An
Open Ended Assignment
on

“Simulate analog to digital conversion of speech signal”

Course: 20EC402L ANALOG AND DIGITAL COMMUNICATION LAB

Prepared By
Batch - B2
221 – Kanawade Nandini
223 – Kedari Kanchan
229 – Lomate Mayuri

Department of Electronics and Telecommunication Engineering,
MKSSSs, Cummins College of Engineering for Women, Pune,

Academic Year – 2022-23

CONTENTS

1.An Overview

“Converting the audio signal to a binary stream is necessary in digital communication systems because digital signals can be easily processed, transmitted, and stored using computer systems. Digital signals can be efficiently compressed, encrypted, and transmitted over communication channels with minimal errors compared to analog signals.”

Below is the basic flow of the MATLAB simulation.

1. Firstly, the code creates a file called **'output.txt'** and initializes several variables. The variable fs specifies the sampling frequency of the audio recording, channel specifies the number of channels in the recording, datatype specifies the data type of each sample in the recording, and **nbits** specifies the number of bits per sample.

2. Next, the code uses the **audiorecorder()** function to create an object called recorder, which records audio for a duration specified by the variable AudioTime. The user is prompted to start speaking and then the code waits for the recording to finish. After the recording is complete, the audio data is stored in an array called **x** and is saved to a .wav file called **'test.wav'**.

3. The code then reads the data from the 'test.wav' file using the **audioread()** function and takes the FFT of the audio signal to get the frequency domain representation of the signal. The magnitude of the FFT is computed using the **abs()** function and the frequency axis is generated using the **linspace()** function. The index of the maximum value of the magnitude is found using the **max()** function and the corresponding frequency is computed. The modulating frequency of the audio is calculated and stored in the variable fm, which is then printed to the output file using the **fprintf()** function.

4. The code then plots the sampled speech signal in the time domain using the **plot()** function.

5. Next, the code quantizes the audio signal using the **quantiz()** function. The number of quantization levels and the quantization step size are computed based on the maximum value of the signal and the number of bits used for quantization. The quantized signal is then normalized and converted to an 8-bit binary stream using the **de2bi()** function. The binary stream is stored in a variable called **x_binary_stream** and is printed to the output file using the fprintf() function.

6. Finally, the code plots a random range of the binary stream using the **plot()** function. The starting point for the range is selected randomly and 50 continuous values from the binary stream are plotted. The resulting plot shows the bit stream in the time domain.

2. Select a speech signal.

```
% Clearing the Screen
clc
close all
warning off

% Create a file
fid = fopen('output.txt', 'wt');
```

1. clc
   - Explanation: This function clears the Command Window, providing a clean slate for displaying new outputs.

2. close all
   - Explanation: This function closes all open figure windows. It ensures that any previously displayed figures are closed before running new code that may generate plots or graphs.

3. warning off
   - Explanation: This function turns off the display of warning messages in MATLAB. It suppresses any warning messages that might occur during the execution of the code.

4. fid = fopen('output.txt', 'wt')
   - Syntax: fid = fopen(filename, permission)
   - Explanation: This function opens a file specified by the filename and returns a file identifier (fid) that can be used to perform operations on the file. In this case, it opens the file named 'output.txt' in write text mode ('wt'). This allows writing text data to the file. The returned fid is stored in the variable fid for later use.

These functions are used in the given code snippet to perform initial setup tasks like clearing the screen, closing all figures, turning off warnings, and creating/opening a file for writing (output.txt).

```
% Sampling Frequnency (Hz)
fs=8000;

% Mono-audio
channel=1;

% Each sample will have a value from 1-255
datatype='uint8';

% Number of Bits per sample
nbits=16;

% Duration of Recording
AudioTime=2;
```

These variable assignments define important parameters for recording and processing the audio signal, such as the sampling frequency, number of channels, data type, number of bits per sample, and recording duration.

