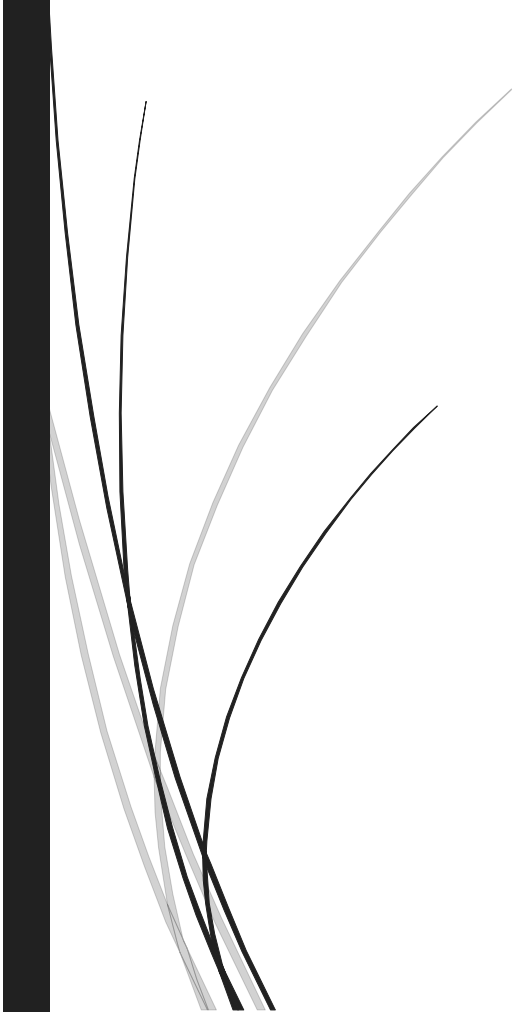
DSP

Punto fijo, punto flotante y filtros digitales



Julian Camargo
TDIII

REPRESENTACIÓN FINITA DE NÚMEROS REALES EN PUNTO FIJO

Los programadores deben ajustar la representación de números reales para cada aplicación particular.

Enteros sin signo

Una palabra binaria de N bits puede representar un total de 2^N valores en un rango de 0 a $2^N - 1$.

$$n_{10} = 2^{N-1}b_{N-1} + 2^{N-2}b_{N-2} + \cdots + \cdots 2\, b_1 + 2^0 b_0$$

La base de tiempo de cualquier reloj de sistemas binarios debe ser en un numero base 2.

Enteros con signo complemento a dos

Típicamente en una computadora los números se representan en complemento a dos por razones de eficiencia de hardware, ya que con así puede usarse un **sumador** para sumar y restar. El rango de va de -2^{N-1} a $2^{N-1} - 1$.

$$n_{10} = -b_{N-1}\, 2^{N-1} + \sum_{i=0}^{N-2} b_i\, 2^i$$

Esto nos dice que todos los números comenzados con 1 son negativos.

Patrón de bits	Entero sin signo	Complemento a 2
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
-	-	-
-	-	-
0111 1110	126	126
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
-	-	-
-	-	-
1111 1110	254	-2
1111 1111	255	-1

Representación de números en C

Existen los siguientes tipos de enteros en el lenguaje de programación C:

- **8 bits** (char, int8_t): [-128, 127]
- **16 bits** (short, int16_t): [-32768, 32767]
- **32 bits** (int, long, int32_t): [-2147483648, 2147483647]

Representación en Punto fijo

Un número real x es representado por un entero X con $N = m + n + 1$ bits, donde:

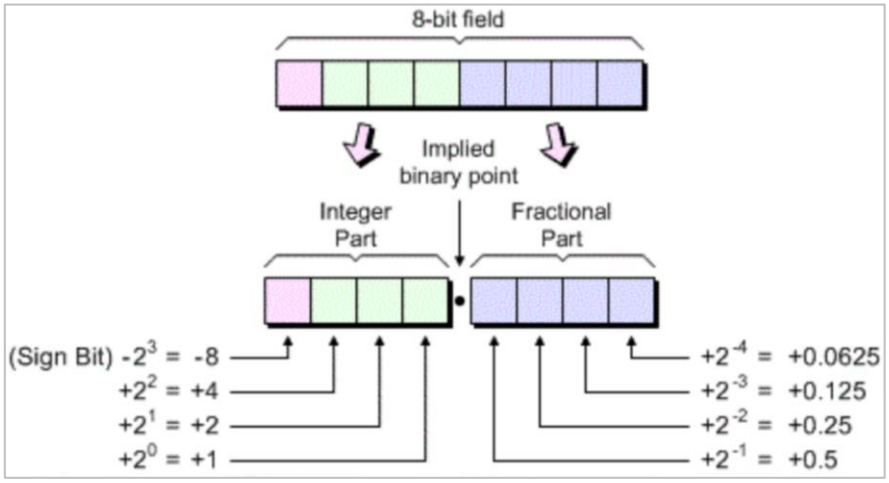
- N es la longitud de la palabra
- m el número de bits que representan la parte entera

- n el número de bits que representan la parte fraccional
- Un bit extra para representar el signo

En punto fijo se consideran las potencias negativas para representar la parte fraccional $\left(2^{-1} = \frac{1}{2}, 2^{-2} = \frac{1}{4} \ldots\right)$. La formula para pasar un número punto fijo a real en base 10 es:

$$n_{10} = -b_m\, 2^m + \left(\sum_{i=0}^{m-1} b_i 2^i + \sum_{i=1}^{n} b_i 2^{-i}\right)$$

El primer bit b_m determina el signo, la primer sumatoria la parte entera (potencias de dos) y la segunda la parte fraccional (potencias negativas de dos).



Se denomina *punto fijo* debido a que al seleccionar una cantidad de bits para la parte fraccional, ésta no varía de tamaño, es decir, el punto no se mueve.

Al elegir la cantidad de bits para la parte entera y la parte decimal se definen dos parámetros que determinan el rango de presentación, estos son:

- Precisión: 2^{-n} (siendo n los bits de la parte decimal)
- Rango: -2^m a $2^m - 2^{-n}$

Si $n = 0$, estamos en el caso de complemento a dos. Por ello, se dice que complemento a dos es un caso particular de punto fijo.

NOTACIÓN Q

Es una forma de expresar rápidamente la cantidad de bits que representan la parte entera y la parte decimal. No es un estándar de la IEEE. Algunos ejemplos son:

- **Q0.15 (Q15):**
  - 16 bits (15 para el numero y 1 para el signo)
  - 0 bits para la parte entera y 15 bits para la parte decimal
  - Rango: -1 a 0.99996948
  - Precisión: $\frac{1}{32786} = 2^{-15}$
- **Q3.12:**
  - 16 bits
  - 3 bits para la parte entera y 12 para la parte decimal
  - Rango: -8 a 7.9998

- Precisión: 2^{-12}
- **Q0.31 (Q31):**
  - 32 bits, 31 para la parte fraccional y 1 de signo
  - Rango: -1 a 0.999999999534339
  - Precisión: 2^{-31}

CONVERSIÓN A Y DESDE PUNTO FIJO

- Representación de un 1 en punto fijo: tomamos el número 1 y agregamos tantos bits a su derecha como bits tenga la parte decimal.

  $$z = 1 \ll n = 1 \cdot 2^n$$

  Esta operación en lenguaje C es un desplazamiento a izquierda.
- Representación de $\frac{1}{2}$ en punto fijo: en este caso tomamos un 1 y lo desplazamos $n-1$ lugares.

  $$z = 1 \ll (n-1) = 1 \cdot 2^{n-1}$$

Conversión de un valor real "x" desde punto flotante a punto fijo usando casteo

La expresión en punto flotante de la computadora es la más cercana a un número en formato real. Para pasarlo a un formato en punto fijo utilizamos alguna de las siguientes expresiones (equivalentes):

```
X = (int) (x * (1 << n))

X = (int) (x * 2 ^ n)
```

Conversión de un valor en punto fijo a punto flotante

La operación es similar. Tomamos el valor en punto fijo y lo dividimos por 1 desplazado n veces a la izquierda, que es lo mismo que multiplicar por 2^{-n}. Al resultado lo casteamos a *float*, que es un número en punto flotante de 32 bits:

```
x = (float) (X) / (1 << n)

x = (float) (X) * 2 ^ (-n)
```

FACTOR DE ESCALA

La ALU de una CPU no distingue entre números en formato entero o punto fijo. Solo el programador conoce el tipo de formato con el que está operando la máquina. Esto es porque el programador no tiene forma de indicar si una variable está en formato punto fijo o en formato int, en realidad ambas variables se definen como int.

De esta diferencia nace el concepto de factor de escala. Se dice que este factor solo está presente en la cabeza del programador. La ALU solo opera con números enteros, podemos entonces considerar que un número en punto fijo es un entero multiplicado por un factor de escala. Si usamos la notación $Qm.n$, 2^{-n} es la precisión.

El factor de escala puede ser un valor arbitrario, no necesariamente debe ser potencia de dos.

Ejemplo: se desea utilizar números de 16 bits para representar valores decimales de entre -5 y 5.

Esta transformación es conveniente realizarla en 3 pasos:

1. Primero tomamos los números enteros, que van de **-32768** a **32767** (16 bits).
2. A este rango de números lo multiplicamos por $\mathbf{2^{-15}}$, obteniendo un rango que va desde **-1 a 0.99999**…
3. Ahora a esto lo multiplicamos por **5**, obteniendo finalmente el rango deseado de **-5 a 4.99984741211**.

El factor de escala y la precisión son lo mismo, y en este ejemplo valen $\mathbf{5 \cdot 2^{-15}}$.

RANGO DINÁMICO

El rango dinámico se define como:

$$DR_{dB} = 20\log_{10}\left(\frac{mayor\ valor\ representable}{menor\ valor\ representable}\right)[dB]$$

Nos da una idea de cuantos números podemos representar con cierta cantidad de bits. Si operamos podemos llegar a una ecuación aproximada más rápida:

$$DR_{dB} = 20\log_{10}\left(\frac{2^{N-1}-1}{1}\right)[dB]$$
$$DR_{dB} \cong 20\,[(N-1)\log_{10}(2)]$$
$$DR_{dB} \cong 20\log_{10}(2)\cdot(N-1)$$
$$DR_{dB} \cong 6.02\cdot(N-1)\,[dB]$$

En la siguiente tabla se observan diferentes valores de rango dinámico y precisión para distintos formatos de punto fijo ($Qn.m$):

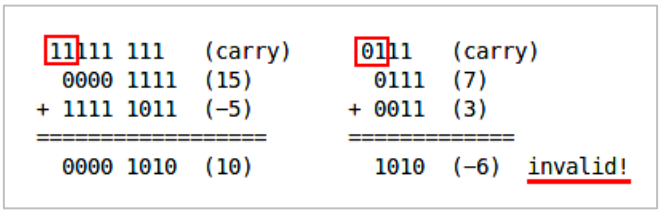
Format (N.M)		Largest positive value (0x7FFF)	Least negative value (0x8000)	Precision (0x0001)		DR(dB)
0	15	0,999969482421875	-1	3,05176E-05	2^-15	90,30873362
1	14	1,99993896484375	-2	6,10352E-05	2^-14	90,30873362
2	13	3,9998779296875	-4	0,00012207	2^-13	90,30873362
3	12	7,999755859375	-8	0,000244141	2^-12	90,30873362
4	11	15,99951171875	-16	0,000488281	2^-11	90,30873362
5	10	31,99902344	-32	0,000976563	2^-10	90,30873362
6	9	63,99804688	-64	0,001953125	2^-9	90,30873362
7	8	127,9960938	-128	0,00390625	2^-8	90,30873362
8	7	255,9921875	-256	0,0078125	2^-7	90,30873362
9	6	511,984375	-512	0,015625	2^-6	90,30873362
10	5	1023,96875	-1024	0,03125	2^-5	90,30873362
11	4	2047,9375	-2048	0,0625	2^-4	90,30873362
12	3	4095,875	-4096	0,125	2^-3	90,30873362
13	2	8191,75	-8192	0,25	2^-2	90,30873362
14	1	16383,5	-16384	0,5	2^-1	90,30873362
15	0	32767	-32768	1	2^-0	90,30873362

Todos los rangos dinámicos son iguales, debido a que todos los formatos representados en la tabla usan 16 bits.

Aritmética de números reales en punto fijo

- La suma de dos números de N bits puede producir un resultado de hasta N+1 bits.
- El resultado tendrá siempre la misma cantidad de bits en la parte decimal.
- Solo la parte entera puede crecer.

ADICIÓN - OVERFLOW



En el ejemplo de la izquierda se realiza la suma de dos números complemento a 2 de 8 bits. Al ser de 8 bits se pueden representar valores desde -128 a 127.

Al realizar la suma vemos que los últimos dos bits del acarreo son iguales (dos unos), esto significa que el resultado es válido.

En el ejemplo de la derecha no pasa lo mismo. Vemos que los dos bits más altos de acarreo son distintos. Esto produce un resultado de signo diferente al de los operandos, generando un absurdo matemático. Vemos que el resultado es invalido. Esto se denomina **Overflow**.

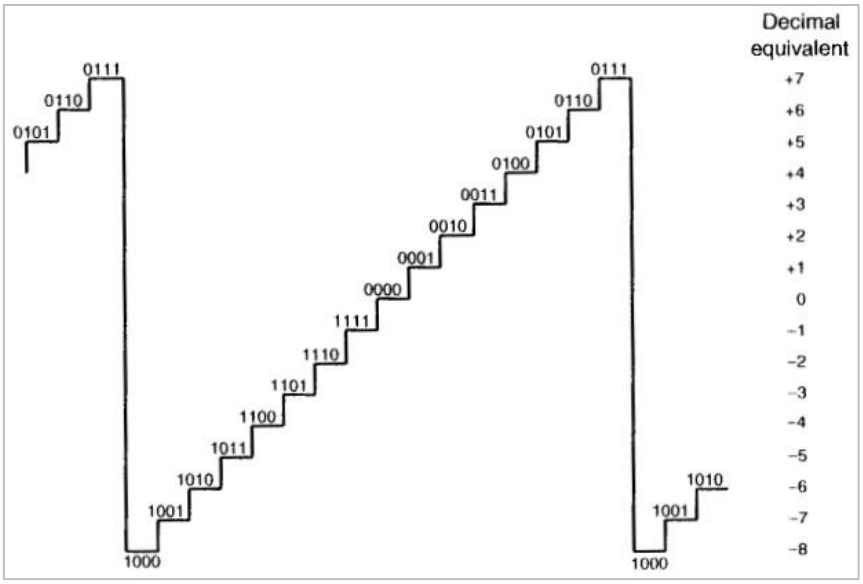
Esto sucede porque son números de 4 bits, que representan valores entre -8 y 7, y la suma resultante (7+3) es un valor mayor a 7.

Las computadoras pueden detectar si ha ocurrido Overflow justamente leyendo los dos últimos bits de acarreo. Si estos son iguales, no hubo Overflow, caso contrario si lo hubo.

La arquitectura de la computadora tiene un bit o Flag destinado para indicar si se produjo Overflow.

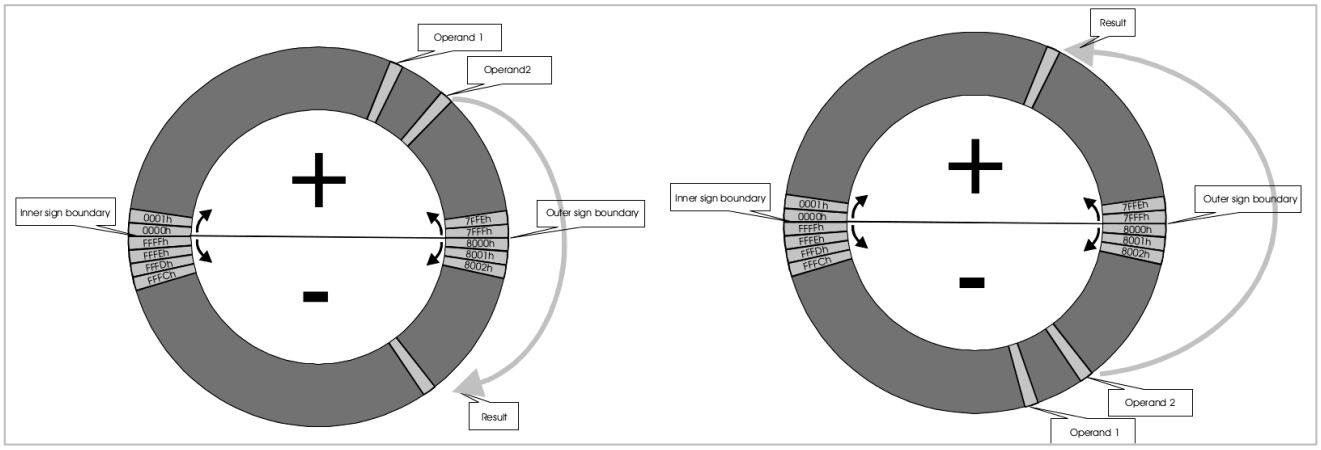
En definitiva, el Overflow se produce cuando al realizar una operación el número resultante sale del rango de los números en dicha operación. Puede suceder al sumar dos números positivos o al restar dos negativos.

Overflow suele denominarse también **roll-over** (darse vuelta). Esto es porque, como se ve en la siguiente imagen, al exceder el rango, el número resultante pasa al otro extremo del mismo.



En el ejemplo anterior, el resultado de 10 quedaba representado por un -6 porque al exceder el límite superior de 7 del rango, el siguiente número es -8, el siguiente -7 y el siguiente -6.

Al sumar dos números muy positivos, el resultado, debido al Overflow es negativo. Lo mismo para dos valores muy negativos, el resultado es positivo.



EVITAR OVERFLOW

- **Acumulador de n+1 bits:** la forma más sencilla de evitar Overflow es guardando el resultado de la operación de dos números de N bits en un acumulador de N+1 bits.
  La regla general indica que la suma de **s** números individuales de **m** bits puede requerir un acumulador de como mucho $m + \log_2(\boldsymbol{s})$ bits.
  Por ejemplo, para sumar 256 palabras de 8 bits, necesitamos un acumulador de $8 + \log_2(256) = 16$ bits para asegurar que no ocurra Overflow.
  Generalmente un procesador DSP de 16 bits tiene un acumulador ALU de 40 bits. Basándonos en la regla general podemos calcular cuantos número de 16 bits pueden sumarse sin correr riesgo de Overflow:

$$16 + \log_2(\boldsymbol{s}) = 40$$

$$\log_2(s) = 40 - 16$$

$$2^{\log_2(s)} = 2^{24}$$

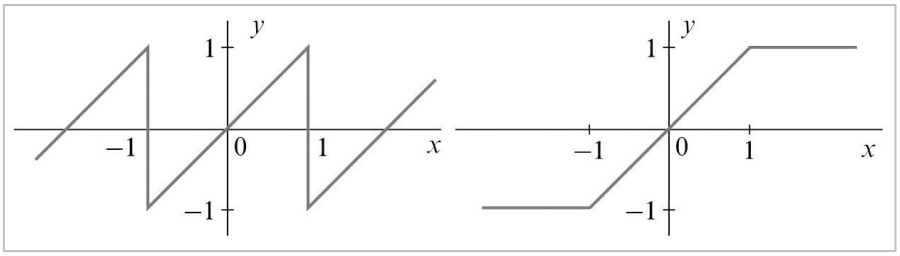
$$s = 16777216$$

Ejemplo de implementación de acumulador de mayor tamaño en C:

```
int32_t a[K];
int64_t c = 0;

for (i=0; i<K; i++){
      c = c + (int64_t) a[i];
}
```

Se usa una variable *c* de 64 bits con signo para acumular los *K* valores de 32bits con signo del vector *a*.

- **Aritmética de saturación:** otra forma de evitar Overflow es implementar **aritmética de saturación**. En este caso los valores que exceden el rango toman como valor dicho límite. En el caso de los números de 4 bits cuyo resultado era 10, con aritmética de saturación el resultado quedaría en 7.
  La curva de roll-over (izquierda) queda modificada de la siguiente forma:



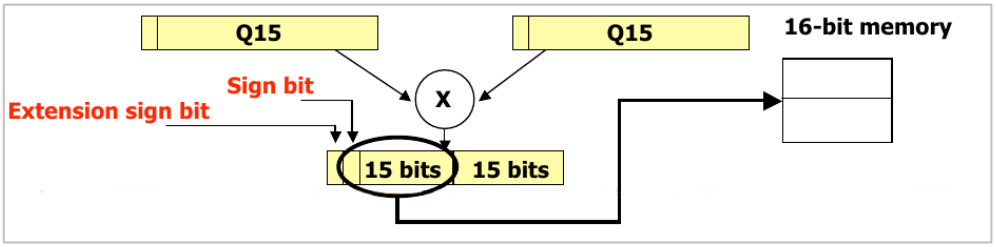
De este modo la situación con overflow pasa de ser catastrófica a simplemente inconveniente. Debe tenerse cuidado al utilizar esta técnica ya que si en gran parte de la operación nos encontramos en la parte truncada o saturada, podemos convertir un sistema lineal en un sistema no lineal.

Algunos procesadores DSP permiten implementar la técnica de aritmética de saturación por hardware, por lo que el programador puede olvidarse de ella. Sin embargo, en otros procesadores, como ARM o el de computadores personales, la aritmética de saturación debe implementarse a través de funciones que debe desarrollar el programador.

MULTIPLICACIÓN

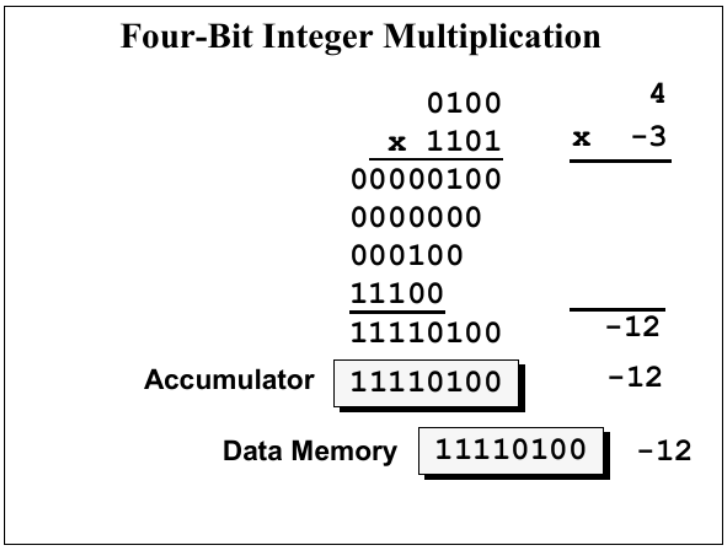
Al multiplicar dos números de N bits, se requiere como mucho de 2N bits para representar todos los posibles valores del resultado.

