Ecualizador paramétrico multibanda en paralelo

TRATAMIENTO DIGITAL DE SEÑALES

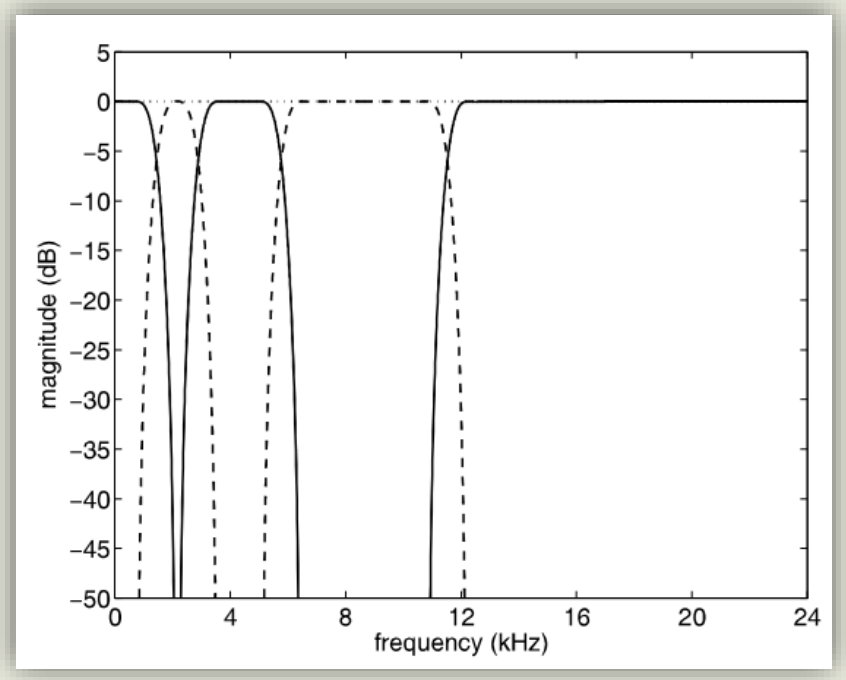
CARLES SERRA VENDRELL Y JORDI FERRANDO VILA

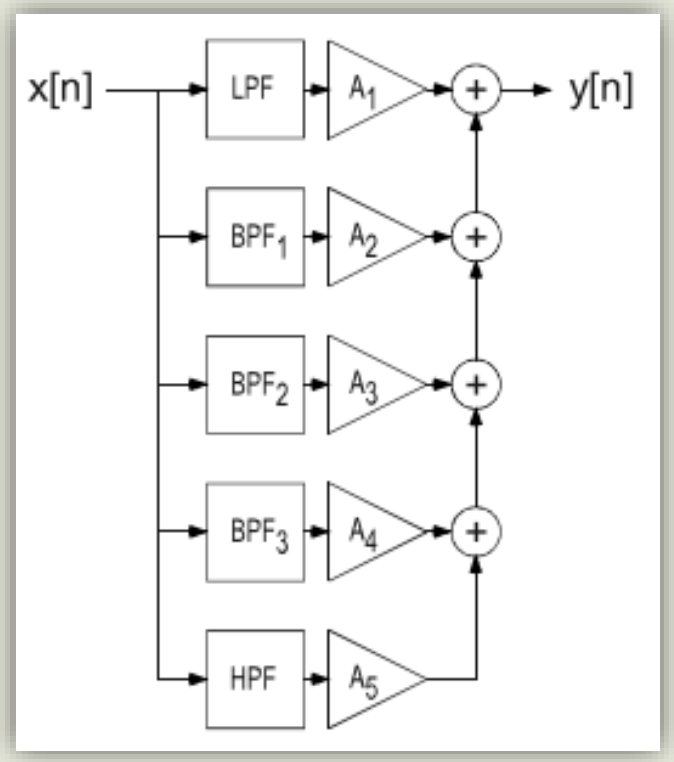
Índice

- Introducción
- Fase1 (Matlab)
  - Filtros: LowPass.m, HighPass.m, PassBand.m
  - EcualizaParalelo.m
  - Principal.m
- Fase2 (DSP)
  - Exportación de coeficientes
  - Primera solución
  - Solución final
- Conclusiones

Introducción:

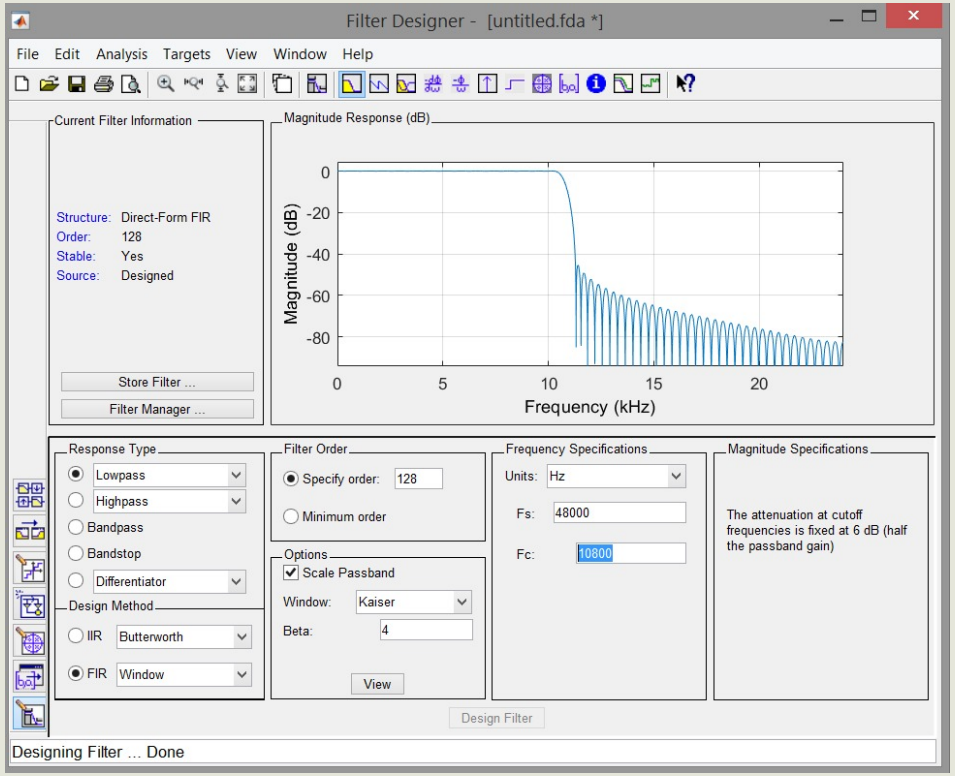
- Implementación del ecualizador paramétrico multibanda
- Filtros en paralelo
- Diferentes modos ecualización.





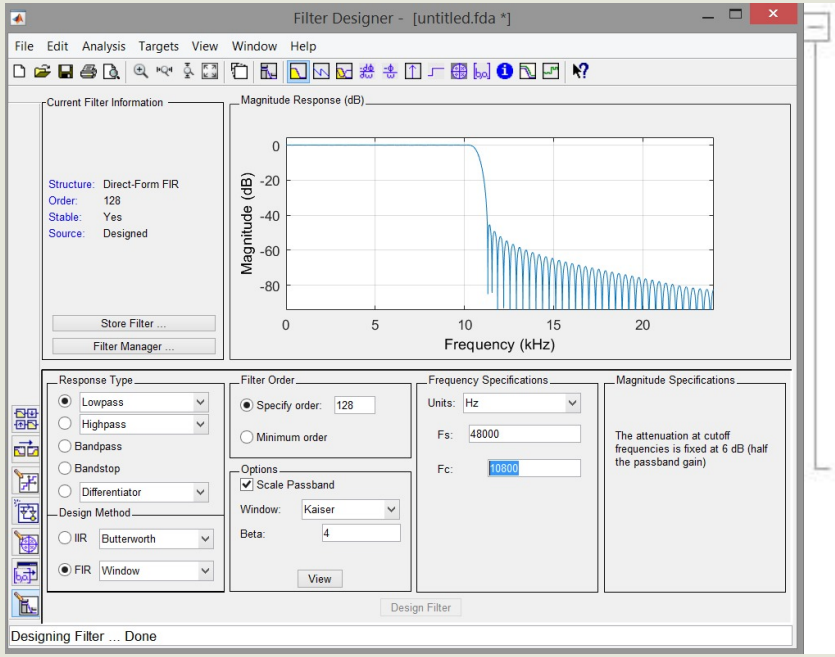
Fase1 (Matlab)
-Filtros

- Especificaciones
  - Frecuencia de corte a -6dB
  - Atenuación banda pasante a -40dB
- Filtro FIR por enventanado de Kaiser
- Especificación de orden
- Beta = 4



Fase1 (Matlab)
-Filtro LowPass

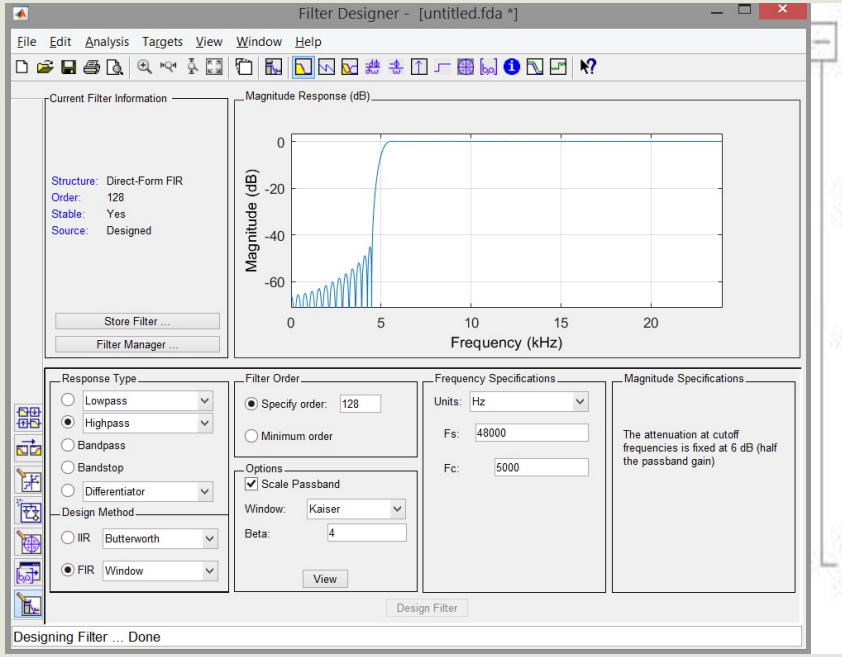
- Modificaciones: Variables por parámetros




```
function b = LowPass(Fc,Fs,N)
flag = 'scale';  % Sampling Flag
Beta = 4;        % Window Parameter

% Create the window vector for the design algorithm.
win = kaiser(N+1, Beta);

% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Fc/(Fs/2), 'low', win, flag);
```