1. fs = 8000;
   - Explanation: This line assigns the value 8000 to the variable fs , which represents the sampling frequency in Hertz (Hz). The sampling frequency determines the number of samples taken per second when recording or processing audio signals.

2. channel = 1;
   - Explanation: This line assigns the value 1 to the variable channel . It indicates that the audio is mono, meaning it has a single channel. In stereo audio, there would be two channels.

3. datatype = 'uint8';
   - Explanation: This line assigns the string 'uint8' to the variable datatype . It specifies the data type of each sample in the audio signal. 'uint8' stands for unsigned 8-bit integer, which means each sample can have a value ranging from 0 to 255.

4. nbits = 16;
   - Explanation: This line assigns the value 16 to the variable nbits . It represents the number of bits used to represent each sample in the audio signal. In this case, each sample is represented using 16 bits, allowing for a greater dynamic range and precision compared to lower bit depths.

5. AudioTime = 2;
   - Explanation: This line assigns the value 2 to the variable AudioTime . It represents the duration of the recording in seconds. In this case, the recording duration is set to 2 seconds. We can set this to our desired value.

```
% Making an object of audiorecorder() function
recorder=audiorecorder(fs,nbits,channel);
disp('Start Speaking ');

%Stop Recording
recordblocking(recorder,AudioTime);
disp('Recorded.');

% Storing the audio data in an array
x=getaudiodata(recorder,datatype);

% store audio in .wav file
audiowrite('test.wav',x,fs);

filename = 'test.wav';
[x,fs] = audioread(filename);
```

These function calls are used to record audio, retrieve the recorded audio data, save it as a WAV file, and read an audio file for further processing.

1. recorder = audiorecorder(fs, nbits, channel);
   - Syntax: recorder = audiorecorder(Fs, nBits, nChannels)
   - Explanation: This line creates an object recorder using the audiorecorder() function. The function takes three arguments: fs (sampling frequency), nbits (number of bits per sample), and channel (number of audio channels). The object recorder represents an audio recording device with specified properties.

2. disp('Start Speaking ');
It serves as a message prompt for the user to start speaking or providing input for the audio recording.

3. recordblocking(recorder, AudioTime);
   - Syntax: recordblocking(recorderObj, length)
   - Explanation: This line records audio using the recordblocking() function. It takes two arguments: recorder (the audio recording object created in the previous step) and AudioTime (the duration of the recording in seconds). The function records audio for the specified duration and blocks the code execution until the recording is complete.

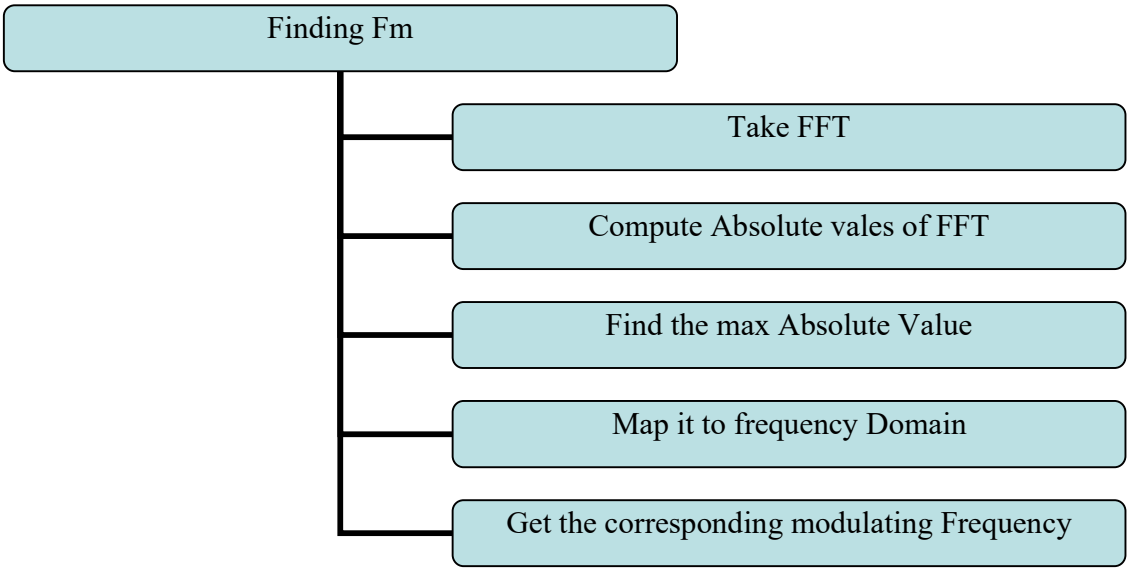
4. disp('Recorded.');
This line displays the text 'Recorded.' in the Command Window using the disp() function. It serves as a message indicating that the audio recording has been completed.

