Embedded System Design: DSP Implementation - Assignment

Student 1: Frank Conway —— 202032117
Student 2: Andrew Law —— 202009334
Class Code: EE580

February 22, 2025

Introduction

In this assignment, we will explore designing 2 FIR filters to match specifications and them in MATLAB. They will then be quantised and adapted to implement them on OMAP-L138 Experimenter Kit which uses the DSP C6748 chip, via CCS12 (Code Composer Studio v12). Afterwards, the CCS12 and MATLAB version will be compared using MATLAB to determine how close the hardware implemented version is to the true software version.

Part 1: Design an FIR filter in Matlab, and Filter Signals

Using *'repmat()'*, two arrays that stored the student reference numbers ($[2, 0, 2, 0, 0, 9, 3, 3, 4]$ for Andrew Law, and $[2, 0, 2, 0, 3, 2, 1, 1, 7]$ for Frank Conway) were replicated 100 times. The mean of each signal was taken and subtracted from each respective signal, removing the DC component, and making them have a zero mean. These were stored as x_1 (for Andrew - referenced as '1' throughout) and x_2 (for Frank - reference as '2' throughout). The sampling frequency was calculated by averaging the concatenated last 4 digits of each student reference number. Code for this can be found here Listing 1 Appendix .

The time domain plot of each signal (only 3 cycles, therefore 27 samples) is shown in Figure 1 and 2.

The *'fft()'* was used to transform x_1 and x_2 into the frequency domain. The FFT length was set to 1024 as this was greater than the number of samples, hence a more accurate spectrum would be produced due to the number of frequency bins equal to nf_s/N. This will be further improved by making a longer transition band later on. The value of 1024 is also due to being a power-of-two value, which may force the FFT computation to be more efficiently implemented in both MATLAB software and OMAP-L138 hardware.

The frequency domain plots of signals $x1$ and $x2$ are shown in Figure 3 and 4.

Note the spectrum plots are symmetric around $f_s/2$, as there is a spectral replication from $f_s/2 \rightarrow f_s$, due to the signal's frequency components being periodic at integer multiples of $f_s/2$.

The filters were designed by analysing the plots in Figure 3 and 4. Since the strongest component of both spectrums occurred at $1906.8Hz$ (962.2 for x_1 and 652.7 for x_2), the strongest component of x_1 ($1906.8Hz$) and the second-strongest component of x_2 ($1274.9Hz$) were chosen to be removed. Both signals required a band stop filter, since both aforementioned components were not the lowest frequency components.

The *'filterDesigner'* tool in MATLAB was used to design the filters, which used several parameters. *'Bandstop'* filter response type was selected, along with *'FIR'* design method. '*Specify order'* determines number of coefficients of each filter. Both were set to 28 to reduce number of coefficients, plus some experimentation deemed

it to provide sufficient results. *'Fpass1'* is the first passband edge frequency, which is the upper edge of the lower passband, just before the stopband start, defining the frequency below which the filter allows signals to pass with minimal attenuation and must be less than *'Fstop1'*. *'Fstop1'* is the first stopband edge frequency and is the lower edge of the stopband, defining where the attenuation starts, between *'Fpass1'* and *'Fstop1'*. *'Fstop2'* is the second stopband edge frequency and is the lower edge of the stopband, defining where the attenuation ends, between *'Fstop2'* and *'Fpass2'*. *'Fpass2'* is the second passband edge frequency, and is the lower edge of the stopband, defining frequency above which the filter allows signals to pass with minimal interference, and must be greater than *'Fstop2'*. Finally, the *'Fs'* is set to 5726 Hz.

$'filter'_a$, the filter for signal $x1$, is set as the following: *'Fpass1'* = 1500, *'Fstop1'* = 1850, *'Fstop2'* = 1950, *'Fpass2'* = 2300. $'filter'_b$, the filter for signal x_2, is set as the following: *'Fpass1'* = 750, *'Fstop1'* = 1150, *'Fstop2'* = 1250, *'Fpass2'* = 1650.

For the filter design, the target was to keep the transition band as long as possible. Since the ratio of the transition band to f_s dictates the number of coefficients, and f_s can't be changed, a longer transition band produces a less sharp ratio and there are less coefficients. Due to the nature of the design, both FIR filters become partial notch filters, due to the narrow stopband, whereby at particular frequencies the signal are completely attenuated.

The time domain, magnitude, and phase response of $'filter'_a$ is shown in Figure 7, 9, and 11. For $'filter'_b$', they are shown in Figure 8, 10, and 12.

Using the filter coefficients, and the *'zplane()'*, the pole-zero diagram was derived. The filter coefficients were also quantised using *'quantize()'*, creating an object that can quantise arrays and variables into single precision. The pole-zero diagram before and after quantisation of $'filter'_a$ is shown in Figure 13 and 15, with Figure 14 and 16 for $'filter'_b$.

Analysing those figures, the poles and zeros don't change positions, hence the characteristics do not change. All poles are at the origin, hence both filters are FIR filters, and therefore inherently stable since all poles are within the unit circle. The number of poles and zeros are equal, both 28, hence the filters are causal. There are zeros outside the unit circle, hence the filters are not minimum phase systems. This means they cannot be inverted, however that is not a problem in this scenario, as there is no desire to invert the fitlers. These properties mean both filters can be implemented with little worry, as OMAP-L138 will not have any issues around stability.

The signals $x1$ and $x2$ were filtered with $'filter'_a$ and $'filter'_b$, respectively, using *'conv()'*. The time domain plot of the filtered signal of $x1$ is shown in Figure 17 and $x2$'s is shown in Figure 18.

The magnitude responses were taken using the FFT method as before, with $x1$'s response shown in Figure 19 $x2$'s response shown in Figure 20. By comparing before and after filtering, $'filter'_a$ successfully removed the $1.9kHz$ component, and $'filter'_b$ successfully removed the $1.3kHz$ component, both without impacting any other components.

Part 2: Implement an FIR filter in C, and Filter signals

The file 'data.h' stored the floating point representations of the filter coefficients, under $'filter'_b$ and $'filter'_a$.

Similarly to the MATLAB script, two arrays were defined that stored the student reference numbers, *'AL'* and *'FC'*. Arrays *'AL_sig'* and *'FC_sig'* were defined to store the repetitions of *'AL'* and *'FC'*, respectively, with *'AL_sig_mu'* and *'FC_sig_mu'* being defined to store the final input signals, and *'AL_mu'* and *'FC_mu'* storing the mean values. Note that all variables and arrays were defined as single due to the nature of the values (and the floating point filter coefficient weights), and all bar *'AL'* and *'FC'* instantiated as $'0'$ to remove any errors (or aggregiously large values that were initially met during a test run).

The means were calculated by summing the respective values in a for loop, then dividing them by the length, '9'. Another for loop was used to subtract the mean value away from the signal. The modulo operator was used to easily iterate through the original student number several times, as it functionaility provides the wrap around ability without requiring an if statement. This is due to the modulo operator taking the remainder of a division, which in the case of the number 9, is either the value of the iterator being divided by 9, 0 to 8, or goes back to 0 if the iterator $i = 9$. Also, removing the if statment is crucial as DSP architecture does not handle if statments well, since it causes the algorithm to jump around, potentially leading to inaccurate operations as it may just leave all the samples in a buffer and not transfer them over.

Before convolution, the input signals are padded with 28 zeros to ensure the output has the same length as the input. The convolution was implemented by analysing the convolutional equation (Equ. 1) [1] and the difference equation ((Equ. 2) [2] - as this can be implemented in hardware:

$$y[n] = \sum_{v=-\infty}^{\infty} h[v]x[n-v] \tag{1}$$

$$y[n] = \frac{b_0 x[n] + \cdots + b_{M-1} x[n-M+1]}{a_0} \tag{2}$$

The pattern of MAC operations were studied to determine how Equ. 2 could be implemented, with the pattern of a simple example shown in Figure 21.

From the pattern, it is clear that the convolution can be performed using two nested loops: the outer loop iterates over each value of n from 0 to the filtered signal length

of 928, while the inner loop iterates over the filter. The inner loop follows two conditions: the filter index v must be less than or equal to the current input index ($v \leq$n) and less than the filter length ($v < 29$).

For the first 28 multiply-accumulate (MAC) operations samples, the filter overlaps only partially with the input signal, convolving up to the current input index. After this initial phase, the entire filter fully overlaps with the input signal at each step. Due to zero-padding, the convolution near the end of the signal appears as if only the final filter elements interact with the trailing input values, similar to how a filter and signal slide over each other in a graphical representation.

Similarly to demonstrate how the convolution is performed a signal of length 4 is convolved with a filter of length 7 (4 filter coefficient and padded with 3 zeros), the code for this is shown in Listing 3. The Figures 22 and 23 demonstrate how a convolution is carried out using 2 *for* loops. The outer loop set the initial output values to zero then the inner loop collects all the products of the terms and sums them and assigns the total to the current index of the output signal. Padding with zeros allows the loop to convolve the final elements of the signal and filter without additional logic, such as an *if* statement. Introducing an *if* statement would disrupt pipelining, which enables certain operations to run concurrently, leading to more efficient computation.

To further illustrate the convolution a break point was set at line 20 and using the *'Memory Browser'* each time the register, for the output vector, was updated the memory was observed. This shows how the OMAP-L138 computes the convolution. This is shown in Figures 32 to Figure 63.

Many variables were selected to be under *'watched expressions'*, so that their time domain signal could be plotted via *'Single Time'* under *'Graph'*, as well as their magnitude response, via *'FFT Magnitude'* under *'Graph'*.

In relation to the time domain graphs, the *'Start Address'* was set to the variable name. *'Acquisition Buffer Size'* and *'Display Data Size'* was set to the length of the array, so either 900 or 928. *'DSP Data Type'* was set to *32-bit floating point*. *'Sampling Rate'* was set to 5726. In relation to the magnitude response graphs, the *'Start Address'* was set to the variable name. *'Acquisition Buffer Size'* was set to the length of the array, so either 900 or 928. *'DSP Data Type'* was set to *32-bit floating point*. *'FFT Frame Size'* was set to 1024 and *'FFT Order'* was set to 10, so that the size was greater than the number of samples.

Part 3: Compare the outputs from the C and Matlab filters

To compare the outputs from CCS12, the output of the OMAP-L138 needed to be saved. This was achieved in *C* by creating a *FILE* object from the *stdio* library. Using the *fopen()* function, the output array was iterated over, and each element was written to a file using *fprintf()*. The resulting values were stored in a text file,

which could then be imported into MATLAB as a vector.

They were read into MATLAB using *'readmatrix()'*, to store the two signals into arrays. The time and frequency domain of the filtered $x1$ signal is shown in Figure 64 and 66, with the filtered $x2$ signal shown in Figure 65 and 67.

To compare the CCS12 and MATLAB outputs, the sum of squared error (SSE), Equ. 3, was calculated by taking the difference between the two outputs, squaring them, and summing them up.

$$SSE = \sum_{n=o}^{N} (y[n]_{matlab} - y[n]_{ccs})^2 \tag{3}$$

The SSE for the filtered signal y_1 is $3.324e-10 = -94.782\ dB$ and for y_2 is $1.342e-10 = -98.720\ dB$. The logarithm was also calculated, due to how small the difference was. Analysing the MATLAB Figures showing MATLAB and CCS12 results, the time domain signals all had the same shape and values. For the frequency domain, the unfiltered signals had the compoennts in the same positions and magnitudes, which makes sense as both have the same data from the start (might be very small differences in precision, but this is not seen in any Figures). For the filtered $x1$, both MATLAB and CCS12 have the $1.9kHz$ frequency attentuated, and for the filtered $x2$, the $1.3kHz$ frequency is attentuated. All remaining components are left unaffected.

Conclusion

Overall, MATLAB was successfully used to create 2 filters, and plotted to demonstrate the success. By quantising the filter coefficients and using theory on convolution and difference equations, both filters were successfully implemented in CCS12 using OMAP-L138 board, using *'Watch Expressions'* and *'Memory Browser'* to analyse the results. Finally, the data outputted by the OMAP-L138 board was saved into a file and displayed on MATLAB, and both filtered outputs from MATLAB and CCS12 were compared in MATLAB to show that both methods produced the same results.

Appendix A

Part 1 Figures and Code.

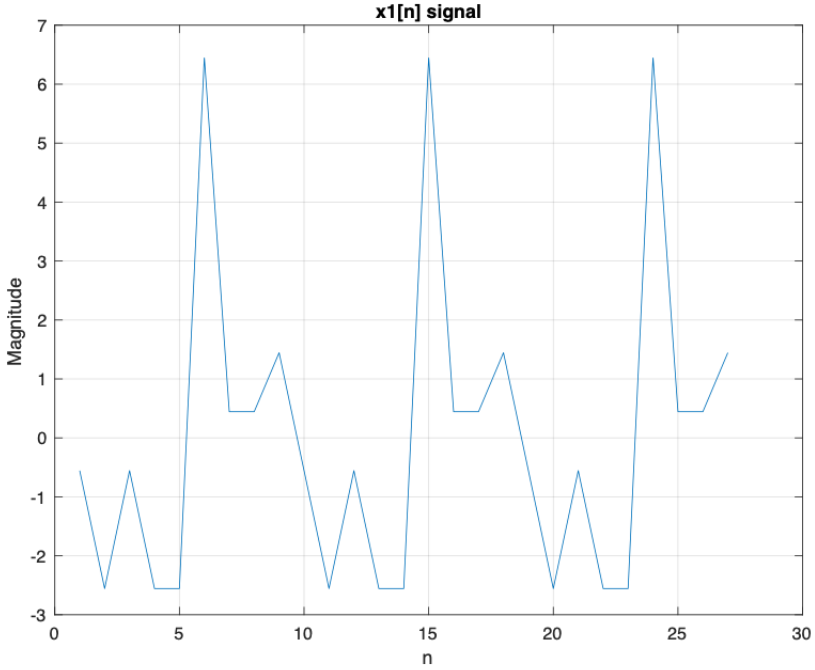

Figure 1: Signal $x_1[n]$ (AL)

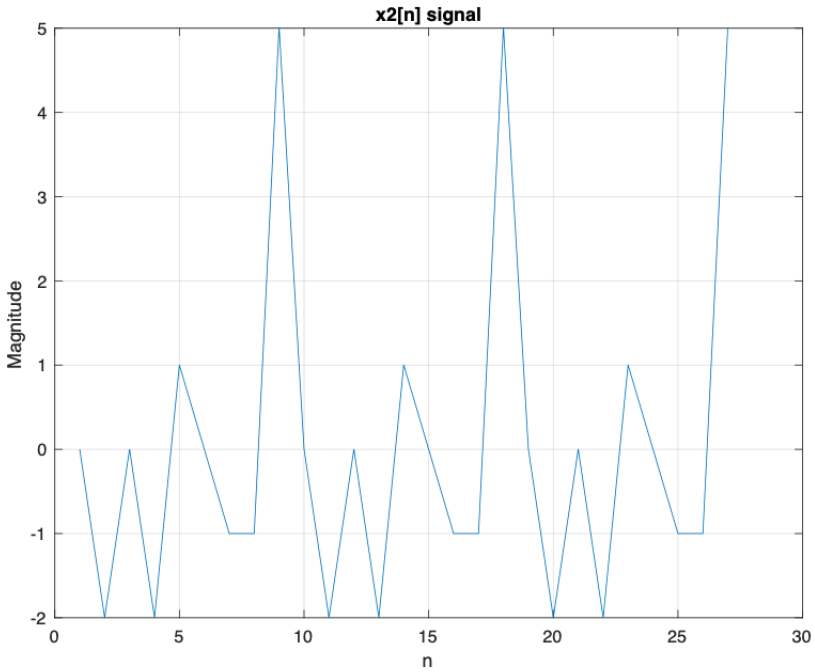

Figure 2: Signal $x_2[n]$ (FC)

