

14 Tratamiento digital de señales

El tratamiento digital de señales es una línea de la tecnología demasiado extensa para ser tratada en un solo capítulo; sin embargo, este capítulo se centra en las nociones de adquisición, tratamiento y reconstrucción de señales. Cabe denotar que el alcance de los microcontroladores PICMicro de baja gama como los 16F y 18F sólo permiten realizar tratamientos sobre señales de baja frecuencia. De todos modos, muchas aplicaciones son de utilidad bajo estas características tales como sistemas de comunicación, generación de tonos DTMF y adquisición de señales bioeléctricas, entre otras. Si el desarrollador desea trabajar y tratar señales de mayor espectro, realizando tratamientos de mayor complejidad será necesario trabajar con unidades de procesamiento más robustas como la familia de microcontroladores PICMicro ds. Estos microcontroladores no pueden ser programados con el compilador MikroC PRO, para estos se debe utilizar el compilador MikroC PRO para dsPIC.

Un diagrama en bloques del procesamiento digital de una señal se puede apreciar en la siguiente figura:

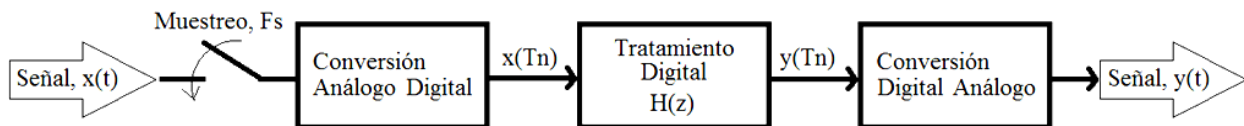


Figura 14-1

14.1 Muestreo

Como primera medida se requiere adquirir muestras periódicamente de la señal analógica, esto se logra haciendo una conversión analógica digital, con una frecuencia constante denominada frecuencia de muestreo F_s . El periodo de muestreo T , es el tiempo en segundos que existe entre muestra y muestra, y a su vez es el inverso de la frecuencia de muestreo $T = 1/F_s$. El muestreo permite convertir una señal continua en una señal discreta, es decir que pasa de $x(t)$ a $x(Tn)$, donde T es el periodo de muestreo, y n es el número de la muestra digitalizada. Para definir el periodo de muestreo T , o lo que es igual la frecuencia de muestreo F_s , se debe tener la siguiente consideración: La frecuencia de muestreo debe ser mayor que la máxima componente espectral contenida en la señal $x(t)$. Por ejemplo, si la máxima frecuencia contenida en la señal $x(t)$ es 500Hz, la frecuencia de muestreo debe ser como mínimo el doble, es decir $F_s = 2 \times 500\text{Hz} = 1000\text{Hz}$. Este concepto es conocido como teorema de muestreo de Nyquist. Sin embargo la frecuencia de muestreo puede ser mayor al doble del máximo en frecuencia de la señal $x(t)$, de hecho cuanto mayor sea la frecuencia de muestreo mayor será la fidelidad de la señal digitalizada, pero a su vez mayor serán los recursos requeridos en velocidad, y memoria de procesamiento.

$$F_s \geq 2F_{MAX} \quad \text{Ecuación 14-1}$$

14.2 Funciones de transferencia

Una función de transferencia es la relación que existe entre la entrada de una señal y su salida al pasar por el bloque de proceso. Las funciones de transferencia se estudian y se manipulan en términos de la frecuencia compleja y no en el dominio del tiempo continuo. Esto se debe a que al convertir las funciones a la frecuencia compleja las operaciones que implican manipulación de números complejos y sistemas integro diferenciales que se hacen lineales, y esto simplifica su tratamiento. Las funciones de transferencia para el caso de los sistemas de tiempo discreto se hacen en términos de la variable compleja z , para el tratamiento de estos sistemas se recurre a la transformación z , que representa la frecuencia compleja para el tiempo discreto. Las funciones de transferencia se denotan por excelencia con la letra H , para las funciones en términos de la variable z , y con h , para las funciones en términos del tiempo discreto T_n . A continuación se puede ver un ejemplo:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A}{B} \quad \text{Ecuación 14-2}$$

De la ecuación anterior se puede deducir que:

$$H(z)X(z) = Y(z) \quad \text{Ecuación 14-3}$$

Como se puede ver en la anterior ecuación la salida Y , de un sistema discreto es igual a la entrada X , multiplicada por la función de transferencia H . Se debe recordar que el tratamiento real y la programación se realizan en términos del tiempo discreto, por esta razón siempre se deberá realizar la transformada inversa de z . La transformación inversa de la ecuación anterior da como resultado la siguiente ecuación en función del tiempo discreto T_n :

$$h(n) * x(n) = y(n) \quad \text{Ecuación 14-4}$$

De la anterior ecuación se puede observar que se omite la escritura del periodo T , dado que este es un valor constante, y no tiene relevancia escribirlo siempre. Por otro lado se puede observar que la función h y la señal x , no se multiplican como en la frecuencia compleja, en este caso al realizar la transformación su equivalencia es la convolución. Para el caso del tratamiento de señales, la longitud de $h(n)$, es finita, y el número máximo que asume n se denomina M , este a su vez determina el orden de la función de transferencia. Por otra parte la señal de entrada $x(n)$, es una serie de muestras de longitud indeterminada, igual que la salida $y(n)$.

De lo anterior se puede concluir que para realizar un tratamiento digital sobre una señal se debe realizar la operación de convolución continua entre los coeficiente de la función $h(n)$ y la señal $x(n)$. Sin embargo la operación de convolución puede ser de dos formas, una en la cual la convolución se hace con las muestras pasadas y actuales de la señal $x(n)$, esto se conoce como sistema sin memoria o sistemas FIR (Respuesta Finita al Impulso). Por otra parte cuando la convolución requiere información de las muestras pasadas de $x(n)$, y de $y(n)$, estos sistemas se denominan sistemas con memoria o IIR (Respuesta Infinita al Impulso). Los sistemas IIR son sistemáticamente más simples, en su implementación y requieren de campos de memoria pequeños, por otra parte realizar su diseño y lograr su estabilidad es más complicado y requiere de tediosos procesos matemáticos. Los sistemas FIR, ofrecen características contrarias a los sistemas IIR, los sistemas FIR, son de fácil implementación y diseño, pero consumen recursos de

memoria y procesamiento mayor, una de las virtudes de estos sistemas es que siempre son estables. Los análisis y ejemplos de este capítulo se concentrarán en los sistemas FIR e IIR.

14.3 Convolución

Para iniciar es importante conocer la estructura de una convolución continua en forma matemática y por medio de un segmento de código en lenguaje C:

$$y(n) = \sum_{k=-\infty}^{+\infty} [h(k)x(n-k)] \quad \text{Ecuación 14-5}$$

La ecuación anterior describe la forma general de la convolución, sin embargo se debe recordar que la longitud de la función $h(n)$, es finita y su máximo es M , por lo tanto la ecuación se puede reescribir de la siguiente forma:

$$y(n) = \sum_{k=0}^M [h(k)x(n-k)] \quad \text{Ecuación 14-6}$$

Cada vez que una muestra de la salida $y(n)$, es calculada por medio de la convolución, se requieren M , muestras de la señal $x(n)$, incluida la muestra actual, esto quiere decir que para hacer la convolución se debe tener presente la necesidad de un campo de memoria igual a M para guardar las últimas muestras durante el proceso.

A continuación se puede observar un ejemplo de cómo se implementar una convolución, en lenguaje C:

```
//Orden del sistema FIR de orden 17, M=17.
#define M 17
float x[M];
float h[M];

//Rutina para hacer la convolución.

float yn=0.0;
short k;
for( k=M-1; k>=1; k-- )x[n]=x[n-1];
x[0]=x0;
for( k=0; k<M; k++ )yn += h[k]*x[k];
```

14.4 Filtros FIR

El diseño de los filtros FIR, es relativamente simple y utiliza estructuras ya definidas para cada tipo de filtro. Los filtros pueden ser de cinco naturalezas: Pasa bajas, pasa altas, pasa bandas, rechaza banda, y multi banda. Para el tratamiento de los filtros se debe tener presente las siguientes consideraciones:

Los cálculos de los filtros se hacen en radianes y no en hercios.

La frecuencia de muestreo en hercios F_s , es equivalente a una frecuencia en radianes/segundo de 2π .

La máxima frecuencia tratada por los filtros es $F_s/2$, o π Radianes.

La relación que existe entre radianes y hercios es: $\omega = 2\pi F$, y $F = \omega/2\pi$.

Las frecuencias de corte de los filtros se denotan como ω_c , y F_c .

La frecuencia de corte se calcula como $\omega_c = 2\pi F_c / F_s$.

La banda de transición del filtro se denota como $d\omega$ en radianes/segundo y dF en hercios.

El orden de los sistemas FIR se denota como M .

Se recomienda usar un número impar para M .

A continuación se estudiará la forma de diseño para cada uno de estos filtros.

14.4.1 Filtro Pasa bajas

Los filtros pasa bajas se caracterizan por tener una frecuencia de corte a partir de la cual las frecuencias inferiores son permitidas, y las frecuencias superiores son atenuadas. La función de transferencia $h(n)$, para este filtro es:

$$h(n) = \begin{cases} \frac{\text{Sen}(\omega_c n)}{\pi n}, & n \neq 0 \\ \frac{\omega_c}{\pi}, & n = 0 \end{cases} \quad \text{Ecuación 14-7}$$

$$-\frac{M}{2} < n < \frac{M}{2}$$

La respuesta espectral de este filtro de orden 7, en escala lineal es la siguiente:

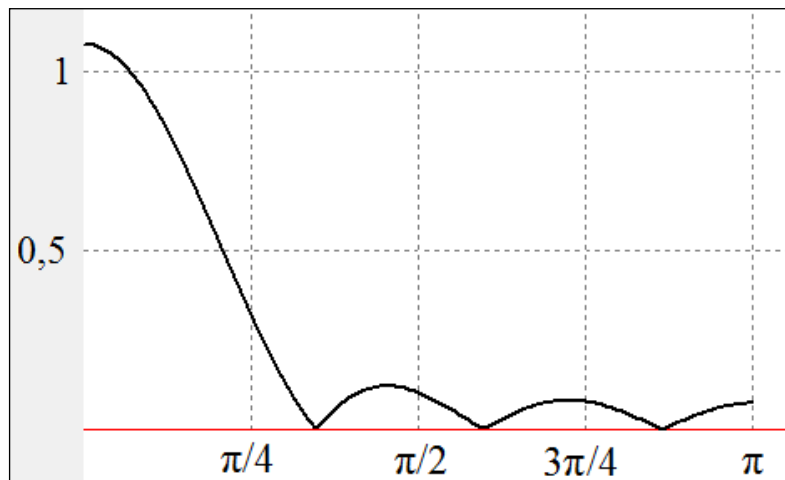


Figura 14-2

14.4.2 Filtros Pasa Altas

Los filtros pasa altos permiten el paso de las frecuencias superiores a la frecuencia de corte, y suprimen las inferiores a la misma. El cálculo de la función de transferencia $h(n)$, para estos filtros está definida por la siguiente ecuación:

$$h(n) = \begin{cases} \frac{-\text{Sen}(w_c n)}{\pi n}, & n \neq 0 \\ 1 - \frac{w_c}{\pi}, & n = 0 \end{cases} \quad \text{Ecuación 14-8}$$

$$-\frac{M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

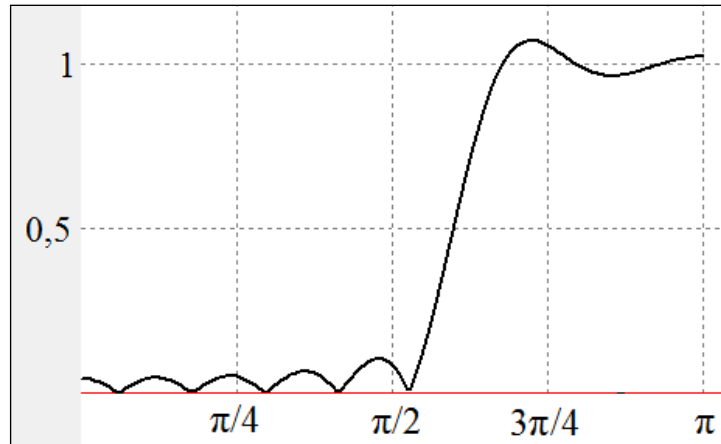


Figura 14-3

14.4.3 Filtro Pasa Banda

Los filtros pasa banda permiten el paso de una porción de frecuencias entre W_{c1} y W_{c2} , y el resto de frecuencias son eliminadas. La función de transferencia para este tipo de filtros se calcula por medio de la siguiente ecuación:

$$h(n) = \begin{cases} \frac{\text{Sen}(w_{c2}n) - \text{Sen}(w_{c1}n)}{\pi n}, & n \neq 0 \\ \frac{w_{c2} - w_{c1}}{\pi}, & n = 0 \end{cases} \quad \text{Ecuación 14-9}$$

$$-\frac{M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

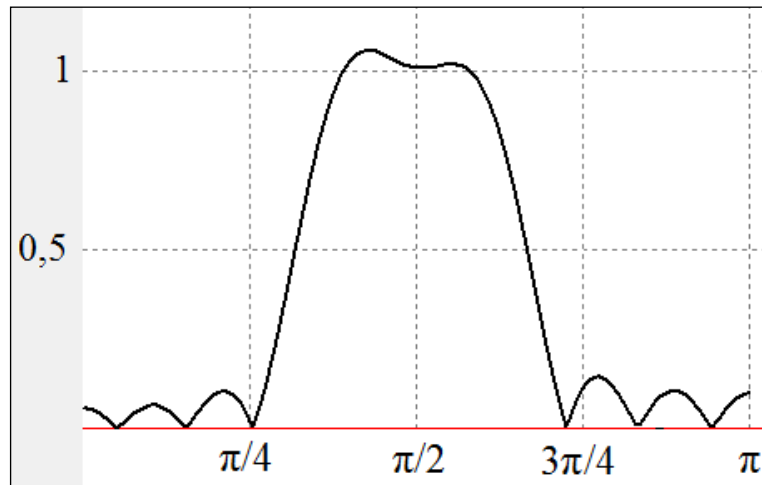


Figura 14-4

14.4.4 Filtro Rechaza Banda

Los filtros rechaza banda tienen un comportamiento similar a los filtros pasa banda, y su proceder es inverso a los filtros pasa banda, la función que caracteriza los coeficientes de la función de transferencia es la siguiente:

$$h(n) = \begin{cases} \frac{\text{Sen}(w_{c1}n) - \text{Sen}(w_{c2}n)}{\pi n}, & n \neq 0 \\ 1 - \frac{w_{c2} - w_{c1}}{\pi}, & n = 0 \end{cases}$$

