EE 430 Digital Signal Processing Term Project

Part 2 Report

Real Time Transmission of Acoustic Waves

Rafet Kavak 2166783 – Ersin Keskin 2166817
Middle East Technical University
Department of Electrical and Electronics Engineering
Ankara, Turkey
E-mail: e216678@metu.edu.tr - e216681@metu.edu.tr

***Abstract*— This report explains Frequency-Shift Keying (FSK), Binary Phase-Shift Keying (BPSK) and Quadrature Phase-Shift Keying (QPSK) modulation techniques for the audio signals.**

***Keywords*—Modulation, Demodulation, Preamble, Filtering, FSK, BPSK, QPSK.**

I. INTRODUCTION

Real time signal processing is widely used in many areas especially in daily life. From phone calls to voice messages, it becomes indispensable part of the life. Therefore, this project is very good opportunity to understand how real time applications were made basically. In the previous part of the project, system requirements were obtained, and literature research were done for the rest. In the second part of the project, transmission of the recorded voice signal was tried without any constraints in MATLAB. According to the system which was designed in part 1, required modulation methods were implemented and tested. Although the result was not crystal clear, modulation and demodulation of the voice message were completed within the restrictions. In this report, three modulation techniques which are Frequency-Shift Keying (FSK), Binary Phase-Shift Keying (BPSK) and Quadrature Phase-Shift Keying (QPSK) will be explained detailed with the necessary MATLAB codes and figures.

II. MODULATION TECHNIQUES

For the sake of consistency, "430_dnm.wav" audiofile will be used through the report for three modulation techniques.

```
% Input                                  .wav --> X
Fs=8*1e3; % Sampling Frequency : 8kHz
[X, Fs]=audioread('430_dnm.wav'); % Loading
sound file
```

By using "audioread" command of MATLAB "430_dnm.wav" audiofile was quantized and it can be seen in the Figure 1.

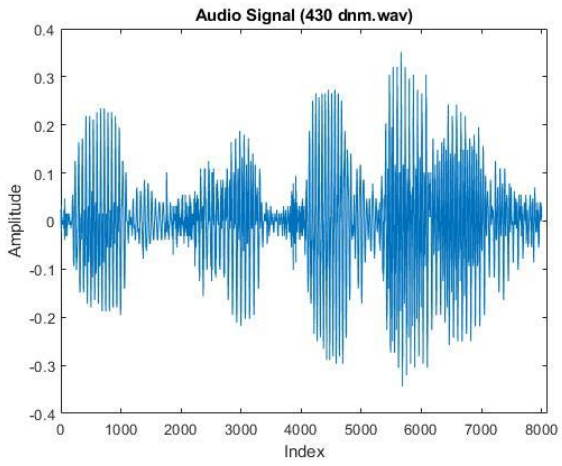

Figure 1: Quantized audio file

A. Frequency-Shift Keying (FSK)

Frequency Shift Keying FSK is the digital modulation technique in which the frequency of the carrier signal varies according to the digital signal changes. The output of a FSK modulated wave is *High* in frequency for a *binary High* input and is *Low* in frequency for a *binary Low* input. The binary 1s and 0s are called Mark and Space frequencies.

In this section, the developed code will be presented alongside with representative figures of the several sequences that is formed throughout the modulation process.

```
Clc, clear;

% Initializations
% Sampling Frequency
Fs = 8*1e3;
% Bit
b_p = 1e-6;   % Period
b_r = 1/ b_p; % Rate
% Amaunt of bit increment
inc = 100;
% Time step
t_step = b_p / 1e2;
```

```
% Carrier
carr_amp = 1;    % Amplitude
f1 = b_r * 8;    % freq (bit == 1)
f2 = b_r * 2;    % freq (bit == 0)
t  = b_p /(inc-1) : b_p / (inc-1) : b_p;
```