En la siguiente imagen se representa la multiplicación de dos números en formato Q15 (1 bit de signo y 15 bits para representar la parte decimal).



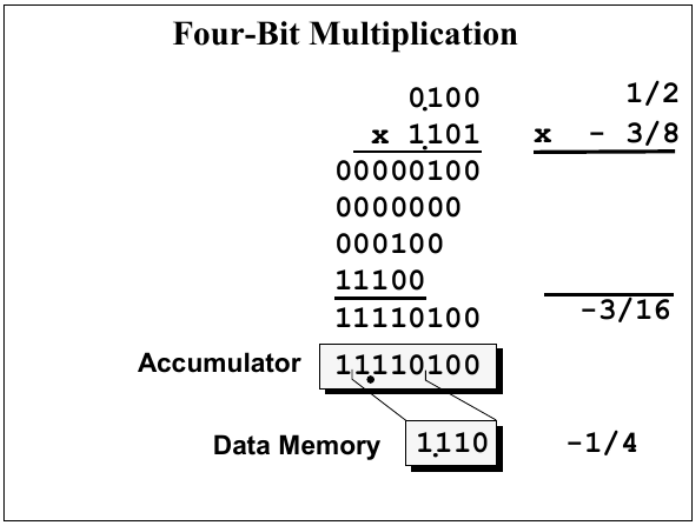
Vemos que el resultado ahora tiene 30 bits para la parte decimal, un bit de signo y un bit de extensión de signo. Este último es eliminado por la ALU.

Debe tenerse en cuenta que la precisión del valor resultado se incrementó de 2^{-15} a 2^{-30}, es decir, es un número mucho más exacto. En algún momento puede decidirse que este resultado de 32 bits debe almacenarse en memoria. En la figura esta memoria es de 16 bits, por lo que el programador debe plantearse qué hacer con los bits extra.



En este ejemplo se multiplican dos enteros de 4 bits. El resultado en el acumulador es de 8 bits y se almacena en memoria con los 8 bits, sin descartar ninguno.

En este ejemplo se multiplican dos números en formato *Q0.3* (1 bit de signo y 3 para la parte decimal). El patrón de bits es igual al del ejemplo anterior, pero en este caso cada posición de bit representa un valor diferente.

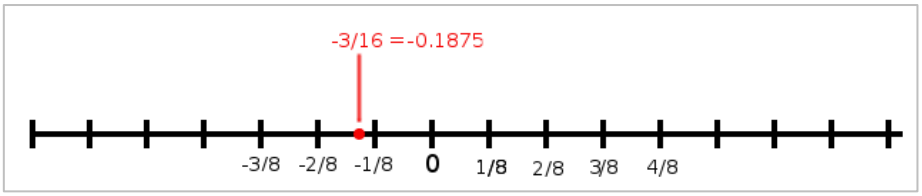


Se obtiene en el acumulador un número de 8 bits que representa al valor $-\frac{3}{16}$.

Se decide solo tomar 4 bits del resultado para volver al formato *Q0.3*. Para ello se descarta el primer bit (extensión de signo) y se toman solo los siguientes 4 bits. El resultado guardado en memoria es 1110 y representa al valor $-\frac{1}{4}$. Se pasó entonces de un valor igual a -0.1875 a uno de -0.25 (disminuyó la precisión).

Al fenómeno de disminuir la precisión luego de realizar una multiplicación se lo conoce como **underflow**.

En la siguiente figura se observan los valores que podemos representar con un formato *Q0.3*:



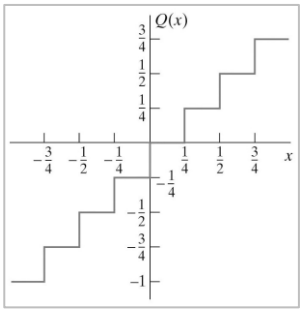
Como $-\frac{3}{16}$ no posee representación en *Q0.3* debe elegirse qué valor debe tomar ($-\frac{1}{8}$ o $-\frac{2}{8}$). A los métodos que definen cual de dos posibles valores debe tomar el resultado de una multiplicación luego de producirse underflow se los conoce como **esquemas de redondeo**. A continuación se detallan dos esquemas de redondeo.

- **Truncación o redondeo a menos infinito:** al resultado x de una multiplicación le aplicamos la función truncación Q y lo almacenamos en y. Donde y tendrá el mismo formato que los valores multiplicados (a y b).

$$x = a \cdot b \quad , \quad y = Q(x)$$

Este método elimina los bits extra que se generaron en la parte decimal del resultado luego de la multiplicación. A continuación se muestra una posible implementación en C:

```
int32_t truncation(int64_t X){

	int32_t a;

	a = (int32_t) (X>>n);

	return a;

}
```



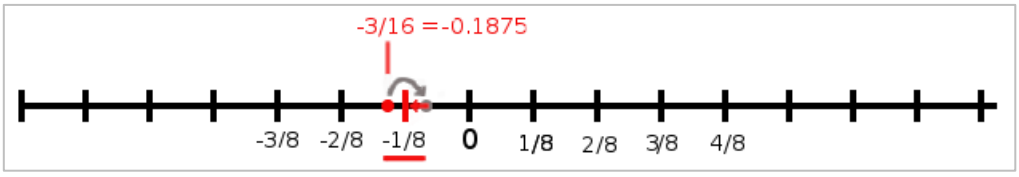
En este ejemplo, la función truncación recibe un número X de 64 bits y se eliminan los bits extra de la parte decimal. Se realiza un desplazamiento de n bits a la derecha en el valor X, este resultado se castea a 32 bits y se guarda en la variable a.

A la derecha se observa de manera gráfica la precisión de la función truncación de números en punto fijo cuya precisión es igual a $\frac{1}{4}$. Los infinitos valores comprendidos en un intervalo, toman el valor inmediato inferior (redondeo a $-\infty$).

- **Round-OFF o redondeo al más cercano:** al valor x resultante de multiplicar los valores a y b, se le suma $2^{-(n+1)}$ y luego se le aplica la truncación.

$$x = a \cdot b \quad , \quad y = Q\left(x + 2^{-(n+1)}\right)$$

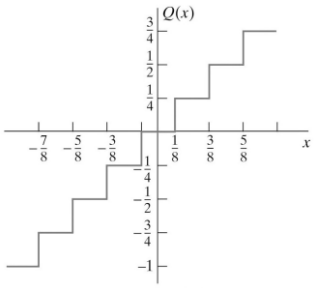
El valor $2^{-(n+1)}$ es la mitad de la precisión.



Así, valores más cercanos al superior del intervalo, al sumarle la mitad de la precisión lo rebasan, por lo que al aplicar luego la truncación se igualan a dicho valor. Si el valor es más

próximo al inferior del intervalo de precisión, al sumarle la mitad de la precisión, no rebasan el valor superior del intervalo, por lo que al truncar se igualan al inferior.

```
int32_t roundoff(int64_t X){

        int32_t a;

        a = ( X + ( 1 << (n - 1) ) ) ;

        return truncation(a);

}
```

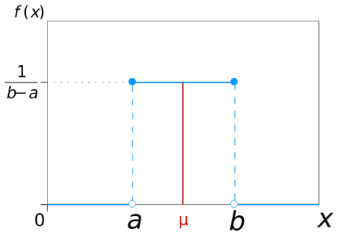


En este ejemplo, la función recibe un número X de 64 bits producto de una multiplicación y lo lleva a uno de 32. Primero suma media precisión del formato original de los operandos. Para la suma de media precisión se le adiciona a X un 1 en la posición n-1.

- **Comparación de los dos métodos:**

mean, $\mu = \frac{a+b}{2}$

variance, $\sigma^2 = \frac{(b-a)^2}{12}$



- Truncation: $e = Q(x) - x, \quad -2^{-n} \le e < 0, \quad \boxed{\mu = -\frac{2^{-n}}{2}} \quad \boxed{\sigma^2 = \frac{2^{-n}}{12}}$
- Round-off: $e = Q\left(x + 2^{-(n+1)}\right) - x, \quad -\frac{2^{-n}}{2} < e \le \frac{2^{-n}}{2}, \quad \boxed{\mu = 0} \quad \boxed{\sigma^2 = \frac{2^{-n}}{12}}$

Vemos que el que menor sesgo produce es *round-off* ($\mu = 0$).

OPERACIÓN MAC

La operación MAC significa Multiplicar y Acumular. Esta operación implementa la convolución entre dos señales y es esencial en el cálculo de la transformada rápida de Fourier (optimizacion de la transformada discreta de Fourier).

El funcionamiento básico de la operación MAC es realizar la multiplicación de dos vectores elemento a elemento y acumular este resultado.

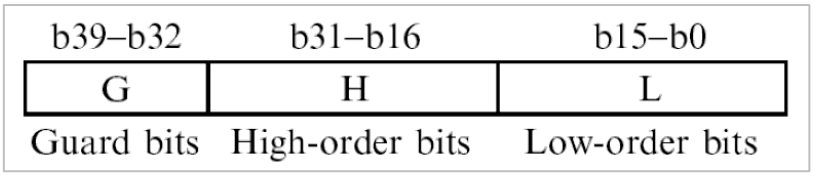
La operación MAC incluye operaciones de suma y multiplicación, por lo que resulta susceptible a overflow y underflow.

Ejemplo de implementación en C:

```
int32_t a[K];

int32_t b[K];

int64_t c = 0;

for (i = 0 ; i<K ; i++ ){

c = c + ( (int64_t) a[i] * (int64_t) b[i]);

}
```

La operación MAC es la operación básica al trabajar con procesamiento digital de señales, por ello, los procesadores DSP están optimizados para implementar esta operación. Generalmente la realizan en un ciclo de reloj y cuentan con un acumulador en su ALU que permite sortear los problemas de overflow y underflow de mejor manera.

La siguiente figura representa un esquema del acumulador de un procesador DSP de Texas Intrument (TI DSP C55505, acumulador de 40 bits).



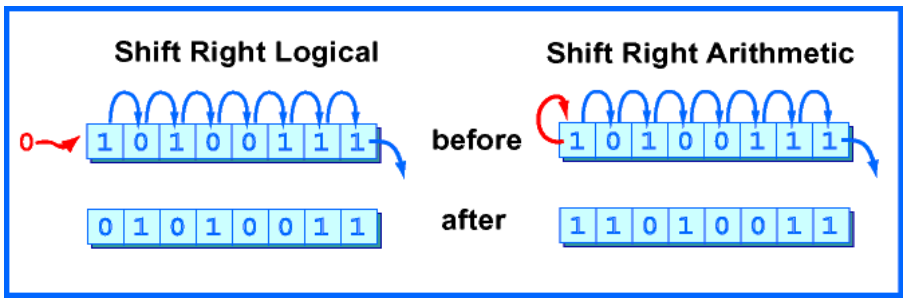
Este opera con números de 16 bit y su acumulador posee 40 bits, 32 para almacenar resultados de multiplicación y 8 extras para manejar los potenciales overflow.

SHIFTS (DESPLAZAMIENTO)

Desplazar un número binario a la izquierda implica multiplicarlo por dos y desplazarlo a la derecha implica dividirlo por dos. Desde el punto de vista del esfuerzo de cálculo es mucho más sencillo hacer un desplazamiento que multiplicar o dividir por dos.

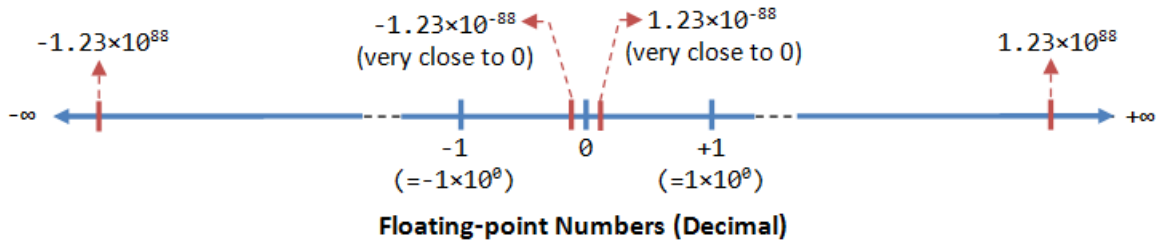
Los desplazamientos se dividen en **desplazamientos lógicos** y **desplazamientos aritméticos**.

Un desplazamiento lógico hacia la derecha consiste en desplazar todos los bits a la derecha una posición y rellenar el más significativo con un cero. Esto trae un problema en aritmética complemento a dos si el número es negativo, ya que estaríamos modificando su signo. Por ello es que existe el desplazamiento aritmético, en el cual se desplaza el número pero en el bit más significativo se coloca el mismo bit que había anteriormente.



REPRESENTACIÓN FINITA DE NÚMEROS REALES EN PUNTO FLOTANTE

La característica más importante es que puede representar una gran cantidad de números.



Floating-point Numbers (Decimal)

Esto se logra representando los números con el siguiente formato:

$$(-1)^S \cdot F \cdot r^E$$

Donde:

- S es un bit de signo
- F es la parte fraccional o mantisa
- r es la base del número y E el exponente (desplazamiento)

El estándar adoptado mayormente por las computadoras actuales en su FPU es el IEEE 754. Este contempla punto flotante en base 2 y 10. Dentro del estándar de base 2 tenemos números de 16, 32, 64 y 128 bits.

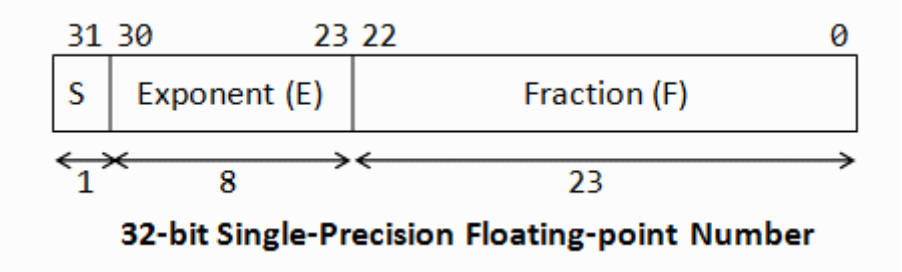
	Binary formats ($B = 2$)				Decimal formats ($B = 10$)		
Parameter	Binary 16	Binary 32	Binary 64	Binary 128	Decimal 132	Decimal 164	Decimal 128
p, digits	$10 + 1$	$23 + 1$	$52 + 1$	$112 + 1$	7	16	34
e_{max}	+15	+127	+1023	+16383	+96	+384	+16,383
e_{min}	−14	−126	−1022	−16382	−95	−383	−16,382
Common name	Half precision	Single precision	Double precision	Quadruple precision			

El estándar también cubre esquemas de redondeo, operaciones aritméticas, funciones trigonométricas y manejo de excepciones.

Formato de 32 bits – Single precision

Es el que más se usa en sistemas embebidos, ya que estos difícilmente utilicen números de 64 bits o más.

En el siguiente diagrama vemos como se posicionan los bits en una palabra de 32 bits. El primero es el de signo, le siguen 8 bits para el exponente y los restantes 23 para la parte fraccional.



$$(-1)^S \cdot F \cdot r^{(E-bias)}$$

Donde:

- S es el bit de signo, **0** para números positivos y **1** para negativos. En este bit se encuentra el 1 implícito.
- E es el exponente de 8 bits. Puede tomar valores entre 1 y 254 (los valores 0 y 255 están reservados).
- F es la parte fraccional, tiene 23 bits y representa potencias negativas de 2: $[2^{-1}\ 2^{-2} \dots 2^{-23}]$
- $bias$ es el sesgo y para este formato (single precisión) vale **127**, por lo que el exponente de r puede tomar valores entre: $-126 \leq (E - bias) \leq 127$. Esto se hace para evitar colocar un bit de signo en el exponente.

PROBLEMA DEL ESTÁNDAR

Este formato de punto flotante para representar números tiene el inconveniente de no tener una representación única para todos los números. Por ejemplo:

El número 13.25 puede representarse mediante el formato visto de las siguientes maneras:

$$1101.01_2 \cdot (2^0) = 110.101_2 \cdot (2^1) = 11.0101_2 \cdot (2^2) = 1.10101_2 \cdot (2^3)$$

El formato también **establece** que la única representación válida de un número es aquella cuya parte fraccional comienza con "*1.* ". Es decir, la parte entera de la mantisa debe ser 1. En el ejemplo, la representación válida es la última $(1.10101_2 \cdot (2^3))$.

El campo F tiene un 1 implícito, el cual no es parte de los 32 bits que conforman al número. Se forma entonces con el campo F la siguiente representación**:** $\mathbf{1}.\boldsymbol{F} = \mathbf{1}.[\mathbf{2^{-1}}\ \mathbf{2^{-2}} \dots \mathbf{2^{-23}}]$.

A esta manera de representar un número en punto flotante la IEEE la denomina **forma normalizada**.

Ejemplo 1:

Valor decimal	3215.020002	
Bit de signo	Campo exponente	Fracción
0	1000 1010	1.10010001111000001010010
Valor representado	3215.0200	

Ejemplo 2:

Valor decimal	$3215.020002 \cdot 2 = 6430.040004$	
Bit de signo	Campo exponente	Fracción
0	1000 1011	1.10010001111000001010010
Valor representado	6430.0400	

De la observación de múltiples ejemplos de representaciones en punto flotante podemos llegar a las siguientes conclusiones:

1. La multiplicación y división por 2 corresponde a incrementar y decrementar en 1 el exponente.
2. No pueden representarse todos los números reales, independiente de la cantidad de bits del formato.
3. Los números flotantes son **autorango**.

AUTORANGO

Se dice que los números en punto flotante son autorango porque para un determinado valor de exponente ($E - bias$) tenemos un rango de posibles números representables. Esto se ve en la siguiente tabla:

2^E	MIN F 1..00000000000000000000000 (b2) 1 (b10)	MAX F 1..11111111111111111111111 (b2) 1.999999881 (b10)
2^-126	1.1755E-38	2.3510E-38
2^-30	9.3132E-10	1.8626E-09
2^-20	9.5367E-07	1.9073E-06
2^-10	9.7656E-04	1.9531E-03
2^-3	0.12500000	0.24999999
2^-2	0.25000000	0.49999997
2^-1	0.50000000	0.99999994
2^0	**1.00000000**	**1.99999988**
2^1	2.00000000	3.99999976
2^2	4.00000000	7.99999952
2^3	8.00000000	15.99999905
2^10	1.0240E+03	2.0480E+03
2^20	1.0486E+06	2.0972E+06
2^30	1.0737E+09	2.1475E+09
2^127	1.7014E+38	3.4028E+38

Estos tramos o rangos de números van creciendo conforme aumenta el valor del exponente.

FORMA DE – NORMALIZADA

Vimos que el formato en punto flotante posee un 1 implícito ubicado delante de los 23 bits del campo F. Esto trae el inconveniente de no poder representar el valor **0**.

Para evitar este inconveniente se considera que sí $E = 0$, en lugar de un 1 implícito, el número tiene un 0 implícito. En este caso el exponente $E - bias$ es igual a -127, por lo que el número finalmente queda:

$$0.F = 0.[2^{-1}\ 2^{-2} \dots 2^{-23}]$$

Ejemplo:

Valor decimal	$-3.4 \cdot 10^{-39}$	
Bit de signo	Campo exponente	Fracción
1	0000 0000	0.10010100000101110100 01
Valor representado	$-3.39999999 \cdot 10^{-39}$	

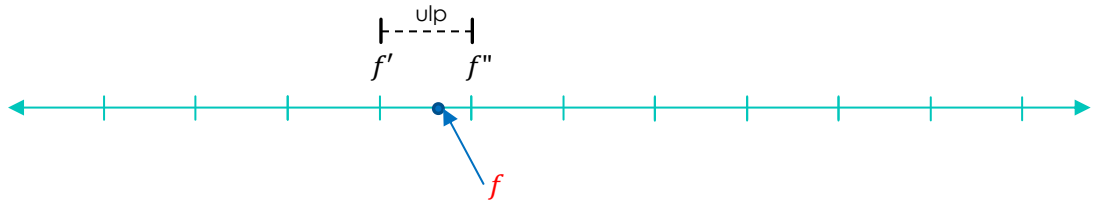
Valores especiales

La norma de la IEEE contempla 3 valores especiales, estos son:

- **Cero**: $E = 0$, $F = 0$. Dos posibles representaciones: $+0$ $(S = 0)$ y -0 $(S = 1)$.
- **Infinito**: $E = 0xFF$, $F = 0$. Dos posibles representaciones: $+Inf$ $(S = 0)$, $-Inf (S = 1)$.
- **NaN (Not a Number)**: $E = 0xFF$. $F \neq 0$. Valor que no puede ser representado como un número real, por ejemplo $\frac{0}{0}$.

Esquemas de redondeo

En el siguiente esquema, el valor f no tiene una representación exacta en punto flotante y ha caído entre dos valores f' y $f"$ que si tienen representación en punto flotante. La distancia entre estos últimos dos números se denomina **ulp** (unit of least precision) y es equivalente a la denominada precisión en punto fijo.