5. x = getaudiodata(recorder, datatype);
   - Syntax: y = getaudiodata(recorder, dataType);
   - Explanation: This line retrieves the audio data from the recorder object using the getaudiodata() function. It takes two arguments: recorder (the audio recording object) and datatype (the desired data type for the audio samples). The function returns the audio data as an array and assigns it to the variable x .

6. audiowrite('test.wav', x, fs);
   - Syntax: audiowrite(filename, y, Fs);
   - Explanation: This line writes the audio data to a WAV file using the audiowrite() function. It takes three arguments: 'test.wav' (the name of the output file), x (the audio data array), and fs (the sampling frequency). The function saves the audio data as a WAV file with the specified filename.

7. [x, fs] = audioread(filename);
   - Syntax: [y,Fs] = audioread(filename,samples);
   - Explanation: This line reads an audio file using the audioread() function. It takes one argument: filename (the name of the audio file to read). The function returns two values: x (the audio data array) and fs (the sampling frequency). The returned values are assigned to the variables x and fs , respectively.

3. Finding its modulating frequency fm.




```
% Take the FFT of the audio signal to get fm
X = fft(x);

% Find the absolute values
mag_X = abs(X);

% Frequnency Axis
freq_axis = linspace(0, fs/2, length(mag_X)/2);

% GEtting index of the max number
[~, max_index] = max(mag_X(1:length(mag_X)/2));

% Map the index on the frequency Axis
max_freq = freq_axis(max_index);
omega = max_freq * 2 * pi;
fm = omega / 2;
fprintf(fid, 'The modulating frequency of the audio is: %f Hz\n', fm);
```

These function calls are used to analyze the audio signal's frequency content, find the modulating frequency, and write the result to a file.

1. X = fft(x);
   - Syntax: Y=ff(x);
   - Explanation: This line applies the FFT (Fast Fourier Transform) to the audio signal x . The fft() function computes the complex spectrum of the input signal. The resulting spectrum is stored in the variable X .

2. mag_X = abs(X);
   - Syntax: output = abs(input);
   - Explanation: This line computes the absolute values of the complex spectrum X obtained from the FFT. The abs() function calculates the magnitude of each complex element in X and stores the result in the variable mag_X .

3. freq_axis = linspace(0, fs/2, length(mag_X)/2);
   - Syntax: output = linspace(start, stop, n);
   - Explanation: This line generates a frequency axis freq_axis using the linspace() function. It takes three arguments: 0 (the starting frequency), fs/2 (the maximum frequency, which is half of the sampling frequency), and length(mag_X)/2 (the number of frequency points). The function evenly divides the frequency range from start to stop into n points and stores the result in freq_axis .

4. [~, max_index] = max(mag_X(1:length(mag_X)/2));
   - Syntax: [max_value, max_index] = max(input);
   - Explanation: This line finds the maximum value and its corresponding index from the magnitude spectrum mag_X . The max() function is applied to mag_X(1:length(mag_X)/2) to find the maximum value within the first half of the spectrum. The resulting maximum value is ignored by using the ~ placeholder, and the corresponding index is stored in the variable max_index .