Initialization block includes several parameters that going to be used throughout the code. Please refer to this block whenever needed.

```
% LPC_tx
[y, Fs] = audioread('430_dnm.wav');
LPC_out = LPC_tx_s(y);
figure, stem(LPC_out);
xlim([0 100]), ylim([-0.1 1.1]);
title('430dnm as an Input to LPCtx ');
```

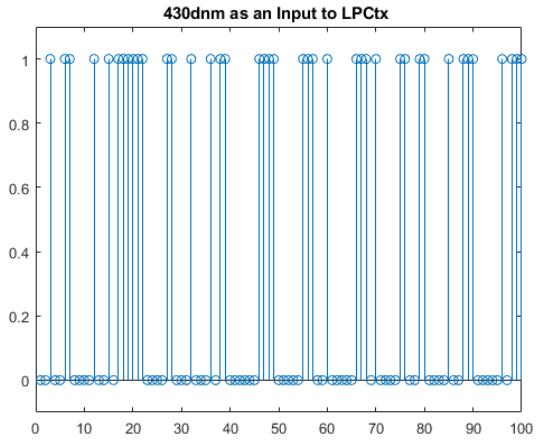

Figure 2: LPC_tx output when the input is selected as 430_dnm.wav file

The input signal is quantized in the LPC_tx block. This represents the initial step to modulate the signal with FSK, whose characteristics representative characteristics is illustrated in Figure 2.

```
% Modulation

% Preamble Addition
seq = [1 0 1 0 1 0 1 0 ];
pre_FSK_m = [seq transpose(LPC_out)];
figure, stem(pre_FSK_m);
xlim([0 100]), ylim([-0.1 1.1]);
title('Preamble addition to the Signal ');
```

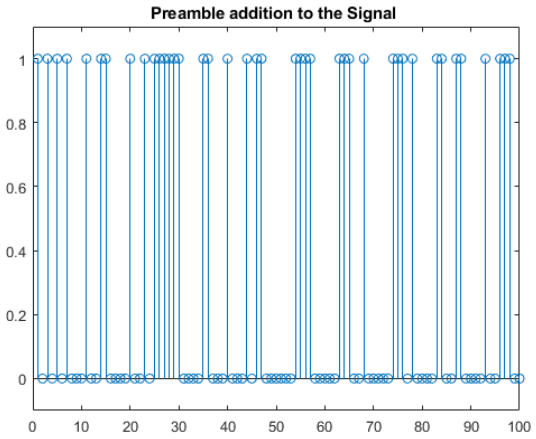

Figure 3: Preamble added signal

In order for the receiver side of the communication couple to detect the presence of the logical input, the receiver side shall transmit a predefined starting signal which is called preamble. In this block, the preamble sequence is combined with the data to be transmitted into one sequence. The resultant representative signal can be seen in Figure 3. Note that, this signal corresponds to preamble added sequence provided in Figure 2.

```
% Frequency Shift Keying
out_FSK_m = [];
for i = 1 : length(pre_FSK_m)
    if pre_FSK_m(i) == 1
        y = carr_amp * cos(2*pi * f1 * t);
    else
        y = carr_amp * cos(2*pi * f2 * t);
    end
    out_FSK_m = [out_FSK_m y];
end
figure, plot(out_FSK_m);
xlim([0 1600]), ylim([-1.1 1.1]);
title('Frequency Shift Modulated Output');
```

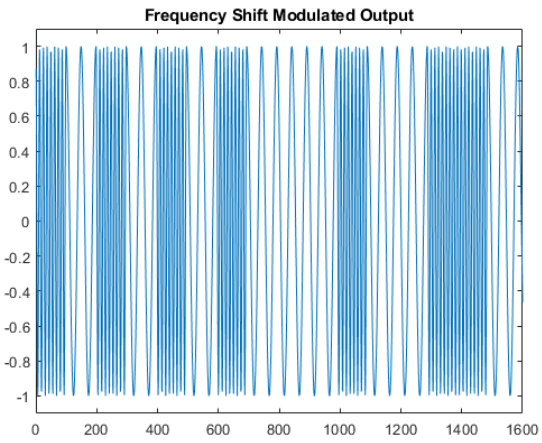

Figure 4: FSK modulation output signal

The sequence that is obtained by preamble addition is now ready to be transmitted. In order to convey the information to the receiver part, the bits are modulated by a Sinusoidal.

