ENGG*3390- Signal Processing

Lab 4

Bilal Ayyache- 0988616

Palak Sood - 0986802

Kathlene Titus – 0954584

November 22nd, 2018

Introduction

The purpose of lab 4 is to design and evaluate the performance of several FIR and IIR digital filters. Digital filters are comprised of a system that performs mathematical operations on a discrete time signal. The main objective of a filter is to perform a computation on an input and create a new output sequence. The general form of a constant-coefficient difference equation for a discrete time linear time invariant system can be described as:

$$a_0 y[n] + a_1 y[n-1] + \cdots + a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]$$

*Equation 1: Constant-Coefficient difference equation (*where x[n] is the input, y[n] is the output).

The following system can enhance or reduce aspects of a signal. A digital filter consists of an analog to digital converter, digital to analog converter, and a microprocessor. In lab 4, a Texas instrument TMS320C5505 eZdsp USB Stick was used as a signal processor in which it was programmed to perform digital filtering through discrete time system. This discrete time system includes Finite Impulse Response (FIR), Infinite impulse response (IIR). In equation 2 below, a and b represent the systems coefficients. In FIR filters the "a" coefficient is zero while in IIR filters, the "a" coefficient is a nonzero value.

$$y[n] = \frac{1}{a_0}\left(-a_1 y[n-1] - \cdots - a_N y[n-N] + b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]\right)$$

Equation 2: Constant-Coefficient difference equation in terms of Y[n].

Materials

During the lab, the following tools and equipment we used:

1. Function generator and oscilloscope
2. Input audio source, e.g. PC workstation, MP3 player, headphones, etc.
3. TI TMS320C5505 eZdsp USB Stick with cable, audio patch cords
4. Lab 4 project files as posted on CourseLink
5. TI Code Composer Studio (CCS) SDK

Procedure

In signal processing, a FIR filter is a filter whose impulse response is of finite duration as it settles to zero in finite time. A FIR filter has several useful properties such as the fact that it does not require any feedback, is inherently stable since the output is a sum of a finite number, can easily be designed to be linear phase, uses a fractional arithmetic and very simple to implement. One disadvantage is that is requires a lot of memory due to a lot of calculations performed during the filtering process. Referring to Equation 2, it is noted that the "a" coefficient equals to 0 since the impulse response has a finite duration. The Basic block diagram of a FIR filter is described below:

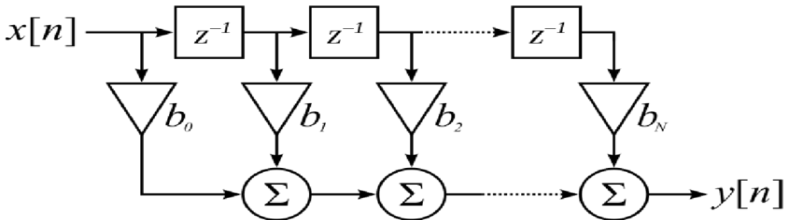


Figure 1: Basic Block Diagram of FIR Filter

IIR filters are digital filters with an infinite impulse response due to recursive calculations. Unlike the FIR filters, IIR filters have feedback. IIR filters are the most efficient type of filter as they are very easy to implement in digital signal processing due to the low amount of processing required to perform the filtering. In summary, IIR filters require less memory. Cascade of second order IIR filters are used to reduce the quantization error of higher order due to lack of precision with smaller numbers. The following block diagram describes a basic block diagram of an IIR Filter:

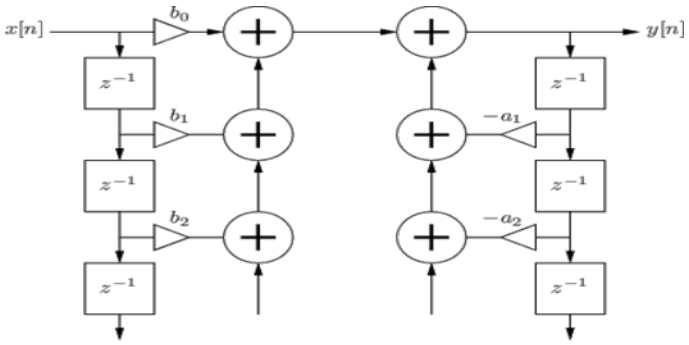


Figure 2: Basic Block Diagram of an IIR Filter

The main objective of lab 4 part 1 is to design a rectangular, a hamming, and a Blackman FIR Filter using the Window function. Part 2 involved designing A Butterworth, a Chebyl, and a Bessel IIR filter. A window function is a mathematical function that zeroes values outside a chosen interval to ease the signal analysis process. It zeroes the interval chosen (window) by reducing the amplitude of the discontinuities at the boundaries of a finite sequence (Note that coefficients described in equation 2 should be zero as this is a finite impulse response).

$$H(j\omega) = \begin{cases} 1 & |\omega| < \omega_c \\ 0 & \omega_c < |\omega| < \pi \end{cases}$$

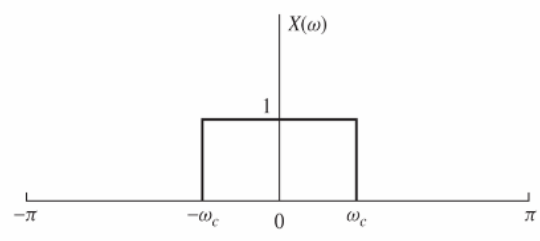


Figure 3: Window function

$$h(n) = \frac{\sin \omega_c n}{\pi n}$$

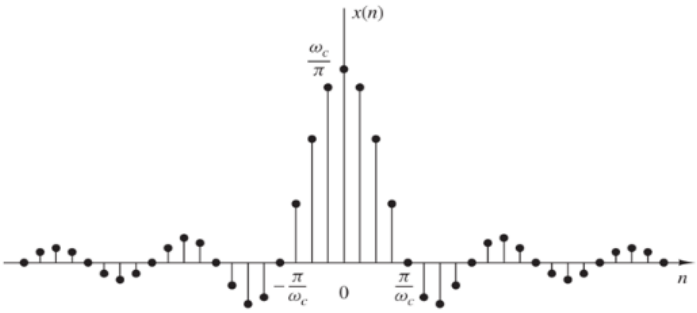


Figure 4: The figure above shows the windowing process in action

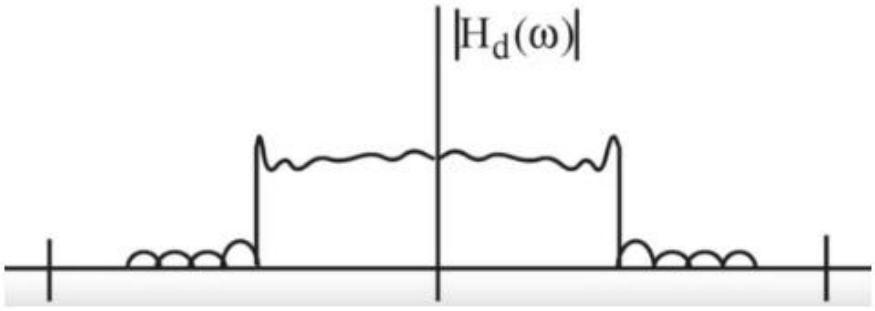


Figure 5: Windowing process

There are 3 different types of window functions explored in lab 4. Windows include rectangular window, hamming window, and Blackman window. The general equation for radian frequency is described in figure 6:

Common windowing functions W[n] for 0 < n < M:

- Rectangular: w[n] = 1
- Hamming: w[n] = 0.54 − 0.46 cos (2πn/(M − 1))
- Blackman: w[n] = 0.42 − 0.5 cos (2πn/(M − 1)) + 0.08 sin (4πn/(M − 1))

$$w[n] = w_0\left(n - \frac{N-1}{2}\right),\ 0 \le n \le N-1.$$

$$w[n] = \begin{cases} 1 & 0 \le n \le M \\ 0 & \text{otherwise} \end{cases}$$

Figure 6: Frequency Properties for FIR Windowing

$$w[n]h_d[n]$$

In figure 6, W is radian frequency, n is size, N represents the width in samples of a discrete time window function.

- Rectangular Windowing appears when a function is constant inside the interval and zero elsewhere giving the rectangular shape and is not equivalent to any window.
- Hamming appear when better frequency resolutions are desired as the window is optimized to minimize and maximize side lobes. This results in wide peaks and low side lobes. Best for noise measurements of signals.
- Blackman is similar to hamming with wide peaks but has good side lobe compression.

For designing IIR filters a different method is used. Network synthesis is used to obtain component values of a filter from a rational function that represents a transfer function of the IIR filter desired. In lab 4 we look at 2 different types of network synthesis. This includes low pass Bessel filter described by figure 6 and Butterworth filter described in figure 7 below:

$$H(s) = \frac{\theta_n(0)}{\theta_n(s/\omega_0)}$$

Figure 7: Bessel Filter

$$H(s) = \frac{G_0}{B_n(a)}$$

Figure 8: Butterworth Filter

Where Θn is the reverse bessel polynomial, ωo is frequency, G is gain, Bn is normalized butterworth polynmial, and a is s/ωc. ωc is cuttoff frequency which is assumed to be 1.

In lab 4, we look at 3 different IIR filter:

- Bessel filter gives a maximally linear phase response preserving ewace shape of filter signals in passband.
- Butterworth filter is used to give a flat frequency response in the passbad region of the signal, which makes analysis easier.
- The chebyshev filter has a steep roll off and more ripple in passband. This is explained more in the results section as we review the graphs created.