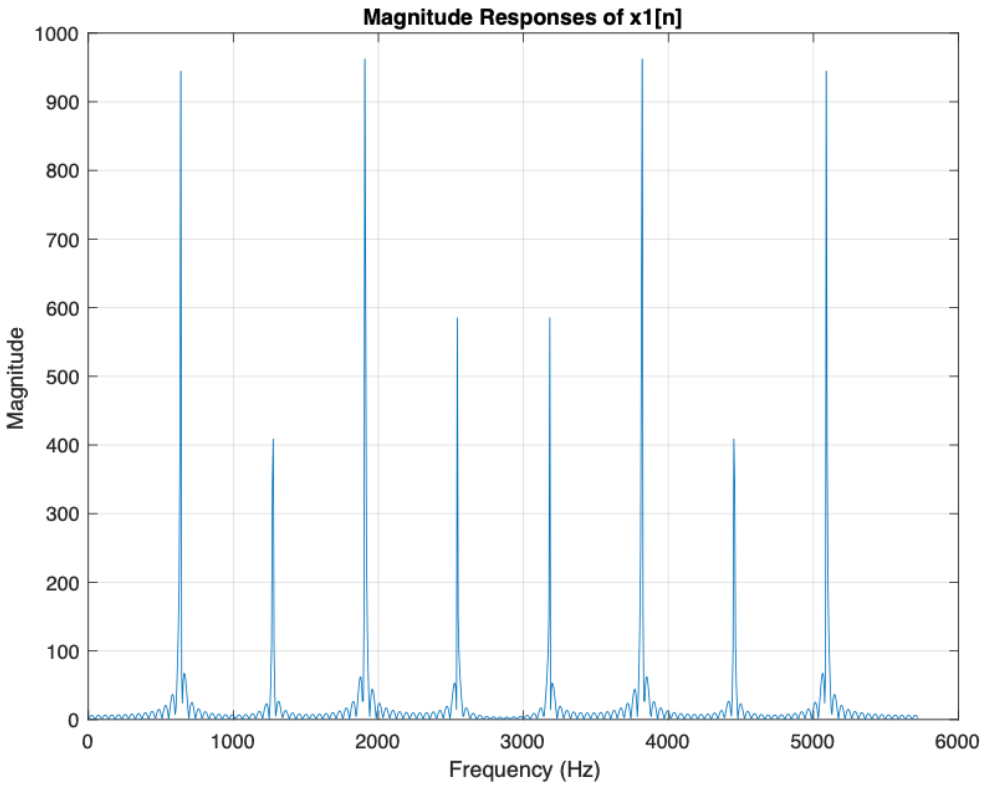

Figure 3: Magnitude Response of $x_1[n]$ (AL)

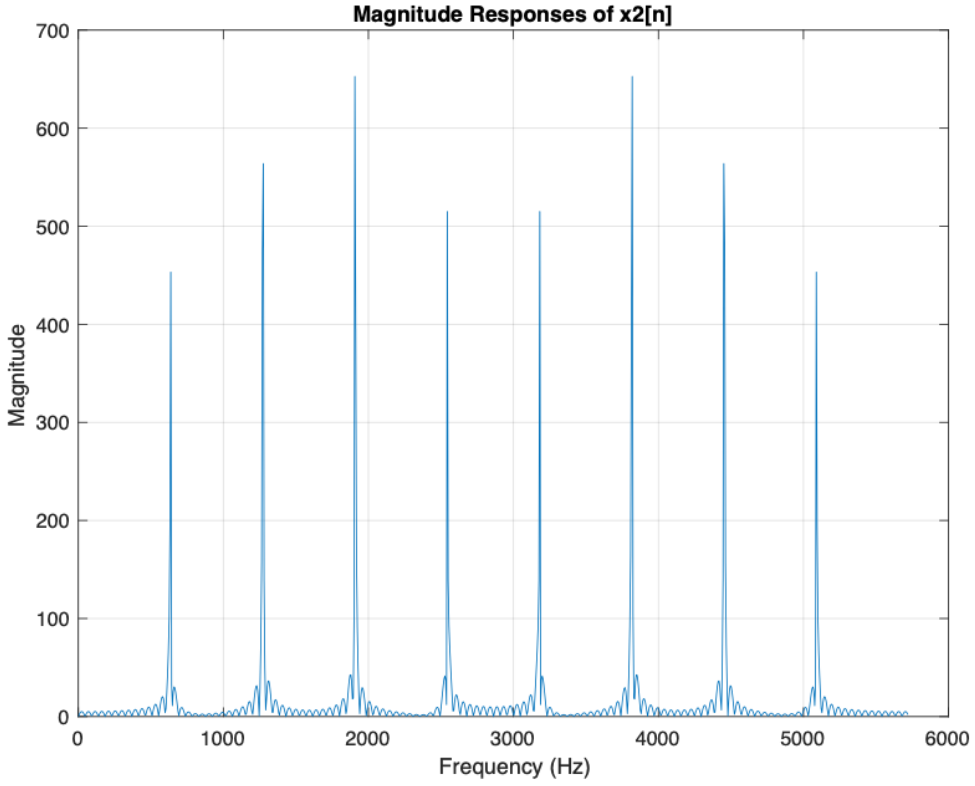

Figure 4: Magnitude Response of $x_2[n]$ (FC)

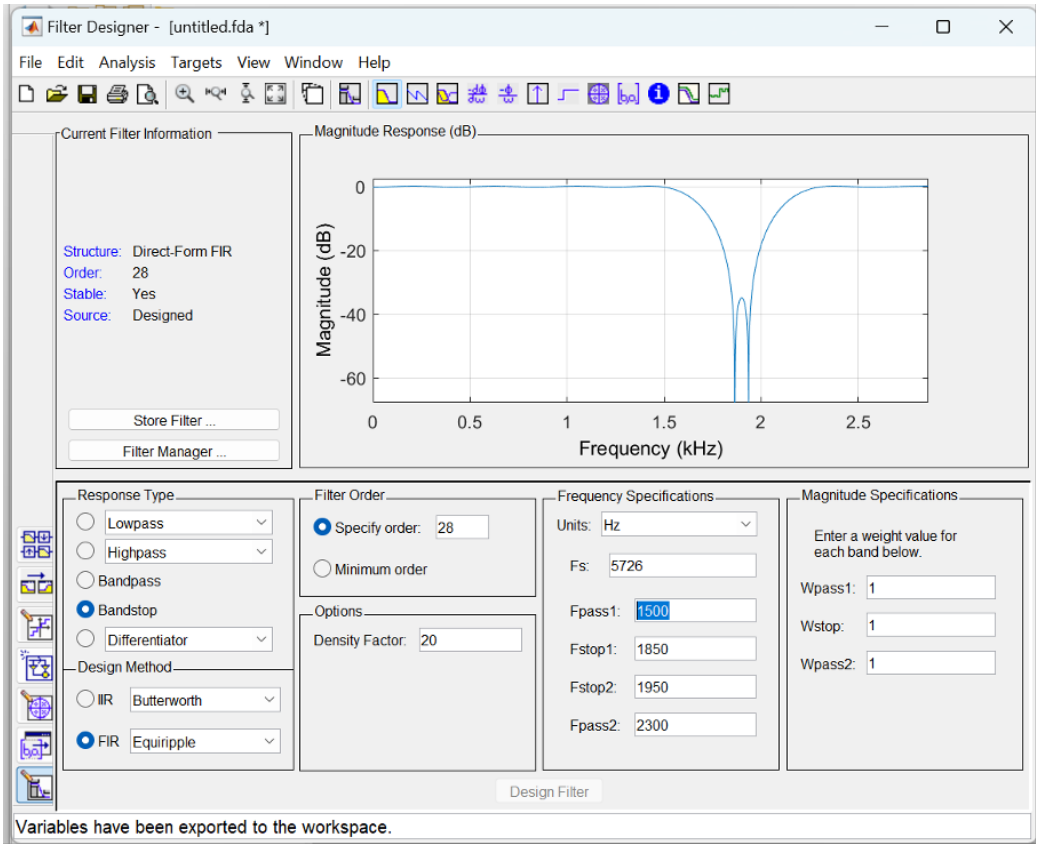

Figure 5: Filter for Signal $x_1[n]$ (AL)

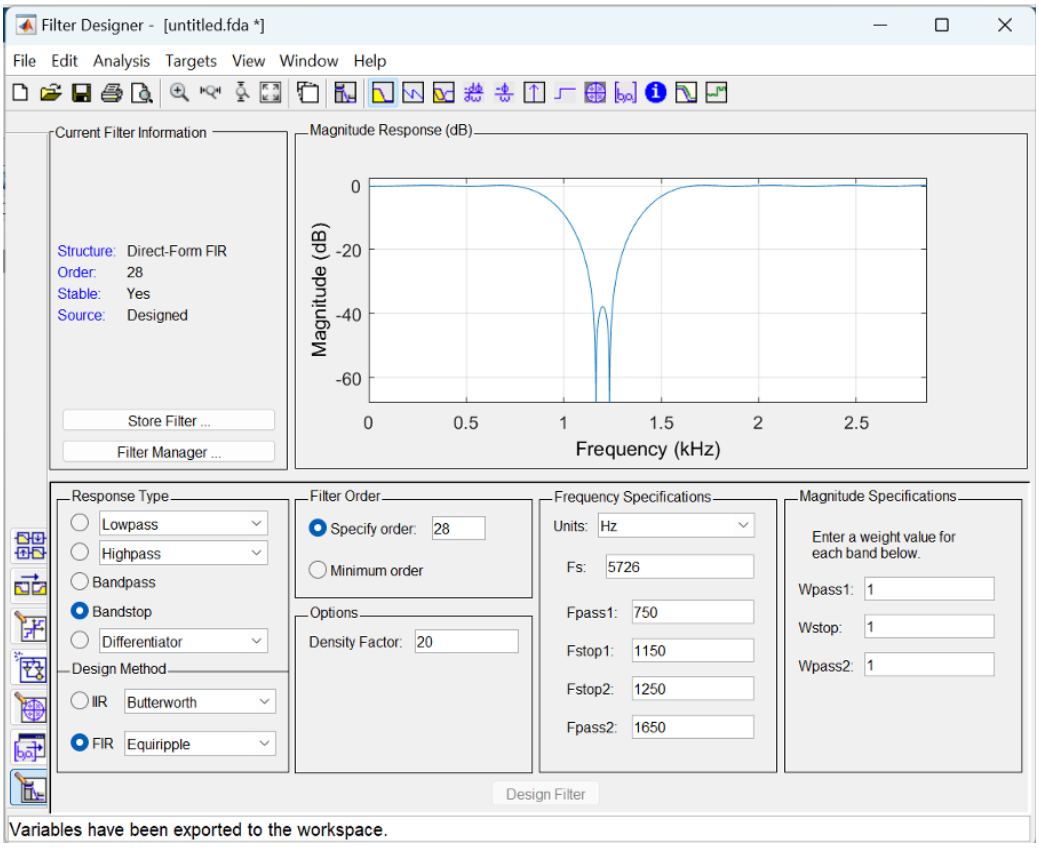

Figure 6: Filter for Signal $x_2[n]$ (FC)

