# CAPÍTULO VIII

**INTRODUCCION**

Un filtro es un sistema, que dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Los filtros digitales tienen como entrada una señal digital y a su salida tienen otra señal digital, pudiendo haber cambiado en amplitud, frecuencia o fase dependiendo de las características del filtro.

El filtrado digital es parte del procesado de señal digital. Se le da la denominación de digital más por su funcionamiento interno que por su dependencia del tipo de señal a filtrar, así podríamos llamar filtro digital tanto a un filtro que realiza el procesado de señales digitales como a otro que lo haga de señales analógicas.

El filtrado digital consiste en la realización interna de un procesado de datos de entrada. El valor de la muestra de la entrada actual y algunas muestras anteriores (que previamente habían sido almacenadas) son multiplicadas, por unos coeficientes definidos. También podría tomar valores de la salida en instantes pasados y multiplicarlos por otros coeficientes. Finalmente todos los resultados de todas estas multiplicaciones son sumados, dando una salida para el instante actual. Esto implica que internamente tanto la salida como la entrada del filtro serán digitales, por lo que puede ser necesario una conversión analógico-digital o digital-analógico para uso de filtros digitales en señales analógicas.

Los filtros digitales se usan frecuentemente para tratamiento digital de la imagen o para tratamiento del sonido digital.

**TIPOS DE FILTROS**

Hay varios tipos de filtros así como distintas clasificaciones para estos filtros:

De acuerdo con la parte del espectro que dejan pasar y que atenúan hay:

- Filtros pasa alto.
- Filtros pasa bajo.
- Filtros pasa banda.
    - Banda eliminada.
    - Multibanda.

- Pasa todo.
- Resonador.
- Oscilador.
- Filtro peine (Comb filter).
- Filtro ranura o filtro rechaza banda (Notch filter), etc.

De acuerdo con su orden:

- primer orden
- segundo orden, etc.

De acuerdo con el tipo de respuesta ante entrada unitaria:

- FIR (Finite Impulse Response)
- IIR (Infinite Impulse Response)
- TIIR (Truncated Infinite Impulse Response)

De acuerdo con la estructura con que se implementa:

- Directa
- Transpuesta
- Cascada
- Fase Lineal
- Laticce

**EXPRESIÓN GENERAL DE UN FILTRO DIGITAL**

En muchas aplicaciones del procesado de señales es necesario diseñar dispositivos o algoritmos que realicen operaciones sobre las señales y que los englobaremos bajo la denominación genérica de **sistemas.**

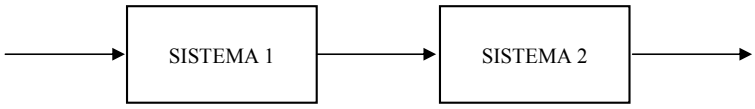
Un sistema opera sobre una señal de entrada o excitación según una regla preestablecida, para generar otra señal llamada salida o respuesta del sistema a la excitación propuesta y que puede simbolizarse:

$$y[n] = T\left(x[n]\right) \tag{1}$$

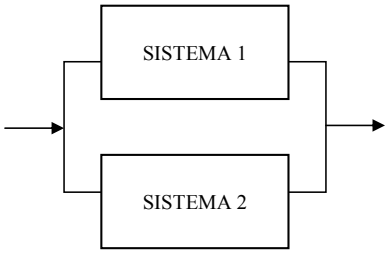
donde *T* simboliza la transformación, operador o procesado realizado por el sistema sobre la señal " *x* " para producir la señal " *y* ", ver Figura 1. Una de las motivaciones más fuertes para el desarrollo de herramientas generales para el análisis y diseño de sistemas es que proviniendo a menudo de aplicaciones muy diferentes tienen descripciones matemáticas similares.

Existen varias maneras de representar un sistema, ya que muchos sistemas reales están construidos como interconexiones de varios subsistemas, tal como se grafica en la Figura 2.

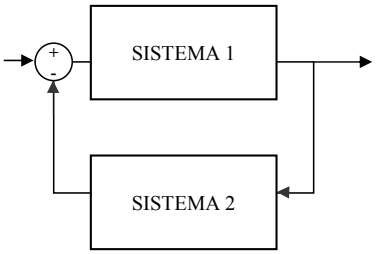

Figura 1: Esquema de sistema, señal de entrada y respuesta o salida del sistema.




a) serie o cascada




b) paralelo




c) retroalimentación

Figura 2: Interconexión de Sistemas

Existen dos métodos básicos para el análisis del comportamiento o respuesta de un sistema lineal a una determinada entrada. Un primer camino se basa en obtener la solución de la ecuación entrada-salida del sistema que en general tiene la forma de las ecuaciones en diferencias lineales a coeficientes constantes $a_m$, $b_k$:

$$\sum_{m=0}^{N_a-1} a_m y[n-m] = \sum_{k=0}^{N_b-1} b_k x[n-k] \qquad (2)$$

siendo $N_a$ y $N_b$ los ordenes máximos de las diferencias en la ecuación correspondientes a la salida y entrada del sistema.

El segundo método para el análisis del comportamiento del sistema reside en la aplicación del principio de superposición y consiste en descomponer la señal de entrada en una suma pesada de señales elementales las cuales se escogen de manera que sea conocida la respuesta del sistema a las mismas. Siguiendo esta línea, una señal a tiempo discreto puede visualizarse como una secuencia pesada de impulsos unitarios:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k].\delta[n-k] \qquad (3)$$

Aplicando la propiedad de superposición de los SLIT (Sistemas Lineales e Invariantes en el Tiempo) (Oppenheim y Willsky, 1983), se puede determinar la salida del sistema ante una cierta entrada de la siguiente manera:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k] \qquad (4)$$

siendo $h[n]$ la respuesta o salida del sistema ante una entrada equivalente a un impulso unitario $\delta[n]$, denominada *respuesta al impulso del sistema*. El segundo miembro de la expresión representa el producto de convolución de la señal de entrada $x[n]$ y la respuesta al impulso del sistema $h[n]$, esto es:

$$y[n] = x[n] * h[n] = h[n] * x[n] \qquad (5)$$

Tanto en el caso continuo como en el caso discreto, la respuesta al impulso del sistema LTI presenta las siguientes propiedades:

a) sin memoria: $h[n] = 0 \quad para \quad n \neq 0$

b) causal: $h[n] = 0 \quad para \quad n < 0$

c) invertible: dado $h[n]$ existe $h'[n]$ tal que $h[n] * h'[n] = \delta[n]$

d) estable: $\sum_{k=-\infty}^{\infty} |h[n]| < \infty$

Existen otras formas de representar un filtro, todas estas equivalentes a la respuesta al impulso unitario de sistema SLIT, sin embargo muchas veces conviene más una u otra representación. En el caso aplicar la transformada Z, a la ecuación en diferencias de (2) se obtiene la función de transferencia del sistema (Oppenheim y Willsky, 1983; Proakis y Manolakis; 1996; Oppenheim y Schafer, 1999):

$$H(z) = \frac{\sum_{k=o}^{N_b-1} b_k z^{-k}}{\sum_{m=0}^{N_a-1} a_m z^{-m}} \tag{6}$$

en donde $z = Ae^{j\Omega}$ es la variable compleja en forma polar. Particularmente si el modulo $A = 1$, la expresión (6) se reduce a la respuesta en frecuencia del sistema a través de la transformada de Fourier a tiempo discreto (Oppenheim y Willsky, 1983; Proakis y Manolakis; 1996; Oppenheim y Schafer, 1999):

$$H(\Omega) = \frac{\sum_{k=o}^{N_b-1} b_k e^{-j\Omega k}}{\sum_{m=0}^{N_a-1} a_m e^{-j\Omega k}} \tag{7}$$

en donde $\Omega$ es la frecuencia angular relacionada a dicha transformada. Por otro lado, para el caso de realizar la representación en el domino temporal discreto o de la variable *n*, se obtiene la salida del sistema:

$$y[n] = \sum_{k=0}^{N_b-1} b_k x[n-k] - \sum_{m=1}^{N_a-1} a_m y[n-m] \tag{8}$$

donde los coeficientes $a_m$ y $b_k$ son los coeficientes que definen el filtro, por lo tanto el diseño consiste en calcularlos. Como regla general se suele dejar el término $a_0 = 1$.

**FILTROS DIGITALES FIR**

Los filtros digitales de Respuesta Finita Impulsiva o filtros FIR por sus siglas en ingles Finite Impulse Response, se trata de un tipo de filtros digitales en el que, como su nombre indica, si la entrada es una señal impulso la salida tendrá un número finito de términos no nulos. La estructura de señal a la salida del filtro se basa solamente en la combinación lineal de las entradas actuales y anteriores, esto es:

$$y[n]=\sum_{k=0}^{N-1} b_k x[n-k]=\sum_{k=0}^{N-1} h[k]x[n-k] \;\; con \;\; h[k]=\{h_0\; h_1 \ldots h_{N-1}\} \tag{9}$$

en donde *N* es el orden del filtro, que también coincide con el número de términos no nulos y con el número de coeficientes $b_k$ del filtro. Observe que la expresión de la ecuación (9) corresponde a la convolución de la señal de entrada *x[n]* con la respuesta impulsional del filtro FIR *h[n]*.

Aplicando la transformada Z a la respuesta impulsional del filtro FIR *h[n],* se tiene:

$$H(z)=\sum_{k=0}^{N-1} h_k\, z^{-k} = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-(N-1)} \tag{10}$$

En la Figura 3, se muestra el diagrama en bloques de la estructura básica del filtro FIR, para una cantidad de 12 coeficientes.

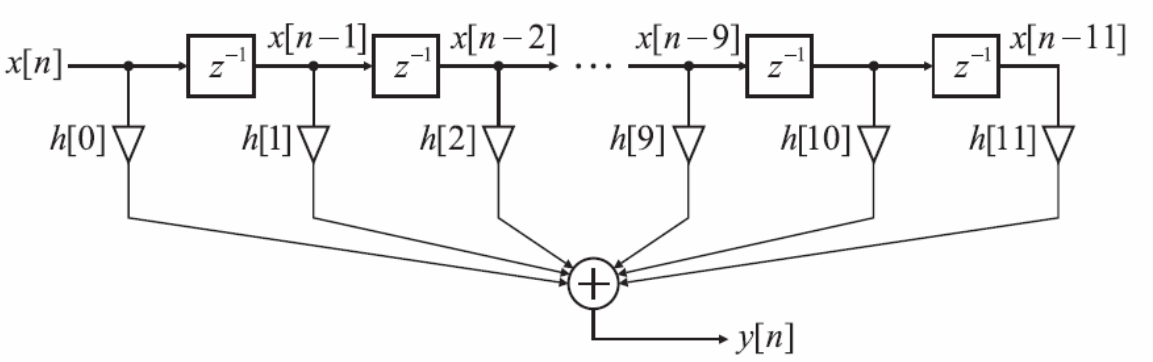



Figura 3: Representación en diagrama de bloques del filtro FIR, para un total de 12 coeficientes.

**RESPUESTA A LA FRECUENCIA IDEAL**

Ya que un filtro FIR ideal es aquel sistema discreto cuya salida es una versión escalada, a través de un factor $A_0$ , y desfasada de la entrada, por *D* muestras, es decir:

$$y[n]=A_0 x[n-D] \tag{11}$$

el filtro no distorsiona la excitaci6n x[n] al permitir el paso de sus componentes frecuenciales en el rango de interés. Rescribiendo la función de transferencia de la ecuación (6) a partir del concepto planteado en la ecuación (10), tenemos:

$$H(z)=\frac{Y(z)}{X(z)}=A_0 z^{-D} \tag{12}$$

siendo la respuesta en frecuencia o la transformada de Fourier a tiempo discreto del filtro FIR igual a:

$$H(e^{j\Omega}) = H(\Omega) = A_0 e^{-j\Omega D} \tag{13}$$

De la ecuación (13) se observa que la magnitud de $H(e^{j\Omega})$ es constante en el rango de frecuencias para el cual fue diseñado el filtro, y la fase varia linealmente con la frecuencia. Del análisis anterior, la magnitud de la respuesta a la frecuencia de diferentes tipos de filtros digitales se muestra en la Figura 4. Recordar que $H(\Omega)$ es periódica con periodo $2\pi$ y simétrica alrededor de $\pi$. Obsérvese que el rango de frecuencias del filtro es: $-\pi \leq \Omega \leq \pi$ , lo equivale a decir que $-(f_m/2) \leq f \leq (f_m/2)$ para una señal continua. La ganancia del filtro es T con el fin de compensar el factor de atenuación (1/T) generado en el proceso de muestreo.

