BILKENT UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINERING

EEE-491

Electrical and Electronics Engineering Design I

Spoken Number Recognition Project

LAB REPORT

Assignment: Lab-System

Prepared by

Yusuf Ziya Dilek

Can Doğa Patır

Dilara Halavurt

17.06.2020

TABLE OF CONTENTS

1. LAB-DEBUG

The design of the debugger for the entire project has been the first step of the entire project. It consisted of transmitting 64 bit data from Basys-3 board to MATLAB program running on PC via USB connection. Universal Asynchronous Receiver and Transmitter (UART) is a serial communication protocol and is made use of in this part of the project.

A finite state machine called FSM_DEBUG, is designed in such a way that, UART component is triggered 8 times in a row to transmit 64 bit data. FSM_DEBUG makes use of a dual-port RAM of Basys-3 board to store and then transmit the data. For the communication the bandrate is adjusted to 115200.

The in-out signal diagram for the finite state machine is designed as given below by the project specifications:

- reset signal
- 100mhz clock signal
- 14 bit ram_addr_out signal
- 64 bit ram_data_in signal
- txd signal (output of UART)
- start signal (indicating the start of the transfer)
- busy signal (indicating the data transfer status of the connection)

The schematic of the debugger is given in Figure 1.

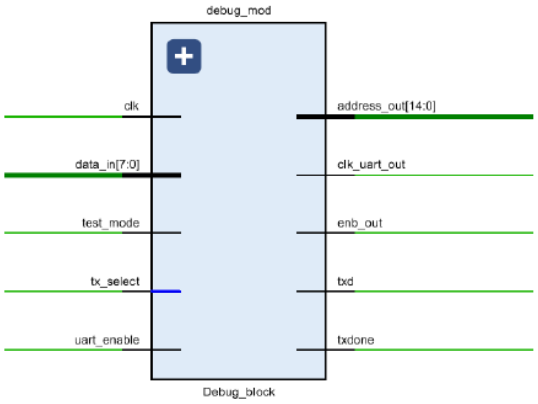


Figure 1. Showing the FSM_DEBUG module

In order to test the debugger module, data with constant intervals is sent from the Basys-3 RAM to the MATLAB. Below graph is created by the data collected from the FPGA to the MATLAB.

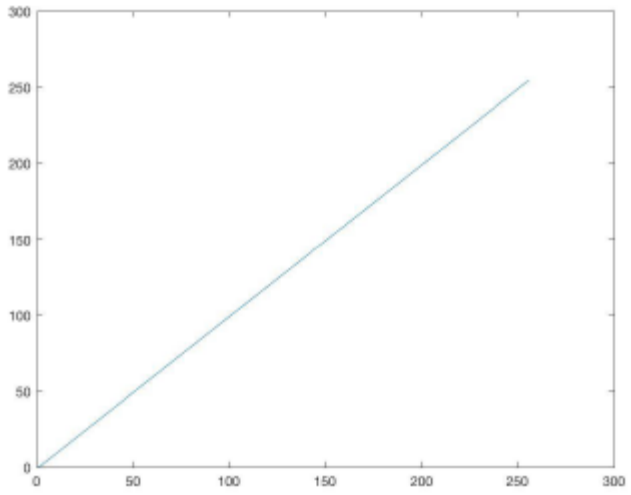


Figure 2. Showing the data from debugger

2. LAB-ADC

ADC lab consisted of designing a sampler with SPI interface to 12 bit ADC converter within the Basys-3 board. The SPI interface is designed to be operated at 10MHz and the sample rate will be 16 KHz. The dual-port RAM of the debugger is used to store the 12 bit data. The RAM size will be limited to store 16384 samples of 12 bit data. For the ADC component, instead of an external ADC, the built in ADC from Basys-3 board is used.

The in-out signal diagram for the finite state machine is designed as given below by the project specifications:

- reset signal

- 100mhz clock signal
- sclk output signal (the clock of SPI interface)
- cs_n output signal (chip select signal of SPI interface to ADC board)
- sdata input signal
- 14 bit ram_addr_in signal
- 16 bit ram_data_put signal
- start signal
- busy signal

In order to test the ADC module,sinusiodal signal is sent to the Basys-3 board, the data is stored in dual-port RAM. For testing purposes, debugger is used to send the sampled sinusoidal data from Basys-3 to MATLAB. The results are given in Figure 3.

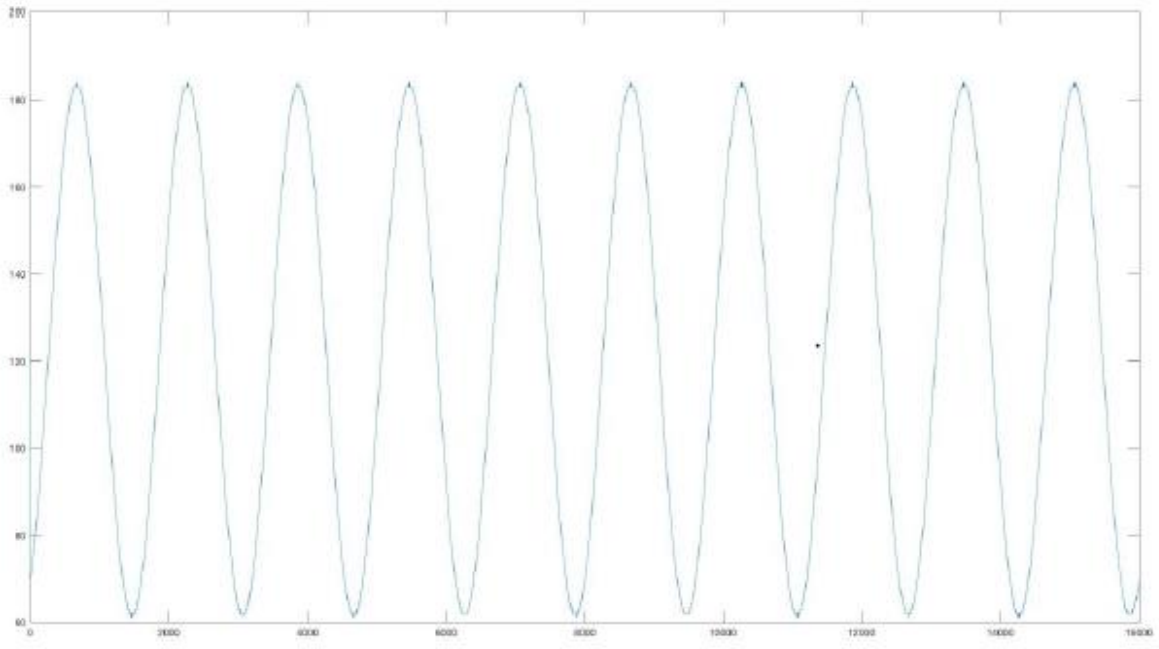
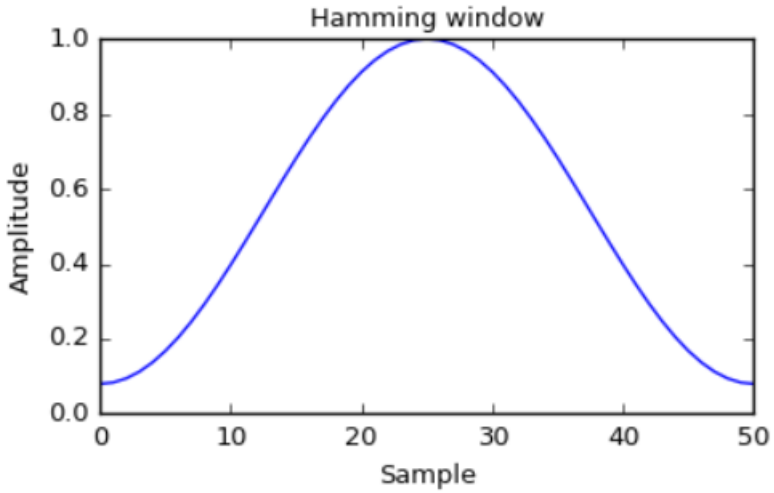


Figure 3. Showing the data from ADC module

3. LAB-WINDOW

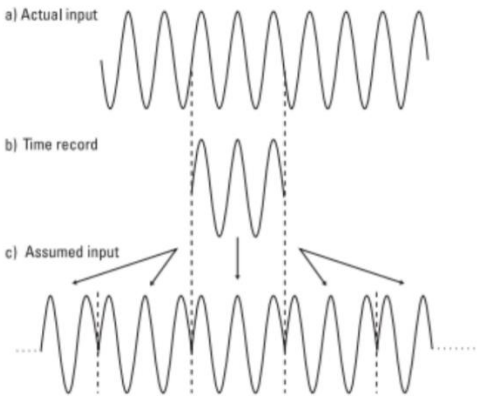
In order to provide a viable input signal for our FFT function, we need a windowing function. Windowing functions are being used to minimize the leakage as much as possible, which may cause noise. For our project, our assignment is to fragment the data into 512 sample frames, which we acquire from an external RAM as 16-bit length. Each frame will overlap with the neighbor frames at an overlap amount of %50. Hamming Window will be used for this project, and 512 samples of each frame will be multiplied with the coefficients of hamming window, chosen to be 16-bit.

3.1. **Hamming Windowing Function**



(figure 1.)

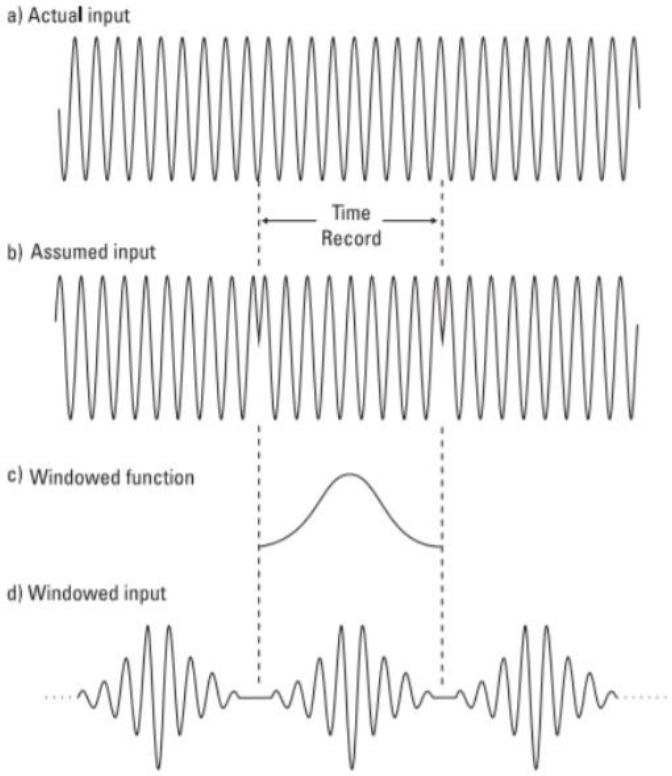
Hamming windowing function (figure 1.) is being used in order to reduce the amplitudes at each end of a sample, in order to prevent misconnections of the sampled signals during the FFT progress, which causes the leakage.



