DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING IMPERIAL COLLEGE LONDON

EE3-19
REAL-TIME DIGITAL SIGNAL PROCESSING
LAB 5 REPORT

YASMIN BABA - 01196634
ZHI WEN SI - 01243716

2nd of March, 2019

TABLE OF CONTENTS

1. RECAP OF IIR FILTERS

The following report explores the use of infinite impulse response, or IIR, filters within digital signal processing. Compared to the FIR filter implemented in *Lab 4*, IIR filters are more efficient, requiring less computation time on the DSP and less memory to achieve the same filtering characteristic.

An IIR filter is represented by the following time-domain equation, in which the output is a weighted sum of the current and previous inputs in addition to previous outputs.

$$y(n) = \sum_{k=0}^{M} b(k)x(n-k) - \sum_{k=1}^{N} a(k)y(n-k)$$

Expanding this equation yields:

$$y(n) = b_0x(n) + b_1x(n-1) + \cdots + b_Mx(n-M) - a_1y(n-1) - a_2y(n-2) - \cdots - a_Ny(n-N)$$

And taking Z-transforms:

$$Y(z) = b_0X(z) + b_1X(z)z^{-1} + \cdots + b_MX(z)z^{-M} - a_1Y(z)z^{-1} - a_2Y(z)z^{-2} - \cdots - a_NY(z)z^{-N}$$

Therefore, the transfer function of an IIR filter is:

$$H(z) = \frac{b_0 + b_1z^{-1} + \cdots + b_Mz^{-M}}{1 + a_1z^{-1} + a_2z^{-2} + \cdots + a_Nz^{-N}}$$

An IIR filter therefore differs from the FIR filter previously implemented in *Lab 4*, in which all poles were restricted only to the origin of the z-plane. Due to this restriction, FIR filters were found to have a slow roll-off in the transition band, eating into wanted passband frequencies, as shown in *Figure 1* (effects on the magnitude response due to poles and zeros are explained in more depth in Section 3.1).

On the other hand, IIR filters can have both zeros and poles anywhere in the z-plane, in whose locations depend on the coefficients b and a respectively. With freedom to place poles closer to the unit circle, the significant slope caused by zeros can be reversed. This results in a faster roll-off in the filter's transition band, extending the useful range of the passband, as shown in *Figure 2*.

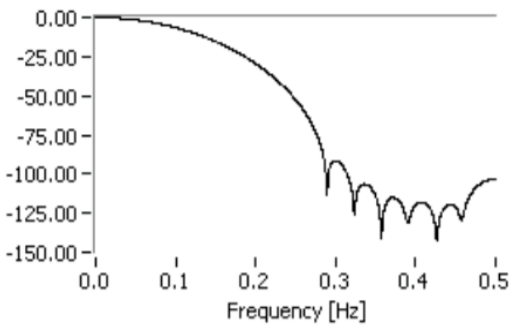

Figure 1: FIR Filter

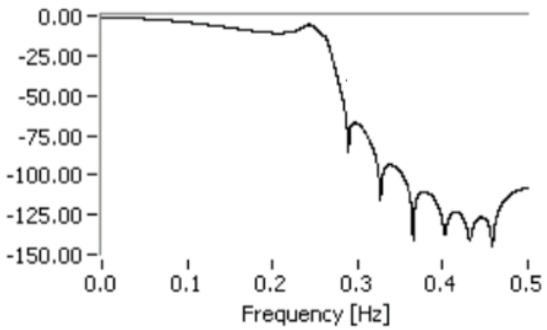

Figure 2: IIR Filter

However, poles that exist outside of the unit circle will cause instability hence must be avoided.

2. SINGLE-POLE FILTER

2.1 DERIVATION OF FILTER COEFFICIENTS

The first task is to map the following low-pass analogue filter into a discrete time version (as an IIR filter) using the Tustin transform.

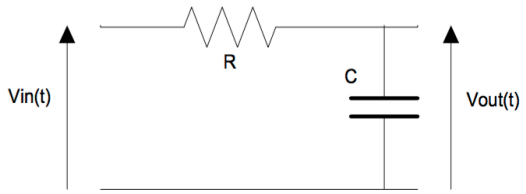

Figure 3: Low-Pass Analogue Filter

The transfer function of the circuit is found via the following calculations.

$$H(s) = \frac{V_{out}}{V_{in}} = \frac{\frac{1}{sC}}{R + \frac{1}{sC}} = \frac{1}{sRC + 1}$$

Using the Tustin transform,

$$s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right)$$

The transfer function is converted into the Z–domain as follows.

$$H(z) = \frac{V_{out}}{V_{in}} = \frac{1}{\frac{2}{T_s}\left(\frac{z-1}{z+1}\right)RC + 1} = \frac{T_s(z+1)}{2(z-1)RC + T_s(z+1)}$$

$$= \frac{T_s z + T_s}{(2RC + T_s)z + (T_s - 2RC)} = \frac{T_s + T_s z^{-1}}{(2RC + T_s) + (T_s - 2RC)z^{-1}}$$

$$= \frac{\frac{T_s}{2RC + T_s} + \frac{T_s}{2RC + T_s}z^{-1}}{1 + \frac{(T_s - 2RC)}{2RC + T_s}z^{-1}}$$

Substituting the real component values of $R = 1k\Omega$ and $C = 1\mu F$, the transfer function of the IIR filter that represents the analogue circuit is:

$$H(z) = \frac{\frac{1}{17} + \frac{1}{17}z^{-1}}{1 - \frac{15}{17}z^{-1}}$$

Finally, the time-domain difference equation is acquired by taking the inverse Z-transform:

$$y(n) = \frac{1}{17}x(n) + \frac{1}{17}x(n-1) + \frac{15}{17}y(n-1)$$

where the coefficients are,

$$b_0 = b_1 = \frac{1}{17} \;\&\; a_1 = -\frac{15}{17}$$

It is important to note that the Tustin transform is a type of bilinear transform. It maps positions on the $j\omega$ axis in the s-plane onto positions on the unit circle in the z-plane, and thus, is a non-linear transformation. This results in a mismatch between the corresponding frequencies in the continuous-time domain and the discrete-time domain, represented by the following relationship:

$$\omega = \frac{2}{T_s}\tan\left(\frac{\omega_p T_s}{2}\right)$$

where ω_p is the frequency in the discrete domain and ω is the frequency in the continuous domain. Note that at low frequencies $tan\left(\frac{\omega_p T_s}{2}\right) \approx \frac{\omega_p T_s}{2}$ and the mismatch between the domains is insignificant.

In some cases, this mismatch may result in a significant difference between the corner frequency in the continuous-time domain and the discrete-time domain. It can be corrected by calculating the mismatch and adding it to the frequency in the continuous-time domain before analogue-to-digital conversion takes place – this is known as frequency pre-warping. Investigation into whether the current transformation of the simple RC circuit needs pre-warping will take place in Section 2.2.

2.2 MATLAB DESIGN

The IIR filter was designed and investigated using the following Matlab code shown below in *Listing 1*.

```
%Converting an anaolgue filter into a digital filter

w = 1000;       %w = 1/RC
Ts = 1/8000;    %sampling period
%analogue transfer function
G = tf(1, [1/w 1]);
%digital transfer function via Tustin transform
Gd = c2d(G, Ts, 'tustin');
%analogue transfer function with frequency warping
G2 = tf(1, [1/(2*pi*159.3625) 1]);
%digital transfer function via Tustin transform
Gd2 = c2d(G2, Ts, 'tustin');

%bode plots
bode(G)
hold on
bode(Gd)
bode(Gd2)
```

Listing 1

Figures 4 and *5* show the frequency response of the IIR filter.

