior, también creara un archivo con los coeficientes del filtro diseñado, este archivo, el de los coeficientes, es denominado coeficientes2.asm, la respuesta de este filtro es como se muestra en la siguiente figura:

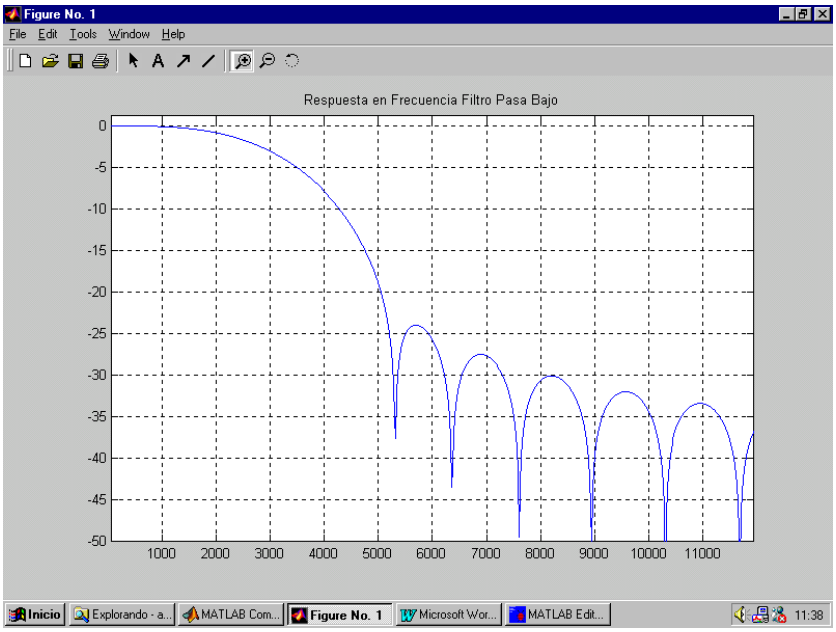


Figura 5.3

Siguiendo con la rutina antes mencionada, (experiencia anterior), se debe tener el siguiente programa, donde se utilizara el archivo coeficientes2.asm, este programa será nombrado laboratorio2.asm, que es el mismo que el anterior, con la salvedad de que se debe cambiar dentro del programa donde llama a los coeficientes por el nombre coeficientes2.asm.

Los dos programas el laboratorio2.asm y el coeficientes2 fueron llevados a la carpeta donde se encuentran los archivos ejecutables dsk3a y dsk3d, para realizar la carga en el DSP, se deben seguir los mismos pasos nombrados en la experiencia anterior, solamente cambiando los nombres que están asociados a este diseño.

Experiencia N°3: diseño de un filtro rechaza banda entre 2 y 4 Khz, a través de Code Composer. Para realizar esta experiencia solo es necesario tener algún programa que cumpla con la condición de diseño del filtro, el cual debe ser con una extensión .asm o .c, en esta oportunidad será .c, como lo muestra el programa siguiente:

```
/*=============================================================================
===
  VALDIVIA, 05 DE MARZO DE 2004
  Jaime Solar

  FILTRB.C

  PROGRAMA QUE TOMA UNA SEÑAL EN LA ENTRADA DE LA TARJETA A PARTIR
DEL GENERADOR Y LA SACA POR LA SALIDA TRAS PASAR POR UN FILTRO
RECHAZA BANDA  DE FRECUENCIA ENTRE 2 KHz Y 4 KHz APROX.
  ESTE PROGRAMA FUNCIONA BIEN HASTA UNA SEÑAL DE ENTRADA DE
APROXIMADAMENTE 8 KHz.

=============================================================================
===*/

 #include "C3MMR.H"
/*============================================================================
  Aqui comienza la aplicacion del codigo.
  Se define una serie de constantes que se usan en varias funciones
============================================================================*/
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
#define  TIM0_prd  2          /* TIM0 es el reloj de referencia del AIC*/
#define  TA        6          /* programacion del DAC                  */
#define  TB        25         /*                                       */
#define  RA        10         /* programacion del ADC                  */
#define  RB        15         /*                                       */
#define  A_REG   ((TA<<9)+(RA<<2)+0) /* intervalo de trabajo de los      */
#define  B_REG   ((TB<<9)+(RB<<2)+2) /* registros TA y TB del AIC        */
#define  C_REG   0x3              /*                                    */
#define  S0gctrl 0x0E973300       /*                                    */
#define  S0xctrl 0x00000111       /*                                    */
#define  S0rctrl 0x00000111       /*                                    */
#define  bigval  000010000h       /* usado en caso de overflow         */
/*============================================================================
  Aqui se definen las funciones que se van a utilizar en el resto del  programa
 ============================================================================*/
```

```
void ST_STUB (void);
float Input   (void);
void  Output  (float);
void  prog_AIC(int );
void  AIC_INIT(void);
float fir_1 (float, int);
/*************************************************************************/

/*=========================================================================
  El bucle principal consiste en esperar a recibir una nueva muestra del ADC. cuando ocurre una
interrupción interan el nuevo dato  se carga  en el buffer de espera seguido por la SFFT y las
funcione sde salida.
  la salida se obtiene por un osciloscopio a través de las funciones que llevas los datos al bus
externo simulando una aplicación en  tiempo real.
=========================================================================
=/*

 /*definición de las variables de entrada y salida de datos del filtro*/

      float coef1[83];          /* coeficientes del filtro         */
      float datos1[83];         /* datos de entrada del porgrama    */
      float datos2[83];
/*************************************************************************/
 /*comienza el programa principal*/

  void main(void)
  {
      float f,y;
      int orden =24;       /*orden del filtro paso bajo*/
      int cont,i;
   for (cont=0;cont<orden;++cont)
      {
      datos1[cont]=0.0;      /*inicializacion a 0 de los datos de salida*/
      datos2[cont]=0.0;
      }
    /*COEFICIENTES DEL FILTRO RECHAZABANDA ENTRE 2KHz y 4 KHz*/

      coef1[0]=   -1.46321423849823e-002 ;
      coef1[1]=    -2.75946193368967e-002 ;
      coef1[2]=  -2.98724249944976e-002 ;
      coef1[3]=    -1.32132303306722e-002 ;
      coef1[4]=    2.19482135774734e-002 ;
      coef1[5]=    6.47004196683361e-002 ;
      coef1[6]=   9.71379824050879e-002 ;
      coef1[7]=  1.01990838557187e-001;
```

```
    coef1[8]=   7.10259111213878e-002 ;
    coef1[9]=    1.01476839988125e-002  ;
    coef1[10]=  -6.15692706625942e-002 ;
    coef1[11]=  -1.18919072976051e-001 ;
    coef1[12]=   7.97694422714819e-001 ;
    coef1[13]=  -1.18919072976051e-001 ;
    coef1[14]= -6.15692706625942e-002 ;
    coef1[15]=  1.01476839988125e-002 ;
    coef1[16]= 7.10259111213878e-002 ;
    coef1[17]=  1.01990838557187e-001 ;
    coef1[18]=    9.71379824050879e-002 ;
    coef1[19]=  6.47004196683361e-002 ;
    coef1[20]=   2.19482135774734e-002 ;
    coef1[21]= -1.32132303306722e-002;
    coef1[22]=   -2.98724249944976e-002 ;
    coef1[23]=   -2.75946193368967e-002  ;
    coef1[24]=   -1.46321423849823e-002 ;



  ST_STUB();                /*   llamada a la función ST_STUB          */
  asm("  ldi   0E4h,IE     "); /*   habilita XINT/RINT/INT2              */
  asm("  idle              "); /*   espera hasta recibir una interrupción */
  f = (float)*S0_rdata;       /*   la primera interrupción ocurre justo  */
  *S0_xdata = 0;              /*   después de la inicialización de AIC   */

  for(;;)
  {
   asm("  idle    ");         /*   Espera hasta recibir una interrupción */
   f = Input();               /*   Pone la muestra del ADC en el buffer  */
   y=fir_1(f,orden);          /*   hace el filtro de la señal            */
   Output(y);                 /*   saca el resultado por el osciloscopio */
  }
 }
/*=============================================================================
=====
 función que filtra la entrada con el filtro fir
=============================================================================*/
 float fir_1 (float muestra, int orden)

     {
     float *pcoef = &coef1[0];
     float *pdatos = &datos1[0];
     float *pdatos2;
     float out = 0.0;
     int i=0;
```

```
        *pdatos = muestra;

        /* Filtrado */

        for (i = 0; i < orden; i++) out += *pdatos++ * *pcoef++;
        /* Actualizacion de la memoria de muestras */
        --orden;
        pdatos = &datos1[orden-1];
        pdatos2 = &datos1[orden];
        for (i = 0; i < orden; i++) {
                *pdatos2 = *pdatos--;
                --pdatos2;
        }

        return(out);
}

/*=====================================================================
  Función para leer el dato del ADC y ponerlo en el buffer
 =====================================================================*/
float Input(void)
{
  int   x;
  float f;
  x = *S0_rdata;     /* toma el dato del ADC                */
  x = x >> 16;       /* extensión del signo antes de muestrear*/
  f = x;             /* Convierte el dato del ADC a float     */
  return f;
}

/*=====================================================================
 Función que saca el dato de salida a través del osciloscopio
 =====================================================================*/

void Output(float f)
{
  int x;
  x = f;
  x &= 0xFFFC;
  *S0_xdata = x;
}
/*=====================================================================
  Esta función se usa solo durante la inicialización
 =====================================================================*/
```

```
void ST_STUB(void)
{
  *T0_ctrl = 0;        /* Halt TIM0                  */
  *T0_count= 0;        /* inicializa contador a 0    */
  *T0_prd  = TIM0_prd; /* inicializa el periodo      */
  *T0_ctrl = 0x2C1;    /* comienzo de tiempo de control */
  /* - - - - - - - - - - - - - - - - - - - - - */
  *S0_xctrl = S0xctrl; /* control de transmisión     */
  *S0_rctrl = S0rctrl; /* control de recepción       */
  *S0_xdata =      0; /* valor del dato      DXR    */
  *S0_gctrl = S0gctrl; /* control general            */
   AIC_INIT();
}
/*================================================
  esta función inicializa el AIC
 ================================================*/
void AIC_INIT(void)
{
 asm("  andn  034h,IF  ");
 asm("  ldi   004h,IE  "); /* habilita solo INT2      */
 *S0_xdata = 0;
 asm(" rpts  0040h     ");
 asm(" ldi   2,IOF     "); /* XF0=0 resetea AIC       */
 asm(" ldi   6,IOF     "); /* XF0=1 arranca AIC       */
 asm(" rpts  040h      ");
 asm(" nop             ");
 asm("  andn  034h,IF  ");
 asm("  ldi   014h,IE  "); /* habilita solo da interrupción XINT */
 /*- - - - - - - - - - - -*/
 prog_AIC(C_REG   );     /* registro de control del programa    */
 prog_AIC(0xFFFC  );     /* Programa el AIC para tiempo real    */
 prog_AIC(0xFFFC|2);     /* Programa el AIC para tiempo real    */
 prog_AIC(B_REG   );     /* frecuencia de muestreo Fs           */
 prog_AIC(A_REG   );     /* divisores pequeños se envían primero */
 asm("  or  080h,ST");   /* usa el modo overflow para saturación */
}
/*=========================================================================

  Esta función se usa para transmitir nuevas configuraciones de tiempo al AIC
  ========================================================================*/

void prog_AIC(int xmit2)
{
 int x;
 *S0_xdata =     0; asm(" idle "); /* pretransmite una muestra segura */
```

```
 *S0_xdata =     3; asm(" idle "); /* espera respuesta              */
 *S0_xdata = xmit2; asm(" idle "); /* envia el valor de registro de programa*/
 *S0_xdata =     0; asm(" idle "); /* abandona tras la muestra fiable */
 x = *S0_rdata;
}

/*=========================================================================
  Instalación de los vectores XINT/RINT ISR

 =========================================================================*/


 asm(" .sect  \"SP0VECTS\"");
 asm(" reti           ");  /* XINT0              */
 asm(" reti           ");  /* RINT0              */
```