El valor de f será igual a f' o a $f"$ según el esquema de redondeo que se utilice. El estándar de la IEEE soporta 4 esquemas de redondeo:

- **Truncación**: también denominado redondeo a cero.
  - Si f es positivo, $round(f) = f'$
  - Si f es negativo, $round(f) = f"$
- **Redondeo a más infinito:** $round(f) = f"$
- **Redondeo a menos infinito:** $round(f) = f'$
- **Redondeo al valor más cercano:** por defecto utilizado por la FPU
  - Si $f < f' + \frac{ulp}{2}$, $round(f) = f'$
  - Si $f \geq f' + \frac{ulp}{2}$, $round(f) = f"$

Rango dinámico

Indica la cantidad de números que podemos representar con un formato determinado. Matemáticamente:

$$DR_{dB} = 20 \log_{10} \left(\frac{mayor\ número\ representable}{menor\ número\ representable} \right) [dB]$$

Para números flotantes se define como:

$$DR_{dB} \cong 6.02 \cdot 2^{b_E}$$

Donde b_E es el número de bits del exponente. Para el formato simple precisión (32 bits) este valor es de:

$$DR_{dB} = 6.02 \cdot 2^8 \cong 1541\ dB$$

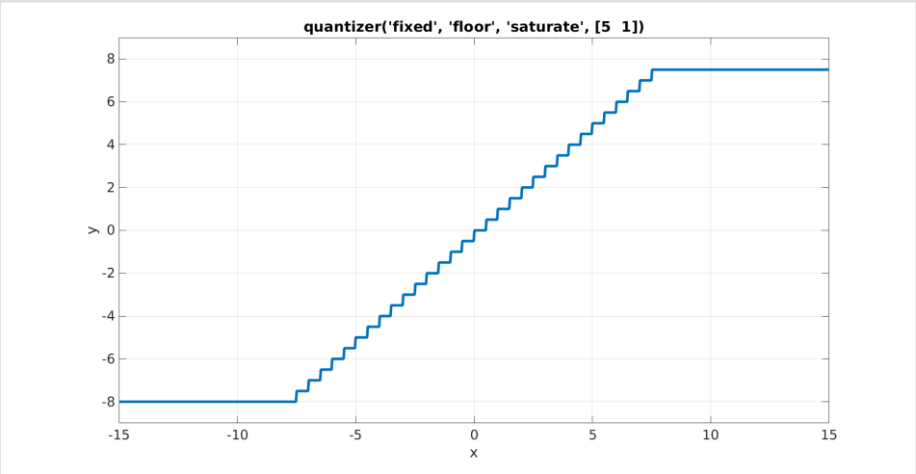
Si lo comparamos con un número de 32 bits en punto fijo, este tiene un rango dinámico de:

$$DR_{dB} = 6.02 \cdot 31 \cong 186\ dB$$

Es evidente que con punto flotante pueden representarse mucha mayor cantidad de números con formatos de la misma cantidad de bits.

Precisión en punto fijo

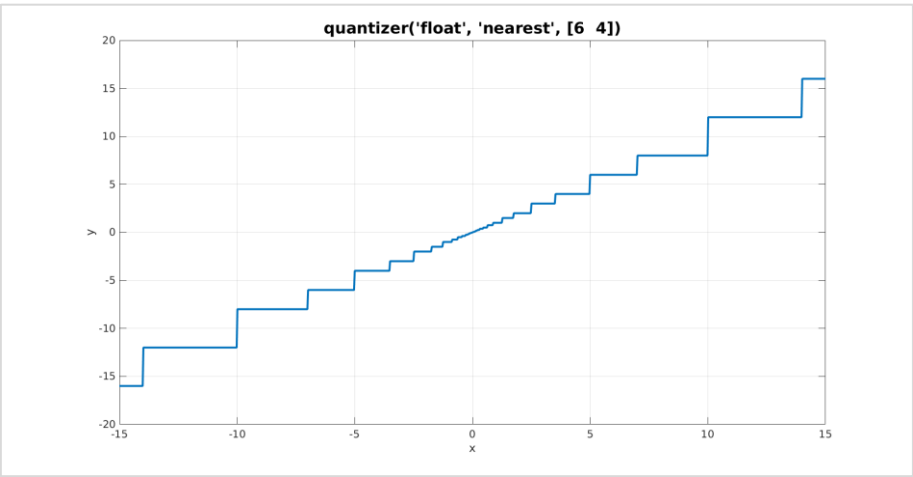
La precisión es la distancia entre dos posibles números en un sistema de numeración. En punto fijo esta es 2^{-n}, donde n es la cantidad de bits usados para representar la parte fraccional. La precisión es constante mientras n no sea modificado.



La forma de escalera está dada por la precisión, cada escalón vale 0.5, que es la distancia entre dos valores representados para este formato. Se observa entonces que la precisión es constante en todo el rango.

Precisión en punto flotante

En punto flotante la precisión se define como $2^{E} \cdot 2^{-23}$, donde E representa el exponente efectivo, es decir, $E - bias$. Observamos entonces que la precisión no es constante para todo el rango de números en punto flotante que podemos



Se observa ahora que la precisión varía en todo el rango. Cerca del cero, la precisión es mínima (exactitud máxima) y a medida que nos acercamos a los extremos la precisión crece.

Que la precisión sea variable y tenga una variación tan significativa, tiene impactos importantes al operar con números en punto flotante.

En el siguiente ejemplo de Matlab se observan dos errores típicos provocados por esta variación significativa de la precisión al trabajar con números flotantes.

En el **primer ejemplo**, al trabajar con dos números tan disimiles como 2^{53} y 1, la precisión del primer número es mayor a 1, por lo que el simple calculo que debería dar 1, para la maquina es 0.

En el **segundo ejemplo** ocurre lo mismo, al operar con números muy disimiles, la precisión puede ser tal que la operación arroje resultados absurdos.

MATLAB

```
» a = (2^53 + 1) - 2^53
» a = 0
» if (a == 0)
»    disp('Turn off nuclear reactor')
» else
»    disp('Do not turn off nuclear reactor')
```

```
» x = 0;
» t = tan(x) - sin(x)/cos(x)
» t = 0
» x = 1;
» t = tan(x) - sin(x)/cos(x)
» t = 2.2204e-16 % eps(1)
```

Suma de dos números en punto flotante

NÚMEROS DE RANGO SIMILAR

Se realizará la suma $0.5 + (-0.4375)$ usando 4 bits para la mantisa.

- **Representación en punto flotante**:
  - $0.5_{10} = 0.1000_2 \cdot 2^0 = 1.0000_2 \cdot 2^{-1}$ (normalizado)
  - $-0.4375_{10} = -0.0111_2 \cdot 2^0 = -1.1100_2 \cdot 2^{-2}$(normalizado)
- **Igualación de exponentes al mayor**:
  - El número de mayor exponente es $0.5_{10} = 1.000_2 \cdot 2^{-1}$.
  - Se desplaza al n veces otro valor, $-0.4375_{10} = -1.1100_2 \cdot 2^{-2}$
  - Resultando: $-0.4375_{10} = -1.1100_2 \cdot 2^{-2} = -0.1110_2 \cdot 2^{-1}$
- **Suma de las mantisas**:
  - $(1.0000_2 - 0.1110_2) \cdot 2^{-1} = 0.0010_2 \cdot 2^{-1}$
- **Normalización del resultado**:
  - $0.0010_2 \cdot 2^{-1} = 1.0000_2 \cdot 2^{-4} = \mathbf{0.0625}$
  - Se verifica que $-126 < -4 < 127$, en este caso no hay overflow ni underflow en la operación.
- **Redondeo**:
  - En este caso, la suma se ajusta en 23 bits, por lo que no es requerido.

NÚMEROS DE RANGO MUY DIFERENTE

Se realizará la suma de $1e10 + 1500$ usando el formato precisión simple de IEEE-754 (32 bits):

- **Representación en punto flotante**:
  - $10{,}000{,}000{,}000_{10} = 1.00101010000001011111100 1_2 \cdot 2^{33}$ (normalizado)
  - $1500_{10} = 1.01110111000000000000000000000000_2 \cdot 2^{10}$(normalizado)
- **Igualación de exponentes al mayor**:
  - El mayor exponente es 33.
  - Se desplaza al n veces otro valor para que tenga el mismo exponente

- Resultando: $1500_{10} \cong 0.00000000000000000000000000000001_2 \cdot 2^{23} = 1.0 \cdot 2^{10} = \mathbf{1024}$

- **Suma de las mantisas**:
  - $(1.00101010000001011111100 1_2 - 0.00000000000000000000000000000001_2) \cdot 2^{33}$
- **Normalización del resultado**:
  - $1.00101010000001011111101 0_2 \cdot 2^{33} = 1.16415333710_{10} \cdot 2^{33} = \mathbf{10,000,001,024}$
  - Se verifica que $-126 < 33 < 127$, en este caso no hay overflow ni underflow en la operación.
- **Redondeo**:
  - En este caso, la suma se ajusta en 23 bits, por lo que no es requerido.

Al representar el número **1500** en el rango del número **10,000,000,000** se comete un error apreciable, ya que pasamos de tener el valor 1500 a tener un valor de 1024.

Conclusiones Punto Fijo y flotante

Cada sistema tiene sus ventajas y desventajas.

Punto flotante presenta un gran rango dinámico y el tiempo de desarrollo es menor, ya que el programador no debe verificar overflow o underflow, ya que difícilmente esto suceda.

Punto fijo presenta la ventaja de dar una precisión constante en todo el rango de trabajo. Además los procesadores que soportan solo punto fijo (no tienen FPU) son más económicos.

Ambos sistemas pueden usarse en un mismo desarrollo para potenciar ventajas de cada uno.

FILTRO ANTIALIASING

Procesamiento digital de señales analógicas

Cuando se desea operar en forma discreta una señal analógica se necesitan al menos 3 bloques.

El primero es el **conversor analógico/digital**, que trabaja con una frecuencia de muestreo de periodo igual a **T**. Este bloque recibe una señal analógica $x_c(t)$ y entrega una señal $x[n]$ discretizada.

Luego viene un **bloque de sistema de tiempo discreto**, típicamente se trata de un filtro digital, que entrega una señal $y[n]$.

Esta señal $y[n]$ alimenta al tercer bloque, que es un **conversor digital/analógico**, gobernado también por una frecuencia de muestreo con periodo **T**.

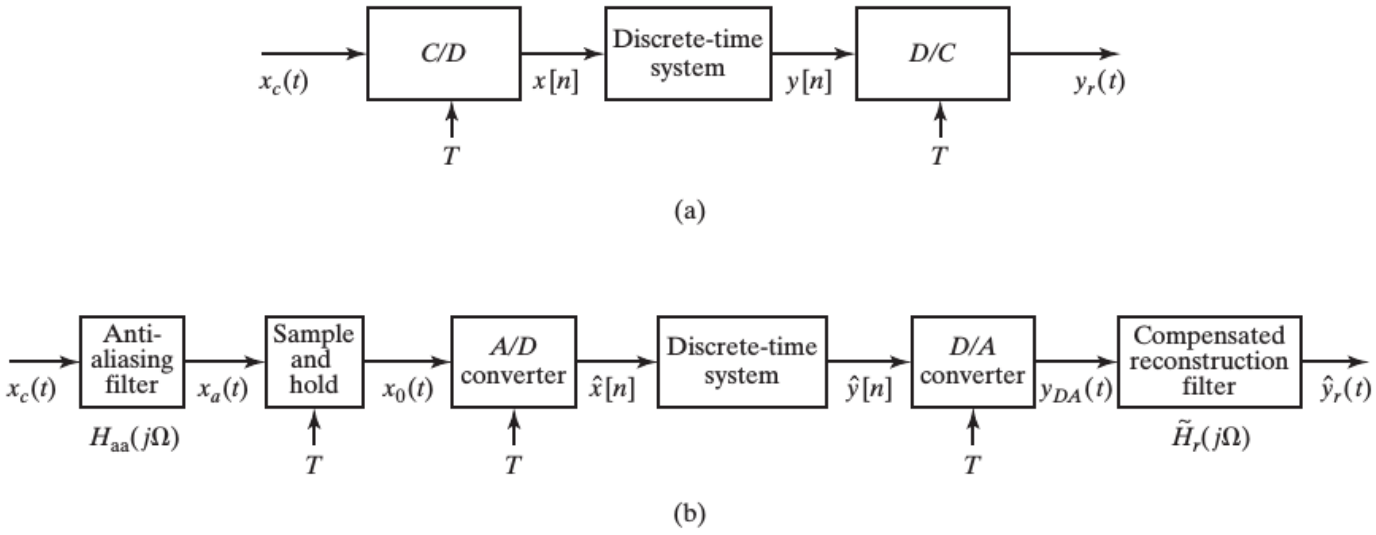


Figure 4.47 (a) Discrete-time filtering of continuous-time signals. (b) Digital processing of analog signals.

En la figura inferior se observa una cadena de procesamiento de una señal analógica. El primer bloque es el **filtro antialiasing**. Los siguientes dos bloques representan un conversor analógico/digital real. A la salida finalmente tenemos un conversor digital/analógico y un filtro de reconstrucción.

Una señal digitalizada puede representarse como:

$$x[n] = x_c(nT)\ , \quad -\infty < n < +\infty$$

Donde T es el periodo de muestreo, por lo que la frecuencia de muestreo es $f_s = \frac{1}{T}$, o en radianes $\Omega_s = \frac{2\pi}{T}$.

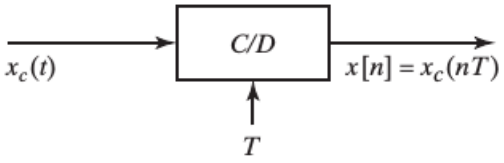
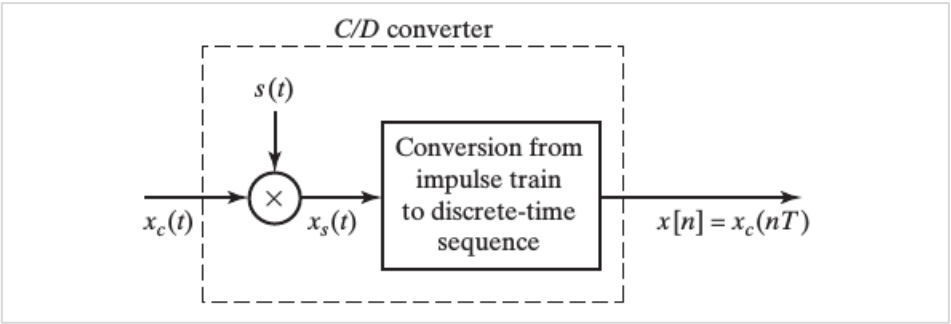


Figure 4.1 Block diagram representation of an ideal continuous-to-discrete-time (C/D) converter.

Muestreo periódico de la señal en el dominio de la frecuencia

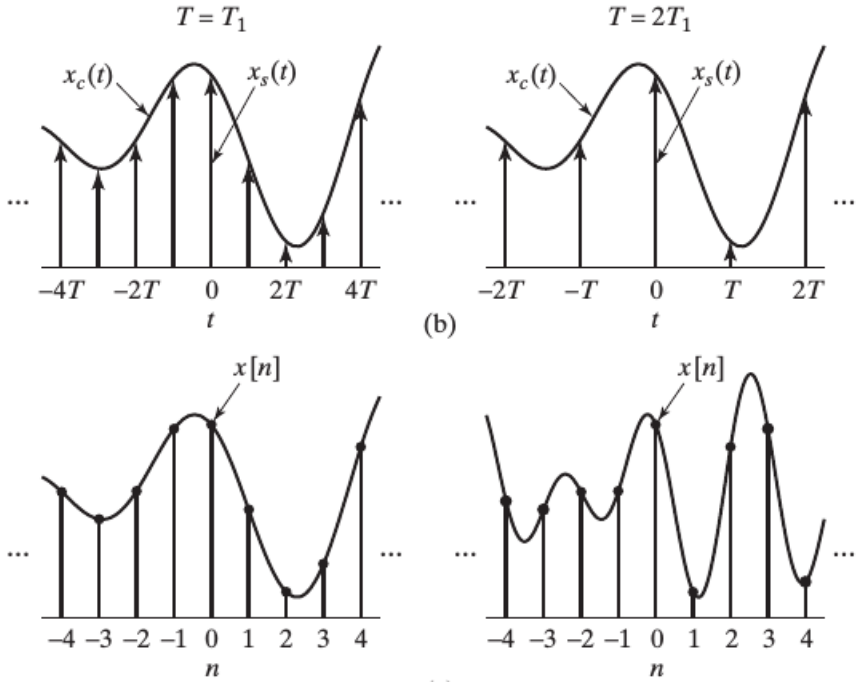
El proceso de muestreo puede representarse matemáticamente de la siguiente forma:



La señal analógica $x_c(t)$ es multiplicada en el dominio del tiempo por un tren de impulsos $s(t)$, el cual da como resultado un tren de impulsos modulados $x_s(t)$. Luego este es convertido en valores binarios

(objetivo del bloque). A la salida del bloque obtenemos una $x[n]$, que es una secuencia discreta de valores que representa a la señal x_c evaluada en tiempos discretos.

A continuación se muestra un ejemplo de este proceso.



La señal analógica x_c es la envolvente de la figura superior y los valores que toma la señal discretizada x_s son los impulsos representados con flechas en la misma figura. En la primer figura los impulsos están separados un valor igual a T. En la figura inferior se observa cómo podría reconstruirse la señal analógica a partir de la señal discretizada.

En la figura de la derecha se toma un periodo de muestreo igual al doble del ejemplo de la izquierda. En este caso los valores muestreados están espaciados un valor igual a 2T. Al reconstruir la señal, figura inferior, se observa que la misma no representa la señal real analógica. Este problema se desprende del teorema de muestreo de Nyquist – Shannon.

Este teorema establece que siendo x_c una señal analógica de banda limitada, lo cual significa que:

$X_c(j\Omega) = 0$ para $|\Omega| \geq \Omega_N$, donde Ω_N es la máxima frecuencia de la señal.

La señal x_c puede ser determinada únicamente por las muestras de $x[n] = x_c(nT); n = 0, \pm 1, \pm 2, \ldots$ si se cumple:

$$\Omega_s = \frac{2\pi}{T} \geq 2\Omega_N$$

Donde Ω_s es la frecuencia de muestreo en radianes y Ω_N la máxima frecuencia de la señal x_c.

Representación en el dominio de la frecuencia de la señal muestreada

La señal $x_s(t)$, como dijimos anteriormente, se obtiene multiplicando a $x_c(t)$ por un tren de impulsos periódico $s(t)$:

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

$$x_s(t) = x_c(t)\, s(t)$$

$$x_s(t) = x_c(t) \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

Solo tiene sentido evaluar a x_c en valores de nT, porque fuera de estos valores la expresión de δ vale cero, entonces incluimos a x_c dentro de la sumatoria pero evaluándolo solo en nT:

$$x_s(t) = \sum x_c(nT)\delta(t - nT)$$

Utilizando la transformada de Fourier, el tren de impulsos resulta:

$$S(j\Omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\Omega - j\Omega_s), \text{ donde } \Omega_s = \frac{2\pi}{T}$$

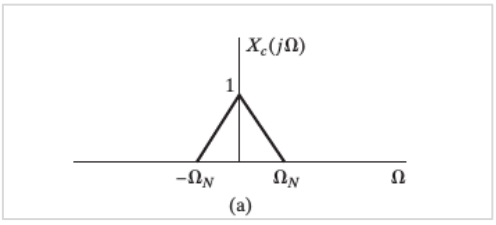
Por lo que al multiplicarlo en el dominio del tiempo por la señal x_c obtenemos en el dominio de la frecuencia la convolución de las dos señales:

$$X_s(j\Omega) = \frac{1}{2\pi}\, X_c(j\Omega) * S(j\Omega)$$

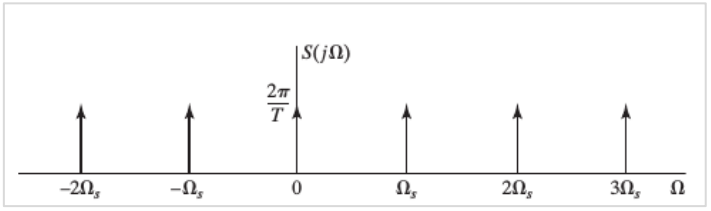
$$X_s(j\Omega) = \frac{1}{T} X_c[j(\Omega - k\Omega_s)]$$

Obtenemos el espectro de X_c desplazado en el dominio de la frecuencia $k\Omega_s$. Estos espectros estarán replicados y separados según la Ω_s.

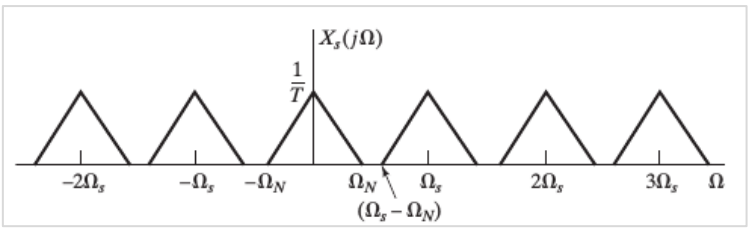
Si el espectro de la señal X_c es:



La respuesta de la señal de impulsos es:

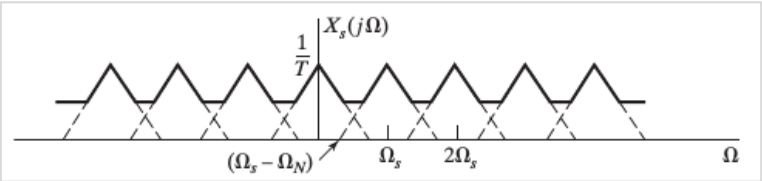


Al hacer la convolución entre estas dos señales obtenemos:



Lo cual es el espectro original multiplicado y centrado en los valores de la frecuencia de muestreo.

El teorema de Nyquist nos dice que la frecuencia de muestreo debe ser al menos dos veces la máxima frecuencia de la señal que se quiere muestrear. Si esto no se cumple se produce espectralmente lo siguiente:



Como se observa, hay superposición entre los espectros, es decir, se está mezclando parte de la frecuencia de la señal muestreada. Esto es lo que se denomina **aliasing**.

Como se ve en el siguiente esquema, el filtro antialiasing siempre precede al conversor A/D.

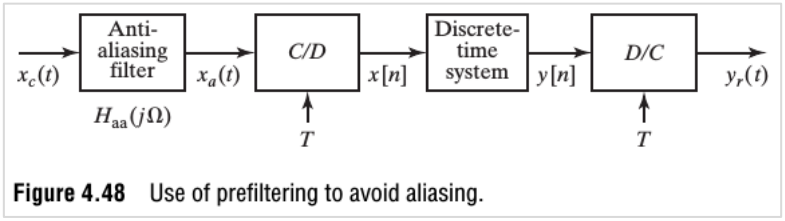
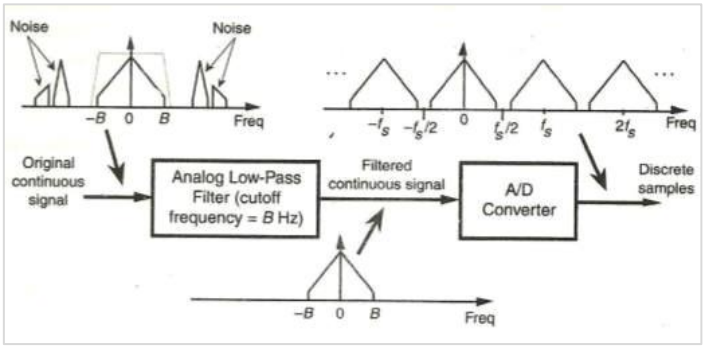


Figure 4.48 Use of prefiltering to avoid aliasing.