However, the 1' s are modulated with a frequency f1, whereas the 0' s are modulated with a frequency f2 in order to store the information properly. Notice that, $\Delta f = f_1 - f_2$ has a significant difference so that a strict distinction can be observed while detecting the corresponding bits in the receiver side. Referring to Figure 4, the data is stored in the fluctuations in the frequency of the modulated signal.

```
% Contribution of preamble
det_pre_FSK_m = [];
for i = 1 : length(seq)
    if seq(i) == 1
        y = carr_amp * cos(2*pi * f1 * t);
    else
        y = carr_amp * cos(2*pi * f2 * t);
    end
    det_pre_FSK_m = [det_pre_FSK_m y];
end
figure, plot(det_pre_FSK_m);
xlim([0 800]), ylim([-1.1 1.1]);
title('Contibution of Preamble');
```

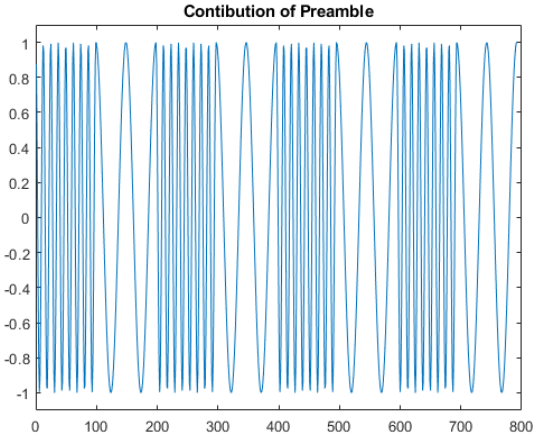

Figure 5: Predefined Preamble characteristic

Referring to Figure 4, the starting bits is consisting of preamble signal whose characteristics is provided in Figure 5. This information is to be used in the receiver part in order to be informed that the logical transmission is about to occur.

```
% Demodulation
in_FSK_d = out_FSK_m;

% Preamble
sn_FSK =fliplr(det_pre_FSK_m(2:end));
sn_FSK = [sn_FSK(1) sn_FSK];

buff_FSK_d = conv(sn_FSK, in_FSK_d);
figure, plot(buff_FSK_d);
xlim([0 2.5*1e3]);
title('Preamble Detection');

% Obtaining data part
[VAL, IDX] = max(buff_FSK_d);   % The maximum
value is stored in IDX
in_FSK_d = in_FSK_d(IDX : end); % Preamble is
discarded
```