Como se ve dentro de este programa es necesario pegar los coeficientes del filtro a ser diseñado, los cuales se pueden obtener como se dijo en las experiencias anteriores, es decir, mediante un programa en Matlab.

Para cargar y poder trabajar a través de Code Composer, diríjase al anexo C, donde se detallan todos los pasos a seguir para llevar a cabo la manipulación de este software.

5.5.- Resumen.

Este capítulo fue desarrollado con la intención de poder complementar los capítulos posteriores, realizando experiencias de laboratorio, ya sean de forma práctica como en simulación de los diversos tópicos involucrados en este trabajo de titulación.

Para poder llevar a cabo esto fue necesario utilizar el software Matlab y su librería Simulink, estas dos herramientas de análisis nos permitieron desarrollar las simulaciones correspondientes ya sea en las señales como en los diseños de filtros digitales.

También para llevar a la práctica nuestras experiencias se utilizo las tarjetas de procesamiento digital de señales DSP, que se encuentran el laboratorio de nuestro instituto, con los cuales se puede trabajar a nivel de DOS o utilizando el software Code Composer, el cual hace más amigable el desarrollo de diferentes prácticas para análisis y su posterior comprensión de los diferentes fenómenos que se producen.

El Code Composer es una herramienta realmente efectiva, la cual nos ofrece, como característica principal, la administración de las herramientas necesarias para el desarrollo de proyectos destinados al trabajo con DSP: Compilación, ensamblado y linkado de un programa, en una sola aplicación, lo cual antes debía hacerse por separado con distintos programas cada uno, de diversa y engorrosa utilización, la cual aumentaba el tiempo de trabajo y disminuía la eficiencia de ello.

Code Composer, además ofrece grandes posibilidades al usuario de realizar aplicaciones sencillas con sólo crear un nuevo proyecto, el que se programa en lenguaje C o Assembler, dando con ello también, una gran ventaja para el usuario, ya que son lenguajes conocidos y de mediana complejidad.

CONCLUSIONES

En el desafío de poder comenzar el estudio de tratamiento digital de señales, fue necesario realizar una exhaustiva investigación con respecto a las diferentes materias que se iban a involucrar en esta área para entregar una herramienta de docencia que sea útil para el análisis y comprensión del tema.

Al concluir este trabajo se puede mencionar que para comenzar el estudio del tratamiento digital de señales es necesario partir con los conceptos de señales y sistemas, como así también dominar de cierta manera algunas herramientas de análisis matemático ya que estos son imprescindibles a la hora de realizar las primeras nociones de este tema.

La utilización de los diversos software y hardware que se encuentran disponibles en nuestra universidad constituyen una muy interesante herramienta para la modelación de filtros digitales.

Para el complemento del aprendizaje del tratamiento digital de señales, la creación de una página Web con el contenido de este trabajo de titulación, ofrece al usuario una forma más amena e interactiva para ir desarrollando este tema, en comparación con un texto, el cual a parte de ser más complicado en el transporte no siempre se encuentra disponible, como el de tener la información en la Internet.

Finalmente quisiera agregar que en lo personal este trabajo me ayudó a crecer no sólo en conocer un área de estudio nueva, sino además a expandir mis conocimientos en lo concerniente de la programación y el diseño Web, dos cosas de importancia en el mundo actual y que además complementan mi formación profesional, dándome así una herramienta que me ayudarán a enfrentar el mundo laboral actual.

BIBLIOGRAFIA

Libros y apuntes:

[1] TMS320C3x User Guide, Texas Instruments, INC, Dallas Texas, 1995.

[2] Jhon Proakis, Dimitris Manolakis, "Tratamiento Digital de Señales", Tercera edición.

[3] Allan Oppenheim, Alan Willsky, "Señales y Sistemas", Segunda edición.

[4] Mitra, J kaiser, "Handbook for Digital Signal Processing, 1993.

[5] S Smith, " The scientist and Engineers Guide to digital Signal Procering", Edición de California, EEUU 1997.

[6] B.P Lathi, "Sistemas de Comunicación", Editorial Interamenricana S.A 1986.

[7] Néstor Fierro, Juan Ortega, "Desarrollo de una Interfaz de Comunicación entre una Tarjeta de Procesado de Señal y Matlab", 2002.

[9] Néstor Fierro, " Manual de Code Composer", 2001.

[10] Edminister Joseph, " Circuitos Eléctricos", 2° Edición, Serie Schaum, 1992.

[11] Ferrel Stremler, " Sistemas de Comunicación, Fondo edición Interamericano.

[12] Marven, Ewer, " A Simple Approach to Digital Signal Processing", Edición de California, EEUU, 1997.

[13] Cristián Yantani. " Procesado de Señales", 1999.

Direcciones Web:

[1] www.ti.com

[2] www.esi.unav.es/asignaturas/tratamiento

[3] www.utfsm.elo.cl

[4] www.monografias/trabajos/matlab.com

[5] www.monografias/trabajos/codecomposer.com

Procesador Digital de Señales

Los Procesadores Digitales de Señales están diseñados para realizar operaciones matemáticas. Las computadoras diseñadas para negocios y otras aplicaciones generales no están optimizados para ejecutar algoritmos como filtrado digital y análisis de Fourier. Los Procesadores Digitales de Señales son microprocesadores diseñados específicamente para realizar tareas de Procesamiento Digital de Señales. Estos dispositivos han tenido un exponencial crecimiento en la última década, ya que los encontramos en una gama de aplicaciones que van desde: teléfonos celulares hasta instrumentos científicos avanzados.
Él termino DSP significa Procesadores Digitales de Señales para los ingenieros de hardware, y para los desarrolladores de algoritmos DSP significa Digital Signal Processing (PDS).

Arquitectura de los DSP

Los DSP son sistemas mínimos, que permiten el tratamiento digital de las señales. Estos sistemas mínimos, constan con una unidad central de procesos (CPU), como también, unidad de memoria, almacenamiento (RAM, Y ROM) y una o varias unidades que corresponderán a nuestros periféricos de entrada y salida de datos.
Una de las principales características de los DSP, se podrían resumir en lo siguiente:

- ALU flexible y rápida: las operaciones estandard (bit a bit, suma, etc.) multiplicaciones, multiplicaciones con acumulación y desplazamientos arbitrarios deben ser realizadas en un ciclo de reloj.
- Capacidad de ejecutar una multiplicación - acumulación en un único ciclo de instrucción (operación MAC), algunos DSP´s de alto desempeño tienen dos unidades multiplicadoras que les permite la ejecución de dos MAC en paralelo por ciclo.
- Modos de dirección especializado, por ejemplo punteros de direccionamiento con pre y post modificación, direccionamiento circular (implementación de filtros digitales) y direccionamiento por bit invertido (implementación de FFT).

- Control de ejecución especializado. Usualmente los DSP proveen instrucciones de lazo que permiten lazos estables de repetición sin gastar ciclos de instrucción adicionales en actualizar y testear el contador de lazo o para saltar de regreso al inicio del lazo.