In this example, since the edge frequency magnitudes does not match during the FFT progress, the actual input and assumed input by FFT function are totally different, which will cause a noisy and undesired signal.

(an example of leakage)

However, when we scale these taken samples with our hamming function, those edge frequency magnitudes will be reduced to almost 0, which will therefore create a matching connection for FFT;



Windowing function eliminates the edge frequency responses, resulting with a satisfied connection for FFT function. This process may cause some amplitude loss at the edges, however the frequency of input signal will be seen by FFT function correctly, which is the important part over the amplitude, in our case.

(Prevented leakage)

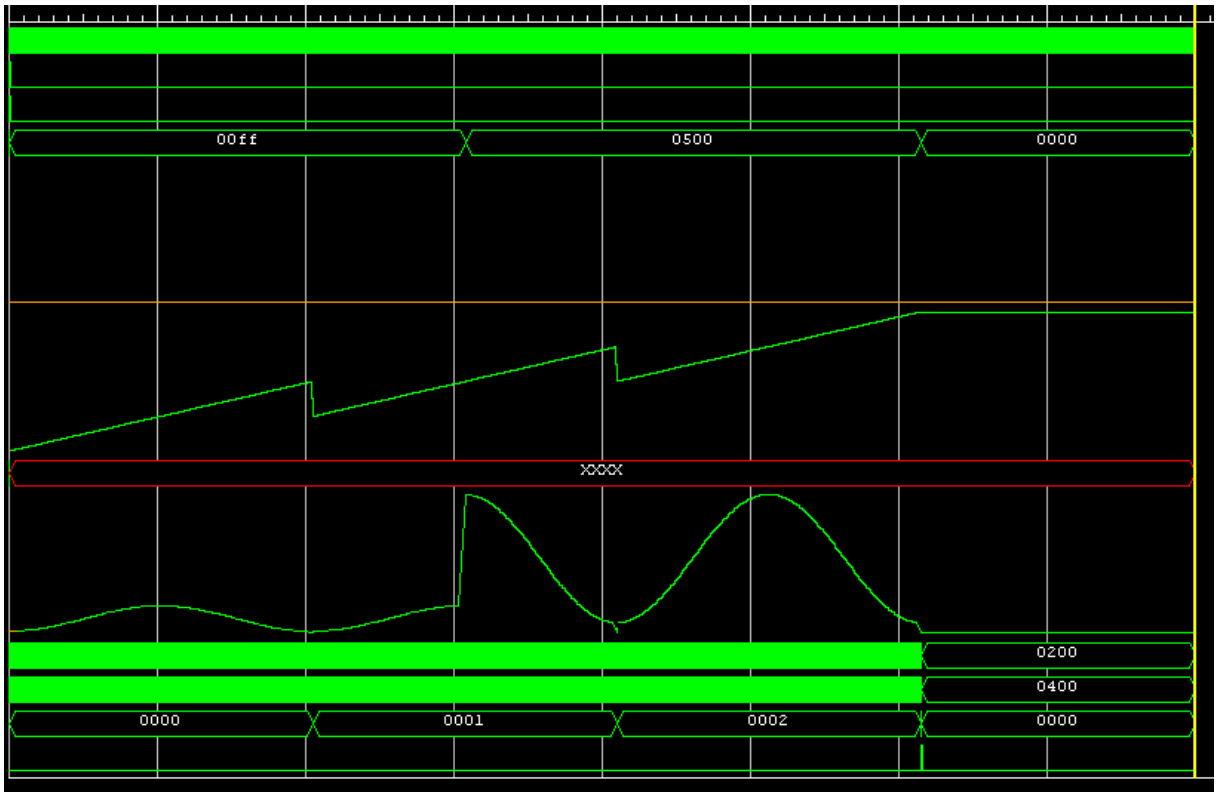
3.2. **Demonstration of Hamming Function on LAB Window**

First, we have constructed the hamming window in the MATLAB and converted into hex like this:

```
fileID = fopen('hamming.txt','w');

coefficients = int16(round(hamming(512)*(2^15-1)));
coefficients = transpose(coefficients);
A = [0:511; coefficients];

fprintf(fileID, '%d => x"%x",\n', A);%%% hex
fclose(fileID);
```

Then we entered the all 512 by hand to ROM. You can find that part in the Appendix. I won't put the code here since it is rather long.

To demonstrate and check our code we designed a test-bench where only lab_window is device under test (DUT). We fill 512 with hex#00ff and 512 with hex#0500 and rest is hex#0000. We should get 3 frames since there is 50% overlap.



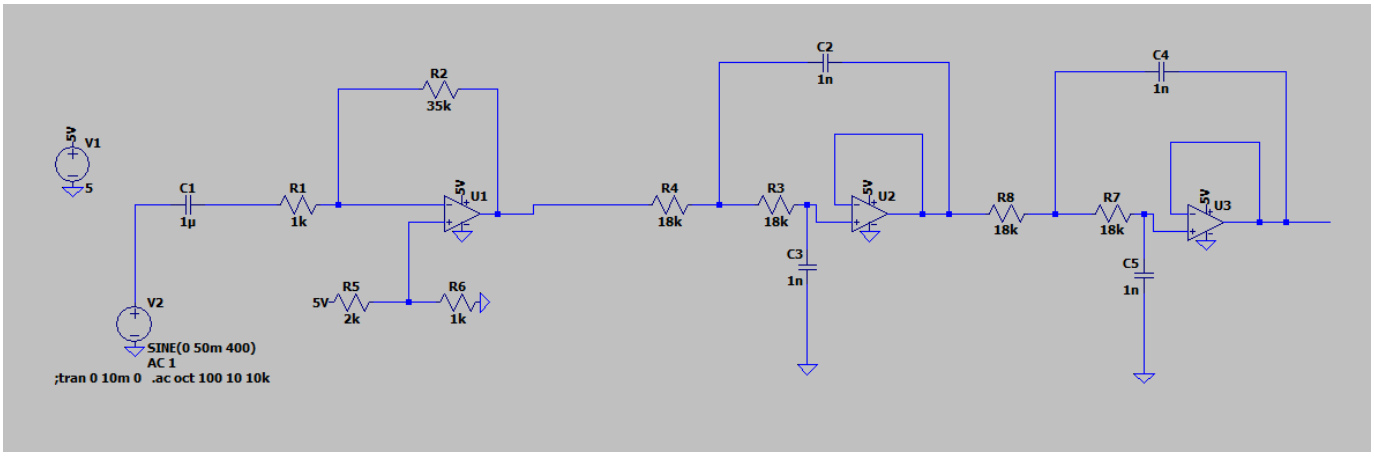
Simulation of the lab_window

From the simulation plot we can see the hamming window shape clearly in the first frame which corresponds to hamming function is multiplied by hex#00ff. In the second frame, because of the 50%overlap, half of it hamming*hex#00ff and the other half is hex#0500. Our testbench results

show that our function is able perform windowing without issues and the 50% frame overlap is also works as intented.

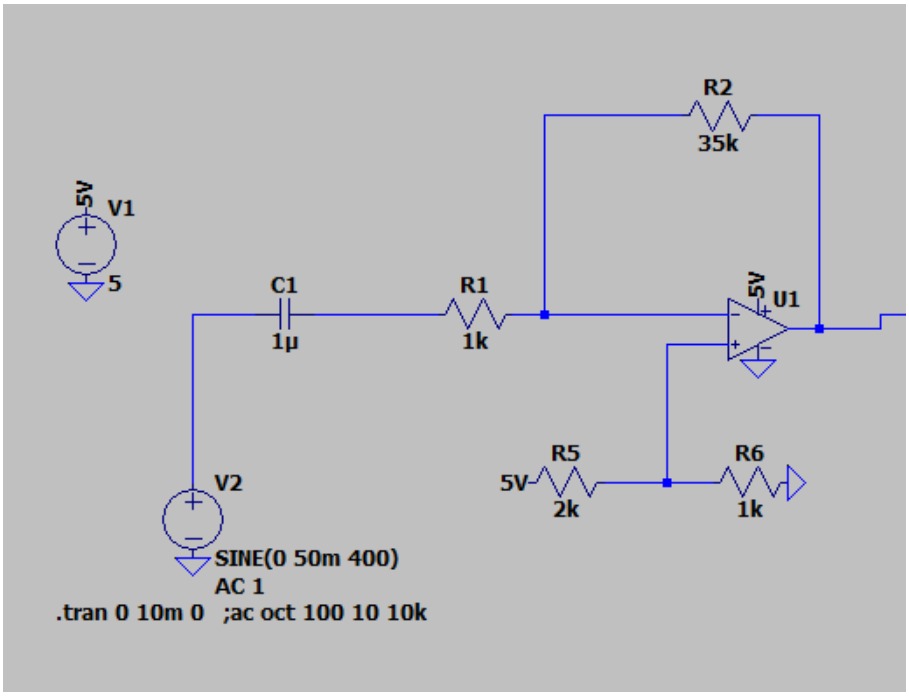
4. **LAB-PCB**

This part of the project is focused on creating a circuit, which takes the sound input from microphone, with around 10mV amplitude, and amplifies it to 3.3 V peak-to-peak, in order to be used by ADC system that we created previously. For the amplification purpose, a single OPAMP will be used, as an indication of the assignment. Then, this amplified signal will be passed through a cascaded 2 Low-Pass Filters, with Sallen-Key topology; which will serve the purpose of eliminating the noise and any other disturbance, due to high frequencies.