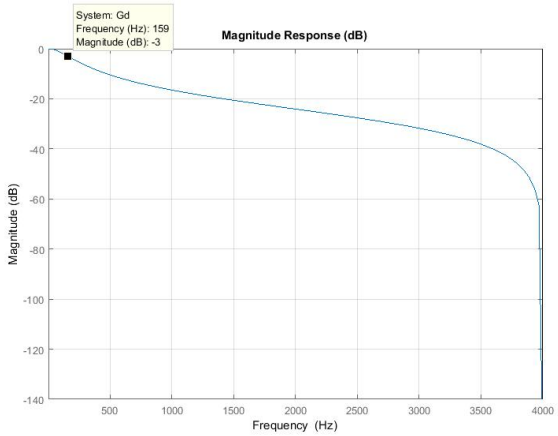

Figure 4: Magnitude Response

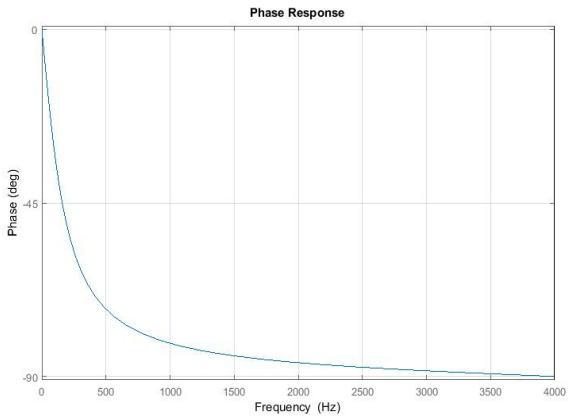

Figure 5: Phase Response

In order to check that the digital filter matches the analogue filter, the corner frequencies have been compared. The corner frequency of the analogue circuit is calculated as follows.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{(2\pi)(1\times10^{3})(1\times10^{-6})} \approx 159.155\ Hz$$

The corner frequency of the digital filter occurs at a gain of $-3\ dB$ and has been measured to be $159\ Hz$ (refer to *Figure 4*), thus it approximately matches the analogue corner frequency. This finding satisfies the claim in the lab instructions that the filter does not need frequency pre-warping.

In order to illustrate this claim visually, the digital filter was compared against a frequency pre-warped version. This was implemented in *Listing 1* via the following calculation.

$$f_c = \frac{2}{(2\pi)(\frac{1}{8000})}\tan\left(\frac{(1000)\left(\frac{1}{8000}\right)}{2}\right) \approx 159.3625\ Hz$$

The cut-off frequencies of the pre-warped and non-pre-warped filters were compared to the continuous-time filter. As illustrated in *Figure 6*, and more clearly in *Figure 7*, the pre-warped version (red) matches the continuous-time filter (blue), unlike the non-pre-warped function (green). However, since the difference between the non-pre-warped and pre-warped cases is miniscule, pre-warping is indeed deemed unnecessary.

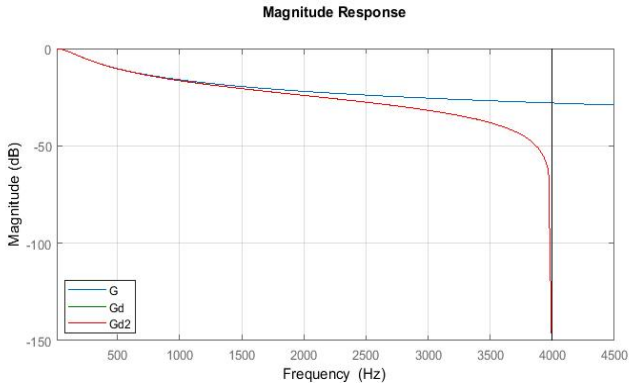

Figure 6: Magnitude Response Comparison

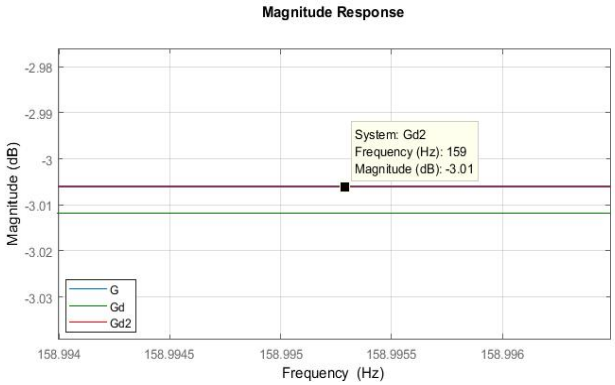

Figure 7: Zoomed In Magnitude Response

Moreover, unlike the analogue filter, the digital filters' responses are mirrored at the Nyquist frequency of $4\ kHz$ (half the sampling frequency). Therefore, this digital filter will never perfectly match the frequency response of the analogue as its gain eventually rapidly shoots down, as shown in *Figure 6*. The closer the cut-off frequency to the Nyquist frequency, the larger the difference between the two filters will become.

Finally, as explained previously in *Lab 4,* in order to achieve a linear phase in the passband, the filter coefficients must be symmetrical. However, since this is not the case in an IIR filter, the phase is non-linear, as shown in *Figure 3*.

2.3 DSK IMPLEMENTATION

The filter was implemented on the DSK via the following listings. The filter coefficients stored in buffers ***a*** and ***b*** were defined as follows,

```
39 // define two 2-element delay buffers
40 #define M 2
41 #define N 2
42 // initialise input and output buffers
43 short x[M] = {0};
44 double y[N] = {0};
45 // IIR Filter coefficients
46 double b[M] = {1.0/17, 1.0/17};
47 double a[N] = {1, -15.0/17};
```

Listing 2: Global Declarations

The filter was then implemented using a simple non-circular buffering method, as shown in *Listing 3*.

```
148 /******************** Convolution Function ***********************/
149 double non_circ_IIR(short sample){
150
151     double sum;
152
153     x[1] = x[0];
154     y[1] = y[0];
155
156     x[0] = sample;
157
158     sum = b[0]*x[0] + b[1]*x[1] - a[1]*y[1];
159     y[0] = sum;
160
161     return sum;
162 }
```

Listing 3: ***non_circ_IIR*** *Function*

Using a non-circular buffering implementation, the filter is perceived as a 'delay line'. This is where all samples are stored in an array and are shuffled down each time a new sample arrives. In other words, the array is buffered via a FIFO method (first element in, first element out).

As the simple pole filter is of order 1, there are only 2 coefficients. Therefore, the shifting process is simply executed by replacing the value in the next buffer location with the previous value as follows,

$$\boldsymbol{x}[\mathbf{1}] = \boldsymbol{x}[\mathbf{0}] \text{ and } \boldsymbol{y}[\mathbf{1}] = \boldsymbol{y}[\mathbf{0}]$$

Once the shifting is complete, the current sample is stored in the beginning of the buffer: $\boldsymbol{x}[\mathbf{0}] = \boldsymbol{sample}$.

Unlike the non-circular function implemented in *Lab 4*, no ***for*** loop is needed to implement the MAC operation for convolution as the number of coefficients is small enough for it to take place in a single line. The result is then stored in the variable ***sum*** which is copied into the output buffer in order for it to be accessed when the next sample arrives.

2.4 Measuring the Time Constant

To check the validity of the implemented digital filter, it's time constant has been measured and compared to the time constant of the analogue circuit.

The theoretical time constant of the analogue circuit is,

$$\tau = RC = (1\times10^{3})(1\times10^{-6}) = 1\ ms$$

In order to measure the time constant of the digital filter, a square wave has been inputted into the board. To avoid the attenuation from the DSK's high-pass filter and the IIR filter itself, a signal frequency of $150\ Hz$ was used. The outputted signal is shown below in *Figure 8* and has the waveform of a charging and discharging capacitor.

Using the scope, the time constant of the filter was found by measuring the time taken for the signal's amplitude to increase by 63.2% of its steady-state value. As shown in *Figure 8*, it was measured to be $0.88\ ms$ – slightly less than the $1\ ms$ RC time constant of the original analogue circuit.

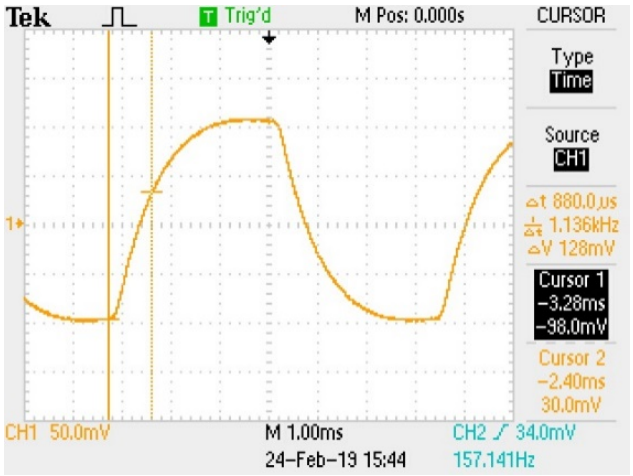

Figure 8: 150 Hz signal

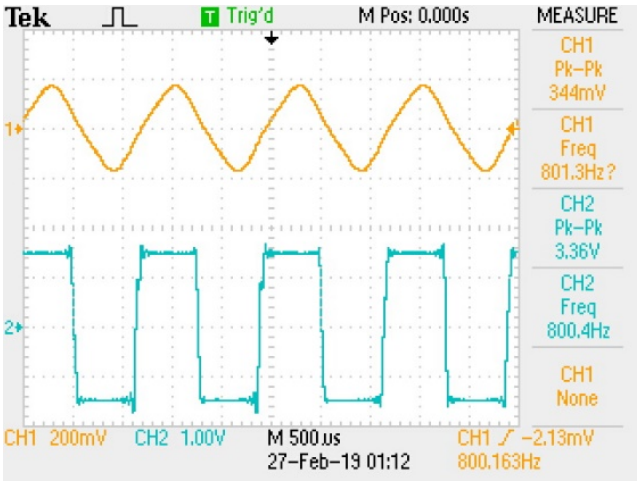

Figure 9: 800 Hz Signal

This difference between the time constants is believed to be due to the signal not reaching its steady-state value before the waveform started to decrease. This concept can be observed more clearly when signals of higher frequencies are inputted to the board, as shown in *Figure 9*. In this case, it is obvious that the time constant cannot be measured accurately as the signal never reaches its steady-state amplitude.