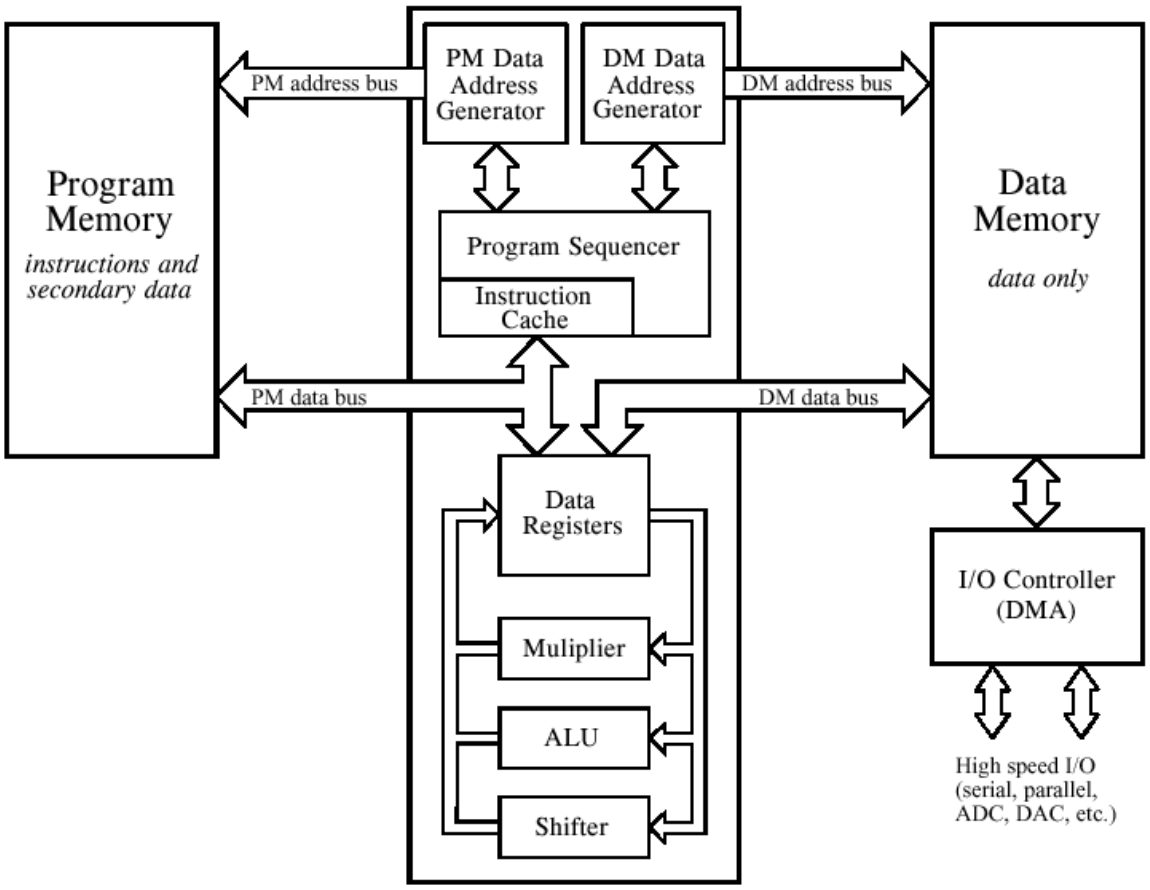
- Los procesadores DSP son conocidos por sus sets de instrucciones irregulares que generalmente permiten que muchas operaciones sean codificadas en una sola instrucción. Por ejemplo un procesador que usa instrucciones de 32 bits puede codificar dos sumas, dos multiplicaciones y cuatro movimientos de datos de 16 bits en una única instrucción. En general los sets de instrucciones de los DSP's permiten movimientos de datos en paralelo con operaciones aritméticas.

- Rango dinámico extendido para acumulaciones, mas acumulación de tal forma que no ocurra overflow. (desbordamiento)
- Carga de operandos con un ciclo de reloj, lo que significa un direccionamiento flexible para varias memorias de datos.
- No se requieren de instrucciones adicionales (condiciones, o comparaciones), para bucles y saltos.

El objetivo de los DSP (a diferencia de los sistemas de arquitectura tradicional) es el procesado de señales dinámicas, las cuales requieren, por sus características, de un elevado desempeño y performance de hardware que normalmente los sistemas de propósito general no son capaces de cumplir pues los requerimientos exigen guardar una relación con el tiempo real. Al existir estos requerimientos implica que necesariamente, que en un sistema normal exista una aglomeración de datos al ejecutar algoritmos de procesamiento de señales, que por lo general son de gran complejidad e implica una gran transferencia de información desde y hacia la memoria. Tal es el caso en que se carga un filtro digital con ciertas características o coeficientes y es procesado con una señal de entrada, la entrada de datos en este caso y la salida del filtro que prácticamente tiene que ser instantáneo

La mayoría de los algoritmos aplicados en el procesamiento digital consisten en bloques de sucesivas multiplicaciones o loops, en las cuales se relacionan señales de entrada con señales de salida en un tiempo real, es decir el tiempo de captación tiene que ser lo mas optimo posible, de tal manera que la respuesta no interfiera con la fluidez de la señal.

Es por este motivo que los DSP se basan en una arquitectura de tipo Harvard, (o mejor dicho super Harvard, por adicionar una memoria cache al DSP), que incorpora dos buses separados de datos, permitiendo un mejor manejo de instrucciones e interrupciones, con un alto grado de paralelismo en la ejecución de programas. A modo de ejemplo la CPU realiza la operación de lectura, escritura de datos y simultáneamente ya esta buscado la siguiente instrucción. En la siguiente figura podemos observar la estructura típica de un DSP



Principal Elementos de la CPU en los DSP

En esta unidad se definirán los elementos de memoria característicos de cualquier sistema mínimo, pero orientados específicamente a DSP.

Unidad aritmética: Es la unidad donde se ejecutan las diferentes operaciones aritméticas y lógicas de datos. Generalmente trabaja utilizando palabras de 16 bits. El tipo de operación que esta unidad realice dependerá exclusivamente de la unidad de control.

La AU, en su estructura esta compuesta por:

- Un multiplicador, que se encarga de la multiplicación en paralelo.
- Por una unidad aritmética lógica ALU, encargada de realizar operaciones de adición, substracción, transporte, y operaciones lógicas.
- Registros de propósito general GRP para el almacenamiento temporal de los datos.

Unidad de memoria: en esta unidad se guardará el código binario que representará las instrucciones guardadas en código binario, así como los datos que serán procesados por el programa todos provenientes de dispositivos de memoria.

La unidad de memoria se compone de los siguientes elementos de memoria:

- La memoria de coeficientes; que almacena coeficientes, tal es el caso de los coeficientes de filtro, o tablas de senos.
- Memoria de programa (PM) que almacena instrucciones.
- Memoria de datos (DM), que almacena datos.

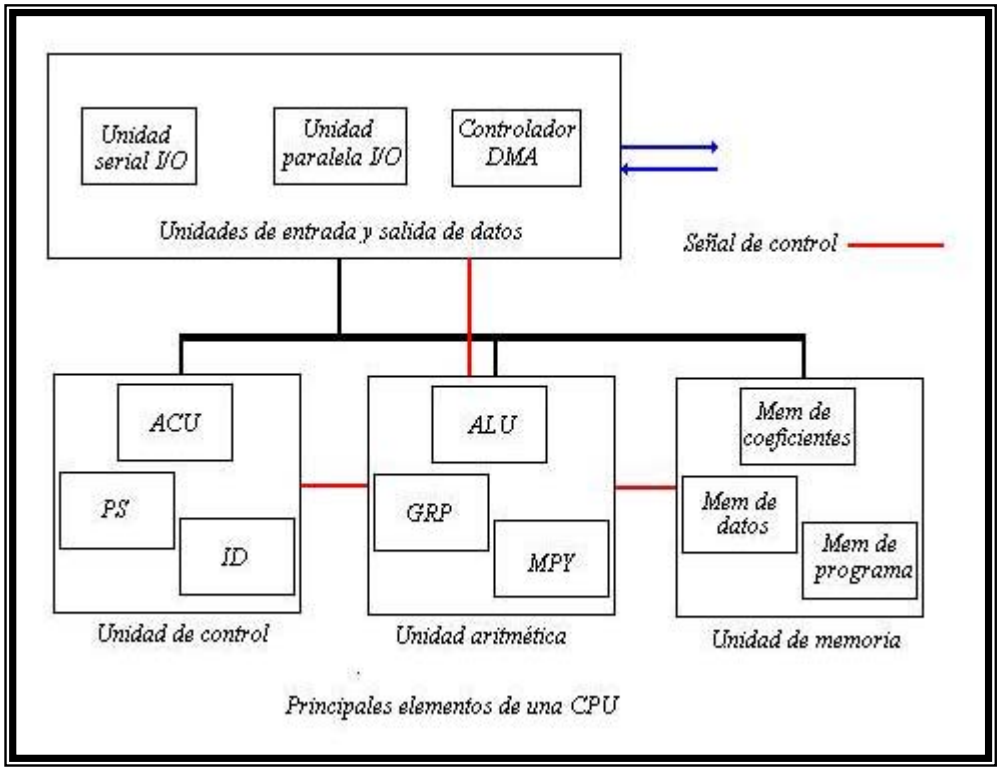
Unidad de entrada y salida: es la que tiene por función almacenar, recibir, y entregar datos, desde y hacia los dispositivos tales como conversores A/D y D/A, timers y punteros de comunicación. Se compone de las unidades:

- Unidad serial I/O
- Unidad paralela I/O
- Controlador de acceso directo a memoria (DMA)

Unidad de control: Es la encargada de gobernar y sincronizar todas las unidades a través de señales de temporización y control.

Esta unidad consta de los siguientes elementos:

- El secuenciador de programa (PS), que genera la dirección de la próxima instrucción a ser ejecutada.
- Decodificador de instrucciones (ID), traduce o decodifica la instrucción que sé esta ejecutando, para generar las señales de control según sea el caso.
- La unidad de cálculo de direcciones (ACU), encargada de computar direcciones, a las que hace referencia la instrucción, con respecto a operandos y datos.