Lab 4 consists of two parts, IIR and FIR filtering. Filtering was completed using a switch statement in C that performed functions which was accessed using the header files IIR_Filter.h and FIR_filter_asm.h.

```
24
25
26 const Int16 IIR_4th_OrderButter[12] = {12  , 13,  12,
27                                        32767  , -24306 , 18301,
28                                        32767,   32767,   32767,
29                                        32767  , -27915 , 25885};
30
31
32 const Int16 IIR_4th_OrderBessel[12] = {11  , 12 , 11,
33                                        32767,   -24534 , 18506,
34                                        32767,   32710,   32653,
35                                        32767  , -25818 , 21803};
36
```

Figure 9: IIR_Filter.h

The switch statement was similar to the one used in lab 3 (Code can be found in the appendix, Figure 44).

Coefficients in header files was calculated using the following Matlab Programs:

IIR_bessel.m

```
clear all;
close all;

wp = 2400/24000;
fc = 2400*2*pi;

[b1 a1] = besself(4,fc) %Analog

figure(1)
freqs(b1, a1)

%Converting to digital
[b1d a1d] = bilinear (b1,a1,48000)
figure(2)
freqz(b1d, a1d)

sos = tf2sos(b1d,a1d)
```

Figure 10: MATLAB code for bessel filter

IIR_Butter.m

```
fc=2400;
fs= 48000;
[b1 a1] = butter(4,2*pi*fc, 's');
[b2 a2] = butter(8,2*pi*fc, 's');
figure(1)
freqs(b1, a1)
figure(2)
freqs(b2,a2)
[b1d a1d] = bilinear(b1,a1,fs)
figure(3)
freqz(b1d, a1d)
sos= tf2sos(b1d, a1d)
```

Figure 11: MATLAB code for butter filter

The figures above describe the process of designing an IIR Filtering. The process of designing FIR Filter was very similar as the matlab code in Figures 17 and 18 in the appendix. Figures 30, 34 and 38 was used to create coefficients of filter which can be found in appendix. Numbers/coefficients from the MATLAB code wer multiplied by 2^15 and then subtrated by 1.

The code in Figures 42, 43 and 44 was first executed as an audio source was connected (input). The frequency response plot for FIR filter order 4 is shown in Figures 32, 35 and 40. The function generator and the oscilloscope were later connected to characterize the frequency response of only the FIR filter with an M (N in code) value of 4 and 8. The ratio of the output to input peak to peak voltages was compared for the following frequencies: 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000. These values were obtained using the freqz plot in MATLAB.

The code in Figures 43 and 44 includes an IIR Filters with an N order of 4. The frequency response plot for IIR filter order 4 is shown in Figures 32, 35 and 40. The code was first executed as an audio source was connected (input). The function generator and the oscilloscope were later connected to characterize the frequency response of only the IIR filter with an N value of 4. The ratio of the output to input peak to peak voltages was compared for the following frequencies: 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000. These values were obtained using the freqz plot in MATLAB.

Results/ Discussion

Frequency (Hz)	Input (V)	Output
1000	4 V	5.62 V
2000	4 V	4.60 V
3000	4 V	2.30 V
4000	4 V	1 V
5000	4 V	1.2 V
6000	4 V	780 mV
7000	4 V	860 mV
8000	4 V	600 mV
9000	4 V	800 mV
10000	4 V	560 mV
11000	4 V	500 mV
12000	4 V	480 mV
13000	4 V	560 mV
14000	4 V	500 mV
15000	4 V	360 mV

Table 1: Data results for Rectangular Window

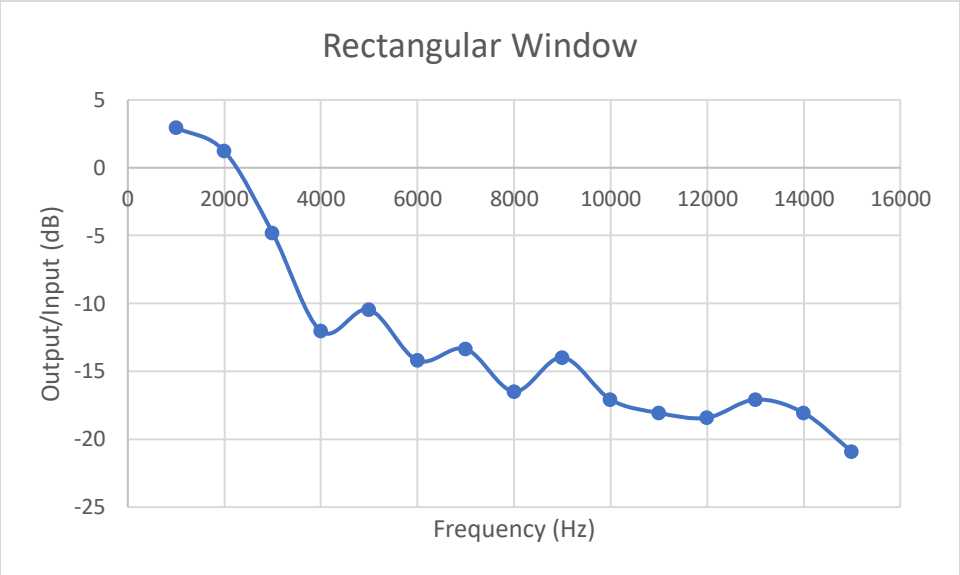


Figure 12: Graph for Rectangular window

Frequency (Hz)	Input (V)	Output (V)
1000	4 V	11.6 V
2000	4 V	4.8 V

3000	4 V	5.6 V
4000	4 V	4.4 V
5000	4 V	4 V
6000	4 V	4.4 V
7000	4 V	3.6 V
8000	4 V	4 V
9000	4 V	3.2 V
10000	4 V	3.6 V

Table 2: Data results for Hamming Window

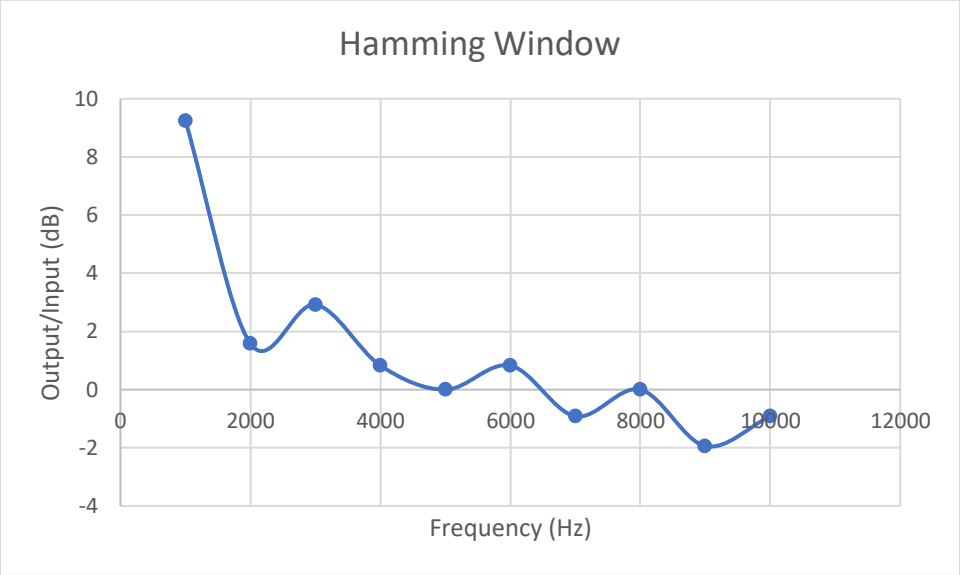


Figure 13: Graph for Hamming window

Frequency (Hz)	Input (V)	Output (V)
1000	4 V	11.4 V
2000	4 V	5.8 V
3000	4 V	5.2 V
4000	4 V	4.6 V
5000	4 V	4.4 V
6000	4 V	4.6 V
7000	4 V	3.6 V
8000	4 V	4.4 V
9000	4 V	3.4 V
10000	4 V	4 V
11000	4 V	3.6 V
12000	4 V	3.8 V
13000	4 V	3.8 V
14000	4 V	3.6 V
15000	4 V	3.6 V

Table 3: Data results for Blackman window

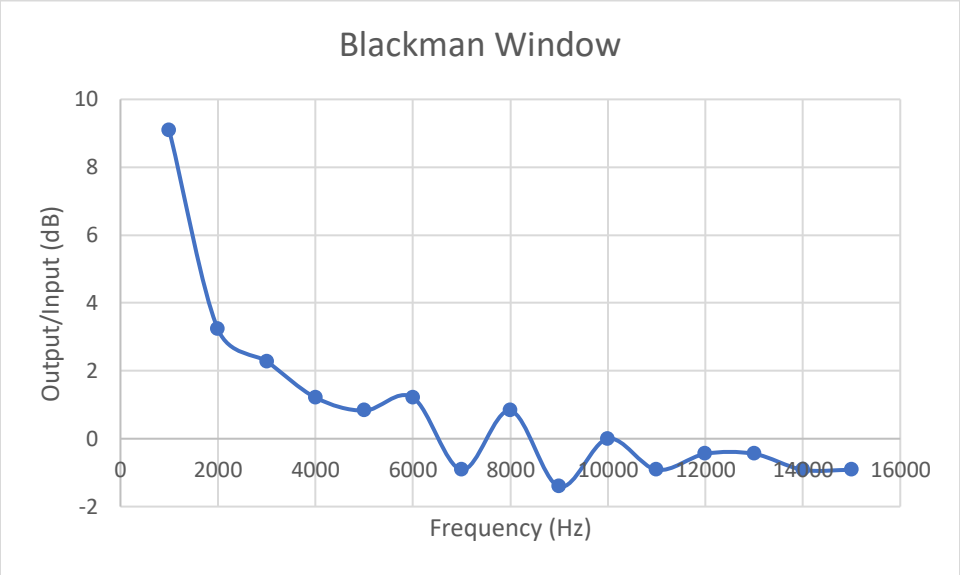


Figure 14: Graph for Blackmam window

Result analysis

FIR Type	FIR Size	Comment
Rectangular	M = 8	Quieter, Bass more emphasized
	M = 32	Less emphasis on bass
Blackman	M = 8	Quieter, treble muffled
	M = 32	Muffled