En la práctica, las respuestas a la frecuencia mostradas en la Figura 4, no se obtienen. Un filtro implementado físicamente tiene bandas de paso, transición y rechazo y no solo frecuencias de corte. Un filtro pasa bajos cuya respuesta a la frecuencia tiene estas características, se muestran en la Figura 5.

EI siguiente paso es obtener los coeficientes del filtro FIR cuya respuesta a la frecuencia cumpla las especificaciones. Asimismo, se observa que el filtro FIR es estable ya que su función de transferencia discreta no tiene polos fuera del círculo unitario, (ver ecuación 9).

**CARACTERISTICAS DE LOS FILTROS FIR Y ALGUNAS CONSIDERACIONES DE DISEÑO**

Para diseñar un filtro FIR es necesario tener en cuenta que la cantidad de coeficientes o duración de la respuesta al impulso del filtro es siempre finita, a diferencia de la repuesta al impulso de su respectivo filtro ideal. Por lo que la respuesta al impulso del filtro FIR exhibirá cierto truncamiento implícito en comparación con la respuesta al impulso del filtro ideal. Este truncamiento se manifiesta en la respuesta en frecuencia del filtro FIR, como un fenómeno de Gibbs (Oppenheim y Willsky, 1983; Oppenheim y Schafer, 1999), el cual produce ondulaciones antes y después de cualquier discontinuidad. Es por ello que se realiza comúnmente en toda aplicación de filtros FIR, el enventanado a través de un número finito de secuencias de $w[n]$ (Proakis y Manolakis; 1996; Oppenheim y Schafer, 1999), de manera de aplanar principalmente los rizos o lóbulos de la banda de rechazo en la respuesta en frecuencia del filtro. En la Figura 6, se muestra una grafica de tal situación y a continuación en la Tabla 1 se muestran las ventanas típicas utilizadas. Puede ampliar información respecto de los filtros digitales FIR en Proakis y Manolakis (1996) o en Oppenheim y Schafer (1999).

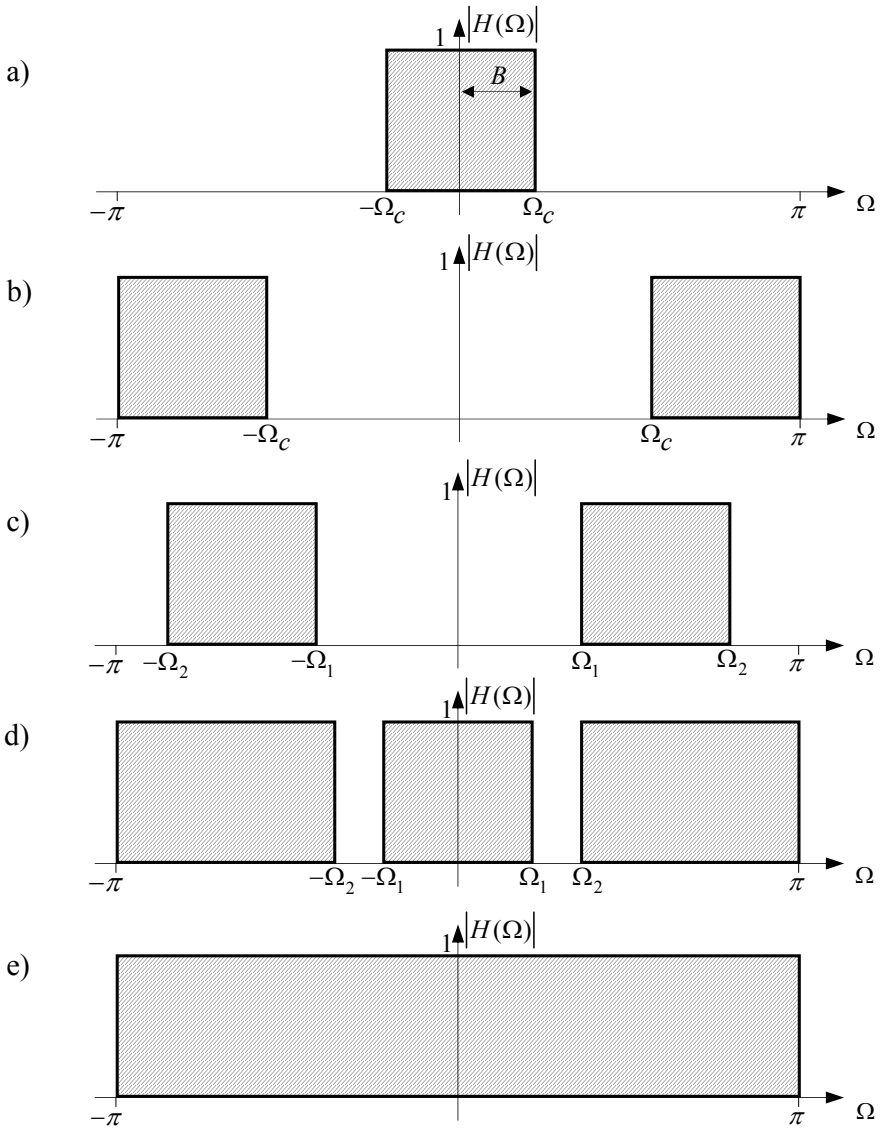



Figura 4: Magnitud de la respuesta a la frecuencia de filtros ideales, a) Pasa bajas, b) Pasa altas, c) Pasa banda, d) Rechazo de banda, e) Rendija ('notch'), f) Pasa todo.

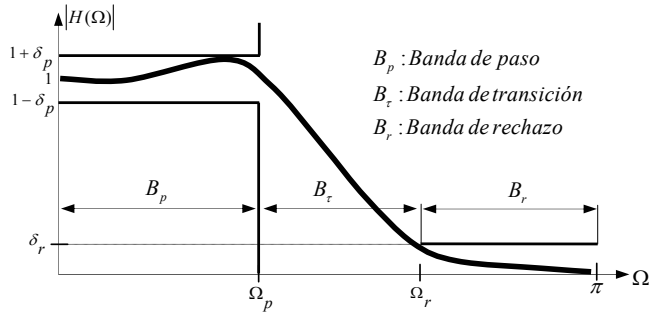



Figura 5: Parámetros de un filtro digital pasa bajas.