Ecuación 14-10

$$-\frac{M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

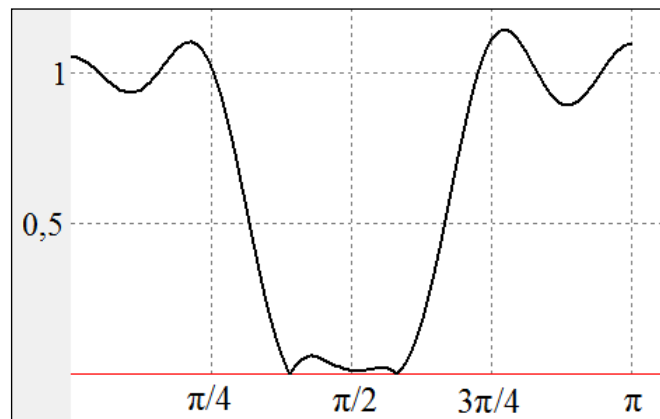


Figura 14-5

14.4.5 Filtro Multi Band

Los filtros multi banda son arreglos que integran diferentes filtros en uno solo, estos pueden dar ganancias arbitrarias en un rango de frecuencias. La siguiente ecuación integra las ganancias A del filtro, y las frecuencias de corte w:

$$h(n) = \begin{cases} \sum_{l=1}^L \left[(A_l - A_{l+1}) \frac{\sin(w_l n)}{\pi n} \right], & n \neq 0 \\ \sum_{l=1}^L \left[(A_l - A_{l+1}) \frac{w_l}{\pi} \right], & n = 0 \end{cases}$$

Ecuación 14-11

$$-\frac{M}{2} < n < \frac{M}{2}$$

La siguiente gráfica muestra la respuesta, de un filtro multi banda de orden 65, con cuatro bandas diferentes:

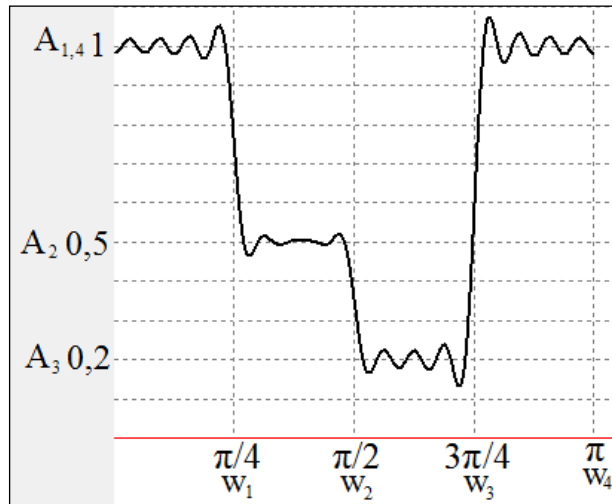


Figura 14-6

Para fines de diseño se debe tener presente que la ultima frecuencia calculada debe ser igual a π . Este tipo de filtros son ideales para el diseño de sistemas que requieran efectuar ecualización, sobre una señal.

14.4.6 Ventanas fijas

Las ventanas se aplican a las funciones de transferencia $h(n)$, el objetivo de las ventanas es mejorar y suavizar la respuesta espectral de los filtros FIR. Las ventanas de mayor uso son las siguientes:

- Rectangular
- Hamming
- Hanning
- Blackman

Los filtros que se demostraron anteriormente por defecto son filtros con ventana rectangular. Estos filtros tienen la menor transición en la frecuencia de corte, lo que los hace más cercanos a los filtros ideales, sin embargo esta propiedad produce en los filtros sobresaltos y oscilaciones en el espectro. Este efecto es conocido como: fenómeno Gibbs. Este efecto puede apreciarse en la siguiente gráfica:

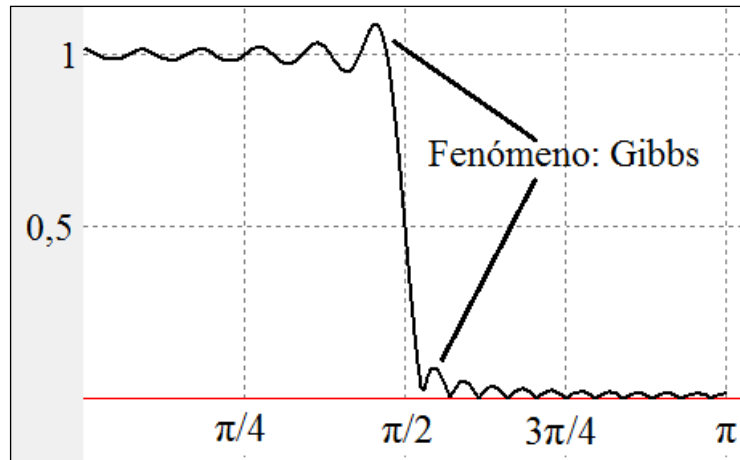


Figura 14-7

La ventana Rectangular ofrece una banda de transición igual a la siguiente relación: $dW = 1,8\pi/M$

14.4.7 Ventanas Rectangulares

Una ventana rectangular ofrece un patrón lineal, y constante. Se debe recordar que al realizar el cálculo de un filtro FIR, por defecto ya se encuentra con una ventana de este tipo.

La aplicación de las ventanas involucra la creación de una nueva función de factores $w(n)$, que posteriormente debe ser multiplicada término a término con la función de transferencia $h(n)$, para crear la función $h(n)$ definitiva.

Una ventana Rectangular se representa por medio de la siguiente función $w(n)$:

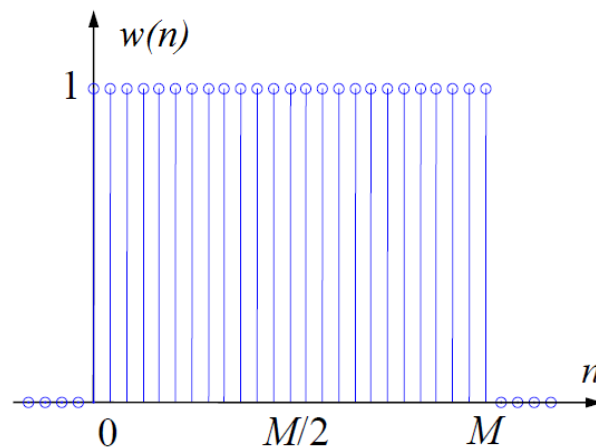


Figura 14-8

La respuesta espectral de la ventana Rectangular en escala logarítmica es la siguiente:

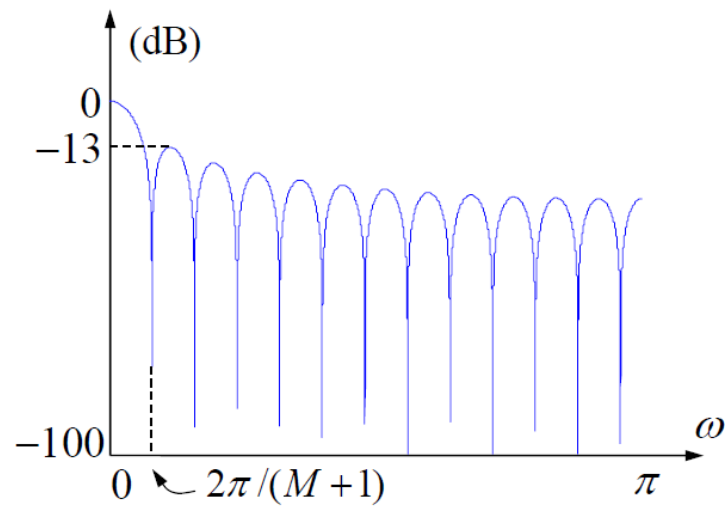


Figura 14-9

La ventana Rectangular ofrece una atenuación del fenómeno Gibbs de -13dB.

14.4.8 Ventanas Hamming

Para el caso de la ventana Hamming, los factores $w(n)$, se calculan de la siguiente forma:

$$w(n) = \frac{1 - \cos(2\pi n / M)}{2}, 0 \leq n \leq M \quad \text{Ecuación 14-12}$$

La misma respuesta espectral aplicando una ventana Hamming, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

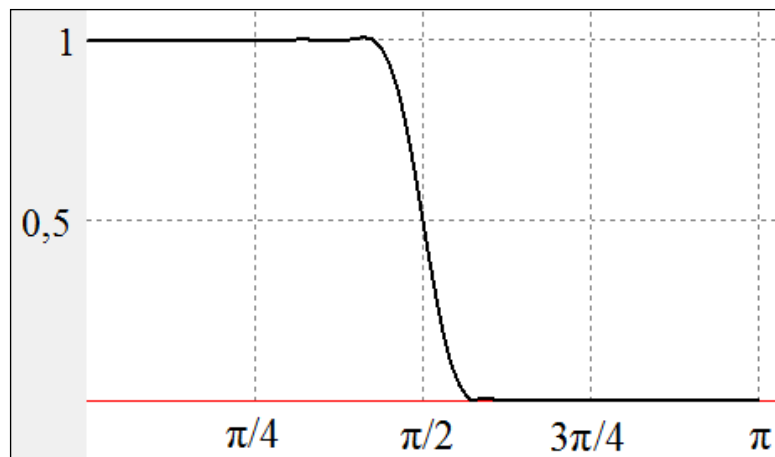


Figura 14-10

La ventana Hamming ofrece una banda de transición igual a la siguiente relación: $dW = 6,2\pi/M$

Una ventana Hamming se representa por medio de la siguiente función $w(n)$:

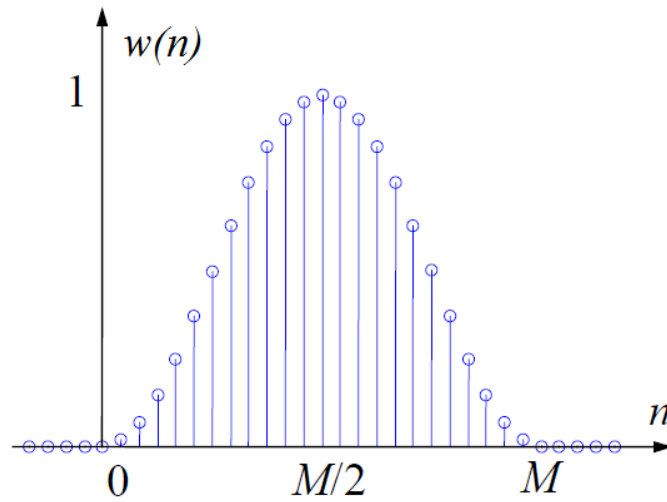


Figura 14-11

La respuesta espectral de la ventana Hamming en escala logarítmica es la siguiente:

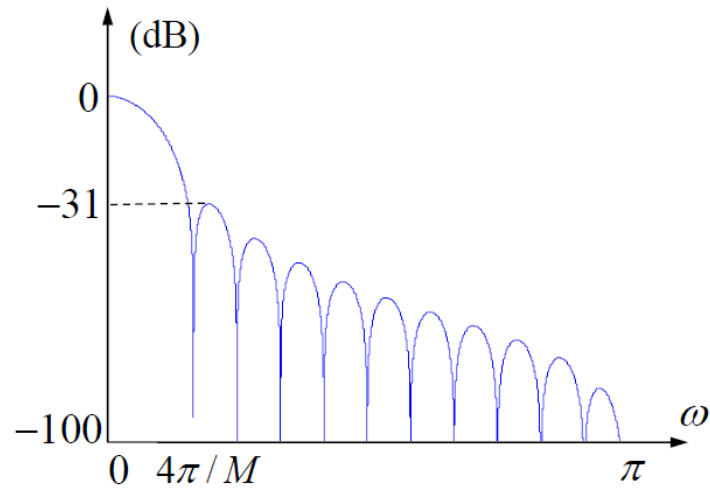


Figura 14-12

En la ventana Hamming se ofrece una atenuación del fenómeno Gibbs de -31dB.