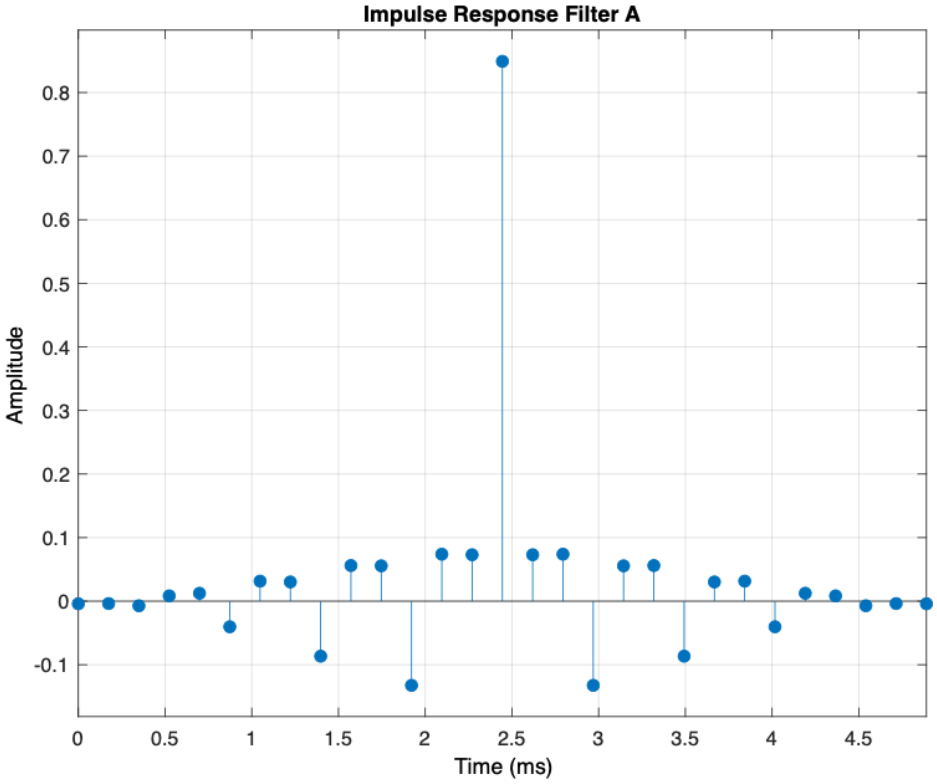

Figure 7: Impulse Response for Filter A

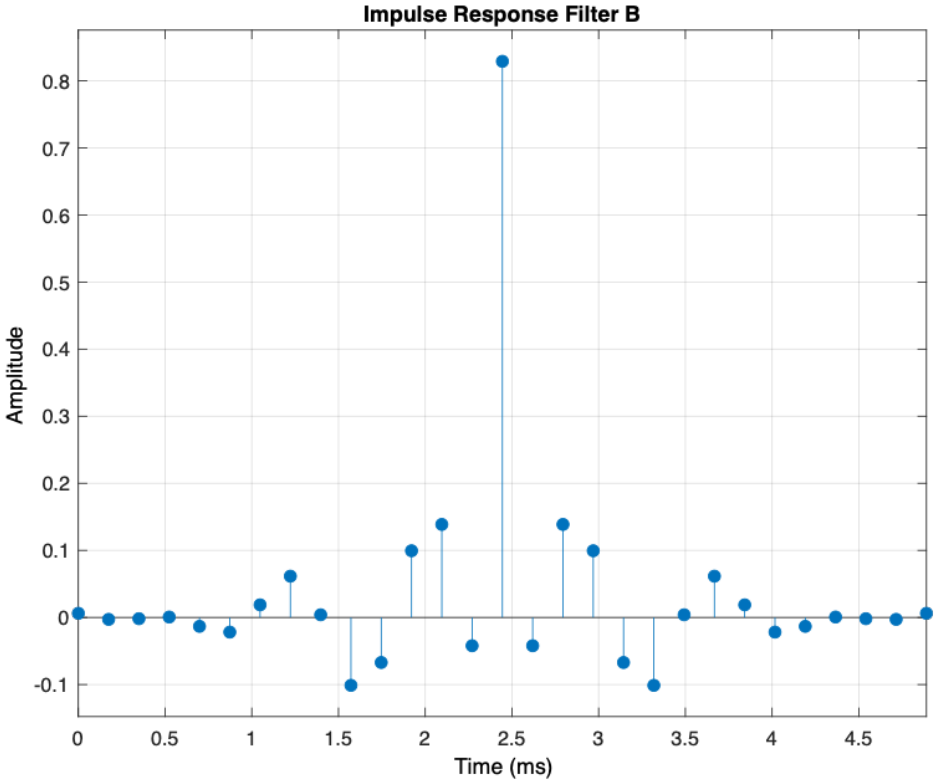

Figure 8: Impulse Response for Filter A

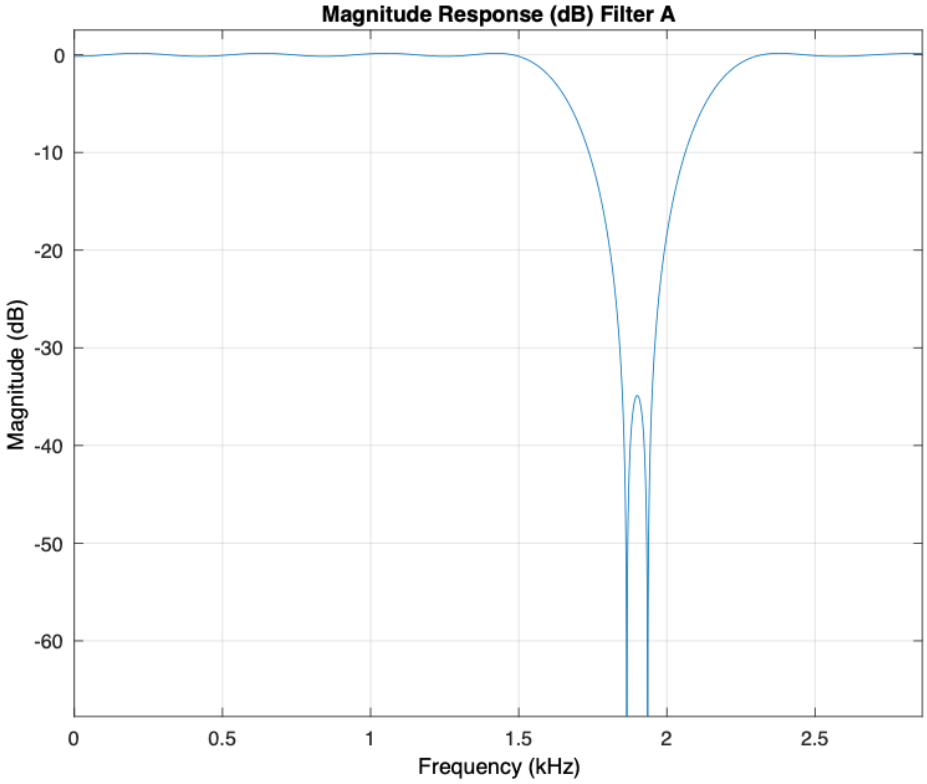

Figure 9: Magnitude Response for Filter A

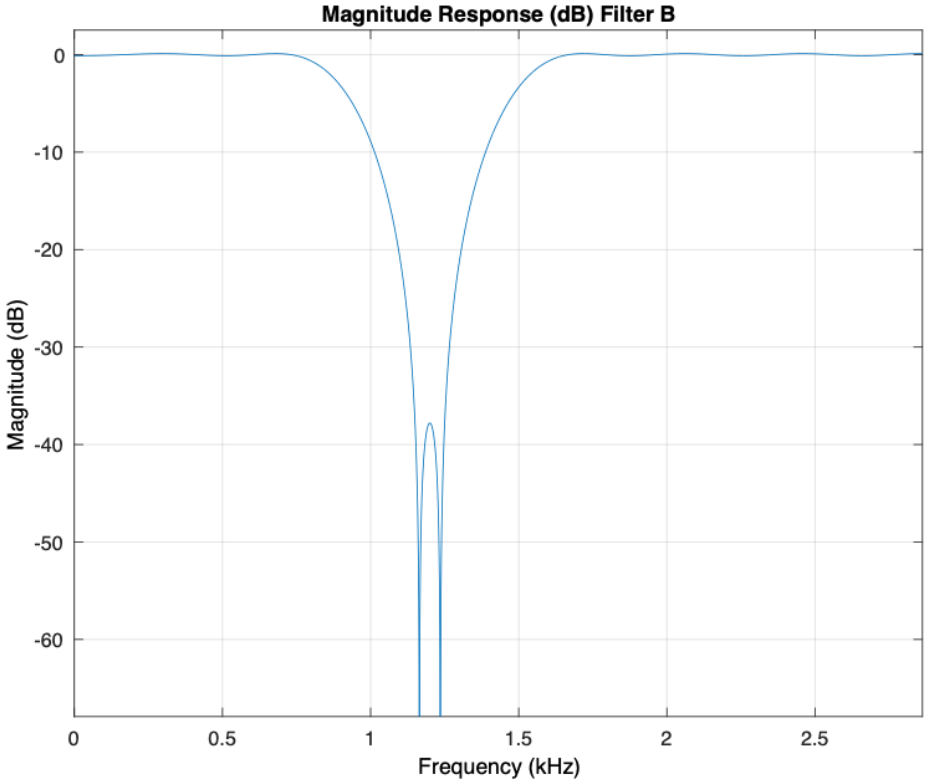

Figure 10: Magnitude Response for Filter A

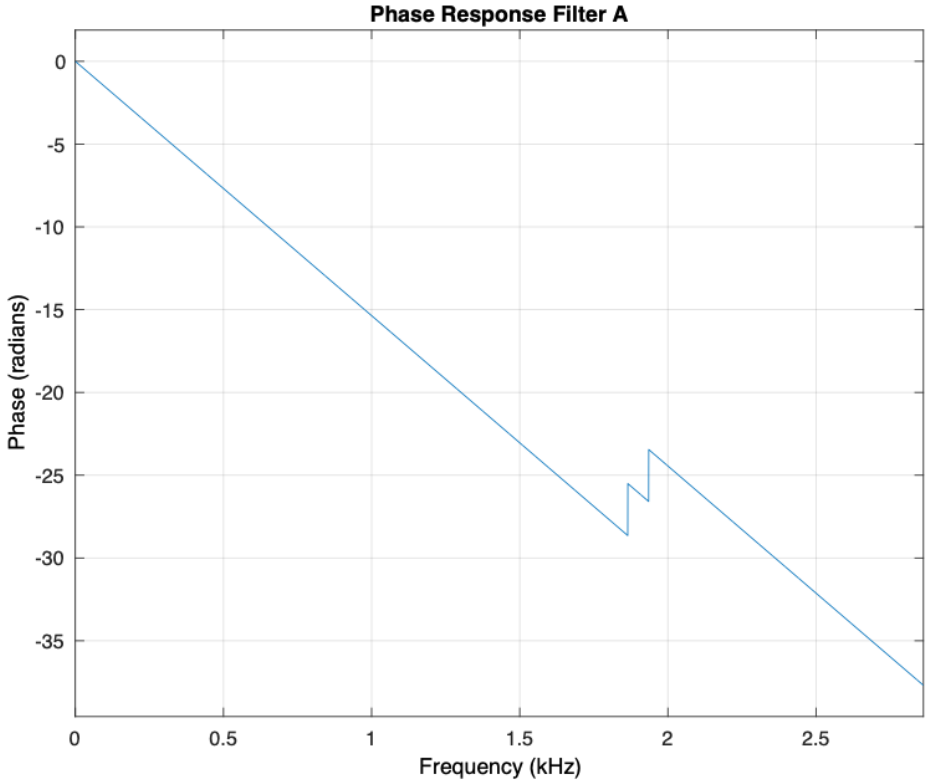

Figure 11: Phase Response for Filter A

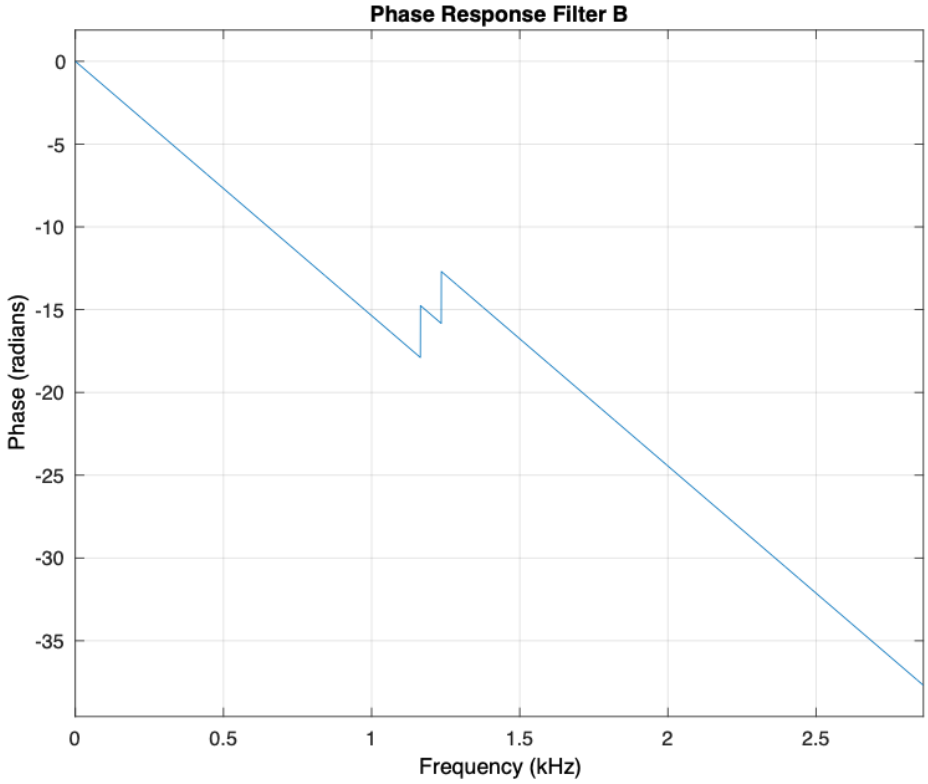

Figure 12: Phase Response for Filter A

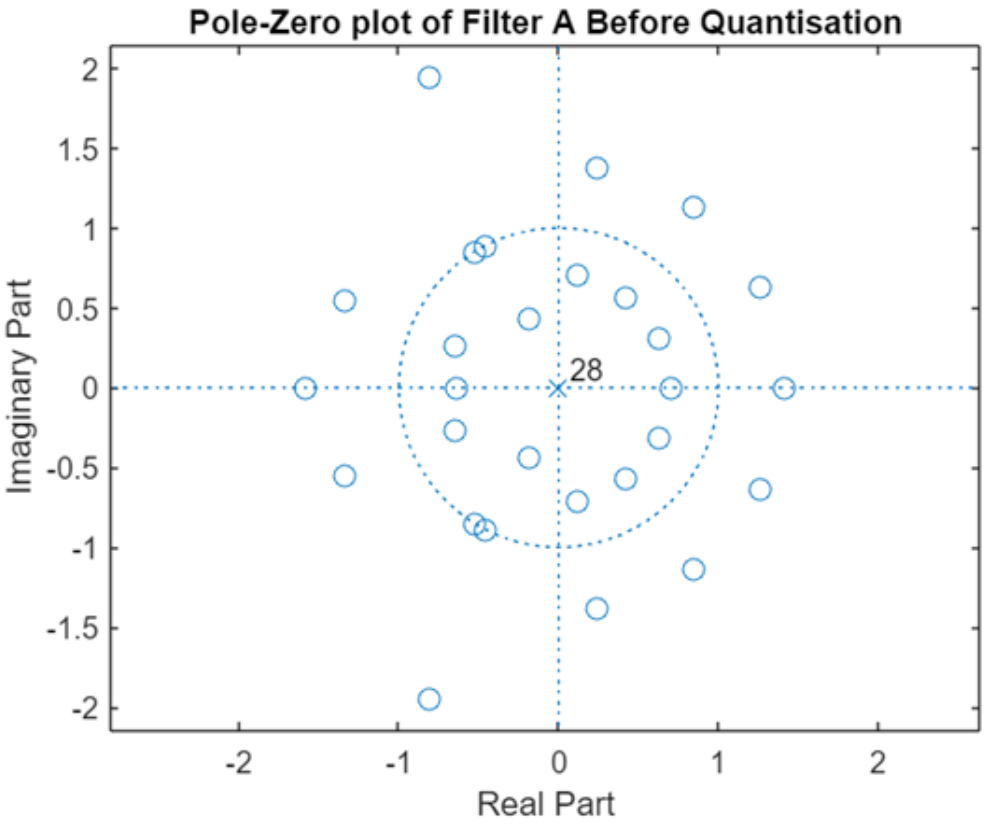

Figure 13: Filter A Z Plane Before Quantisation

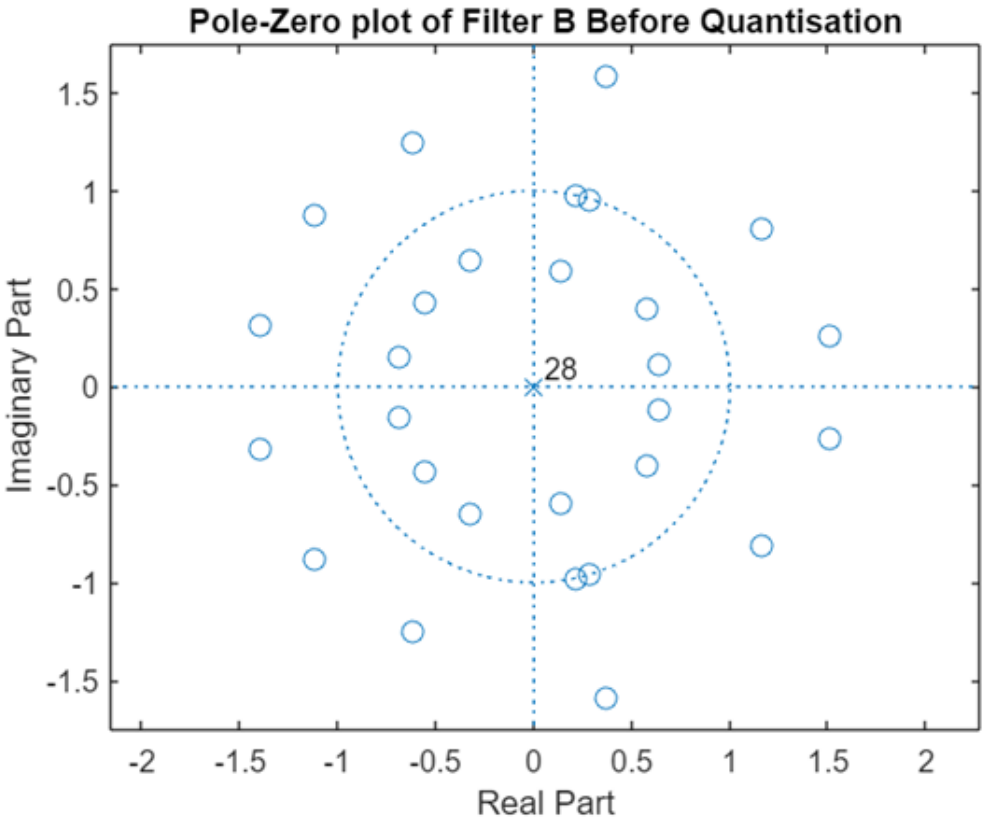

Figure 14: Filter B Z Plane Before Quantisation

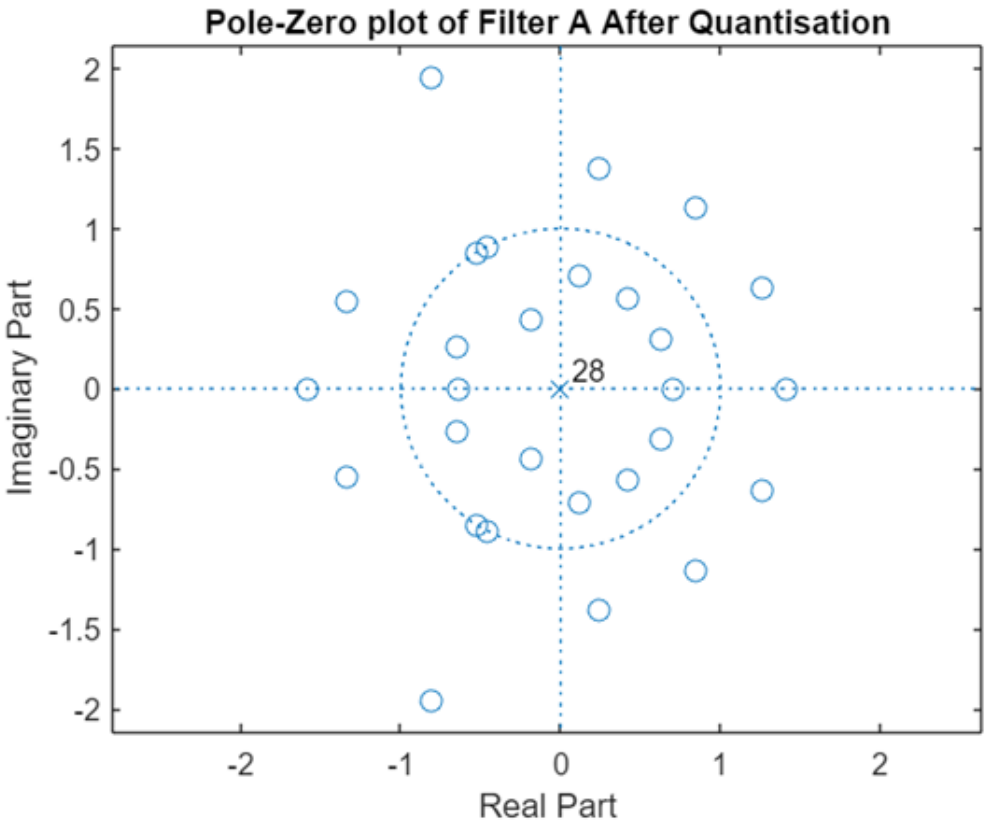

Figure 15: Filter A Z Plane After Quantisation

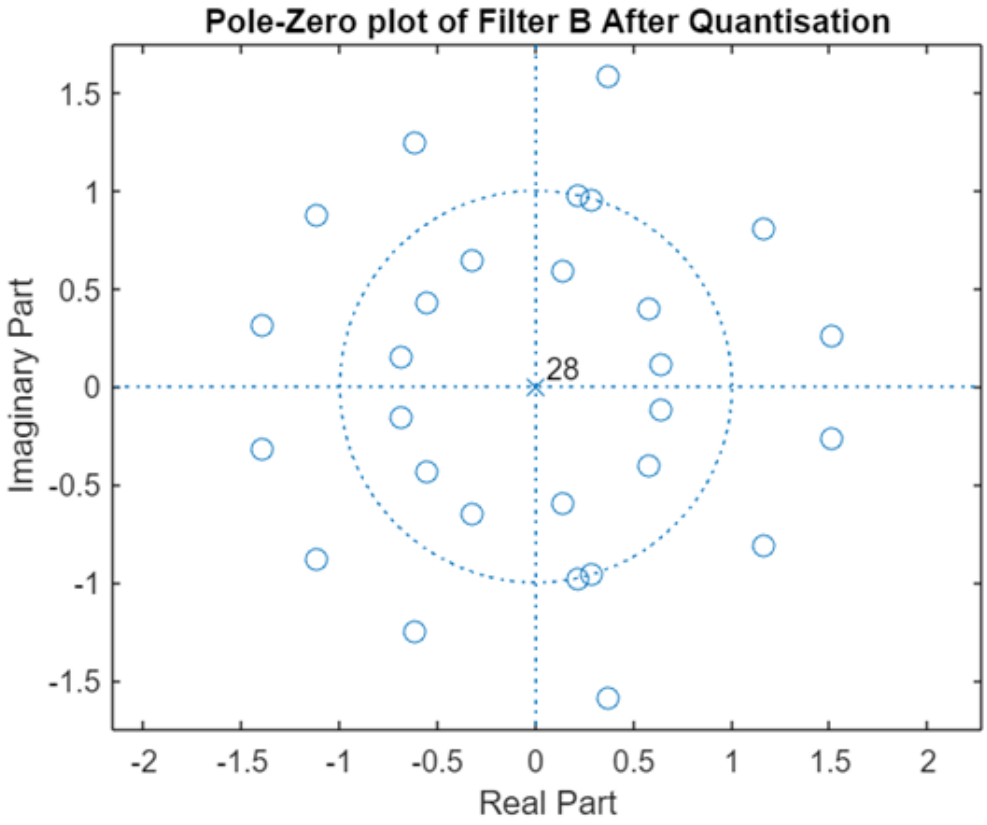

Figure 16: Filter B Z Plane After Quantisation

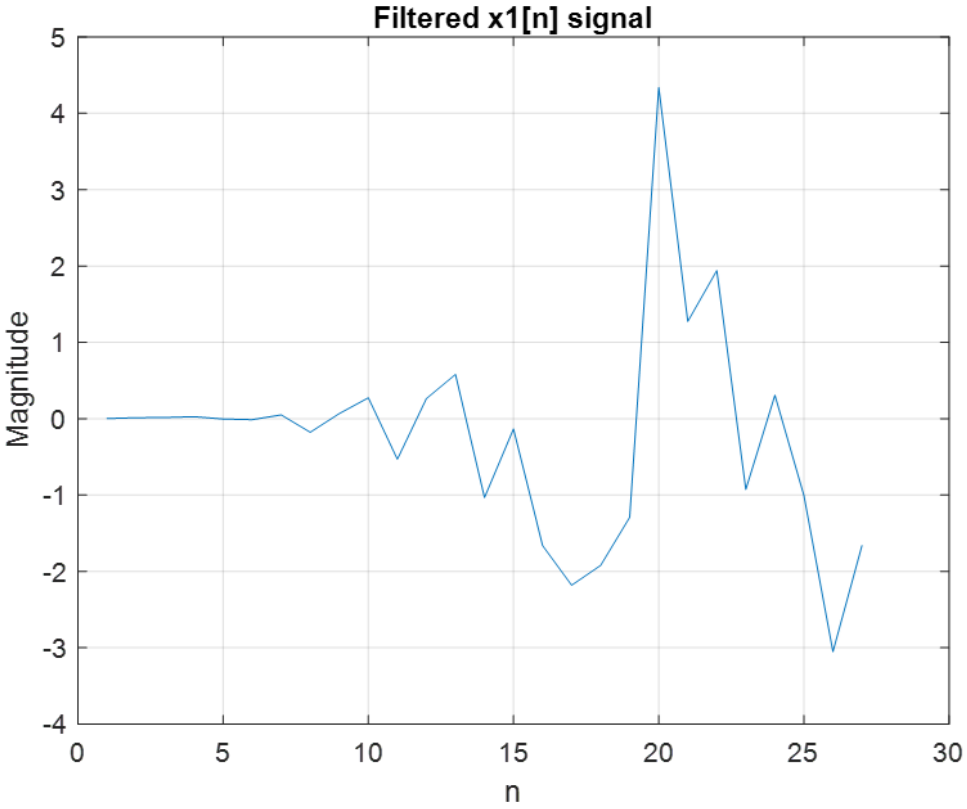

Figure 17: Filtered signal $x_1[n]$

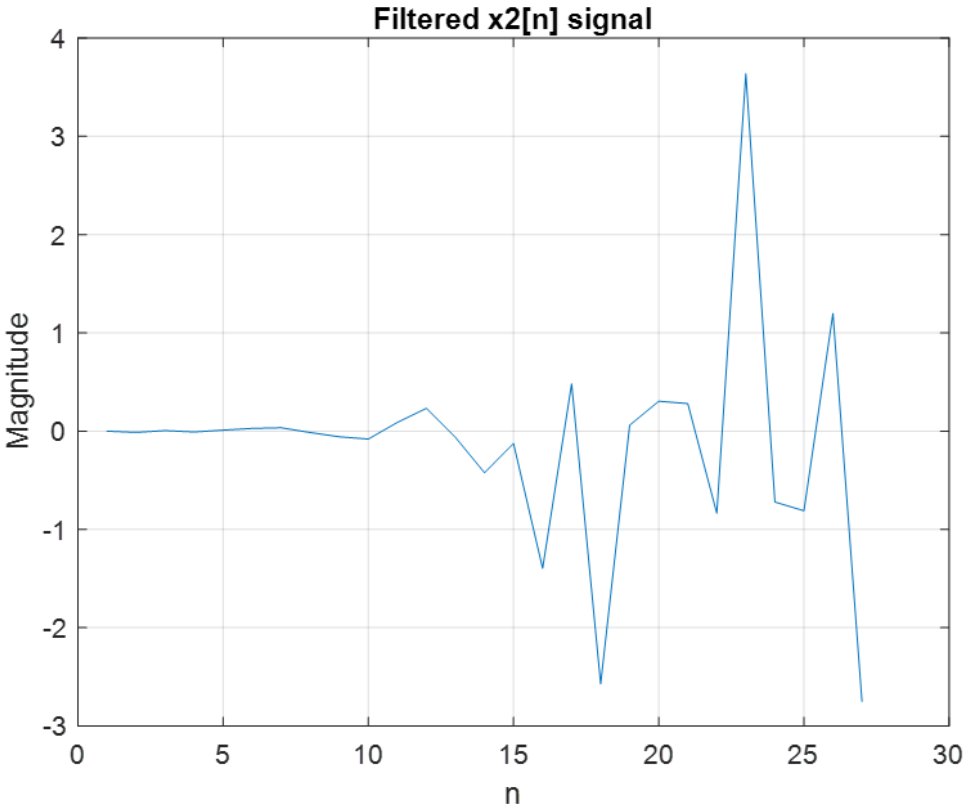

Figure 18: Filtered signal $x_2[n]$

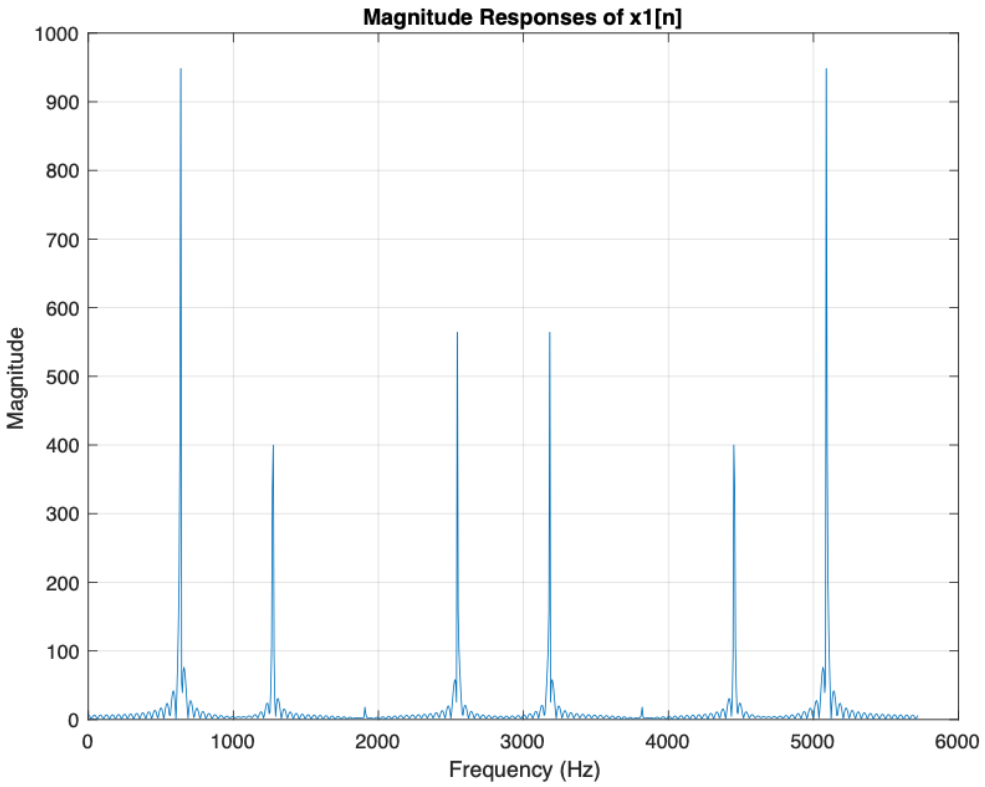

Figure 19: FFT of Filtered signal $x_1[n]$

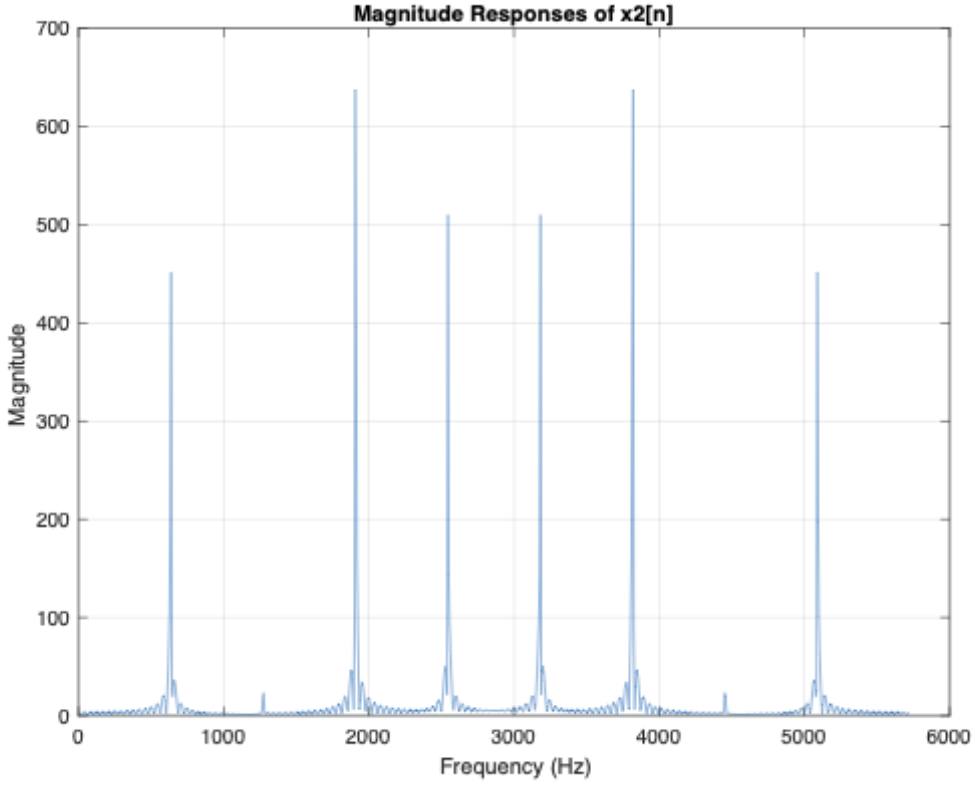

Figure 20: FFT of Filtered signal $x_2[n]$

```
% Generate random signal,
AL_ref = [2,0,2,0,0,9,3,3,4]; % Andrew's student number
FC_ref = [2,0,2,0,3,2,1,1,7]; % Frank's student number
x1 = repmat(AL_ref,1,100) % Repeat sequence 100 times
x1 = x1 - repmat(mean(x1),1,900) % Subtract mean to give signal
    zero mean
x2 = repmat(FC_ref,1,100) % Repeat sequence 100 times
x2 = x2 - repmat(mean(x2),1,900) % Subtract mean to give signal
    zero mean
fs = round((9334+2117)/2) % Determine sampling rate by finding
    average of ends of student #s

% Plot time signals
figure;
plot(1:27,x1(1:27)); % Plot the first 27 samples