El DSP C31

La firma Texas Instruments es la proveedora de módulos de sistemas para desarrollo DSK, especialmente en nuestra Universidad contamos con los módulos TMS 320C26, TMS 320 C31, Y TMS 320C5000.

Específicamente se dará una explicación mas detallada del funcionamiento de la tarjeta TMS 320 C31. En las páginas anteriores se da una pequeña descripción de los DSP en términos generales, pues el principio de funcionamiento de estos es común en prácticamente todos los procesadores, por lo que a continuación explicare características más específicas de la tarjeta como también aplicaciones y capacidades.

A manera de resumen el C31 consta de una unidad central de proceso, que contiene, 28 registros, que pueden ser operados por la ALU y un multiplicador, estos registros tienen por función soportar el direccionamiento, operaciones en punto flotante o fijo, manejo de pila, estado del procesador y bloque de repetición de instrucciones.

La DSK contiene, además del microprocesador TMS320C31, un circuito de interfaz analógica (AIC), el TI TLC32040. Este elemento constituye un completo sistema de entrada / salida chip CMOS monolítico. El dispositivo se conecta directamente al puerto serie del TMS320C31. La referencia de reloj y la señal de reset le son suministrada por el ´C31. Para mayor flexibilidad y expansibilidad se puede prescindir del uso del AIC cuando se utiliza un CODEC (codificador-decodificador). El AIC integra un filtro de entrada anti-aliasing paso-banda de condensador conmutado, cuatro modos de puerto serie compatibles con el microprocesador, un conversor digital-analógico de 14 bits de resolución y un filtro de reconstrucción de salida paso-bajo, de condensador conmutado. Ofrece numerosas combinaciones de frecuencias de entrada del reloj patrón y frecuencias de conversión-muestreo que pueden cambiarse mediante el control digital del procesador. El TLC32040 soporta velocidades de muestreo variables, de conversión analógica-digital y digital-analógico, que llegan a las 20.000 muestras por segundo. La tarjeta del DSP se conecta al PC mediante una interfaz de puerto paralelo estándar o bidireccional.

Estas características convierten a la tarjeta en un potente instrumento para desarrollar aplicaciones de procesado digital de señales.

El Hardware del TMS320C31

El procesador del **TMS320C31** es la columna vertebral del DSK C31, este procesador es un DSP de 32 bits, fabricado con tecnología CMOS de 1μm y con una matriz de 132 pines con cálculos de operación punto flotante.

Su arquitectura es super Harvard lo cual le permite ejecución de operaciones en paralelo, aumentando con esto en forma notoria su velocidad de procesado, siendo una buena opción para tratar señales en tiempo real, como es el caso de los filtros, que es la aplicación y uso que se dará al procesador en este caso.

Su velocidad de reloj es de: 50 M hertz

Tiempo de ejecución en un ciclo de instrucción es de: 60 n segundos

Arquitectura del TMS320C31

La arquitectura del TMS C31 responde a altos requerimientos en el procesado de señales y calculo de algoritmos de alto nivel para lo que son Telecomunicaciones.

Este procesador se compone de 5 cuerpos principales:

- La unidad central de proceso (CPU)
- Un temporizador (timer)
- Un puerto serie
- Un controlador DMA
- Tres bloque de memoria que están insertos en el dispositivo (on chip)

La unidad central de proceso

El C31 tiene una CPU basada en registros en los que podemos destacar los siguientes componentes:

- Multiplicador de enteros y punto flotante, que realiza operaciones de multiplicaciones de datos enteros en 24 bits, con un resultado de 32 bits y de punto flotante de 32 bits, en un solo ciclo de reloj, con resultado valor de 40 bits.
- Unidad aritmética lógica (ALU), que realiza operaciones aritméticas generales.
- Buses internos, que tienen por función transportar datos, y operandos dentro de la CPU.
- Unidad aritmética de registros auxiliares(ARAU), que genera el direccionamiento de los datos.

El C31 consta con 28 registros, en un archivo de registro de multipuerto, el que esta estrechamente relacionado con la CPU. El contador de programa (PC) no se incluye en los 28 registros. Todos los registros pueden ser operados por el ALU, y pueden ser usados como registros de 32 bits de propósito general, pero hay que destacar que estos registros también pueden cumplir otras funciones muy especiales como por ejemplo; los ocho registros de precisión extendida, que son especificados para mantener operaciones de punto.

Los ocho registros auxiliares soportan una variedad de modos de direccionamiento indirecto.

Nombre Registro	Función
PC	Registro contador de programa
RC	Registro contador de repetición
RE	Registro de dirección final a repetir
RS	Registro de flags de entrada salida I/O
IOF	Registro de flags de interrupción
IF	Registro de habilitación de interrupciones
IE	Registro de status
ST	Puntero de la fila (stack pointer)
SP	Registro de tamaño del bloque
SK	Registros de tamaño de bloque
IR1	Registro índice 1
IR0	Registro índice 0
DP	Puntero de página de datos
AR0, AR1, ...AR7	Registros auxiliares
R0, R1,..... R7	Registros de precisión extendida

Registro	Descripción y uso
PC	Registro contador de programa, que contiene la dirección de la próxima instrucción a ejecutarse
RS	Registro que especifica la dirección inicial final, del bloque a repetir
RE	Registro que especifica la dirección inicial final, del bloque a repetir
RC	Registro que especifica el número de veces que se repite un determinado bloque
DP	Registro que se utiliza en las 256 páginas permitidas para el direccionamiento directo
BK	Registro que indica el tamaño del bloque al usar direccionamiento circular

IOF	Registro de flags específico de los pines XF0 y XF1 (I/O)
IE	Registro habilitador de interrupciones, de la CPU y DMA
IF	Registro de flags o banderas de interrupción
ST	Registro de status de la CPU
SP	Puntero de fila (stack pointer)
IR0-IR7	2 registros de direccionamiento indexado
R0-R7	8 registros de 40 bits, pueden almacenar números de 32 bits punto fijo, y 40 bits punto flotante
AR0-AR7	8 registros auxiliares de 32 bits de propósito general, y direccionamiento indirecto

Unidades de memoria

El C31 consta dentro de su configuración con memoria RAM, que se dividen en 2 bloques de 1kbyte cada uno, para el almacenamiento temporal de datos o instrucciones, como también una memoria cache de 64 bytes, de alta velocidad para el almacenamiento de instrucciones o secciones que a menudo se repiten, logrando con esto una ejecución rápida de los programas.

Matlab y Simulink.

MATLAB es el primer entorno de cálculo técnico hoy en día. La primera versión fue escrita en la Universidad de New México y en la universidad de Stanford a finales de los años 70, y tenía como finalidad utilizarse en cursos de teoría de matrices, álgebra lineal y análisis numérico.

Hoy en día, las capacidades de MATLAB se extienden mucho más allá del original "laboratorio matricial". MATLAB es un sistema interactivo y un lenguaje de programación para científicos en general y para cálculos técnicos. Su elemento de datos básico es una matriz que no requiere dimensionamiento. Esto permite la solución de muchos en una pequeña fracción del tiempo que llevaría escribir un programa en un lenguaje tal como FORTRAN, BASIC o C. Más aún las soluciones del problema se expresan en MATLAB casi exactamente a como se escriben matemáticamente.

Las matemáticas es el lenguaje común de gran parte de la ciencia y de la ingeniería. Matrices, ecuaciones diferenciales, arrays de datos y gráficas son un bloque de construcción básicos de las matemáticas aplicadas y de MATLAB. Es la base matemática fundamental que hace que MATLAB sea accesible y potente.

MATLAB es una aplicación destinada a cálculos matemáticos, si bien dispone de ciertas funciones destinada a temas más específicos, como por ejemplo la TolBox de control, que facilita el estudio de sistemas dinámicos y su regulación.

Además, existe un complemento de MATLAB llamado SIMULINK, que nos permite un enfoque más gráfico de los sistemas de control.

Al ejecutar MATLAB, aparecerá una ventana en blanco, llamada ventana de comandos. La forma de trabajar con MATLAB es como con cualquier calculadora. Este software nos permite realizar funciones que son más complejas que sumar y restar.

A continuación se darán a conocer algunas de las funciones más importantes dentro de MATLAB:

Funciones habituales	
abs(x)	Valor absoluto de x. Si x es un número complejo, abs(x) nos da su módulo
acos(x)	Arco coseno de x
asin(x)	Arco seno de x
atan(x)	Arco tangente de x. Devuelve un ángulo entre –90 y 90 grados
atan2(x, y)	Arco tangente de entre x e y. Devuelve un ángulo entre 0 y 360 grados
cos(x)	Coseno de x
sin(x)	Seno de x
tan(x)	Tangente de x
exp(x)	Función exponencial
log(x)	Logaritmo neperiano de x
log 10(x)	Logaritmo en base 10 de x
rem(x, y)	Resto de la división x / y
unwrap(x)	Sitúa el ángulo x entre π y -π
roots(x)	Halla las raíces del polinomio x
fzero("f(x), n)	Encuentra la solución de la ecuación f(x)=0; n es el valor por donde empieza a iterar para hallar la solución
sqrt(x)	Raíz cuadrada de x
angle(x)	Ángulo de x
real(x)	Parte real de x
imag(x)	Parte imaginaria de x

SIMULINK es un software para modelado, simulación y análisis de sistemas dinámicos. Con SIMULINK se pueden analizar sistemas lineales y no lineales, modelados tanto en tiempo continuo como discreto. En este ultimo caso soporta sistemas en el mismo entorno con diferente frecuencia de muestreo.