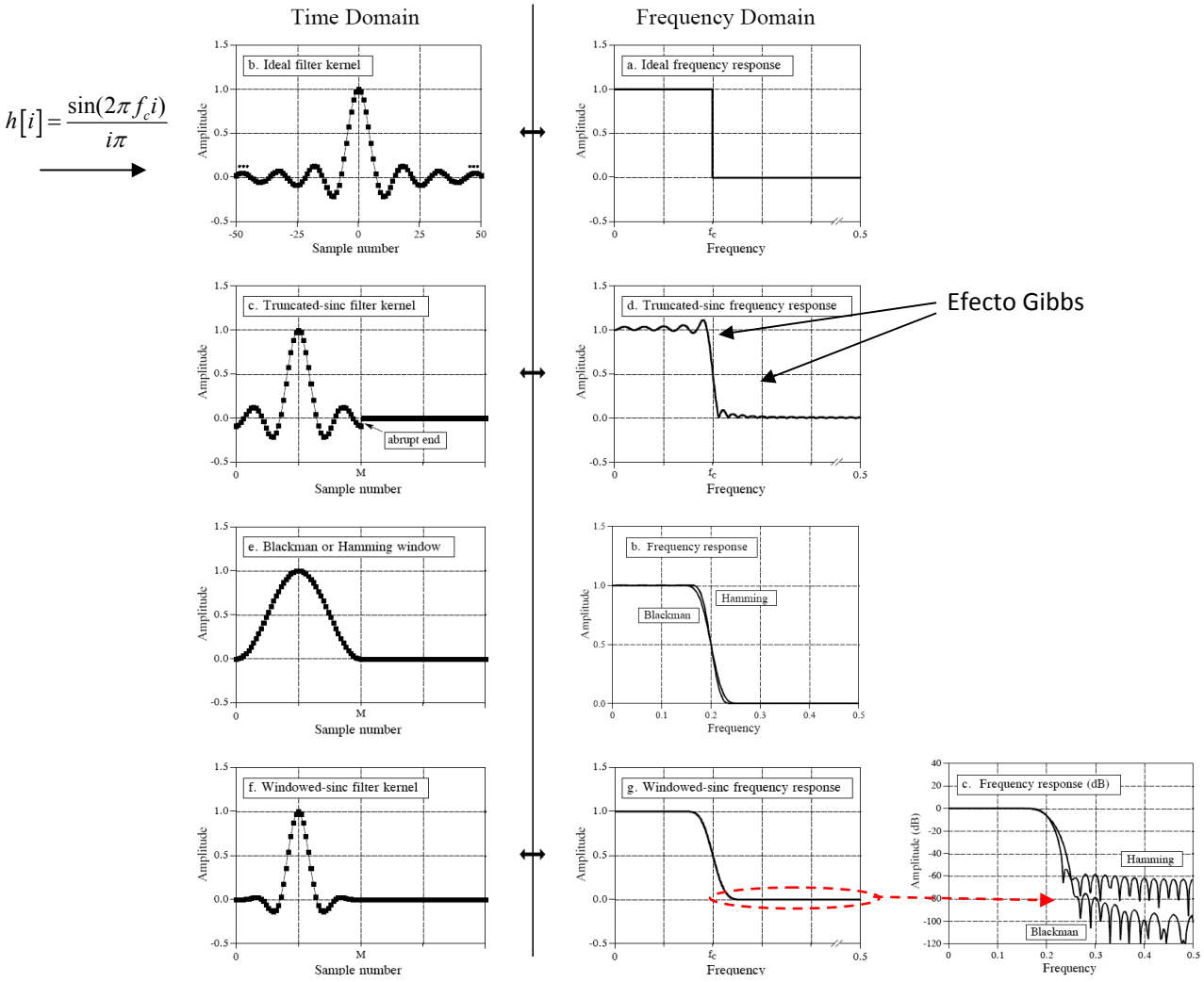



Figura 6: Concepto del diseño de filtros digitaes por ventaneo. Adaptado de Smith (1999)

Tabla 1: Ventanas utilizadas en las operaciones de ventaneo

| VENTANA | EXPRESIÓN |
|---|---|
| **Rectangular** | $w[n]=1 \ para \ 0 \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$ |
| **Barlett o Triangular** | $w[n]=\frac{2n}{N-1} \ para \ 0 \le n \le \frac{N-1}{2}$<br>$w[n]=2-\frac{2n}{N-1} \ para \ \frac{N-2}{2} \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$ |
| **Hann** | $w[n]=\frac{1}{2}\left[1-\cos\left(\frac{2\pi n}{n-1}\right)\right] \ para \ 0 \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$ |
| **Hamming** | $w[n]=0.54-0.46\cos\left(\frac{2\pi n}{N-1}\right) \ para \ 0 \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$ |
| **Blackman** | $w[n]=0.42-0.5\cos\left(\frac{2\pi n}{N-1}\right)+0.08\cos\left(\frac{4\pi n}{N-1}\right) \ para \ 0 \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$ |
| **Kaiser** | $w[n]=\frac{I_0\left[\omega_a\sqrt{\left(\frac{N-1}{2}\right)^2-\left(n-\frac{N-1}{2}\right)^2}\right]}{I_0\left[\omega_a\left(\frac{N-1}{2}\right)\right]} \ para \ 0 \le n \le N-1$<br>$w[n]=0 \ para \ cualquier \ otro \ valor$<br>$siendo \ I_0 \ es \ la \ función \ de \ Bessel \ de \ primer \ tipo \ de \ orden \ cero$ |

Para resumir, hay tres métodos básicos para diseñar los filtros FIR:

- Método de las ventanas. Las más habituales son:
  - Ventana rectangular
  - Ventana de Barlett
  - Ventana de Hann
  - Ventana de Hamming
  - Ventana de Blackman

- Ventana de Kaiser

- Muestreo en frecuencia.
- Rizado constante (Aproximación de Chebyshov y algoritmo de intercambio de Remez).

Las características principales de los filtros FIR es que no cuentan con retroalimentación, por lo cual cuenta con mayores localidades de memoria. Por otro lado, estos filtros son siempre estables, por lo general son de fase lineal y son mas fácil de implementar que in filtro IIR. Este tipo de filtros tiene especial interés en aplicaciones de audio.

A la hora de realizar el diseño de los filtros, se busca tener cierto comportamiento ya sea en el dominio del tiempo como en el dominio de la frecuencia. Dicho comportamiento comienza por una "cuestión estética", que se traduce en la estabilidad, distorsión y ubicación espacial de los ceros del filtro. En las Figuras 7 y 8, se muestran las apariencias de diferentes respuestas en el tiempo y en la frecuencia, relacionadas a un filtro pasa bajos digital y que deben ser tenidas en cuenta a la hora de realizar el diseño del filtro. Una descripción detallada del tema puede encontrarse en Orfanidis (1996).