Overall PCB circuit

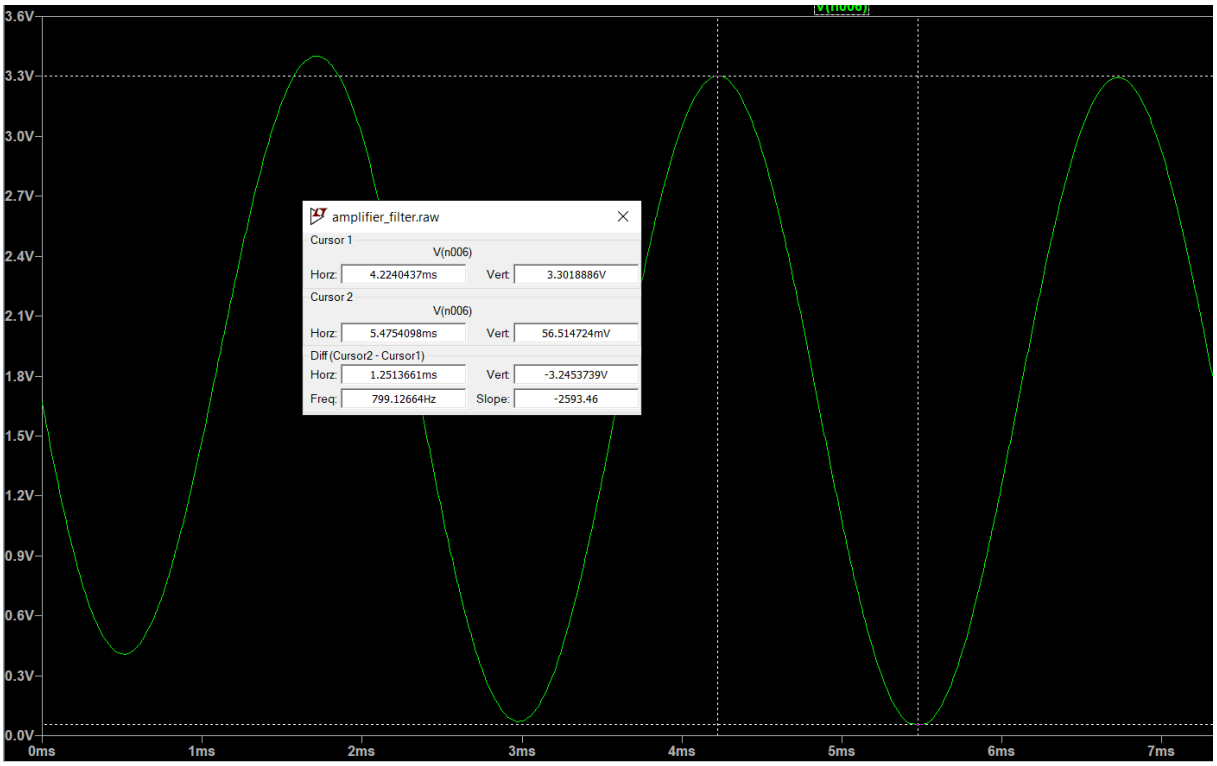
4.1. **Sound Amplifier**



(figure 1.)

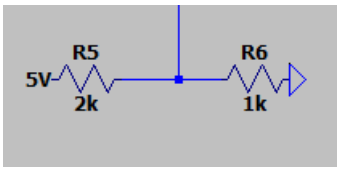
To process our input signal (sound) further, it should be amplified to the level of viable input level of ADC, which is, in our case given as 3.2-3.3 V peak-to-peak. However, our input signal's amplitude is around 100mV peak-to-peak. So for this purpose, we have used a single OPAMP amplifier (figure 1.) which has a gain of;

$$\frac{R1 + R2}{R1} = \frac{1k + 35k}{1k} = 36$$



(figure 2.)

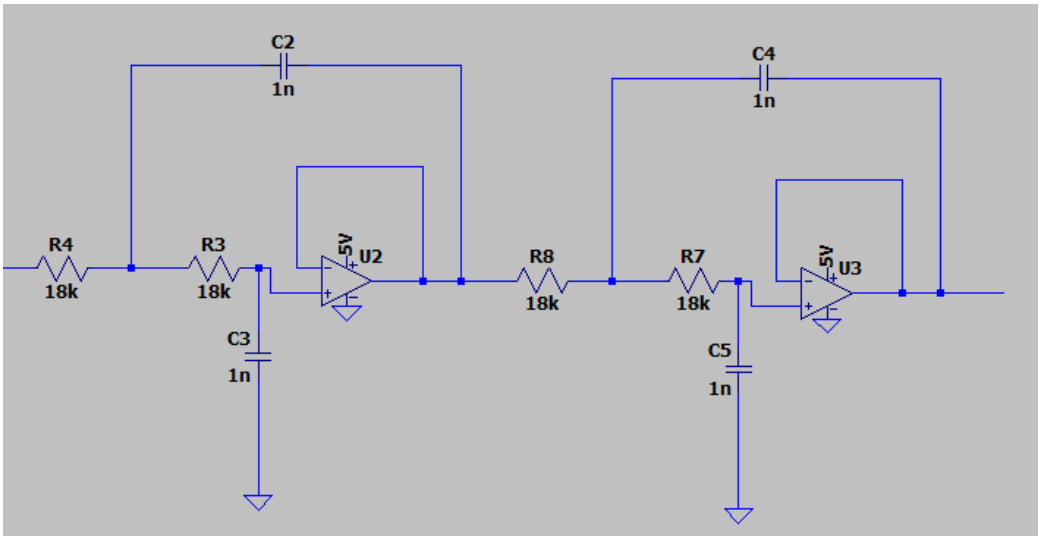
Figure 2. demonstrates an instance of output signal from amplifier, after the circuit reaches to it's stable state. Peak-to-peak voltage is estimated as roughly 3.25V, which will serve well in our project.



This part (figure 3.) add offset to our signal, so we can get the output signal between 0 V and 3.3 V

(figure 3.)

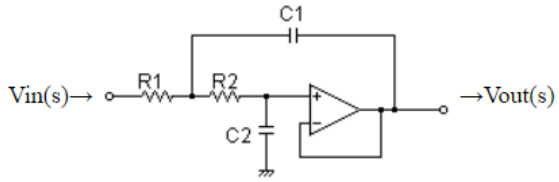
After we set the voltage level to satisfy requirements, next, we have implemented Sallen-Key LPF (figure 4.). The main purpose of this part is to avoid aliasing and create the specific frequency range for our system. Corner frequency is required to be around 4 kHz.



(figure 4.)

In this part, cascaded two filter have been used, each being second order. So in total, 4th order filter circuit provides 80dB/decade attenuation performance, as the assignment requires. The resistor values were estimated through the following website at first (figure 5.), however, the site doesn't provide a selection for cascaded systems, so we had changed the values found through the website, in order to fit it into our case;

http://sim.okawa-denshi.jp/en/OPstool.php



Transfer Function:

$$G(s)=\frac{657462195.92373}{s^2+51282.051282051s+657462195.92373}$$

R1 = 39kΩ
R2 = 39kΩ
C1 = 0.001uF
C2 = 0.001uF

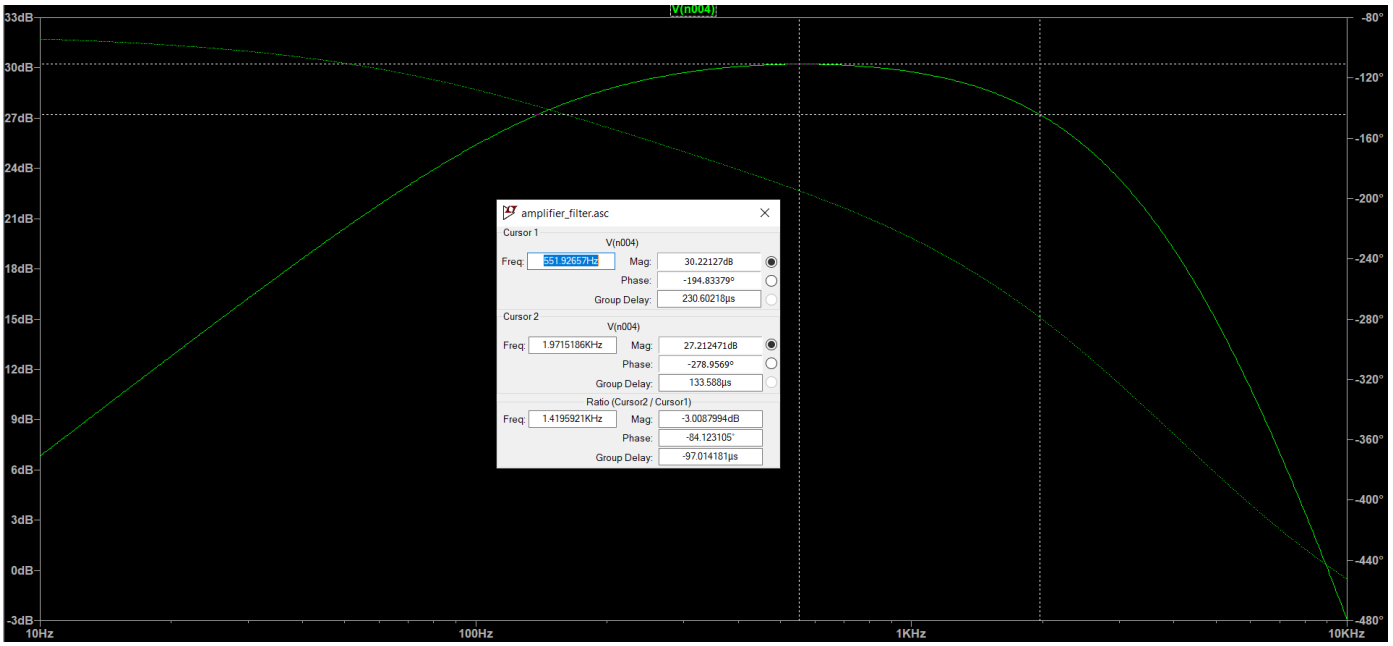
Cut-off frequency

fc = 4080.8959767153[Hz]

(figure 5.)