5. max_freq = freq_axis(max_index);
   - Syntax: variable = array(index);
   - Explanation: This line extracts the value at max_index from the frequency axis freq_axis and assigns it to the variable max_freq . It represents the frequency corresponding to the maximum magnitude in the audio spectrum.

6. omega = max_freq * 2 * pi;
   - Syntax: variable = expression;
   - Explanation: This line calculates the angular frequency omega by multiplying max_freq with 2 * pi . It converts the frequency from Hz to radians per second.

7. fm = omega / 2;
   - Syntax: variable = expression;

- Explanation: This line computes the modulating frequency fm by dividing omega by 2. It represents half of the angular frequency.

8. fprintf(fid, 'The modulating frequency of the audio is: %f Hz\n', fm);
- Syntax: fprintf(fileID, format, data);
- Explanation: This line writes a formatted string to the file identified by fid . The string contains the modulating frequency value fm in the specified format %f (float) followed by the unit 'Hz'. The newline character \n indicates the end of the line.

NOTE: When taking the FFT of a real-valued signal, the spectrum is symmetric about the Nyquist frequency (half the sampling frequency). Therefore, the second half of the spectrum is a mirror image of the first half, and contains redundant information. The second half of the spectrum can be safely discarded without losing any information.

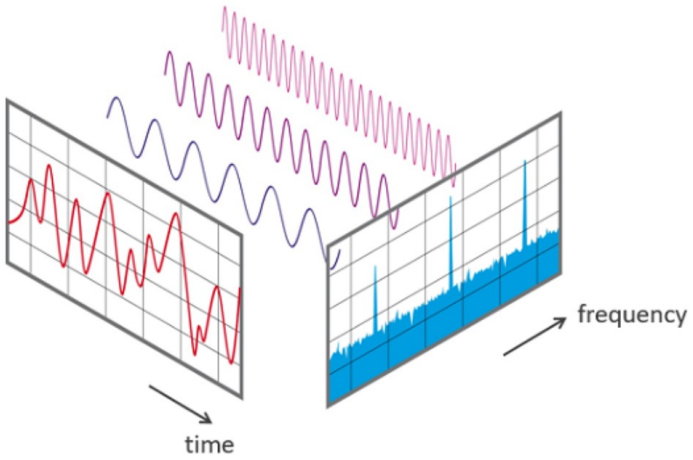


Fig. 1: FT Demostration

```
%Plotting the Sampled Speech Signal.
figure(1);
t = linspace(0, length(x)/fs, length(x));
plot(t, x);
xlabel('Time');
ylabel('Amplitude');
title('Sampled Speech signal');
```

These function calls are used to create a plot of the sampled speech signal with appropriate axis labels and a title. The time axis is generated using linspace() , and the signal is plotted using plot() . The resulting plot provides a visualization of the speech signal's amplitude over time.

1. figure(1);
   - Explanation: This line creates a new figure window with the specified number. In this case, it creates figure number 1.

2. t = linspace(0, length(x)/fs, length(x));
   - Syntax: output = linspace(start, stop, n);
   - Explanation: This line generates a time axis t using the linspace() function. It takes three arguments: 0 (the starting time), length(x)/fs (the maximum time, calculated based on the length of the audio signal x and the sampling frequency fs), and length(x) (the number of time points). The function evenly divides the time range from start to stop into n points and stores the result in t .

3. plot(t, x);
   - Syntax: plot(x, y);
   - Explanation: This line creates a line plot of the sampled speech signal. The plot() function takes two arguments: t (the time axis) and x (the sampled speech signal). It plots the signal as a function of time.

4. xlabel('Time');
   - Explanation: This line sets the label for the x-axis of the plot. The xlabel() function takes a string argument and assigns it as the label for the x-axis. In this case, the label is set as 'Time' .

