Engineering Assignment Coversheet

Please note that you:

- Must keep a full copy of your submission for this assignment
- Must staple this assignment
- Must NOT use binders or plastic folders except for large assignments

Student Number(s)
906348 683557
Group Code (if applicable):

Assignment Title:	Implementation of linear filters
Subject Number:	ELEN90058
Subject Name:	Signal Processing
Student Name:	WeiCheng Duan Shiyu Zhang
Lecturer/Tutor:	
Due Date:	17/OCT/2017

For Late Assignments Only

Has an extension been granted? Yes / No (circle)

A per-day late penalty may apply if you submit this assignment after the due date/extension. Please check with your Department/coordinator for further information.

Plagiarism

Plagiarism is the act of representing as one's own original work the creative works of another, without appropriate acknowledgment of the author or source.

Collusion

Collusion is the presentation by a student of an assignment as his or her own which is in fact the result in whole or in part of unauthorised collaboration with another person or persons. Collusion involves the cooperation of two or more students in plagiarism or other forms of academic misconduct.

Both collusion and plagiarism can occur in group work. For examples of plagiarism, collusion and academic misconduct in group work please see the University's policy on Academic Honesty and Plagiarism: http://academichonesty.unimelb.edu.au/

Plagiarism and collusion constitute cheating. Disciplinary action will be taken against students who engage in plagiarism and collusion as outlined in University policy. Proven involvement in plagiarism or collusion may be recorded on my academic file in accordance with Statute 13.1.18.

STUDENT DECLARATION

Please sign below to indicate that you understand the following statements:

I declare that:

- This assignment is my own original work, except where I have appropriately cited the original source.
- This assignment has not previously been submitted for assessment in this or any other subject.

For the purposes of assessment, I give the assessor of this assignment the permission to:

- Reproduce this assignment and provide a copy to another member of staff; and
- Take steps to authenticate the assignment, including communicating a copy of this assignment to a checking service (which may retain a copy of the assignment on its database for future plagiarism checking).

Student signature …… Date 17/OCT/2017 ……

Part A

Q1

a).

Divide the long sequence x[n] into arbitrary length segment (consider the length is L), use the divided segment to calculate the circular convolution with filter h(n), then get the output y[n] by adding the results of convolution of these segments and filter.

Q2

a).

As

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk}$$

$$y[n] = \sum_{n=0}^{N-1} X[k]\, W_N^{nk}$$

$$x[n] = \frac{1}{N} y[< -k >_N]$$

$$= \frac{1}{N} y[N-n]$$

$$= \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, W_N^{(N-n)k}$$

$$= \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, W_N^{Nk} \cdot W_N^{-nk}$$

$$W_N^{Nk} = 1$$

Hence,

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, W_N^{-nk}$$

Based on the process of proof, it can be concluded that after we get the results of DFT of x[n] (which is X[k]), we calculate the DFT of X[k]. When flipping the DFT of X[k] in the left-right direction, calculate the N-point circular shift. Multiplying the $\frac{1}{N}$, then we get the initial sequence x[n].

Q3

a).

if X[k] is conjugate symmetric, and x[n] = IDFT{X[k]} is real.

Consider

$$x[n] = x_r[n] + jx_i[n]$$

then

$$X[k] = X_{ep}[k] + X_{op}[k]$$

Hence

$$X_{ep}[k] = DFT\{x_r[n]\}, \qquad X_{op}[k] = DFT\{jx_i[n]\}$$

As x[n] is real, $X_{op}[k] = DFT\{jx_i[n]\} = 0$;

So

$$X[k] = DFT\{x[n]\} = DFT\{x_r[n]\} = X_{ep}[k] = X^*[k]$$

b).

$$X_o^*[< -k >_N] = X_o^*[N-k]$$

if substitute k with n, then we get

$$X^*[< -k >_N] = X^*[N-n]$$

so

$$X_o^*[N-n] = \sum_{n=0}^{N-1} x_o^* W_N^{-k(N-n)}$$

$$= \sum_{n=0}^{N-1} x_o^* W_N^{-kN} \cdot W_N^{kn} = X_o^*[n]$$

if substitute n with k, calculate the conjugate of $X_0[k]$

$$X_o^*[k] = X^*[k] + X^*[k+N]$$

as X[k] is conjugate symmetric, hence

$$X_o^*[k] = X^*[k] + X^*[k+N] = X[k] + X[k+N] = X_0[k]$$

Because

$$X_o^*[< -k >_N] = X_o^*[k] = X_0[k]$$

it can prove that $X_0[k]$ is conjugate symmetric.