Website calculated the required resistor values as 39k Ohm each, however, when we try these values, the cut-off frequency was observed* as around 2 kHz, which is lower than the desired frequency (figure 6.).

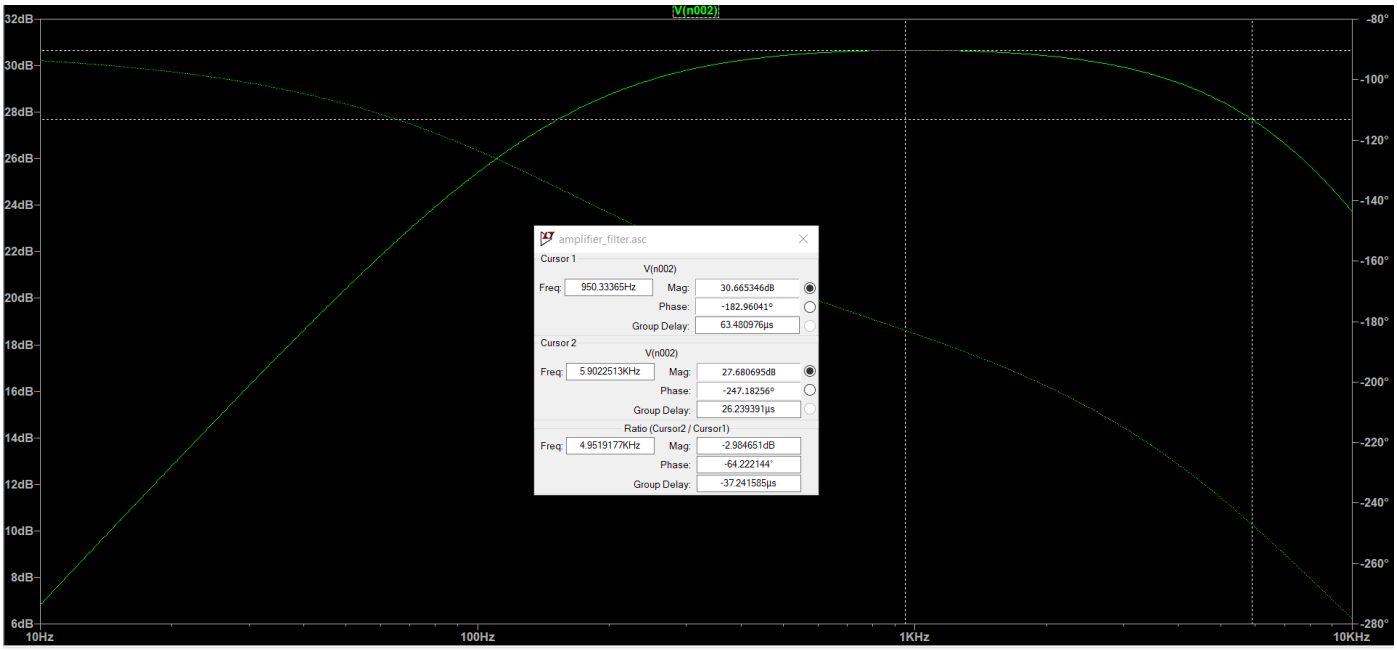
*Cut-off frequency is determined as the frequency where the maximum magnitude drops 3db below.



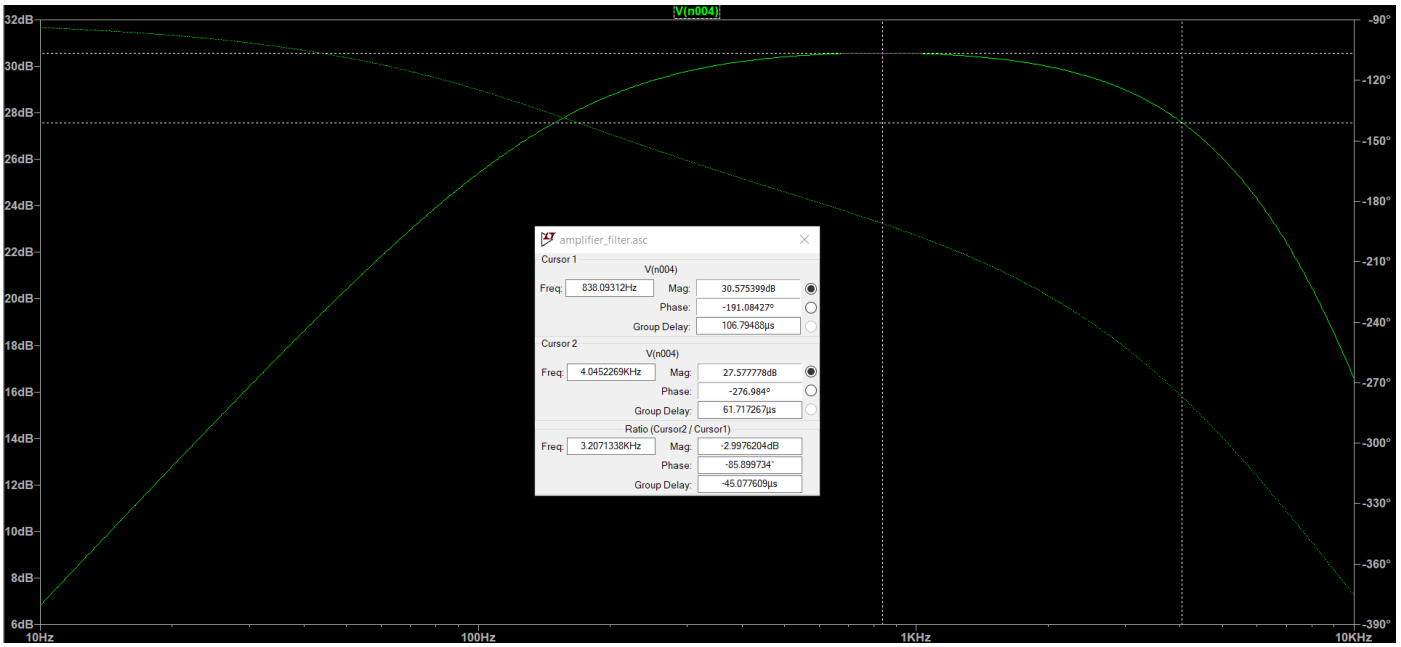
(figure 6.) (f_c=1.97kHz)

In this first attempt (with 39k Ohm resistors) the cuf-off frequency was found as 1.97KHz.

When we arranged the resistors so that we are able to use two cascaded filters, we found the new values as 18k Ohm each.



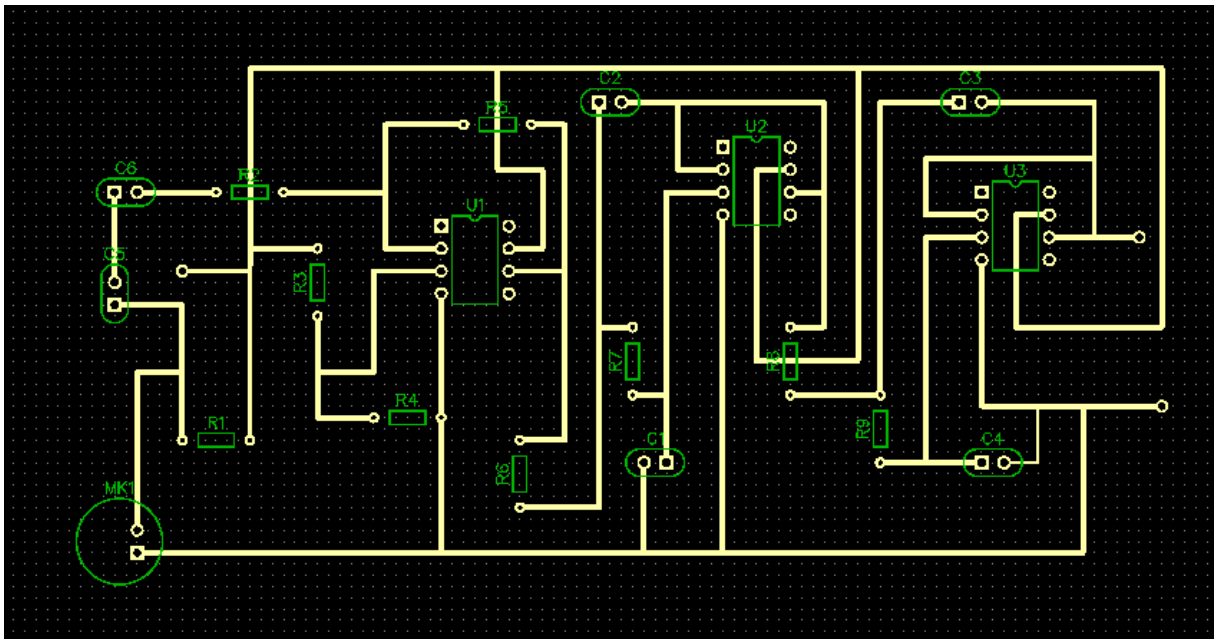
(figure 7.) (f_c= 5.90kHz)



(figure 8.) (f_c=4.04kHz)

In this final simulation (figure 7, 8.), cut-off frequency before the second filter was found as around 6kHz, and by the use of second filter, this cuf-off frequency reaches down to 4.04 kHz, which almost same as the desired value, 4kHz.