14.4.9 Ventanas Hanning

Para el cálculo de la ventana Hanning, los factores $w(n)$, se deducen de la siguiente forma:

$$w(n) = \frac{27 - 23\cos(2\pi n/M)}{50}, 0 \leq n \leq M \quad \text{Ecuación 14-13}$$

La misma respuesta espectral aplicando una ventana Hanning, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

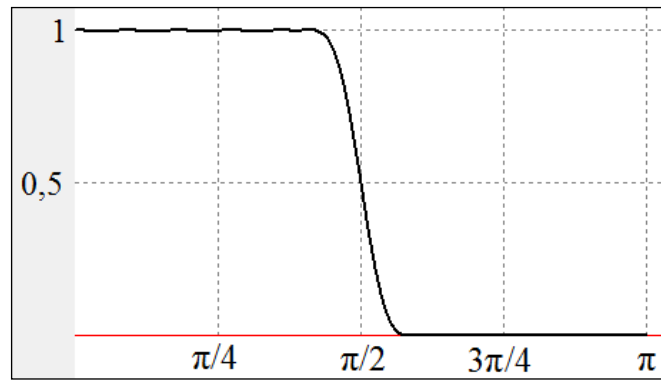


Figura 14-13

La ventana Hanning ofrece una banda de transición igual a la siguiente relación: $dW = 6,6\pi/M$

Una ventana Hanning se representa por medio de la siguiente función $w(n)$:

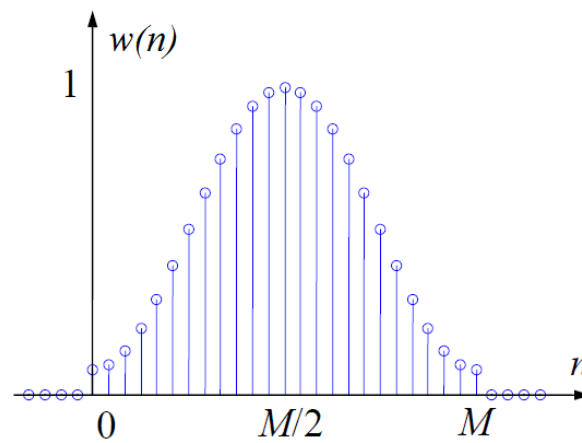


Figura 14-14

La respuesta espectral de la ventana Hanning en escala logarítmica es la siguiente:

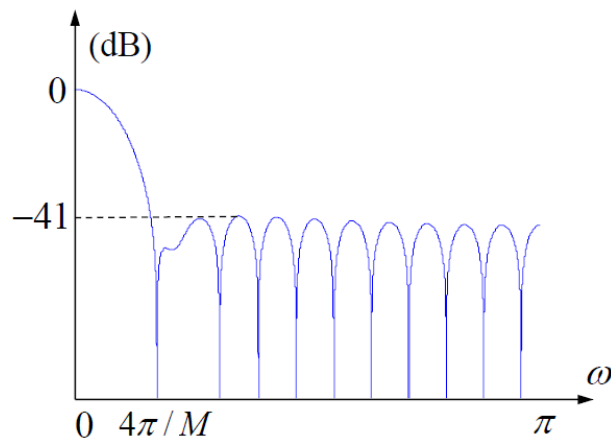


Figura 14-15

En la ventana Hanning se ofrece una atenuación del fenómeno Gibbs de -41dB.

14.4.10 Ventanas Blackman

Para el diseño de una ventana Blackman, los factores $w(n)$, se calculan de la siguiente forma:

$$w(n) = \frac{21 - 25\cos(2\pi n/M) + 4\cos(4\pi n/M)}{50}, 0 \leq n \leq M \quad \text{Ecuación 14-14}$$

La misma respuesta espectral aplicando una ventana Blackman, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

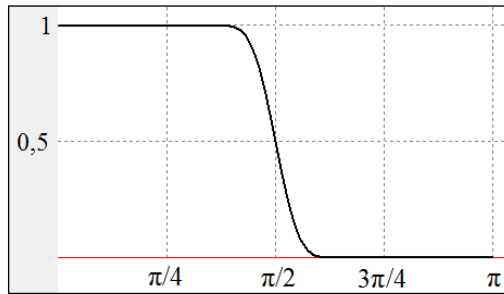


Figura 14-16

La ventana Blackman ofrece una banda de transición igual a la siguiente relación: $dW = 11\pi/M$

Una ventana Blackman se representa por medio de la siguiente función $w(n)$:

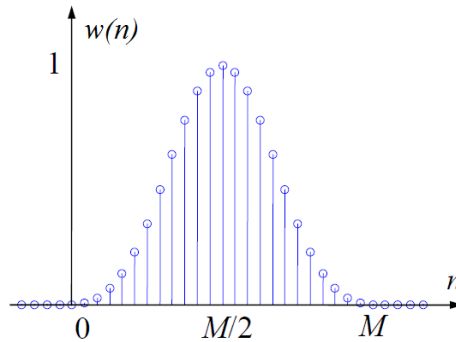


Figura 14-17

La respuesta espectral de la ventana Blackman en escala logarítmica es la siguiente:

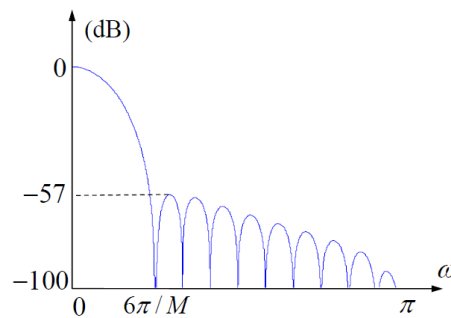



Figura 14-18

La ventana Blackman ofrece una atenuación del fenómeno Gibbs de -57dB.

14.5 Ejemplos de diseño para filtros FIR

En la siguiente sección se mostrarán ejemplos de diseño para filtros FIR, implementando un microcontrolador 18F452, con una fuente de 40MHz como reloj. Para fines prácticos reales la frecuencia de 40MHz, se logra utilizando un cristal de 10MHz, y activando la fuente de reloj HS-PLL en el PIC. Esta opción implementa internamente en el PIC, un PLL que multiplica la frecuencia externa por un factor de cuatro, y el resultado es usado como fuente de reloj para el procesador del microcontrolador. Para todos los ejemplos FIR que se mostrarán se deben simular con el mismo arreglo en ISIS, para este fin se implementa un circuito con los dispositivos 18F452, RES, OSCILLOSCOPE, Generador virtual Seno.

Para usar los generadores de señal virtual en ISIS, se debe picar en la barra de herramientas de la izquierda el icono de Generator Mode, este tiene la siguiente apariencia visual: , dentro de este se escoge la opción SINE, y se pega dentro del área de trabajo. Este generador tiene por defecto una frecuencia de 1Hz, y una amplitud de 1 voltio. La adquisición de señales se hace en el microcontrolador por medio del módulo AD, los niveles de tensión que las entradas análogas admiten, no pueden salirse de los confines de la polarización del microcontrolador. En otras palabras los niveles de las entradas análogas no pueden ser superiores a 5 voltios, ni voltajes negativos. Para evitar las circunstancias antes nombradas, se debe manipular la configuración del generador virtual de señal, para este propósito, se hace doble clic, sobre el generador y se editan los parámetros: Offset, y Amplitud. El parámetro offset se configura a 2,5 voltios, y el parámetro Amplitud a 2 voltios. Esta acción evita que las señales generadas se salgan de los parámetros de PIC. Para fines de simulación el parámetro Frequency, puede ser alterado al gusto del desarrollador.

Una vista de esta edición se puede apreciar en la siguiente figura:

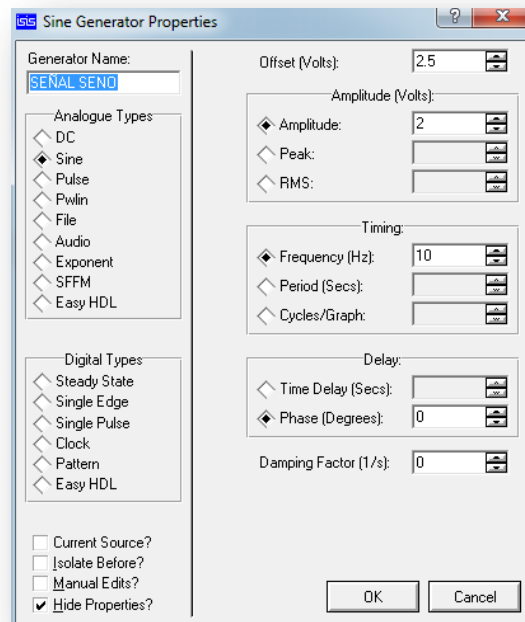
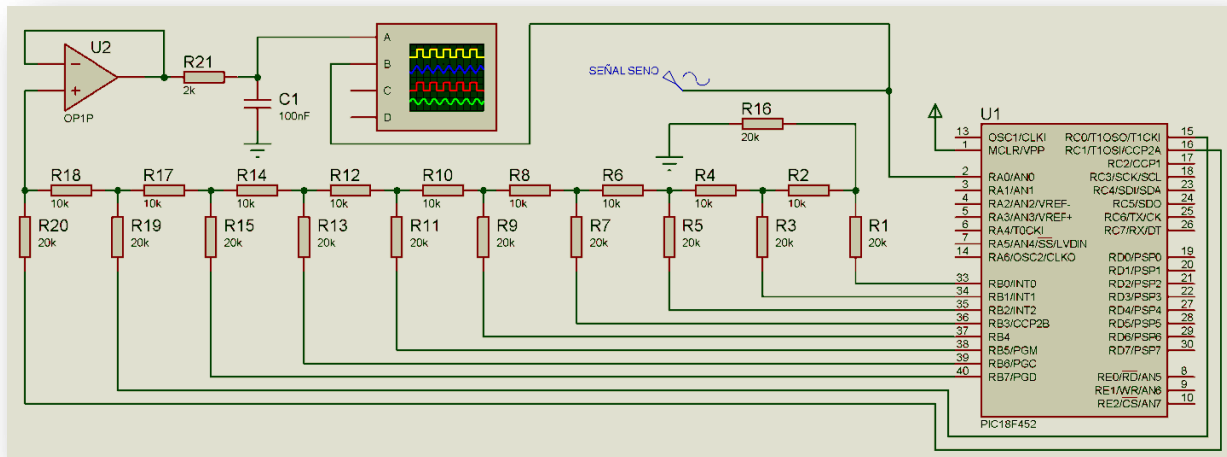


Figura 14-19

Indiferente al ejemplo que se simule en esta sección, se puede implementar el siguiente circuito electrónico en ISIS:



Circuito 14-1

Para la reconstrucción de la señal procesada, se configuran 10 bits de salida, para hacer un convertidor DA, por medio de un arreglo R-2R.

Para todos los ejemplos usados en este apartado, se usará la interrupción por TIMER 0, para crear el periodo de muestreo, y por defecto la frecuencia de muestreo.