**NUMEROS FRACIONARIOS EN LA FAMILIA DSP56800**

Por lo general en el lenguaje C, los números fraccionales son tratados con codificación de punto flotante, sin embrago la arquitectura DSP en general tienen implementado, a nivel de la lógica de la ALU, operaciones con codificación fraccional de punto fijo. De esta manera CodeWarrior permite declarar variables de este tipo con una sintaxis especial en la declaración ( *__fixed__* , palabra de 16 bit ) , para que el compilador traduzca directamente las operaciones de multiplicación en instrucciones de máquina que las implementan. De esta manera para implementar rutinas como filtros o tratamiento sobre datos fraccionales se prefiere este tipo de codificación que potencia el uso de la naturaleza de la arquitectura. En la Figura 9 se muestra un esquema de la estructura binaria del tipo de codificación numérica que soportan los DSP de la familia DSP56800.

Es claro de la codificación, que esta es equivalente a la codificación entera complemento dos (variables shrort y int a nivel de C), para implementación de suma, resta, shift, operaciones sobre bit, por lo tanto no existen instrucciones de máquina distintas para actuar sobre este tipo de variables en las operaciones antes descritas. Un escenario distinto es en el caso de la multiplicación pues los números que se representan con notación fraccional son menores que 1 en magnitud (Ej: 0.5 x 0.5 = 0.25; 0.9 x 0.9 = 0.81). De esta manera al multiplicar dos números enteros se necesitan los bit de la parte de la codificación de los bits más significativos para representar dicho resultado correctamente. Sin embargo al multiplicar dos números fraccionales se necesita mayores bits adicionales en la zona de los bits menos

significativos para poder representar dicha operación. De esta manera a nivel de lenguaje de máquina existen instrucciones especializadas para multiplicar números con codificación entera o fraccional (ver Tabla 1). En la Figura 10 se muestra un diagrama de la lógica implementada en la ALU del core que permite implementar multiplicación de punto fijo o entera.

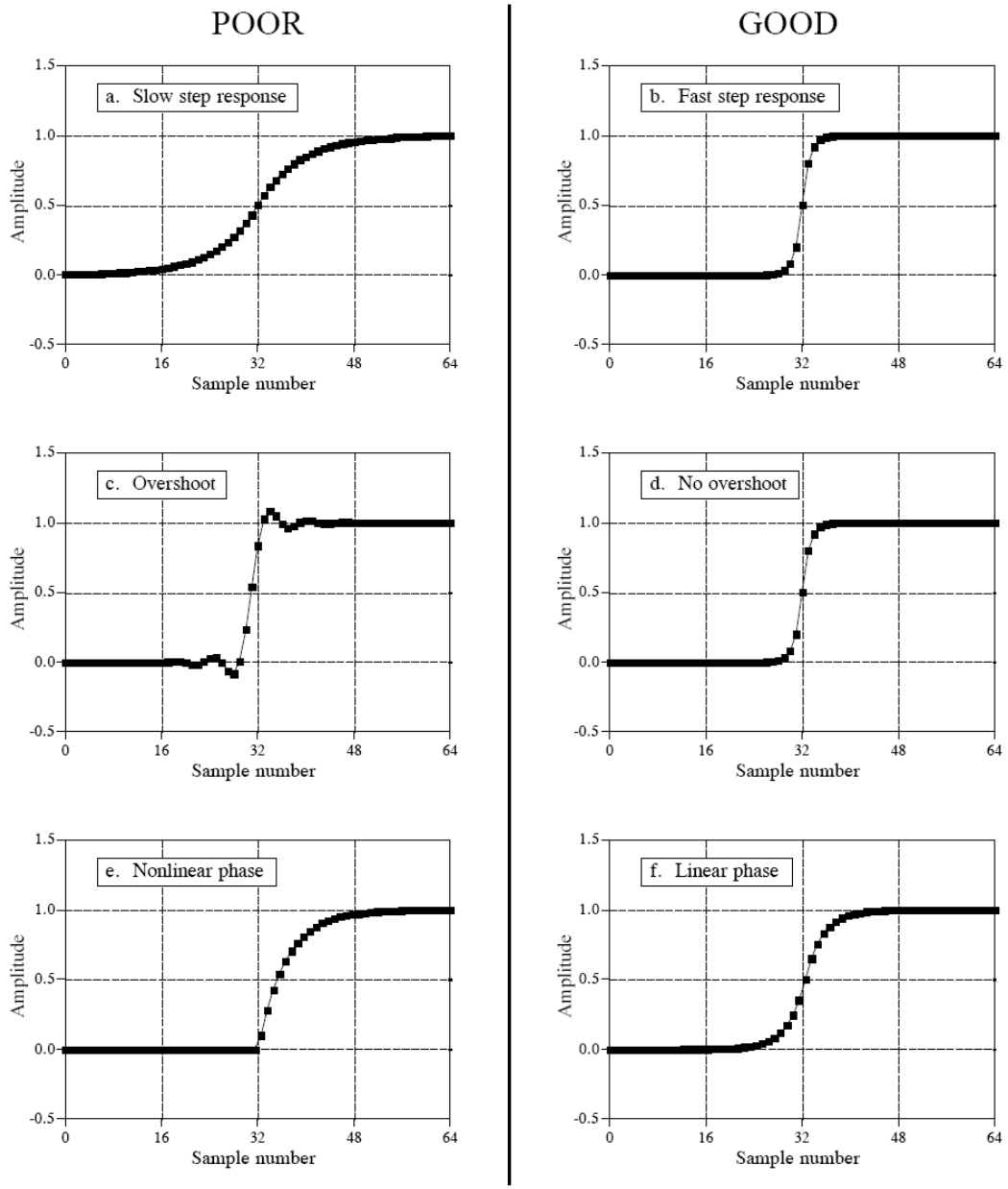



Figura 7: Distinción entre un buen y un mal comportamiento de un filtro digital, en lo que respecta a diferentes características de la respuesta temporal del filtro. Adaptado de Smith (1999)