Fase1 (Matlab)
-Filtro HighPass

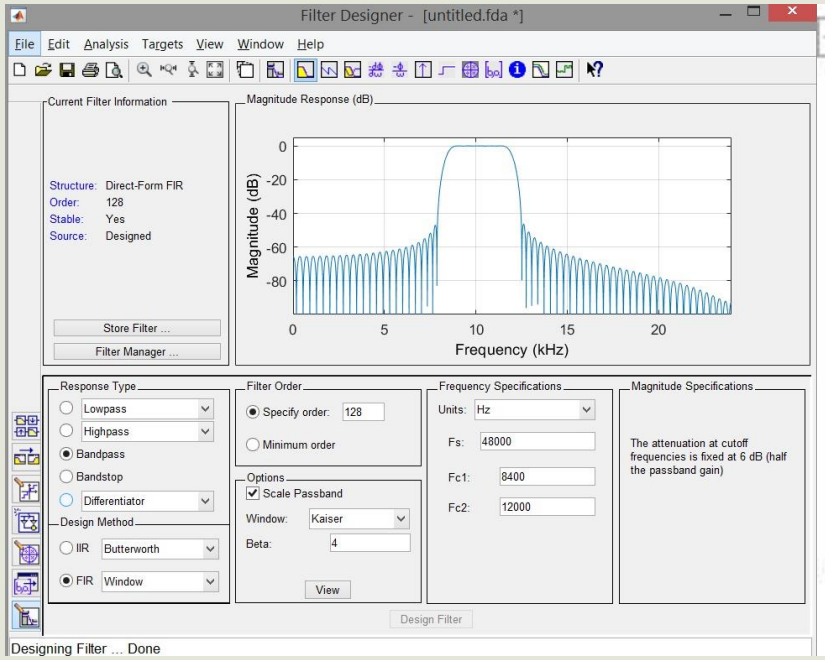
- Modificaciones: Variables por parámetros




```
function b = HighPass(Fc,Fs,N)
flag = 'scale';  % Sampling Flag
Beta = 4;        % Window Parameter
% Create the window vector for the design algorithm.
win = kaiser(N+1, Beta);

% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Fc/(Fs/2), 'high', win, flag);
```

Fase1 (Matlab)
-Filtro BandPass

- Modificaciones: Variables por parámetros y Frecuencias de corte




```
function b = BandPass(Fc1,Fc2,Fs,N)
flag = 'scale';  % Sampling Flag
Beta = 4;        % Window Parameter
% Create the window vector for the design algorithm.
win = kaiser(N+1, Beta);

% Calculate the coefficients using the FIR1 function.
b  = fir1(N, [Fc1 Fc2]/(Fs/2), 'bandpass', win, flag);
```

Fase1 (Matlab)
-Ecualización paralelo

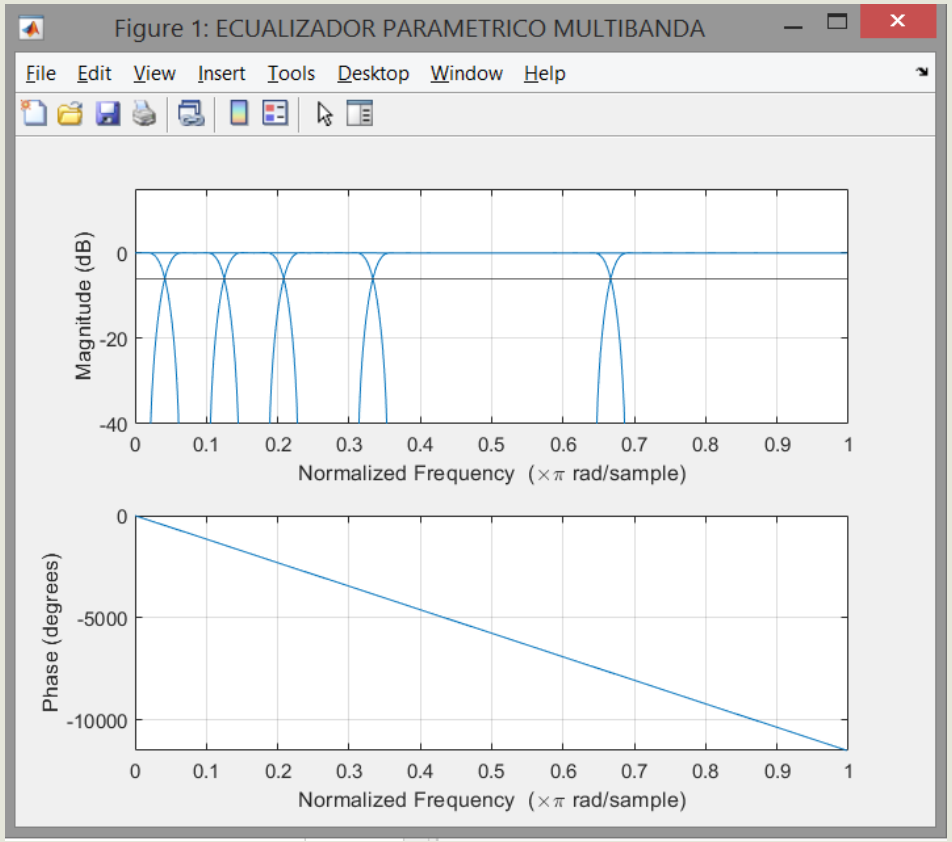
```
y = ecualizaParalelo(F,Ganancia,N,Fm,x)
```