5. ylabel('Amplitude');
   - Explanation: This line sets the label for the y-axis of the plot. The ylabel() function takes a string argument and assigns it as the label for the y-axis. In this case, the label is set as 'Amplitude' .

6. title('Sampled Speech signal');
   - Explanation: This line sets the title for the plot. The title() function takes a string argument and assigns it as the title for the plot. In this case, the title is set as 'Sampled Speech signal' .

4. Perform sampling &quantization by assigning appropriate quantization levels

```
%defining quantization levels
nQBits = 8;
nLevels = 2^nQBits;

% Compute Quantization step size
maxVal = max(abs(x));
stepSize = maxVal / (nLevels -1);

%Quantize Signals
x_quantized = quantiz(x, linspace(-maxVal, maxVal, nLevels-1));
```

These function calls are used to determine the quantization levels, compute the quantization step size, and perform quantization on the sampled speech signal. The resulting quantized signal, x_quantized , represents a discretized version of the original signal with a reduced number of possible amplitude values.

1. nQBits = 8;
   - Explanation: This line assigns the value 8 to the variable nQBits . It represents the number of quantization bits, indicating the desired resolution for quantizing the signal.

2. nLevels = 2^nQBits;
   - Explanation: This line calculates the number of quantization levels nLevels based on the number of quantization bits nQBits . It raises 2 to the power of nQBits using the ^ operator and assigns the result to nLevels . The number of quantization levels determines the range of values that the quantized signal can take.

3. maxVal = max(abs(x));
   - Syntax: variable = max(input);
   - Explanation: This line finds the maximum absolute value of the sampled speech signal x using the max() function. The abs() function is applied to x to ensure that negative values are treated as positive. The resulting maximum value is stored in the variable maxVal .

4. stepSize = maxVal / (nLevels - 1);
   - Explanation: This line computes the quantization step size stepSize by dividing the maximum value maxVal by nLevels - 1 . The quantization step size represents the interval between adjacent quantization levels, determining the resolution of the quantization process.

5. x_quantized = quantiz(x, linspace(-maxVal, maxVal, nLevels-1));
   - Syntax: output = quantiz(input, quantLevels);
   - Explanation: This line quantizes the sampled speech signal x using the quantiz() function. It takes two arguments: x (the input signal) and linspace(-maxVal, maxVal, nLevels-1) (a vector specifying the quantization levels). The linspace() function generates nLevels-1 equally spaced quantization levels between -maxVal and maxVal . The quantiz() function assigns each sample in x to its nearest quantization level and stores the quantized signal in x_quantized .

5. Convert the generated quantized signal into stream of Binary bits

```
%converting quantized signal to 8 bit binary stream
nBBits = 8;
x_quantized_norm = (x_quantized - min(x_quantized)) /
(max(x_quantized) - min(x_quantized));
x_binary = de2bi(round(x_quantized_norm * (2^nBBits-1)),
nBBits, 'left-msb');
x_binary_stream = x_binary(:)';

%Displaying binary stream
fprintf(fid, 'Binary Stream: \n');
fprintf(fid, '%d ', x_binary_stream);
fprintf(fid, '\n');
```

These function calls are used to convert the quantized signal to an 8-bit binary stream and write it to a file. The binary stream is generated by normalizing the quantized signal, converting it to binary using de2bi() , and then reshaping it into a stream format.

1. nBBits = 8;
   - Explanation: This line assigns the value 8 to the variable nBBits . It represents the number of bits per sample in the binary stream, indicating the desired binary representation precision.

2. x_quantized_norm = (x_quantized - min(x_quantized)) / (max(x_quantized) - min(x_quantized));
   - Syntax: variable = (input - min(input)) / (max(input) - min(input));
   - Explanation: This line normalizes the quantized signal x_quantized by subtracting its minimum value and dividing by the range between its minimum and maximum values. The purpose is to scale the quantized signal to the range of [0, 1], which will facilitate the subsequent binary conversion.