The first part of the lab involved designing a rectangular FIR filter and a Blackman FIR Filter. The coefficients were calculated using MATLAB code below:

FIRFilter.m

```
clc;
close all;
M1 =8; M2 = 32; M3 = 128;
f = 2400/48000
rect1 = 2*f* sinc(2*f*(-M1/2+0.5:M1/2-0.5));
rect2 = 2*f* sinc(2*f*(-M2/2+0.5:M2/2-0.5));
rect3 = 2*f* sinc(2*f*(-M3/2+0.5:M3/2-0.5));
figure(1)
freqz(rect1)
figure(2)
freqz(rect2)
figure(3)
freqz(rect3)
```

```
B1= blackman(M1);
B2= blackman(M2);
B3= blackman(M3);
Black1= 2*f*sinc(2*f*B1);
Black2= 2*f*sinc(2*f*B2);
Black3= 2*f*sinc(2*f*B3);
figure(100)
freqz(Black1)
figure(200)
freqz(Black2)
figure(300)
freqz(Black3)
```

The following equation was used to define the Blackman window of length N:

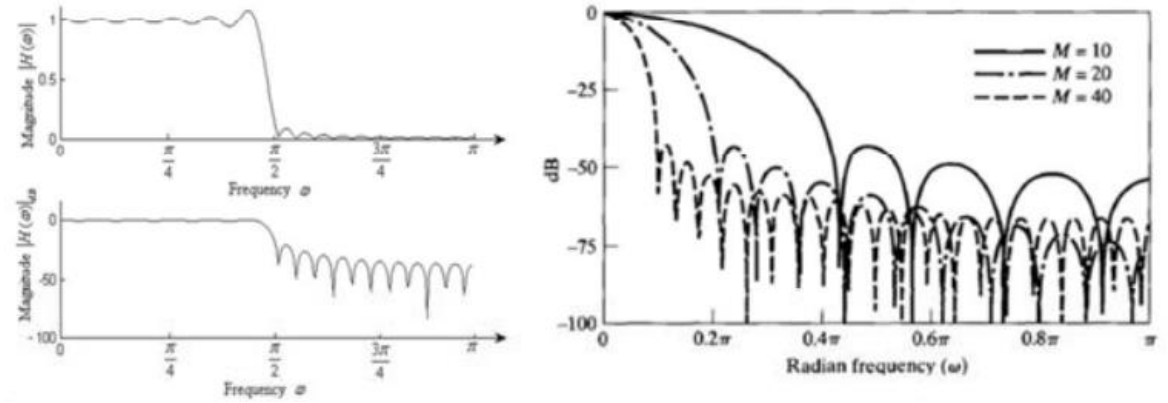
$$w(n) = 0.42 - 0.5\cos\frac{2\pi n}{N-1} + 0.08\cos\frac{4\pi n}{N-1}, \quad 0 \le n \le M-1$$

The following equation was used to define the rectangular window of length N:

$$w[n] = 0.54 - 0.46\cos(2\pi n/(M-1))$$

where M is N/2 for N even and (N + 1)/2 for N odd.

A sin wave with varying frequency was applied to the filters designed. It was observed that a decrease in amplitude of the signal occors for both filters. The recrangular filter is a low pass filter this allows the bass to be more emphasized. On the other hand, it is observed that the blackman filter has less sudden changes. This allows the audio to be more muffled. When 6Khz was applied, it was observed that both filters had an almost flat output signal. This is when the transition phase occurs. After 6Khz the signal becomes a decreased amplitude version of the input signal. At 24Khz, the output of the signal becomes approximatelty 0.



For the rectangular FIR filter, it is observed that the width of the transition is not sharp. The width of the transition depends on the width of the main lobe of the window. Ripples in the passband/ stopband are proportional to the peaks of side lobes of the window. The transition width can be controlled using the size of the window and the shape of the window. This means that windows with a fixed size that have different shapes can have different main lobe width.

By comparing results with theoretical frequency response plots from the prelab, it can be observed that the magnitude of the output signal decreases as frequency increases. This can be seen in figure 14 and table 3.

IIR Bessel

Frequency (Hz)	Input (V)	Output (V)
1000	4 V	8 V
1100	4 V	7.6 V
1200	4 V	7.4 V
1300	4 V	7.2 V
1400	4 V	6.8 V
1500	4 V	6.4 V
1600	4 V	6.4 V
1700	4 V	6.2 V
1800	4 V	6 V
1900	4 V	6 V
2000	4V	5.6 V
2300	4V	5.2 V
2500	4V	4.8 V
2800	4 V	4.4 V
3000	4 V	4 V
3500	4V	3.6 V
4000	4V	3.2V
4500	4V	3.2V
5000	4V	3.2 V

Table 4: Data results for IIR Bessel

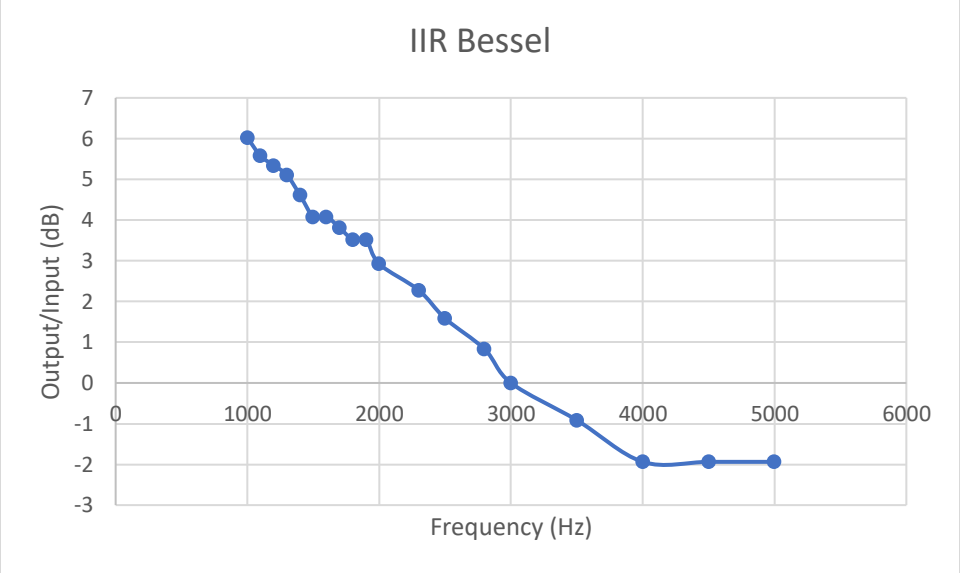


Figure 15: Graph for IIR Bessel

Frequency (Hz)	Input (V)	Output (V)
1000	4 V	8.4 V
1100	4 V	8 V
1200	4 V	8 V
1300	4 V	8 V
1400	4 V	8 V
1500	4 V	8 V
1600	4 V	8 V
1700	4 V	8V
1800	4 V	8V
1900	4 V	8V
2000	4V	8V
2100	4V	7.8 V
2200	4V	7.2 V
2300	4 V	6.8V
2400	4 V	6.6 V
2500	4V	6.4 V
2600	4V	6 V
2700	4V	5.8 V
2800	4V	5.2 V
2900	4V	4.8 V
3000	4V	4.4 V
3500	4V	4 V
4000	4V	3.6 V
4500	4V	3.6V
5000	4V	3.2V
6000	4V	3.2 V

Table 5: Data results for IIR Butter

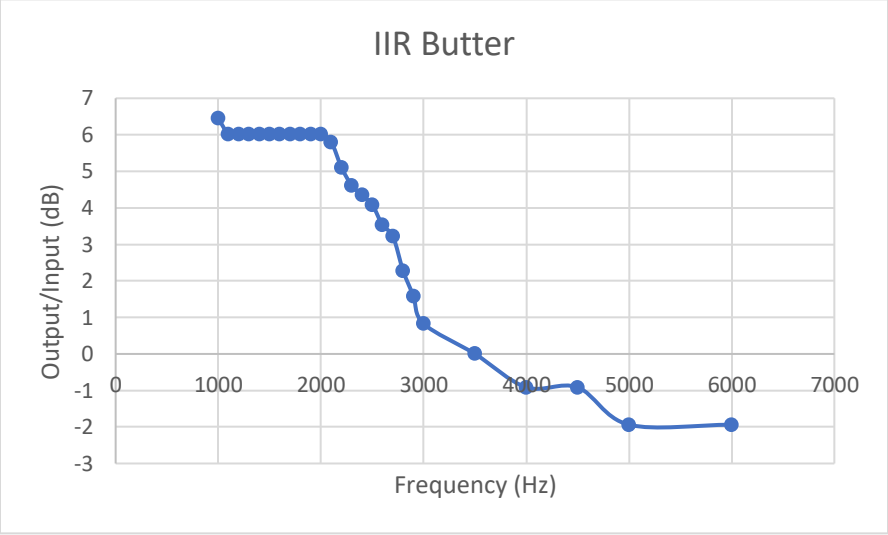


Figure 16: Graph for IIR Butter

IIR Type	FIR Size	Comment
Butterworth	M =4	Buzzing (High frequency) sound on top of music
Bessel	M = 4	Beep (Alternating frequency) sound on top of music

The second part of the lab involved designing a Butterworth IIR filter and a Bessel IIR Filter. The coefficients were calculated using MATLAB code below:

IIR_bessel.m

```
clear all;
close all;

wp = 2400/24000;
fc = 2400*2*pi;

[b1 a1] = besself(4,fc) %Analog

figure(1)
freqs(b1, a1)

%Converting to digital
[b1d a1d] = bilinear (b1,a1,48000)
figure(2)
freqz(b1d, a1d)

sos = tf2sos(b1d,a1d)
```

IIR_Butter.m

```
fc=2400;
fs= 48000;
[b1 a1] = butter(4,2*pi*fc, 's');
[b2 a2] = butter(8,2*pi*fc, 's');
figure(1)
freqs(b1, a1)
figure(2)
freqs(b2,a2)
[b1d a1d] = bilinear(b1,a1,fs)
figure(3)
freqz(b1d, a1d)
sos= tf2sos(b1d, a1d)
```

Fourth order Bessel and butterworth IIR Filters were converted to a series of cascaded second order systems using the matlab function tf2sos. It is observed that the filter coefficients were greater than 1. Coefficients were normalized using the MATLAB ceiling function.