Entradas:

F : Vector con frecuencias de los limites de las bandas.
Ganancia : Vector ganacias de las bandas en dB.
N : Orden de los filtros.
Fm : Frecuencia de muestreo.
x : Señal de entrada que filtraremos.

Salidas:

y: Señal de salida una vez aplicado el filtro a la señal x.



Fase1 (Matlab)
-Ecualización paralelo

- Convertimos de dB a lineal vector ganancia

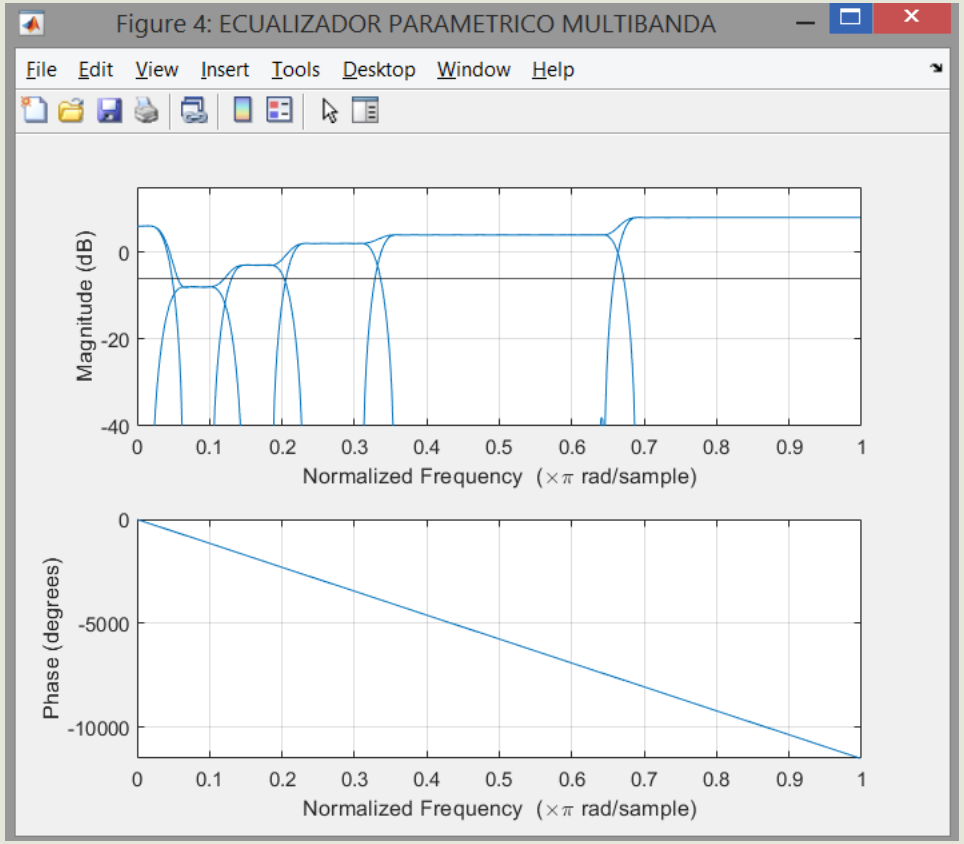
```
G = db2mag(Ganancia)
```

- Creamos el filtro Low Pass

```
lp = LowPass(F(1),Fm,N)*G(1)
```

- Creamos el filtro HighPass

```
hp = HighPass(F(length(F)),Fm,N)*G(length(G))
```