El siguiente código fuente muestra un ejemplo del muestreo de la señal por medio del TIMER 0:

```
//Declaración de variables.
float x0, y0;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //
        //Espacio para procesar la señal.
        //
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(x0+512);
        PORTC = (YY>8)&3;
        PORTB = YY&255;
    }
}
```

```

    INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.5.1 Ejemplo de diseño para filtro pasa bajas

$F_s = 1291,32\text{KHz}$.

$F_c = 150\text{Hz}$.

Ventana Rectangular.

Se determina la frecuencia de corte digital:

$$W_c = 2 \pi F_c / F_s = 2 \pi 150 / 1291,32 = 0,72985.$$

Usando la formula (14.7), se designan los coeficientes de la función $h(n)$:

```

h(-8)=-0.0171035387965417
h(-7)=-0.0419431579233366
h(-6)=-0.0501329294124475
h(-5)=-0.0309497847516785
h(-4)=0.0175345019583181
h(-3)=0.0864308262744764
h(-2)=0.158173992108178
h(-1)=0.212237065988464
h(0)=0.232320416318186
h(1)=0.212237065988464
h(2)=0.158173992108178
h(3)=0.0864308262744764
h(4)=0.0175345019583181
h(5)=-0.0309497847516785
h(6)=-0.0501329294124475
h(7)=-0.0419431579233366
h(8)=-0.0171035387965417

```

Se debe tener presente que para fines de programación los valores de n no pueden ser negativos, por esta razón se debe iniciar el vector en 0, y para usarlo en el código en lenguaje C, se implementa de la siguiente manera:

```
const float h[]=  
{  
  -0.0171035387965417, //h(0)  
  -0.0419431579233366, //h(1)  
  -0.0501329294124475, //h(2)  
  -0.0309497847516785, //h(3)  
  0.0175345019583181, //h(4)  
  0.0864308262744764, //h(5)  
  0.158173992108178, //h(6)  
  0.212237065988464, //h(7)  
  0.232320416318186, //h(8)  
  0.212237065988464, //h(9)  
  0.158173992108178, //h(10)  
  0.0864308262744764, //h(11)  
  0.0175345019583181, //h(12)  
  -0.0309497847516785, //h(13)  
  -0.0501329294124475, //h(14)  
  -0.0419431579233366, //h(15)  
  -0.0171035387965417 //h(16)  
};
```

El programa definitivo con la función de transferencia $h(n)$, será de la siguiente forma:

```
#define M 17  
//Función de transferencia h[n]  
const float h[]=  
{  
  -0.0171035387965417, //h(0)  
  -0.0419431579233366, //h(1)  
  -0.0501329294124475, //h(2)  
  -0.0309497847516785, //h(3)  
  0.0175345019583181, //h(4)  
  0.0864308262744764, //h(5)  
  0.158173992108178, //h(6)  
  0.212237065988464, //h(7)  
  0.232320416318186, //h(8)  
  0.212237065988464, //h(9)  
  0.158173992108178, //h(10)  
  0.0864308262744764, //h(11)  
  0.0175345019583181, //h(12)  
  -0.0309497847516785, //h(13)  
  -0.0501329294124475, //h(14)  
  -0.0419431579233366, //h(15)  
  -0.0171035387965417 //h(16)  
};
```



```

//Declaración de variables.
float x0, y0;
float x[M];
unsigned int YY;
unsigned short i;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        PORTC.F7=1;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Corrimiento continuo de la señal x[n]
        for( i=M-1; i!=0; i-- )x[i]=x[i-1];
        //Adquisición de una muestra de 10 bits en, x[0].
        x[0] = (float)(ADC_Read(0)-512.0);
        //Convolución continua.
        y0 = 0.0; for( i=0; i<M; i++ ) y0 += h[i]*x[i];
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        PORTC.F7=0;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

Por limitaciones de velocidad en este microcontrolador, y teniendo presente la frecuencia de muestreo de 1291,32Hz no es posible usar un orden del filtro superior a 17. Sin embargo es posible usar órdenes menores como; 15, 13, 11, 9, 7, 5, o 3. Para fines prácticos y didácticos en los próximos ejemplos se implementarán filtros de orden 17 igual a este.

14.5.2 Ejemplo de diseño para filtro pasa altas

$F_s = 1291,32\text{KHz}$.

$F_c = 200\text{Hz}$.

Ventana Hamming.

Se determina la frecuencia de corte digital:

$$W_c = 2 \pi F_c / F_s = 2 \pi 200 / 1291,32 = 0,97314.$$

Usando la formula (14.8), se definen los coeficientes de la función $h(n)$ multiplicados por la ventana Hamming $w(n)$ de la ecuación (14.12), dando como resultado la siguiente función $h(n)$:

$h(-8)=0$
 $h(-7)=-0.000774581987451374$
 $h(-6)=0.00297591407324525$
 $h(-5)=0.0174357425540551$
 $h(-4)=0.0246447931670207$
 $h(-3)=-0.0148886974624375$
 $h(-2)=-0.118648252092459$
 $h(-1)=-0.243426834227261$
 $h(0)=0.684363125797732$
 $h(1)=-0.260893089082499$
 $h(2)=-0.136977589378571$
 $h(3)=-0.0187342667800403$
 $h(4)=0.0345797805794857$
 $h(5)=0.028555061237806$
 $h(6)=0.00631989035694063$
 $h(7)=-0.00299371636029323$
 $h(8)=-0.00134023946307802$

La implementación del código fuente es igual al filtro pasa bajas, sustituyendo el arreglo $h[]$, por el siguiente:

```
const float h[]=  
{  
    0, //h(0)  
    -0.000774581987451374, //h(1)  
    0.00297591407324525, //h(2)  
    0.0174357425540551, //h(3)  
    0.0246447931670207, //h(4)  
    -0.0148886974624375, //h(5)  
    -0.118648252092459, //h(6)  
    -0.243426834227261, //h(7)  
    0.684363125797732, //h(8)  
    -0.260893089082499, //h(9)  
    -0.136977589378571, //h(10)  
    -0.0187342667800403, //h(11)  
    0.0345797805794857, //h(12)  
    0.028555061237806, //h(13)  
}
```

```

0.00631989035694063, //h(14)
-0.00299371636029323, //h(15)
-0.00134023946307802 //h(16)
};

```

14.5.3 Ejemplo de diseño para filtro pasa banda

$F_s = 1291,32\text{KHz}$.
 $F_{c1} = 150\text{Hz}$.
 $F_{c2} = 400\text{Hz}$.
 Ventana Hanning.

Se determinan las frecuencias de corte digital:
 $W_{c1} = 2 \pi F_c / F_s = 2 \pi 150 / 1291,32 = 0,72985$.
 $W_{c2} = 2 \pi F_c / F_s = 2 \pi 400 / 1291,32 = 1,94628$.

Usando la formula (14.9), se definen los coeficientes de la función $h(n)$ multiplicados por la ventana Hanning $w(n)$ de la ecuación (14.13), dando como resultado la siguiente función $h(n)$:

```

h(-8)=0.0018052104538582
h(-7)=0.00905810215732946
h(-6)=0.00179096961917994
h(-5)=0.00393014193876244
h(-4)=0.0307760790492056
h(-3)=-0.0879236273273945
h(-2)=-0.218010454414228
h(-1)=0.078115497545518
h(0)=0.384167993159943
h(1)=0.0832388331308223
h(2)=-0.248392736747743
h(3)=-0.107904878578323
h(4)=0.0411879382357919
h(5)=0.00583790841302624
h(6)=0.00299868100666665
h(7)=0.0163162389068139
h(8)=0.00250614601893693

```

La función $h[]$, para implementar en el código fuente en lenguaje C es:

```

const float h[]=
{
0.0018052104538582, //h(0)
0.00905810215732946, //h(1)
0.00179096961917994, //h(2)
0.00393014193876244, //h(3)
0.0307760790492056, //h(4)
-0.0879236273273945, //h(5)
-0.218010454414228, //h(6)
0.078115497545518, //h(7)

```

```

0.384167993159943, //h(8)
0.0832388331308223, //h(9)
-0.248392736747743, //h(10)
-0.107904878578323, //h(11)
0.0411879382357919, //h(12)
0.00583790841302624, //h(13)
0.00299868100666665, //h(14)
0.0163162389068139, //h(15)
0.00250614601893693 //h(16)
};

```

14.5.4 Ejemplo de diseño para filtro rechaza banda

$F_s = 1291,32\text{KHz}$.

$F_{c1} = 150\text{Hz}$.

$F_{c2} = 400\text{Hz}$.

Ventana Blackman.

Se determinan las frecuencias de corte digital:

$W_{c1} = 2 \pi F_{c1} / F_s = 2 \pi 150 / 1291,32 = 0,72985$.

$W_{c2} = 2 \pi F_{c2} / F_s = 2 \pi 400 / 1291,32 = 1,94628$.

Usando la formula (14.10), se definen los coeficientes de la función $h(n)$ multiplicados por la ventana Blackman $w(n)$ de la ecuación (14.14), dando como resultado la siguiente función $h(n)$:

```

h(-8)=3.13154095338657E-19
h(-7)=-0.00105084724932717
h(-6)=-0.000518135020348656
h(-5)=-0.00174730211401576
h(-4)=-0.0182611591144528
h(-3)=0.0645431455464317
h(-2)=0.186587834540544
h(-1)=-0.0738928265716166
h(0)=0.604271792109402
h(1)=-0.0827284691705043
h(2)=0.234965429330525
h(3)=0.0923521657912548
h(4)=-0.0302353209611255
h(5)=-0.00346395569934133
h(6)=-0.00133318382487159
h(7)=-0.00472035632958784
h(8)=-0.000290742652784183

```

La función $h[]$, para implementar en el código fuente en lenguaje C es:

```

const float h[]=
{
    3.13154095338657E-19, //h(0)
    -0.00105084724932717, //h(1)

```

```

-0.000518135020348656, //h(2)
-0.00174730211401576, //h(3)
-0.0182611591144528, //h(4)
0.0645431455464317, //h(5)
0.186587834540544, //h(6)
-0.0738928265716166, //h(7)
0.604271792109402, //h(8)
-0.0827284691705043, //h(9)
0.234965429330525, //h(10)
0.0923521657912548, //h(11)
-0.0302353209611255, //h(12)
-0.00346395569934133, //h(13)
-0.00133318382487159, //h(14)
-0.00472035632958784, //h(15)
-0.000290742652784183 //h(16)
};

```

14.5.5 Ejemplo de diseño para filtro multi banda

$F_s = 1291,32\text{KHz}$.

$F_{c1} = 161\text{Hz}$.

$F_{c2} = 322\text{Hz}$.

$F_{c3} = 484\text{Hz}$.

$F_{c4} = 645.66\text{Hz}$.

$A_1 = 1$.

$A_2 = 0,5$.

$A_3 = 0,2$.

$A_4 = 1$.

Ventana Rectangular.

Se determinan las frecuencias de corte digital:

$W_{c1} = 2 \pi F_{c1} / F_s = 2 \pi 161 / 1291,32 = 0,78337$.

$W_{c2} = 2 \pi F_{c2} / F_s = 2 \pi 322 / 1291,32 = 1,56675$.

$W_{c3} = 2 \pi F_{c3} / F_s = 2 \pi 484 / 1291,32 = 2,355$.

$W_{c4} = 2 \pi F_{c4} / F_s = 2 \pi 645,66 / 1291,32 = \pi$.

Usando la formula (14.11), se definen los coeficientes de la función $h(n)$:

$h(-8) = -0.000403386658426763$

$h(-7) = -0.00443133542115946$

$h(-6) = -0.0685784954060509$

$h(-5) = 0.0330418473673128$

$h(-4) = -0.000367826230314909$

$h(-3) = -0.0538949660023408$

$h(-2) = 0.207286062892996$

$h(-1) = 0.0275264614524777$

$h(0) = 0.674596536876994$

$h(1) = 0.0275264614524777$

$h(2) = 0.207286062892996$

```
h(3)=-0.0538949660023408
h(4)=-0.000367826230314909
h(5)=0.0330418473673128
h(6)=-0.0685784954060509
h(7)=-0.00443133542115946
h(8)=-0.000403386658426763
```

La función `h[]`, para implementar en el código fuente en lenguaje C es la siguiente:

```
const float h[]=
{
-0.000403386658426763, //h(0)
-0.00443133542115946, //h(1)
-0.0685784954060509, //h(2)
0.0330418473673128, //h(3)
-0.000367826230314909, //h(4)
-0.0538949660023408, //h(5)
0.207286062892996, //h(6)
0.0275264614524777, //h(7)
0.674596536876994, //h(8)
0.0275264614524777, //h(9)
0.207286062892996, //h(10)
-0.0538949660023408, //h(11)
-0.000367826230314909, //h(12)
0.0330418473673128, //h(13)
-0.0685784954060509, //h(14)
-0.00443133542115946, //h(15)
-0.000403386658426763, //h(16)
};
```

14.6 Filtros IIR

Los filtros IIR, son de implementación computacional más simple, pero su diseño y sus cálculos son de mayor complejidad. Para diseñar un filtro IIR, existen diferentes técnicas, sin embargo este capítulo se centrará en el diseño por medio de la transformación bilineal. La transformación bilineal es una relación que existe entre la variable compleja s , y z . Esto quiere decir que para realizar un filtro IIR, se requiere una función de transferencia en términos de la variable s . En síntesis para iniciar el diseño de un filtro IIR, se requiere como punto de partida, la función de transferencia de un filtro análogo. Para fines de estudio en este capítulo se mostrarán a continuación las funciones de transferencia de los filtros básicos de segundo orden:

Filtro Pasa Bajas:

$$H(s) = \frac{\Omega_c^2}{s^2 + 2\zeta\Omega_c s + \Omega_c^2} \quad \text{Ecuación 14-15}$$

Filtro Pasa Altas:

$$H(s) = \frac{s^2}{s^2 + 2\zeta\Omega_c s + \Omega_c^2} \quad \text{Ecuación 14-16}$$

Filtro Pasa Banda:

$$H(s) = \frac{s\Omega_B}{s^2 + \Omega_B s + \Omega_0^2} \quad \text{Ecuación 14-17}$$

Filtro Rechaza Banda:

$$H(s) = \frac{s^2 + \Omega_0^2}{s^2 + \Omega_B s + \Omega_0^2} \quad \text{Ecuación 14-18}$$

Las frecuencias Ω están definidas en radianes/segundo. Las frecuencias Ω_c representan la frecuencia de corte en los filtros pasa bajas, y altas. Las frecuencias Ω_0 representan la frecuencia de resonancia en los filtros pasa banda y rechaza banda. Las frecuencias Ω_B representan el ancho de banda de los filtros pasa banda y rechaza banda. El coeficiente ζ , representa el amortiguamiento del filtro, el valor que este coeficiente debe tomar es mayor que 0, cuando el coeficiente de amortiguamiento vale 0, el filtro se convierte en resonante, y tendría un comportamiento similar a los filtros pasa banda o rechaza banda. Por otra parte el coeficiente de amortiguamiento no podrá ser negativo, esto generaría una función de transferencia inestable. Cabe denotar que la implementación de filtros IIR, produce en la respuesta de la señal distorsiones mayores a las que se generan con los filtros FIR.