Similarly, if we substitute k with n in formula $jX_1[< -k >_N]$, then we can get

$$jX_1[< -n >_N] = j\sum_{n=0}^{N-1} x_1[n] W_N^{k(N-n)}$$

$$= j\sum_{n=0}^{N-1} x_1[n]W_N^{-kn}$$

as

$$\{jX_1[< -k >_N]\}^* = -j\sum_{n=0}^{N-1} x_1^*[n]W_N^{kn}$$

hence

$$-\{jX_1[< -k >_N]\}^* = j\sum_{n=0}^{N-1} x_1^*[n]W_N^{kn}$$

$$= jX_1^*[k]$$

So, $jX_1^*[k] = jW_{2N}^{-k}(X^*[k] - X^*[k+N])$

As X[k] is conjugate symmetric,

$$-\{jX_1[< -k >_N]\}^* = jX_1^*[k] = jW_{2N}^{-k}(X[k] - X[k+N]) = jX_1[k]$$

$X_1[k]$ is conjugate anti-symmetric.

c).

As shown in the title, we know

$$X_0[k] = X[k] + X[k+N] = X[2k], k = 0, \dots, N-1$$

$$X_1[k] = W_{2N}^{-k}\{X[k] - X[k+N]\} = X[2k+1], k = 0, \dots, N-1$$

Q[k]= $X_0[k] + jX_1[k]$, q[n]=IDFT{Q[k]};

$$\mathrm{Re}\{\mathrm{q[n]}\} = \mathrm{IDFT}\{X_0[k]\} = \frac{1}{N}\sum_{n=0}^{N-1} X_o[k]\, W_N^{-nk}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} X[k]\, W_N^{-nk} + \frac{1}{N}\sum_{n=0}^{N-1} X[k+N]\, W_N^{-nk}$$

$$= \frac{1}{N}\sum_{n=0}^{2N-1} X[k]\, W_{2N}^{-nk} + \frac{1}{N}\sum_{n=0}^{2N-1} X[k]\, W_{2N}^{-nk}$$

$$= \frac{2}{N}\sum_{n=0}^{N-1} X[k]\, W_{2N}^{-nk}$$

$$= 2x[2n]$$

Hence, $\frac{1}{2}Re\{q[n]\} = x[2n]$.

Similarly,

$$Im\{q[n]\} = IDFT\{X_1[k]\} = W_{2N}^{-k}[\frac{1}{N}\sum_{n=0}^{N-1} X[k]\,W_N^{-nk} - \frac{1}{N}\sum_{n=0}^{N-1} X[k+N]\,W_N^{-nk}]$$

As X[N-n]=-X[n]

$$X[k+N] = X^*[< -k-N >_N] = X^*[-k] = X[-k] = -X[k]$$

$$Im\{q[n]\} = W_{2N}^{-k}[\frac{1}{N}\sum_{n=0}^{N-1} X[k]\,W_N^{-nk} + \frac{1}{N}\sum_{n=0}^{N-1} X[k]\,W_N^{-nk}]$$

$$= W_{2N}^{-k}[\frac{1}{N}\sum_{n=0}^{2N-1} X[k]\,W_{2N}^{-nk} + \frac{1}{N}\sum_{n=0}^{2N-1} X[k]\,W_{2N}^{-nk}]$$

$$= W_{2N}^{-k}\frac{2}{N}\sum_{n=0}^{2N-1} X[k]\,W_{2N}^{-2nk}$$

$$= \frac{2}{N}\sum_{n=0}^{2N-1} X[k]\,W_{2N}^{-(2n+1)k}$$

=2x[2n+1]

Hence, $\frac{1}{2}Im\{q[n]\} = x[2n+1]$.

d).

Suppose there is a real sequence x[n] with a length of 2N, and we know the DFT of it which is X[k].

Let

$$X_0[k] = X[k] + X[k+N]$$

$$X_1[k] = W_{2N}^{-k}(X[k] - X[k+N])$$

$$Q[k] = X_0[k] + jX_1[k]$$

Then

$$x[2n] = \frac{1}{2}Re\{q[n]\}$$

$$x[2n+1] = \frac{1}{2}Im\{q[n]\}$$

By these two formulas we can compute the initial sequence x[n].

e).