For the IIR filters, it is observed that the width of the transition is sharp. For the IIR Filters Ripples in the passband/ stopband did not occur.

By comparing results with theoretical frequency response plots from the prelab, it can be observed that the magnitude of the output signal decreases as frequency increases. This can be seen in figure 15 and table 4.

Conclusion

Though this lab, the FIR and IIR filters were designed using windowing and Network synthesis. The observations recorded in the results section verify that the described filters were created successfully. When applying a rectangular window function on the FIR filter the results show a box like figure. Results also show a decrease in amplitude. This can be seen in Figures 20-22. In summary we learn that as the frequency increase the filter length increase resulting in a more muffled output. This is due to the increase in length as the filter is applied resulting in more dramatic changes in the input (Audio). For the IIR Filters designed, we see that the second order was still clear but the 4^{th} order resulted in a more muffled output. We learn that as IIR increase in order there's is a greater chance of quantization error and poor precision. The higher the order the less precise an output becomes, and that is why a second order cascade IIR is always favored. Based on the results from this experiment it is safe to conclude that FIR filters are stronger (greater power means more flexibility and ability to finely adjust the response of your active loudspeaker) than IIR filters but IIR filters are more efficient (requires less processing power and less work to setup). This conclusion also matches the conclusion from analyzing the FIR and IIR in lab 3. These observations are crucial to understanding the system behind different types of digital filters. Understanding the system is very important in design for applications such as sound filtering, communications, and digital conversions of audio files. In lab 4, error is bound to occur. Errors that could have occurred were possibly due to human error such as audio selection, or parallax errors such as reading the oscilloscope incorrectly while obtaining the peak to peak voltages. Another possible error could have been due to the equipment used during the lab.

Appendix

FIRFilter.m

```
clc;
close all;
M1 =8; M2 = 32; M3 = 128;
f = 2400/48000
rect1 = 2*f* sinc(2*f*(-M1/2+0.5:M1/2-0.5));
rect2 = 2*f* sinc(2*f*(-M2/2+0.5:M2/2-0.5));
rect3 = 2*f* sinc(2*f*(-M3/2+0.5:M3/2-0.5));
figure(1)
freqz(rect1)
figure(2)
freqz(rect2)
figure(3)
freqz(rect3)
H1= hamming(M1);
H2= hamming(M2);
H3= hamming(M3);
ham1= 2*f*sinc(2*f*H1);
ham2= 2*f*sinc(2*f*H2);
ham3= 2*f*sinc(2*f*H3);
figure(10)
freqz(ham1)
figure(20)
freqz(ham2)
figure(30)
freqz(ham3)
```

Figure 17: Part 1 code for FIR filter

```
B1= blackman(M1);
B2= blackman(M2);
B3= blackman(M3);
Black1= 2*f*sinc(2*f*B1);
Black2= 2*f*sinc(2*f*B2);
Black3= 2*f*sinc(2*f*B3);
figure(100)
freqz(Black1)
figure(200)
freqz(Black2)
figure(300)
freqz(Black3)
```

Figure 18: Part 2 code for FIR filter

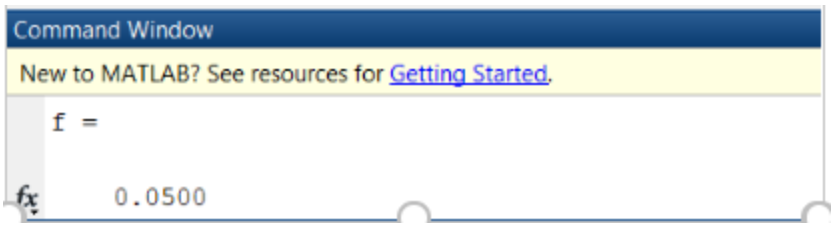


Figure 19: Results for FIR filter

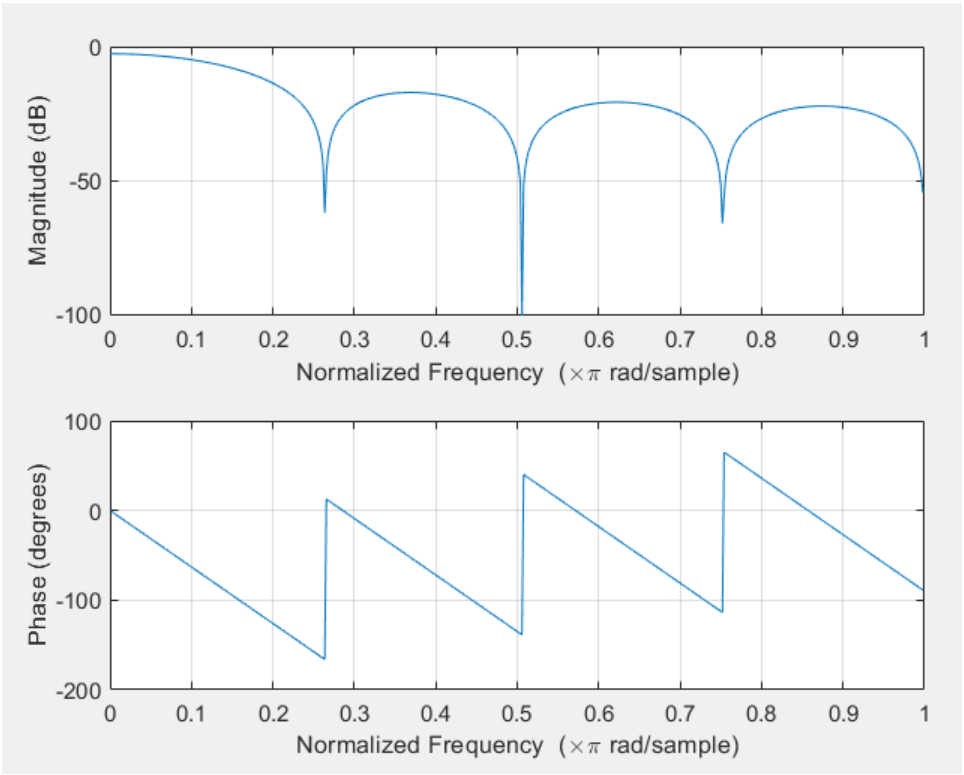


Figure 20: Frequency response plot for rectangular window when M = 8.

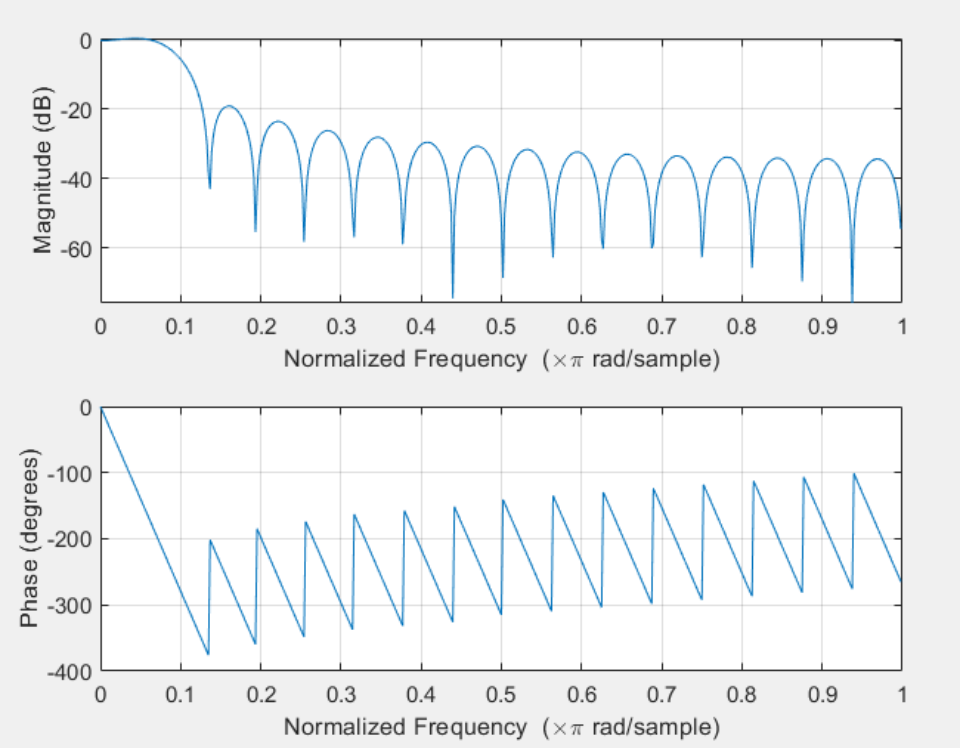


Figure 21: Frequency response plot for rectangular window when M = 32.