Este filtro típicamente es un pasa bajo o pasa banda, pero siempre trabaja en el dominio continuo del tiempo. Su objetivo es garantizar que la máxima señal que va a entrar al sistema es Ω_N, por lo que este debe ser el valor de su frecuencia de corte.

Esto es necesario para eliminar señales de frecuencia superior a la máxima frecuencia de la señal tratada, como puede ser el ruido.

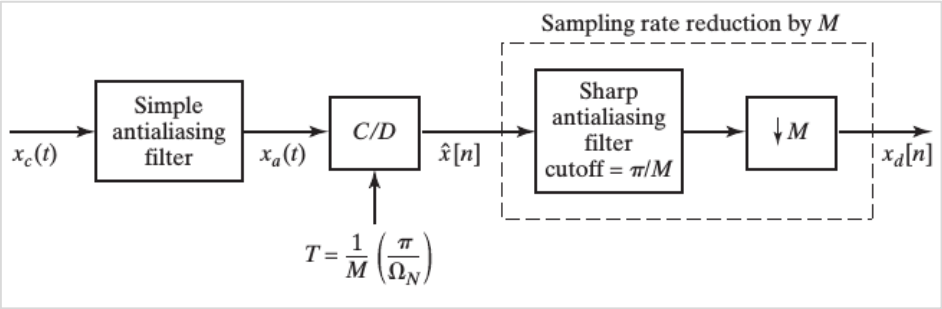


Como se ve en la imagen anterior, en presencia de ruido podemos sufrir aliasing por el solapamiento entre el espectro de la señal trabajada y el ruido. Al pasar por el filtro antialiasing este ruido es eliminado por lo que al producirse la conversión, con un periodo de muestreo determinado según la máxima frecuencia de la señal trabajada, evitamos el aliasing.

Oversampling

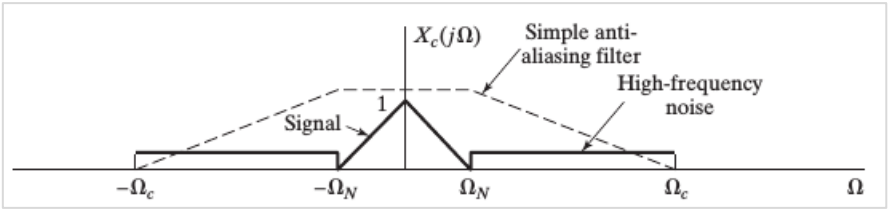
Una técnica que se puede implementar para el filtro antialiasing es el **oversampling**. Normalmente una señal con ruido requiere de un filtro antialiasing muy exigente, difícil de lograr y muy costoso. Esta técnica consiste en usar un filtro antialiasing simple, combinado con un aumento en la frecuencia de muestreo, dividiendo el periodo de muestreo por un valor M. Luego en el dominio digital se usa un filtro digital con frecuencia de corte abrupta, para luego bajar la frecuencia de muestreo a la que se

desee, dividiéndola por M, para volver a trabajar con la deseada, como se muestra en la siguiente figura:

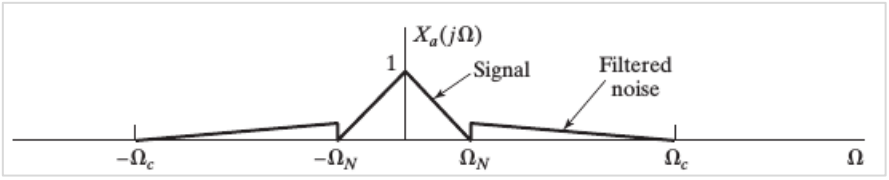


En el dominio de la frecuencia el oversampling funciona de la siguiente manera:

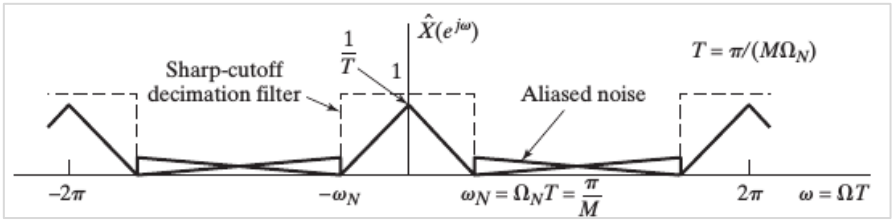
La señal está representada por el triángulo centrado, el ruido son los rectángulos a los costados y la línea punteada representa la respuesta del filtro antialiasing simple.



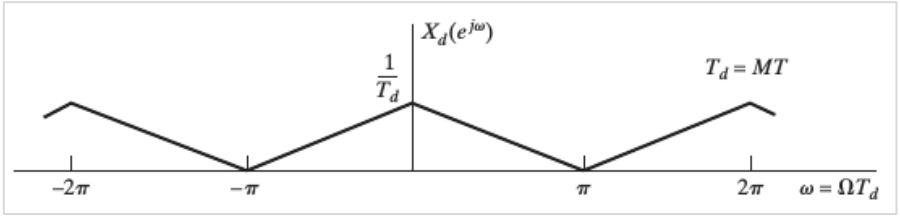
La señal luego de pasar por el filtro simple antialiasing posee la siguiente respuesta en frecuencia.



Ya en el dominio digital, se produce aliasing entre el ruido. En este dominio podemos pasar la señal por un filtro digital pasa bajo, eliminando el ruido, ya que podemos obtener filtros de corte abrupto. Nos quedamos entonces con el espectro encerrado por las líneas punteadas.

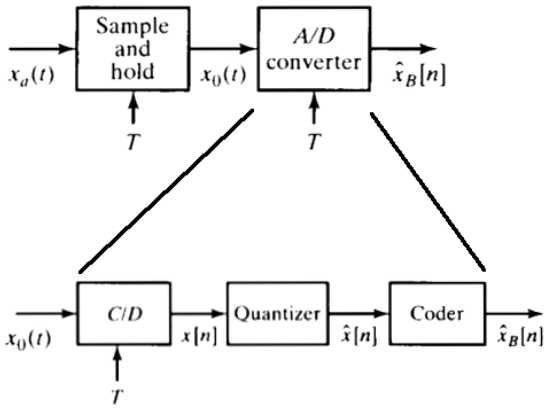


Finalmente, se disminuye la frecuencia de muestreo al límite de la frecuencia de Nyquist. Esto es opcional.



CONVERSOR ANALÓGICO DIGITAL

El conversor A/D puede ser dividido en dos partes, un circuito denominado "sample and hold", y el conversor A/D en si, como se ve en la siguiente figura:



El objetivo del primer bloque es mantener por un tiempo determinado el valor de tensión eléctrica presente a la entrada del conversor para que el siguiente bloque tenga tiempo suficiente para realizar la conversión.

La etapa conversora en sí puede dividirse en las 3 etapas graficadas en la parte inferior de la figura, estas son:

- **C/D:** conversor ideal que ante una muestra de tensión $x_0(t)$ entrega un valor digitalizado $x[n]$.
- **Quantizer:** recibe el valor digitalizado $x[n]$ de la etapa anterior y le otorga un valor cuantizado, el cual depende de la cantidad de bits que tenga la etapa, $\hat{x}[n]$.
- **Coder:** toma el stream de bits $\hat{x}[n]$ de la etapa anterior y lo acomoda a un sistema de numeración especifico $\hat{x}_B[n]$ (uint32, int16, punto fijo, flotante, etc.).

Quantizer

En la figura se observa la respuesta o función de un cuantizador espaciado uniformemente:

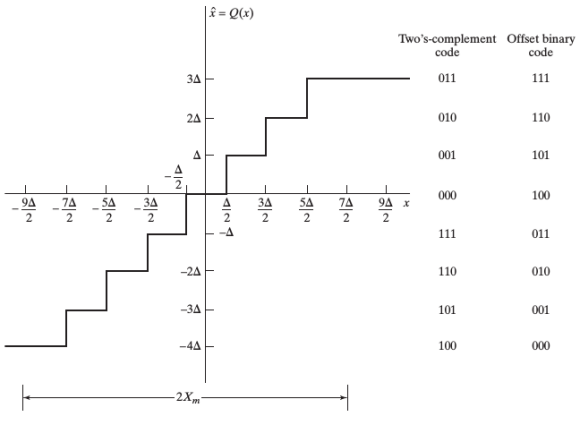


Figure 4.54 Typical quantizer for A/D conversion.

Esta función se denomina $Q(x)$. Los valores Δ se definen como la relación entre el rango de tensión y la cantidad de niveles binarios representables con *B* bits, es decir:

$$\Delta = \frac{rango\ completo\ de\ voltage}{2^{longitud\ de\ palabra}} = \frac{2V_p}{2^B}$$

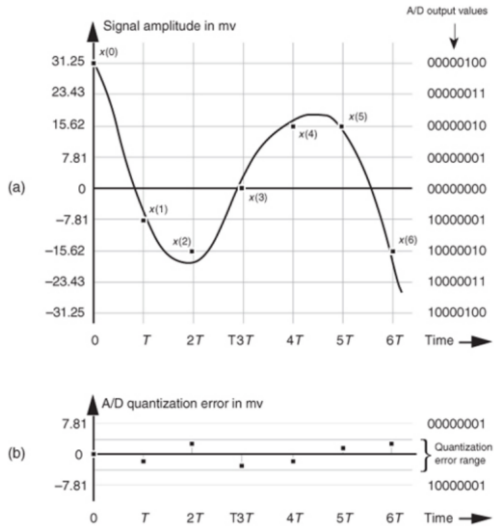
Este valor representa la **precisión** del cuantizador.

En la gráfica el eje de las abscisas representa la tensión de entrada y el de las ordenadas el valor que entrega el cuantizador.

La tabla a la derecha de la figura representa la conversión o adaptación que realiza el coder con el flujo de bits que recibe del quantizer, dependiendo del sistema de numeración que utilice.

Error de cuantizador

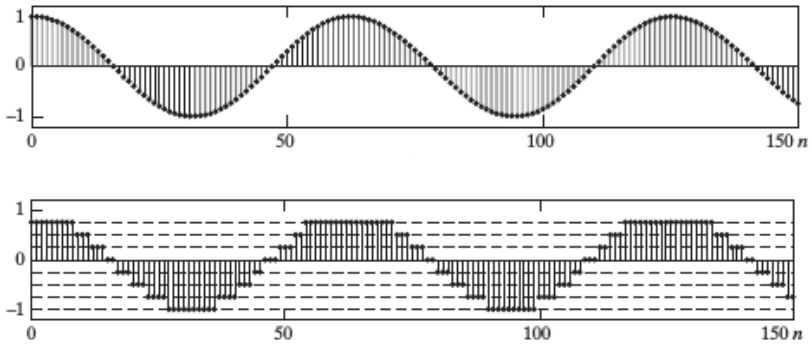
En la siguiente figura se representan los valores binarios que toma una función continua al pasar por el bloque de quantizer (obviamente luego de pasar por el bloque C/D).



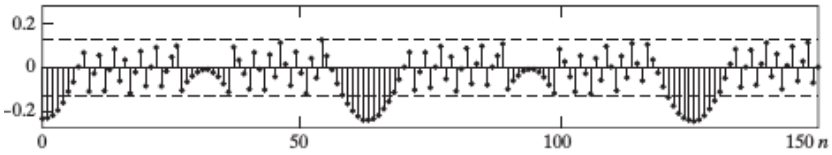
Vemos que para el valor $x(0)$ el valor real es casi igual al cuantizado. Para $x(1)$ se observa una pequeña diferencia entre estos dos valores y ya para $x(2)$ la diferencia es notoria.

En la figura inferior se observa la diferencia entre estos valores. Este valor de error varía entre $\pm\Delta$.

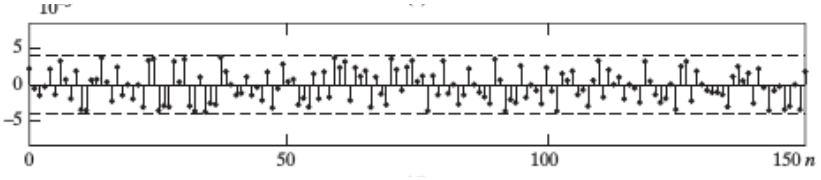
Para ver más detalladamente el efecto que produce el error en la cuantización observemos las siguientes dos figuras. La primera es una señal perfectamente discretizada, la segunda es la misma señal luego de ser cuantizada en un cuantizador de 3 bits:



La siguiente figura indica la diferencia entre estas dos señales:

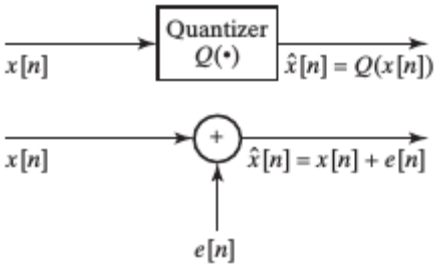


Por último observemos la forma de esta última señal, si la señal cuantizada tuviera 8 bits en lugar de 3:



Al observar estas dos últimas formas de onda concluimos que al cuantizar una señal discretizada el error de cuantización no es más que la adición de ruido a la señal.

El modelo del error de cuantización entonces resulta:



Este modelo se denomina *modelo de ruido aditivo para cuantizador*.

SNR en el conversor A/D

La precisión del cuantizador es:

$$\Delta = q = \frac{2\,V_p}{2^B}$$

La relación señal a ruido es:

$$SNR = \frac{P_{señal}}{P_{ruido}}$$

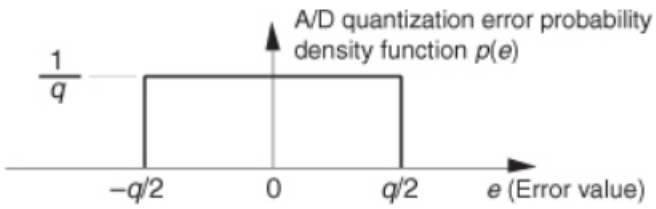
Como es un valor aleatorio, se usa una relación estadística de varianzas para definir la SNR:

$$SNR_{ADC} = 10\log_{10}\left(\frac{varianza\ de\ la\ señal\ de\ entrada}{varianza\ del\ ruido\ de\ cuantizacion\ A\backslash D}\right)\ [dB]$$

$$SNR_{ADC} = 10\log_{10}\left(\frac{\sigma^2_{señal}}{\sigma^2_{ADC}}\right)$$

Este valor de varianza σ^2_{ADC} se define de la siguiente manera:

- Proponemos un modelo probabilístico, suponiendo que el error de cuantización tiene una distribución uniforme, de la siguiente forma:



- Calculamos la varianza a partir de este modelo, y llegamos a la expresión final

$$SNR_{ADC} = 20\log_{10}(LF) + 4.77 + 6.02\,B$$

Lo que es igual a:

$$SNR_{ADC} = 20\log_{10}\left(\frac{rms_{señal}}{V_p}\right) + 4.77 + 6.02\ B$$

Para obtener una SNR alta debemos ajustar los términos de la ecuación anterior adecuadamente.

Si consideramos el primer término, lo ideal sería hacer $rms_{señal} \gg V_p$, lo cual no suele ser posible porque existen límites físicos, sin ir más lejos los conversores A/D son de baja potencia por lo que no pueden entregar una potencia efectiva tan alta. Sin embargo tener valores $rms_{señal} \ll V_p$ decrementa la SNR, por lo cual no es conveniente alimentar conversores A/D con señales de baja potencia efectiva.

Si vamos al tercer término $6.02\ B$ vemos que por cada bit que tenga el conversor, la SNR aumenta en 6.02 dB, esto quiere decir que cuantos más bits tenga el conversor mejor será la SNR.

Para llegar a estas conclusiones y a la ecuación hemos tomado en cuenta supuestos que nos inducen a errores. Uno de esos supuestos es el de considerar que la tensión pico en el conversor A/D es igual a la tensión máxima que este puede manejar. Otro error surge de suponer que la densidad de probabilidad de error tiene una función uniforme. Por estas consideraciones, a la ecuación de SNR se le decrementa un valor de 3 o 6 dB.

Para una señal senoidal pura, la tensión eficaz es:

$$rms_{señal} = \frac{V_p}{\sqrt{2}}$$

La relación señal a ruido resulta:

$$SNR_{ADC} = 20\log_{10}\left(\frac{rms_{señal}}{V_p}\right) + 4.77 + 6.02\ B$$

$$SNR_{ADC} = 20\log_{10}\left(\frac{V_p/\sqrt{2}}{V_p}\right) + 4.77 + 6.02\ B$$

Entonces, la máxima SNR resulta:

$$SNR_{ADC} = 20\log_{10}\left(\frac{1}{\sqrt{2}}\right) + 4.77 + 6.02\ B$$

$$SNR_{ADC} = -3.01 + 4.77 + 6.02\ B$$

$$SNR_{ADC} = 1.76 + 6.02\ B\ [dB]$$

Se observa que el primer término aporta un valor negativo en dB de SNR, ya que la tensión eficaz es 0.707 veces la tensión fija.

Resolución A/C para una señal particular

Consideremos el siguiente ejemplo:

- La SNR para la salida de un amplificador de audio es de 110 dB.
- El conversor A/D seleccionado tiene 24 bits (audio profesional).
- El amplificador de audio es excitado por una señal de entrada sinusoidal. La SNR agregando el factor de seguridad de -3 dB resulta.

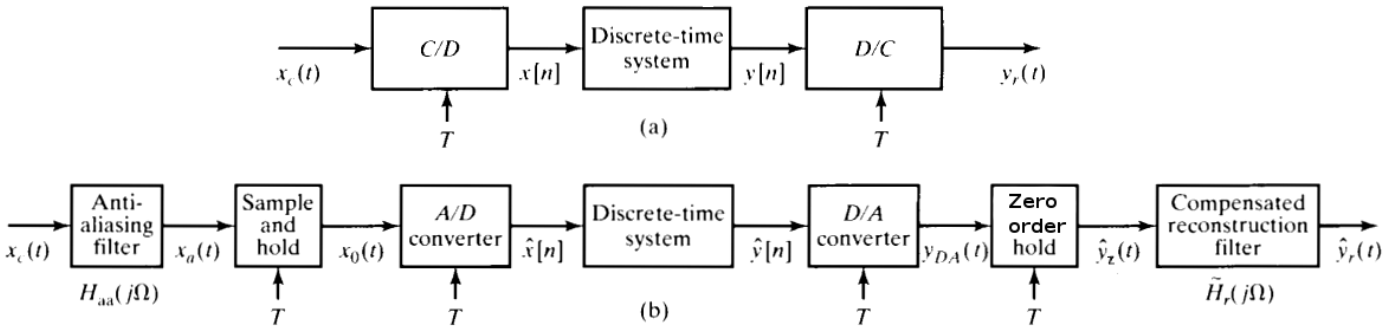
$$SNR_{ADC} = 1.76 + 6.02 \cdot 24 - 3 = 143.24\ dB$$

Vemos que en dB la SNR del conversor es mayor que la del amplificador que queremos digitalizar. Esto quiere decir que estamos utilizando algunos bits para medir ruido.

- Para calcular cuantos bits están usándose para medir ruido hacemos: $\frac{143-110}{6} \cong 5.5\ bits$. El valor 6 es porque cada bit añade aproximadamente 6 dB de SNR (término $6.02\ B$).
- Suponiendo que usamos ahora un conversor de 10 bits. Ahora la SNR será de aproximadamente $62\ dB$, con lo cual ahora nos faltan bits, más precisamente nos faltan $\frac{110-61.96}{6} \cong 8\ bits$. Esto significa que se está deteriorando la señal convertida.
    - En resumen, el número de bits B en un conversor A/D debe proveer una $SNR_{ADC} \geq SNR_{señal}$
- Una regla utilizada indica que siempre debemos estar 6 dB por encima de la $SNR_{señal}$ (1 bit).
- Los bits adicionales pueden descartarse realizando un desplazamiento a la derecha. Un ejemplo de código en C es:
    - `adc_read >>= 5`

CONVERSOR DIGITAL ANALÓGICO

Este conversor recibe un numero binario, generalmente en punto fijo, y lo transforma en un valor de tensión.

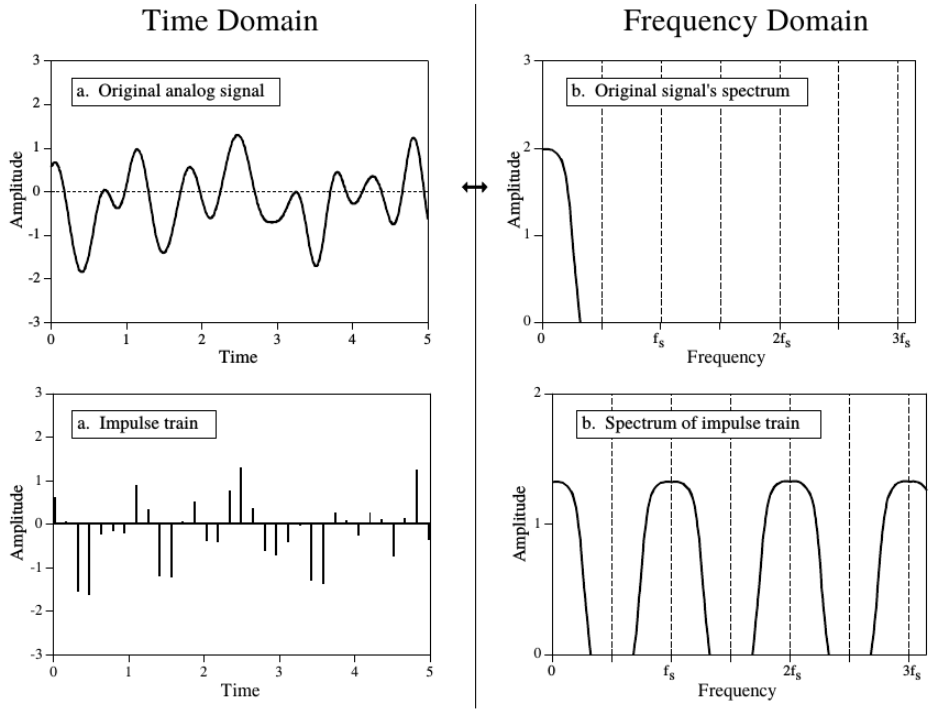


Dos características importantes de un conversor DAC son:

- **Resolución:** número de posibles niveles de salida que puede reproducir. Es usualmente fijado por el número de bits del DAC.
- **Frecuencia de muestreo:** la salida del DAC se puede ver como un tren de pulsos, definidos por esta frecuencia. Esta frecuencia puede o no ser igual a la del conversor AD.