Note that in order to check this mismatch was not due to neglecting frequency pre-warping, the time constant of the frequency pre-warped filter was measured but was found to be the same as the non-pre-warped version. Thus, it is concluded that pre-warping had no obvious impact on the time constant measurement.

2.4 Spectrum Analyser Frequency Response

As discovered in *Lab 4*, the DSK has its own frequency response due to its internal filters that will superimpose onto the frequency response of the any additionally designed filters. The magnitude response of the DSK is illustrated in *Figure 10*.

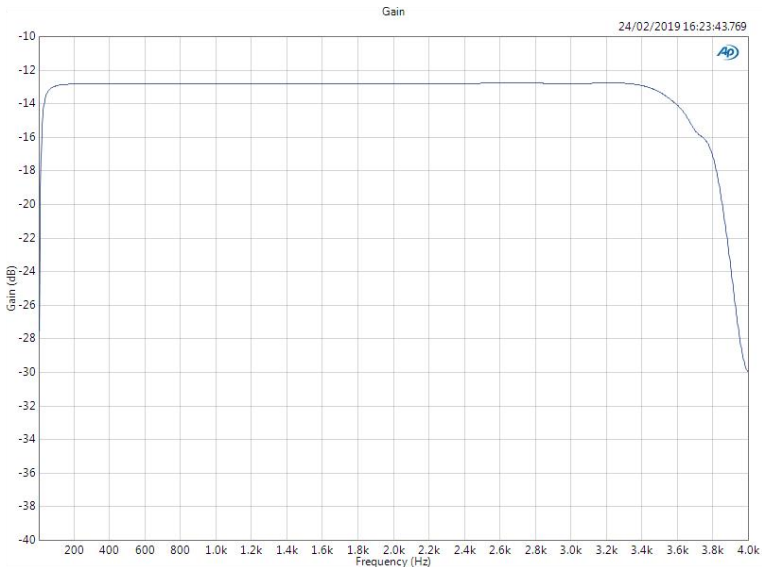

Figure 10: Magnitude Response of the DSK

The gain in the passband lies around $-12\ dB$. This is due to a gain factor of $\frac{1}{4}$ found between the spectrum analyser and the input of the codec, as shown below in *Figure 11*. This has been implemented as a safety precaution, preventing signals of significant amplitudes damaging the DSK unit.

$$20 \log\left(\frac{1}{4}\right) \approx -12.04\ dB$$

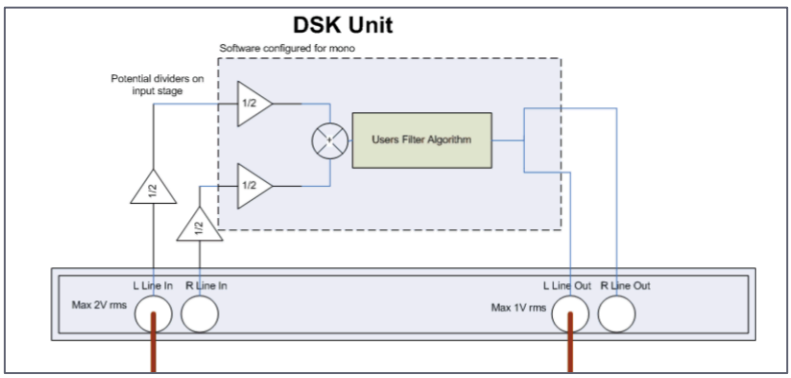

Figure 11: Gain Between the Spectrum Analyser and DSK

Furthermore, notice how the magnitude drops off at low and high frequencies. This is due to the high pass filters found at the input and output of the audio chip and the anti-aliasing low pass filter of the audio chip respectively.

The phase response of the unfiltered system is found to be linearly decreasing, as shown below in *Figure 12*.

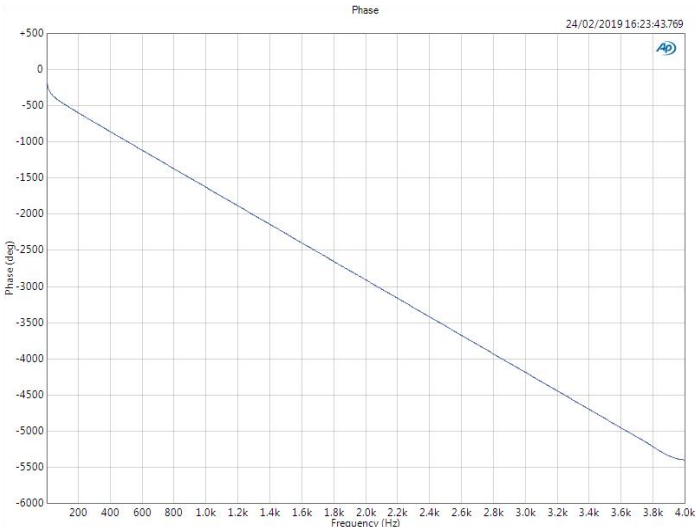

Figure 12: Phase Response of the DSK

The group delay of the system is the reason for this linear phase response. As discussed previously in the *Lab 4* report, the group delay is calculated as the derivative of the phase, hence a constant group delay results in a linear phase response. *Figure 13* shows the group delay of the unfiltered system.

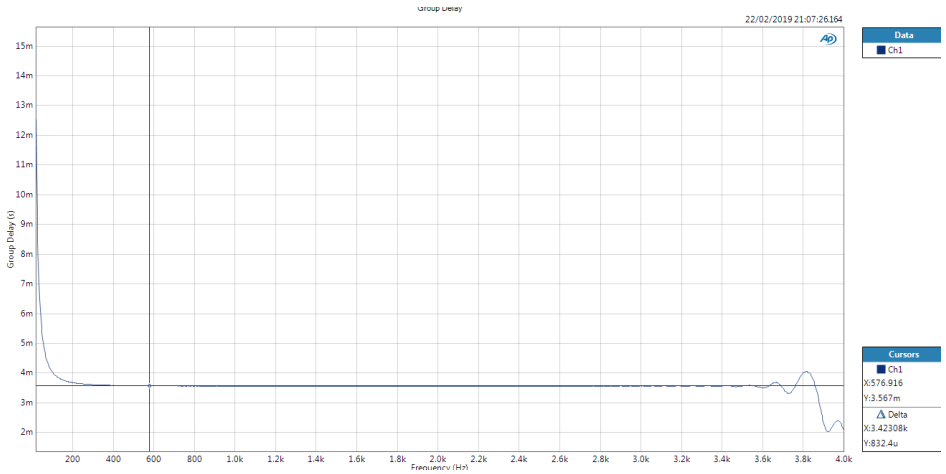

Figure 13

The delay was measured to be around 3.567×10^{-3} and is believed to be caused by the circuitry within overall system.

Knowing the frequency response of the unfiltered system, it is obvious that measuring the filtered system's frequency response will not portray the frequency response of the IIR filter alone. The frequency response of the filtered system was measured as shown in *Figures 14* and *15*.