Para modelar, SIMULINK provee una interfaz gráfica de usuario (GUI) para construir modelos basados en diagramas de bloque, utilizando simples operaciones con el ratón. Con este interfaz se diseña de forma similar que como se realizaría sobre un papel.

SIMULINK contiene un conjunto de librerías de bloques de fuentes de señal, componentes lineales y no lineales, etc. Se pueden, adicionalmente, crear bloques específicos (funciones-S).

Los modelos se construyen de forma jerárquica, los cuales pueden ser vistos desde un nivel superior que al seleccionarlos (doble click) se entra en un nivel más detallado, esto permite organizar el modelo de forma similar a como sus partes interactúan.

Después de definir el modelo, este se puede simular utilizando una de las técnicas de integración que este tiene incorporado. Usando los osciloscopios y otros bloques de visualización se pueden ver los resultados de simulación mientras esta se realiza. Los resultados de simulación se pueden colocar en el espacio de trabajo de MATLAB para propósitos de postprocesamiento y visualización.

Elementos fundamentales para construir modelos con SIMULINK.

1.- Bloques: Los bloques son los elementos basados en los cuales se construye un modelo en SIMULINK. Se puede crear un modelo virtualmente de cualquier sistema dinámico creando e interconectando bloques de forma apropiada.

Las operaciones fundamentales que se pueden realizar con los bloques son:

a.- Copiar, mover, borrar y duplicar bloques.

b.- Especificar parámetros del bloque (Edit - Block Properties).

c.- Rotar bloques (Ctrl – R).

d.- Cambiar tamaño del bloque.

e.- Cambiar el nombre del bloque.

2.- Líneas: Las líneas en un modelo construido en SIMULINK transporta señales. Cada línea puede transportar un escalar o un vector que conecta la salida de un bloque a la entrada de otro, adicionalmente, puede conectar un puerto de salida de un bloque con varios puertos de entrada de varios bloques usando líneas de ramificación.

Las operaciones fundamentales que se pueden realizar con las líneas son:

a.- Dibujar línea entre bloques.

b.- Dibujar línea ramificada (Transporta señal desde una línea existente).

c.- Mostrar el número de señales en una línea vector (Format – Vector Line Widths).

d.- Definir etiqueta que identifique la línea (debe tenerse especial cuidado en hacer doble click en la línea, si se realiza en una proximidad puede crear una nota dentro del modelo).

3.- Subsistemas: Se utilizan para simplificar un conjunto de bloque por agrupamiento cuando se incrementa el tamaño y la complejidad del modelo.

Los subsistemas tienen las siguientes ventajas:

a.- Ayudan a reducir el número de bloques que se muestran en la ventana.

b.- Mantiene la funcionalidad entre los bloques conectados.

c.- Permite crear un sistema de bloques jerárquicos.

Aunque existen dos formas diferentes de crear un subsistema, la más utilizada es crearlo a partir del agrupamiento de bloques existentes. Los pasos para ello son:

a.- Crear con el ratón una "caja de entorno" que incluyan los bloques que constituirán el subsubsistema.

b.- Del menu Edit – Create Subsystem.

Algunas recomendaciones para construir modelos:

a.- Use sistemas jerárquicos: Los modelos complejos se benefician de la incorporación de jerarquías usando subsistemas. La agrupación de bloques simplifica el nivel superior del modelo haciéndolo fácil de entender.

b.- Cree modelos claros: Modelos bien organizados y documentados son fáciles de entender e interpretar. Para lo anterior, utilice etiquetas para las señales y anotaciones sobre el modelo.

c.- Diseñe el modelo en un papel y después use el computador, coloque primero los bloques que se van a utilizar y luego realice las conexiones, esto último reduce el tiempo de diseño a través de sucesivas aperturas de las bibliotecas de bloques.

Ejecución de una simulación.

La ejecución de una simulación en SIMULINK puede ser a través de ventanas en el menú de comandos o directamente a través de la línea de comandos en el espacio de trabajo. La primera es la más utilizada en la fase de modelado y simulación de sistemas.

Uso de comandos por ventanas.

El uso de comandos a través de ventanas es fácil e interactivo. Los comandos permiten seleccionar el método de solución de las ecuaciones diferenciales ordinarias (ODE) y definir los parámetros de simulación sin tener que recordar la sintaxis de los comandos.
Una importante ventaja es que se pueden realizar ciertas operaciones de forma interactiva mientras la simulación se está ejecutando:

1.- Se puede alterar el tiempo de parada (stop time), el método de solución de ecuaciones diferenciales (solver) y el tamaño de ejecución de un paso (step size).

2.- Se puede ver la señal que porta una línea por un clic de ratón o a través de un osciloscopio.

3.- Se pueden modificar los parámetros de un bloque:

a.- Número de estados de entrada y salida.

b.- Período de muestreo.

c.- Número de cruces por cero.

Durante la ejecución de la simulación no se puede cambiar la estructura de un modelo: adicionar o borrar líneas de un bloque, etc. Para lo anterior se necesita detener la simulación, realizar los cambios, y reinicializar la simulación.

Para establecer los parámetros de simulación se selecciona Simulation-Parameters en el menú de comandos, este está formado:

a.- Solver: Permite establecer los tiempos de inicio y parada, seleccionar el método de solución de ecuaciones ordinarias y otras opciones de salida.

b.- Workspace I/O: Controla la Entrada-Salida del Espacio de Trabajo.

c.- Diagnostics: Selecciona el nivel de mensajes de error durante la simulación.

Cuando se ha seleccionado el método de resolución de ecuaciones y los parámetros de simulación se puede ejecutar la misma a través del comando Simulation-Start.

Un sonido del ordenador indica cuando ha finalizado la simulación. Los comandos Simulation-Stop o Simulation-Pause detienen la misma. En caso de utilizar la segunda opción, después de realizar los cambios, se puede continuar la misma usando Simulation-Continue.

Cuando SIMULINK detecta un error en el proceso de simulación, aparece una ventana Simulation-Diagnostics que muestra los errores durante la simulación. Simultáneamente, el bloque que generó el error se destaca en el modelo establecido por el diseñador, para que se realicen las correcciones oportunas.

La librería de bloques proporciona centenares de funciones predefinidas para la creación de modelos de diagramas-bloque de sistemas lineales, no lineales, de tiempo discreto, tiempo continuo, híbrido, SISO, SIMO y sistemas multitarea, de forma que cada usuario pueda crear sus propios bloques modificando los ya existentes o incorporando código MATLAB, C o FORTRAN.
Las operaciones de modelado y simulación se controlan de forma interactiva mediante menúes desplegables o empleando la línea de comando MATLAB para simulaciones en modo batch.
La librería no lineal incluye un complemento completo de componentes no lineales para modelar un comportamiento como sistema de mundo real.
Los modelos Simulink múltiples y las capas establecidas de la jerarquía pueden permanecer abiertos simultáneamente, facilitando así las operaciones de edición de bloques y de corte-pegado.

Los parámetros de bloque se introducen en los cuadros de dialogo como escalares, vectores, o matrices empleando ya sean valores numéricos o variables y expresiones MATLAB.
El gráfico ajusta automáticamente su escala a la amplitud de la señal. Estos valores pueden imprimirse desde, SIMULINK o ser guardados en una amplia gama de formatos de archivos gráficos para su incorporación a informes.
Para simulaciones en vivo, una variedad de bloques gráficos monitorizan la respuesta del sistema mientras avanza la simulación.
Una línea gruesa identifica canales múltiples enlazados como un vector único. Las conexiones vectorizadas entre bloques son transparentes.
Un bloque Hit Crossing proporciona una precisión mayor para sistemas con discontinuidades.
Los gráficos desarrollados en 2D y 3D son accesibles desde MATLAB para la ampliación de los resultados de la simulación.
La integración con MATLAB facilita el acceso inmediato a la potencia matemática, gráfica y de programación de MATLAB para analizar datos, automatizar procedimientos y optimizar parámetros.

Acelerador de SIMULINK.

Para incrementar la velocidad de SIMULINK se debe instalar el acelerador "Accelerator". Este permite automáticamente generar una versión mejorada de los modelos los cuales correrán diez veces más rápido que el original. El acelerador puede ser usado sobre modelos continuos, discretos en el tiempo y híbridos.
El acelerador trabaja generando y compilando un código-C para un modelo dado. Una vez se completa la compilación, la simulación es ejecutada en la ventada de modelos de SIMULINK exactamente igual que antes sólo que más rápidamente. El propósito del acelerador es aumentar la velocidad de simulación.
Si el programa MATLAB posee instalado el "Accelerator" podrá iniciarse la acción aceleradora seleccionando la opción simulation en el menú principal del SIMULINK y dentro de esta seleccionando la opción Accelerate. Esta acción es totalmente transparente en el sentido de que el incremento de la velocidad se presenta sin ningún otro requerimiento por parte del usuario.

Generador de código-C en SIMULINK

Una vez se ha creado un modelo dinámico en SIMULINK, se puede invocar el generador de código-C que permite convertir el diagrama de bloques implementado en un código C. Este puede ser útil para varios propósitos: puede ser usado para control en tiempo real, simulación en tiempo real o simulación acelerada en tiempo no real. Sus aplicaciones pueden ser control de movimiento, control de procesos, sistemas automotores, equipos médicos, robótica, etc.

El código-C es diseñado tal que puede ser ejecutado en tiempo real. No requiere ser escrito manualmente por un programador pues es creado a nivel de diagramas de bloques en SIMULINK. El código generado puede correr sobre un amplio rango de hardware ubicado en estaciones de trabajo, PC o microprocesadores. Este código es la forma en la que puede usare el SIMULINK para adquisición de datos.

Code Composer.