title('x1[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;
figure;
plot(1:27,x2(1:27)); % Plot the first 27 samples
title('x2[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;

% Find magnitude reponse
N_FFT = 1024; % FFT size
X1 = fft(x1,N_FFT); % Calculate magnitude coefficients for x1
X2 = fft(x2,N_FFT); % Calculate magnitude coefficients for x2
f_axis = (0:N_FFT-1)*fs/N_FFT; % Frequency axis

% Plot magnitude
figure;
plot(f_axis,abs(X1));
title('Magnitude Responses of x1[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
figure;
plot(f_axis,abs(X2));
title('Magnitude Responses of x2[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

filt_x1 = filter(Num1, 1,x1); % Filter x1 with fiter 1's
    coefficients
filt_x2 = filter(Num2, 1,x2); % Filter x2 with fiter 2's
    coefficients

% Plot time signals
figure;
plot(1:27,filt_x1(1:27));
title('Filtered x1[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;
figure;
plot(1:27,filt_x2(1:27));
```

```
title('Filtered x2[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;

% Find magnitude reponse
N_FFT = 1024; % FFT size
X1 = fft(filt_x1,N_FFT); % Calculate magnitude coefficients for x1's filter coefficients
X2 = fft(filt_x2,N_FFT); % Calculate magnitude coefficients for x2's filter coefficients
f_axis = (0:N_FFT-1)*fs/N_FFT; % Frequency axis

% Plot magnitude
figure;
plot(f_axis,abs(X1));
title('Magnitude Responses of x1[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
figure;
plot(f_axis,abs(X2));
title('Magnitude Responses of x2[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

filter_a = single(Num1); % Set the precision to single
filter_b = single(Num2); % Set the precision to single

FC_filtered = conv(filter_b,x2,"full")' % Convolve x2 with it's filter