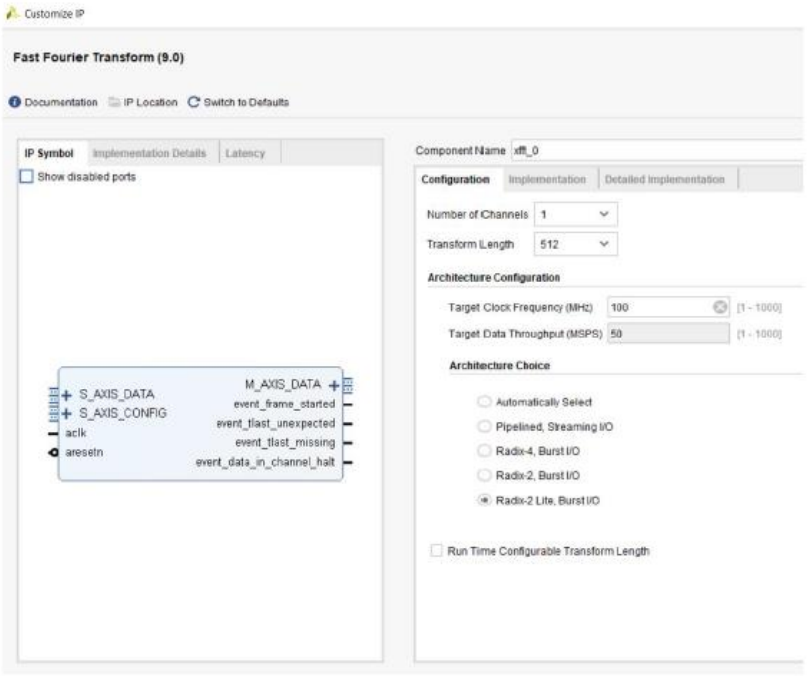
4.2. **PCB Layout**



PCB layout was generated by the use of DipTrace software. We were able to construct this PCB card and placed each component without any struggle, however, the only problem was the use of IC slots. We had created the layout with arbitrary IC slots (U1, U2, U3), without considering a real IC's connection information. This might create a problem while finding ICs which fit into this layout, if we were to actually implement the hardware circuit.

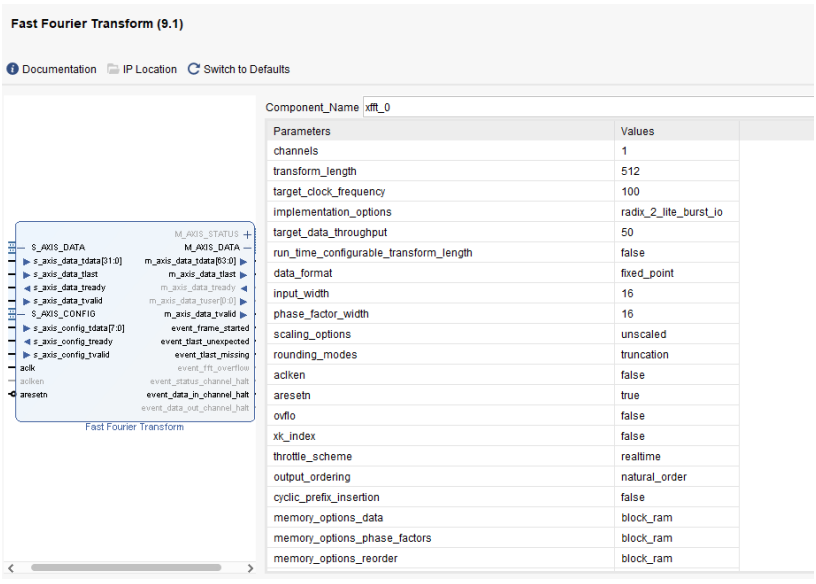
5. LAB-FFT

For the FFT part of this project, we were tasked to get 16-bit data from a RAM (depth of 512) and executing 512 point FFT (Fast Fourier Transform). This operation needs to be executed by Basys 3. For this, we need to use an IP block of Vivado Design Suite since FFT operation would be implausible to create from scratch for our team considering the time constraints attached to this project. Already available FFT IP block in the Vivado should be suitable for our needs. However, we first need to configure the FFT IP block. We used *logiCore* IP product guide (official guide) and the step-by-step guide is given to us in the assignment. While adding the FFT IP block we can configure from this window:



FFT IP block config window

From here we followed the guide in the assignment and configured our IP block accordingly. 512 point transform length, 100 MHz clock frequency (clock of Basys 3) is chosen and for the architecture, we selected the last one as instructed.



Parameters	Values
channels	1
transform_length	512
target_clock_frequency	100
implementation_options	radix_2_lite_burst_io
target_data_throughput	50
run_time_configurable_transform_length	false
data_format	fixed_point
input_width	16
phase_factor_width	16
scaling_options	unscaled
rounding_modes	truncation
aclken	false
aresetn	true
ovflo	false
xk_index	false
throttle_scheme	realtime
output_ordering	natural_order
cyclic_prefix_insertion	false
memory_options_data	block_ram
memory_options_phase_factors	block_ram
memory_options_reorder	block_ram

Details of the FFT IP block

The IP block takes 16-bit real data and outputs 64-bit (32-bit real, 32-bit imaginary) data. However, we here our output width is actually 16 +log2(512) = 26-bits which corresponds to 0-bit to 25-bit. The imaginary part starts from bit 32 and ends in bit 57.

To check our IP block, we prepared the test-bench code according to the instructions in the assignment. Note that we are constructing a cosine wave inside of the test-bench and write it as input.txt and after taking the FFT we write the results in the output.txt like this:

file writeFile : TEXT open WRITE_MODE is "C:\Users\Yusuf Ziya Dilek\Desktop\output.txt";

file writeFile : TEXT open WRITE_MODE is "C:\Users\Yusuf Ziya Dilek\Desktop\input.txt";

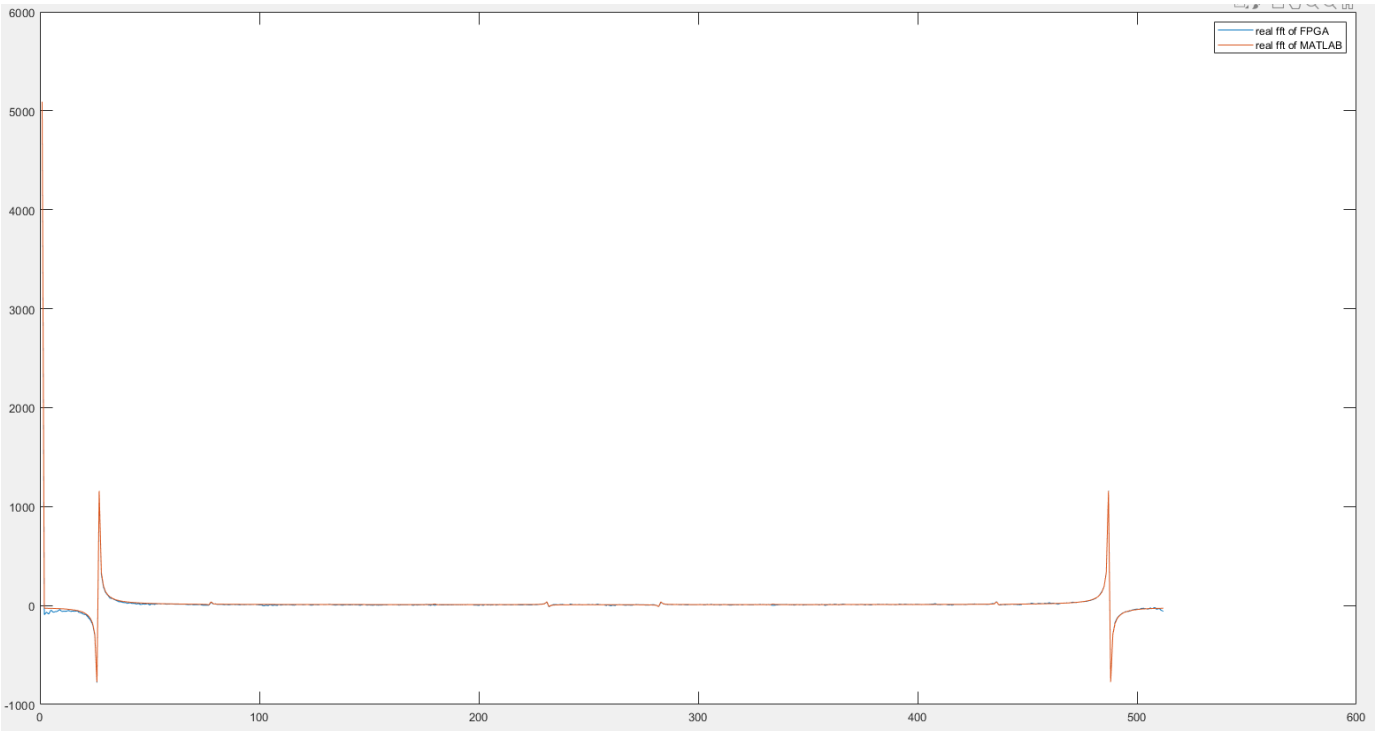
Notice that the input is created inside the test bench, not taken from a file.

```
for i in 0 to 1000000000 loop
        data_in_int <= integer(cos(real(i)*0.1*MATH_PI)*10.0+10.0);
        data_in <= std_logic_vector(to_unsigned(data_in_int, 16));
        wait for clock_period;
end loop;
```

This is kept in the input.txt so that we can later take the FFT inside the MATLAB and compare it to the FFT taken by the IP block.

We have written a very simple MATLAB code where we can check the FFT taken by the IP block like this:

```
figure(1)
fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\input.txt','r');
input = fscanf(fileID,'%d');
plot(input)

fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\output.txt','r');
fft_real_fpga = fscanf(fileID,'%d');
figure(2);
plot(fft_real_fpga)
hold on

fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\input.txt','r');
sine_fpga = fscanf(fileID,'%d');
sine_fpga = sine_fpga(1:512);
fft_real_matlab = real(fft(sine_fpga));

plot(fft_real_matlab)
legend('real fft of FPGA', 'real fft of MATLAB')
hold off
```



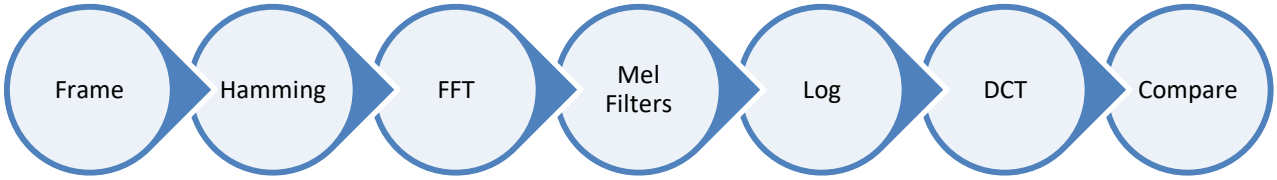
FFT taken by the MATLAB is compared with the FFT taken by the IP block

You can see they are very similar to each other. Since we are using a cosine signal for this test, we can see two frequency peaks of the input sinusoidal also a peak at the DC level.