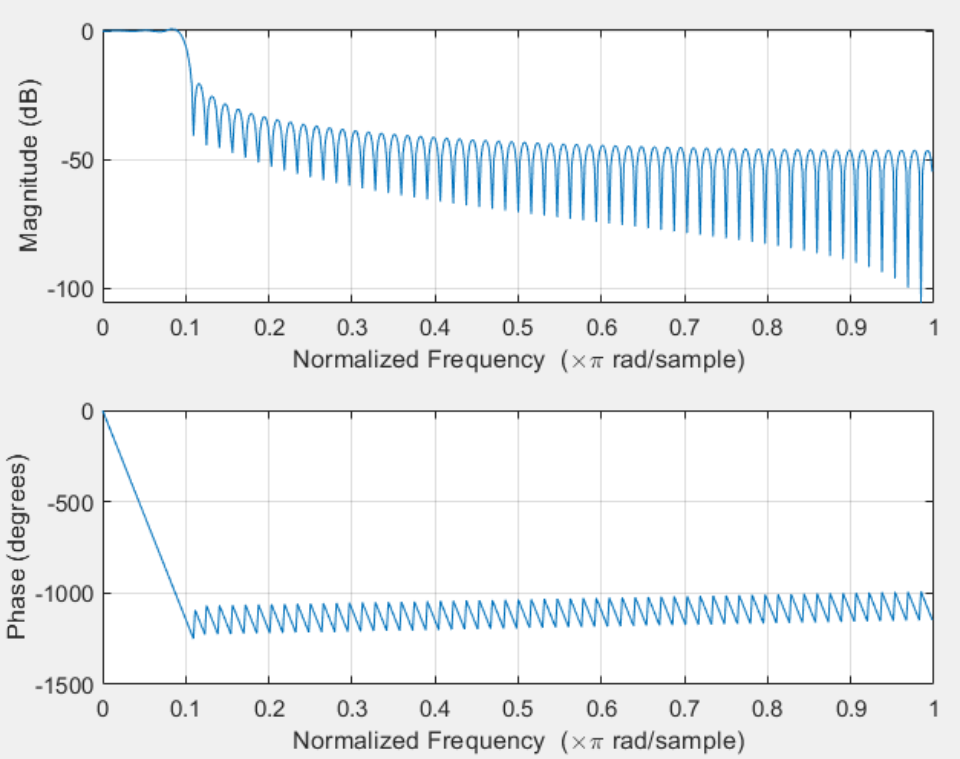


Figure 23: Frequency response plot for rectangular window when M = 128.

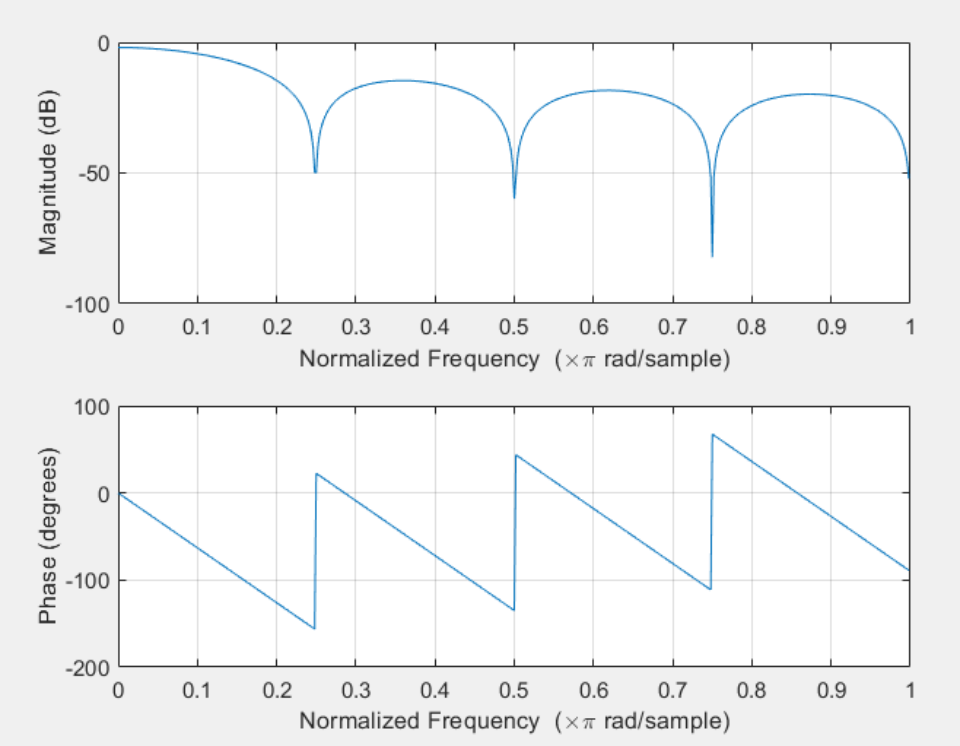


Figure 24: Frequency response plot for hamming window when M = 8.

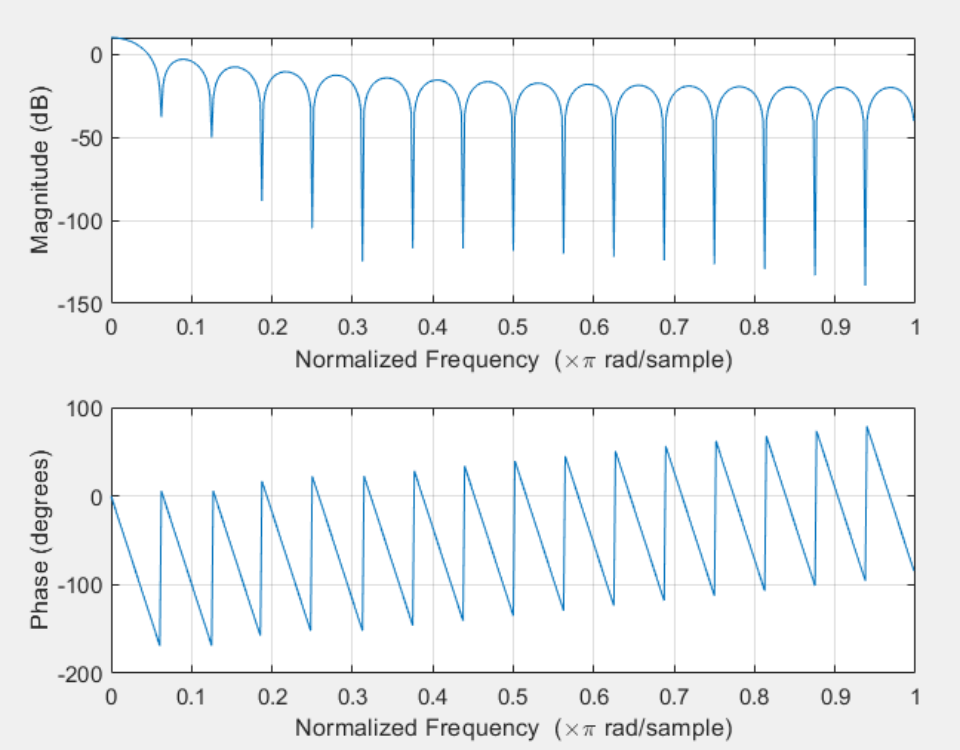


Figure 25: Frequency response plot for hamming window when M = 32.

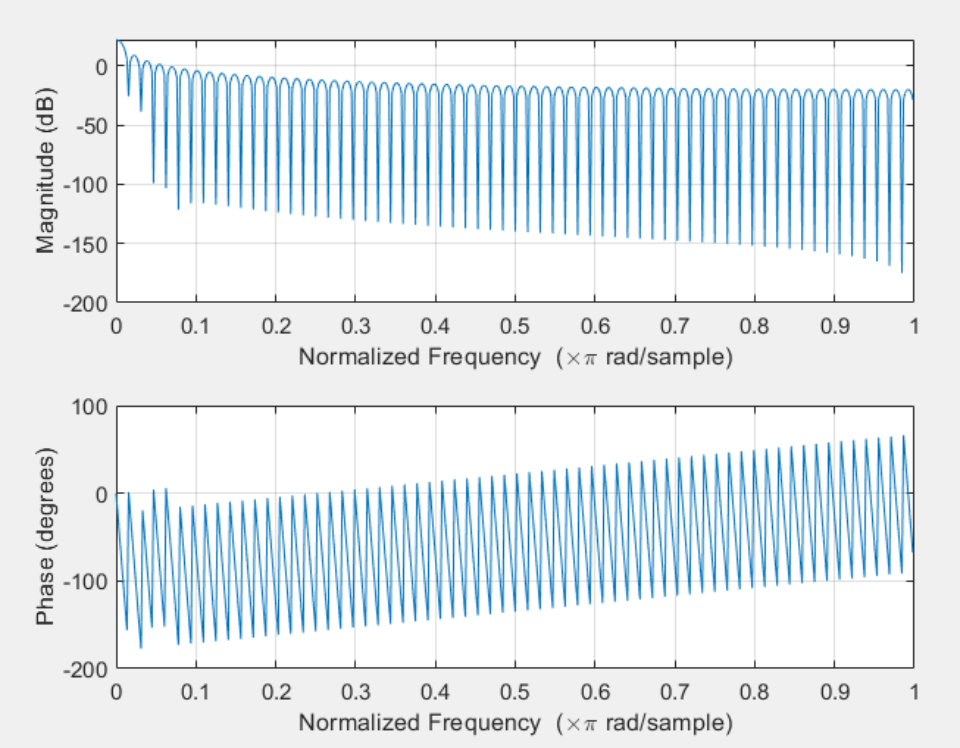


Figure 26: Frequency response plot for hamming window when M = 128.