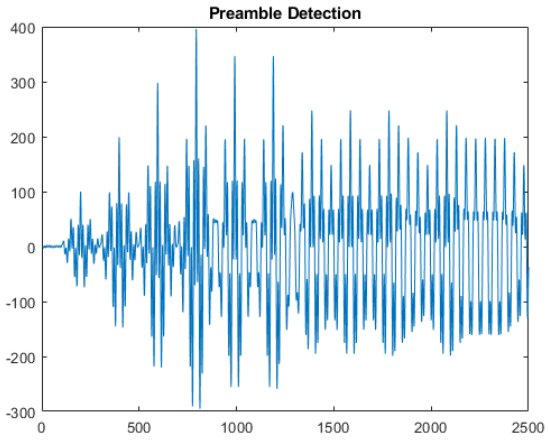

Figure 6: Peak detection of preamble

In the demodulation block, the received data is to be retransformed into analog signals in order for the speaker to operate properly. The first step to accomplish this task is to detect the presence of the preamble signal. One way to detect the such a signal is to have a filter in the receiver side that informs the system accordingly. When the impulse response of this filter is adjusted specifically to the circularly time reversed version of the preamble sequence, the output gives rise to a peak when the predefined preamble is to pass from this filter. Thus, the origin for a logical data receival is detected when the peak value of the received signal has a certain amount. The data received before the peak can be safely ignored, and the logical data can be stored afterwards.

```
% Detector
det_FSK_d=[];
for i = length(t) : length(t) : length(in_FSK_d)
  det1_FSK_d = cos(2*pi * f1 * t);
% carrier signal for information 1
  mult_det1_FSK_d = det1_FSK_d .* in_FSK_d((i-
(length(t)-1)) : i);

  det2_FSK_d = cos(2*pi * f2 * t);
% carrier signal for information 0
  mult_det2_FSH_d = det2_FSK_d .* in_FSK_d((i-
(length(t)-1)) : i);

  trapz_det1_FSK_d = trapz(t, mult_det1_FSK_d);
% intregation
  trapz_det2_FSK_d = trapz(t, mult_det2_FSH_d);
% intregation
  round_det1_FSK_d = round(2 *
trapz_det1_FSK_d/b_p);
  round_det2_FSK_d = round(2 *
trapz_det2_FSK_d/b_p);

  % Logic level detection  (0+carr_amp)/2 or
```

```
(carr_amp+0)/2
  if round_det1_FSK_d > carr_amp/2
    bit_detect = 1;
  elseif(round_det2_FSK_d > carr_amp/2)
    bit_detect = 0;
  end
  det_FSK_d = [det_FSK_d bit_detect];
end
figure, plot(det_FSK_d);
xlim([0 100]), ylim([-0.1 1.1]);
title('Signal at the Output-end of
Demodulator');
```

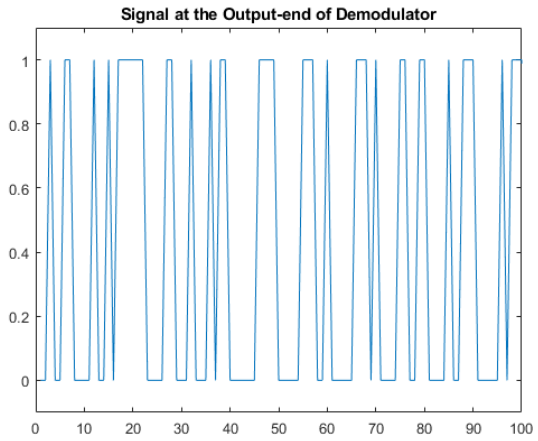

Figure 7: FSK Demodulation output

Having wiped out the preamble, the remaining data should be digitalized so that the LPC_rx operate properly. The detection of the corresponding bits is done in this block by first multiplying the received signal again with the modulation Sinusoidal signals, then integrating the resultant signal numerically with respect to time by *trapz* function readily installed in MATLAB. The bits can now be detected by rounding the values that are greater than half of the carrier amplitude.

```
% LPC_rx
det_FSK_d = transpose(det_FSK_d);
synth_speech_agr=LPC_rx_s(det_FSK_d);
p = audioplayer(synth_speech_agr, Fs);
figure, plot(synth_speech_agr);
title('FSK Modulated Signal');
play(p);
```

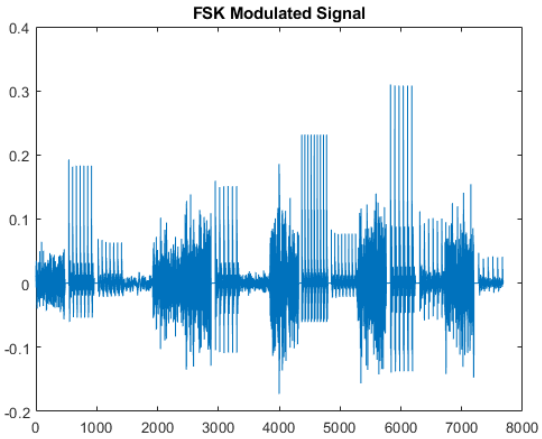

Figure 8: Signal at the Output-end of LPC_rx

The overall output characteristics of the FSK modulation is provided in Figure 8. The resultant sound when this signal is fed to speaker is understandable enough although the LPC_rx has distorted the input signal (refer to Figure 1) significantly.