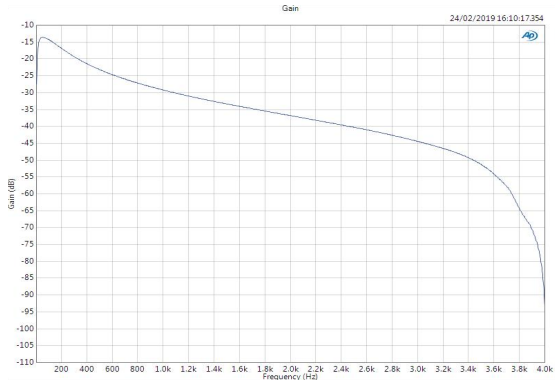

Figure 14: Magnitude Response of Whole System

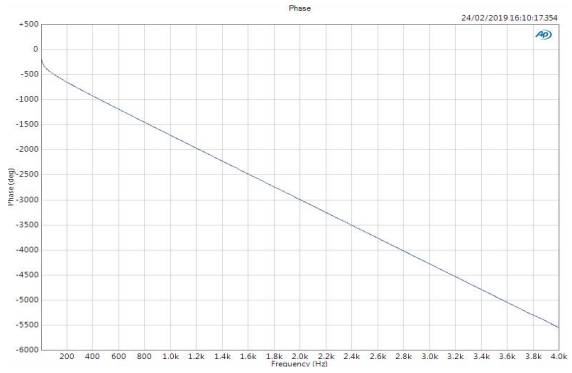

Figure 15: Phase Response of Whole System

In order to clearly observe the frequency response of the IIR filter, both frequency response data from the unfiltered and filtered responses were exported to Matlab. They were then subtracted from one another to obtain solely the IIR filter's frequency response. *Figures 16* and *17* illustrate the frequency response measured via the spectrum analyser against the 'ideal' frequency response previously produced by Matlab.

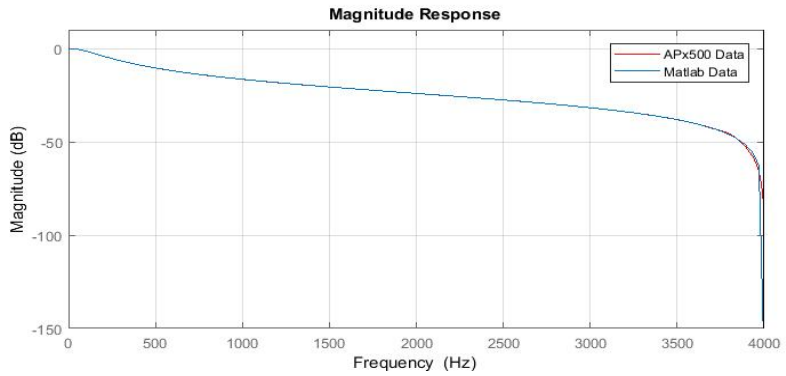

Figure 16: IIR Filter vs Matlab Magnitude Responses

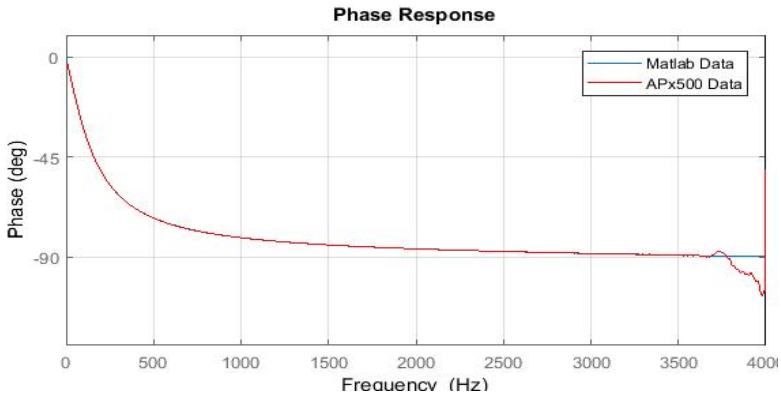

Figure 17: IIR Filter vs Matlab Phase Responses

The frequency response almost matches perfectly to the ideal frequency response produced in Matlab (see Section 2.2 for how well the filter matches the desired specifications). However, it is obvious that something is going on as 4 kHz is approached. The distortion may be due the fact that the gain shoots down rapidly as described earlier in Section 2.2.

3. Bandpass Filter Direct Form II

3.1 Elliptic Filters

The next task is to program an elliptic bandpass filter with the following specifications:

Order	**4**
Passband	$270 - 450\ Hz$
Passband Ripple	$0.3\ dB$
Stopband Attenuation	$20\ dB$

Table 1

An elliptic filter is one which has all its zeros on the imaginary axis, and its poles on an ellipse (see *Figure 18*). Zeros lead to a decrease in gain, and can be viewed to produce an inverted peak in the magnitude response, as shown in *Figure 19*. The closer the zero to the imaginary axis, the more prominent this peak is on the magnitude response. Thus, zeros on the imaginary axis are desired for large gain roll offs yet will cause ripple in the stopband.

Poles lead to an increase in gain, and can be viewed to produce a peak in the magnitude response. Again, the closer the pole to the imaginary axis, the more prominent this peak is. However, since poles are desired in the passband, prominent peaks will cause a large ripple in the pass band. Therefore, poles should not lie exactly on the imaginary axis, but close enough to keep the gain high and relatively stable.

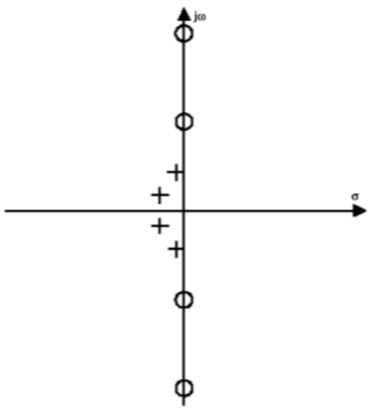

Figure 18: Zero/Pole Plot of an Elliptic Filter

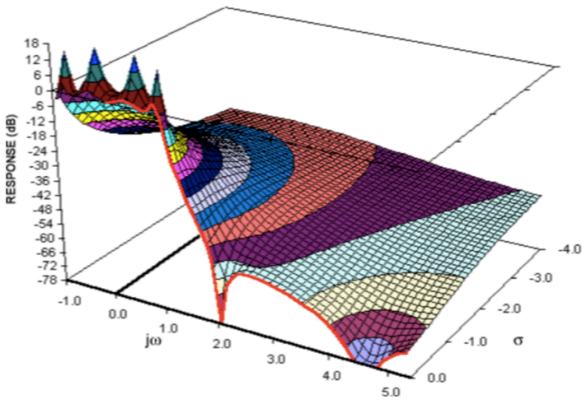

Figure 19: Magnitude Response of an Elliptic Filer

Overall, an elliptic filter is able to produce a steep roll-off compared to other filters such as Chebyshev and Butterworth, which have smaller or no passband ripple, but additionally have no zeros in their response and thus a slower roll-off in the transition band.

3.2 MATLAB DESIGN

The elliptic filter was designed in Matlab as follows,

```
%Elliptic Bandpass Filter

N = 2;  % 2N is the filter order
Rp = 0.3;   %Passband Ripple
Rs = 20;    %Max Stopband Gain
Wp = [0.0675 0.1125]; %Normalised Passband Frequency
fs = 8000; %Sampling Frequency

[B,A] = ellip(N,Rp,Rs,Wp);%design filter coefficients

freqz(B, A, 1000,'whole', fs)
fvtool(B,A)

save iir_coef.txt B A -ascii -double -tabs
```

Listing 4

Where the normalised passband frequencies are calculated as follows:

$$\frac{270}{8{,}000}\times 2 = 0.0675 \text{ and } \frac{450}{8{,}000}\times 2 = 0.1125$$

The designed filter only approximately meets the passband specifications, as shown below in *Figures 20* and *22* (note that the stopband attenuation does remain below $-20\ dB$ as desired). As explored in *Lab 4*, this could be improved if the filter was of a higher order.