If the order of filter is M and the block size is N

The total complex multiplication is $Nlog_2(2N)$.

So computations per output data point equals to $\frac{Nlog_2(2N)}{N-M+1}$, and this can be approximated to be $\frac{2^v(0.5v+1)+1}{2^v-M+1}$ according to the Proakis textbook.

Part B

Task 1.

a).

the two functions are shown below:

iffta(x);

```
function x = iffta(X)
   X_conj = conj(X);
   x_conj = fft(X_conj);
   x = conj(x_conj);
   x = x / length(X);
end
```

ifftb(x);

```
function x = ifftb(X)

  y = fft(X);

  y = y / length(X);

  y = fliplr(y);          % flip array left to right

  x = circshift(y, 1, 2);      % shift by 1 in the 2nd dimension (right shift by 1)

end
```

b).

The code to test the function we designed and the function in the Matlab library.

```
clear;

close all;


x = randn(1, 16);


N = length(x) / 2;

n = 1:N;
```

```
Z = fft(x);


%iffta; Z = X;
X_conj = conj(Z);
x_conj = fft(X_conj);
a_1 = conj(x_conj);
a = a_1 / length(Z);


%ifftb; Z = X;
y = fft(Z);
y = y / length(Z);
y = fliplr(y);          % flip array left to right
b = circshift(y, 1, 2);     % shift by 1 in the 2nd dimension (right shift by 1)


%inbuilt ifft function


c = ifft(Z);
```

The results are shown below:

Results of iffta(x);

a | b | c

1x16 double

	1	2	3	4	5	6	7	8	9	10
1	-1.0689	-0.8095	-2.9443	1.4384	0.3252	-0.7549	1.3703	-1.7115	-0.1022	-0.2414
2										

Results of ifftb(x);

a | b | c

1x16 double

	1	2	3	4	5	6	7	8	9	10
1	-1.0689	-0.8095	-2.9443	1.4384	0.3252	-0.7549	1.3703	-1.7115	-0.1022	-0.2414

Results of ifft function in Matlab;

a | b | c

1x16 double

	1	2	3	4	5	6	7	8	9	10
1	-1.0689	-0.8095	-2.9443	1.4384	0.3252	-0.7549	1.3703	-1.7115	-0.1022	-0.2414
2										

Task 2

a).

The function which used to test conjugate symnmetry;

```
function y = isConjugateSymmetric(X)
  X = fft(x);
  Y = conj(X);
  y = fliplr(Y);            % flip array left to right
  x = circshift(y, 1, 2);      % shift by 1 in the 2nd dimension (right shift by 1)
  compare_number = eps('single');

  bool = any(abs(x-X) > tolerance);
  outcome = ~bool
end
```

b).

The functions ifftc(X);

```
function x = ifftcs(X)
   if ~isConjugateSymmetric(X)
      error('input is not conjugate symmetric');
   end
   N = length(X) / 2;
   if N ~= round(N)
      error('input is not a length-2N sequence');
   end
   k = 1:N;
   X0(k) = X(k) + X(k + N);
   W = exp((k * 1j * pi) / N);
   X1(k) = W .* (X(k) - X(k + N));
   Q = X0 + 1j * X1;
```

```
    q = ifft(Q);
    x(k*2+1) = 0.5 * real(q(k));
    x(k*2) = 0.5 * imag(q(k));
end
```

c).

The results of test functions we designed;

x[2n];

a1

1x8 double

	1	2	3	4	5	6	7	8
1	1.1093	0.0774	-1.1135	1.5326	0.3714	1.1174	0.0326	1.1006

x[2n+1]

a1 a2

1x8 double

	1	2	3	4	5	6	7	8	9
1	1.0933	-0.8637	-1.2141	-0.0068	-0.7697	-0.2256	-1.0891	0.5525	
2									

ifft(X[k]);

a1 a2 b

1x16 double

	1	2	3	4	5	6	7	8	9	10
1	1.0933	1.1093	-0.8637	0.0774	-1.2141	-1.1135	-0.0068	1.5326	-0.7697	0.3714
2										

Task 3

a) Write a Matlab function *overlapaddreal(B,x,N)*.