B. Binary Phase-Shift Keying (BPSK)

In BPSK Modulation, phase of the carrier wave is changed according to the binary inputs (1 or 0). To do this operation, audio signal was processed with many operations. As can be seen in the Figure 9, audio signal was quantized at first with "audioread" command of MATLAB.

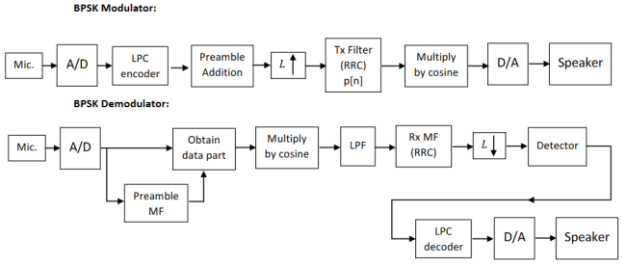

Figure 9: Block diagram of BPSK Modulation

After audio signal was quantized, Linear Predictive Order (LPC) Encoder produced about 1200 bits for each second of the audio signal.

```
clear, clc;
% Initializations
Fs=8*1e3; % Sampling Frequency : 8kHz

% BPSK Modulator
% LPC Encoder Output   X-->y_LPC_tx
LPC_out = transpose(LPC_tx_s(X));
```

LPC encoding/decoding is very important process for real time application because it compresses the audio signal and reduces the bit rate requirements for the communication. Although the quality of the audio signal will be decreased at the received side, overall process can be completed shorter time in the presence of LPC. The output of the LPC encoder can be seen in the Figure 10.

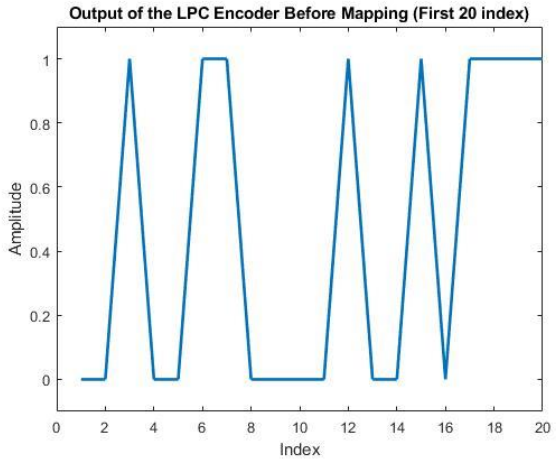

Figure 10: Output Of the LPC Encoder

However, the output of the LPC encoder only consists of zeros and ones. For the upsampling process, these zeros must be mapped into -1's.

```
% LPC_out ( 0--> -1 )
LPC_out_sw = int16(LPC_out);
LPC_out_sw(LPC_out_sw == 0) = -1;
```

The result of the mapping process can be seen in the Figure 11.

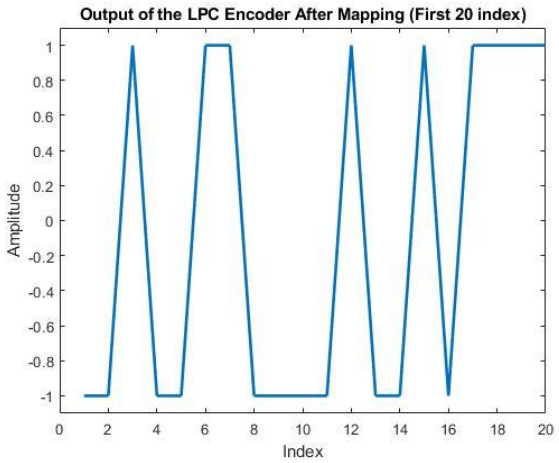

Figure 11: Output of the LPC Encoder after mapping operation

For real time application, determining the beginning of the transmitted signal at the received side is crucial. Therefore, to indicate the beginning, a known bit sequence should be added beginning of the signal. In this case 8-bits preamble data, which is [1 -1 1 -1 1 -1 1 -1], was added to the original signal.

```
% Preamble addition
seq = [1 -1 1 -1 1 -1 1 -1];
pre_out_BPSK_m = [seq LPC_out_sw];
```

The resulting signal can be seen in the Figure 12.