AL_filtered = conv(filter_a,x1,"full")' % Convolve x2 with it's filter

q = quantizer('single'); % Convert to single precision
filter_a_q = quantize(q, filter_a); % Quantise filter a coefficients
filter_b_q = quantize(q, filter_b); % Quantise filter b coefficients
H_a_q = dfilt.df2t(filter_a_q,1); % Direct Form II Transposed
H_b_q = dfilt.df2t(filter_b_q,1); % Direct Form II Transposed

figure;
zplane(filter_a);
title('Pole-Zero plot of Filter A Before Quantisation');
figure;
zplane(filter_a_q);
title('Pole-Zero plot of Filter A After Quantisation');

figure;
zplane(filter_b);
title('Pole-Zero plot of Filter B Before Quantisation');
figure;
zplane(filter_b_q);
title('Pole-Zero plot of Filter B After Quantisation');

figure;
subplot(1,2,1);
zplane(filter_a);
```

```
title('Pole-Zero plot Before Quantisation');
subplot(1,2,2);
zplane(filter_a_q);
title('Pole-Zero plot After Quantisation');
sgtitle('Filter A');

figure;
subplot(1,2,1);
zplane(filter_b);
title('Pole-Zero plot Before Quantisation');
subplot(1,2,2);
zplane(filter_b_q);
title('Pole-Zero plot After Quantisation');
sgtitle('Filter B');
```

Listing 1: Generation and Normalization of Signal

Appendix B

Part 2 Figures and code.

y	x			h		
[0]	[0]			[0]		
[1]	[0]	[1]		[1]	[0]	
[2]	[0]	[1]	[2]	[2]	[1]	[0]
[3]	[1]	[2]	[3]	[2]	[1]	
[4]	[2]	[3]		[2]		
[5]	[3]			[2]		

Figure 21: Pattern of Convolution

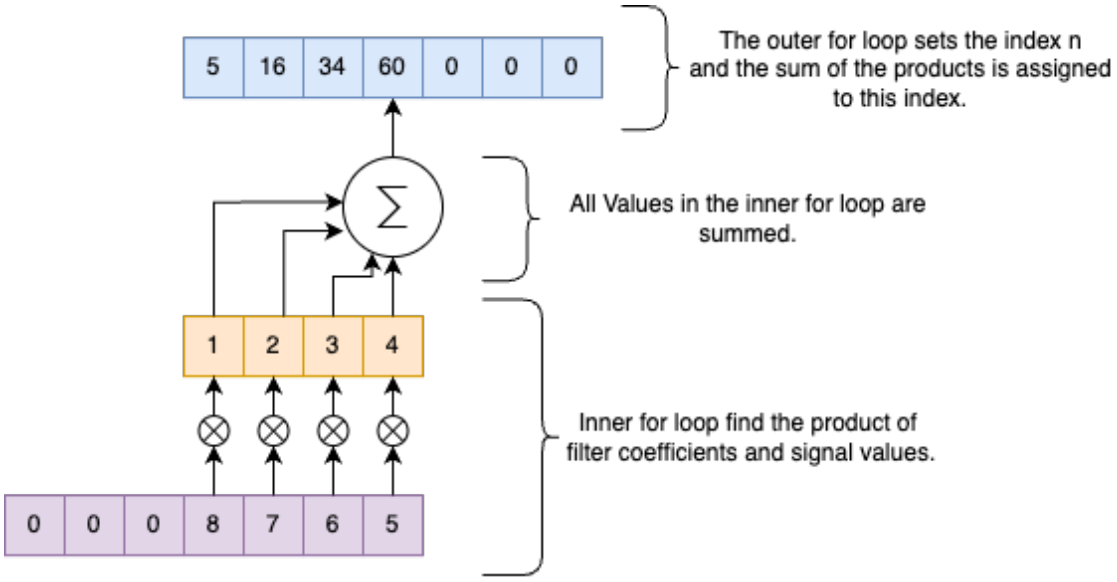

Figure 22: Convolution Sum Demonstration

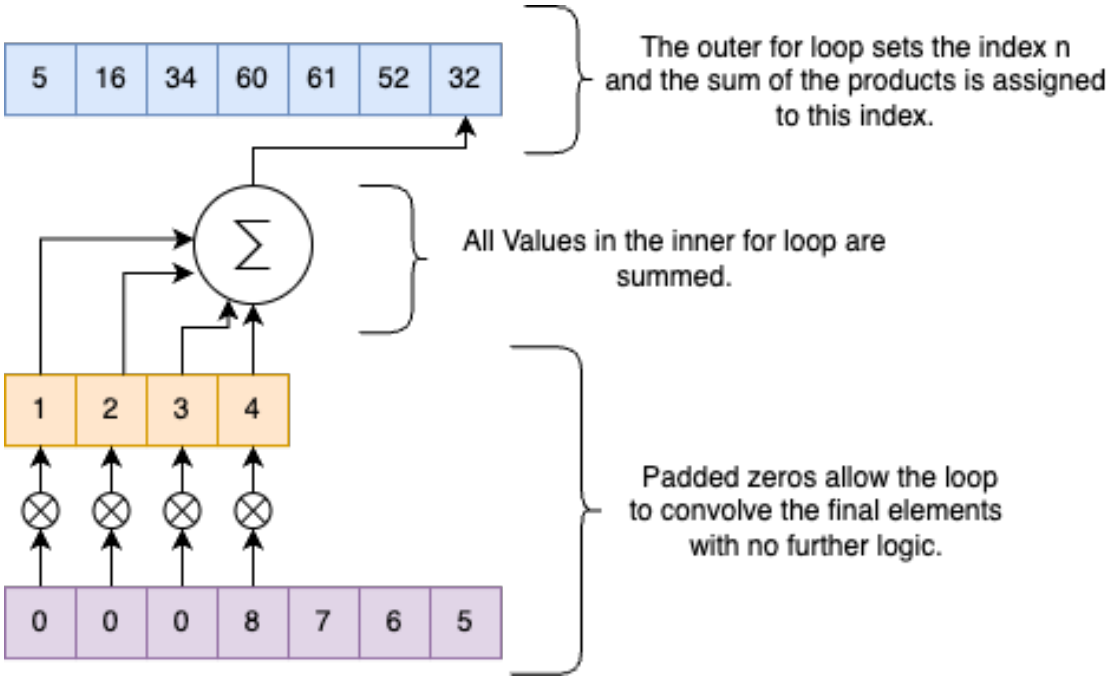

Figure 23: Zero padded Convolution

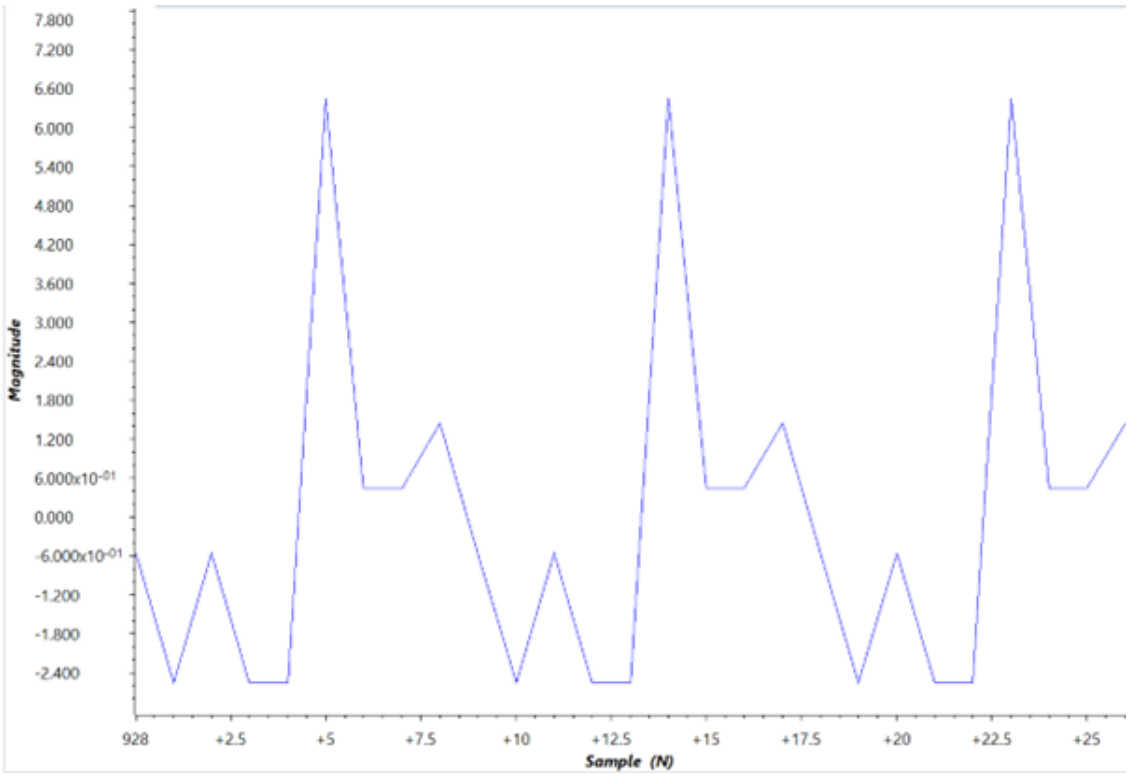

Figure 24: Time Domain of Zero Mean signal $x_1[n]$ (AL)

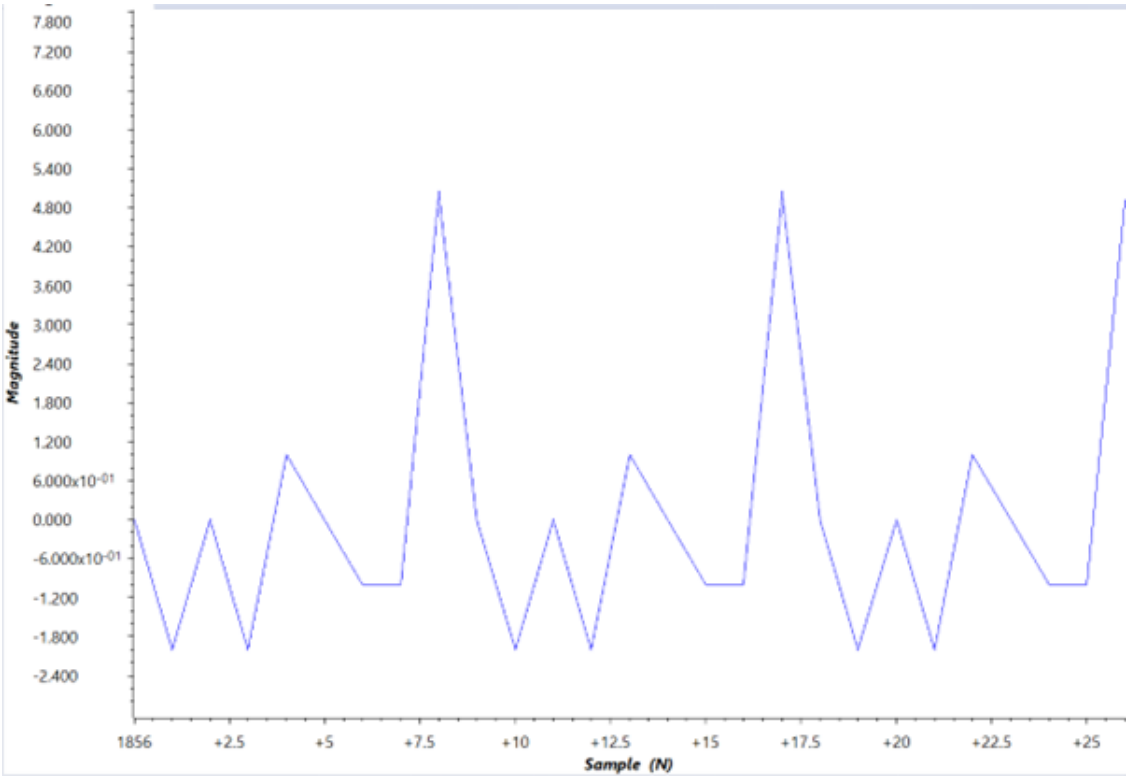

Figure 25: Time Domain of Zero Mean signal $x_2[n]$ (FC)