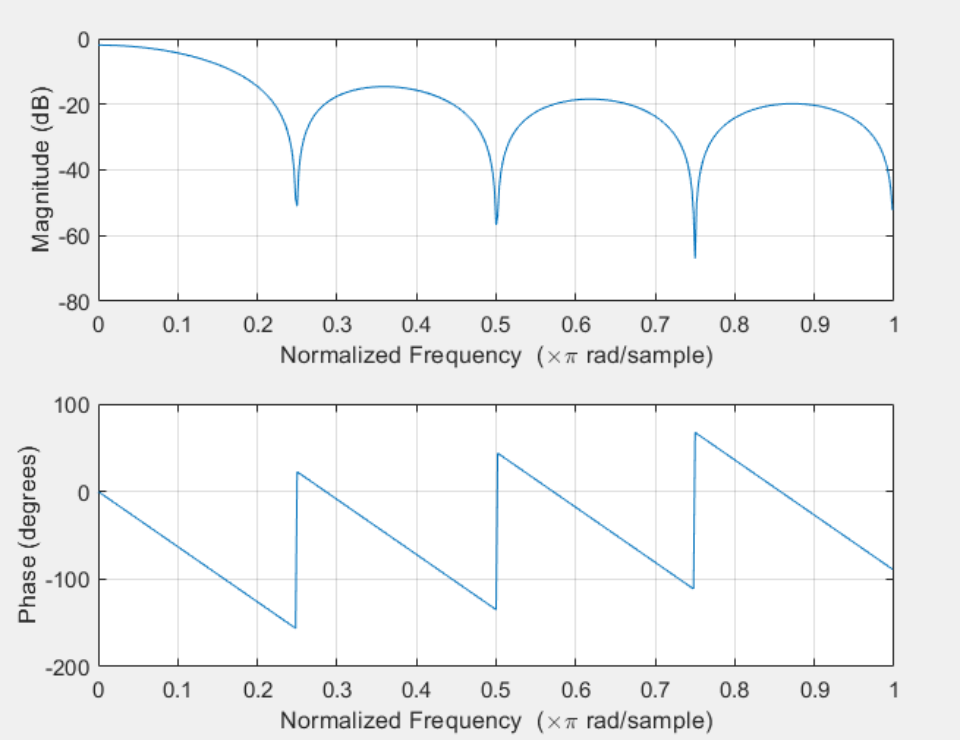


Figure 27: Frequency response plot for blackman window when M = 8.

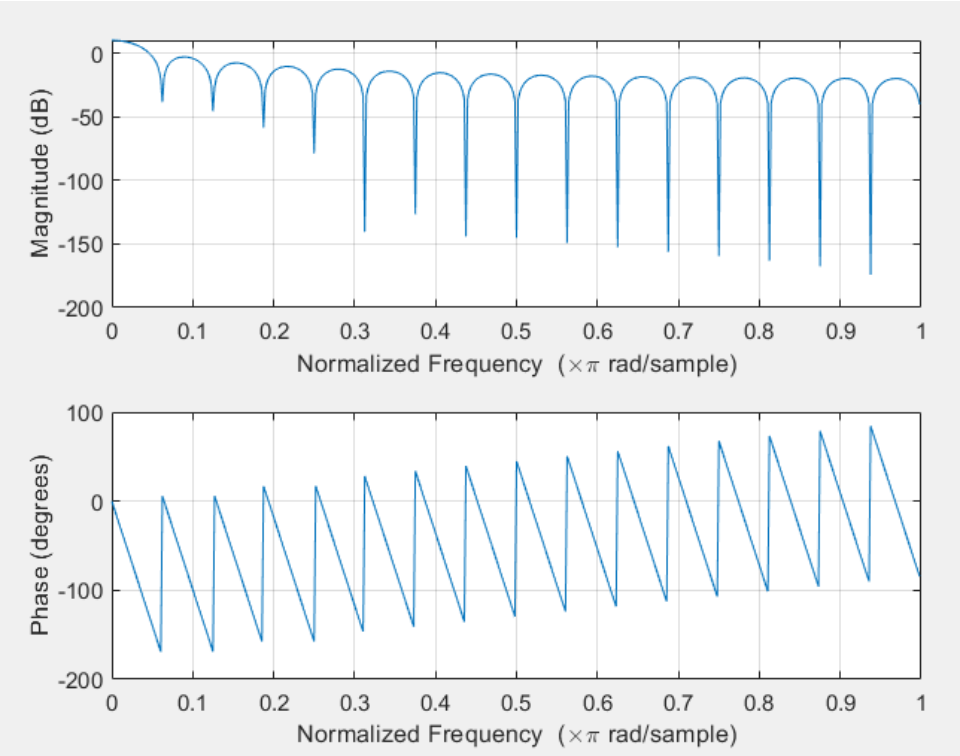


Figure 28: Frequency response plot for blackman window when M = 32.

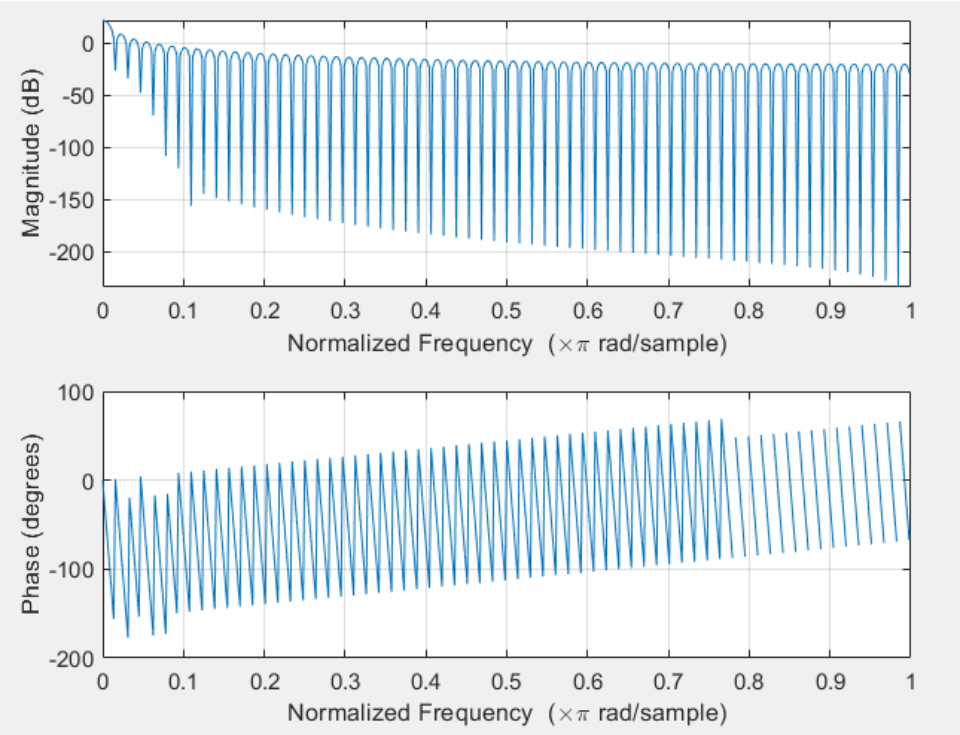


Figure 29: Frequency response plot for blackman window when M = 128.

IIR_bessel.m

```
clear all;
close all;

wp = 2400/24000;
fc = 2400*2*pi;

[b1 a1] = besself(4,fc) %Analog

figure(1)
freqs(b1, a1)

%Converting to digital
[b1d a1d] = bilinear (b1,a1,48000)
figure(2)
freqz(b1d, a1d)

sos = tf2sos(b1d,a1d)
```

Figure 30: Part 1 code for IIR Bessel filter

```
>> IIR_bessel

b1 =

   1.0e+16 *

         0         0         0         0    5.1709


a1 =

   1.0e+16 *

    0.0000    0.0000    0.0000    0.0011    5.1709


b1d =

    0.0004    0.0015    0.0023    0.0015    0.0004


a1d =

    1.0000   -3.0732    3.5897   -1.8864    0.3758


sos =

    0.0004    0.0008    0.0004    1.0000   -1.4974    0.5648
    1.0000    2.0000    1.0000    1.0000   -1.5757    0.6654
```

Figure 31: results for IIR Bessel filter

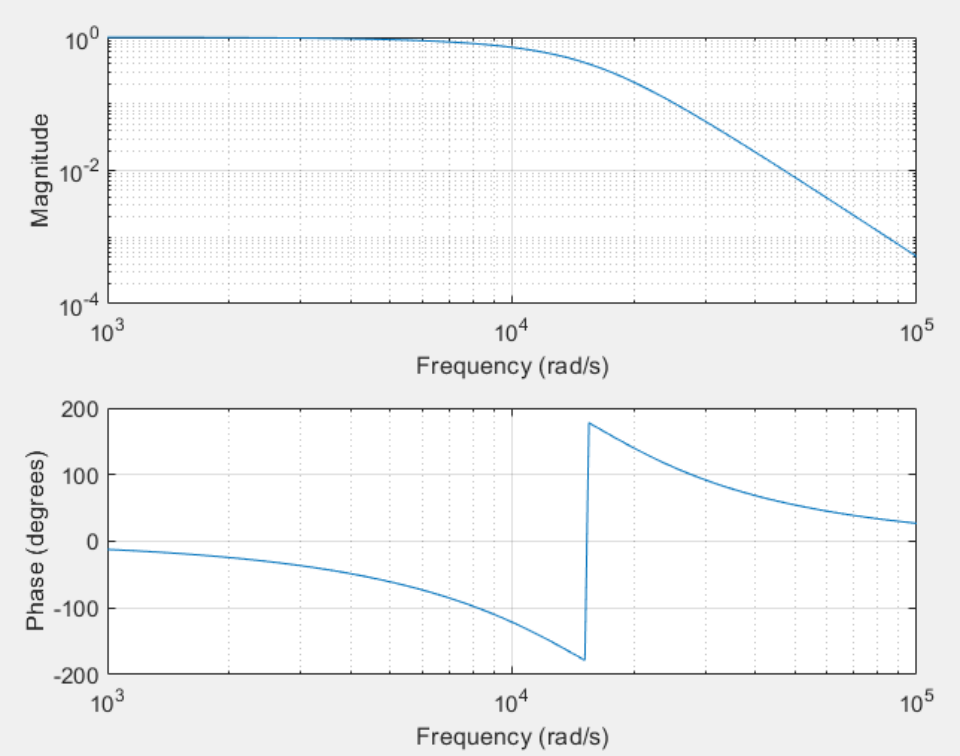


Figure 32: Frequency response plot for low pass Bessel filter of order 4.