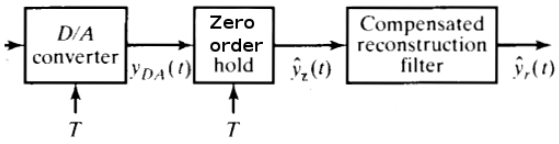
Operación del DAC

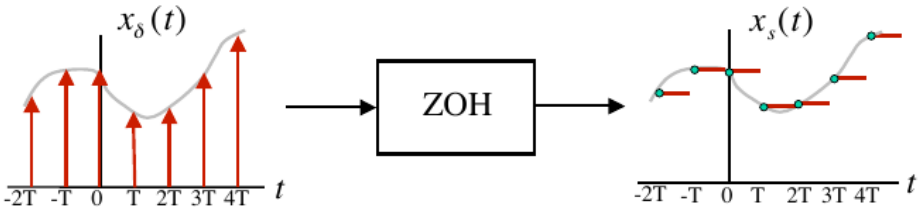
Un DAC ideal recibe una secuencia de números binarios y la transforma en un tren de impulso. El tren de impulsos contiene una duplicación del espectro de la señal analógica. La señal analógica original es reconstruida pasando este tren de impulsos a través de un filtro pasa bajos, con una frecuencia de corte igual a $\frac{f_s}{2}$.



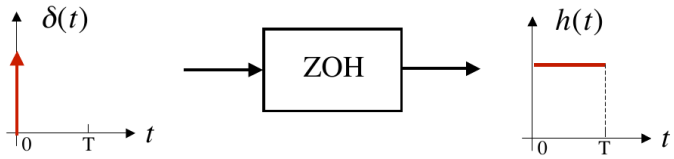
ZERO ORDER HOLD

A la salida del DAC tenemos otro bloque, denominado **Zero order hold**, encargado de recibir el tren de impulsos y transformarlo en escalones. Básicamente recibe el valor de un impulso y lo sostiene hasta recibir un nuevo valor, es decir, interpola sosteniendo el valor de entrada un tiempo igual a la frecuencia de muestreo.





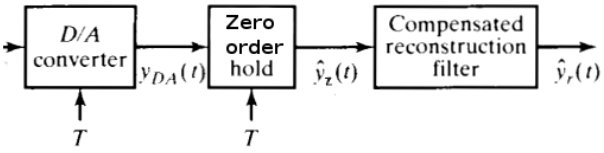
A continuación se analiza la respuesta en frecuencia característica del bloque ZOH. La respuesta ante un impulso de este bloque resulta:



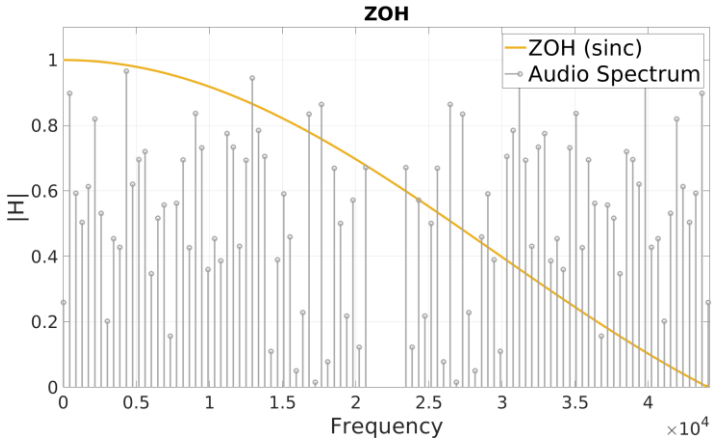
Es decir, recibe un impulso y devuelve un pulso de ancho T (periodo de muestreo). En el dominio de la frecuencia:

$$H_{ZOH}(f) = \mathcal{F}\left\{rect\left(\frac{t-\frac{T_s}{2}}{T_s}\right)\right\} = \frac{\sin\left(\frac{\pi f}{f_s}\right)}{\frac{\pi f}{f_s}} = sinc\left(\frac{\pi f}{f_s}\right)$$

ZOH es la convolución de un tren de impulsos con un pulso rectangular. En el dominio de la frecuencia, la transformada de Fourier de ZOH ($sinc$) está siendo multiplicada por el espectro del tren de impulsos.

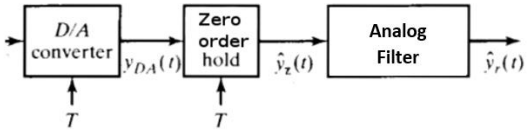


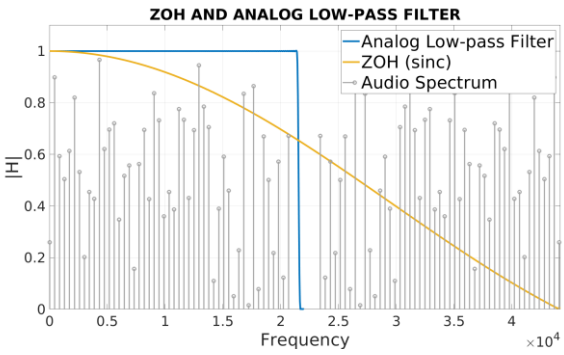
Por ejemplo, para una aplicación de audio donde la frecuencia a máxima es $f_N = 22050\ Hz$ y la de muestreo $f_s = 44100\ Hz$.



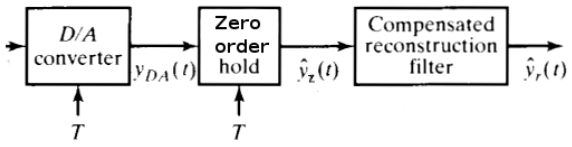
En gris se observa el espectro de audio de la señal dada, se observa que el mismo está espejado en la frecuencia de muestreo. La línea amarilla representa la respuesta en frecuencia del ZOH, la cual se observa es un filtro pasa bajos malo, debido a que su corte no es abrupto.

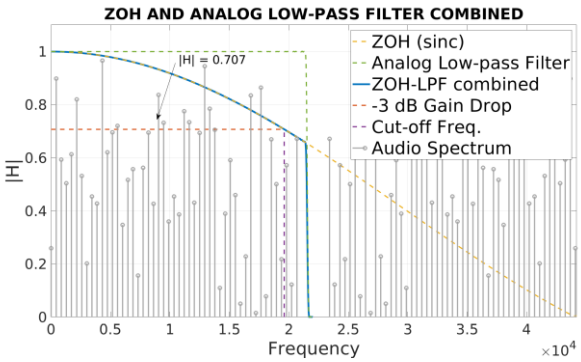
Agregamos un filtro a la salida del sistema (salida del ZOH) para eliminar el espectro a la derecha de la f_N y para eliminar las componentes de alta frecuencia debidas a la forma de onda escalonada provista por el ZOH, suavizando dichos escalones.





Al convolucionar la respuesta en el dominio del tiempo de los 3 bloque (DAC, ZOH y filtro) obtenemos el producto de las respuesta en frecuencia de los mismos. Esta respuesta tiene la siguiente forma:

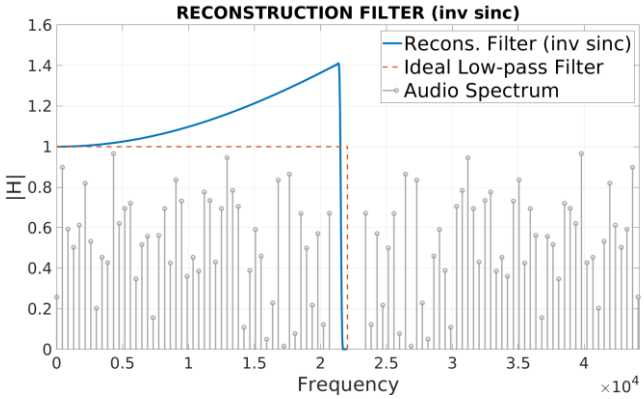




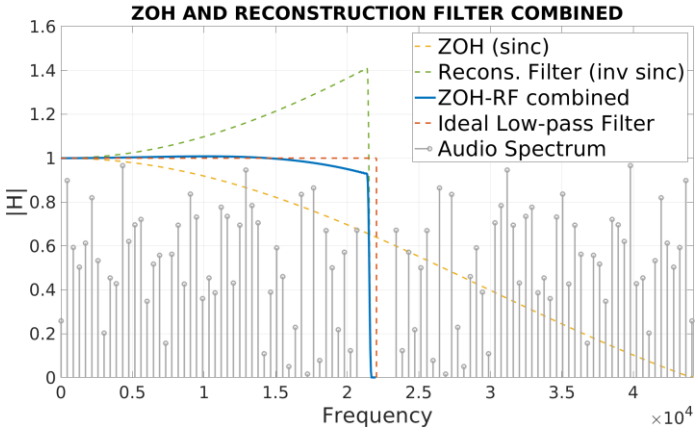
La línea azul representa la respuesta efectiva del sistema. En la intersección con la línea punteada anaranjada la amplitud de la señal es de 0.707, por lo que la señal ha perdido la mitad de la potencia. En dicho punto se considera que la señal ya comienza a atenuarse, es decir, que la frecuencia de corte del filtro final es menor a la del último filtro añadido.

FILTRO DE RECONSTRUCCIÓN

El problema anterior se soluciona quitando el filtro analógico añadido recientemente y colocando un filtro especial denominado **filtro de reconstrucción**. Este filtro implementa la inversa de la caída del ZOH, es decir, tiene una respuesta en frecuencia igual a la inversa del coseno cardinal.



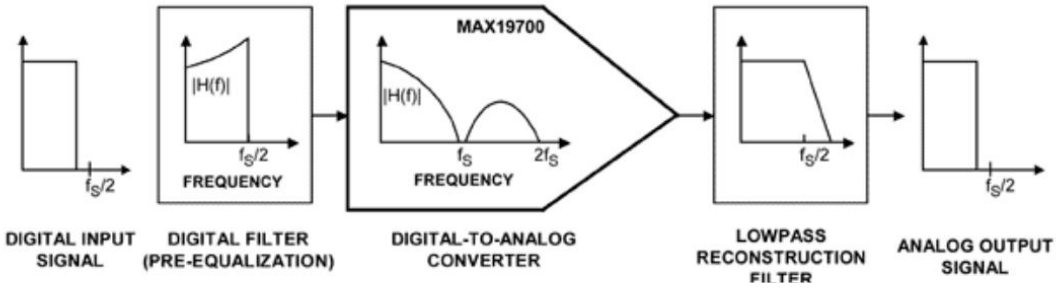
Si entonces reemplazamos el filtro analógico genérico colocado en el último bloque por el filtro de reconstrucción mencionado, la respuesta del sistema resulta:



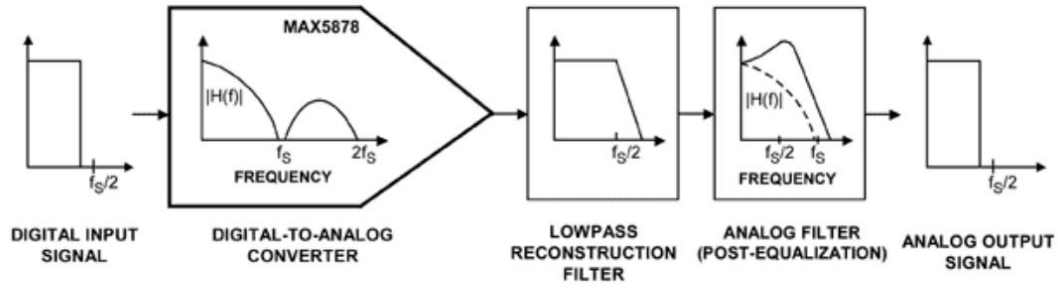
La caída cerca de la frecuencia de corte es mínima, no llega al 10%, por lo que no se tiene el problema que se tenía al utilizar un filtro común.

Existen varias estrategias para el filtrado de reconstrucción:

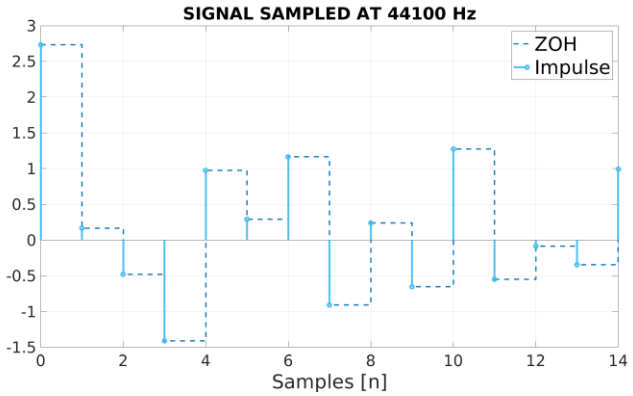
1. **Ignorar** los efectos del ZOH y aceptar las consecuencias.
2. **Pre ecualizar:** implementamos en el dominio digital un filtro de reconstrucción en la entrada del DAC. Luego del ZOH se coloca un filtro analógico corriente.



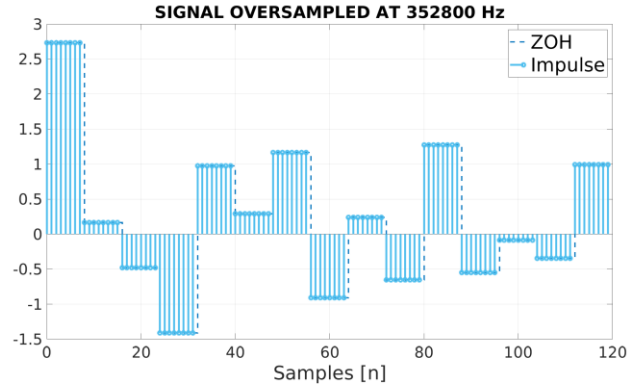
3. **Post ecualizar:** colocamos el filtro de reconstrucción a la salida. En este caso el filtro es analógico y se coloca luego del filtro analógico pasa bajo.



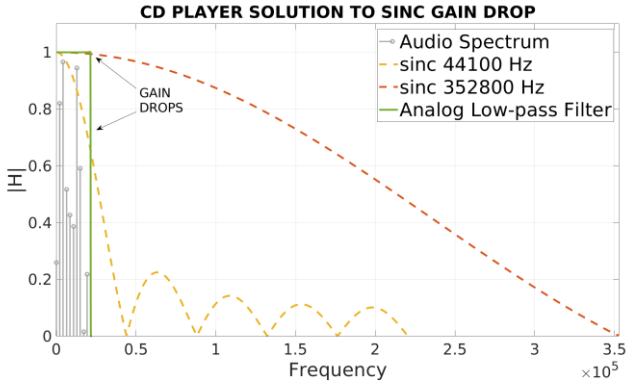
4. **Oversampling:** a la salida del DAC aumentamos la frecuencia de muestreo para modificar la respuesta del ZOH.



Esta estrategia es utilizada por los reproductores de CD, los cuales multiplican la frecuencia de muestreo por un factor de 8, como se ve en la siguiente figura:



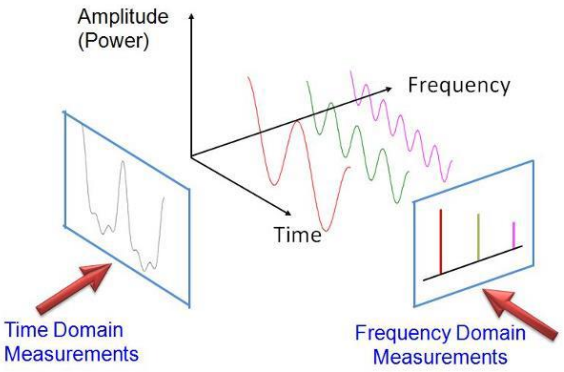
Esto hace que el pulso a la salida del ZOH sea 8 veces más corto, por lo que en el dominio de la frecuencia, la función $sinc$ (respuesta en frecuencia del ZOH) se hace más ancha, como se ve a continuación.



Se observa en línea punteada amarilla la respuesta en frecuencia del ZOH trabajando con la frecuencia de muestreo predeterminada. En línea punteada anaranjada se representa la misma respuesta en frecuencia al realizar oversampling. En este último caso se observa que la caída de 50% de la potencia si no existiese filtro pasa bajo analógico se produciría en una frecuencia alejada de la banda de la señal trabajada, por lo que no haría falta un filtro de reconstrucción.

FILTROS FIR

El filtrado puede hacerse en dos dominios diferentes, el dominio del **tiempo** y el dominio de la **frecuencia**.



Al filtrar en el dominio de la frecuencia buscamos eliminar alguna componente de la señal. Al hacerlo en el dominio del tiempo generalmente se busca eliminar ruido, suavizando así la señal.

La utilización de un filtro en el domino del tiempo o de la frecuencia depende del dominio en el que se encuentre contenida la información en la señal.

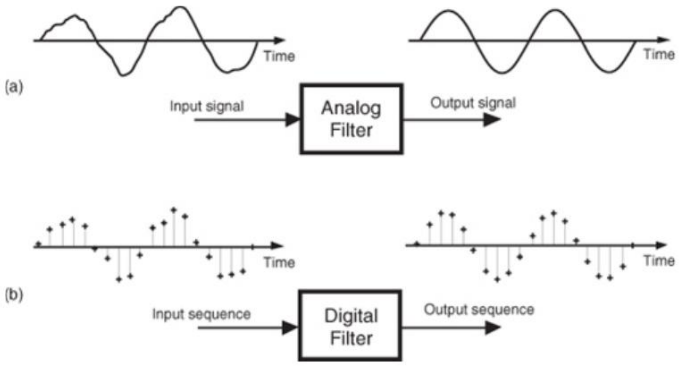
Señales y filtrado en el dominio del tiempo

Para que una señal posea información contenida en el dominio del tiempo, esta debe estar representada por la amplitud y tiempo de la señal. En este caso cada muestra de la señal contiene información por sí misma.

A los sistemas diseñados para filtrar en el dominio del tiempo se los evalúa utilizando una **señal escalón**. Esta señal describe como la información en el dominio del tiempo está siendo modificada por el sistema.

Ejemplos de este tipo de señal son los electrocardiogramas, registros de acelerómetros, registros de temperatura, etc.

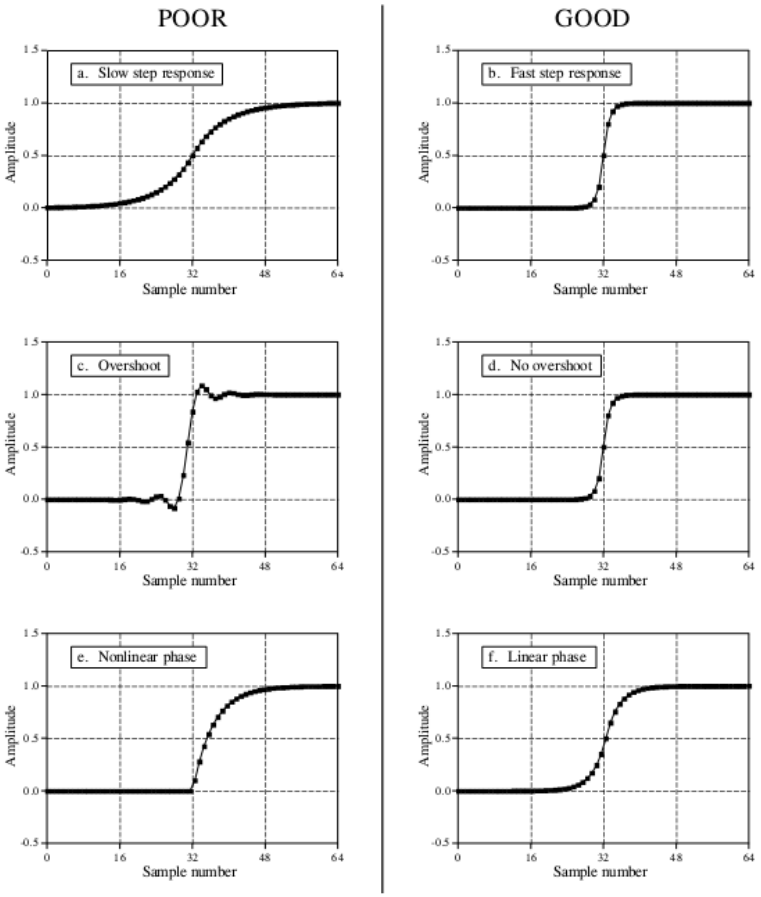
En la siguiente figura se muestra una comparación entre un filtro analógico y un filtro digital en el dominio del tiempo:



Cabe destacar que en **no existen los filtros en el dominio del tiempo analógicos**. Para filtrado en el dominio del tiempo deben usarse siempre filtros digitales.

Los parámetros para evaluar un filtro FIR en el dominio del tiempo son los siguientes:

- Tiempo de elevación.
    - Es el tiempo que toma en pasar de 10 a 90% la amplitud
- Sobre pico
    - Es la magnitud del pico que se genera una vez que el sistema responde al escalón
- Linealidad en fase
    - Un filtro de fase lineal ofrece una respuesta al escalón simétrica.



A la izquierda se muestran filtros con mal rendimiento en estos parámetros y a la derecha filtros con buen rendimiento.

No es posible optimizar filtros para ambos dominios. Un buen rendimiento en el dominio del tiempo resulta en un pobre rendimiento en el dominio de la frecuencia, y viceversa.

Filtro fir moving average

Este filtro se define como la **convolución** de una señal de entrada y un pulso rectangular de área igual a uno. También se lo conoce como *local average*.