The Matlab implementation is shown as below

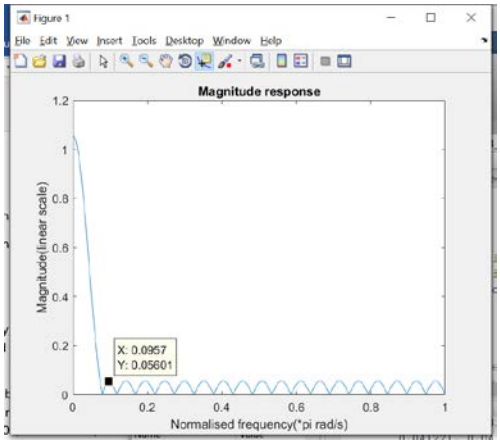
```
function X = overlapadd(B, x, N)

    %N is given in this case
    N1=length(x);
    M=length(B);
    %therefore we calculate L based on N and M
    L = N - M +1;

    %new zero padded array of x
    x=[x zeros(1,mod(-N1,L))];%to make sure x has interger number * L's length

    N2=length(x);
    %pad zeros to filter coef array
    B=[B zeros(1,L-1)];
    %perform N = L+M-1 point FFT to the h
    H_k=fft(B,L+M-1);
```

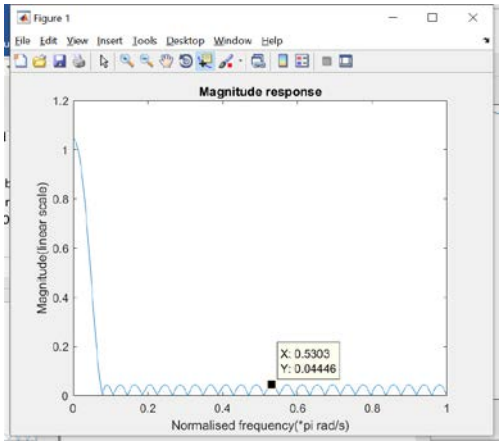
```
    %divide the new x array into
    S=N2/L;
    index=1:L;
    X=zeros(M-1);%output
    for stage=1:S
        %take L elements from the x array, pad M-1 zeros to the end
        xm=[x(index) zeros(1,M-1)];
        %take N point DFT of this subsequence
        X1=fft(xm,L+M-1);
        %do linear convolution of this subsequence x_1(eg) and the H array
        Y=X1.*H_k;
        %do inverse DFT of sub y sequence
        Y=ifft(Y);
        %Samples Added in every stage
        Z=X((length(X)-M+2):length(X))+Y(1:M-1);
        X=[X(1:(stage-1)*L) Z Y(M:M+L-1)];
        index=stage*L+1:(stage+1)*L;
    end
        %overlap DTF output is X
        X = X(1:length(X)-length(Z)-mod(-N1,L));

end
```

Task 4

a) **The sampling frequency is 24kHz. Design an FIR filter according to the following specifications Passband [0, 220 Hz] Stopband Above 880 Hz Passband ripple 0.05 Stopband ripple 0.05**

The filtered has been designed by using Matlab function *firpm()* with the result from *firpmord()*, however, the original filter order by firpmord() doesn't satisfy the design requirements, where stopband ripple is 0.005601>0.05, to solve this problem, we increase the filter order by 10, and the maximum stopband ripple is 0.04446<0.05 now.



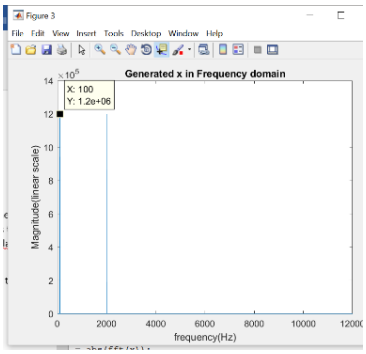


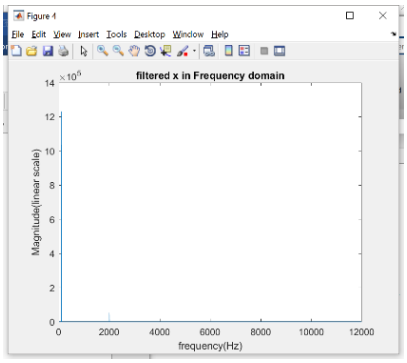
The Code for filter design has been attached to the appendix.

b) Generate a suitable test signal and implement the filter in a) using the Matlab routine you developed in Task 3. What is the optimal block length? Verify that your routine gives the same output as the filter and fftfilt command in Matlab. Explain briefly how the Matlab commands ***filter*** and ***fftfilt*** implements a linear filter.