14.6.1 Transformación bilineal

La transformación bilineal es una relación matemática entre las variables complejas s , y z . Esta relación se rige por las siguientes ecuaciones:

$$s = \frac{2(1 - z^{-1})}{T_s(1 + z^{-1})} = \frac{2F_s(1 - z^{-1})}{(1 + z^{-1})} \quad \text{Ecuación 14-19}$$

En la ecuación (14.19), se puede observar la relación entre las dos variables, donde T_s , y F_s , son respectivamente el periodo de muestreo y frecuencia de muestreo implementado. De la misma forma existe una relación entre las frecuencias del filtro análogo y el filtro digital, esta relación se puede apreciar en la siguiente ecuación:

$$\Omega = \frac{2 \tan(\frac{w_d}{2})}{T_s} = 2F_s \tan(\frac{w_d}{2}) \quad \text{Ecuación 14-20}$$

Donde la frecuencia digital es: w_d , la frecuencia análoga es: Ω , y T_s , F_s , son respectivamente el periodo de muestreo y la frecuencia de muestreo.

De la misma forma que en los filtros FIR, la relación entre la frecuencia digital y la frecuencia en hercios es la siguiente:

$$w_d = \frac{2\pi F_x}{F_s} \Rightarrow \Omega_x = \frac{2 \tan(\frac{\pi F_x}{F_s})}{T_s} = 2F_s \tan(\frac{\pi F_x}{F_s}) \quad \text{Ecuación 14-21}$$

Para contextualizar este proceso, se mostrará a continuación la transformación bilineal de los cuatro filtros básicos en segundo orden.

14.6.2 Filtro Pasa Bajas IIR

Como primera medida, se realiza el reemplazo de variables en la ecuación (14.15), para obtener la función $h(z)$:

$$H(z) = \frac{\Omega_c^2}{\left[\frac{2F_s(1 - z^{-1})}{(1 + z^{-1})} \right]^2 + 2\zeta\Omega_c \left[\frac{2F_s(1 - z^{-1})}{(1 + z^{-1})} \right] + \Omega_c^2}$$

$$H(z) = \frac{\Omega_c^2(1 + z^{-1})^2}{4F_s^2(1 - z^{-1})^2 + 4\zeta\Omega_c F_s(1 - z^{-1})(1 + z^{-1}) + \Omega_c^2(1 + z^{-1})^2}$$

$$H(z) = \frac{\Omega_c^2(1 + 2z^{-1} + z^{-2})}{4F_s^2(1 - 2z^{-1} + z^{-2}) + 4\zeta\Omega_c F_s(1 - z^{-2}) + \Omega_c^2(1 + 2z^{-1} + z^{-2})}$$

$$H(z) = \frac{\Omega_c^2 + 2\Omega_c^2 z^{-1} + \Omega_c^2 z^{-2}}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2] + [2\Omega_c^2 - 8F_s^2]z^{-1} + [4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]z^{-2}}$$

En este punto se tiene una función de transferencia con polinomios ordenados en los polos y los ceros, de la forma $H(z) = A(z)/B(z)$. Para los polinomios A, B, se tiene en este caso expresiones de segundo orden con la forma: $A(z) = a_1z^0 + a_2z^{-1} + a_3z^{-2}$, $B(z) = b_1z^0 + b_2z^{-1} + b_3z^{-2}$.

Después de tener la función de transferencia de esta forma se debe implementar una ecuación en diferencias para la ejecución en la programación, para deducir la ecuación en diferencias es necesario forzar el coeficiente del denominador b_1 a valer 1. Para lograr este objetivo todos los coeficientes a y b se dividen en b_1 . Está condición aplicada en la última ecuación del filtro pasa bajas, da como resultado lo siguiente:

$$H(z) = \frac{\frac{\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} + \frac{2\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]}z^{-1} + \frac{\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]}z^{-2}}{1 + \frac{[2\Omega_c^2 - 8F_s^2]}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]}z^{-1} + \frac{[4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]}z^{-2}}$$

La misma ecuación en términos de los coeficientes, es de la siguiente forma:

$$H(z) = \frac{a_1z^0 + a_2z^{-1} + a_3z^{-2}}{b_1z^0 + b_2z^{-1} + b_3z^{-2}},$$

$$a_1 = \frac{\Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$a_2 = 2a_1$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

Para realizar la ecuación en diferencias se hace la siguiente transformación de z a n.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A(z)}{B(z)} \Rightarrow Y(z)B(z) = X(z)A(z) \rightarrow y(n) * b(n) = x(n) * a(n)$$

Realizando la transformación inversa de z, se tiene una doble convolución, que genera una ecuación de la siguiente forma:

$$b_1y(n) + b_2y(n-1) + b_3y(n-2) = a_1x(n) + a_2x(n-1) + a_3x(n-2)$$

En conclusión la salida $y(n)$ queda de la siguiente manera:

$$y(n) = a_1x(n) + a_2x(n-1) + a_3x(n-2) - b_2y(n-1) - b_3y(n-2)$$

14.6.3 Ejemplo de diseño para filtro pasa bajas

$F_s = 1291.32\text{Hz}$.

$F_c = 100\text{Hz}$.

$$\zeta = \frac{\sqrt{2}}{2}.$$

Para el caso particular de los filtros pasa bajas, y pasa altas de segundo orden el valor de ζ debe ser $\sqrt{2}/2$ para que la frecuencia de corte no sufra desplazamientos con respecto a las condiciones establecidas para los filtros en general.

Para deducir la frecuencia de corte análoga en función de las frecuencias reales se usa la ecuación (14.21), dando como resultado la siguiente frecuencia de corte:

$$\Omega_c = 2F_s \tan\left(\frac{\pi F_c}{F_s}\right) = 2(1291,32) \tan\left(\frac{100\pi}{1291,32}\right) = 641.0154$$

El paso siguiente es remplazar las constantes en las relaciones que determinan los coeficientes a, y b. Este paso se puede apreciar a continuación:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{\Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{(641,154)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = 0,0436286852600961$$

$$a_2 = 2a_1 = 0,0872573705201923$$

$$a_3 = a_1 = 0,0436286852600961$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{2(641,0154)^2 - 8(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = -1,32843480532316$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2 - 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = 0,502949546363549$$

La implementación de un filtro IIR, de orden N, requiere un búfer de datos igual a: 2(N+1). En el caso de un filtro de segundo orden se requieren 3 campos para guardar los últimos valores de la entrada x, y 3 campos más para los 3 últimos valores de la salida y. Para ilustrar la implementación de este proceso se usará la misma arquitectura de software y hardware implementada en los filtros FIR. A continuación se puede observar el siguiente ejemplo en lenguaje C:

//Declaración de variables.

float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;

```

unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //Implementación de la ecuación en diferencias.
        y0 = (x0+x2)*0.0436286852600961 + x1*0.0872573705201923 + y1*1.32843480532316
        -y2*0.502949546363549;
        //Corrimiento de los valores x(n), y y(n).
        y2 = y1;
        y1 = y0;
        x2 = x1;
        x1 = x0;
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.6.4 Filtro Pasa Altas IIR

De la misma forma que se hizo en el filtro pasa bajas, se realiza el reemplazo de variable en la ecuación (14.16), para obtener la función $h(z)$:

$$H(z) = \frac{\left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right]^2}{\left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right]^2 + 2\zeta\Omega_c \left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_c^2}$$

$$H(z) = \frac{4F_s^2(1-z^{-1})^2}{4F_s^2(1-z^{-1})^2 + 4\zeta\Omega_c F_s(1+z^{-1})(1-z^{-1}) + \Omega_c^2(1+z^{-1})^2}$$

$$H(z) = \frac{4F_s^2(1-2z^{-1}+z^{-2})}{4F_s^2(1-2z^{-1}+z^{-2}) + 4\zeta\Omega_c F_s(1-z^{-2}) + \Omega_c^2(1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{4F_s^2 - 8F_s^2 z^{-1} + 4F_s^2 z^{-2}}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2] + [2\Omega_c^2 - 8F_s^2]z^{-1} + [4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]z^{-2}}$$

Realizando la normalización de los coeficientes se obtiene lo siguiente:

$$H(z) = \frac{\frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} - \frac{8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} z^{-1} + \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} z^{-2}}{1 + \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} z^{-1} + \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} z^{-2}}$$

De esta forma los coeficientes a, y b quedan definidos así:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$a_2 = -2a_1$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

14.6.5 Ejemplo de diseño para filtro pasa altas

Fs = 1291.32Hz.

Fc = 200Hz.

$$\zeta = \frac{\sqrt{2}}{2}.$$

Se determina la frecuencia de corte análoga en términos de la frecuencia digital:

$$\Omega_c = 2F_s \tan\left(\frac{\pi F_c}{F_s}\right) = 2(1291,32) \tan\left(\frac{200\pi}{1291,32}\right) = 1366,19404$$

Seguidamente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = 0,493018837784645$$

$$a_2 = -2a_1 = -0,986037675569289$$

$$a_3 = a_1 = 0,493018837784645$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{2(1366,19404)^2 - 8(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = -0,709951943923452$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2 - 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = 0,262123407215126$$

La programación de este filtro pasa altas, bajo las mismas condiciones del filtro pasa bajas es el siguiente:

//Declaración de variables.

float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;

unsigned int YY;

//Declaración de la función de interrupciones.

void interrupt (**void**)

{

if(INTCON.F2)

 {

 TMR0L=135;

//Timer0 con periodo de 774,4u segundo.

// Fs = 1291,32 Hz.

//Adquisición de una muestra de 10 bits en, x[0].

 x0 = (**float**)(ADC_Read(0)-512.0);

//Implementación de la ecuación en diferencias.

 y0 = (x0+x2)*0.493018837784645 - x1*0.986037675569289 +

 y1*0.709951943923452 - y2*0.262123407215126;

//Corrimiento de los valores x(n), y y(n).

```

    y2 = y1;
    y1 = y0;
    x2 = x1;
    x1 = x0;
    //Reconstrucción de la señal: y en 10 bits.
    YY = (unsigned int)(y0+512.0);
    PORTC = (YY>>8)&3;
    PORTB = YY&255;
    INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.6.6 Filtro Pasa Banda IIR

Para la función de transferencia $h(z)$ en el filtro pasa banda se realiza el cambio de variable bilineal, tomando como base la ecuación (14.17):

$$H(z) = \frac{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right] \Omega_B}{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right]^2 + \Omega_B \left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_0^2}$$

$$H(z) = \frac{2F_s \Omega_B (1-z^{-1})(1+z^{-1})}{4F_s^2 (1-z^{-1})^2 + 2F_s \Omega_B (1-z^{-1})(1+z^{-1}) + \Omega_0^2 (1+z^{-1})^2}$$

$$H(z) = \frac{2F_s \Omega_B (1-z^{-2})}{4F_s^2 (1-2z^{-1}+z^{-2}) + 2F_s \Omega_B (1-z^{-2}) + \Omega_0^2 (1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{2F_s \Omega_B - 2F_s \Omega_B z^{-2}}{[4F_s^2 + 2F_s \Omega_B + \Omega_0^2] + [2\Omega_0^2 - 2F_s \Omega_B] z^{-1} + [4F_s^2 - 2F_s \Omega_B + \Omega_0^2] z^{-2}}$$

Normalizando la ecuación se obtiene la siguiente relación:

$$H(z) = \frac{\frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} - \frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} z^{-2}}{1 + \frac{2\Omega_0^2 - 2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} z^{-1} + \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} z^{-2}}$$

De la anterior ecuación, se pueden establecer las siguientes constantes:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_2 = 0$$

$$a_3 = -a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

14.6.7 Ejemplo de diseño para filtro pasa banda

Fs = 1291.32Hz.

Fo = 300Hz.

Fb = 10Hz.