El uso de este filtro introduce un retraso de $\frac{N}{2}$ muestras entre entrada y salida, donde **N es el orden del filtro**.

En las siguientes ecuaciones vemos la definición del filtro:

$$h[n] = \frac{1}{n} \sum_{k=0}^{N-1} \delta[n-k]$$

A $h[n]$ se lo conoce como el **kernel** del filtro. Otra forma de definirlo es:

$$h[n] = \begin{cases} \frac{1}{N} & 0 \leq n < N \\ 0 & otherwise \end{cases}$$

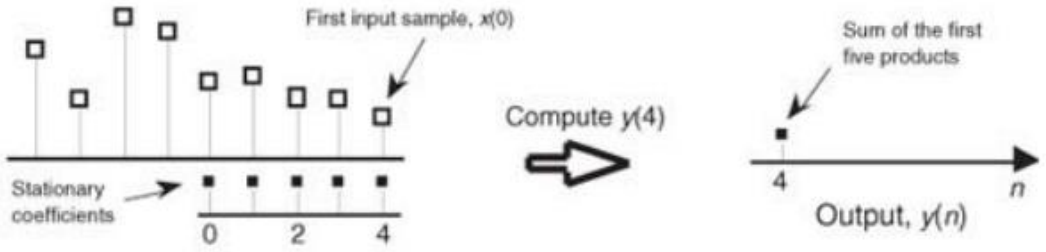
Esta fórmula define un rectángulo de área 1, ya que posee N muestras y una altura de $\frac{1}{N}$.

En definitiva, $h[n]$ es un vector de N elementos que contiene los coeficientes del filtro moving average. Para este caso es fácil de encontrar el valor de los coeficientes, ya que por ejemplo, si $N = 10$, el filtro va a tener 10 elementos de valor igual a $\frac{1}{10}$. Finalmente, la salida del filtro es igual a :

$$y[n] = x[n] * h[n] = \frac{1}{n} \sum_{k=0}^{N-1} x[n-k]$$

Observando esta ecuación final, notamos fácilmente que se trata de un filtro **FIR**, ya que la salida actual $y[n]$ no depende de salidas pasadas ($y[n-1]$, $y[n-2]$). Con esto estamos seguros de que no hay realimentación de salida a entrada, por lo que se trata de un filtro FIR.

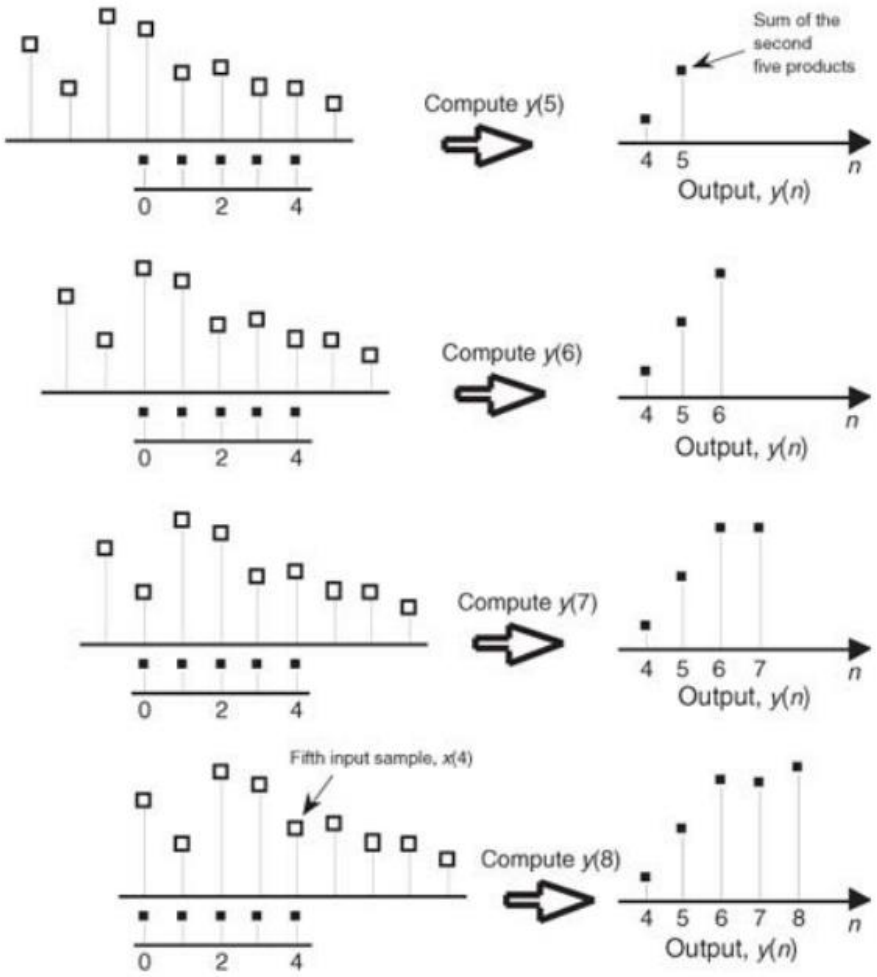
A continuación se observa gráficamente el funcionamiento de un filtro FIR moving average:



En la parte inferior de la gráfica a la izquierda se observan los coeficientes del filtro. Vemos que es un filtro de orden $N = 5$. Arriba de estos coeficientes se representa a la señal.

La operación consta de multiplicar elemento a elemento los coeficientes del filtro con los valores de la señal y sumar el resultado de cada operación, es decir, se trata de una operación MAC.

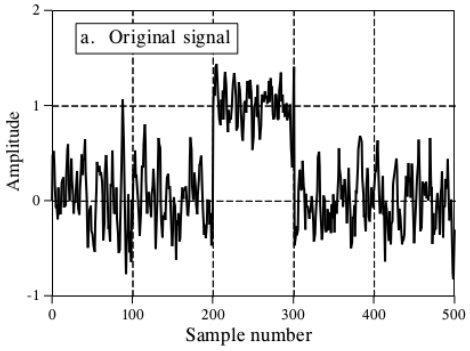
A la salida (derecha) obtenemos el primer elemento de la salida en el tiempo 4, que es el tiempo en el que el filtro ya está trabajando en régimen, es decir, todos los valores de entrada de x[n] están enfrentados con todos los coeficientes del filtro. (el retraso entonces es de N, no de N/2?????)



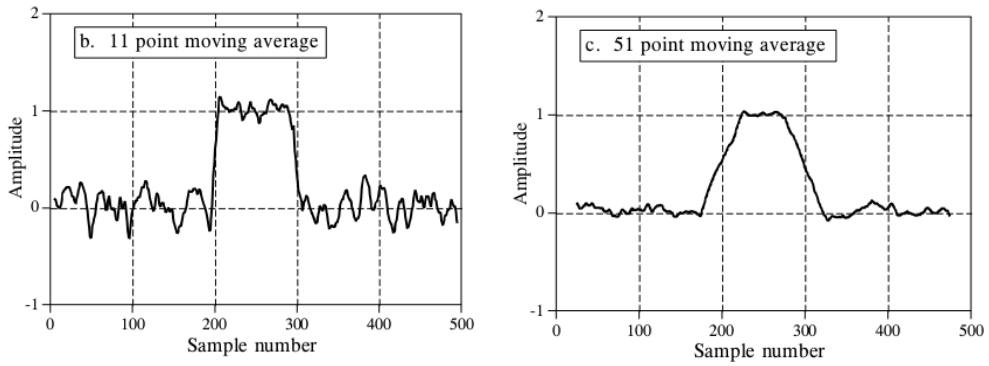
En las gráficas se observa cómo va progresando la salida a medida que progresa la señal de entrada, conforme va realizando la operación de convolución entre los coeficientes del filtro y la señal de entrada.

FUNCIONAMIENTO

Conceptualmente el filtro moving average funciona de la siguiente manera:



Se observa en la figura anterior una señal, la cual se trata de un pulso afectado por ruido.

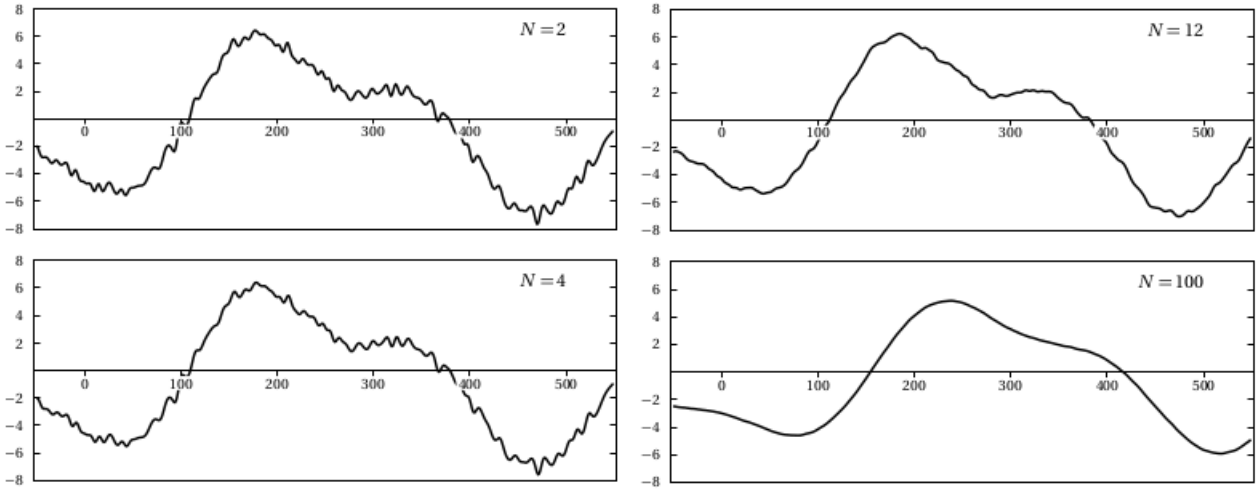


Aquí se observa la salida de dos filtros moving average, el de la izquierda es de orden 11 y el de la derecha de orden 51.

Lo que hace el filtro es tomar todas las muestras de su ventana (cantidad igual al orden), promediarlas y dar un valor de salida. Evidentemente esto está eliminando ruido.

Al aumentar el orden del filtro se reduce aún más el ruido, pero se pierden componentes de alta frecuencia. Esto se observa en la figura derecha, del filtro de orden 51, donde los flancos de subida y bajada de la señal no son tan abruptos como los de la figura a la izquierda, pero presenta menor ruido.

En las siguientes figuras se observa no solo como se reduce el ruido al incrementar el orden del filtro, sino también como se produce un retraso en la señal de salida:

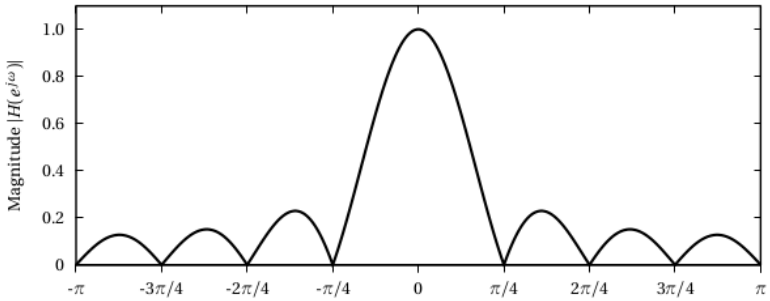


RESPUESTA EN FRECUENCIA

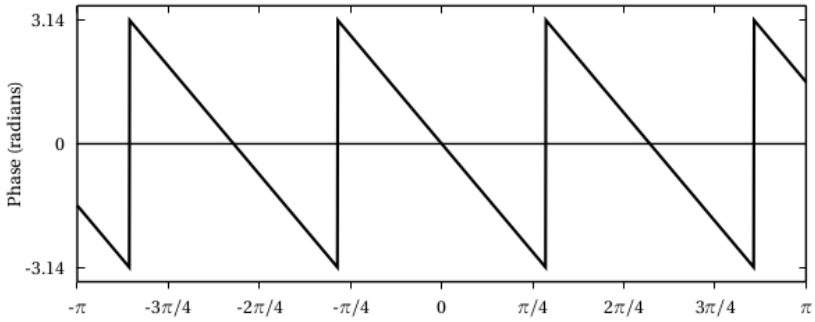
Si bien el filtro moving average es diseñado para filtrar en el dominio del tiempo, como representa una ventana de tiempo en este dominio, su representación en el dominio de la frecuencia va a ser un seno cardinal (una función cuadrada en el dominio del tiempo tiene una representación seno cardinal en el dominio de la frecuencia, y una señal cuadrada en el dominio de la frecuencia es un seno cuadrado en el dominio del tiempo). Matemáticamente es:

$$|H[f]| = \frac{1}{N} \left| \frac{\sin(\pi \cdot f \cdot N)}{\sin(\pi \cdot f)} \right|$$

En la siguiente figura se observa la respuesta del filtro:

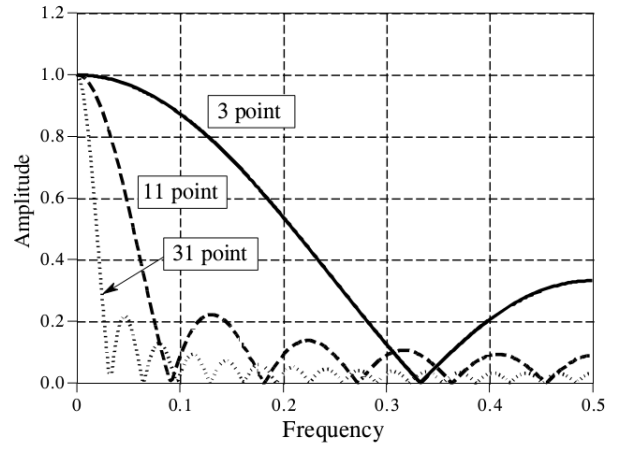


Si observamos esta representación notamos que se trata de un filtro pasa bajos (de mala calidad). Esto tiene sentido ya que este filtro elimina ruido, y el ruido contiene componentes de alta frecuencia.



En esta última figura se representa la respuesta en fase del filtro. Vemos que en la banda pasante la respuesta es lineal, lo cual es típico de todo filtro FIR.

A continuación se observa cómo afecta el orden del filtro a su respuesta en frecuencia:



Vemos la respuesta para filtros de orden 3, 11 y 31. El eje de frecuencia está normalizado según la frecuencia de muestreo, siendo el límite igual a 0.5, es decir, igual a la mitad de la frecuencia de muestreo.

Vemos que a medida que aumenta el orden del filtro, la respuesta se hace más selectiva, pudiendo llegar a solo dejar pasar componentes de valor cercano a 0, es decir, continua. Se trata entonces de un buen filtro de ruido, pero un mal filtro pasabajos.

Debemos entonces desarrollar estrategias para elegir el orden del filtro de tal manera que no elimine componentes de alta frecuencia que son parte de nuestra señal de interés. Para ello se parte de la ecuación en el dominio de la frecuencia del filtro (seno cardinal):

$$|H[f]| = \frac{1}{N}\left|\frac{\sin(\pi \cdot f \cdot N)}{\sin(\pi \cdot f)}\right|$$

Reemplazamos la f por un valor de f_{co} que representa la frecuencia de corte deseada y la amplitud $|H[f]|$ por un valor igual a la amplitud para la frecuencia de corte, es decir 0.707.

$$0.707 = \frac{1}{N}\left|\frac{\sin(\pi \cdot f_{co} \cdot N)}{\sin(\pi \cdot f_{co})}\right|$$

Partiendo de esta ecuación y normalizando la frecuencia de corte como $F_{co} = \frac{f_{co}}{f_s}$:

$$N^2 = \sqrt{\frac{0.885894^2}{F_{co}^2} - 1}$$

Por último definimos un $N_{máx}$ como:

$$N_{máx} = round\left(\sqrt{\frac{0.885894^2 \cdot f_s^2}{f_{co}^2} - 1}\right)$$

Donde la función $round$ redondea al entero más cercano. ¿No debería redondear al menor? Ya que se trata de un $N_{máx}$, por lo que si redonda al inmediato superior no se cumpliría con la f_{co} especificada.

Ejemplo: *si tenemos un robot cuyas dinámicas, estudiadas con un acelerómetro que muestrea con 100 Hz, poseen una frecuencia máxima de 10 Hz, el orden máximo del filtro moving average será:*

$$N_{máx} = round\left(\sqrt{\frac{0.885894^2 \cdot 100^2}{10^2} - 1}\right) = 9$$

¿Qué sucede si aumenta la frecuencia de muestreo?

Señales y filtrado en el dominio de la frecuencia

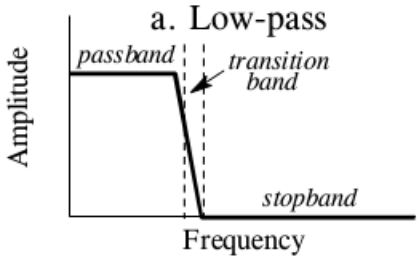
Cuando al información en una señal está contenida en el dominio de la frecuencia, la misma está representada por la respuesta en frecuencia, por la fase y amplitud.

A diferencia de una señal con información en el dominio del tiempo, se requieren de **varias** muestras para hacer un análisis en frecuencia.

La respuesta en frecuencia muestra cómo la información en el dominio de la frecuencia está siendo modificada. Ejemplos de estas señales son los canales telefónicos, ecualizadores de audio, etc.

Algunos parámetros que definen la respuesta en frecuencia de una señal son:

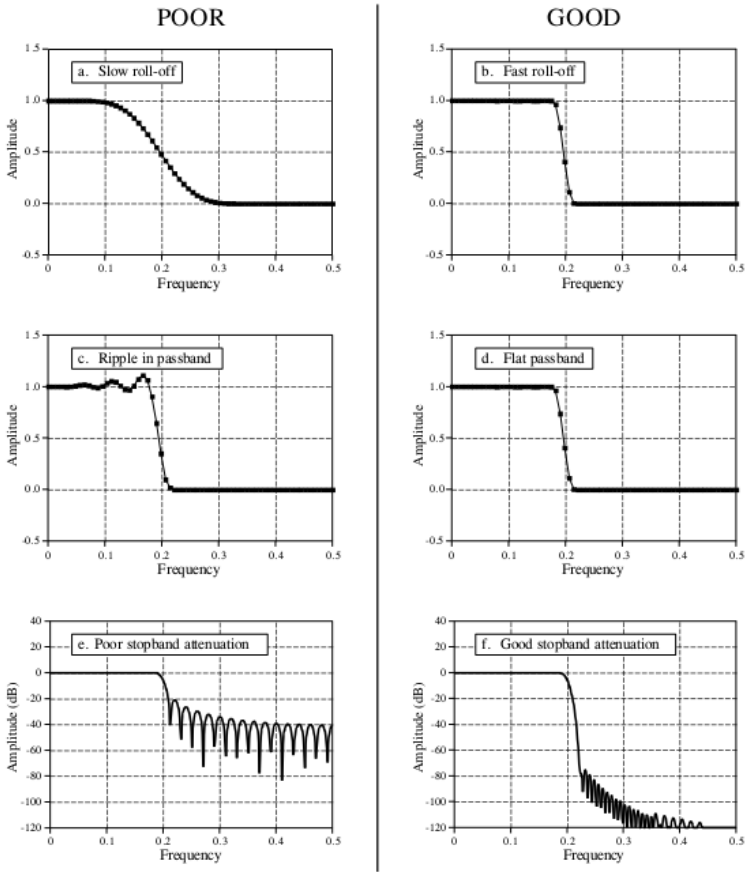
- Banda pasante
- Banda de corte
- Frecuencia de corte
- Banda de transición
- Ripple en banda pasante
- Ripple en banda de corte



En la figura de la derecha se observan estos parámetros en un filtro genérico pasabajos.

Las características que determinan un buen filtro FIR en el dominio de la frecuencia son:

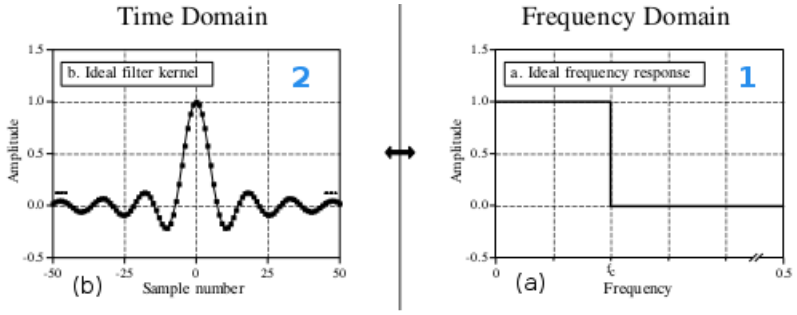
- Banda de transición angosta
- Ripple bajo en banda pasante (plana)
- Atenuación alta en banda de corte o banda suprimida.



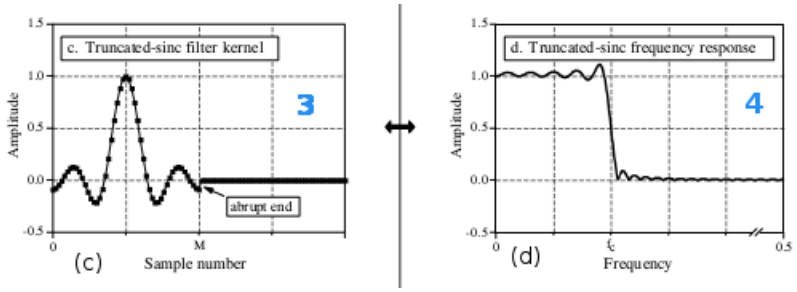
Filtro FIR windowed-sinc

A partir de las siguientes figuras se demostrará el funcionamiento del filtro FIR por el método de ventana (windowed-sinc), en todas ella se ilustra el dominio del tiempo de lado derecho y el dominio de la frecuencia del lado izquierdo.

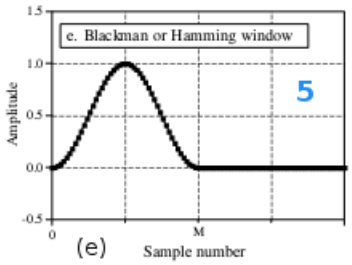
En la siguiente figura vemos la respuesta en frecuencia de un filtro pasabajos ideal. Para este caso en la frecuencia de corte f_c tenemos dos valores (1 y 0), lo cual hace evidente que en la realidad este filtro es imposible. Observamos además que su respuesta en el dominio del tiempo es un seno cardinal de infinitos términos, por lo cual se confirma que es imposible de implementar. Es por ello que debe llegarse a una solución de compromiso para representar este filtro en una computadora.