A test signal consist of 2 sine wave components (w1 = 2*pi*100Hz, which will be able to pass the filter, and w2 = 2*pi*1000Hz, which is at the stopband of the filter) has been generated.

The test signal and after the FIR filter in frequency domain has been plotted in the figures below.





Optimal block length

It has been analysed that the total number of multiplication for a N point FFT is $c(v) = \frac{2^v*(0.5v+1)+1}{2^v-M+1}, where\ M = 49\ in\ our\ case$. A local minimum of this function at v = 8.65 has been found. and we rounded it up to 9. Therefore, the optimal block length of our filter will be $N = 2^9 = 512$.

Verification

The routine we developed in Part3. (c) has been verified by passing the result of function overlapadd() into a for loop to c to compare with the output from ***fftfilt(),*** and it can be shown the results are identical.

How *fftfilt()* works comparing to *filter()*

It can be found in the official documentation that fftfilt() use the implementation of overlapadd method as based on FFT, which is the same as the filter we implemented in overlapadd(). The advantage of this method is overlapadd implementation of linear FIR filter can process a longer input sequence more efficiently (less computation) comparing to the normal *filter() function.*

(c) How many complex multiplications and additions per second are required to implement the filter in each case?

It can be estimated that if we implement the filter using *filter(),* linear convolution will be performed between h[n](filter coefficients) and input signal sequence, which will result in M*Fs= 50*24000 = **1200000** multiplication per second.

In fftfilt() where overlapadd is implemented, $F_s * \left(1 + log2(F_s)\right) = 24000 * \left(1 + log2(24000)\right) =$ **373218** multiplication will be carried out.

In overlapadd with symmetry, it has been calculated that we have optimal block length when v = 8.75, and the number of multiplication per output data point is $\frac{2^v(0.5v+1)+1}{2^v-50+1} = 6.06$, therefore, the total number of multiplication per second is 6.06*24000 = **145630**.

The matlab code for generating the signal and graphs above has been added to the appendix.

Part C DSP implementation

The coding for DSP has been implemented for nomal time domain FIR filter in function *process_time()*, and the overlapadd implemtation is in function *process_block(fract32 output[]).*

The audio test signal generated from Matlab in PartB task 4 has ben inputted to the DSP board and it can be heard that only the low frequency component of the signal is outputted from the DSP board for both the time domain implementation and FFT overlap add implementation. The result has been estimated as the same we can hear from Matlab.

The code _LabTaskC.c and prams.h has been attached to the appendix.

Appendix

Part B Task 1

iffta(x)

```
function x = iffta(X)
```

```
    X_conj = conj(X);
    x_conj = fft(X_conj);
    x = conj(x_conj);
    x = x / length(X);
end

ifftb(x)
function x = ifftb(X)
   y = fft(X);
   y = y / length(X);
   y = fliplr(y);              % flip array left to right
   x = circshift(y, 1, 2);       % shift by 1 in the 2nd dimension (right shift by 1)

end
```

test of ifft and function designed

```
clear;
close all;

x = randn(1, 16);

N = length(x) / 2;
n = 1:N;

Z = fft(x);

%iffta; Z = X;
X_conj = conj(Z);
x_conj = fft(X_conj);
a_1 = conj(x_conj);
a = a_1 / length(Z);

%ifftb; Z = X;
y = fft(Z);
y = y / length(Z);
y = fliplr(y);              % flip array left to right
b = circshift(y, 1, 2);       % shift by 1 in the 2nd dimension (right shift by 1)

%inbuilt ifft function

c = ifft(Z);

test conjugate symmetry
function y = isConjugateSymmetric(X)
   X = fft(x);
   Y = conj(X);
   y = fliplr(Y);              % flip array left to right
   x = circshift(y, 1, 2);       % shift by 1 in the 2nd dimension (right shift by 1)
   compare_number = eps('single');

   bool = any(abs(x-X) > tolerance);
   outcome = ~bool
end

ifftc(x);

function x = ifftcs(X)
    if ~isConjugateSymmetric(X)
        error('input is not conjugate symmetric');
    end

    N = length(X) / 2;
    if N ~= round(N)
        error('input is not a length-2N sequence');
    end

    k = 1:N;
```

```
    X0(k) = X(k) + X(k + N);
    W = exp((k * 1j * pi) / N);
    X1(k) = W .* (X(k) - X(k + N));

    Q = X0 + 1j * X1;

    q = ifft(Q);

    x(k*2+1) = 0.5 * real(q(k));
    x(k*2) = 0.5 * imag(q(k));
end
```