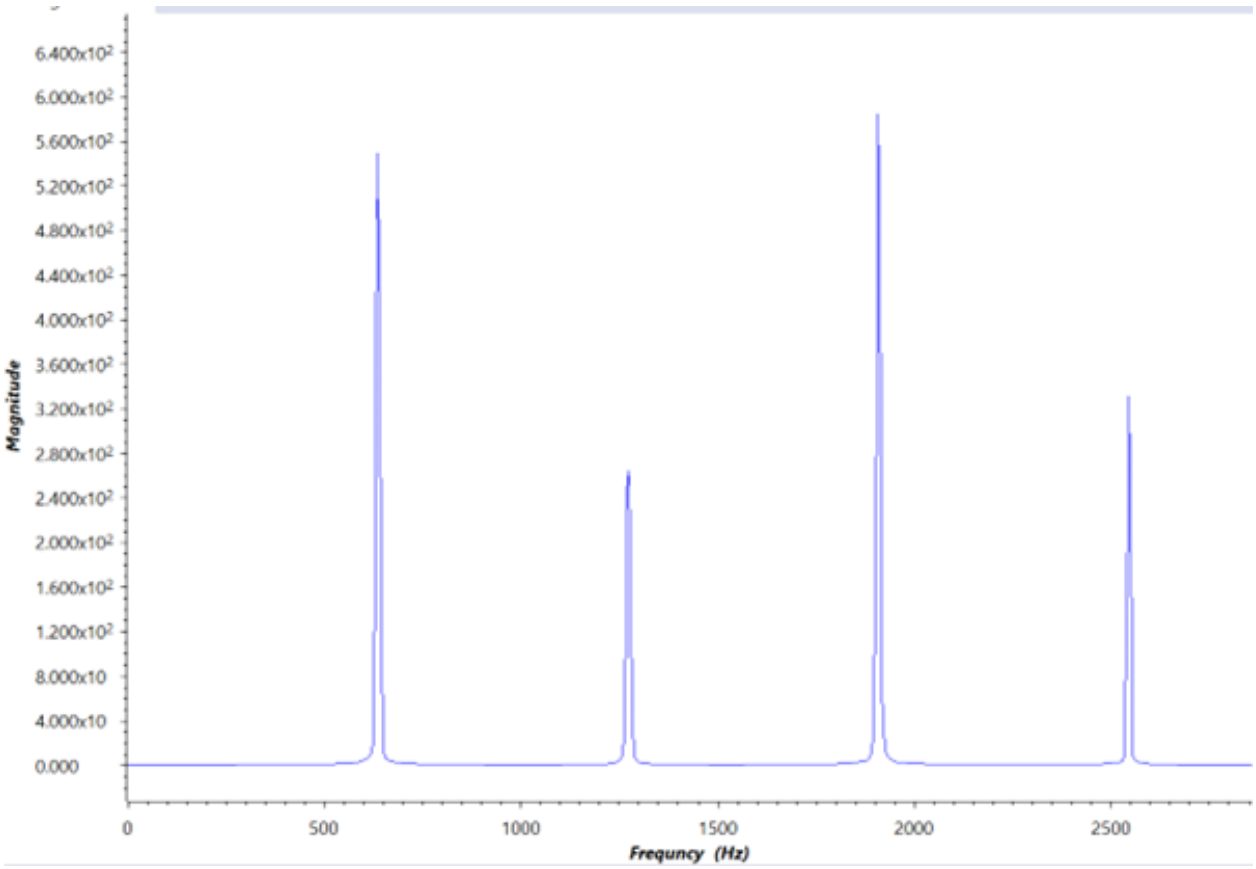

Figure 26: FFT of Zero Mean signal $x_1[n]$ (AL)

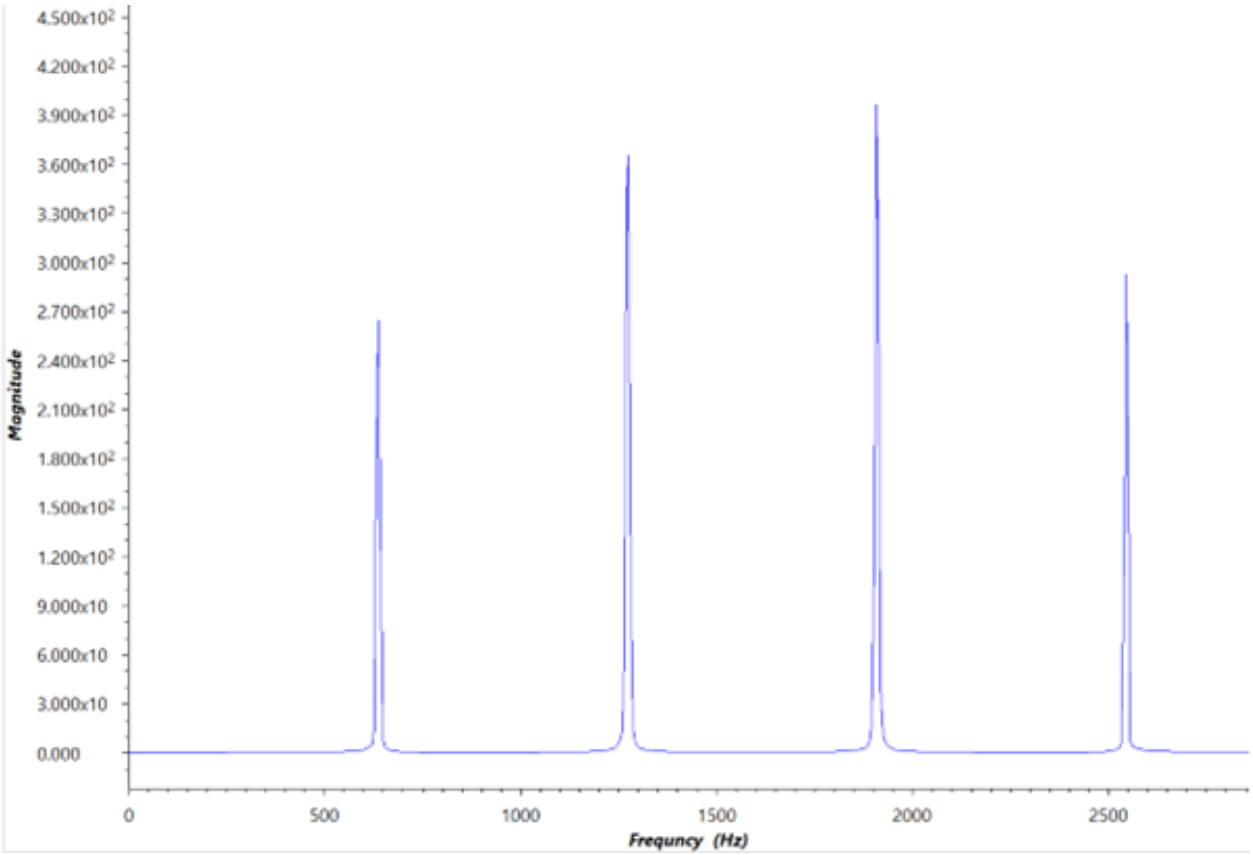

Figure 27: FFT of Zero Mean signal $x_2[n]$ (FC)

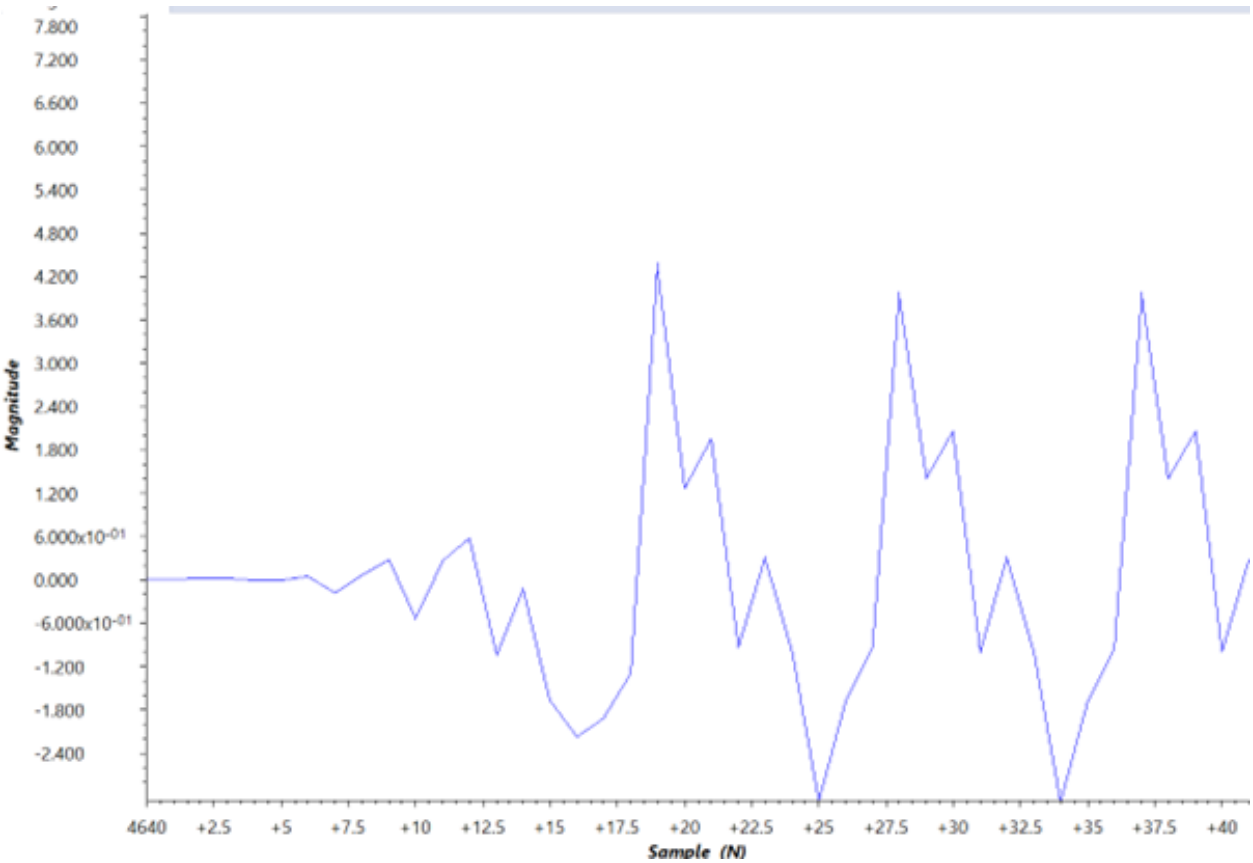

Figure 28: Time Domain of Filtered signal $x_1[n]$ (AL)

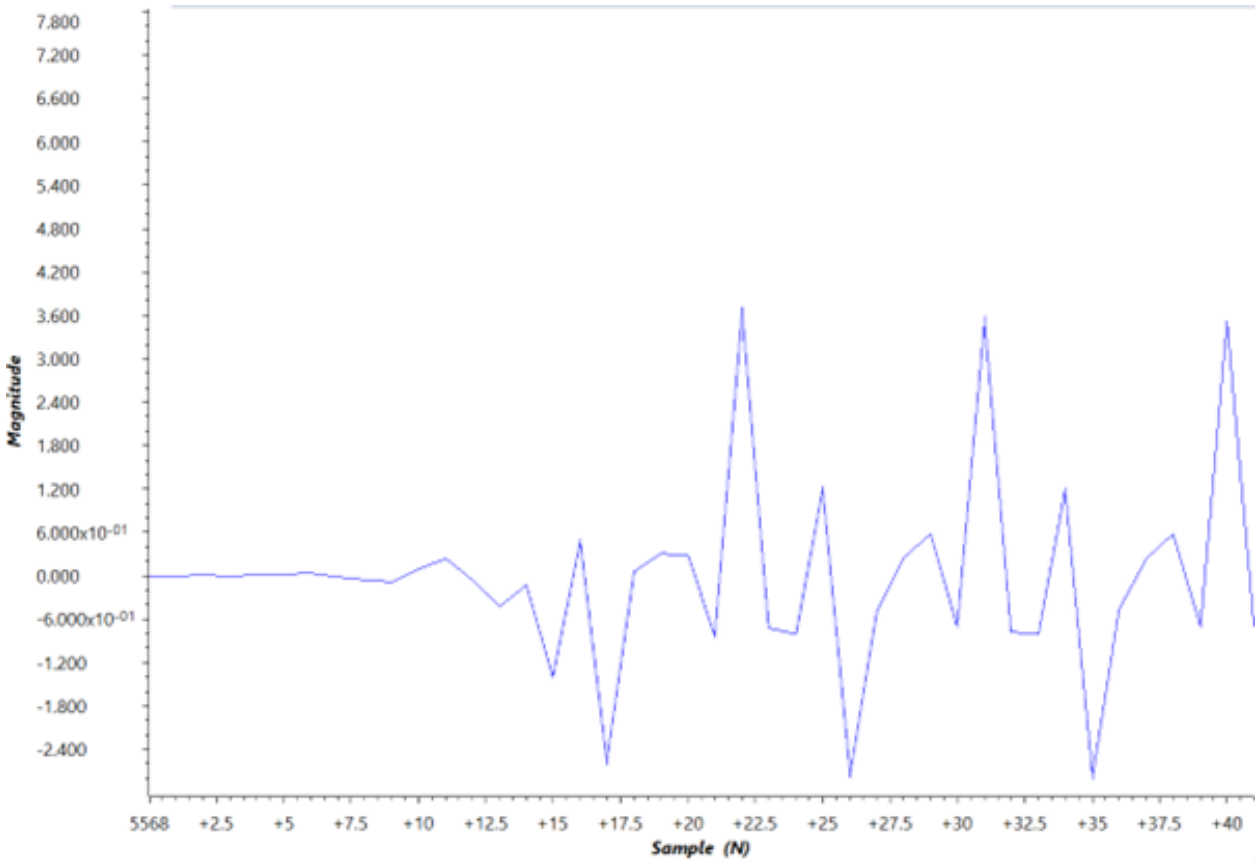

Figure 29: Time Domain of Filtered signal $x_2[n]$ (FC)