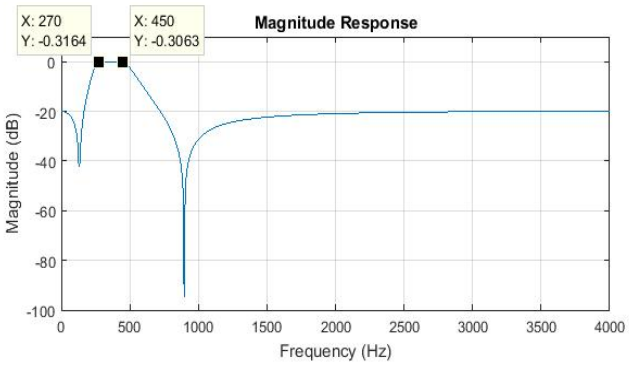

Figure 20: Magnitude Response & Passband Measurement

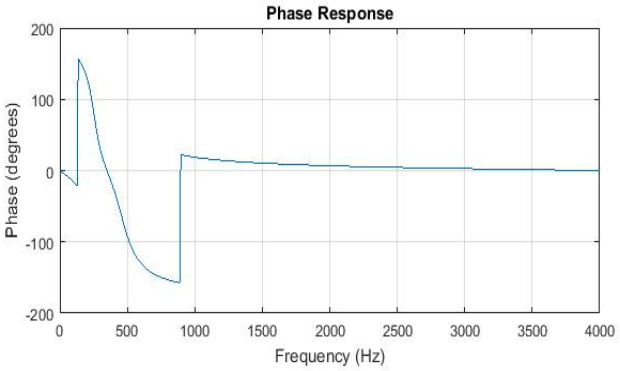

Figure 21: Phase Response

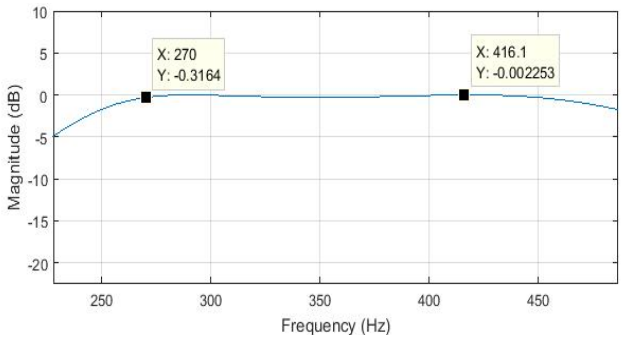

Figure 22: Passband Ripple Measurement

As the filter is of order 4, the following 5 coefficients have been produced:

```
double b[] = {9.7564943933063919e-02,-3.4286742699734390e-01,
4.9110053796258135e-01,-3.4286742699734385e-01, 9.7564943933063947e-02};

double a[] = {1.0000000000000000e+00,-3.6227382800818599e+00,
5.0638877901317381e+00,-3.2346102598649416e+00, 7.9841646815527811e-01};
```

Figure 23

Figure 24 shows the pole/zero plot of the IIR filter in the z-plane. As discussed in Section 3.1, notice that the zeros lie on the unit circle (equivalent to the $j\omega$ axis in the s-plane) and that the poles are not restricted to the origin. In this case, no poles lie outside the unit circle thus the 4th order filter is stable.

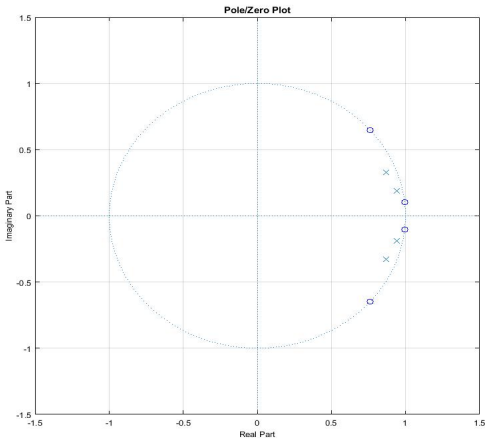

Figure 24: Pole Zero Plot

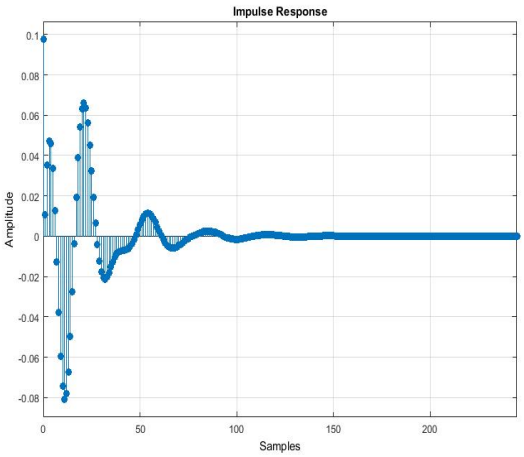

Figure 25: Impulse Response

Moreover, *Figure 25* illustrates the impulse response of the IIR filter. As its names suggests, the impulse response is infinite, and since this filter is stable, the impulse response decays to zero. The 'infinite' impulse response, in terms of implementation on the DSK, is actually achieved via a feedback loop.

As discussed in Section 2.2, the phase is non-linear and since the group delay is calculated as the derivative of the phase, the group delay is not constant, as shown in *Figure 26.*

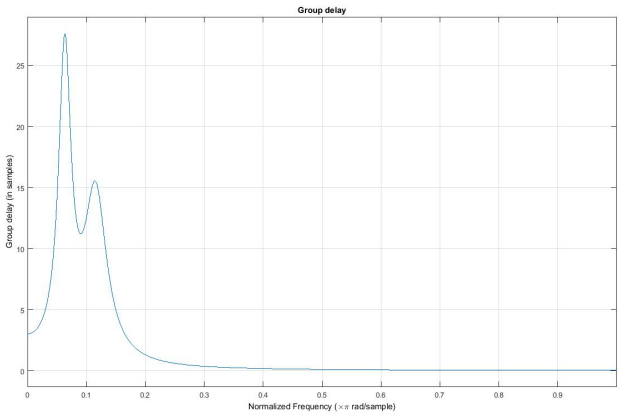

Figure 26: Group Delay

3.3 DSK IMPLEMENTATION

When implementing an IIR filter, there are two methods one can take. By rearranging the transfer function (refer to Section 1.1), it can be rewritten as a cascade of two subsystems in series.

$$H(z) = H_1(z) * H_2(z) = (b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}) * \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}}$$

Figure 27 represents this visually, and is known as Direct Form I.

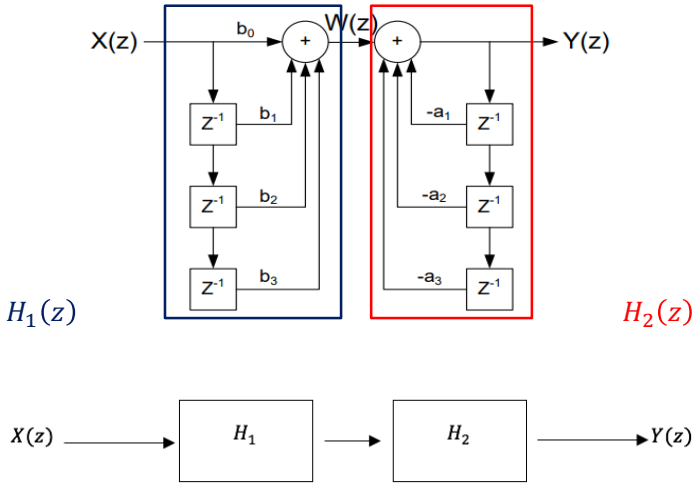

Figure 27

In order to implement the Direct Form I, two delay lines via two buffers will be needed. If the order of the two transfer functions is swapped, the system can be redrawn as follows.

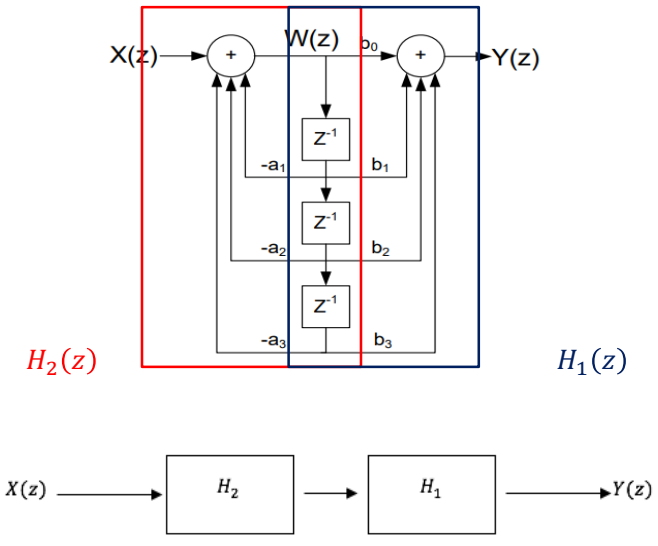

Figure 28

This is known as Direct Form II, where only one buffer is needed for implementation, allowing for easier code and faster run times.

Using Direct Form II, the following code was written to implement the IIR filter.

```
/******************** Circular Convolution Function **********************/
double IIR_Direct_Form_II(short sample){

    int i;
    double sum = 0;

    *index = sample;        //store sample in the array
    *(index+order+1) = sample;  //store sample in the 'copy' array

    //convolution
    for(i = 1; i < order+1; i++){
        *index += -a[i]*(*(index+i));
        sum += b[i]*(*(index+i));
    }

    *(index+order+1) = *index;  //update 'copy' pointer

    sum += b[0]*(*(index));

    index--;
    if(index < x){
        index = x+order; //overflow prevention
    }

    return sum;
}
```

Listing 5

As discovered in *Lab 4*, a circular buffer method with an array size large enough to store two copies of the incoming samples allowed for the smallest execution time. Thus, this method has been implemented.