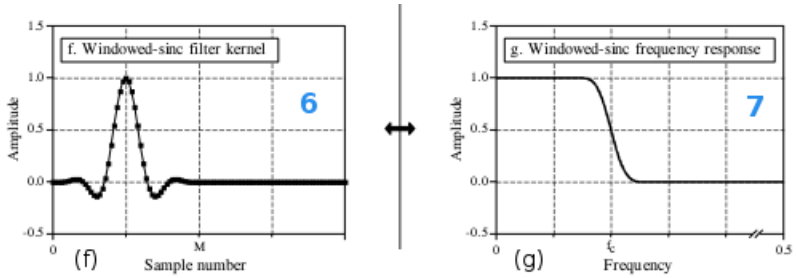
La forma más intuitiva de hacerlo es truncar en algún punto el filtro FIR y quedarnos solo con algunos elementos del seno cardinal. La respuesta en frecuencia resultante posee Ripple tanto en banda pasante como en banda suprimida, lo cual es indeseado.



Para mejorar la respuesta en frecuencia se multiplica elemento a elemento el kernel del filtro truncado por una **ventana**. La ventana se representa con la siguiente figura como una campana de Gauss en el dominio del tiempo.



El resultado de esta multiplicación genera un seno cardinal truncado pero con un corte no abrupto.



Esta ausencia del corte abrupto produce en el dominio de la frecuencia la respuesta de la derecha, donde se ha eliminado prácticamente el ripple en banda pasante, pero con una banda de transición mayor. Este filtro comparado con el generado a partir de simplemente truncar (3 y 4) representa una notable mejora.

Matemáticamente vamos a tener las siguientes ecuaciones:

$$h_w[n] = h_{st}[n] \cdot w[n]$$

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

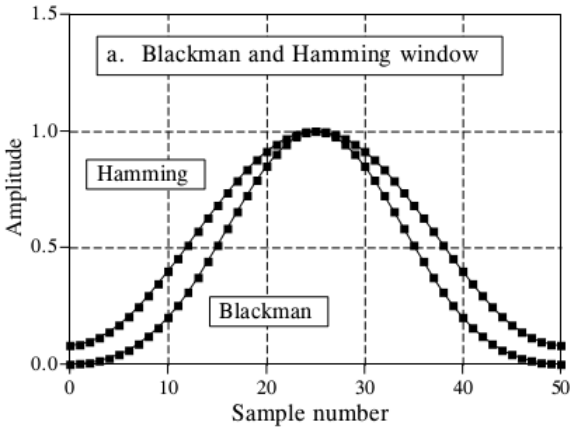
Donde $h_{st}[n]$ representa al kernel del filtro truncado, el cual al ser multiplicado elemento a elemento por una ventana $w[n]$ nos da como resultado un nuevo kernel $h_w[n]$.

Luego a este kernel le implementamos convolución con la señal $x[n]$ para finalmente obtener la salida $y[n]$.

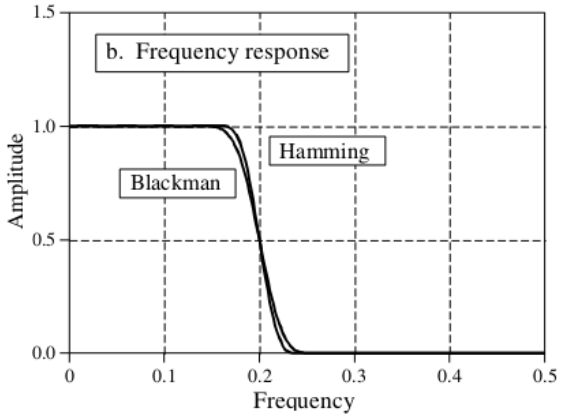
Diferencias entre ventanas

Existen diferentes ventanas, las cuales se aplican según los requisitos del filtro en cuestión. Dos de ellas son las ventanas de **Hamming** y la ventana de **Blackman**.

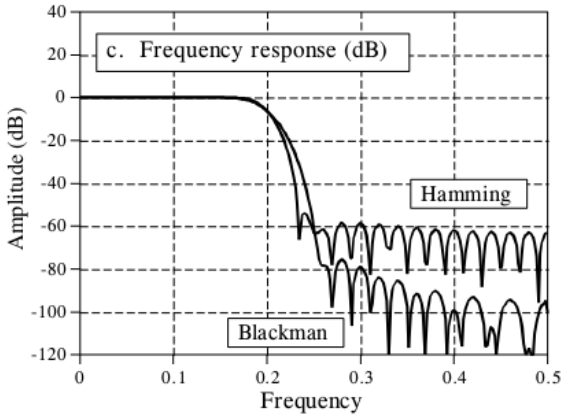
La siguiente figura representa ambas ventanas en el dominio del tiempo, cada una con 51 elementos para este ejemplo.



La respuesta en frecuencia de estas dos ventanas se representa mediante la siguiente figura. En ella se observa que la ventana de Hamming tiene una banda de transición un 20% más angosta que la de Blackman.



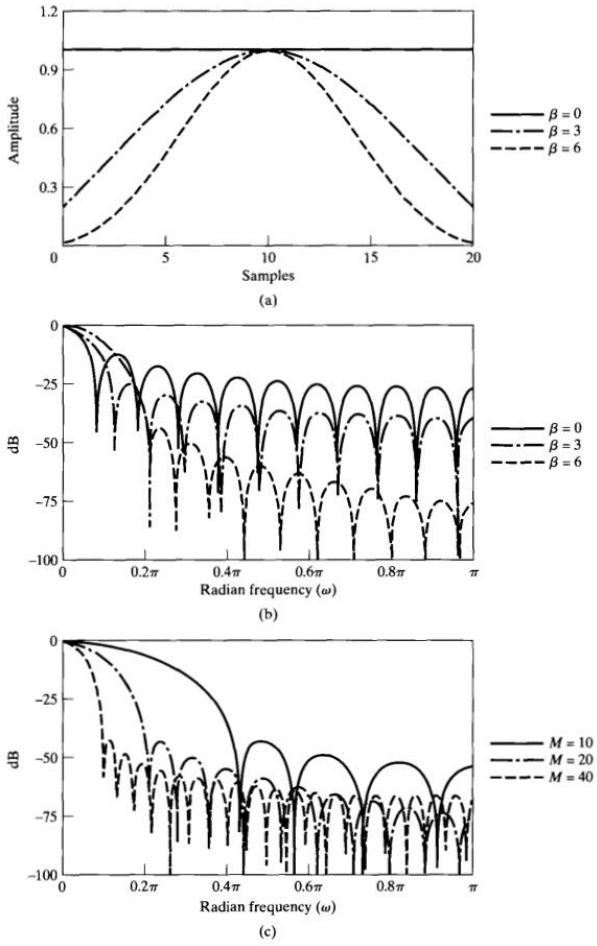
La próxima figura representa la atenuación en banda suprimida de estas dos ventanas. Observamos que la ventana de Hamming tiene una atenuación de -53dB para el primer lóbulo lateral, mientras que la de Blackman -73dB.



Vemos entonces que existe una relación de compromiso entre elegir un ventana u otra. La ventana de Hamming tiene una banda de transición más angosta pero ofrece menor atenuación en banda suprimida, mientras que la de Blackman tiene una transición más ancha pero ofrece mayor atenuación en banda suprimida. Por lo general se comienza eligiendo una ventana de Blackman ya que el ancho de la banda de transición se puede disminuir aumentando el orden de filtro.

Otra ventana es la llamada ventana de **Kaiser**. Esta es una ventana de ventanas. Se configura eligiendo el orden de la ventana (M) y el valor de un parámetro beta, valor con el cual se puede ir intercambiando gradualmente entre distintas ventanas.

Por ejemplo, con un valor de $\beta = 0$ podemos elegir una ventana rectangular, lo cual es igual al truncamiento abrupto realizado anteriormente. A medida que crece el valor de β la campana de Gauss se va haciendo más angosta, con lo cual podemos pasar de una ventana de Hamming a una de Blackman en forma gradual, pudiendo elegir una ventana ubicada entre estas dos. Por otro lado, a medida que variamos el orden del filtro, varia la frecuencia de corte del filtro respecto de la frecuencia de muestreo. Con esta ventana existe una relación de compromiso entre la atenuación y el ancho de banda de la señal pasante.



El siguiente cuadro compara diferentes tipos de ventanas:

Nombre		Ancho de banda de transición normalizado	Ripple en banda pasante en dB	Ripple	Nivel del primer lóbulo lateral en dB	Atenuación en banda suprimida en dB
Rectangular		0.9/N	0.741	0.089	-13	21
Hanning		3.1/N	0.0546	0.063	-31	44
Hamming		3.3/N	0.0194	0.0022	-41	53
Blackman		5.5/N	0.0017	0.000196	-57	74
Kaiser	$\beta = 4.54$	2.93/N	0.0274	-	-	50
	$\beta = 5.65$	3.63/N	0.00867	-	-	60
	$\beta = 6.76$	4.32/N	0.00275	-	-	70
	$\beta = 8.96$	5.71/N	0.000275	-	-	90

En la tabla se observa la relación de compromiso entre el ancho de banda de transición, el ripple y la atenuación en banda suprimida que podemos obtener con cada ventana.

Expresión matemática y mediante diagramas en bloques de filtros fir

La siguiente expresión representa la respuesta al impulso de un filtro FIR expresada usando la transformada Z:

$$H(z) = b_0 + b_1\, z^{-1} + \cdots + b_{M-1}\, z^{M-1}$$

Los términos b representan los coeficientes del filtro FIR y los término z indican retrasos según su exponente.

Se observa que la expresión no contiene un numerador, lo cual se debe a que el filtro no tiene realimentación. Esto significa que no presenta polos, por lo cual se dice que el filtro es **incondicionalmente estable**.

La expresión anterior representada mediante un diagrama de bloques resulta:

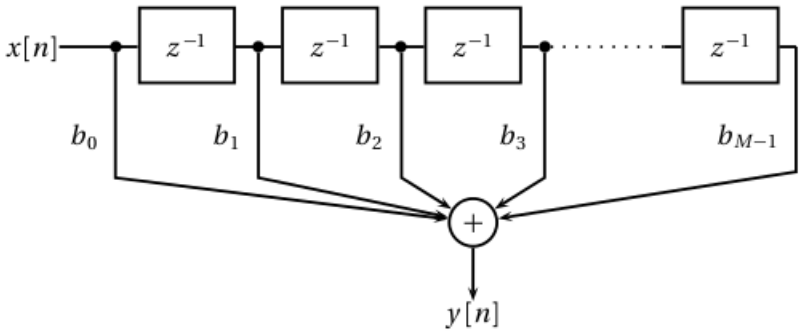


Figure 7.22 Direct FIR implementation.

Este esquema se denomina implementación directa del filtro FIR.

Otra forma de implementar este filtro es mediante una representación transversal, la cual es matemáticamente idéntico a la representación anterior.

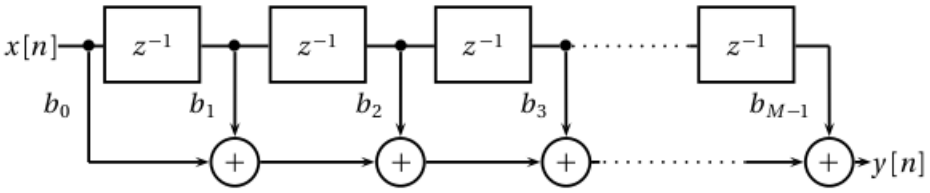


Figure 7.23 Transversal FIR implementation.

Esta implementación se utiliza al realizar la operación MAC, no es más ni menos que implementar la convolución de una señal con los coeficientes del filtro.

FILTROS IIR – FILTRO DE RESPUESTA INFINITA AL IMPULSO

Anteriormente vimos que los filtros digitales tienen 4 posibles categorías, según su estructura (FIR o IIR) y su dominio de funcionamiento (tiempo o frecuencia):

	FIR	*IIR*
Dominio del tiempo	Moving Average	**Leaky Integrator**
Dominio de la frecuencia	Windowed Filters Equirripple Minimax	ZOH method Bilinear z-transform

Filtro Leaky integrator

Como se observa en el cuadro anterior se trata de un filtro IIR en el dominio del tiempo. A continuación se observa la matemática detrás de este filtro:

Comenzamos planteando al ecuación del filtro Moving Average:

$$y[n] = x[n] * h[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

Esta expresión representa el promedio de la señal $x[n]$ en una ventana que va de 0 a $M-1$.

Operamos sobre esta ecuación, sacando un término $x[n]$ de la sumatoria reduciendo el índice de la misma, haciendo que comience desde $k=1$:

$$y[n] = \frac{1}{M}\sum_{k=1}^{M-1} x[n-k] + \boldsymbol{x[n]} = \frac{1}{M}\left[\sum_{k=1}^{M-1} x[n-k]\right] + \frac{1}{M}\,x[n]$$

Por otro lado, planteamos la expresión de un término pasado de la salida del filtro moving average:

$$y[n-1] = \frac{1}{M-1}\left[\sum_{k=1}^{M-1} x[n-k]\right]$$

Se divide por $M-1$ porque tenemos un valor menos, ya no tenemos el valor presente $x[n]$. Operando:

$$y[n-1](M-1) = \left[\sum_{k=1}^{M-1} x[n-k]\right]$$

Si reemplazamos este ecuación en la ecuación de $y[n]$ obtenida anteriormente obtenemos:

$$y[n] = \frac{M-1}{M}\,y[n-1] + \frac{1}{M}x[n]$$

Definimos un nuevo término $\lambda = \frac{M-1}{M}$ y reemplazamos para obtener finalmente:

$$\boldsymbol{y[n] = \lambda\, y[n-1] + (1-\lambda)x[n]}$$

Esta es la expresión final del filtro Leakey integrator. La misma depende de un valor pasado de la salida y del valor actual de la entrada. Es por ello que claramente se trata de un filtro IIR, no de un filtro FIR.

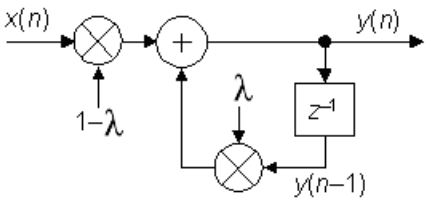
De esta ecuación final podemos determinar las siguientes conclusiones:

- Ya no se trata de una convolución.
- Se trata de una ecuación en diferencias de coeficientes constantes (diferencial discreta) por lo que necesita que se fijen condiciones iniciales, en este caso es el valor de pasado de $y[n-1]$ que suele igualarse a cero.
- También es conocido como **filtro recursivo de polo simple**.
- La ecuación representa un sistema lineal invariante en el tiempo (LTI).
- El sistema es estable siempre que $\lambda < 1$, lo cual se cumple siempre ya que $\lambda = \frac{M-1}{M}$.
- El valor λ regula cuanto suavizado o filtrado va a tener el sistema.

Aplicando transformada Z a la ecuación del sistema obtenemos:

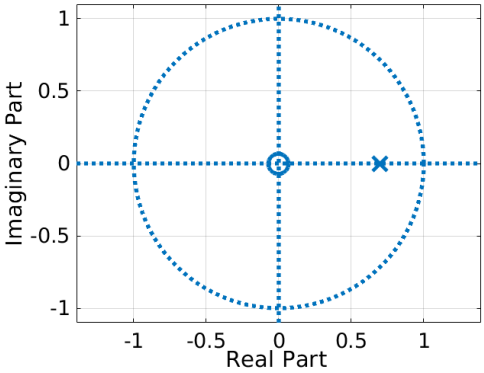
$$\frac{Y(z)}{X(z)} = \frac{1-\lambda}{1-\lambda z^{-1}}$$

Esta es la ecuación característica del sistema y se puede representar en bloques de la siguiente manera:



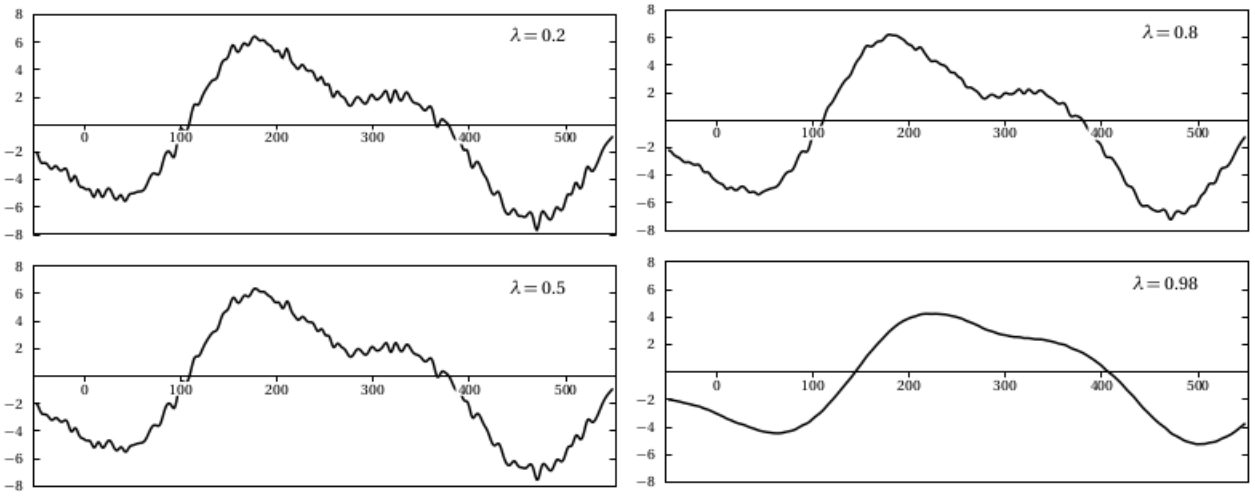
Donde se observa la realimentación de la salida hacia la entrada.

Si analizamos la constelación de polos y ceros obtenemos:

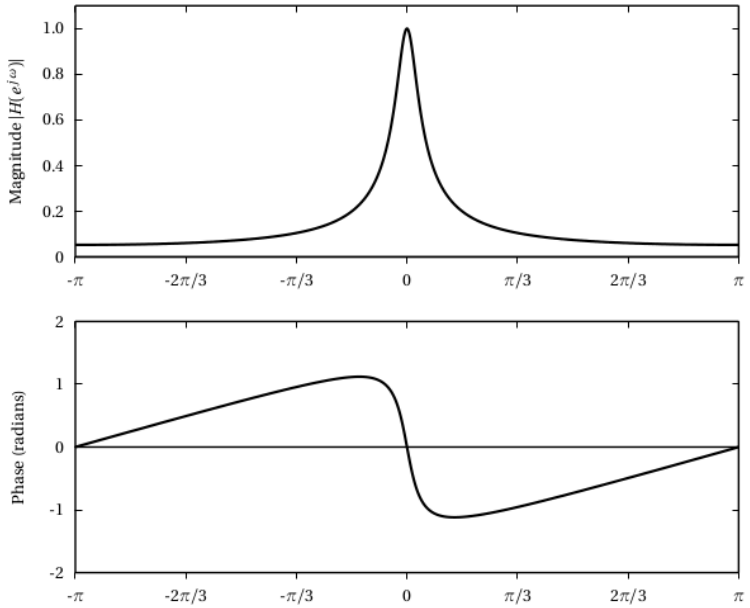


Como el polo debido a λ siempre se encuentra dentro de la circunferencia de radio 1, ya que siempre $\lambda < 1$, el sistema en principio siempre será estable. El problema de inestabilidad puede presentarse al encontrarse este polo relativamente cerca de la circunferencia, donde generará una inestabilidad no permanente o intermitente debida a problemas o ruido de redondeo que lo coloquen por encima de la circunferencia. El cero siempre va a estar cercano a cero, no igual a cero.

En las siguientes figuras se observa la respuesta del filtro ante una entrada ruidosa al ir aumentando el valor de λ:



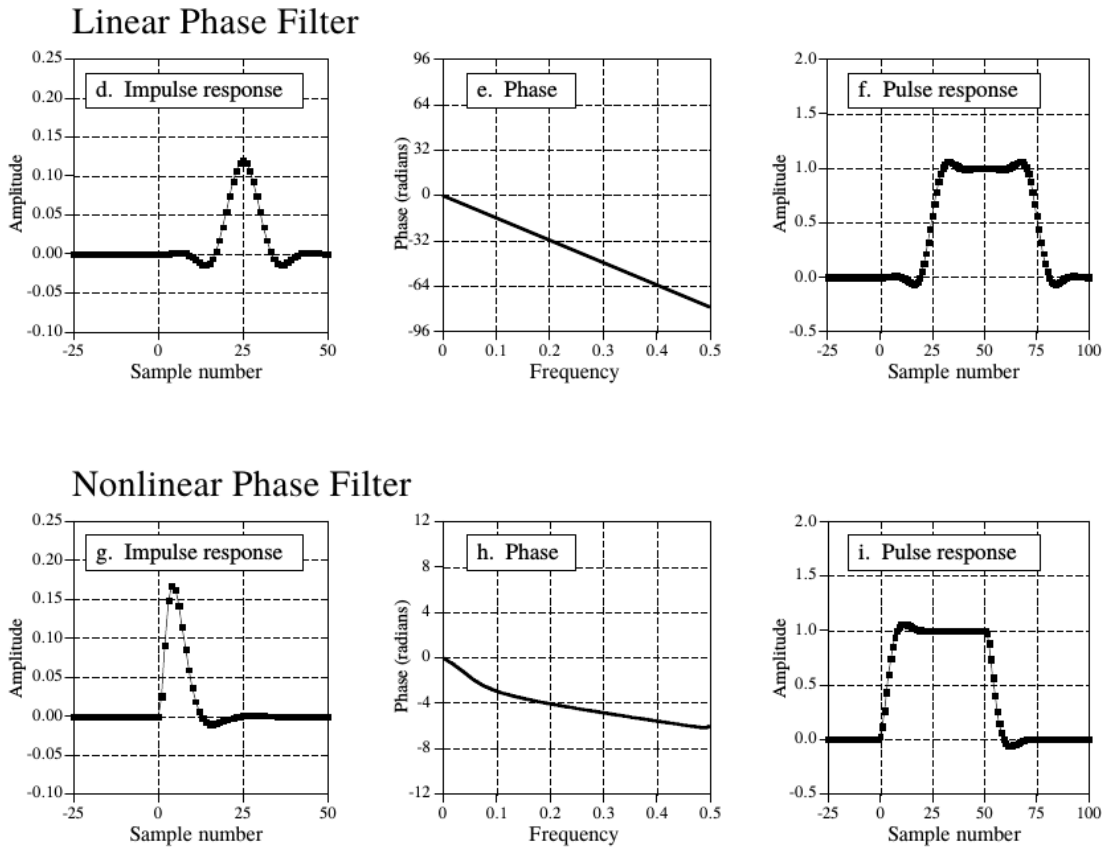
En la siguiente figura se observa la respuesta en frecuencia para un $\lambda = 0.9$:



Se trata de un filtro pasabajos (malo). A medida que aumenta el λ la respuesta en frecuencia se hace más selectiva.