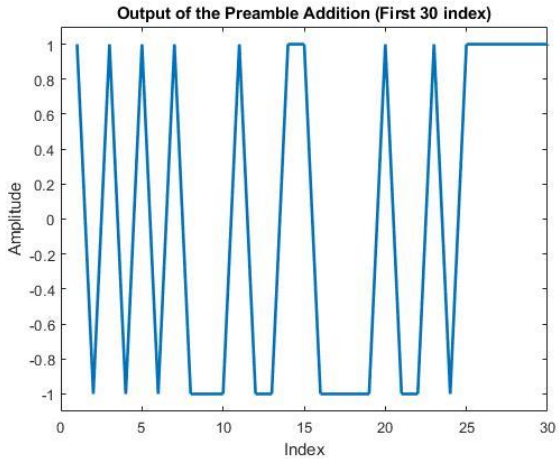

Figure 12: The signal after preamble addition

To increase sampling rate, zero-valued samples were inserted between original samples of the signal. In this case, sampling rate is L=20.

```
% Up Sampling
L=20;
up_out_BPSK_m = upsample(pre_out_BPSK_m, L);
```

The resulting signal can be seen in the Figure 13.

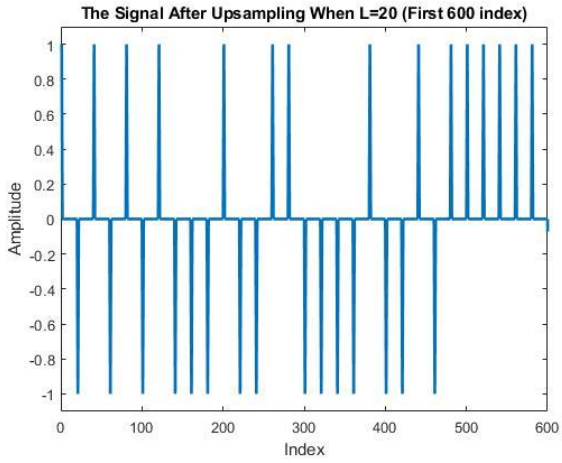

Figure 13: Upsampled signal

In order to transmit the upsampled signal with less distortion, Root-raised Cosine (RRC) Filter must be used, because rectangular pulse has sharp edges, therefore it causes a distortion. By using "rcosdesign" function, RRC filter was obtained.

```
% RRC filter  up_out_BPSK_m --> RRC_t_out
BETA=0.5; SPAN=10;
TxFilter = rcosdesign(BETA, SPAN, L);
RRC_t_out = conv(TxFilter, up_out_BPSK_m);
```

This filter can be seen in the Figure 14. SPAN parameter specifies the length of the pulse and BETA parameter specifies the roll-off factor.

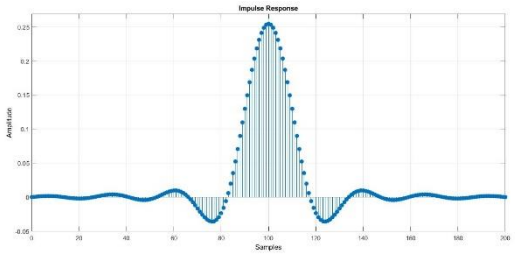

Figure 14: Impulse response of raised-cosine filter

After convolution of the upsampled signal with RRC filter, a new signal was obtained which can be seen in the Figure 15.