6. LAB-MATLAB

For this lab, we were tasked with constructing a model that is able to recognize a given number between zero and nine using Mel-frequency Cepstrum in the MATLAB. The audio frame needs to be 30 msec with 50% overlap. For each frame, the MFCC (Mel-Frequency Cepstral Coefficients) should be found and gathered to acquire a feature vector that we can use for comparison. These vectors will enable us to recognize a spoken number by finding the minimum Euclidian distance between already stored (database)

feature vector and newly acquired feature vector. The general process is something like this:

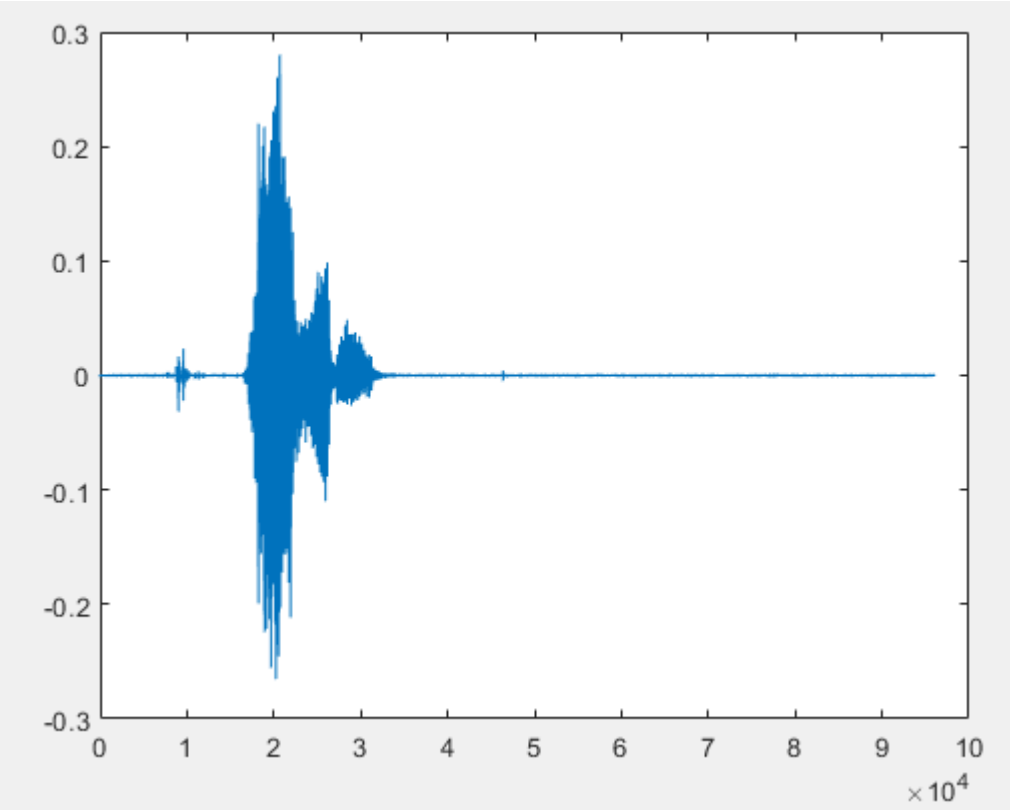


Note that our frequency range is 300Hz-4Khz because the human voice usually doesn't exceed these values.

6.1. Recording sound

Before starting implementing the algorithm, we first need to handle recording sounds and storing them for the process. We can record 3 seconds of audio like this:

```
recObj = audiorecorder(32000, 16, 1); %32k sampling rate and 16 bit
disp('Speak')
recordblocking(recObj, 3);% 3 seconds recording
dataRaw = getaudiodata(recObj, 'double');% Getting the data
```



3 seconds of recording (The spoken number is ZERO for this example)

As you can see, we have 3 seconds of a recording of a spoken number. However, we used 32K sampling rate we have 32000*3 = 96000 points of data. Also, most of the data are very close to zero which we don't need. We only need the relevant part of the recording and 16K of data. To get the relevant part we need to decide some threshold remove the unnecessary parts:

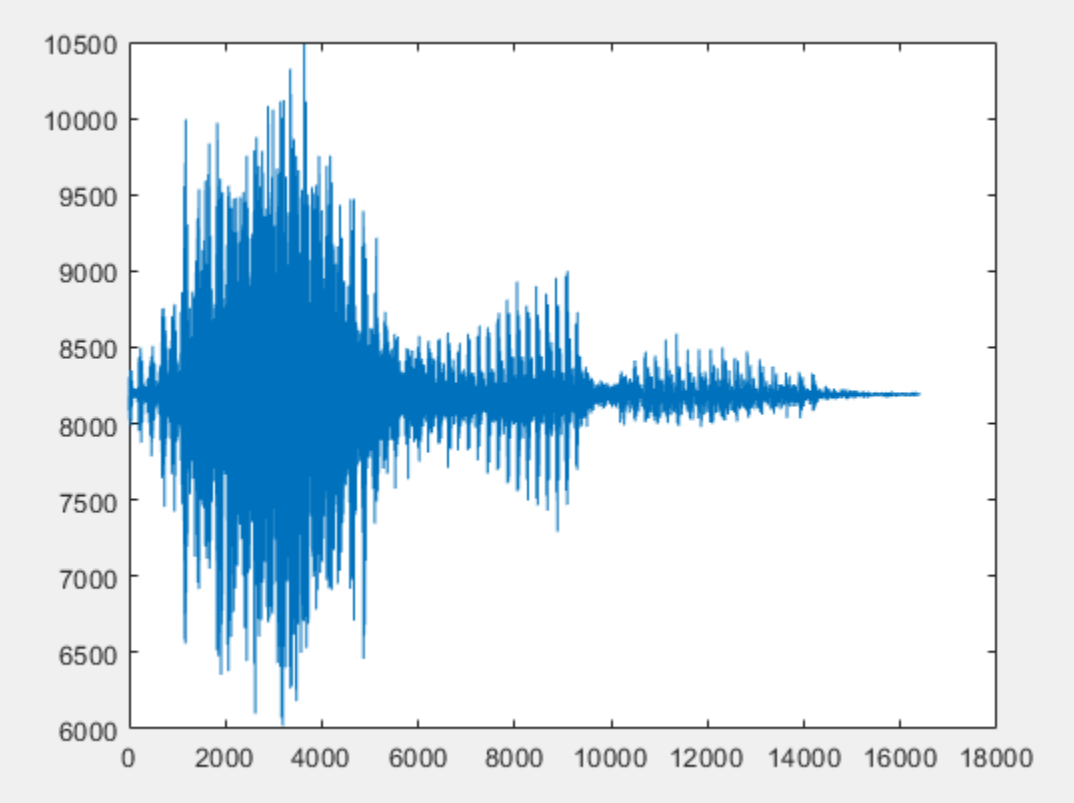
```
%%% for 8k data,we need it to 14 bit
data = floor((dataRaw.*(2^13))+2^13);

%%% Getting the relevant part from the audio clip
dataFinal = newSound(data,100);% Threshold = 100
```

newSound function takes the data and some threshold and return 16K long array only containing part of the recording that is relevant. newSound function is like this:

```
function soundFinal = newSound(soundData,threshold)
    result = ones(1,16384).* mean(soundData);
    count1 =1;
    count2 = 0;
    bool = 0;
    for i=1:length(soundData)
        if bool == 0
           if abs(soundData(i) -mean(soundData)) > threshold
               count2 = count2 + 1;
           end

           if count2 > 10
              bool = 1;
           end
        end
```

```
        if bool == 1
            if count1 < 16385
            result(count1) = soundData(i);
            count1 = count1 +1;
            else
                break;
            end
        end
    end
    soundFinal = result;
end
```

When we give 3 seconds of recording from the before to the function, it returns:



The relevant part of the recording

As you can see, now we have 16K number of data points and only the part where the actual number is spoken.

6.2. Constructing a database

We need a database of recordings in order to estimate the spoken number by finding the closest Euclidean distance between stored data and the new recording. For this purpose, we have prepared a database which has 5 recordings for the numbers from 0 to 9. In total, we have 50 recordings in our database. Our database code is like this:

```
index = 50; %% the recording is saved to this index
recObj = audiorecorder(32000, 16, 1); %32k sampling rate and 16 bit
disp('Speak')
recordblocking(recObj, 3);% 3 seconds recording
dataRaw = getaudiodata(recObj, 'double');% Getting the data

%%% Filter
dataRaw = dataRaw(1:end-1)-dataRaw(2:end);

%%% for 8k data,we need it to 14 bit
data = floor((dataRaw.*(2^13))+2^13);

%%% Getting the relevant part from the audio clip
dataFinal = newSound(data,100);% Threshold = 100

%%%% Saving to the samples
load('samplesounds.mat');
samplesound(index,:) = data_new;
save('samplesounds.mat','samplesound');
```

We are saving our recording by changing the index number by hand and storing them into samplesounds.mat for further use. Note that project member Yusuf Ziya Dilek's voice is recorded for in this database.