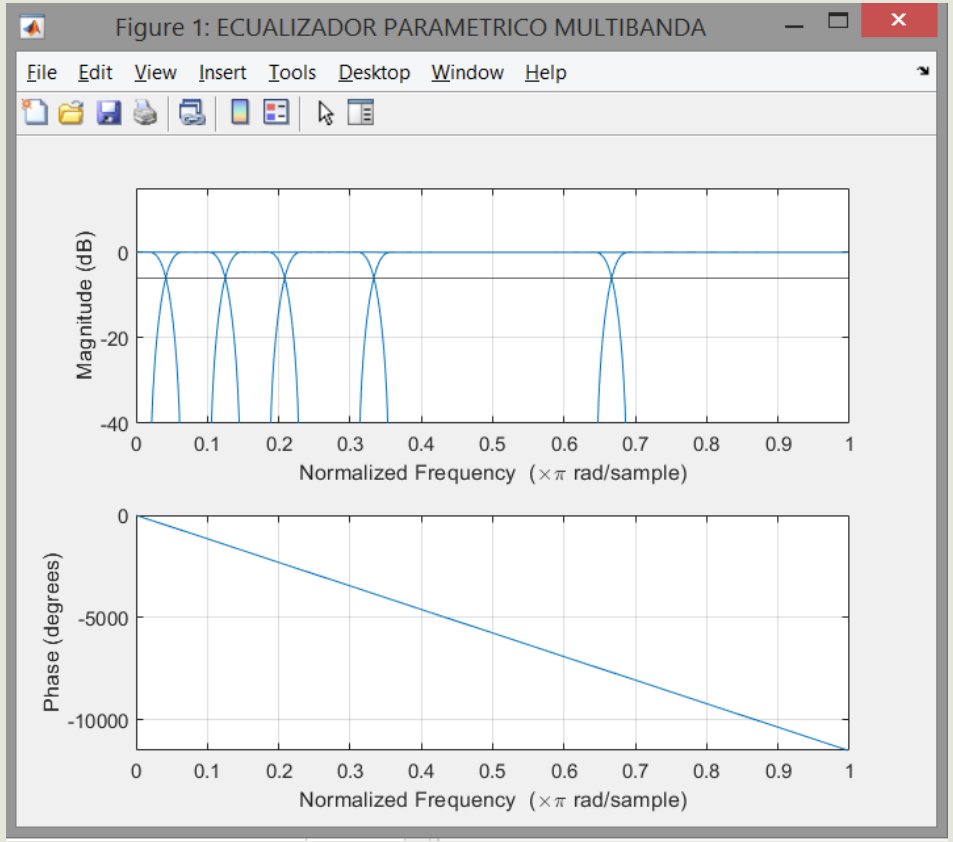
Fase1 (Matlab)
-Ecualización paralelo

- Creamos los filtros BandPass

```
bp_total = 0;    %Variable buffer

for i = 1:(length(G)-2)

   bp(i,:) = BandPass(F(i),F(i+1),Fm,N)*G(i+1)
   bp_total = bp_total + bp(i,:)

end
```

- Suma de las bandas para obtener la total

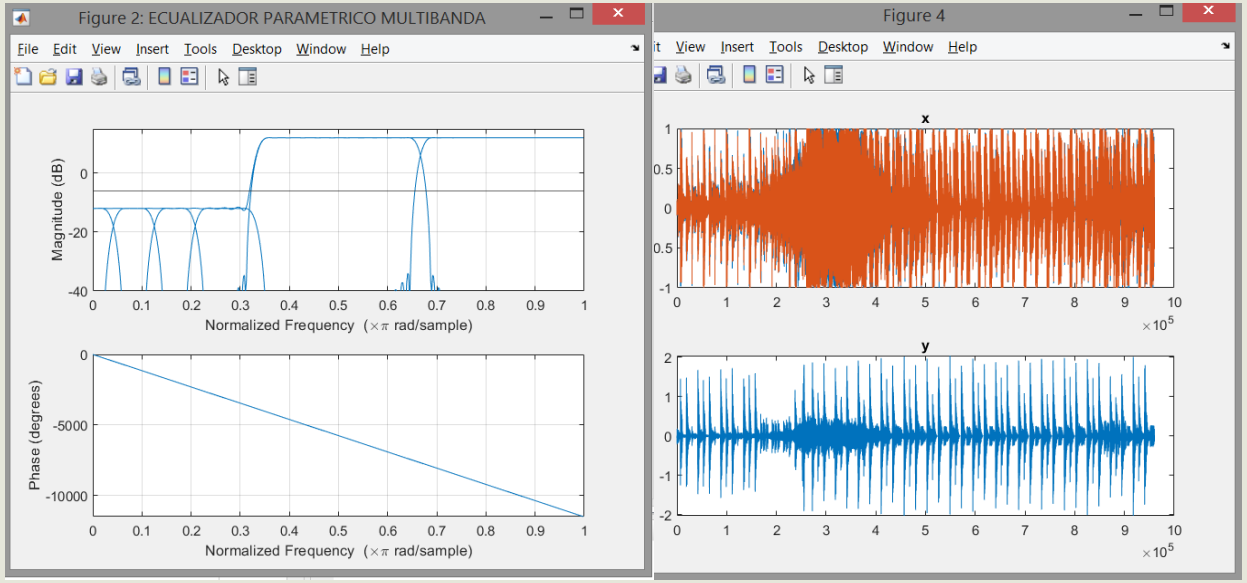
```
parametric = lp + hp + bp_total
```



Fase1 (Matlab)
-Ecualización paralelo

- Filtramos la señal de entrada:

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}$$




```
for n = length(parametric)+1:length(x)
        y(n) = 0;
        for k = 1:length(parametric)
            y(n) = y(n) + parametric(k) * x(n-k)
        end
    end
```

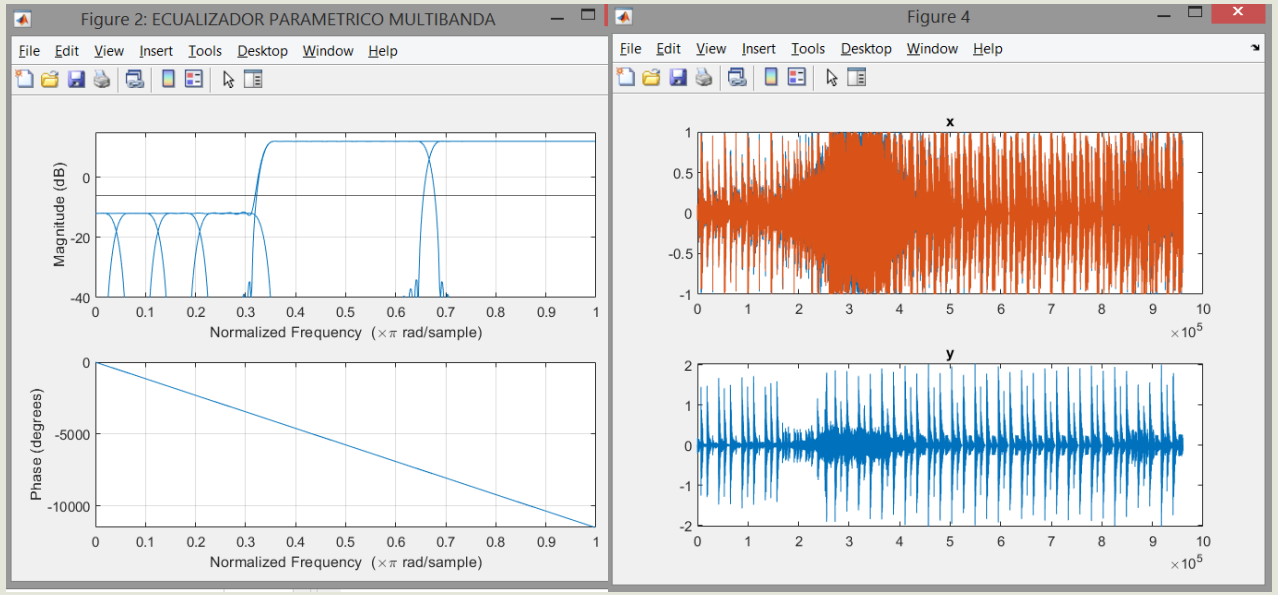
Fase1 (Matlab)
-Principal.m

% Cargamos Audio

```
[x,Fm]=audioread('Fisher-Losing It.wav')

N = 128;

Frecuencias =[1000,3000,5000,8000,16000]
```

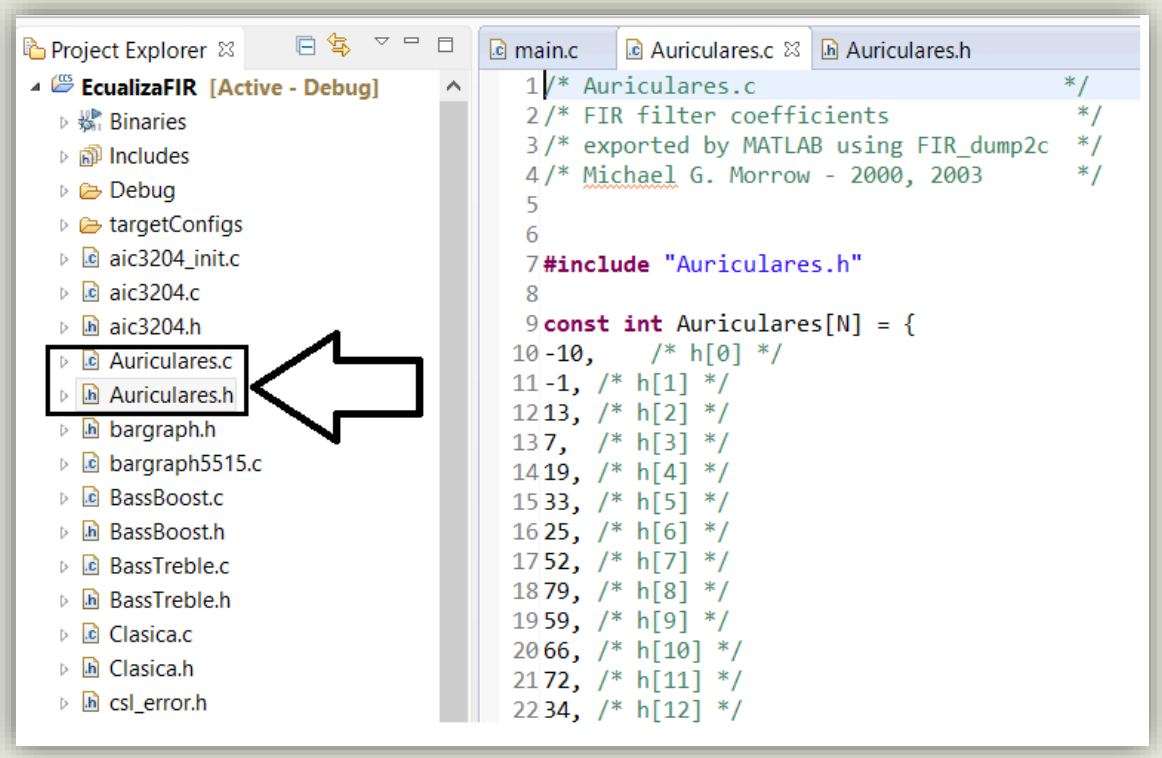
Modo de ecualización:

```
Eq0 = [12 -12 -12 -12 12 12]

y = ecualizaParalelo(Frecuencias, Rock, N, Fm,x(:,1))

sound(y,Fm)
```