La respuesta en fase no es lineal en la banda pasante. Esto es típico de filtros IIR.

En la siguiente figura se muestra la diferencia de respuesta en frecuencia de un filtro lineal y un filtro no lineal en fase:



Se observa que la respuesta es simétrica si el filtro es lineal en fase. En el caso de los filtros no lineales en fase, la respuesta es asimétrica y dicha asimetría se da por la deformación que sufre la señal, la cual es la diferencia en los flancos de subida y de bajada.

Diseño de filtros IIR usando transformada bilineal

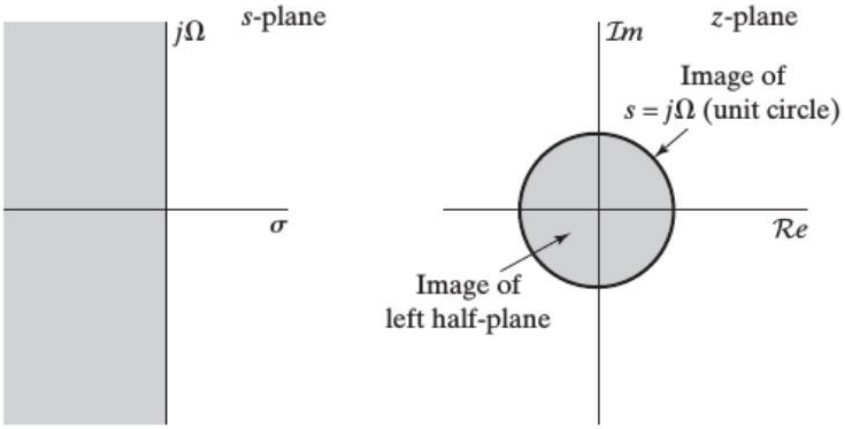
Según la clasificación usada hasta el momento, este tipo de filtro se ubica en las siguientes categorías:

	FIR	*IIR*
Dominio del tiempo	Moving Average	Leaky Integrator
Dominio de la frecuencia	Windowed Filters Equirripple Minimax	ZOH method **Bilinear z-transform**

La idea central detrás del uso de la transformada z bilineal es tomar un filtro que fue desarrollado en el dominio analógico y transformarlo en un filtro de dominio discreto (digital).

Para ello nos centramos en cómo hacer para representar la variable s (de la transformada de Laplace) en el dominio de z, es decir, como llevar del dominio de la frecuencia analógico al dominio de la frecuencia digital.

De este modo, toda la teoría detrás del diseño de filtros analógicos será reutilizada para implementar filtros digitales (filtros Butterworth, Chebyshev, Elíptico, etc.).



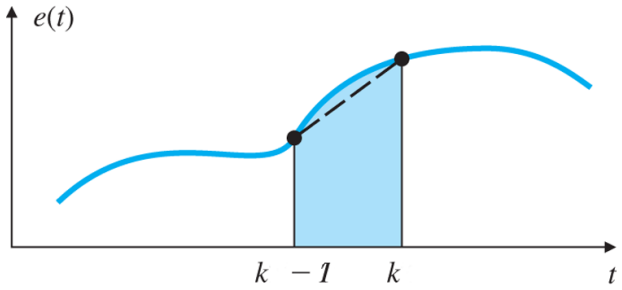
TRANSFORMADA BILINEAL (MÉTODO DE TUSTIN)

Suponemos el siguiente sistema integrador:

$$\frac{U(s)}{E(s)} = D_C(s) = \frac{1}{s}$$

El área bajo la curva de la función graficada en el tiempo, es decir, la integral de la misma es:

$$u(k) = \int_0^{k-1} e(t)\,dt + \int_{k-1}^{k} e(t)\,dt$$



El método de Tustin utiliza integración trapezoidal (recta segmentada en negro que une los puntos de la figura), para aproximar $e(t)$ por un línea recta entre dos muestras. La técnica es una transformación algebraica entre variables s y z.

Esta integración trapezoidal se plantea en la siguiente ecuación:

$$u(k) = u(k-1) + \frac{T}{2}\,[e(k-1) + e(k)]$$

Ahora $u(k)$ es igual a la integral de la función hasta $u(k-1)$ más el valor promedio entre $e(k-1)$ y $e(k)$.

Si aplicamos a esta expresión la transformada Z tenemos:

$$U(s) = Z^{-1}\,U(z) + \frac{T}{2}\,[z^{-1}\,E(z) + E(z)]$$

Hacemos factor común y agrupamos los valores de $U(z)$ y $E(z)$:

$$U(z)(1 - z^{-1}) = \frac{T}{2}[E(z)(1 + z^{-1})]$$

Llegamos finalmente a:

$$\frac{U(z)}{E(z)} = \frac{T}{2}\left(\frac{1 + z^{-1}}{1 - z^{-1}}\right) = \frac{1}{\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)}$$

Comparando este última ecuación con la del integrador $\left(\frac{1}{s}\right)$, obtenemos que:

$$\boldsymbol{s \cong \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)}$$

Esta expresión es la denominada **Transformada Bilineal** y nos permite digitalizar reemplazando s por al expresión dada.

RELACIÓN ENTRE FRECUENCIAS DIGITALES Y ANALÓGICAS

Teniendo en cuenta que:

- Ω es la frecuencia analógica, puede tomar valores $-\infty < \Omega < +\infty$
- ω es la frecuencia "digital", puede tomar valores $-\pi < \omega < +\pi$, es decir $-2\pi\frac{f_s}{2} < \omega < +2\pi\frac{f_s}{s}$

Para encontrar la relación entre estas dos frecuencias hacemos que $s = j\Omega$ (evaluada solo sobre el eje imaginario), por lo que z debe ser evaluada solo sobre la circunferencia de radio 1, es decir $z = e^{j\omega}$. Planteamos ahora la transformada bilineal:

$$s = \frac{2}{T}\left(\frac{1 - e^{-j\omega}}{1 + e^{-j\omega}}\right) = \frac{2}{T}\left[\frac{2e^{\frac{j\omega}{2}}\left(j\sin\left(\frac{\omega}{2}\right)\right)}{2e^{\frac{j\omega}{2}}\left(\cos\left(\frac{\omega}{2}\right)\right)}\right] = j\frac{2}{T}\tan\left(\frac{\omega}{2}\right)$$

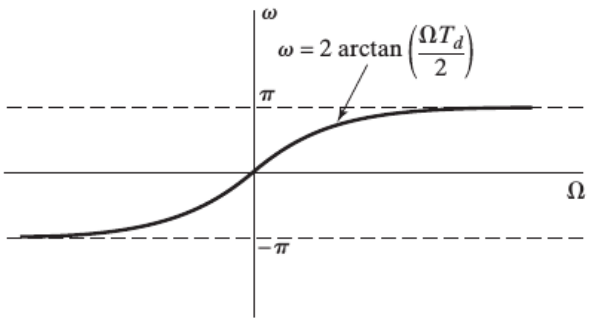
Podemos decir entonces que para una $s = \sigma + j\Omega$:

$$\sigma = 0$$
$$\Omega = \frac{2}{T}\tan\left(\frac{\omega}{2}\right)$$

De la misma forma:

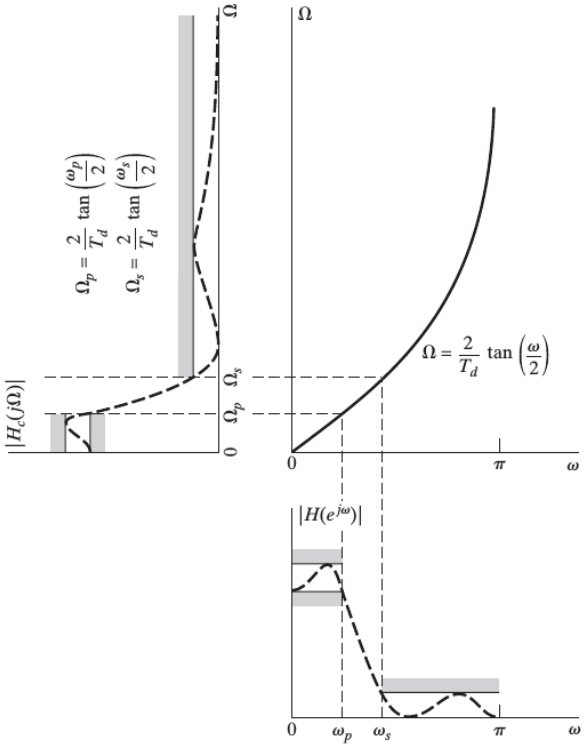
$$\omega = \arctan\left(\Omega\frac{T}{2}\right)$$

Claramente existe una relación no lineal entre las dos frecuencias. Gráficamente resulta:



Esta relación no lineal entre la frecuencia analógica y la frecuencia digital plantea algunos recaudos a tener al momento de diseñar.

Las frecuencias analógicas deben ser ajustadas antes del diseño del filtro analógico. Esto se realiza mediante la técnica de **pre-warping** o pre-combado de frecuencia. Esta técnica se basa en elegir una frecuencia de corte para un filtro analógico, tal que cuando este filtro sea digitalizado con la transformada bilineal obtengamos en el dominio discreto la frecuencia de corte que necesitamos para un filtro particular.



EJEMPLO DE DISEÑO IIR USANDO TRANSFORMADA BILINEAL

1- Tomamos un filtro analógico que cumple con el rendimiento deseado. Por ejemplo, un filtro Butterworth pasa bajos de segundo orden:

$$G(s) = \frac{\Omega_c^2}{s^2 + s\sqrt{2}\Omega_c + \Omega_c^2}$$

2- Normalizamos la frecuencia de corte digital y definimos la frecuencia de muestreo:

$$f_{dc} = 100\,Hz, \qquad f_s = 1000\,Hz, \qquad T = 0.001\,s$$

$$\omega_c = 2\pi\frac{100}{1000} = 0.628\,rad/s$$

Donde f_{dc} es la frecuencia de corte discreta, f_s la frecuencia de muestreo, T el periodo de muestreo y ω_c la frecuencia de corte digital normalizada según la frecuencia de muestreo.

3- Pre-combado de la frecuencia analógica:

$$\Omega_c = \frac{2}{T}\tan\left(\frac{\omega_c}{2}\right) = \frac{2}{0.001}\tan\left(\frac{0.628}{2}\right) = 649.839\ rad/s$$

Si llevamos esta frecuencia a Hz, la frecuencia de corte analógica resulta:

$$f_{ac} = \frac{\Omega_c}{2\pi} = 103.42\ Hz$$

Este valor f_{ac} es el que debo utilizar como frecuencia de corte en el dominio analógico para obtener una frecuencia de corte de $100\ Hz$ en el dominio digital.

4- Tomamos la expresión del filtro Butterworth de segundo orden supuesta anteriormente y reemplazamos el Ω_c obtenido y s con la transformada bilineal $s \cong \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$:

$$H(s) = \frac{\Omega_c^2}{s^2 + s\sqrt{2}\,\Omega_c + \Omega_c^2}$$

Al reemplazar pasamos de una expresión en s a una expresión en z:

$$H(z) = \frac{649.84^2}{\left(\frac{2}{T}\right)^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)\sqrt{2}\,(649.84) + 649.84^2}$$

Quedando finalmente:

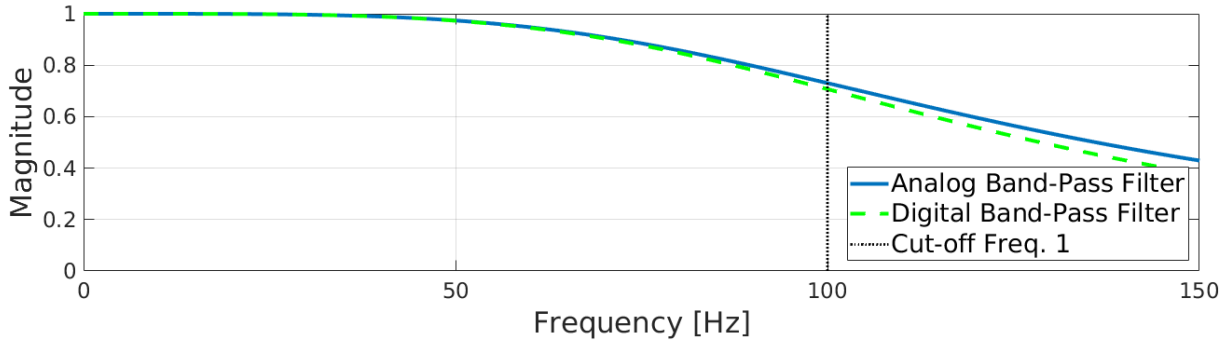
$$H(z) = \frac{0.067 + 0.135z^{-1} + 0.067z^{-2}}{1 - 1.143z^{-1} + 0.413z^{-2}}$$

Esta es la expresión del filtro digital que representa a un filtro analógico pasa bajo Butterworth de segundo orden. Se observa que también el filtro digital es de segundo orden.

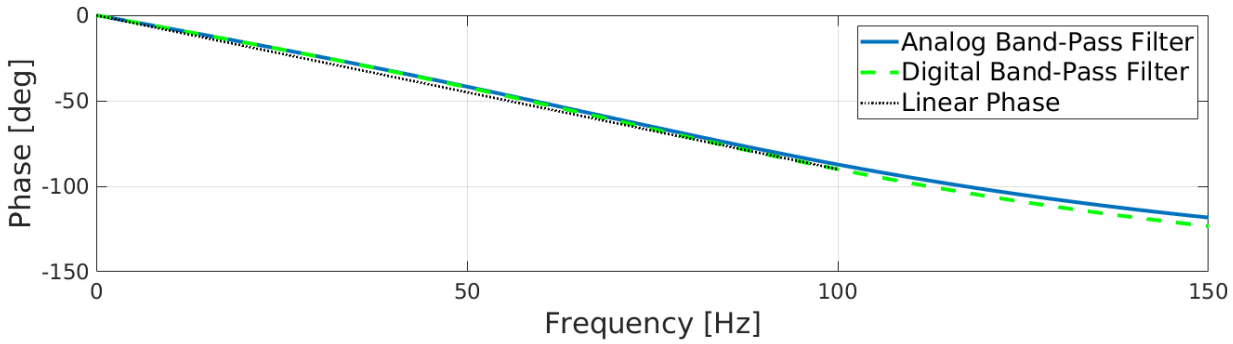
5- Como último paso anti transformamos la ecuación en el dominio z para encontrar la ecuación en diferencias:

$$y[n] = 0.067\,x[n] + 0.135\,x[n-1] + 0.067\,x[n-2] + 1.143\,y[n-1] - 0.413\,y[n-2]$$

En las siguientes figura se observa la comparación de la respuesta en frecuencia y en fase del filtro analógico y el digital:

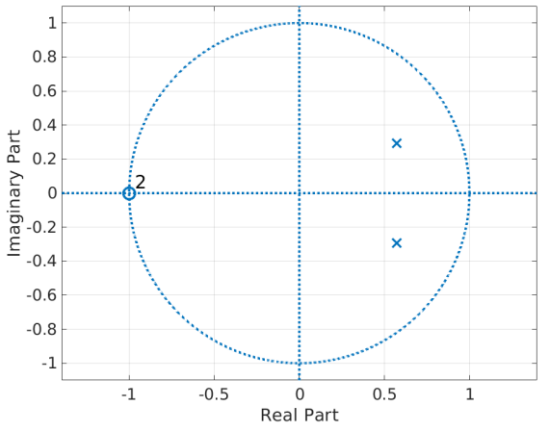


El corte, es decir, el nivel de atenuación de 3dB ocurre exactamente en los 100 Hz del filtro digital. Sin embargo en el filtro analógico, a 100 Hz la atenuación es algo menor. Esto se debe a que en realidad el corte de filtro analógico por el pre-combado quedo en aproximadamente 103 Hz.



La línea negra punteada representa una respuesta lineal en fase. Como la respuesta en fase del filtro Butterworth es moderadamente no lineal, el filtro digital copia o sigue esta forma de respuesta en fase.

Para analizar la estabilidad del filtro analicemos la constelación de polos y ceros del sistema:



Tenemos dos polos y dos ceros al tratarse de un sistema de orden 2. Se observa que los ceros son coincidentes y los polos son complejos conjugados y se ubican dentro de la circunferencia unitaria, alejados de la misma, por lo que la estabilidad no se ve comprometida.

Implementación Direct Form I de filtros IIR

$$H(z) = \frac{b_0 + b_1\, z^{-1} + \cdots + b_{N-1} z^{N-1}}{1 + a_1\, z^{1} + \cdots + a_{M-1}\, z^{M-1}}$$

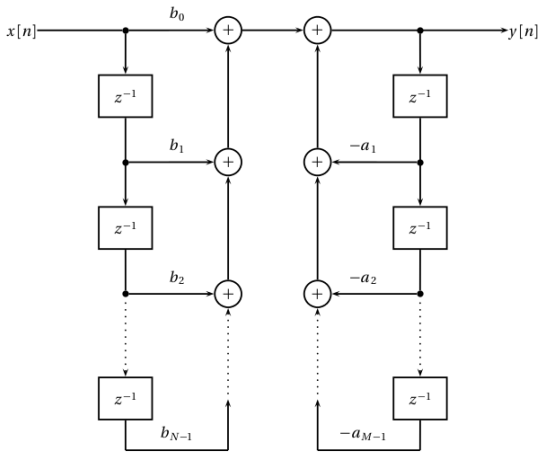


Figure 7.24 Direct Form implementation of an IIR filter.

Implementación Direct Form II de filtros IIR

Por la propiedad conmutativa de la transformada z, podemos invertir el orden de calculo para convertir la estructura Direct Form I en una nueva estructura invertida:

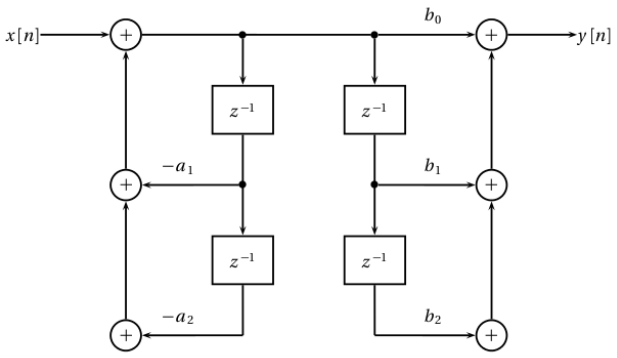


Figure 7.25 Direct form I with inverted order.

Si unimos los dos bloques z^{-1} enfrentados logramos la implementación Direct Form II, la cual tiene la ventaja de reducir el número de elementos de Delay requeridos (ahorro de memoria):

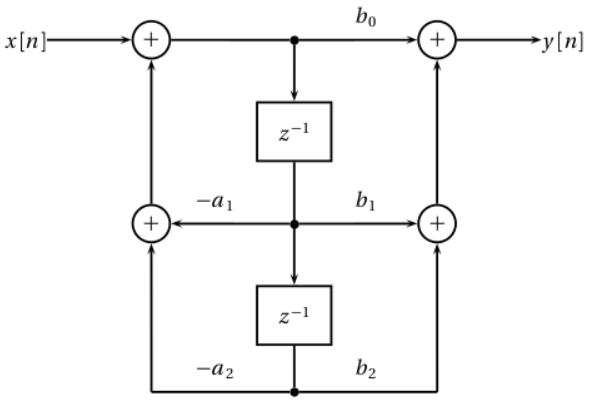


Figure 7.26 Direct Form II implementation of a second-order section.

Esta implementación es útil para representar sistemas de mayor orden en cascada.

Implementación en cascada de filtros IIR

La estructura en cascada de N secciones de segundo orden es mucho menos sensible a errores de cuantización que la implementación previa Form Direct II de orden $2 \cdot N$.

$$H(z) = \prod_{k=1}^{N} G_k \frac{b_{0k} + b_{1k}\, z^{-1} + b_{2k}\, z^{-2}}{1 + a_{1k}\, z^{-1} + a_{2k} z^{-2}}$$

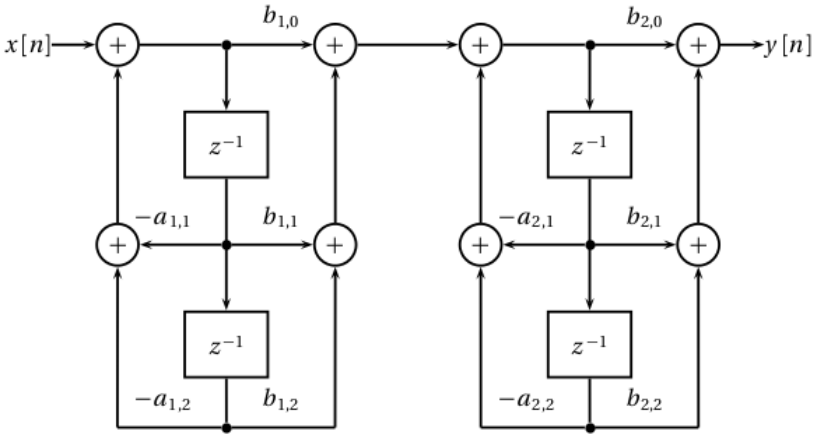


Figure 7.27 4th order IIR: cascade implementation.

El término G_k en la ecuación representa a una ganancia que normalmente se ubica entre las implementaciones Direct Form II en cascada, para el diagrama anterior al no aparecer significa que son de valor unitario.