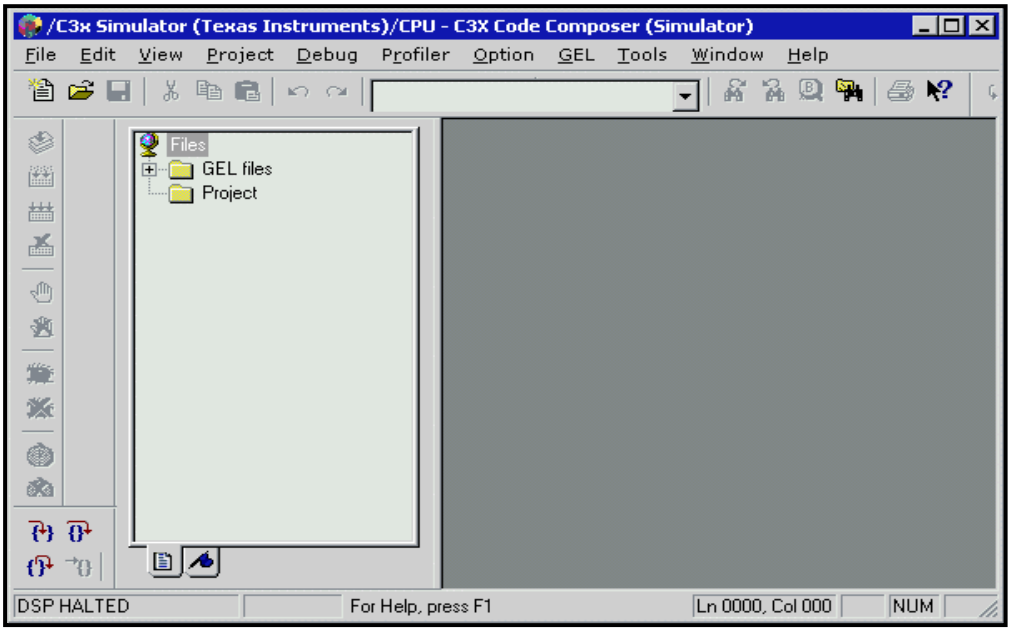
Para poder crear aplicaciones y desarrollar trabajos sobre los DSP, se requiere básicamente de un software, instalado en el PC, que trabaje específicamente con esta tarjeta y que contenga los elementos necesarios para crear programas y visualizarlos a través de la tarjeta. Uno de estos software es: "**CODE COMPOSER**".

Code Composer es un software creado por la Texas Instruments, TI, especializado para la creación, compilación, linkado y creación de archivos ejecutables de programas que interactúen con la tarjetas (DSPs) TMS320C3x y TMS320C4x. La ventaja de éste, es que es el primer software que trae incorporado los cuatro elementos (creación, compilación, linkado y creación de archivos ejecutables), en un solo programa; ya que antes, se debía realizar cada uno de estos cuatro pasos con cuatro softwares distintos, lo cual dificultaba y hacía más tedioso y lento el trabajo del usuario para interactuar con la tarjeta.

Code Composer Studio es un software en un ambiente integrado de desarrollo, construido específicamente para Procesadores Digitales de Señales; DSP. Es así como en él, un usuario puede construir, corregir, perfilar y manejar proyectos en una sola aplicación dentro del mismo programa, sin necesidad de realizar varios pasos repartidos en distintos softwares, y con la ventaja de poder ser modificados, si así se desea. Además, permite realizar análisis gráficos de la señal, Entrada/Salida de los archivos, para lo cual, utiliza las herramientas de Texas Instruments en cada procesamiento de los proyectos que en él se elaboren, es decir, en definitiva es el administrador de estas herramientas: compilación, linkado, ensamblado y creación del ejecutable.

Code Composer es uno de los ambientes de desarrollo más sofisticados y amigables para el trabajo con DSP. Sus estructuras, presentan características únicas que incluyen capacidades adicionales para reducir el tiempo de desarrollo de cualquier aplicación.
Así, el objetivo principal de Code Composer es proporcionar las herramientas de desarrollo más avanzadas y más fáciles de utilizar para el manejo del software de DSP, aumentando con esto la productividad y reduciendo al mínimo el tiempo de desarrollo.

La presentación de Code Composer es la siguiente:



A continuación se detallan los pasos a seguir en el uso de Code Composer, para el desarrollo o creación de cualquier aplicación.

- Creación de una Aplicación.

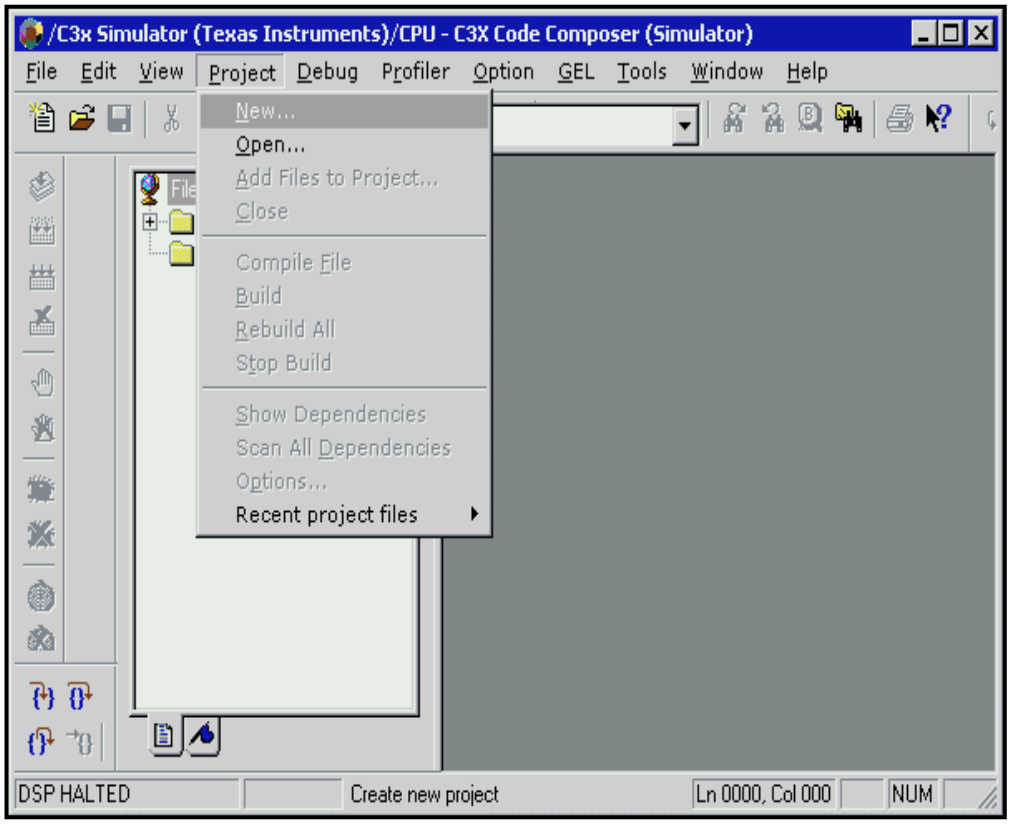
El primer paso a considerar en la creación de un proyecto es el lugar dentro de nuestro computador; donde el mismo, va a estar ubicado. Para ello es necesario crear una carpeta o directorio de trabajo donde se almacenarán los archivos necesarios, tanto intermedios como finales, del proyecto.

Por defecto **Code Composer Studio** comienza dentro de la carpeta my projects. Por su parte; el usuario puede crear un nuevo directorio de trabajo en el lugar que considere más apropiado. En este ejemplo se desea llamar a este directorio ubicado en el escritorio: "ejemplo_cc".

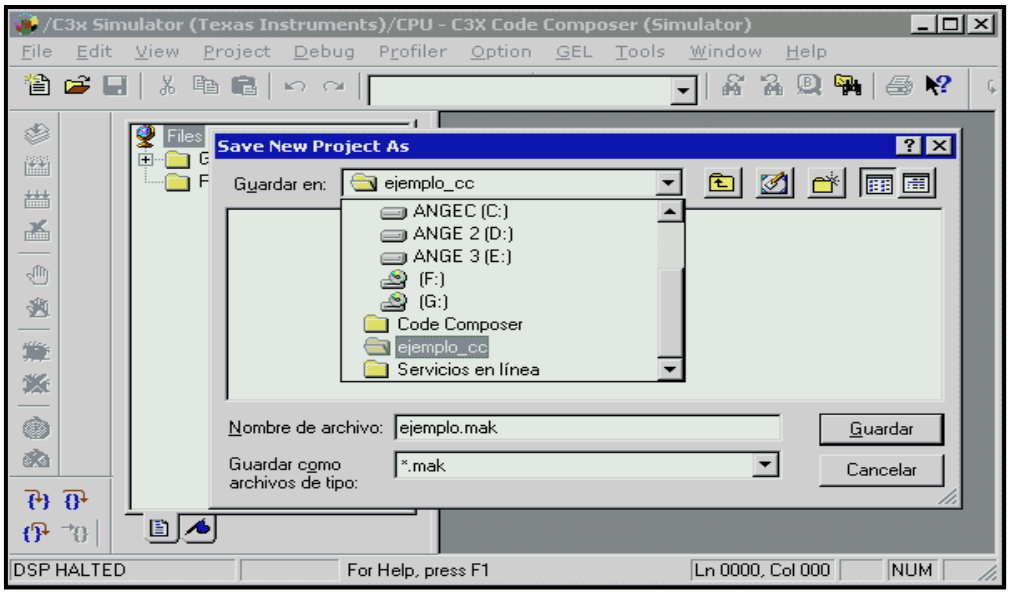
En la carpeta creada se deberá colocar inicialmente el archivo de comandos **comun.cmd**; el cual será posteriormente necesario para el linkado y la creación del ejecutable. Para esto, se copia el archivo **comun.cmd** desde la carpeta de my projects y se pega en el directorio creado.