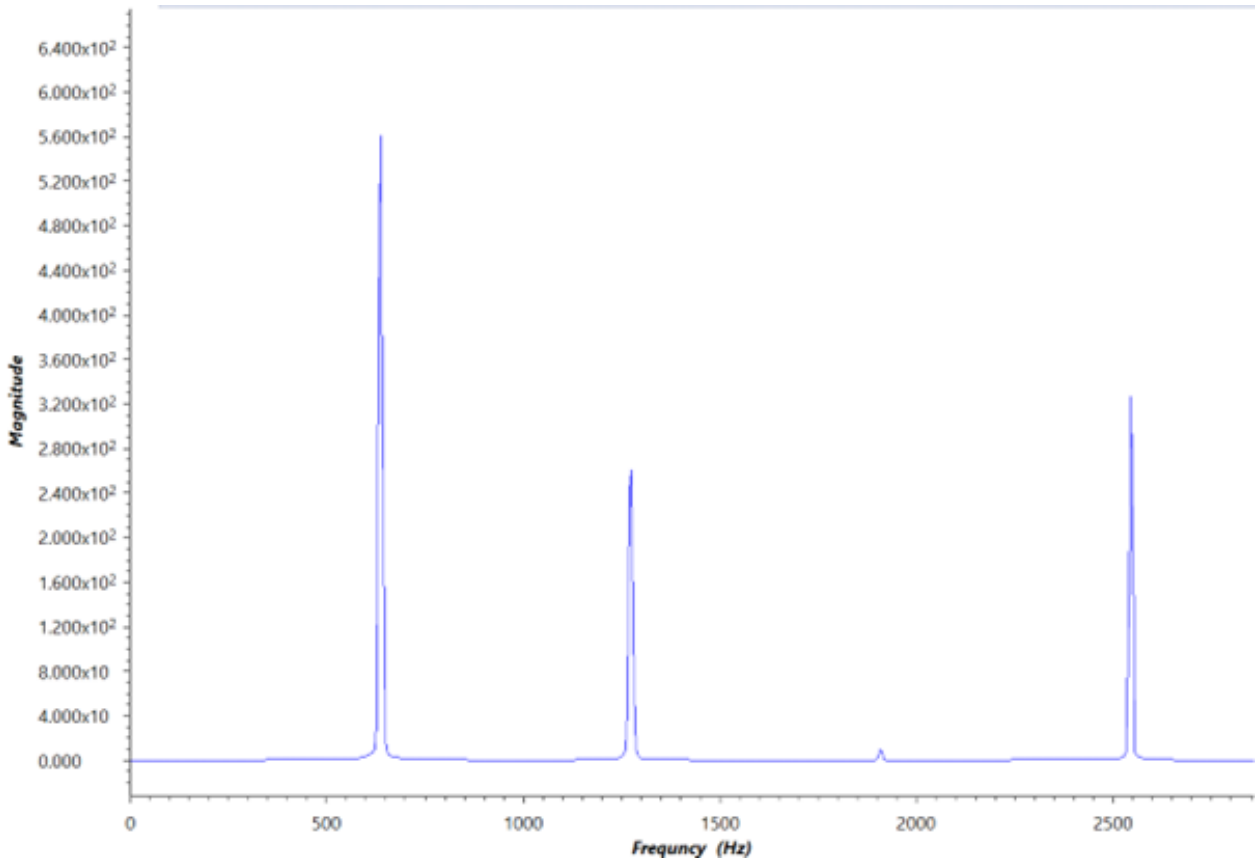

Figure 30: FFT of Filtered signal $x_1[n]$ (AL)

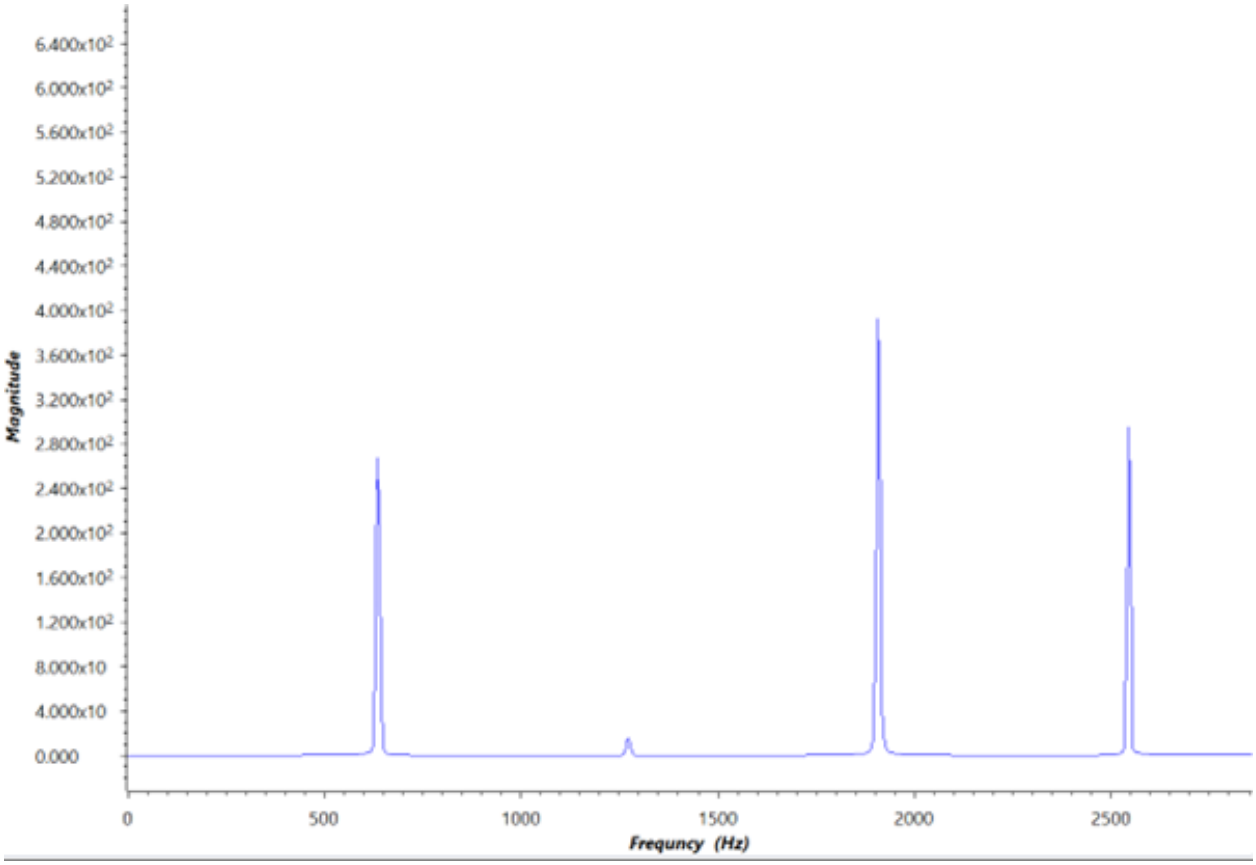

Figure 31: FFT of Filtered signal $x_2[n]$ (FC)

```
1
2 float filter_b[] = { 0.0061602f, -0.0028451f, -0.0018284f,
    0.0006535f, -0.0132398f, -0.0218456f, 0.0188277f, 0.0615422f,
    0.0040076f, -0.1012120f, -0.0671407f, 0.0994334f, 0.1386961f,
    -0.0421460f, 0.8290347f, -0.0421460f, 0.1386961f, 0.0994334f,
    -0.0671407f, -0.1012120f, 0.0040076f, 0.0615422f, 0.0188277f,
    -0.0218456f, -0.0132398f, 0.0006535f, -0.0018284f, -0.0028451f,
    0.0061602f };
3
4 #define N_FIR_A 29
5
6 float filter_a[] = { -0.0041270f, -0.0037774f, -0.0072290f,
    0.0083584f, 0.0124229f, -0.0403159f, 0.0314914f, 0.0301976f,
    -0.0865512f, 0.0560293f, 0.0555848f, -0.1324403f, 0.0738421f,
    0.0729773f, 0.8491320f, 0.0729773f, 0.0738421f, -0.1324403f,
    0.0555848f, 0.0560293f, -0.0865512f, 0.0301976f, 0.0314914f,
    -0.0403159f, 0.0124229f, 0.0083584f, -0.0072290f, -0.0037774f,
    -0.0041270f };
```

Listing 2: Filter Coefficients (data.h)

```
1 #include "data.h"
2 #include "stdio.h"
3
4 //indexes of for loops
5 int i, n, v;
6
7 //Both Student Numbers
8 float AL[9] = { 2.0, 0.0, 2.0, 0.0, 0.0 ,9.0 ,3.0 ,3.0 ,4.0 };
9 float FC[9] = { 2.0, 0.0, 2.0, 0.0, 3.0, 2.0, 1.0, 1.0, 7.0 };
10
11 // padded signal with 28 zeros
12 float AL_sig[928] = { 0.0 };
13 float FC_sig[928] = { 0.0 };
14
15 float AL_sig_mu[900+28] = { 0.0 };
16 float FC_sig_mu[900+28] = { 0.0 };
17
18 // calculate mean of student numbers
19 float AL_mu = 0.0;
20 float FC_mu = 0.0;
21
22 // length of filtered signal is signal length + filter length -1
23 float filtered_AL[928];
24 float filtered_FC[928];
25
26 // Write to file
27 FILE *filePtr;
28
29
30 void main()
31 {
32
33     // calculate the mean of both signals
34     for(i = 0; i<9; i++){
35         AL_mu  += AL[i];
36         FC_mu  += FC[i];
37     }
38
39     AL_mu /= 9.0;
40     FC_mu /= 9.0;
```

```
41
42      // signal minus mean so signal is zero mean
43      for(i = 0; i<900; i++){
44              AL_sig_mu[i] = AL[i%9] - AL_mu;
45              FC_sig_mu[i] = FC[i%9] - FC_mu;
46
47              AL_sig[i] = AL[i%9];
48              FC_sig[i] = FC[i%9];
49      }
50
51      // pad signals with 28 zeros
52      for(i = 900; i<928; i++){
53              AL_sig_mu[i] = 0;
54              FC_sig_mu[i] = 0;
55              AL_sig[i] = 0;
56              FC_sig[i] = 0;
57      }
58
59      // Convolve signals
60      for(n = 0; n<928; n++){
61          filtered_FC[n] = 0; // set all values to zero initally
62          filtered_AL[n] = 0;
63          // difference equation
64          for(v = 0; (v<=n)&&(v<29); v++){
65              filtered_FC[n] += FC_sig_mu[n-v] * filter_b[v];
66              filtered_AL[n] += AL_sig_mu[n-v] * filter_a[v];
67          }
68      }
69
70      // Write to file
71
72      filePtr = fopen("filter_FC.txt","w");
73      for(i =0; i < 928; i++){
74          fprintf(filePtr,"%lf\n",filtered_FC[i]);
75      }
76      fclose(filePtr);
77
78      filePtr = fopen("filter_AL.txt","w");
79      for(i =0; i < 928; i++){
80          fprintf(filePtr,"%lf\n",filtered_AL[i]);
81      }
82      fclose(filePtr);
83
84      return ;
85  }
```

Listing 3: Generation and Filtering of Signals (main.c)

Difference Equation Implementation

```
0x80000EA8 x
0x80000EA8 1           2           3           4
0x80000EB8 b
0x80000EB8 5           6           7           8           0           0           0           2906717708
0x80000ED8 y
0x80000ED8 3563838009  4294897963  4208440574  2080153595  2410270030  3880430674  3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4025465055
0x80000F0C m
0x80000F0C 56726999
0x80000F10 a
0x80000F10 1
```

Figure 32: Initial Memory

```
0x80000EA8 x
0x80000EA8 1           2           3           4
0x80000EB8 b
0x80000EB8 5           6           7           8           0           0           0           2906717708
0x80000ED8 y
0x80000ED8 0           4294897963  4208440574  2080153595  2410270030  3880430674  3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 0
0x80000F0C m
0x80000F0C 56726999
0x80000F10 a
0x80000F10 1
```

Figure 33: Memory After 1 Iteration

```
0x80000EA8 x
0x80000EA8 1           2           3           4
0x80000EB8 b
0x80000EB8 5           6           7           8           0           0           0           2906717708
0x80000ED8 y
0x80000ED8 5           4294897963  4208440574  2080153595  2410270030  3880430674  3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 0
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 34: Memory After 2 Iteration

```
0x80000EA8 x
0x80000EA8 1           2           3           4
0x80000EB8 b
0x80000EB8 5           6           7           8           0           0           0           2906717708
0x80000ED8 y
0x80000ED8 5           0           4208440574  2080153595  2410270030  3880430674  3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 1
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 35: Memory After 3 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          6          4208440574 2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 1
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 36: Memory After 4 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         4208440574 2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 1
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 37: Memory After 5 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         0          2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 2
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 38: Memory After 6 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         7          2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 2
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 39: Memory After 7 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         19         2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 2
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 40: Memory After 8 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         2080153595 2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 2
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 41: Memory After 9 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         0          2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 3
0x80000F0C m
0x80000F0C 3
0x80000F10 a
0x80000F10 1
```

Figure 42: Memory After 10 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         8          2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 3
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 43: Memory After 11 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         22         2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 3
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 44: Memory After 12 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         40         2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 3
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 45: Memory After 13 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         2410270030 3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 3
0x80000F0C m
0x80000F0C 3
0x80000F10 a
0x80000F10 1
```

Figure 46: Memory After 14 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         0          3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4
0x80000F0C m
0x80000F0C 4
0x80000F10 a
0x80000F10 1
```

Figure 47: Memory After 15 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         0          3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 48: Memory After 16 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         16         3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 49: Memory After 17 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         37         3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 50: Memory After 18 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         3880430674 3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 4
0x80000F0C m
0x80000F0C 3
0x80000F10 a
0x80000F10 1
```

Figure 51: Memory After 19 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         0          3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 5
0x80000F0C m
0x80000F0C 4
0x80000F10 a
0x80000F10 1
```

Figure 52: Memory After 20 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         0          3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 5
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 53: Memory After 21 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         0          3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 5
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 54: Memory After 22 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         24         3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 5
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 55: Memory After 23 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         52         3219717499

0x80000F08 .bss, __bss, bss, n
0x80000F08 5
0x80000F0C m
0x80000F0C 3
0x80000F10 a
0x80000F10 1
```

Figure 56: Memory After 24 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         52         0

0x80000F08 .bss, __bss, bss, n
0x80000F08 6
0x80000F0C m
0x80000F0C 4
0x80000F10 a
0x80000F10 1
```

Figure 57: Memory After 25 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         52         0

0x80000F08 .bss, __bss, bss, n
0x80000F08 6
0x80000F0C m
0x80000F0C 0
0x80000F10 a
0x80000F10 1
```

Figure 58: Memory After 26 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         52         0

0x80000F08 .bss, __bss, bss, n
0x80000F08 6
0x80000F0C m
0x80000F0C 1
0x80000F10 a
0x80000F10 1
```

Figure 59: Memory After 27 Iteration

```
0x80000EA8 x
0x80000EA8 1          2          3          4
0x80000EB8 b
0x80000EB8 5          6          7          8          0          0          0          2906717708
0x80000ED8 y
0x80000ED8 5          16         34         60         61         52         0