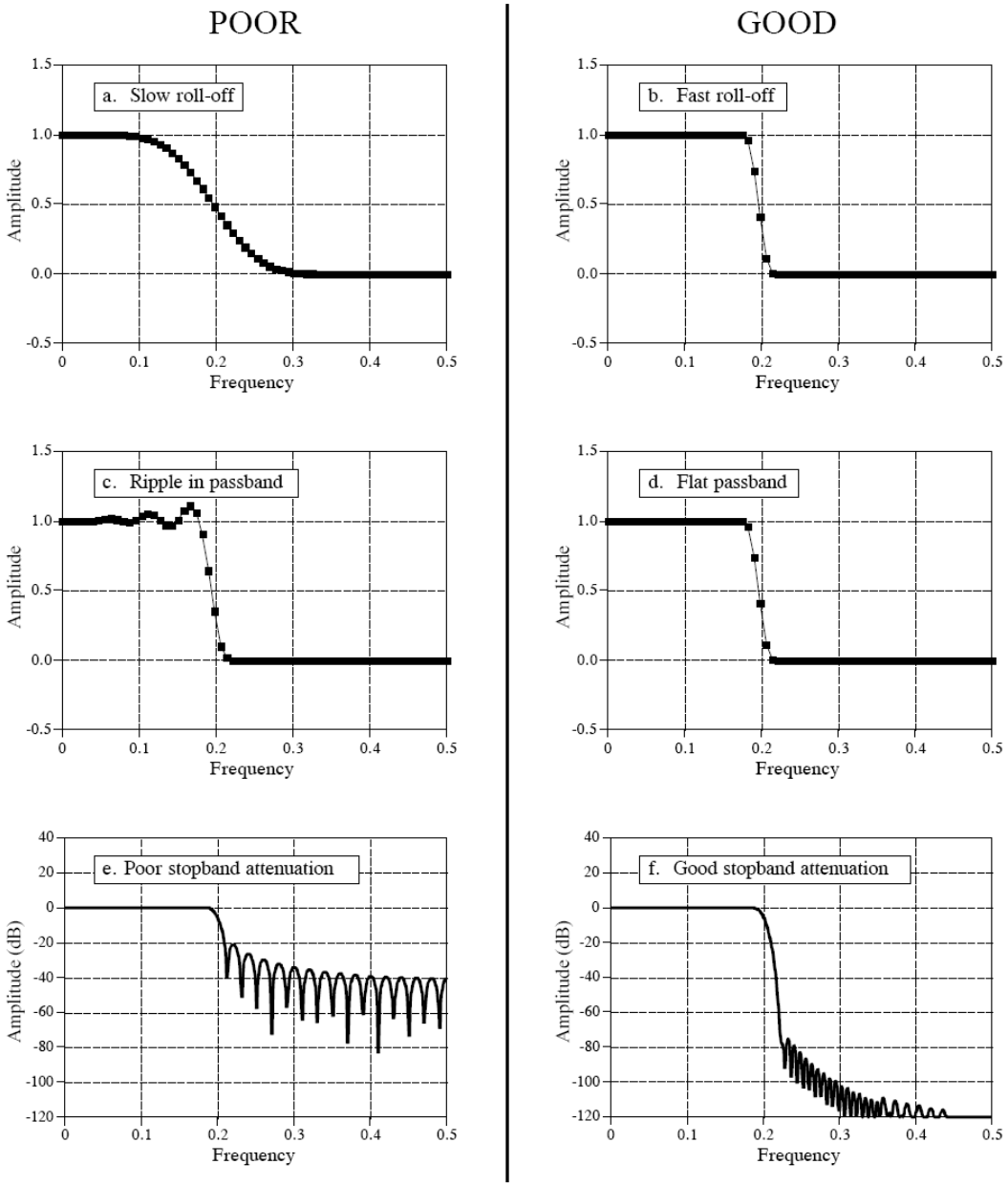



Figura 8: Distinción entre un buen y un mal comportamiento de un filtro digital, en lo que respecta a diferentes características de la respuesta en frecuencia del filtro. Adaptado de Smith (1999)

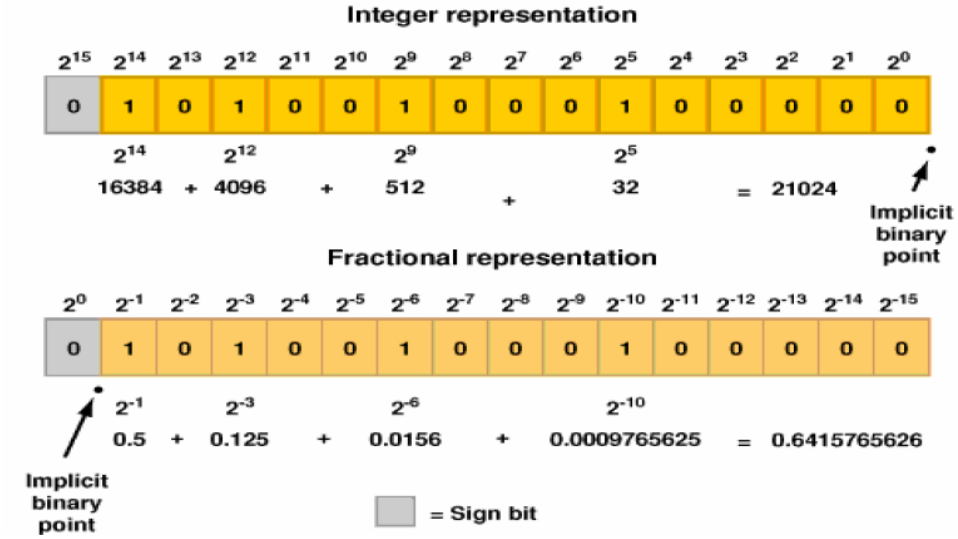



Figura 9: Codificación entera y fraccional que soporta el core DSP56800

Tabla 1: Operaciones de multiplicación dependientes de la codificación de palabras en la familia DSP56800.

| | | | |
|---|---|---|---|
| ENTERA | DEC(W) | Decrementar una palabra | Used primarily in integer arithemtic. |
| | IMPY(16) | Multiplicar un entero | Result must fit in 16 bits or exception occurs. |
| | INC(W) | Incrementar una plabra | Used primarily in integer arithmetic. |
| FRACCIONAL | MAC | Multiplicar y acumular | Perform ASR after operation to obtain integer result. Only useful for fractional data when destination register is a word. |
| | MACR | Multiplicar y acumular con redondeo | |
| | MPY | Multiplicación con signo | Perform ASR after operation to obtain integer result. Only useful for fractional data when destination register is a word. |
| | MPYR | Multiplicación con signo y redondeo | |