A continuación se determinan las frecuencias análogas para el diseño del filtro pasa banda:

$$\Omega_0 = 2F_s \tan\left(\frac{\pi F_0}{F_s}\right) = 2(1291,32) \tan\left(\frac{300\pi}{1291,32}\right) = 2310,58$$

$$\Omega_B = 2F_s \tan\left(\frac{\pi F_b}{F_s}\right) = 2(1291,32) \tan\left(\frac{10\pi}{1291,32}\right) = 62,8442$$

Posteriormente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{2F_s \Omega_B}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{2(1291,32)(62,8442)}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = 0.0133351841372129$$

$$a_2 = 0$$

$$a_3 = -a_1 = -0.0133351841372129$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 2F_s \Omega_B}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{2(2310,58)^2 - 2(1291,32)(62,8442)}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = -0.218755004098491$$

$$b_3 = \frac{4F_s^2 - 2F_s \Omega_B + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 - 2(1291,32)(62,8442) + (2310,58)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = 0.973329631725574$$

14.6.8 Filtro Rechaza Banda IIR

Para la función de transferencia $h(z)$ en el filtro pasa banda se realiza el cambio de variable bilineal, tomando como base la ecuación (14.17):

$$H(z) = \frac{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right]^2 + \Omega_0^2}{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right]^2 + \Omega_B \left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_0^2}$$

$$H(z) = \frac{4F_s^2(1-z^{-1})^2 + \Omega_0^2(1+z^{-1})^2}{4F_s^2(1-z^{-1})^2 + 2F_s \Omega_B(1-z^{-1})(1+z^{-1}) + \Omega_0^2(1+z^{-1})^2}$$

$$H(z) = \frac{4F_s^2(1-2z^{-1}+z^{-2}) + \Omega_0^2(1+2z^{-1}+z^{-2})}{4F_s^2(1-2z^{-1}+z^{-2}) + 2F_s \Omega_B(1-z^{-2}) + \Omega_0^2(1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{[4F_s^2 + \Omega_0^2] + [2\Omega_0^2 - 8F_s^2]z^{-1} + [4F_s^2 + \Omega_0^2]z^{-2}}{[4F_s^2 + 2F_s \Omega_B + \Omega_0^2] + [2\Omega_0^2 - 8F_s^2]z^{-1} + [4F_s^2 - 2F_s \Omega_B + \Omega_0^2]z^{-2}}$$

Normalizando la ecuación se obtiene la siguiente relación:

$$H(z) = \frac{\frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} + \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} z^{-1} + \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} z^{-2}}{1 + \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} z^{-1} + \frac{4F_s^2 - 2F_s \Omega_B + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} z^{-2}}$$

De la anterior ecuación, se pueden establecer las siguientes constantes:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

14.6.9 Ejemplo de diseño filtro rechaza banda

$F_s = 1291.32\text{Hz}$.

$F_o = 100\text{Hz}$.

$F_b = 10\text{Hz}$.

A continuación se determinan las frecuencias análogas para el diseño del filtro rechaza banda:

$$\Omega_0 = 2F_s \tan\left(\frac{\pi F_o}{F_s}\right) = 2(1291,32) \tan\left(\frac{100\pi}{1291,32}\right) = 641,015$$

$$\Omega_B = 2F_s \tan\left(\frac{\pi F_b}{F_s}\right) = 2(1291,32) \tan\left(\frac{10\pi}{1291,32}\right) = 62,8442$$

Posteriormente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 + (641,015)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = 0,977592319507735$$

$$a_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{2(641,015)^2 - 8(1291,32)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = -1.72826896958352$$

$$a_3 = a_1 = 0,977592319507735$$

$$b_1 = 1$$

$$b_2 = a_2 = -1.72826896958352$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 - 2(1291,32)(62,8442) + (641,015)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = 0.95518463901547$$

El código fuente en lenguaje C, correspondiente a este diseño es el siguiente:

```
//Declaración de variables.
float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //Implementación de la ecuación en diferencias.
        y0 = (x0+x2)*0.977592319507735 + (y1-x1)*1.72826896958352 - y2*0.95518463901547;
        //Corrimiento de los valores x(n), y y(n).
        y2 = y1;
        y1 = y0;
        x2 = x1;
        x1 = x0;
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
```

```

PORTB = 0;
TRISC = 0;
PORTC = 0;
//Se configura el TIMER 0, su interrupción.
INTCON = 0b10100000;
T0CON = 0b11000101;
while(1)//Bucle infinito.
{
}
}

```

14.7 Osciladores digitales

Generar una señal seno de forma sintética, implica usar una matriz de rotación que incluye dos valores de salida de la señal y dos valores pasados, de la misma. La forma general de esta matriz se puede apreciar a continuación:

$$\begin{bmatrix} y(n) \\ x(n) \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y(n-1) \\ x(n-1) \end{bmatrix}$$

Los cálculos de la matriz de rotación, permiten crear osciladores de 2 salidas que pueden tener amplitud equitativa o no y salida en cuadratura o no, así como el número de multiplicaciones requeridas por iteración. Estas condiciones se resumen en la siguiente tabla:

<i>Propiedades de los osciladores recursivos</i>					
<i>Oscilador</i>	<i>Multiplicaciones por iteración</i>	<i>Amplitud equitativa</i>	<i>Salida en cuadratura</i>	<i>Factores</i>	<i>Matriz de rotación</i>
<i>Doble cuadrado</i>	2	Si	No	$k=2\cos(\theta)$	$\begin{bmatrix} k & -1 \\ 1 & 0 \end{bmatrix}$
<i>De acople en cuadratura estándar</i>	4	Si	Si	$k=\sin(\theta)$	$\begin{bmatrix} \sqrt{1-k^2} & k \\ -k & \sqrt{1-k^2} \end{bmatrix}$

Tabla 14-1

Para iniciar el diseño de un oscilador o generador digital se debe definir el valor de la constante k , y por defecto, el valor del ángulo θ , este se define por medio de la siguiente ecuación:

$$\theta = \frac{2\pi F_g}{F_s} \quad \text{Ecuación 14-22}$$

Donde F_g es la frecuencia en hercios de la señal que se desea generar, y F_s es la frecuencia de muestreo en hercios. La implementación de estos generadores siempre usa la misma forma de operaciones matemáticas, que es la siguiente:

$$\begin{aligned}y(n) &= ay(n-1) + bx(n-1) \\x(n) &= cy(n-1) + dx(n-1)\end{aligned}$$

Para que el oscilador arranque correctamente es importante dar valores iniciales a las salidas y_1 , y y_2 . Por obvias razones los osciladores de mayor velocidad son los que solo requieren de una multiplicación por iteración.

14.7.1 Ejemplo de diseño oscilador doble cuadrado

$$F_s = 1291,32\text{Hz.}$$

$$F_g = 100\text{Hz.}$$

El valor del ángulo θ , es el que se calcula a continuación:

$$\theta = \frac{2\pi F_g}{F_s} = \frac{2\pi 100}{1291,32} = 0,4865707421$$

Con el valor θ se puede calcular el valor de k , que es el siguiente:

$$k = 2\cos(0,4865707421) = 1,76788313$$

De esta manera se tiene la siguiente matriz de rotación:

$$\begin{bmatrix} 1,76788313 & -1 \\ 1 & 0 \end{bmatrix}$$

Las ecuaciones de implementación son las siguientes:

$$y(n) = 1,76788313y(n-1) + x(n-1)$$

$$x(n) = y(n-1)$$

Las mismas funciones en términos de solo la salida y es igual a la siguiente ecuación:

$$y(n) = 1,76788313y(n-1) + y(n-2)$$

La condición para el funcionamiento de este oscilador es que la magnitud de la señal debe ser 1. Otra condición para el funcionamiento adecuado de este oscilador es dar valores iniciales a las salidas $y(n-1)$, y $y(n-2)$, las cuales se pueden iniciar de la siguiente forma:

$$y(n-1) = \cos(-\theta) = \cos(-0,4865707421).$$

$$y(n-2) = \cos(-2\theta) = \cos(-2(0,4865707421)).$$

Para manipular la amplitud se debe multiplicar la salida $y(n)$, por el factor deseado.

Para fines de simulación, se implementarán las mismas condiciones del hardware usado en los filtros FIR, e IIR, omitiendo el generador de onda seno que se conecta a la entrada análoga. Teniendo presente las anteriores observaciones se puede implementar el siguiente código fuente en lenguaje C, para simular este ejemplo:

```
//Declaración de variables inicializadas.
float y0, y1=1.0, y2=0.883941565;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Implementación de la ecuación del oscilador.
        y0 = 1.76788313*y1 - y2;
        //Se hace el corrimiento de la salida.
        y2 = y1;
        y1 = y0;
        //Reconstrucción de la señal: y en 10 bits, amplificada 500 veces.
        YY = (unsigned int)(500*y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}
```

14.7.2 Ejemplo de diseño para oscilador de acople en cuadratura

$F_s = 1291,32\text{Hz}$.

$F_g = 150\text{Hz}$.

$A = 450$.

Se calcula el ángulo de paso θ :

$$\theta = \frac{2\pi F_g}{F_s} = \frac{2\pi 150}{1291,32} = 0,7298561132$$

Posteriormente se calculan las constantes de la matriz:

$$a = d = \sqrt{1 - [\text{sen}(\theta)]^2} = \sqrt{1 - [\text{sen}(0,7298561132)]^2} = 0,7452703484.$$

$$b = -c = \text{sen}(\theta) = 0,6667624073.$$

El patrón de la matriz de rotación es el siguiente:

$$\begin{bmatrix} 0,7452703484 & 0,6667624073 \\ -0,6667624073 & 0,7452703484 \end{bmatrix}$$

El juego de ecuaciones que rigen las iteraciones del oscilador son las siguientes:

$$y(n) = 0,7452703484y(n-1) + 0,6667624073x(n-1)$$

$$x(n) = -0,6667624073y(n-1) + 0,7452703484x(n-1)$$

Para dar inicio correcto al oscilador se deben dar valores iniciales a las salidas retardadas $y(n-1)$, y $x(n-1)$, para este fin se debe tener presente que este oscilador tiene las dos salidas en cuadratura, es decir que entre las dos salidas siempre existe un desfase de 90 grados. Bajo este concepto se puede concluir que las dos señales en cuadratura tendrán la misma magnitud a $3\pi/4$, de esta manera se asignan los siguientes valores iniciales:

$$y(n-1) = x(n-1) = A \text{Sen}(3\pi/4),$$

$$y(n-1) = x(n-1) = 450 \text{Sen}(3\pi/4) = 318,1980515.$$

Finalmente se implementa, el código fuente en lenguaje C, para este oscilador:

```
//Declaración de variables inicializadas.
float y0, y1=318.1980515, x0, x1=318.1980515;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        //Implementación de las ecuaciones del oscilador.
        y0 = 0.7452703484*y1 + 0.6667624073*x1;
        x0 = -0.6667624073*y1 + 0.7452703484*x1;
        //Se hace el corrimiento de la salida.
```

```

    y1 = y0;
    x1 = x0;
    //Reconstrucción de la señal: y en 10 bits.
    YY = (unsigned int)(y0+512.0);
    PORTC = (YY>>8)&3;
    PORTB = YY&255;
    INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.8 Transformada discreta de Fourier DFT

La transformación de Fourier es un procedimiento matemático que permite analizar las componentes espectrales y la intensidad de potencia de una señal análoga, o discreta. Las aplicaciones de este procedimiento son numerosas en las que se pueden nombrar: Analizadores de espectro, reconocimiento de patrones, en señales, de comunicaciones, biomédicas, y audio, entre otras. La transformada de Fourier, es una temática sumamente densa, sin embargo este capítulo busca mostrar de forma simple la implementación de esta herramienta. La transformada discreta de Fourier o DTF, está definida por la siguiente ecuación:

$$X(j\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} \quad \text{Ecuación 14-23}$$

Como se puede apreciar en la anterior ecuación la potencia de una frecuencia ω , se puede evaluar en un fragmento de señal de longitud L , de igual manera se puede notar que los valores resultantes de la función $X(j\omega)$, son números complejos, por esta razón los análisis de Fourier se hacen por excelencia en términos de la magnitud de estos números complejos. La ecuación (15.23) en términos de un número complejo rectangular es la siguiente:

$$X(j\omega) = \sum_{n=0}^{L-1} x(n)[\cos(\omega n) - j\sin(\omega n)] \quad \text{Ecuación 14-24}$$

De esta forma la relación $X(jw)$, puede ser pasada a la magnitud de sí misma como $|X(jw)|$, y su relación matemática es la siguiente:

$$|X(jw)| = \sqrt{\left[\sum_{n=0}^{L-1} x(n) \cos(wn) \right]^2 + \left[\sum_{n=0}^{L-1} x(n) \sin(wn) \right]^2} \quad \text{Ecuación 14-25}$$

Esta ecuación matemática permite evaluar la potencia, o la intensidad de la componente w , dentro de la muestra $x(n)$, de longitud L .

Un fragmento de código en lenguaje C, que realice esta operación se puede implementar con una función como la siguiente:

```
//Función para determinar la intensidad de potencia
//de la componente w, en la señal x_n.
float TDF( float *x_n, float w, unsigned int L )
{
    //Declaracion de variables.
    unsigned int n;
    float R=0.0, I=0.0;
    //Bucle for para realizar las sumatorias.
    for( n=0; n<L; n++ )
    {
        //Cálculo y sumatoria de los componentes
        //reales e imaginarios.
        R += cos( w*n );
        I += sin( w*n );
    }
    //Se retorna el valor de la magnitud del
    // número complejo.
    return sqrt( R*R + I*I );
}
```

El siguiente ejemplo implementa un programa que permite detectar la presencia de tres frecuencias diferentes en una señal análoga, por medio de la transformada de Fourier. Como primera medida se definen las tres frecuencias análogas y sus respectivas frecuencias digitales, $F_1=100\text{Hz}$, $F_2=200\text{Hz}$, y $F_3=300\text{Hz}$, la frecuencia de muestreo se define de $1291,32\text{Hz}$, con el fin de implementar la misma estructura del microcontrolador 18F452, que se usó en los filtros FIR, e IIR.

$$W_1 = \frac{2\pi 100}{1291.32} = 0,48657707421$$

$$W_2 = \frac{2\pi 200}{1291.32} = 0,9731414842$$

$$W_3 = \frac{2\pi 300}{1291.32} = 1,459712226$$

Para realizar la transformada de Fourier, se usará la función antes citada. Una salida del microcontrolador se activa cuando se detecta la potencia suficiente de la componente espectral. La señal es adquirida por una entrada análoga del PIC, la cual es la suma de tres fuentes diferentes. El programa correspondiente para este ejercicio es el siguiente:

```
//Declaración de variables inicializadas.
float x[64];
unsigned short i, OK=1;
//Declaración de la función de interrupciones.