0x80000F08 .bss, __bss, bss, n
0x80000F08 6
0x80000F0C m
0x80000F0C 2
0x80000F10 a
0x80000F10 1
```

Figure 60: Memory After 28 Iteration

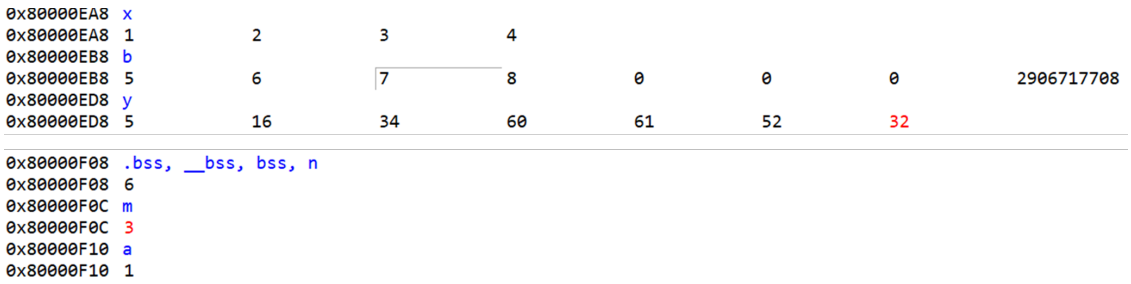

Figure 61: Memory After 29 Iteration

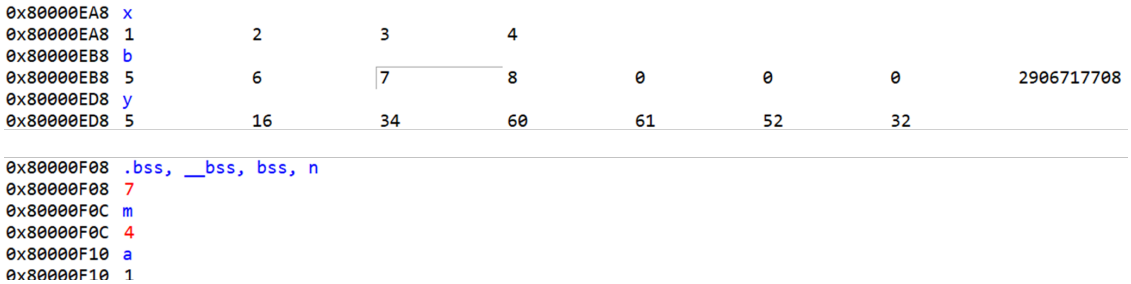

Figure 62: Memory After 30 Iteration

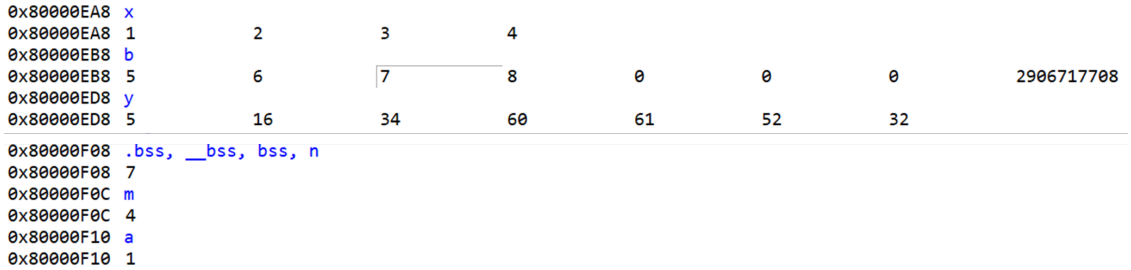

Figure 63: Memory After 31 Iteration

```
// signal x and filter b padded with 3 zeros
int x[4] = { 1, 2, 3, 4 };
int b[7] = { 5, 6, 7, 8, 0, 0, 0 };
// denominator = 1
int a = 1;
// ouput of conv = 7
int y[7];

void main()
{

//Convolve signal and filter
    for(n = 0; n<7; n++){
        y[n] = 0; // set ouput to zero initally
        for(m = 0; (m<=n)&&(m<4); m++){
            // for each input x[m] multiply for
            // filter coefficient and divide by denominator
            // sum result to ouput term
            y[n] += (b[n-m] * x[m])/a;
        }
    }
    return ;
}
```

Listing 4: Convoltuion Using Difference equations

Appendix C

Part 3 Figures and code.

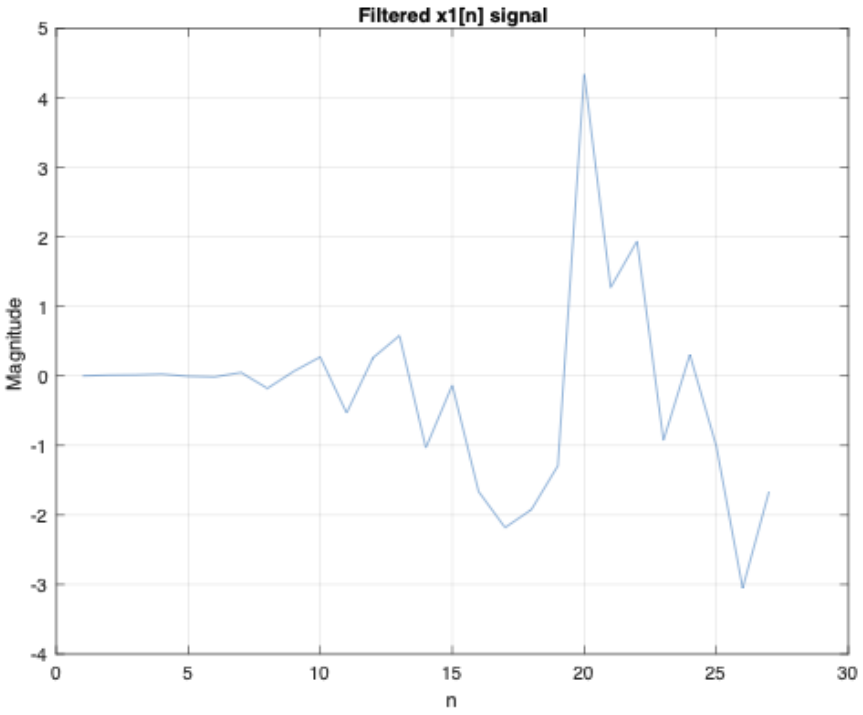

Figure 64: Time Domain of Signal $x_1[n]$ (AL) Filtered in CCS

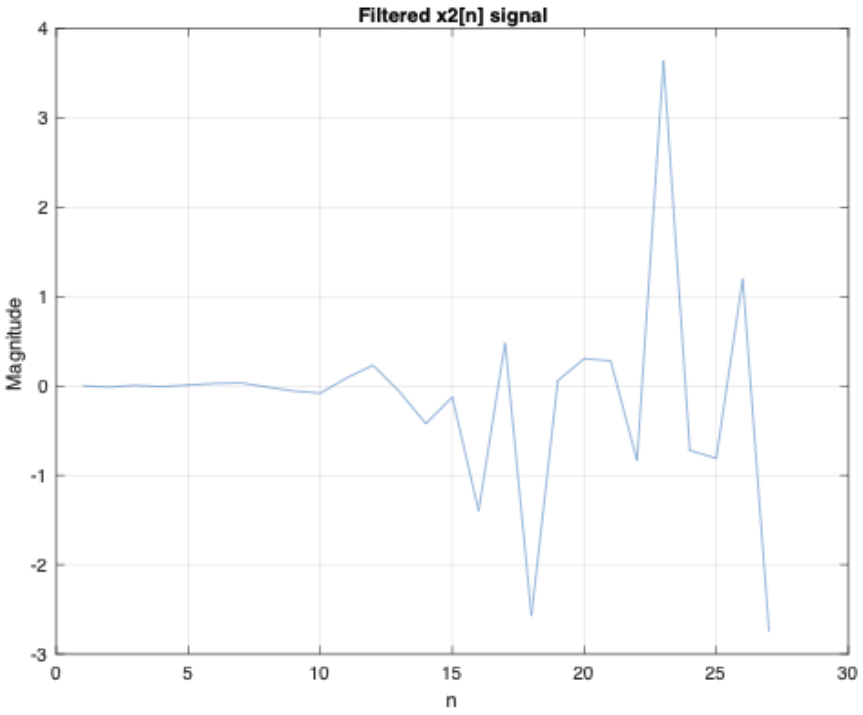

Figure 65: Time Domain of Signal $x_2[n]$ (FC) Filtered in CCS

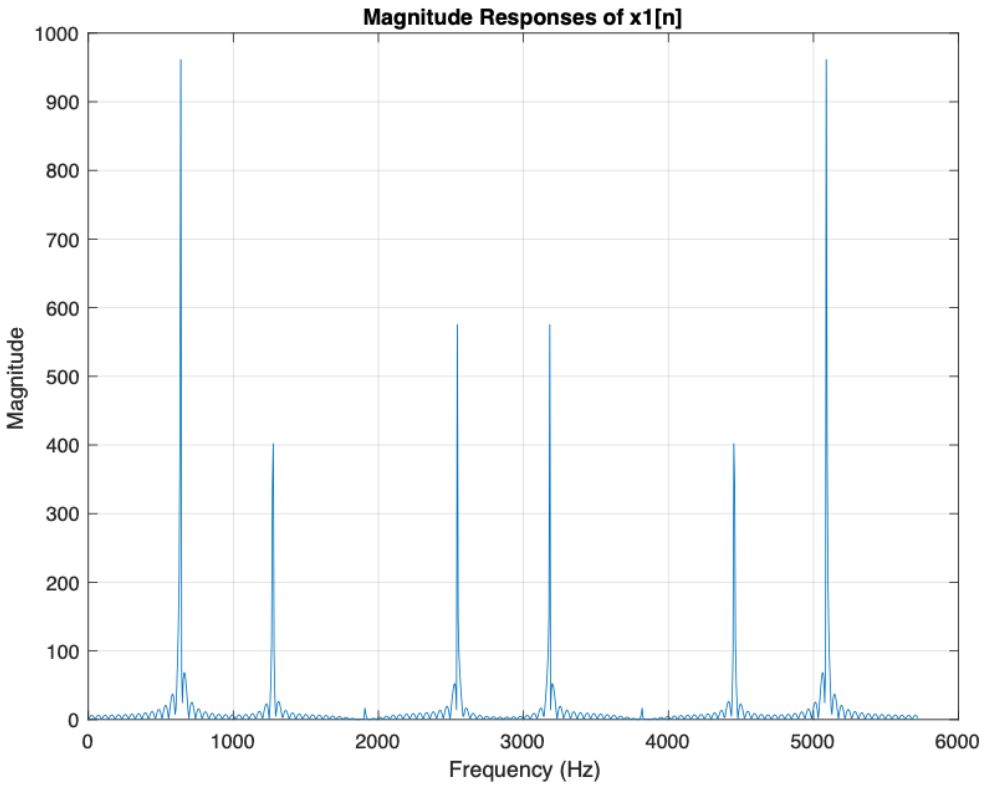

Figure 66: FFT of Signal $x_1[n]$ (AL) Filtered in CCS

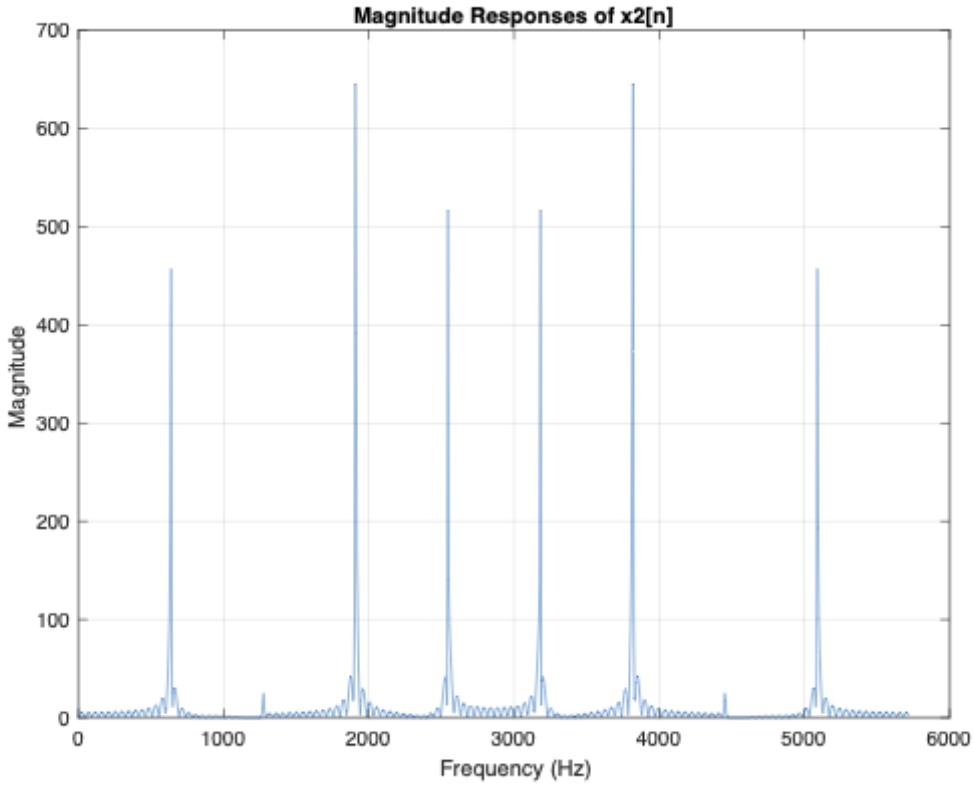

Figure 67: FFT of Signal $x_2[n]$ (FC) Filtered in CCS

```
AL_ccs = readmatrix('Assigment_1_FIR/Debug/filter_AL.txt'); % Read
    filtered output from CCS
FC_ccs = readmatrix('Assigment_1_FIR/Debug/filter_FC.txt'); % Read
    filtered output from CCS

diff_FCdB = 10*log10(sum((FC_ccs-FC_filtered).^2)) % Calculate SSE
    for filtered x2 in log
diff_ALdB = 10*log10(sum((AL_ccs-AL_filtered).^2)) % Calculate SSE
    for filtered x1 in log

diff_FC =(sum((FC_ccs-FC_filtered).^2)) % Calculate SSE for
    filtered x2
diff_AL =(sum((AL_ccs-AL_filtered).^2)) % Calculate SSE for
    filtered x1

% Find magnitude reponse
N_FFT = 1024; % FFT size
Filtered_X1 = fft(AL_ccs,N_FFT); % Compute magnitude coefficients
Filtered_X2 = fft(FC_ccs,N_FFT); % Compute magnitude coefficients
f_axis = (0:N_FFT-1)*fs/N_FFT; % Frequency axis

% Plot magnitude
figure;
plot(f_axis,abs(Filtered_X1));
title('Magnitude Responses of x1[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
figure;
plot(f_axis,abs(Filtered_X2));
title('Magnitude Responses of x2[n]');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Plot time signals
figure;
plot(1:27,AL_ccs(1:27));
title('Filtered x1[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;
figure;
plot(1:27,FC_ccs(1:27));
title('Filtered x2[n] signal')
xlabel('n');
ylabel('Magnitude');
grid on;
```

Listing 5: Comparison of Signals Filtered in Matlab and CCS

References

[1] Weiss, Stephan. "*ADSP Handouts*". "4.6 *PSD and Innovation Filter*", P114

[2] Caterina, Gaetano Di. "*DSP Implementation*". "*Digital Filters*", P3

Contribution

Both students agree that equal contribution was provided as both students undertook all parts of the assignment together on campus. Off campus, both students wrote the report.

Frank Conway: 50%

Andrew Law: 50%