The buffer ***x*** is created during runtime using the ***calloc*** function, which returns a pointer to the start of a block of memory of size $(2 * (\boldsymbol{order} + 1))$. This initialisation allows for a variable array size – this will come in handy later when other filter orders are implemented.

```
x = (double *)calloc(2*(order+1), sizeof(double));
index = x+order;
```

Listing 6

The buffer can be perceived in terms of two arrays – the first which stores the incoming samples, and the second which stores copies of the incoming samples. The pointer ***index*** maps the new sample to a new location in the array. A copy is then stored in the 'second' array by offsetting the pointer by $\boldsymbol{order + 1}$.

Once the samples have been stored, a ***for*** loop implements convolution. Firstly, the current sample (pointed to by ***index***) is added to the previous samples weighted by the coefficients in the buffer ***a***. The output of the convolution is then added with previous samples weighted by the coefficients in buffer ***b***. Finally, the output is added with the current sample weighted by the coefficient ***b***[**0**].

Once convolution is completed, ***index*** is decremented in order to store the next sample in the next location in the buffer. An ***if*** statement is used to prevent the pointer from going out of bound.

3.4 Spectrum Analyser Frequency Response

Using the spectrum analyser, the frequency response of the system was measured as follows:

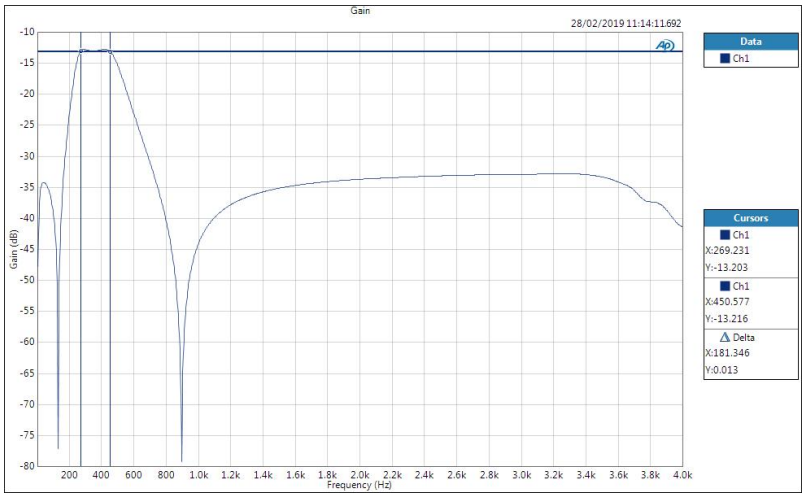

Figure 29: Magnitude Response

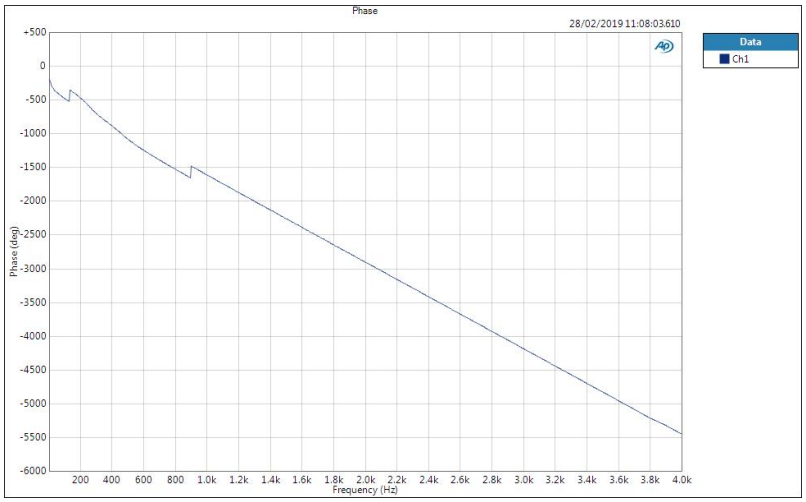

Figure 30: Phase Response

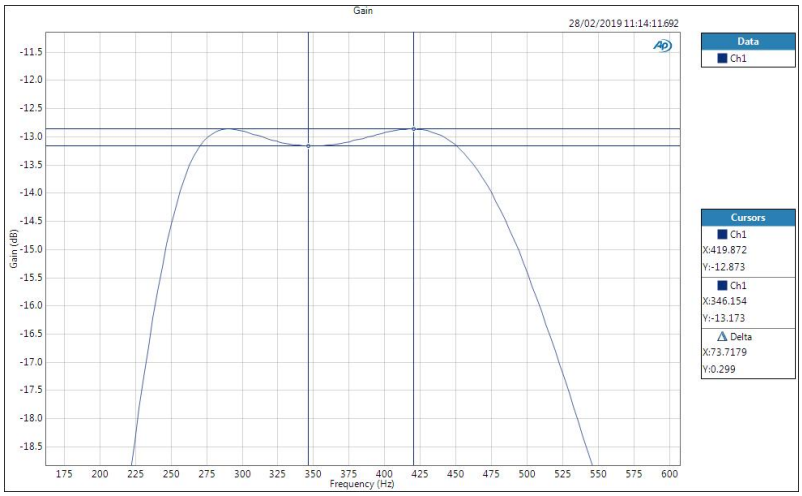

Figure 31: Passband Ripple

Instead of analysing the frequency response of the whole system above in detail, the IIR's response has been found by subtracting the whole filtered system by the unfiltered DSK's response. As illustrated in *Figure 32,* the frequency response matches the ideal response produced by Matlab and thus approximately matches the specifications, as illustrated previously in Section 3.2.

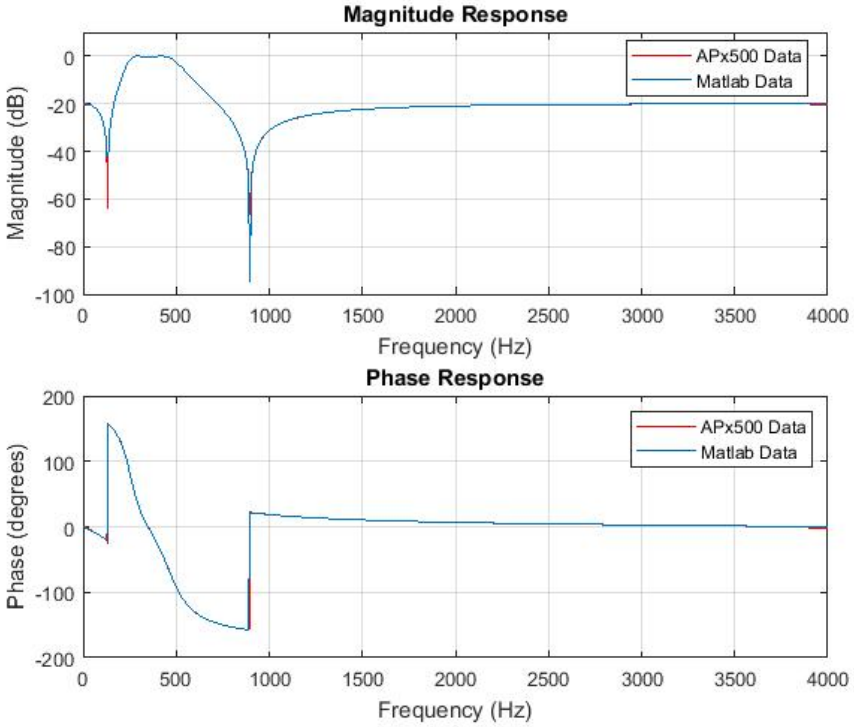

Figure 32

In order to verify that the IIR filter has been successfully implemented, signals of relevant frequencies have been inputted into the DSK, and their subsequent outputs have been observed and analysed as follows.

Output signals of frequencies lying in the passband were observed to have a gain attenuation of 0.5 dB. This is expected due to the potential divider found at the input of the audio chip.