Fase2 (DSP)
-Exportación de coeficientes

```
coeffs = FIR_dump2cIHPDS_INT('Auriculares','Auriculares',parametric,length(parametric))
```




```
#include "Llano.h"
#include "Rock.h"
#include "Dance.h"
#include "Techno.h"
#include "Pop.h"
#include "Clasica.h"
#include "Vivo.h"
#include "BassBoost.h"
#include "Treble.h"
#include "BassTreble.h"
#include "Auriculares.h"
```

Fase2 (DSP)
-Primera solución

- **Buffer delay**

for(j=0; j<N; j++) output += buffer[j]*h[j];

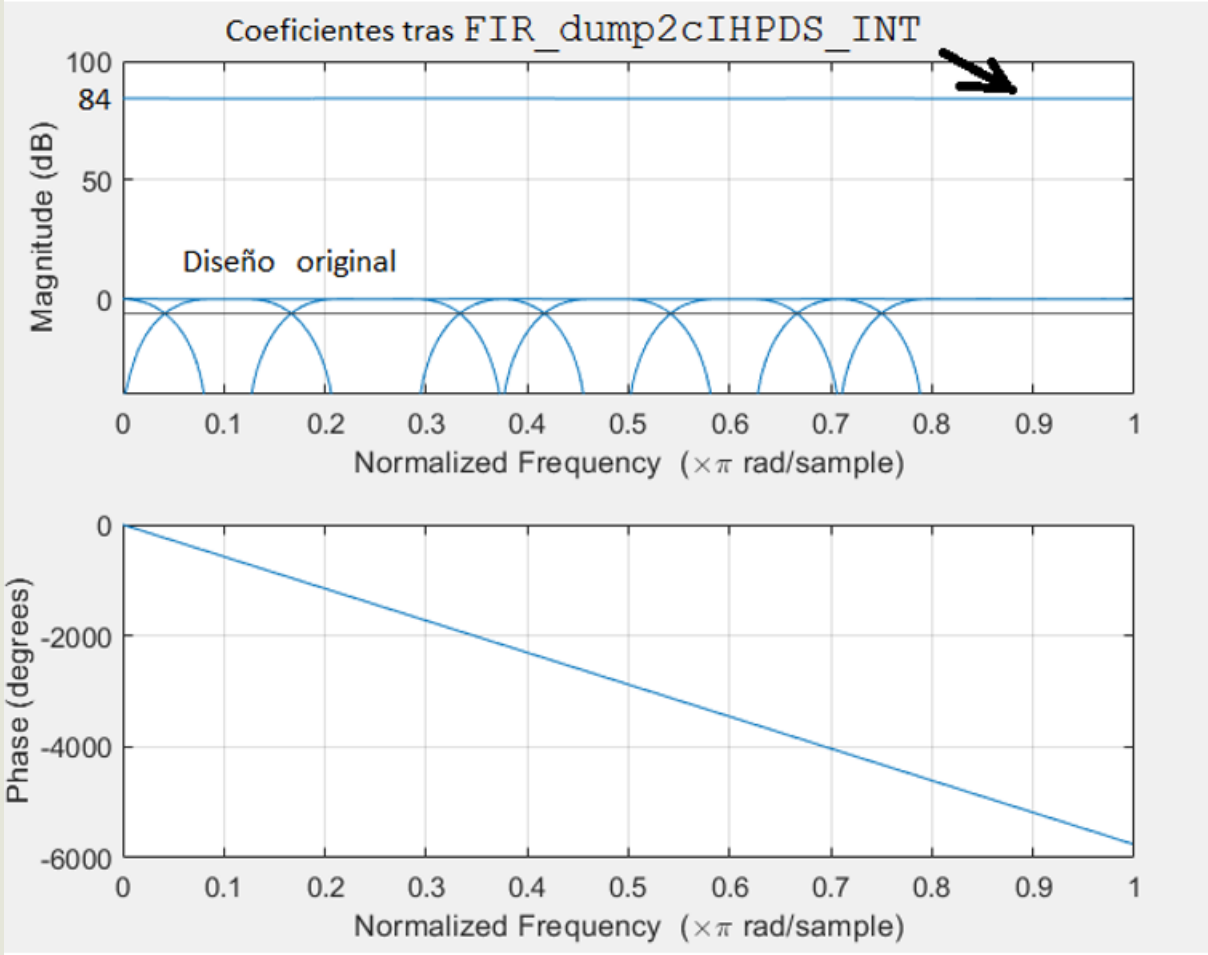
- **Almacenamos la muestra actual al inicio del buffer**

buffer[0] = mono_input;