test ifft and code designed

```
clear;
close all;

x = randn(1, 16);

N = length(x)/2;
k = 1:N;

Z = fft(x);
%ifftcs

 X0(k) = Z(k) + Z(k + N);
 W = exp(1j * pi/ N).^(k-1);
 X1(k) = W .* (Z(k) - Z(k + N));

 Q = X0 + 1j * X1;

 q = ifft(Q);

 x(k*2+1) = 0.5 * real(q(k));
 x(k*2) = 0.5 * imag(q(k));
 a1 = [x(k*2)];
 a2 = [x(k*2+1)];


 %ifft function

 b = ifft(Z);
```

Part B Task 4

```
%filter design
clc;
clear;
close all;


N = 2^9;  % block length

fs = 24E3;%samplin frequency
% FIR filter design
[ORD, passband_edge, frequency_band_mag, w] = firpmord([220 880], [1 0], [0.05
0.05], fs);
ORD = ORD + 10;

num = firpm(ORD, passband_edge, frequency_band_mag, w);

%verify the filter
[h, w] = freqz(num, 1, 2^10);
plot(w/pi, abs(h));
title('Magnitude response');
```

```
xlabel('Normalised frequency(*pi rad/s)');
ylabel('Magnitude(linear scale)');

%PHASE
figure;
plot(w/pi, phase(h));
title('Phase response');
xlabel('Normalised frequency(*pi rad/s)');
ylabel('Phase(rad)');

%num is the filter coefficients we need
%use print array for this array
print_array(num, 'b');

%%%%Test signal genration
%central frequency at around
%r = r1+r2, where r1 at 100Hz,
f1 = 100;
w1 = 2*pi*f1;
%r2 at 1000Hz
f2 = 2000;
w2 = 2*pi*f2;
%test signal length is the same as the sampling frequency
n_sampled = (1:fs*100)/fs;
x = sin(w1*n_sampled)+sin(w2*n_sampled);

%plot this generated signal in frequency domain
figure;
N=fs*100;
X1_mags = abs(fft(x));
fax_bins = [0 : N-1]; %frequency axis in bins
N_2 = ceil(N/2);
plot(fax_bins(1:N_2)*fs/N, X1_mags(1:N_2));
title('Generated x in Frequency domain');
xlabel('frequency(Hz)');
ylabel('Magnitude(linear scale)');

%sound(x, fs);

%%try to filter this signal with the filter we designed

x_filtered = filter(num, 1, x);


%plot this generated signal in frequency domain
figure;
N=fs*100;
X1_mags = abs(fft(x_filtered));
fax_bins = [0 : N-1]; %frequency axis in bins
N_2 = ceil(N/2);
plot(fax_bins(1:N_2)*fs/N, X1_mags(1:N_2));
title('filtered x in Frequency domain');
xlabel('frequency(Hz)');
ylabel('Magnitude(linear scale)');

%sound(x, fs);

y_overlapadd = overlapadd(num, x, N);
y_filter  = filter(num, 1, x);
y_fftfilt  = fftfilt(num, x);

for i= 1:1:length(y_overlapadd)
    if((y_overlapadd(i)) ~= (y_fftfilt(i)))
        fprintf('WRONG!!!\n');
    end
end

fprintf('the overlap add output is the same as the fftfilt() function output' );
```