//Función para determinar la intensidad de potencia
//de la componente w, en la señal x_n.
float TDF( float *x_n, float w, unsigned int L )
{
    //Declaración de variables.
    unsigned short n;
    float R=0.0, I=0.0;
    //Bucle for para realizar las sumatorias.
    for( n=0; n<L; n++ )
    {
        //Cálculo y sumatoria de los componentes
        //reales e imaginarios.
        R += x_n[n]*cos( w*n );
        I += x_n[n]*sin( w*n );
    }
    //Se retorna el valor de la magnitud del
    // número complejo.
    return sqrt( R*R + I*I );
}

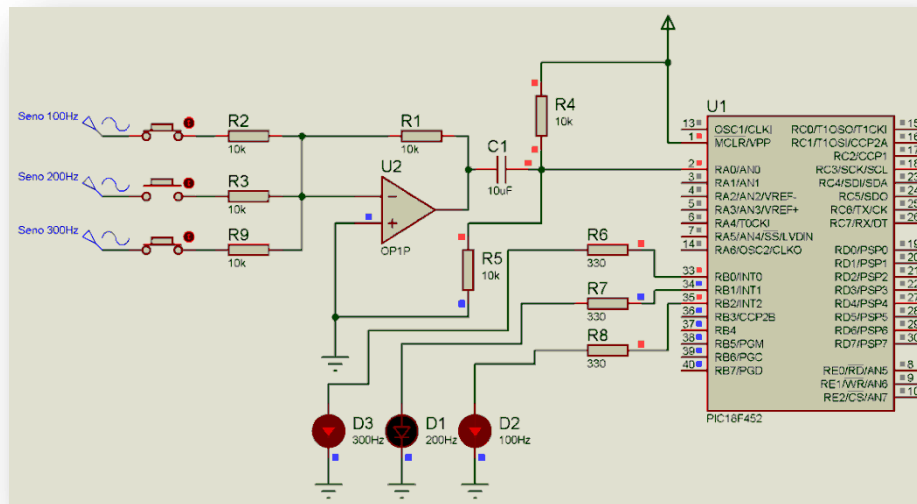
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        if( OK ) //Se evalúa la adquisición de muestras.
        {
            OK++; //Se cuentan las muestras tomadas.
            //Se corren las últimas 64 muestras en el bufer x.
            for( i=63; i!=0; i-- )x[i]=x[i-1];
            //Se guarda la última muestra.
            x[0] = ((float)ADC_Read(0)-512.0);
        }
        INTCON.F2=0;
    }
}
```

```

void main( void )
{
    float RES1, RES2, RES3;
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
        while( OK<65 );//Se espera a recibir 64 muestras.
        OK =0; //Se suspenden la adquisición de muestras.
        //Se hace la transformada de Fourier, de la componente de 100Hz.
        RES1 = TDF( x, 0.4865707421, 64 );
        //Se hace la transformada de Fourier, de la componente de 200Hz.
        RES2 = TDF( x, 0.9731414842, 64 );
        //Se hace la transformada de Fourier, de la componente de 300Hz.
        RES3 = TDF( x, 1.459712226, 64 );
        OK=1; //Se activa la adquisición de muestras.
        //Se evalúa la potencia de la señal de 100Hz.
        if( RES1>800 )PORTB.F2=1; else PORTB.F2=0;
        //Se evalua la potencia de la señal de 200Hz.
        if( RES2>800 )PORTB.F1=1; else PORTB.F1=0;
        //Se evalúa la potencia de la señal de 300Hz.
        if( RES3>800 )PORTB.F0=1; else PORTB.F0=0;
    }
}

```

Para realizar la simulación de este ejercicio se implementa en ISIS, los dispositivos: 18F452, RES, CAP, OP1P, BUTTON, LED-RED, y generador virtual. El circuito correspondiente para está simulación es el siguiente:



Circuito 14-2

La transformada de Fourier, implica múltiples cálculos matemáticos, como sumas, multiplicaciones y evaluaciones trigonométricas. Estos procesos hacen lento el rendimiento de la máquina de proceso. Por esta razón la ciencia de tratamiento de señales, se vio en la necesidad de buscar técnicas de realizar la transformada de Fourier de forma mucho más eficiente, de este esfuerzo, sale como resultado notable el algoritmo de transformación rápida de Fourier, o FFT. Este algoritmo se fundamenta en la división en segmentos cortos para realizar la evaluación de la transformada de Fourier.

14.8.1 Transformada rápida de Fourier FFT

La transformada rápida de Fourier, organiza en una matriz las muestras de entrada y realiza transformadas más pequeñas en las filas y posteriormente en las columnas. Las muestras digitalizadas de la señal análoga se pueden representar de la siguiente forma:

$$x(0), x(1), x(2), x(3), x(4), x(5), \dots, x(N)$$

Donde N es el número máximo de muestras que se desean analizar.

Las muestras son organizadas en una matriz de L filas, con M columnas. En la siguiente matriz se puede apreciar el almacenamiento por columnas:

$$x(l, m) = \begin{bmatrix} x(0) & x(4) & x(8) & x(12) \\ x(1) & x(5) & x(9) & x(13) \\ x(2) & x(6) & x(10) & x(14) \\ x(3) & x(7) & x(11) & x(15) \end{bmatrix}$$

Seguidamente se debe realizar la transformación por filas, haciendo la transformada de M muestras. Esta operación se realiza en función de la siguiente ecuación:

$$F(l, q) = \sum_{m=0}^{M-1} [x(l, m) W_M^{mq}]$$

$$0 \leq l \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_M^{mq} = e^{-j \frac{2\pi m q}{M}} = \cos\left(\frac{2\pi m q}{M}\right) - j \sin\left(\frac{2\pi m q}{M}\right)$$

Ecuación 14-26

La matriz F(l,q), es un arreglo de valores complejos correspondiente a las transformadas de M, puntos sobre cada una de las filas. Seguidamente la matriz F(l,q), se multiplica por los factores de fase W_N, esta operación se define en la siguiente ecuación:

$$G(l, q) = W_N^{lq} F(l, q)$$

$$0 \leq l \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_N^{lq} = e^{-j \frac{2\pi l q}{N}} = \cos\left(\frac{2\pi l q}{N}\right) - j \sin\left(\frac{2\pi l q}{N}\right)$$

Ecuación 14-27

Por último se realiza la transformada por columnas de L muestras. Esta operación se resume en la siguiente ecuación:

$$\begin{aligned}
 X(p, q) &= \sum_{l=0}^{L-1} [G(l, q) W_L^{lp}] \\
 0 \leq p &\leq (L-1) \\
 0 \leq q &\leq (M-1) \\
 W_L^{lp} &= e^{-j \frac{2\pi p l}{L}} = \cos\left(\frac{2\pi p l}{L}\right) - j \sin\left(\frac{2\pi p l}{L}\right)
 \end{aligned}$$

Ecuación 14-28

La matriz X(p,q), es la transformada final de las N muestras iniciales. Esta matriz contiene en cada uno de sus términos un número complejo que representa la magnitud y la fase de las componentes espectrales. La matriz X(p,q), se debe leer por filas, como se aprecia en la siguiente matriz:

$$X(p, q) = \begin{bmatrix} X(0) & X(1) & X(2) & X(3) \\ X(4) & X(5) & X(6) & X(7) \\ X(8) & X(9) & X(10) & X(11) \\ X(12) & X(13) & X(14) & X(15) \end{bmatrix}$$

Cabe denotar que entre más muestras se ingresen para hacer la transformación, mayor será la resolución en el espectro, pero a su vez mayor será la cantidad de recursos de procesamiento y de memoria requeridos. A continuación se resume el proceso antes citado:

- Se archiva la señal de x(n), por columnas en la matriz x(l,m).
- Se calcula la transformada de M muestras de cada una de las filas.
- Se multiplican las anteriores transformadas por el factor de fase Wn.
- Se calcula la transformada de L muestras de cada una de las columnas.
- Se leen los datos de la matriz X(p,q), por filas.

El resultado de una transformada de Fourier, entrega un espectro evaluado de 0 a 2π , lo que significa que los valores superiores a π , son un espejo de las componentes inferiores causadas por los armónicos de las frecuencias inferiores a π . En conclusión la lectura de la matriz solo se debe hacer hasta la mitad, por ejemplo si la cantidad de muestras analizadas son 128, solo se deben leer 64. De la misma manera la resolución de la transformación en hercios está definida por la siguiente ecuación:

$$R_{Hz} = \frac{F_s}{N}$$

Ecuación 14-29

Donde R, es la resolución en hercios, o el mínimo ancho de banda que se puede analizar con la FFT. F_s es la frecuencia de muestreo, y N es el número de muestras ingresadas a la FFT.

También cabe denotar que la primera muestra de la FFT, hace alusión a la componente espectral más baja incluida la frecuencia 0, o en otras palabras los niveles DC. Estas componentes son las más intensas, y para fines prácticos esta primera muestra puede ser ignorada.

Para realizar un ejemplo con la FFT, se implementará el PIC 18F4585, dada su amplia memoria RAM de 3328 Bytes, útil para los procesos de la FFT. Sin embargo esta capacidad de memoria solo permite hacer una FFT, con máximo 169 muestras.

A continuación se presentará un ejemplo en lenguaje C, para realizar la transformación de una señal de $N = ML$, muestras:

```
#define M 16
#define L 8
#define N L*M
#define PI 3.141592654

typedef struct
{
    float R, I;
}Complejo;

//Declaración de variables inicializadas.

unsigned short i, OK=0;
unsigned short Fi=0, Co=0;
//Declaración de la función de interrupciones.

Complejo X[L][M];
Complejo Y[L][M];

//Función para realizar una multiplicación compleja.
Complejo Producto( Complejo A, Complejo B )
{
    Complejo Res;
    Res.R = A.R*B.R - A.I*B.I;
    Res.I = A.R*B.I + A.I*B.R;
    return Res;
}

//Función para realizar una suma compleja.
Complejo Adicion( Complejo A, Complejo B )
{
    Complejo Res;
    Res.R = A.R + B.R;
    Res.I = A.I + B.I;
    return Res;
}

//Función para determinar la magnitud de un número complejo.
float Magnitud( Complejo A )
{
    return sqrt( A.R*A.R + A.I*A.I );
}
```

//Cálculo rápido de Fourier, de N muestras.

void FFT(void)

{

unsigned short l,q,m,p;

Complejo WW;

//Trío de bucles for, para ejecutar la

//ecuación (15.26), la matriz Y[l][], representa F(l,q).

for(l=0; l<L; l++)

{

for(q=0; q<M; q++)

{

Y[l][q].R = 0.0;

Y[l][q].I = 0.0;

for(m=0; m<M; m++)

{

WW.R = cos(2*PI*m*q / M);

WW.I = -sin(2*PI*m*q / M);

WW = Producto(X[l][m], WW);

Y[l][q] = Adicion(Y[l][q], WW);

}

}

}

//Dupla se bucles for, para realizar la multiplicación

//de factores Wn (15.27), Y[l][] representa la matriz F(l,q), y

//X[l][] representa la matriz G(l,q).

for(l=0; l<L; l++)

{

for(q=0; q<M; q++)

{

WW.R = cos(2*PI*l*q / N);

WW.I = -sin(2*PI*l*q / N);

X[l][q] = Producto(Y[l][q], WW);

}

}

//Trío de bucles for, para ejecutar la

//ecuación (15.28), la matriz Y[l][], representa X(p,q),

//y X[l][] representa la matriz G(l,q).

for(p=0; p<L; p++)

{

for(q=0; q<M; q++)

{

Y[p][q].R=0.0;

Y[p][q].I=0.0;

for(l=0; l<L; l++)

{

```

        WW.R = cos( 2*PI*l*p / L );
        WW.I = -sin( 2*PI*l*p / L );
        WW = Producto( X[l][q], WW );
        Y[p][q] = Adicion( Y[p][q], WW );
    }
}

//Doble for anidado para determinar,
//la magnitud de la transformada,
//Este resultado queda en los términos reales de X[][]
for( l=0; l<L; l++ )
for( m=0; m<M; m++ )
{
    X[l][m].R = Magnitud( Y[l][m] );
    X[l][m].I = 0.0;
}