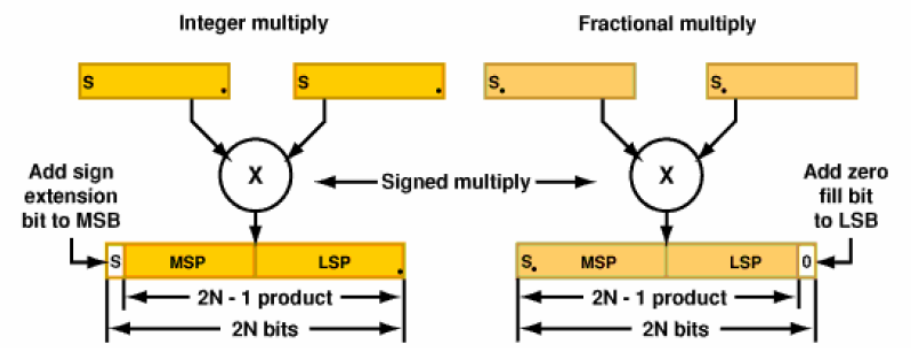



Figura 10: Diagrama de la lógica implementada en la ALU del core que permite implementar multiplicación de punto fijo o entera (ambas con signo).

**IMPLEMENTACIÓN DEL FILTRO FIR**

En este punto mostraremos como implementar un filtro digital FIR en la familia DSP56800. Básicamente se implementará un filtro FIR (de respuesta al impulso finita) el cual se realiza vía la convolución que permite obtener la relación de entrada salida a través de la ecuación (8).

De esta manera este tipo de filtros se implementa en términos de ecuaciones de diferencias, donde la salida n-ésima es la ponderación de las entradas (*x[n], ..x[n-N]*). Los coeficientes ponderadores *h[m]* corresponden a la respuesta al impulso del filtro.

Los siguientes ejemplos corresponde al algoritmo, implementado en lenguaje C, de la convolución entre dos vectores y de la implementación del algoritmo del filtro FIR en forma directa para procesamiento por muestras “Sample Processing” (Orfanidis, 1996). Notar que la definición de las variables son del tipo “double” y recordar que la familia DSP56800 es de punto fijo:

```
 /* conv.c - convolution of x[n] with h[n], resulting in y[n] */


#include <stdlib.h>             /* defines max() and min() */

void conv(M, h, L, x, y)
double *h, *x, *y;              /* h,x,y = filter, input, output arrays */
int M, L;                       /* M = filter order, L = input length */
{
        int n, m;
        for (n = 0; n < L+M; n++)
                for (y[n] = 0, m = max(0, n-L+1); m <= min(n, M); m++)
                        y[n] += h[m] * x[n-m];
}
```

```
/* fir.c - FIR filter in direct form */

double fir(M, h, w, x)        /* Usage: y = fir(M, h, w, x); */
double *h, *w, x;             /* h = filter, w = state, x = input sample */
int M;                        /* M = filter order */
{
       int i;
       double y;              /* output sample */

       w[0] = x;              /* read current input sample x */

       for (y=0, i=0; i<=M; i++)
              y += h[i] * w[i];       /* compute current output sample y */

       for (i=M; i>=1; i--)           /* update states for next call */
              w[i] = w[i-1];          /* done in reverse order */

       return y;
}
```

El algoritmo anterior puede implementarse sin ningún problema en un DSP, sin embargo debido a la configuración o estructura produce un considerable tiempo de procesamiento. Esto es debido a que se utilizan dos ciclos “for” en forma separada. Orfanidis (1996), muestra que uno puede implementar o desarrollar un código en C que contemple las características de funcionamiento de la MAC, de manera de optimizar el código para sacar provecho de las cualidades del DSP. De esta manera si consideramos el algoritmo de procesamiento de un filtro de tercer orden que se lleva a cado en un DSP, por cada muestra que se adquiere se realiza lo siguiente:

$w_0 := x$
$w := h_3\, w_3$
$w_3 := w_2$
$y := y + h_2\, w_2$
$w_2 := w_1$
$y := y + h_1\, w_1$
$w_1 := w_0$
$y := y + h_0\, w_0$

Esto se desarrolla de esta manera debido a las arquitecturas típicas de los DSP (ver Figura 11).

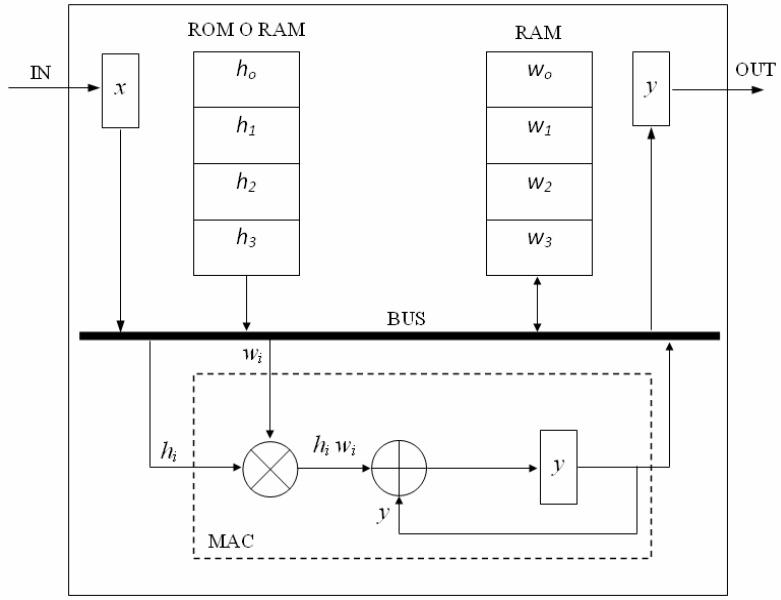



Figura 11: Típica arquitectura de un DSP en la implementación de un filtro. Orfanidis (1996).