6.3. Framing for process

We need 512 frames with 50% overlap. We achieved this like this:

```
%%% Frames with 50% overlap
    index = 1;
    for i=1:63
        frame(i,:) =in_speech(index:index+511);
        index = index + 256; % for 50 percent overlap
    end
```

6.4. Hamming Window

Before the FFT process, we need to perform windowing. The reasons are explained in the Windowing part of this report. We used hamming windowing for this purpose simply constructing in in MATLAB like this:

```
hanning_window = floor(hann(512)'.*(2^16));
```

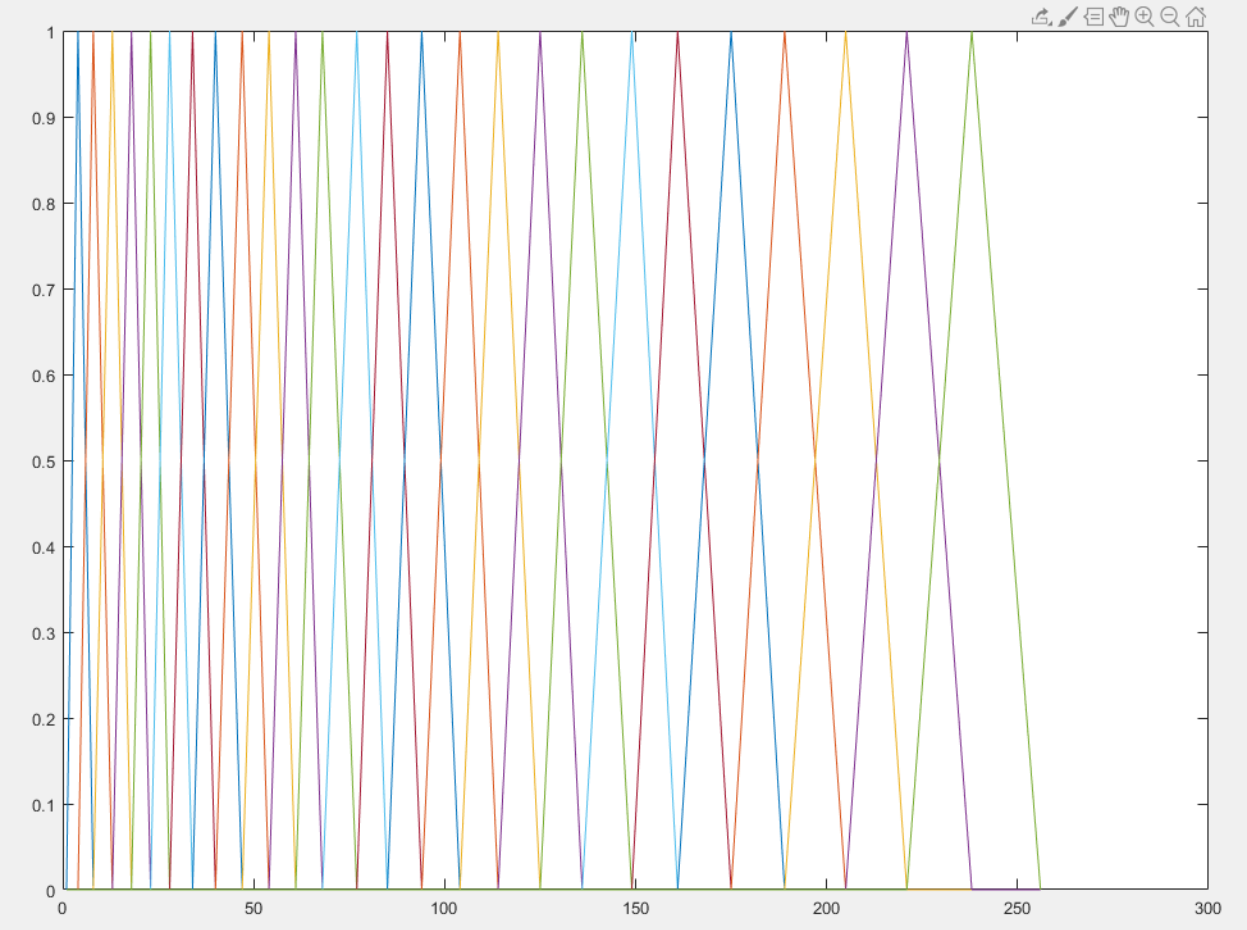
6.5. Mel Filters

We need to multiply the FFT of each frame with a Mel filter. For this purpose first, we constructed 8 Mel filter banks by hand but later constructed an algorithm that can provide us with any number of Mel filters automatically.

```
%%%% http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-
cepstral-coefficients-mfccs/#eqn1
    freqBand = [300 4000];
    nSample = 256;
```

```
    nFilter= 26;

    bandwidth = freqBand(2) - freqBand(1);

    melfreq = @(f) 1127*log(1 + f/700);
    invmel = @(m) 700*(exp(m/1127) - 1);

    melf_centers = linspace(melfreq(freqBand(1)), melfreq(freqBand(2)), nFilter + 2); %Mel filter
centers in Mel domain
    f_centers = invmel(melf_centers); %Filter centers in freq domain

    nCenters = round(((f_centers-freqBand(1))/bandwidth)*256);

    nCenters(1) = 1;
    melFiltersMatrix = zeros(nFilter,nSample);

    for i=1:nFilter
        melFiltersMatrix(i,(nCenters(i):nCenters(i+1))) = 0:1/(nCenters(i+1)-nCenters(i)):1;
        melFiltersMatrix(i,(nCenters(i+1):nCenters(i+2))) = 1:-1/(nCenters(i+2)-nCenters(i+1)):0;
    end
    filter = melFiltersMatrix;
```

Our frequency band is between 300Hz and 4kHz since this is the approximate range for human speech. We are using 26 Mel filters which is more than enough for our purposes. The Banks look like this:



Mel Filter Banks (26 of them)

6.6. Performing FFT and calculating the energy

In here, we need to perform FFT after windowing. We already constructed the windowing function previously by using MATLAB's hamming function. We need to multiply each frame with hamming window element-wise and 256-point FFT is taken for each frame. Then we can calculate the energy. Note that we will use the complex magnitude of the FFT result.

```
%%% taking the fft
    for i=1:63
        fftFrame(i,:) = abs(floor(fft((frame(i,:).*hanning_window))./2^16));
    end

    %%% Calculating Energy
    for i=1:63
        frameEnergy = [];
        for t=1:26 % 26 Mel filter banks % summing energies for each filter bank
            temp = sum(floor(fftFrame(i,1:256).*(filter(t,:).^2))); %% power calculation done
here
            frameEnergy = [frameEnergy temp];
        end
        energy_of_frame_tot(i,:) = frameEnergy; % total energy for a single frame
    end
```

6.7. Log and DCT modules

After taking the FFT, we simply need perform logarithm of the energy of the frames and take their DCT. We don't need the DCT coefficients between 14 and 26. First 13 frames is what we really need. Log and DCT needs to be performed for all off the 63 frames. Our MATLAB code is like this for this part:

```
%%% Taking Log10 then the DCT
    for i=1:63
        tempdata = energy_of_frame_tot(i,:);
        tempdata = floor(log10(tempdata+1).*128);
        tempdata = dct(tempdata); % taking DCT here
        tempdata(1) = 0;
        for j=14:26 %% get rid of the coefficients between 14 and 26
            tempdata(j) = 0;
        end
        tempdata = floor(tempdata);
        DCTResult = [DCTResult tempdata];
    end

    Melcoeff =DCTResult; % MFCC calculated here finally
```

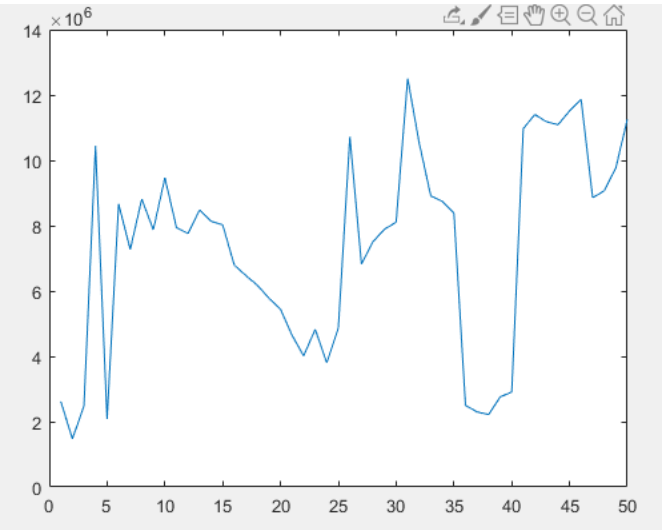
Finally, we have the MFCC (Mel-frequency cepstral coefficients). We can use this for comparisons.

6.8. Detection and comparison

We now have all the functions we need to perform speech recognition. We just need to write a main function that will record the spoken number and calculate the MFFC so that it can compare it with the recording previously saved into the database.

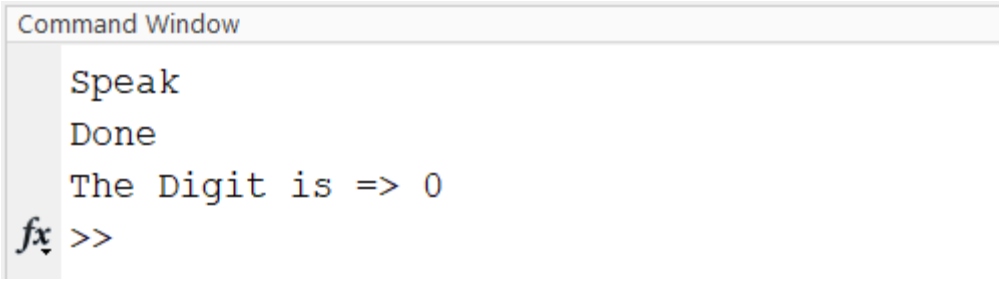
```
recObj = audiorecorder(32000, 16, 1);
disp('Speak')
recordblocking(recObj, 3);
dataRaw = getaudiodata(recObj, 'double');
dataRaw = filtering(dataRaw);
data = floor((dataRaw.*(2^13))+2^13);
disp('Done');

dataFinal = newSound(data,100);
soundsc(dataFinal,32000);    %% listen for confirmation
figure(1)
plot(dataFinal);             %% Plot for confirmation

load('samplesounds.mat');

vectors = [];
MeanSquareError = [];
featureVector = Mel_Freq(dataFinal); % Melcoefficients for the spoken number

% Melcoefficients for the database elements
for i=1:50
    vectors(i,:) = Mel_Freq(samplesound(i,:));
end

%%% Euclidian distance
for i=1:50
    MeanSquareError(i) = sum((vectors(i,:)-featureVector).^2);
end


figure(2)
plot(MeanSquareError)

[k,index] = min(sqrt(MeanSquareError));

fprintf('The Digit is => %d \n',ceil(index/5)- 1);
```

As you can see, we first record the sound and process it to get the relevant part. Then we found its MFCC with the mel_freq function. Then we found MFCC for the 50 elements in our database so that we can calculate the Euclidian distance between every entry in the database and the newly spoken number. The minimum distance should be the number we are looking for.