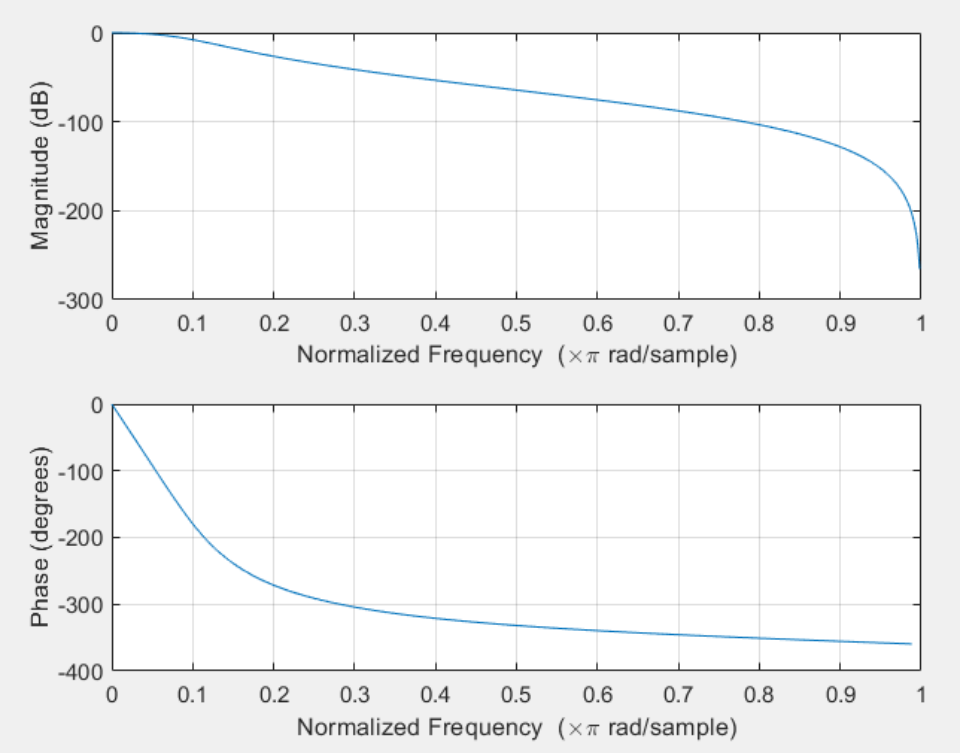


Figure 33: Frequency response plot for lowpass Bessel filter of order 8.

IIR_Butter.m

```
3 -    fc=2400;
4 -    fs= 48000;
5 -    [b1 a1] = butter(4,2*pi*fc, 's');
6 -    [b2 a2] = butter(8,2*pi*fc, 's');
7 -    figure(1)
8 -    freqs(b1, a1)
9 -    figure(2)
10 -   freqs(b2,a2)
11 -   [b1d a1d] = bilinear(b1,a1,fs)
12 -   figure(3)
13 -   freqz(b1d, a1d)
14 -   sos= tf2sos(b1d, a1d)
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> IIR_Butter

b1d =

    0.0004    0.0016    0.0024    0.0016    0.0004


a1d =

    1.0000   -3.1873    3.8760   -2.1235    0.4412


sos =

    0.0004    0.0008    0.0004    1.0000   -1.4835    0.5585
    1.0000    1.9967    0.9967    1.0000   -1.7038    0.7900
```

Figure 34: MATLAB code for IIR Butter filter

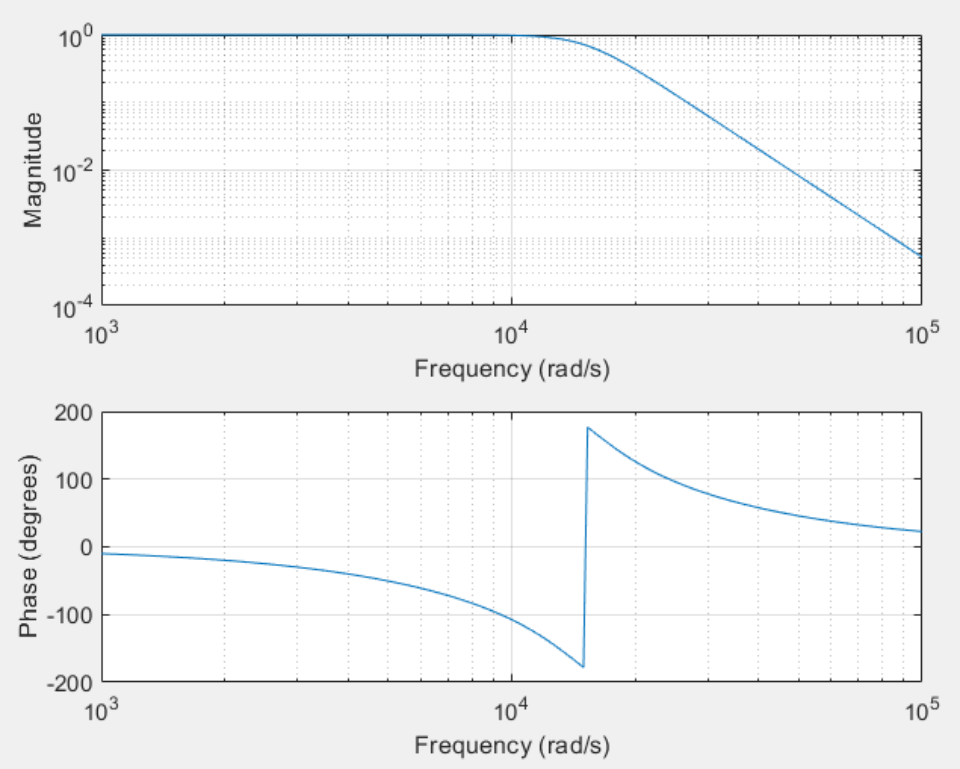


Figure 35: Frequency response plot for lowpass Butterworth filter of order 4.

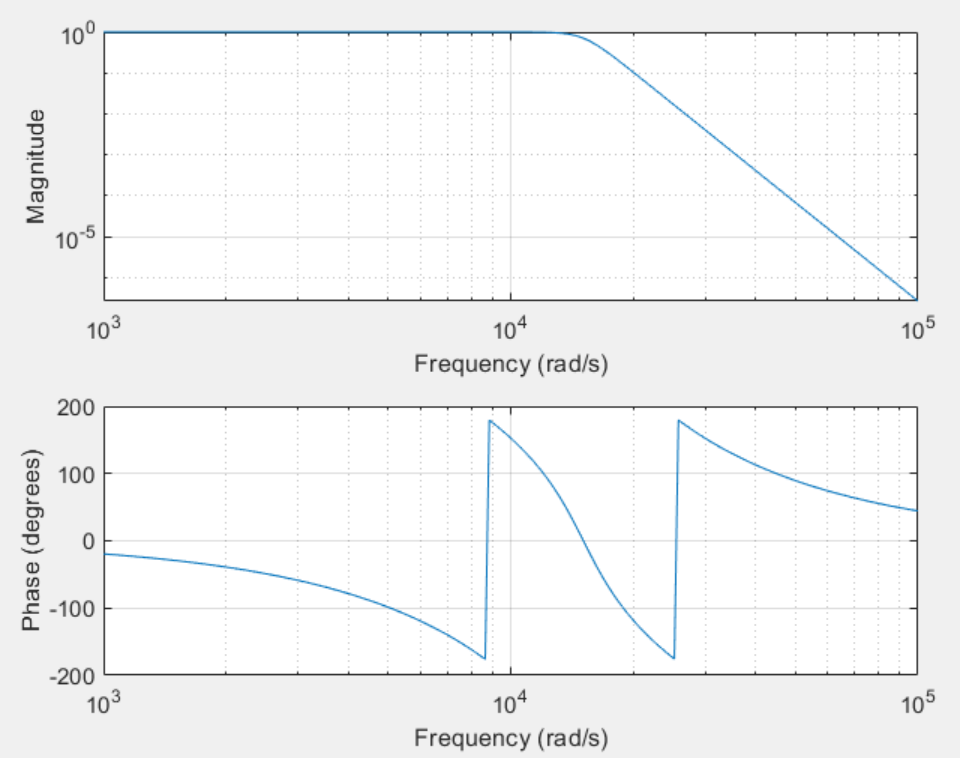


Figure 36: Frequency response plot for lowpass Butterworth filter of order 8.

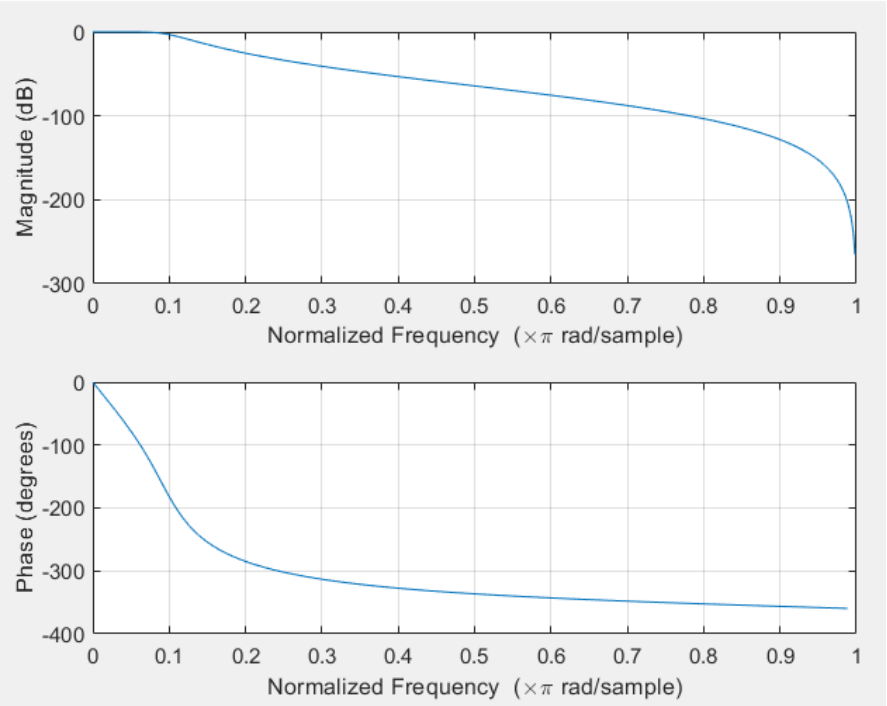


Figure 37: Frequency response plot for lowpass Butterworth filter (Bilinear)