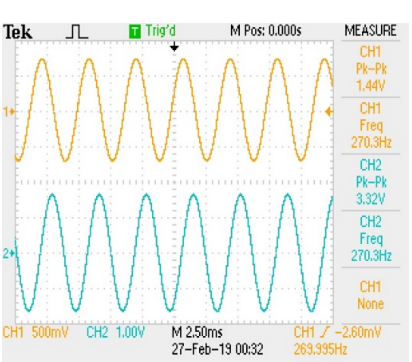

Figure 33: 270 Hz Input Signal

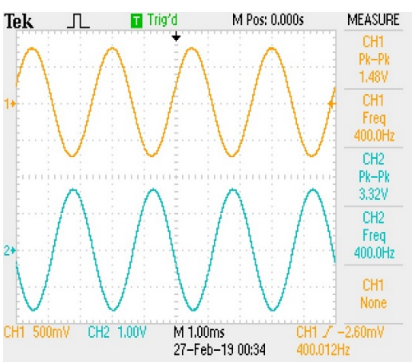

Figure 34: 400 Hz Input Signal

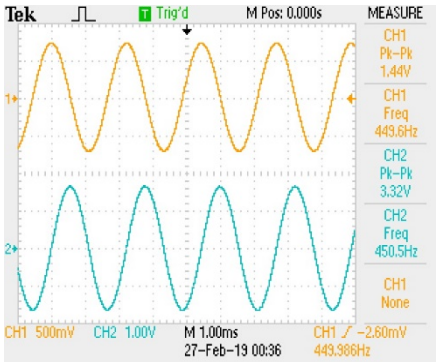

Figure 35:450 Hz Input Signal

Figure 36 illustrates the output signal when a signal of $125\ Hz$ was inputted. The gain of the output signal was measured to be $20\log\left(\frac{0.0208}{3.32}\right) \approx -44.06\ dB$, which roughly coincides to the value of the gain at the first trough in the magnitude response (see *Figure 32*).

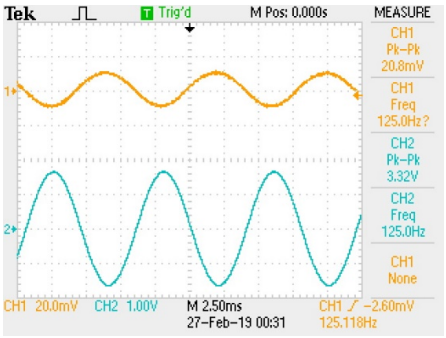

Figure 36: 125 Hz Input Signal

At the second trough of the magnitude response, the effect of the attenuation is much more significant, causing the output signal amplitude to be close to 0V, as shown in the figure below.

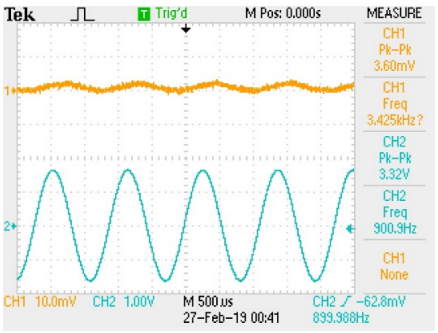

Figure 37: 900 Hz Input Signal

Finally, in order to verify the minimum stopband attenuation, an input signal of 3 kHz was used. The filter satisfies the minimum stopband attenuation of $20\ dB$.

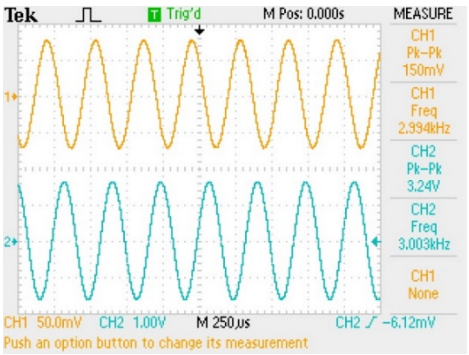

Figure 38: 3 kHz Input Signal

$$Stopband\ Gain = 20\log\left(\frac{0.150\times 2}{3.24}\right) = -20.67$$

4. Bandpass Filter Direct Form II – Transposed

4.1 Filter Design

The Direct Form II network will remain unchanged in terms of behaviour by:

- Reversing the direction of each branch
- Interchanging branch divisions and branch summations
- Swapping input for output

This is known as transposing the network, as is illustrated in *Figure 39*.

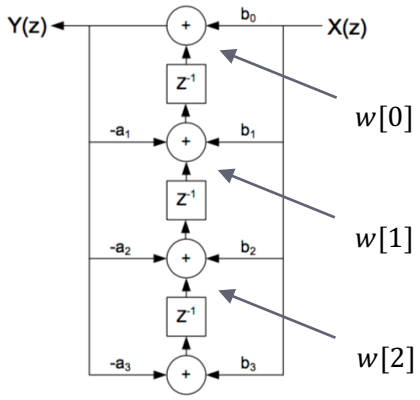

Figure 39

Transposing the network has the benefit of a more efficient code implementation – this will be analysed in more detail in Section 5.

4.2 DSK Implementation

The Direct Form II - Transposed method is implemented via the following listing.

```
/******************** Convolution Function **********************/
double IIR_Direct_Form_II_Transposed(short sample){

    int k;
    double output;

    output = b[0]*sample + *buffer;

    for(k = 0; k < order-1; k++){
        *(buffer+k) = *(buffer+k+1) - a[k+1]*output + b[k+1]*sample;
    }

    *(buffer+order-1) = - a[order]*output + b[order]*sample;

    return output;
}
```

Listing 7

The transposed network can be defined as the following difference equation:

$$y(n) = \sum_{k=1}^{M} [b[k]x(n-k) - a(k)y(n-k)] + b_0x(0)$$

The code simply implements this equation – however, the order of calculations is not so straight forward. Therefore, an example of a 4th order filter will be worked through systematically.

Firstly, the current sample is passed into the function and is stored in the variable ***sample***. The variable ***output*** is the result that is to be outputted by the DSK, or can be viewed as the $y(n)$ in the difference equation. A pointer ***buffer*** points to a value that stores the result of the MAC part of the difference equation (this calculation was performed when the previous sample was received and will be explained later). The buffer is initialised as,

```
buffer = (double *)calloc(order+1, sizeof(double));
```

Listing 8

In order to achieve the final result, this value is added to $\boldsymbol{b}[\mathbf{0}]\times\boldsymbol{sample}$ and the sum is stored in ***output***. Therefore, so far, the following calculation has been made:

$$output = b[0] * sample + w[0]$$

where $\boldsymbol{w}[\mathbf{0}]$ represents the location ***buffer*** points to and thus is where the result of the MAC operation is stored (refer to *Figure 39*, but note that this picture refers to a 3rd order filter).

The rest of the code calculates and stores the MAC part of the equation for when the next sample arrives. A ***for*** loop implements this by incrementing the pointer position and the location in the arrays storing the coefficients as follows:

$$w[0] = w[1] - a[1] * output + b[1] * sample$$

$$w[1] = w[2] - a[2] * output + b[2] * sample$$

$$w[2] = w[3] - a[3] * output + b[3] * sample$$

Once completed, the last section is added, completing the MAC part of the difference equation.

$$w[3] = a[4] * output + b[4] * sample$$

4.3 Frequency Response

Again, subtracting the frequency response found using the spectrum analyser with the frequency response of the DSK, the frequency response of the IIR filter has been found and is shown in *Figure 40*. Again the response matches the ideal Matlab response.

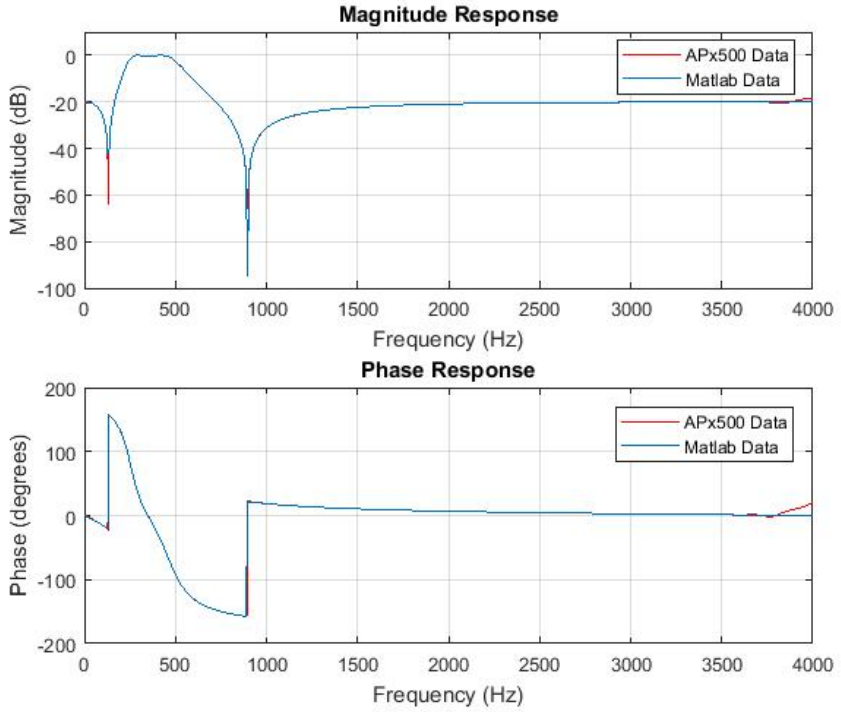

Figure 40

For completeness, the passband ripple found in the APx500 magnitude response has been measured and just about meets the specification of a deviation of $-0.3\ dB$, as shown below in *Figure 41*.