- **Implementación filtro FIR**

for(j=N-1;j>0;j--) buffer[j] = buffer[j-1];

- **Problema saturación** →



Fase2 (DSP)
-Primera solución

- **Dividimos salida para bajar esta ganancia**
- **Logramos escuchar la salida pero muy bajo y con mucha distorsión.**

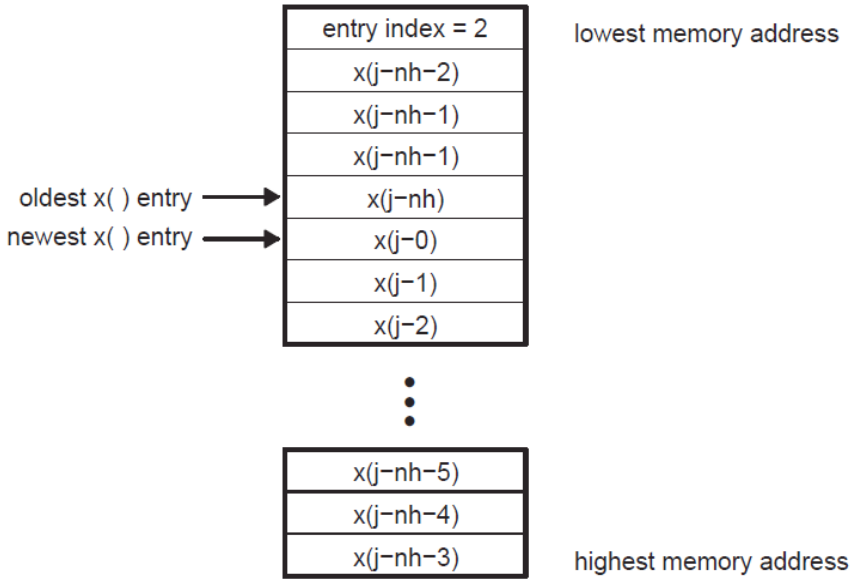
```
    }
//BUFFER DELAY

    for(j=N-1; j>0; j--) buffer_delay[j] = buffer_delay[j-1];   //Movemos a la derecha los retardos
    buffer_delay[0] = mono_input;
    //Añadinos muestra

//CONVOLUCION

    aux =0;                                                    //Vaciamos variable auxiliar.
    for(j=0; j<N; j++) aux += buffer_delay[j] * (Int32)Llano[j] / 64;//h(n)*x(n-k)
    aux /= 30;

    // Con es de 84.44 db --> sqrt(db2mag(84.44)) = 129

//SALIDA AUDIO

    output = (Int16)aux;                                       // Convertimos salida de la conv a INT16

    right_output = output;                                     // Sacamos por la salida el resultado de la conv.
    left_output =  right_output;                               // Salida izquierda lo mismo que derecha. MONO
}
```

Fase2 (DSP)
-Solución final

- **Usamos función fir DSPlib**

$$r[j] = \sum_{k=0}^{nh-1} h[k]\, x[j-k] \qquad 0 \le j \le nx$$

Figure 4−16. dbuffer Array in Memory at Time j




```
ushort oflag = fir (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nx,
ushort nh)
```

Salida:

- Oflag: Overflow erro flag.

Entrada:

- x: puntero hacia el vector de entrada.
- h: puntero hacia el vector de coeficientes.
- r: puntero hacia el vector de salida.
- dbuffer: puntero hacia el vector buffer delay. (N+2)
- nx: número de muestras de entrada.
- nh: numero de coeficientes del filtro.

Fase2 (DSP)
-Solución final

```
// MODO 2: LLANO

        else if ( Step == 2 )
        {
            if ( Step != LastStep )
            {
                oled_display_message("TDS  SERRA/FERRANDO", "2. LLANO           ");
                LastStep = Step;
                for(j = 0; j < N; j++) Parametric[j] = Llano[j];          // Convertimos coefs a INT16 guardandolo en una variable.
            }
            x = mono_input;                                              // Muestra que entrara en el filtro.
            fir(&x, Parametric, &y, dbuffer, 1, N);                      // Aplicamos filtro FIR.
            right_output = y;                                            // Sacamos por la salida el resultado de la conv.
            left_output =  right_output;                                 // Salida izquierda lo mismo que derecha. MONO

        }
```

Conclusiones

FIN

Muchas Gracias

¿Alguna pregunta?