Euclidian Distance between newly spoken number and 50 database entries

For this example, the number ZERO is recorded. As you can see, the second database entry is minimum distance away from the spoken number. From 0 to 9 every number has 5 entry. The first 5 is zero. We get:



This is the expected result. Our algorithm works for other numbers as well.

6.9. Performance of the system

We have tested our system extensively and found that it works more than 95% of the time. However, the system can reach this kind of consistency only if the database is constructed with the voice of the speaker. For instance, we recorded our project member Yusuf Ziya Dilek's voice. When he speaks the system nearly always detects without any problems but other people may have difficulty especially if their voice level and the accent are different from him. This algorithm does not seem to be suited for very general use but it might work well enough if a very large database with varied voices is constructed. For our purposes, the performance of the system is more than enough. It actually exceeded our expectations. The system rarely makes mistakes when our project member speaks.

REFERENCES

(Sample)Sallen-Key Low-pass Filter Design Tool - Result -. [Online]. Available: http://sim.okawa-denshi.jp/en/OPstool.php. [Accessed: 17-Jun-2020].

"Crypto," *Practical Cryptography*. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#eqn1. [Accessed: 17-Jun-2020].

"Operational Amplifier Basics - Op-amp tutorial," *Basic Electronics Tutorials*, 01-May-2020. [Online]. Available: https://www.electronics-tutorials.ws/opamp/opamp_1.html?utm_referrer=https://www.google.com/. [Accessed: 17-Jun-2020].

"Sallen and Key Filter Design for Second Order RC Filters," *Basic Electronics Tutorials*, 11-Jul-2019. [Online]. Available: https://www.electronics-tutorials.ws/filter/sallen-key-filter.html. [Accessed: 17-Jun-2020].

MATLAB CODES

Full system and Lab Matlab

```
recObj = audiorecorder(32000, 16, 1);
disp('Speak')
recordblocking(recObj, 3);
dataRaw = getaudiodata(recObj, 'double');
dataRaw = filtering(dataRaw);
data = floor((dataRaw.*(2^13))+2^13);
disp('Done');

dataFinal = newSound(data,100);
soundsc(dataFinal,32000);    %% listen for confirmation
figure(1)
plot(dataFinal);             %% Plot for confirmation

load('samplesounds.mat');

vectors = [];
MeanSquareError = [];
featureVector = Mel_Freq(dataFinal); % Melcoefficients for the spoken number

% Melcoefficients for the database elements
for i=1:50
    vectors(i,:) = Mel_Freq(samplesound(i,:));
end

%%% Euclidian distance
for i=1:50
    MeanSquareError(i) = sum((vectors(i,:)-featureVector).^2);
end


figure(2)
stem(MeanSquareError)

[k,index] = min(sqrt(MeanSquareError));

fprintf('The Digit is => %d \n',ceil(index/5)- 1);
```

Function that handles MFCC algorithm

```
function Melcoeff = Mel_Freq( in_speech ) % function to calc MFCC % uses 63 samples
    hanning_window = floor(hann(512)'.*(2^16));
    frame = [];
    fftFrame = [];
    frameEnergy = [];
    DCTResult = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constructing % Mel Filter Banks two
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% different ways
%     filter = [];
%     filter(1,:) = [zeros(1,5) linspace(0,1,8) linspace(1,0,8)  zeros(1,235)];
%     filter(2,:) = [zeros(1,13) linspace(0,1,13) linspace(1,0,13)  zeros(1,217)];
%     filter(3,:) = [zeros(1,26) linspace(0,1,18) linspace(1,0,18)  zeros(1,194)];
%     filter(4,:) = [zeros(1,44) linspace(0,1,23) linspace(1,0,23)  zeros(1,166)];
%     filter(5,:) = [zeros(1,67) linspace(0,1,28) linspace(1,0,28)  zeros(1,133)];
%     filter(6,:) = [zeros(1,95) linspace(0,1,33) linspace(1,0,33)  zeros(1,95)];
%     filter(7,:) = [zeros(1,128) linspace(0,1,38) linspace(1,0,38)  zeros(1,52)];

%%%% http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-
cepstral-coefficients-mfccs/#eqn1
    freqBand = [300 4000];
    nSample = 256;
    nFilter= 26;

    bandwidth = freqBand(2) - freqBand(1);
```

```
    melfreq = @(f) 1127*log(1 + f/700);
    invmel = @(m) 700*(exp(m/1127) - 1);

    melf_centers = linspace(melfreq(freqBand(1)), melfreq(freqBand(2)), nFilter + 2); %Mel filter
centers in Mel domain
    f_centers = invmel(melf_centers); %Filter centers in freq domain

    nCenters = round(((f_centers-freqBand(1))/bandwidth)*256);

    nCenters(1) = 1;
    melFiltersMatrix = zeros(nFilter,nSample);

    for i=1:nFilter
        melFiltersMatrix(i,(nCenters(i):nCenters(i+1))) = 0:1/(nCenters(i+1)-nCenters(i)):1;
        melFiltersMatrix(i,(nCenters(i+1):nCenters(i+2))) = 1:-1/(nCenters(i+2)-nCenters(i+1)):0;
    end
    filter = melFiltersMatrix;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%% Frames with 50% overlap
    index = 1;
    for i=1:63
        frame(i,:) =in_speech(index:index+511);
        index = index + 256; % for 50 percent overlap
    end

    %%% taking the fft
    for i=1:63
        fftFrame(i,:) = abs(floor(fft((frame(i,:).*hanning_window))./2^16));
    end

    %%% Calculating Energy
    for i=1:63
        frameEnergy = [];
        for t=1:26 % 26 Mel filter banks % summing energies for each filter bank
            temp = sum(floor(fftFrame(i,1:256).*(filter(t,:).^2))); %% power calculation done
here
            frameEnergy = [frameEnergy temp];
        end
        energy_of_frame_tot(i,:) = frameEnergy; % total energy for a single frame
    end

    %%% Taking Log10 then the DCT
    for i=1:63
        tempdata = energy_of_frame_tot(i,:);
        tempdata = floor(log10(tempdata+1).*128);
        tempdata = dct(tempdata); % taking DCT here
        tempdata(1) = 0;
        for j=14:26 %% get rid of the coefficients  between 14 and 26
            tempdata(j) = 0;
        end
        tempdata = floor(tempdata);
        DCTResult = [DCTResult tempdata];
    end

    Melcoeff =DCTResult; % MFCC calculated here finally
end
```

```
clear all;
clc;

%%%% To listen to a specific index use this
% index = 16; load('samplesounds.mat'); soundsc(samplesound(index,:),32000);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

index = 50; %% the recording is saved to this index
recObj = audiorecorder(32000, 16, 1); %32k sampling rate and 16 bit
disp('Speak')
recordblocking(recObj, 3);% 3 seconds recording
dataRaw = getaudiodata(recObj, 'double');% Getting the data
```

```
%%% Filter
dataRaw = dataRaw(1:end-1)-dataRaw(2:end);

%%% for 8k data,we need it to 14 bit
data = floor((dataRaw.*(2^13))+2^13);

%%% Getting the relevant part from the audio clip
dataFinal = newSound(data,100);% Threshold = 100

%%% listen to the final form of the data
disp('Listen')
soundsc(dataFinal,32000); %% listen scaled data

%%% Plots
figure(1)
plot(dataRaw)
figure(2)
plot(data)
figure(3)
plot(dataFinal)

%%%% Saving to the samples
load('samplesounds.mat');
samplesound(index,:) = data_new;
save('samplesounds.mat','samplesound');
```

```
function soundFinal = newSound(soundData,threshold)
    result = ones(1,16384).* mean(soundData);
    count1 =1;
    count2 = 0;
    bool = 0;
    for i=1:length(soundData)
        if bool == 0
           if abs(soundData(i) -mean(soundData)) > threshold
               count2 = count2 + 1;
           end

           if count2 > 10
              bool = 1;
           end
        end

        if bool == 1
            if count1 < 16385
            result(count1) = soundData(i);
            count1 = count1 +1;
            else
                break;
            end
        end
    end
    soundFinal = result;
end
```

Lab FFT

```
figure(1)
fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\input.txt','r');
input = fscanf(fileID,'%d');
plot(input)

fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\output.txt','r');
fft_real_fpga = fscanf(fileID,'%d');
figure(2);
plot(fft_real_fpga)
hold on
```

```
fileID = fopen('C:\Users\Yusuf Ziya Dilek\Desktop\input.txt','r');
sine_fpga = fscanf(fileID,'%d');
sine_fpga = sine_fpga(1:512);
fft_real_matlab = real(fft(sine_fpga));

plot(fft_real_matlab)
legend('real fft of FPGA', 'real fft of MATLAB')
hold off
```

Mel banks

```
%%Parameters
freq_band = [300 4000];
nSample = 256;
nFilter= 26;
%%Initialization

bandwidth = freq_band(2) - freq_band(1);

melfreq = @(f) 1127*log(1 + f/700);
invmel = @(m) 700*(exp(m/1127) - 1);

melf_centers = linspace(melfreq(freq_band(1)), melfreq(freq_band(2)), nFilter + 2); %Mel filter
centers in Mel domain
f_centers = invmel(melf_centers); %Filter centers in freq domain

nCenters = round(((f_centers-freq_band(1))/bandwidth)*256);

nCenters(1) = 1;
melFiltersMatrix = zeros(nFilter,nSample);

for i=1:nFilter
    melFiltersMatrix(i,(nCenters(i):nCenters(i+1))) = 0:1/(nCenters(i+1)-nCenters(i)):1;
    melFiltersMatrix(i,(nCenters(i+1):nCenters(i+2))) = 1:-1/(nCenters(i+2)-nCenters(i+1)):0;
end
for i = 1:1:26
   plot(melFiltersMatrix(i,:),'DisplayName','melFiltersMatrix')
   hold on
end
```

Hamming Generator

```
fileID = fopen('hamming.txt','w');

coefficients = int16(round(hamming(512)*(2^15-1)));
coefficients = transpose(coefficients);
A = [0:511; coefficients];

fprintf(fileID, '%d => x"%x",\n', A);%%% hex
fclose(fileID);
```