}

void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        if( OK ) //Se evalúa la adquisición de muestras.
        {
            //Se hace la adquisición de las muestras,
            //y se archivan por columnas.
            X[Fi][Co].R = ((float)ADC_Read(0)-127.0);
            X[Fi][Co].I = 0.0;
            Fi++;
            if( Fi==L )
            {
                Fi = 0;
                Co++;
                if( Co==M )
                {
                    Co=0;
                    OK=0;
                }
            }
        }
        INTCON.F2=0;
    }
}

```

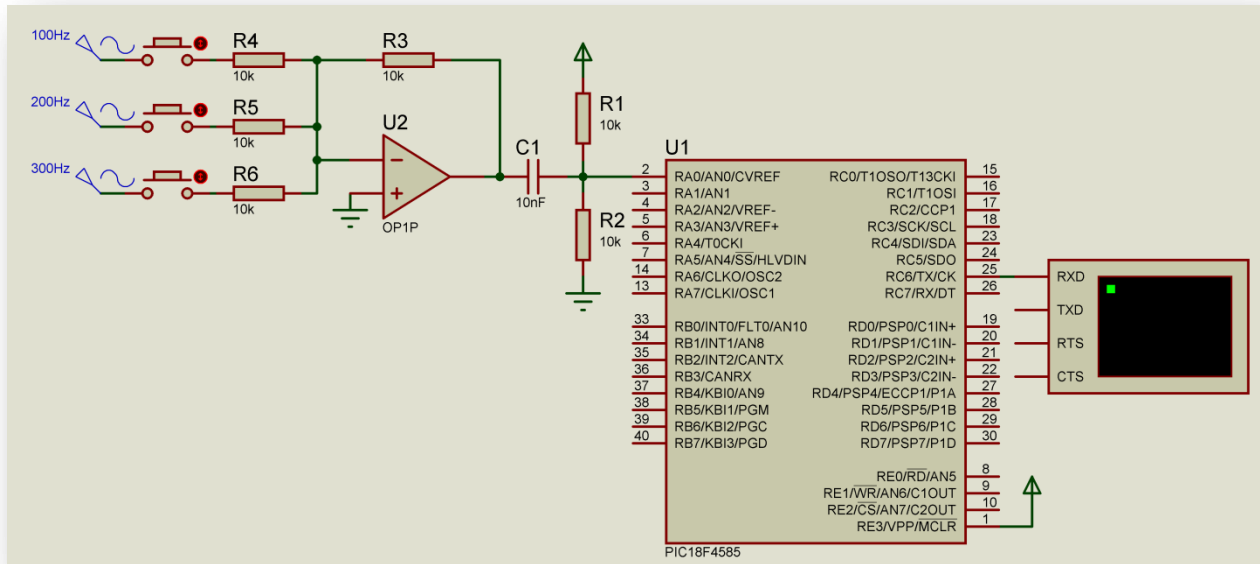
```

void main( void )
{
    char Text[30];
    unsigned short ff, cc, cont;
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    UART1_Init(9600);

    while(1)//Bucle infinito.
    {
        OK = 1; //Se habilita la lectura de muestras.
        UART1_Write_Text("// Adquisicion y FFT de:");
        IntToStr( N, Text );
        UART1_Write_Text(Text);
        UART1_Write_Text(" Muestras //");
        UART1_Write(13); UART1_Write(10);
        while( OK ); //Se espera la adquisicion de las muestras.
        FFT(); //Cálculo de la FFT.
        UART1_Write_Text("// Inicio de las muestras //");
        UART1_Write(13); UART1_Write(10);
        //La primera componente espectral se ignora y se hace igual a cero.
        X[0][0].R = 0.0;
        cont = 0;
        //Se envían la magnitud de la FFT, hasta N/2
        //por medio del puerto serial.
        for( ff=0; ff<L; ff++ )
        {
            for( cc=0; cc<M; cc++ )
            {
                FloatToStr( X[ff][cc].R, Text );
                UART1_Write_Text(Text);
                UART1_Write(13); UART1_Write(10);
                cont++;
                if( cont==N )
                {
                    cc = M;
                    ff = L;
                }
            }
        }
        UART1_Write_Text("// Fin de las muestras //");
        UART1_Write(13); UART1_Write(10);
    }
}

```


Para simular este ejercicio en ISIS, se implementa un circuito similar al ejercicio anterior de transformada discreta de Fourier. Sustituyendo el PIC, por un 18F4585, y anexando un VIRTUAL TERMINAL. El circuito para simular es el siguiente:



Circuito 14-3

14.9 Control digital PID

El en ámbito del control automático la forma de control más popular es el PID, que significa proporcional, integral, derivativo. El control es una rama de la física y la electrónica que permite manipular una variable física para permanecer en un valor deseado arbitrariamente. Todo sistema de control cuenta con las siguientes características:

- Punto de control
- Error
- Controlador (PID)
- Corrección
- Planta
- Variable a controlar
- Sensor o transductor

Estás características se pueden apreciar en el siguiente diagrama en bloques:

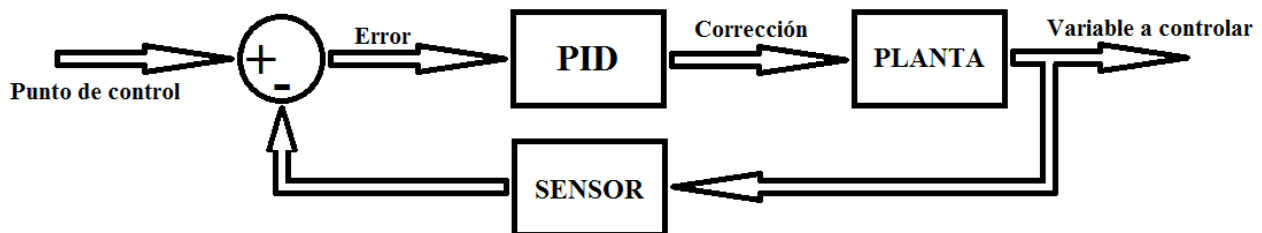


Figura 14-20

Para el caso del microcontrolador su función está sujeta a la entrada del punto de control, la lectura del sensor, y la salida de corrección. El punto de partida de un controlador es el controlador PID análogo que tiene una función de transferencia PID(s), esta función tiene una relación similar a un conjunto de filtros y su tratamiento es equivalente a un filtro IIR. La función de transferencia análoga es la siguiente:

$$PID(s) = Kp + \frac{Ki}{s} + sKd \quad \text{Ecuación 14-30}$$

De la misma forma que se hace con los filtros IIR, se realiza la transformación bilineal, y este desarrollo da el siguiente análisis:

$$PID(z) = Kp + \frac{Ki}{\frac{2Fs(1-z^{-1})}{(1+z^{-1})}} + \frac{2FsKd(1-z^{-1})}{(1+z^{-1})}$$

$$PID(z) = Kp + \frac{Ki(1+z^{-1})}{2Fs(1-z^{-1})} + \frac{2FsKd(1-z^{-1})}{(1+z^{-1})}$$

$$PID(z) = Kp + \frac{\frac{Ki}{2Fs}(1+z^{-1})}{(1-z^{-1})} + \frac{2FsKd(1-z^{-1})}{(1+z^{-1})}$$

Del anterior análisis se pueden deducir tres funciones de transferencia para el controlador donde la entrada, es la señal de error para cada una de ellas. Las respectivas operaciones matemáticas para su implementación son las siguientes:

$$y_p(n) = [Kp]e(n)$$

$$y_i(n) = \frac{Ki}{2Fs} [e(n) + e(n-1)] + y_i(n-1) \quad \text{Ecuación 14-31}$$

$$y_d(n) = 2FsKd[e(n) - e(n-1)] - y_d(n-1)$$

$$y_{pid}(n) = y_p(n) + y_i(n) + y_d(n)$$

Para el caso particular del siguiente ejemplo se emplea la técnica de conversión digital análogo por medio de una salida PWM, y como estrategia para simular el comportamiento de la planta se usa un circuito RC, de segundo orden. El siguiente código fuente muestra un control PID correspondiente a la sintonización del control PID, cuando su Kp crítico es de 1,8 y un periodo crítico de 41,2m segundos.

//Declaración de constantes y variables.

const float Kp=1.08, Ki=52.42718447, Kd=0.005562, Fs = 152.5878906;

//Declaración de coeficientes de integración y derivación.

const float Fi=ki/(2.0*Fs), Fd=2.0*Fs*Kd;

```

float SENSOR, PUNTO_DE_CONTROL=127.0, YN=0.0;
int ADQUI;
unsigned short SALIDA;

float e0=0.0, e1=0.0, yi0=0.0, yi1=0.0, yd0=0.0, yd1=0.0, ypid=0.0;

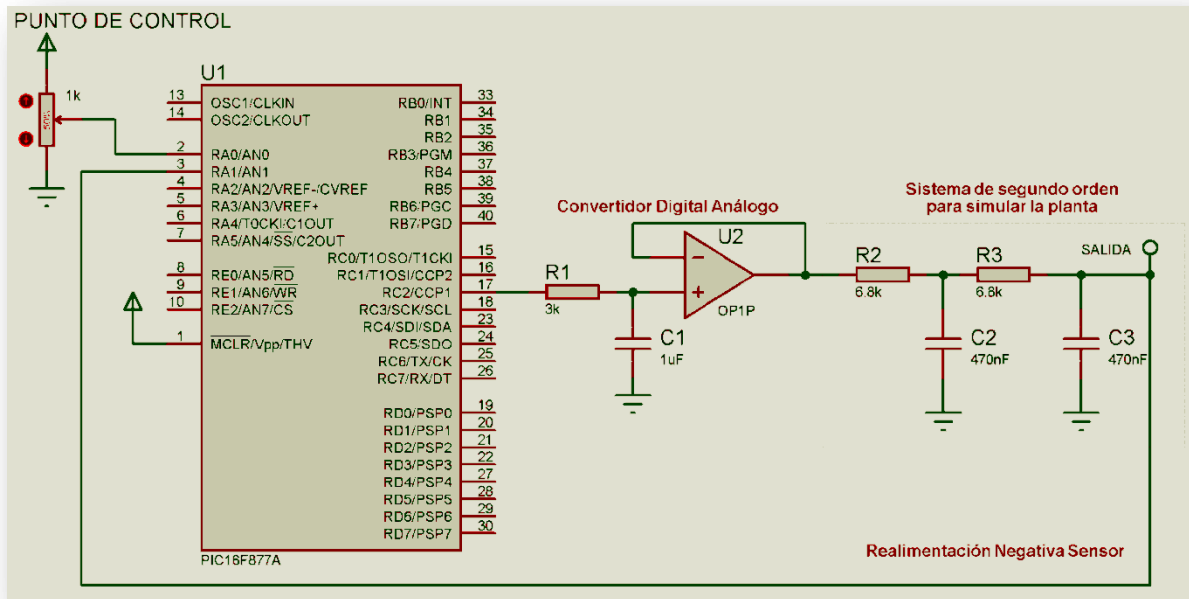
//Función de interrupciones para el Timer 0.
void interrupt()
{
    if( INTCON.F2 )// 6,5536ms :: 152,5878906 Hz
    {
        //Adquisición de la variable controlada.
        SENSOR = (float)((ADC_Read(1)>>2)&0xFF);
        //Adquisición del punto de control.
        PUNTO_DE_CONTROL = (float)((ADC_Read(0)>>2)&0xFF);
        //Calculo del nivel de error.
        e0 = PUNTO_DE_CONTROL - SENSOR;
        //Ecuación en diferencias.
        //Ecuación integral.
        yi0=Fi*(e0+e1)+yi1;
        //Ecuación derivativa.
        yd0=Fd*(e0-e1)-yd1;
        //Resultado PID.
        ypid=Kp*e0+yi0+yd0;
        //Ajuste y corrección de la SALIDA Y(n)
        //delimitada por los límites 0 y 255.
        YN += ypid;
        if(YN>255.0)YN=255.0;
        if(YN<0.0)YN=0.0;
        SALIDA = (unsigned short)(YN);
        PWM1_Set_Duty(SALIDA);
        //Actualización de muestras.
        e1=e0;
        yi1=yi0;
        yd1=yd0;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Configuración del modulo PWM.
    PWM1_Init(10000);
    PWM1_Start();
    //Configuración de la interrupción de Timer 0.
    //a 6,5536ms
    OPTION_REG = 0b00000110;
    INTCON = 0b10100000;
    while( 1 ) //Bucle infinito.

```

{
}
}

El circuito para simular implementa los dispositivos: 16F877A, POT-HG, RES, CAP, OP1P La simulación trabaja con reloj de 20MHz. El esquemático del circuito es el siguiente:



Circuito 14-4

Para la sintonía del controlador, y establecer las constantes K_p , K_i , y K_d , se puede usar la siguiente estrategia:

Primero se establece el controlador con las constantes K_i , y K_d igual a 0. Posteriormente se incrementa el valor de K_p , gradualmente hasta que la salida oscile de forma sostenida. Cuando esto sucede se mide el periodo de la oscilación sostenida y se determina el valor de K_p . Estos dos parámetros se denominan periodo crítico P_c , y K_p crítico K_{pc} . Posteriormente estos valores se rempazan en la siguiente tabla, para determinar los valores definitivos de K_p , K_i , y K_d :

Tipo de control	K_p	T_i	T_d
P	$0,5K_{pc}$	Infinito, $K_i=0$	0
PI	$0,45K_{pc}$	$P_c/1,2$	0
PID	$0,6K_{pc}$	$0,5P_c$	$0,125P_c$

Tabla 14-2

Donde, T_i es el periodo de integración y T_d el periodo derivativo. La relación de estos periodos con las constantes K_i , y K_d se especifica en las siguientes ecuaciones:

$$K_i = \frac{K_p}{T_i}$$

$$K_d = T_d K_p$$

Ecuación 14-32