A continuación se muestra el ejemplo de cómo se implementaría en lenguaje C, una función que realice el filtrado por muestras, teniendo en cuenta la arquitectura del DSP de punto flotante.

```
/* fir3.c - FIR filter emulating a DSP chip */

double fir3(M, h, w, x)
double *h, *w, x;
int M;
{
        int i;
        double y;

        w[0] = x;                                   /* read input */

        for (y=h[M]*w[M], i=M-1; i>=0; i--) {
                w[i+1] = w[i];                      /* data shift instruction */
                y += h[i] * w[i];                   /* MAC instruction */
                }

        return y;
}
```

De la misma manera que se puede implementar los algoritmos de filtros en lenguaje C, se pueden realizar en el lenguaje Assembler del DSP56800. En el Apéndice B, del manual DSP56800 Family Manual, se muestran algunos ejemplos de código. A continuación se muestran los códigos en el lenguaje Assembler de los algoritmos utilizados para el filtrado:

***Real Correlation or Convolution (FIR Filter)***

```
; c(n) = SUM(I=0,...,N-1) { a(I) * b(n-I) }

      opt   cc
      MOVE  #N_,N                              ; 2   2    load size of vector
      MOVE  #A_Vec1,R0                         ; 2   2    pointer to vector
      MOVE  #B_Vec1,R3                         ; 2   2    pointer to vector
      CLR   A       X:(R0)+,Y0                 ; 1   1    clear & load 1st element in A
      MOVE          X:(R3)+,X0                 ; 1   1    load 1st element in B
      REP   N                                  ; 1   3    repeat until done
      MAC   Y0,X0,A X:(R0)+,Y0  X:(R3)+,X0     ; 1   1    correlation and load new val
      RND   A                                  ; 1   1    rounding the result
;                                              ----------
;                              Total:          11   1N+12
```

***Complex Correlation Or Convolution (Complex FIR)***

```
; cr(n) + jci(n) = SUM(I=0,...,N-1)
; { ( ar(I) + jai(I) ) *  ( br(n-I) + jbi(n-I) ) }
; cr(n) = SUM(I=0,...,N-1)                                  Y0=ar  Y1=br
; { ar(I) * br(n-I) - ai(I) * bi(n-I) }
; ci(n) = SUM(I=0,...,N-1)                                  Y0=ai  X0=bi
; { ar(I) * bi(n-I) + ai(I) * br(n-I) }
      opt   cc
      MOVE  #N_,N                              ; 2   2
      MOVE  #A_Vec3,R0                         ; 2   2
      MOVE  #B_Vec3,R3                         ; 2   2
      CLR   A           X:(R0)+,Y0             ; 1   1     ar and clear result
      CLR   B           X:(R3)+,Y1             ; 1   1     br and clear result
      DO    N,EndDO1_3                         ; 2   3
      MAC   Y0,Y1,A     X:(R3)+,X0             ; 1   1     ar*br, get next bi
      MAC   Y0,X0,B     X:(R0)+,Y0             ; 1   1     ar*bi, get next ai
      MAC   Y0,Y1,B     X:(R3)+,Y1             ; 1   1     ar*bi+ai*br, next br
      MAC   -Y0,X0,A                           ; 1   1     ar*br-ai*bi
      MOVE              X:(R0)+,Y0             ; 1   1     get next ar
EndDO1_3:
      RND   A                                  ; 1   1
      RND   B                                  ; 1   1
;                                              ----------
;                                  Total:      17    5N+13
```

Recuerde que la hora de implementar el código de su proyecto en CodeWarrior, uno puede realizar la optimización del código C (ver Figura 12), de manera que este se asemeje o no a la estructura o logia de programación con el lenguaje Assembler y de esta manera aprovechar las características de la arquitectura del DSP. Recuerde además, lo planteado en los párrafos anteriores en donde uno puede implementar un algoritmo en código C orientado a las características particulares de la arquitectura del DSP.

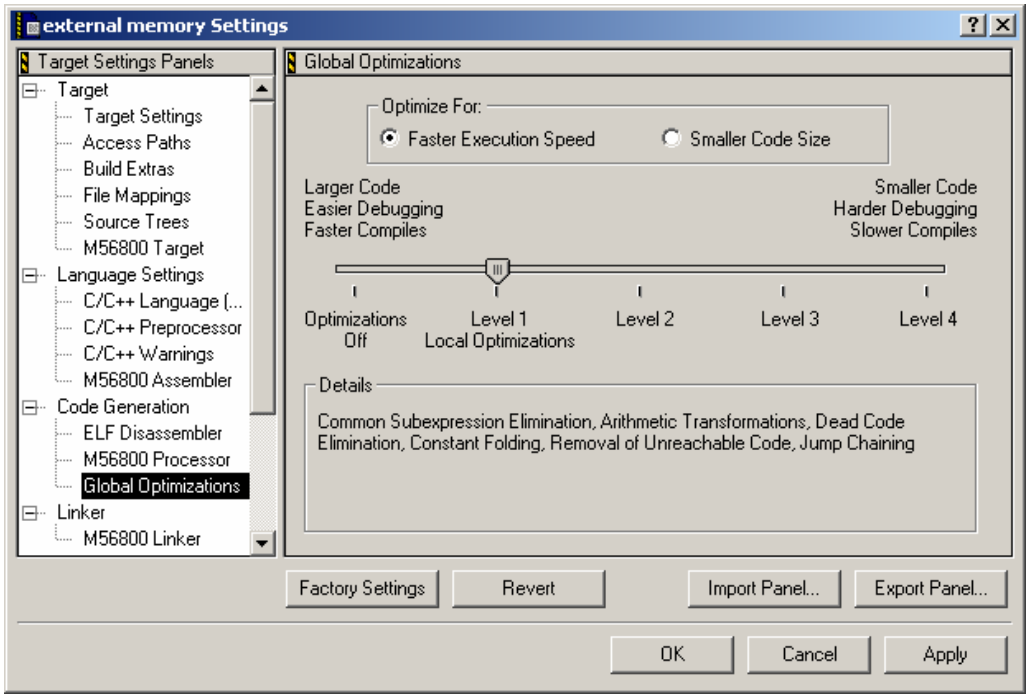



Figura 12: Ventana de configuración de las optimizaciones globales

**BIBIOGRAFIA**

DSP56800 Family Manual. 16-Bit Digital Signal Controllers. DSP56800FM Rev. 3.1 11/2005. Freescale.

Smith, S. W., 1999. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing, Second Edition.

Oppenheim, A. V. and Schafer, R. W., 1999. Discrete-Time Signal Processing. 2nd Ed. Prentice-Hall Signal Processing Series.

Proakis, J. G. and Manolakis, D. G., 1996. Digital Signal Processing: Principles, Algorithms and Applications. Third Edition. Prentice Hall Inc.

Oppenheim, A. and Willsky, A. S., 1983. Signals and Systems. Prentice Hall Inc.

Higgins, R. J., 1990. Digital Signal Processing In Vlsi. Prentice-Hall,Inc. New Jersey.

Orfanidis, S. J., 1996. Introduction to Signal Processing. Prentice may, Inc.