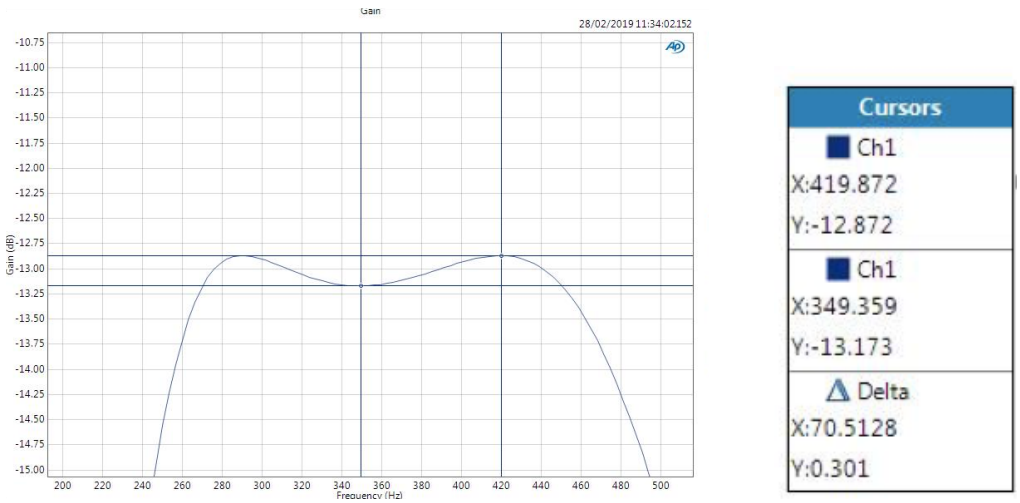

Figure 41: Spectrum Analyser – Passband Ripple Measurement

5. PERFORMANCE COMPARISON

The execution times of the two Direct Form II implementations have been found and placed in *Table 2*. The clock cycles were measured with both no optimisation and at an optimisation level –o2.

Filter Order	**Non-Transposed**		**Transposed**	
	No Optimisation	-o2	No Optimisation	-o2
2	290	169	167	151
4	438	219	285	168
8	734	319	521	188
12	1030	419	757	208
16	1326	519	993	228
20	1622	619	1229	248

Table 2

Whilst increasing the order of the filter, it was observed that a filter of order higher than 14 became unstable. A filter becomes unstable when a pole lies outside the unit circle, as illustrated in *Figure 42*.

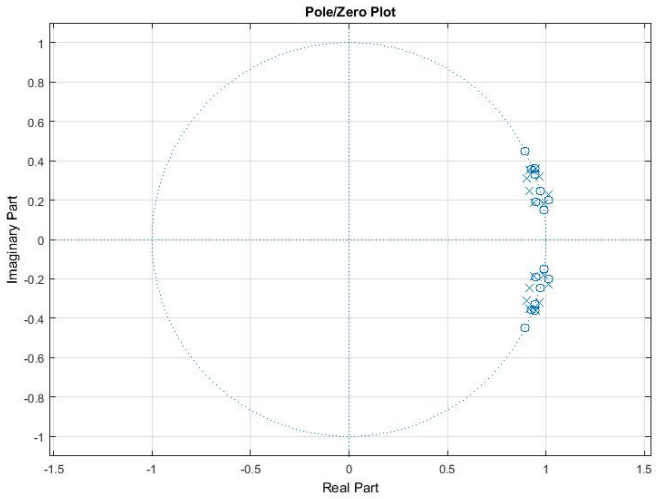

Figure 42: Pole/Zero Plot for a 16th Order IIR Filter

As briefly mentioned in Section 1, instability can occur in an IIR filter as poles can be occur anywhere on the z-plane and thus can occur outside the unit circle. The system becomes BIBO (bounded-input bounded-output) unstable – this happens thanks to the feedback loop.

The effects of a pole lying outside the unit circle can be more intuitively understood by observing the impulse response. As shown in *Figure 43*, the impulse response does not converge to zero, unlike the 4th order filter analysed in Section 3.2, but uncontrollably increasing towards infinity.

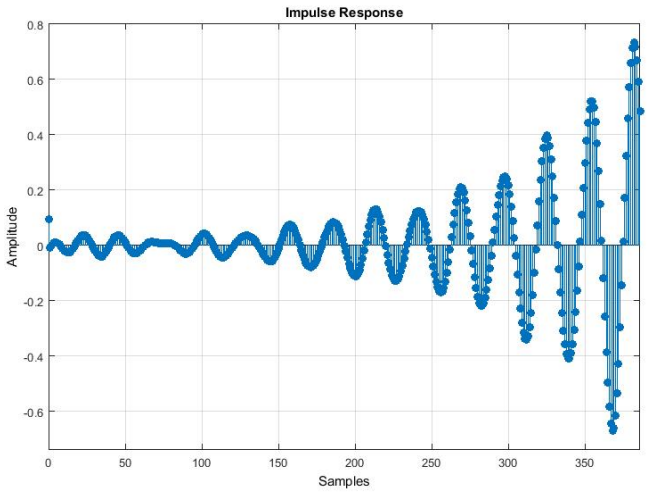

Figure 43: Impulse Response of a 16th Order IIR Filter

The following plot illustrates the difference between the two implementations when no optimisation is applied.

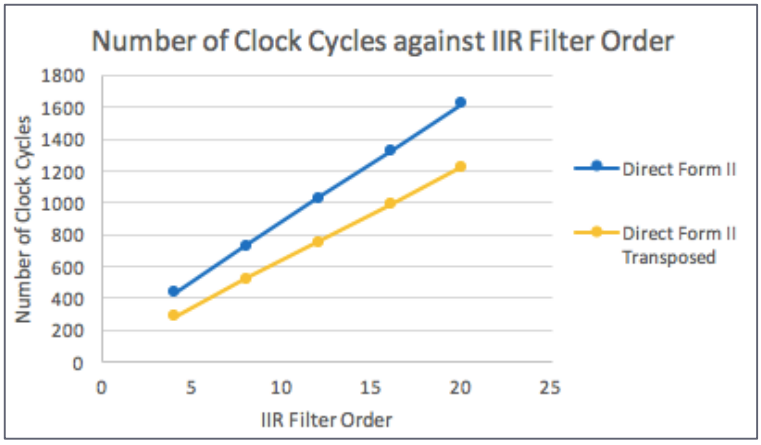

Figure 44

There is therefore an obvious linear relationship between the number of clock cycles and the filter order. The relationship can be characterised via the following equations.

For Direct Form II: $y = 74x + 142$

For Direct Form II – Transposed: $y = 59x + 49$

For optimisation level –o2:

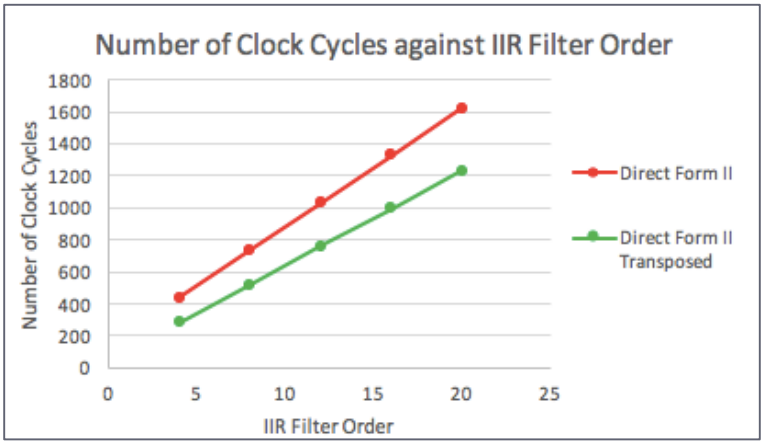

Figure 45

The relationship between number of clock cycles and filter order is:

Direct Form II: $y = 25x + 119$

Direct Form II – Transposed: $y = 5x + 148$

In conclusion, the transposed filter implementation is faster than the non-transposed version. This is because the shifting of samples is inherent in the output calculations. In other words, no circular buffer or shifting of samples is needed as values in the buffer are instead overwritten, reducing the overall execution time of the function.