Part C

_LabTaskC.c

```
#include "SPWS3.h"
#include "Params.h"

complex_fract32 twiddle[N/2] = { 0 };
complex_fract32 filter_fft[N] = { 0 };

complex_fract32 input_freq[N] = { 0 };
complex_fract32 output_fft[N] = { 0 };
fract32 output_save[M-1] = { 0 };

// array b
float b[] = { -0.022345, 0.000024, 0.000477, 0.001261, 0.002341,
    0.003740, 0.005422, 0.007399, 0.009625, 0.012098,
    0.014760, 0.017602, 0.020547, 0.023589, 0.026627,
    0.029677, 0.032549, 0.035320, 0.037891, 0.040180,
    0.042183, 0.043828, 0.045099, 0.045957, 0.046392,
    0.046392, 0.045957, 0.045099, 0.043828, 0.042183,
    0.040180, 0.037891, 0.035320, 0.032549, 0.029677,
    0.026627, 0.023589, 0.020547, 0.017602, 0.014760,
    0.012098, 0.009625, 0.007399, 0.005422, 0.003740,
    0.002341, 0.001261, 0.000477, 0.000024, -0.022345 };

float process_time(float x0)
{
    // TODO: 1. Implement the filter using time domain methods

    static float x[BUFFER_SIZE] = {0.0};  // BUFFER_SIZE = (M-1) is defined in 'Params.h'
    static int current = 0;

    //return back forwading position of the array if current is negative
    float y = b[0] * (x0 + x[(current + BUFFER_SIZE)%BUFFER_SIZE]);      // Macro
'REM(current)' is defined in 'Params.h'

    x[current] = x0;
    // save current x0 into x after 'y' is calculated, thus the size of 'x' can be reduced
by 1 (from M to M-1).

    for (int i = 1; i <= BUFFER_SIZE/2-1; i++) {
        y += b[i] * (x[(current-i) + BUFFER_SIZE)%BUFFER_SIZE)] + x[((current+i) +
BUFFER_SIZE)%BUFFER_SIZE]);
    }

    y += b[BUFFER_SIZE/2] * x[((current-BUFFER_SIZE/2)) + BUFFER_SIZE)%BUFFER_SIZE];
    current++;
```

```
    current %= BUFFER_SIZE;//INCREMENT EVERY BUFFER_SIZE

    return y;
}

void init_process()
{
    int i;

    // calculate twiddle factors
    twidfftrad2_fr32(twiddle, N);

    // copy filter coefficients to input array to do fft
    for (i = 0; i < M; i++)
        input_data[i] = (1 << 30) * b[i];
        // [ note ]
        // Here we should scale by (1 << 31)-1 for full scale, however
        // doing so can cause overflows in fixed point, so we halve it
        // here and put back the factor 2 on output.

    // do fft
    int filter_blk_exp;
    rfft_fr32(input_data, filter_fft, twiddle, 1, N, &filter_blk_exp, 1);

    // rescale data points
    for (i = 0; i < N; i++)
    {
        filter_fft[i].re = filter_fft[i].re << (filter_blk_exp);
        filter_fft[i].im = filter_fft[i].im << (filter_blk_exp);
    }

    // clear input array
    for (i = 0; i < M; i++)
        input_data[i] = 0;
}

void process_block(fract32 output[])
{
    // TODO: 2. Implement the filter using the overlap-add method

    int index = 0;

    int blk_exp;

    //input_data defined, input_fft->input_freq, twiddle, N,
    rfft_fr32(input_data, input_freq, twiddle, 1, N, &blk_exp, 1);

    // conjugate symmetry
```

```
    output_fft[0] = cmlt_fr32(filter_fft[0], input_freq[0]);

    index = 1;
    while(index<N/2){
        //multiply H[k] and x[k]
        output_fft[index] = cmlt_fr32(filter_fft[index], input_fft[index]);
        //secong half of the FFT is conjugate of the first half
        output_fft[N-index] = conj_fr32(output_fft[index]);//complex conjugate
        index++;
    }

    output_fft[N/2] = cmlt_fr32(filter_fft[N/2], input_fft[N/2]);

    complex_fract32 output_complex[N]= { 0 };

    //inverse FFT of the output array
    ifft_fr32(output_fft, output_complex, twiddle, 1, N, &blk_exp, 1);

    int i;
    for (i = 0; i < N; i++)
    {
        //re scale data point as what he did in init_process()
        output[index] = output_complex[index].re << (blk_exp);
    }

    // overlap add
    index = 0;
    while( index < M-1 ) {
        output[index] += output_save[index];
        output_save[index] = output[L+index];
        index++
    }

}
```

prams.h

```
// TODO: 0. Modify these constants to match the filter you have designed

// length of filter
#define M 49

// buffer size
#define N 512 //this is dertermined by the optimal block length

// input data processing block size
#define L (N-M+1)

#define BUFFER_SIZE (M-1)
```