Una vez creado el directorio de trabajo (ejemplo_cc), y con el archivo de comandos ya en el mismo, se crea un nuevo proyecto. Para ello, sobre **Code Composer** se debe seleccionar desde el menú principal la opción:

Project → New,



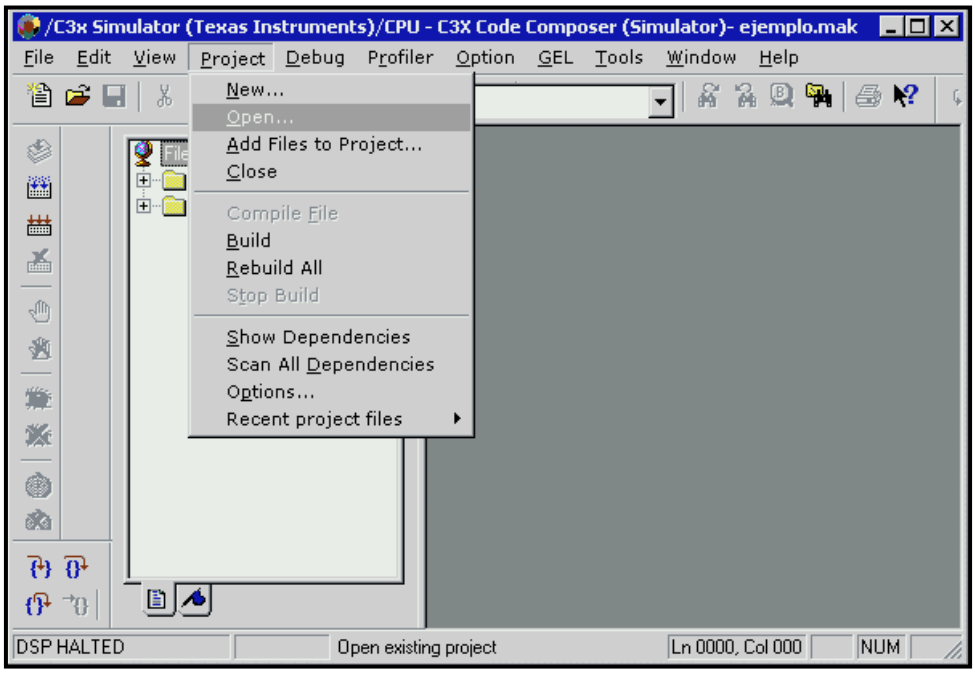
Aparecerá entonces en la pantalla del computador, la ventana donde guardar el proyecto creado. Únicamente será necesario incluir el camino hasta llegar al directorio deseado. Luego coloque el nombre del proyecto que desee crear, el cual debe tener extensión.**mak**.



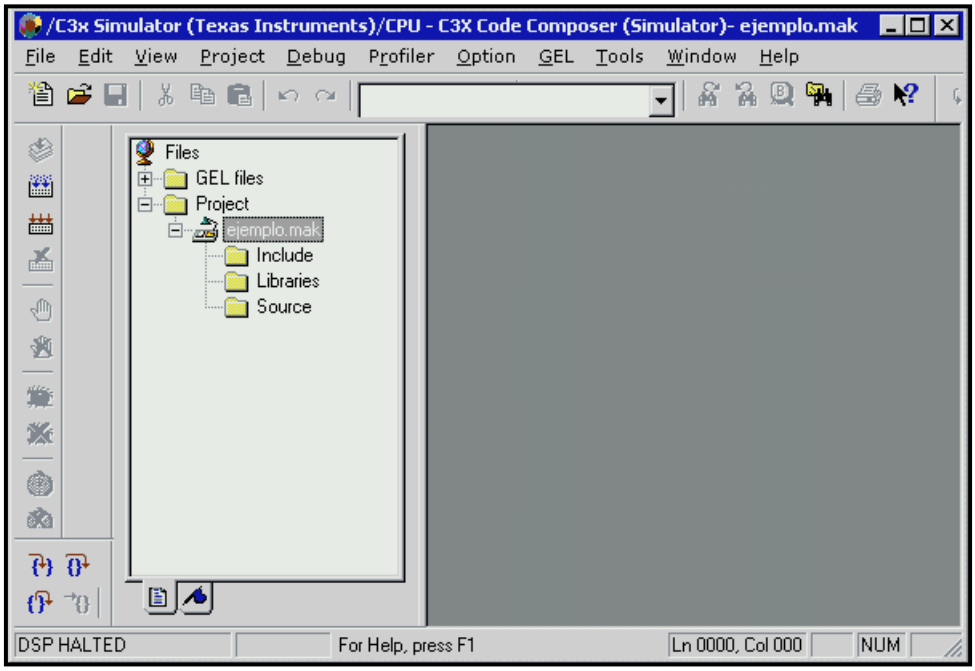
Una vez aceptado, se creará un fichero, **ejemplo.mak**, mientras que en el directorio de trabajo se creará un archivo con ese nombre (no aparece en el mismo reflejado hasta que nos salgamos del proyecto).

Ahora; en caso de que el proyecto haya sido creado con anterioridad, para abrirlo sólo es necesario realizar la siguiente acción:

Project → Open



En ambos casos, ya sea creado con anterioridad o en esta misma sesión, la pantalla de Code Composer mostrará la forma que aparece a continuación, donde en la zona de la izquierda se puede observar los archivos que contiene el proyecto simplemente haciendo click con el ratón en el símbolo "+" de cada carpeta.



Una vez que se ha creado el proyecto sobre el cual trabajar es necesario modificar las opciones que hacen referencia al compilador, ensamblador y linkador, para seleccionar la tarjeta sobre la cual vamos a destinar el archivo salida(.out).

Project → Option.

Aparecerá una pantalla donde se puede modificar los datos de los mismos. Solo nos interesa seleccionar la tarjeta sobre la cual trabajaremos: **C31.**

Dentro de la opción **compiler** se ha de seleccionar las opciones que hacen referencia al compilador, Únicamente habrá que modificar el procesador de la tarjeta.

En la opción **assembler** únicamente se ha de modificar la familia del procesador. El resto de los casilleros son distintas opciones que no hacen falta modificar para el resultado que se busca.

Finalmente en la opción **linker** no hace falta modificar nada, a excepción de la casilla inferior derecho, que es necesario permitir la generación COFF versión 0 en el archivo de salida final, compatible con el programa DSK3D que luego se utiliza para correr el programa.

Nota: Debido a que el DSK3D trabaja en DOS, el archivo de salida debe tener como máximo 8 caracteres.

El siguiente paso es la creación del proyecto consiste en añadir el programa en c (o en assembler):

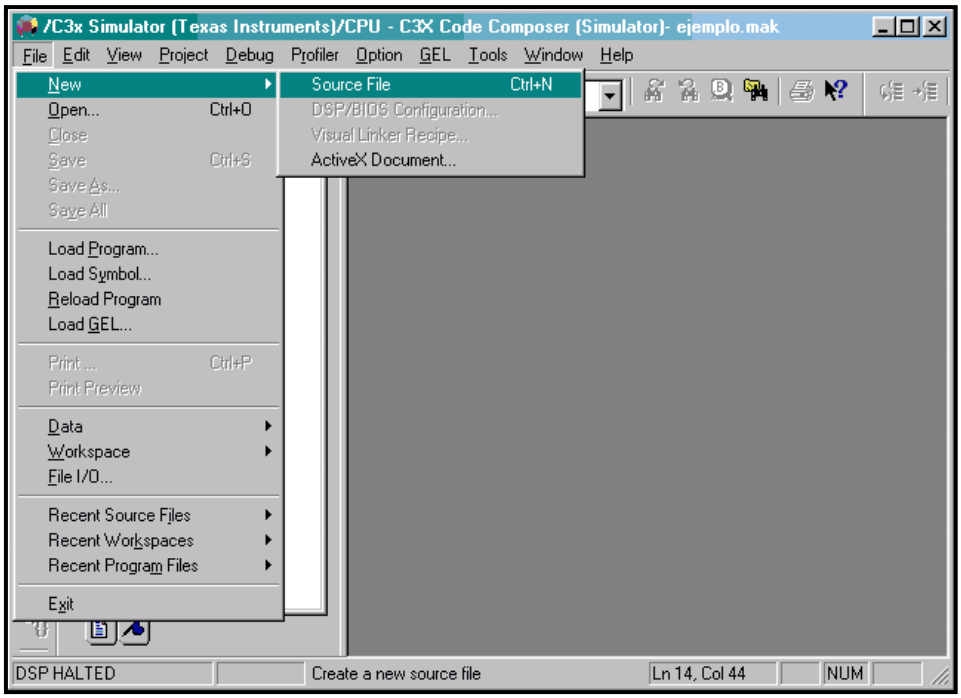
Project → add files to proyect.

Y aparecerá una ventana para buscar el archivo requerido. Se supone que el archivo requerido se encuentra en otra carpeta distinta a la de trabajo. Simplemente se busac a través del navegador y se acepta. Aparecerá en la pantalla del code composer a la izquierda.

Pero el Code composer se puede utilizar también como editor , en este caso la forma de actuar sería la siguiente:

Files → New → Source file

Se abrirá una ventana en blanco correspondiente al editor. Ahí se puede programar tanto en c como en assembler. Después es necesario guardarlo e incluirlo en el proyecto.