IIR_chebyl.m

```
1 -    clear all;
2 -    close all;
3 -    wp = 2400/24000;
4 -    fc = 2400;
5 -    [b1 a1] = cheby1(4, 0.1,2*pi*fc, 's') %Analog
6 -    [b2 a2] = cheby1(8, 0.1,wp) %Digital
7 -    figure(1)
8 -    freqs(b1,a1)
9 -    figure(2)
10 -   freqz(b2,a2)
```

Figure 38: MATLAB code for IIR Chebyl filter

```
>> IIR_chebyl

b1 =

   1.0e+16 *

         0         0         0         0    4.2351

a1 =

   1.0e+16 *

    0.0000    0.0000    0.0000    0.0007    4.2841

b2 =

   1.0e-05 *

    0.0015    0.0119    0.0415    0.0830    0.1037    0.0830    0.0415    0.0119    0.0015

a2 =

    1.0000   -7.2842   23.4087  -43.3360   50.5379  -38.0104   18.0028   -4.9086    0.5898
```

Figure 39: results for IIR Chebyl filter

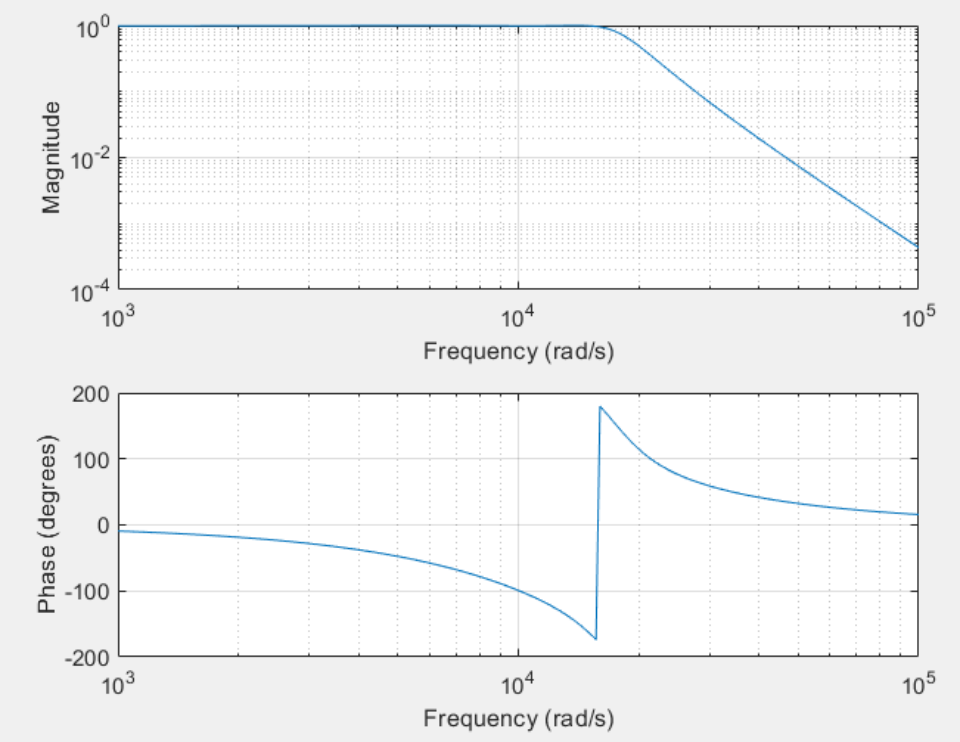


Figure 40: Frequency response plot for lowpass Cheby filter of order 4

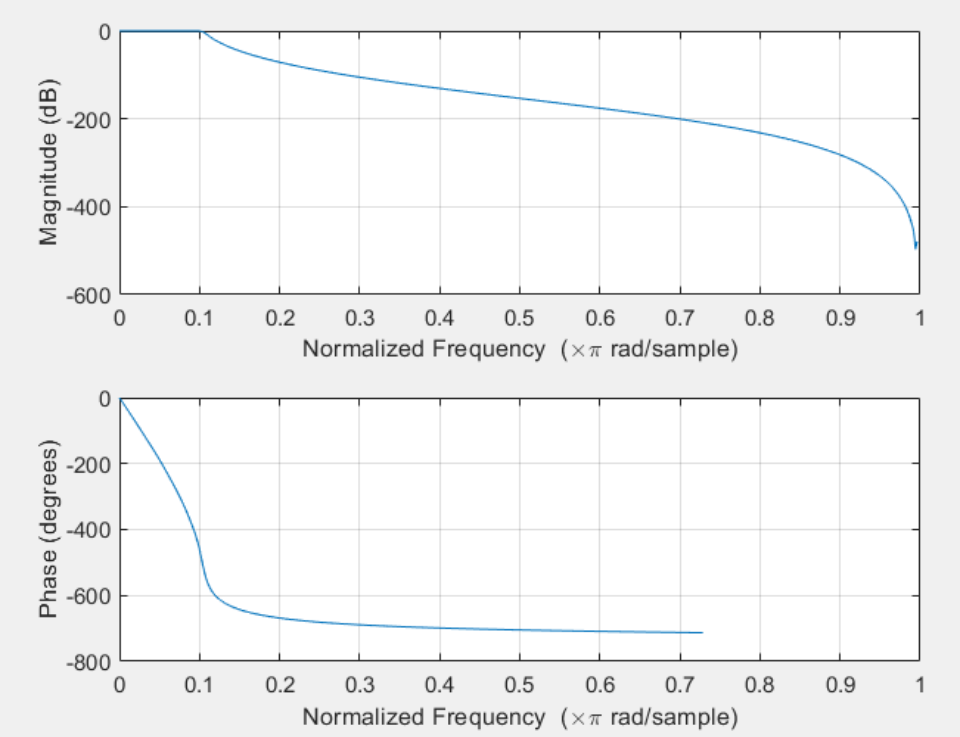


Figure 41: Frequency response plot for lowpass Cheby filter of order 8

```
const Int16 FIR_Order32[N]  = {-666, -711,    -689,    -591,    -413,    -156 ,
                               171, 556, 982, 1429,    1872 ,   2288,    2654,    2949  ,
                               3156,    3262,    3262 ,   3156 ,   2949,    2654,    2288 ,
                               1872 ,   1429 ,   982, 556, 171, -156 ,   -413  ,  -591 ,   -689 ,
                               -711 ,   -666};

const Int16 FIR_Order32Ham[N]  = {3275 ,3275,  3275, 3274   , 3273, 3271,3268,
                                  3264,
                                  3259,
                                  3253,
                                  3246,
                                  3240,
                                  3233,
                                  3228,
                                  3224,
                                  3222,
                                  3222,
                                  3224,
                                  3228,
                                  3233,
                                  3240,
                                  3246,
                                  3253,
                                  3259,
                                  3264,
                                  3268,
                                  3271,
                                  3273,
                                  3274,
                                  3275,
                                  3275,
                                  3275};

const Int16 FIR_Order32black[N]  = {3276,
                                    3276,
                                    3276,
                                    3276,
                                    3276,
                                    3275,
                                    3274,
                                    3272,
                                    3269,
                                    3264,
                                    3257,
                                    3249,
                                    3240,
                                    3232,
                                    3226,
                                    3223,
                                    3223,
                                    3226,
                                    3232,
                                    3240,
                                    3249,
                                    3257,
                                    3264,
                                    3269,
                                    3272,
                                    3274,
                                    3275,
                                    3276,
                                    3276,
                                    3276,
                                    3276,
                                    3276};
```

Figure 42: Converted coefficients for FIR

```
24
25
26 const Int16 IIR_4th_OrderButter[12] = {12   , 13,  12,
27                                         32767  , -24306 , 18301,
28                                         32767,   32767,   32767,
29                                         32767  , -27915 , 25885};
30
31
32 const Int16 IIR_4th_OrderBessel[12] = {11   , 12 , 11,
33                                         32767,   -24534 , 18506,
34                                         32767,   32710,   32653,
35                                         32767  , -25818 , 21803};
36
```

Figure 43: Converted Coefficients for IIR

```
129       switch (Step)
130         {
131         case 1:
132           left_output = mono_input;      // Directly connect inputs to outputs for reference.
133           right_output = mono_input;
134           break;
135
136         case 2:
137           left_output = FIR_filter_asm(FIR_Order5, mono_input);
138           right_output = left_output;
139           break;
140
141         case 3:
142           left_output = FIR_filter_asm(FIR_Order11, mono_input);
143           right_output = left_output;
144           break;
145
146         case 4:
147           left_output = FIR_filter_asm(FIR_Order21, mono_input);
148           right_output = left_output;
149           break;
150
151         case 5:
152           left_output = second_order_IIR_direct_form_I (IIR_2nd_Order, mono_input);
153           right_output = left_output;
154           break;
155
156         case 6:
157           left_output = fourth_order_IIR_direct_form_I(IIR_4th_Order, mono_input);
158           right_output = left_output;
159           break;
160         }
161
162
163       aic3204_codec_write(left_output, right_output);
164     }
165
166     /* Disable I2S and put codec into reset */
167     aic3204_disable();
168
169     printf( "\n***Program has Terminated***\n" );
170     SW_BREAKPOINT;
171 }
172
```

Figure 44: main.c