3. x_binary = de2bi(round(x_quantized_norm * (2^nBBits-1)), nBBits, 'left-msb');
   - Syntax: output = de2bi(input, nBits, bitOrder);
   - Explanation: This line converts the normalized quantized signal to an 8-bit binary representation using the de2bi() function. It takes three arguments: round(x_quantized_norm * (2^nBBits-1)) (the input decimal values to convert), nBBits (the number of bits per sample), and 'left-msb' (the bit order, where 'left-msb' stands for Most Significant Bit first). The function converts each decimal value to its binary representation and stores the result in x_binary .

4. x_binary_stream = x_binary(:)';
   - Explanation: This line reshapes the x_binary matrix into a binary stream by concatenating its columns into a single row vector using the (:) notation. The resulting binary stream is stored in x_binary_stream .

5. fprintf(fid, 'Binary Stream: \n');
   - Syntax: fprintf(fileID, format, data);
   - Explanation: This line writes formatted data to the file specified by fileID . It uses the fprintf() function, where fileID is the identifier of the file opened with fopen() , format is a string specifying the format of the data, and data is the data to be written. In this case, it writes the string 'Binary Stream: \n' to the file.

6. fprintf(fid, '%d ', x_binary_stream);
   - Syntax: fprintf(fileID, format, data);
   - Explanation: This line writes the binary stream x_binary_stream to the file in a formatted manner. The %d format specifier is used to represent integer values. The data is written as space-separated integers, with each integer representing a bit in the binary stream.

7. fprintf(fid, '\n');
   - Syntax: fprintf(fileID, format);
   - Explanation: This line writes a newline character (\n) to the file. It adds a line break after writing the binary stream to separate it from subsequent data.

```
%Ploting a random range of the bitStream
% generate a random starting point within the range of the length of the binary stream
min_x = randi([1, length(x_binary_stream)-50]);

% select 100 continuous values from the binary stream starting from the random point
x_binary_substream = x_binary_stream(min_x:min_x+49);

% Ploting the Bit Stream
figure(2);
time = linspace(0,50,50);
plot(time, x_binary_substream, 'k','LineWidth',1);
xlabel('Time');
ylabel('Amplitude');
title('Bit Stream');

% Close the file
fclose(fid);
```

These lines of code generate a random starting point within the length of the binary stream and then select a range of 50 continuous values from that starting point. The resulting substream is assigned to the variable x_binary_substream, which can be used for further processing or plotting. Then the selected range of the bit stream x_binary_substream against the time vector time and set labels and a title for the plot. Finally, the file that was opened earlier is closed using fclose() .

1. min_x = randi([1, length(x_binary_stream)-50]);
   - Syntax: variable = randi([min, max]);
   - Explanation: This line generates a random integer within the range from 1 to the length of the binary stream (length(x_binary_stream)) minus 50. The randi() function is used with the range specified as [1, length(x_binary_stream)-50] . The resulting random integer is assigned to the variable min_x , which represents the starting index of the selected range.

2. x_binary_substream = x_binary_stream(min_x:min_x+49);
   - Syntax: subvector = vector(startIndex:endIndex);
   - Explanation: This line extracts a substream from the binary stream x_binary_stream . It selects a range of values from min_x (the random starting index) to min_x + 49 (the index that is 49 elements ahead). The resulting substream is assigned to the variable x_binary_substream .

3. Ploting the Bit Stream

4. fclose(fid);
   - Syntax: fclose(fileID);
   - Explanation: This line closes the file specified by fileID . The fclose() function is used to close the file that was previously opened with fopen() . In this case, the file identifier fid is passed as an argument to close the file.

Overall, the code performs audio processing tasks including recording and saving audio, calculating the modulating frequency, and quantizing the audio signal to an 8-bit binary stream. The results are saved to an output file and plotted for visualization purposes.

6. Reference

The complete document is been drafted by taking the references from the website *https://in.mathworks.com/help/matlab/*.
To analyze and this design system we used MATLAB R2022b version is used.