Nota: muchos programas en c presentan una línea #include cuya función es permitir al programa en cuestión hacer uso de una determinada librería. Para poder comprobar si está trabajando de forma correcta, con la acción:

Proyect → Show Dependencies

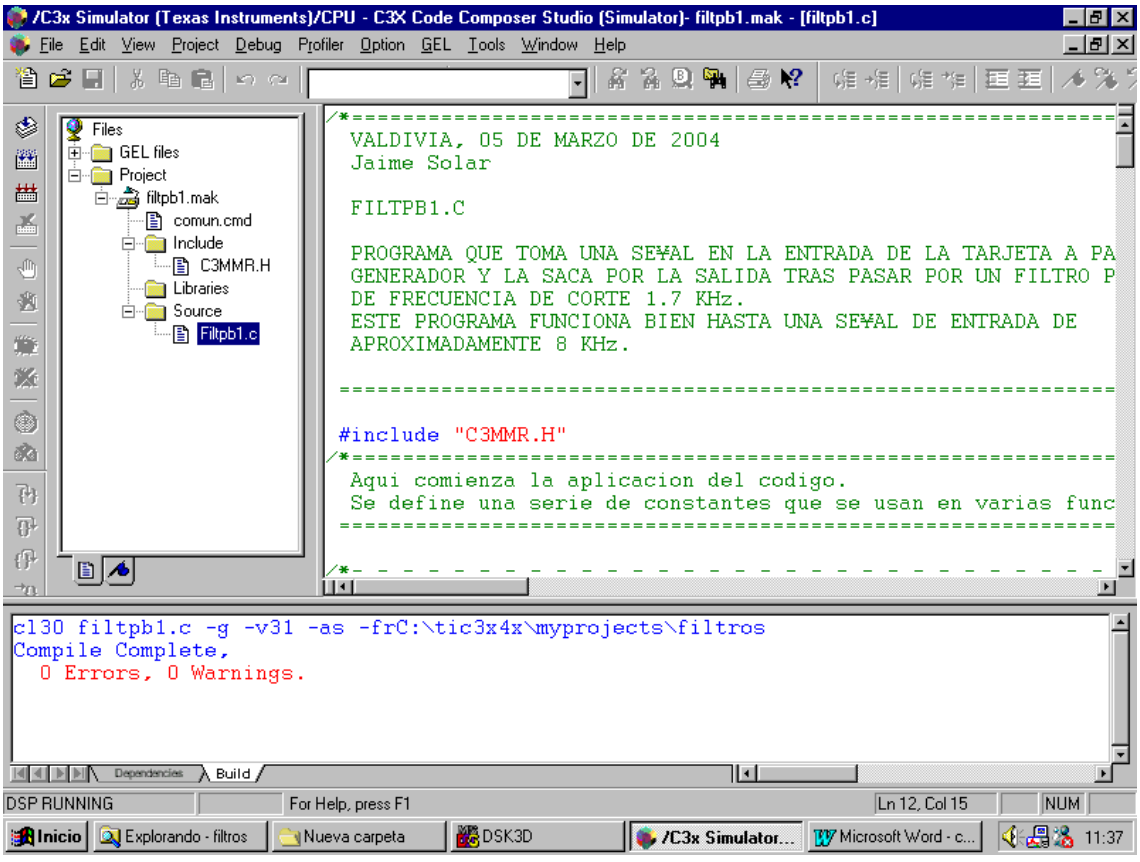
Aparecerá en la parte inferior de Code Composer un mensaje afirmativo (en el caso de que no haya error) o bien un mensaje negativo en color rojo (si no encuentra la librería).

En este segundo caso es necesario incluir la librería a la que hace referencia el error en el directorio donde se encuentre el programa en c.

El siguiente paso consiste en la compilación del programa para ello mismo se debe estar abierto (doble click en el ícono de la izquierda) y seleccionar:

Project → Compile

Si todo va bien Code Composer mostrará el siguiente aspecto y en el directorio de trabajo se habrá creado el archivo objeto (.obj) requerido.



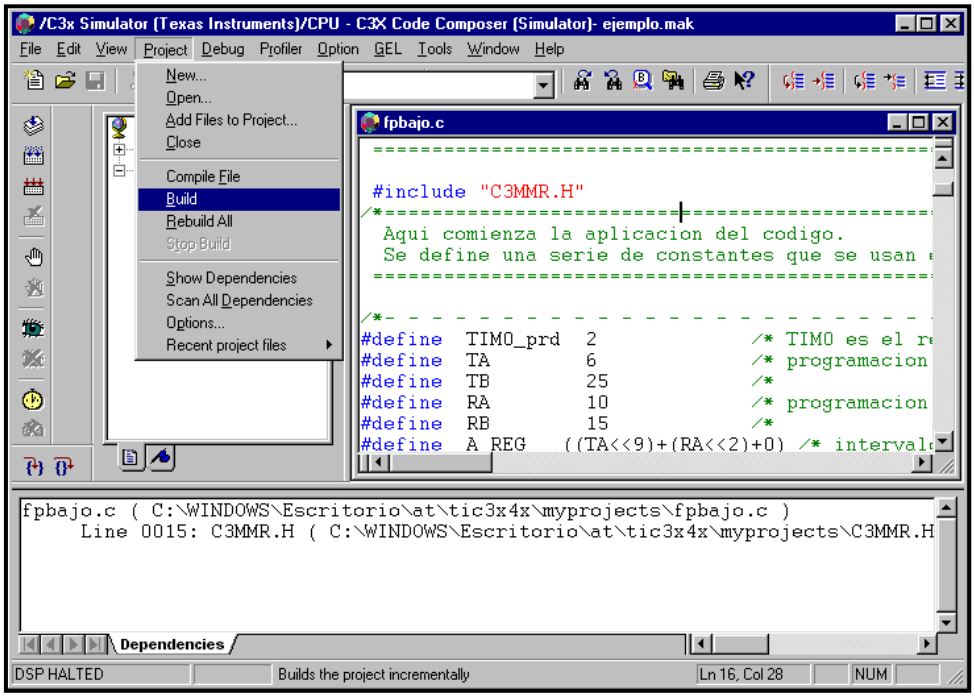
El siguiente paso en el proyecto consiste en la inclusión del archivo de comandos necesario para generar el archivo de salida. Se trabaja entonces como en el caso anterior:

Project → Add files to project

A continuación, se abrirá una ventana dentro de Code Composer, donde sólo hay que navegar en el directorio para seleccionar el archivo **comun.cmd** dentro del directorio de trabajo, (el que se supone fue incluido en el directorio al inicio del proyecto).

Una vez que se tienen todos los archivos necesarios para la realización del proyecto se construye el mismo, mediante la siguiente secuencia de comandos:

Project → Incremental Build

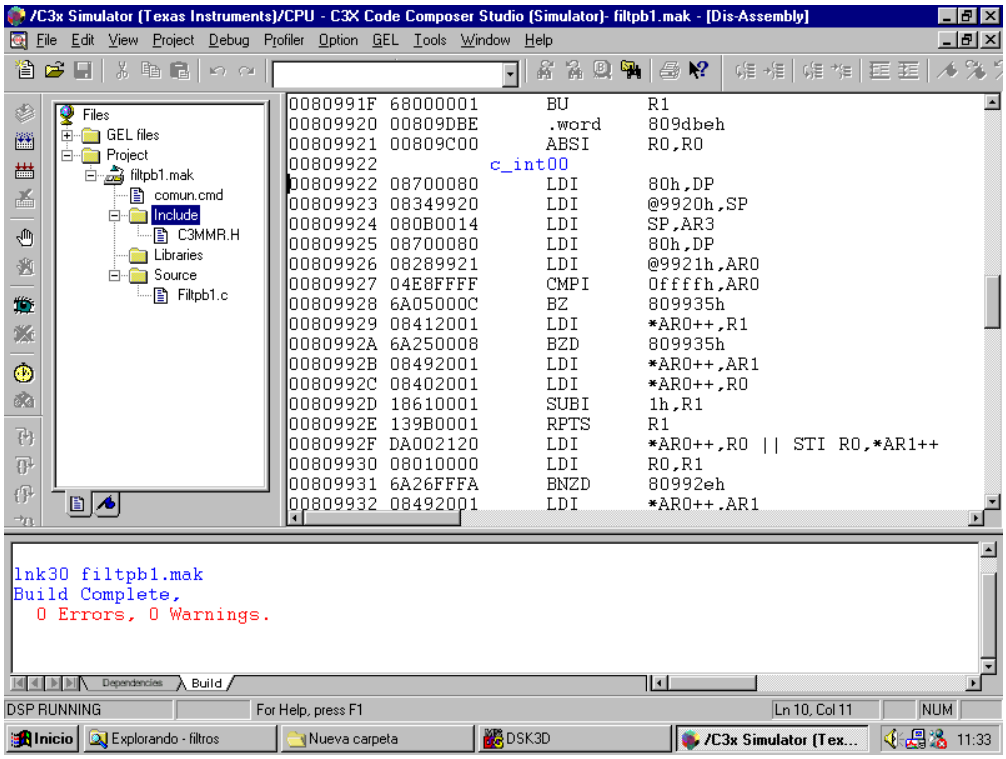


En este momento se tiene en el directorio de trabajo el archivo ejecutable necesario para trabajar con la tarjeta. Solo queda cargarlo en el simulador Code Composer: Para ello se realiza lo siguiente:

Files → Load program

Y en la pantalla del computador aparecerá el programa realizado en la ventana de la derecha, listo para hacerlo ejecutar. Para esta última opción, únicamente hay que realizar la secuencia:

Debug → Run.



Una vez que el proyecto se ha concluido, y se ha generado el archivo ejecutable requerido, se pasa a la carga del mismo en la tarjeta DSP. Para ello se utiliza el programa DSK3D, cuyos comandos más útiles para el usuario son:

LOAD: carga el archivo.out en la tarjeta.

RUN: ejecuta el archivo ejecutable anteriormente cargado.

QUIT: sale del programa DSK3D.