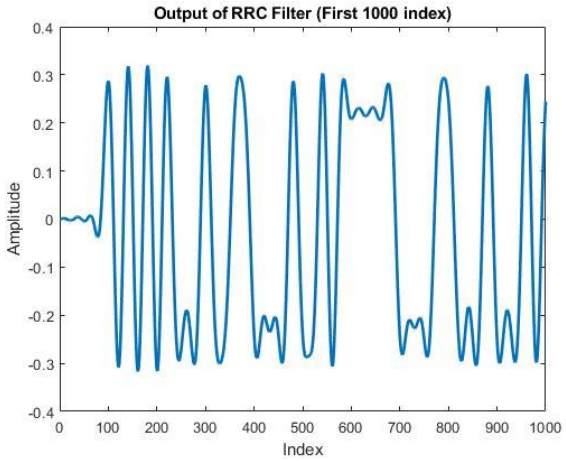

Figure 15: Audio signal at the output of RRC filter

Because of the frequency of audio signal at the output of RRC filter is very low, it cannot be transmitted directly via speaker. Therefore, this signal must be upconverted to a higher frequency in order to avoid distortions.

```
% Carrier Multiplication
carr_freq = 3*1e3; % Frequency
carr_t =
1/(L*1200):1/(L*1200):(length(RRC_t_out)*(1/(L*1
200))); % time
carr_amp = 1; % Amplitude
carr = carr_amp*cos(2*pi*carr_freq*carr_t);

out_BPSK_m = carr .* RRC_t_out;
```

Generated carrier wave can be seen in the Figure 16.

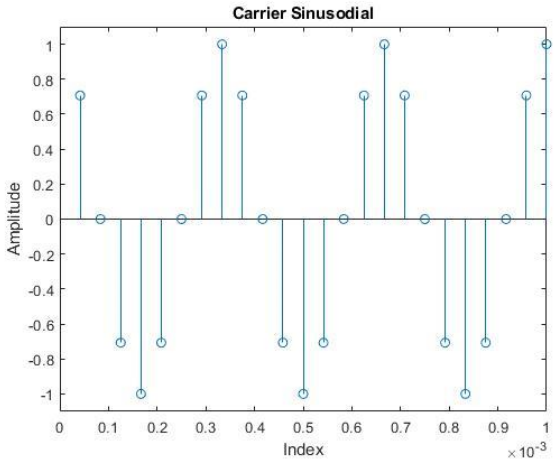

Figure 16: Representation of the carrier sinusoidal

Carrier frequency is determined according to the specifications of the laptop's microphone. Also, according to the sampling rates, time increment was adjusted to get a proper result. The modulated signal can be seen in the Figure 17.

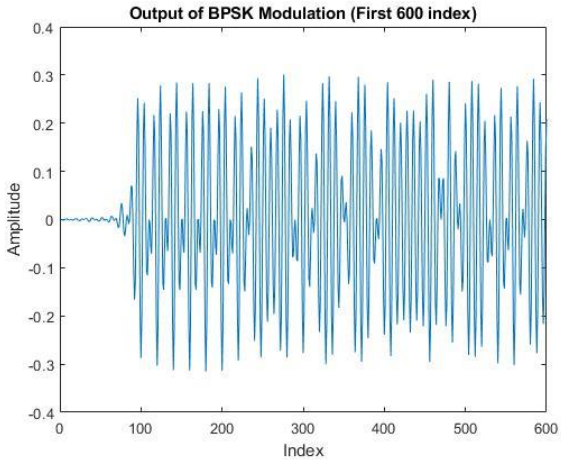

Figure 17: Representation of modulated signal

For the second part of the project, the signal is demodulated after modulation in the same MATLAB file.

```
% BPSK Demodulator
in_BPSK_d = out_BPSK_m;
```

In the demodulation part transmitted signal has to be detected. Therefore, Usage of the Preamble Matching Filter was essential for this purpose. Output of the preamble filter can be seen in the Figure 18.

```
%Preamble
sn =fliplr(out_BPSK_m(2:360));
buff_BPSK_d = conv([in_BPSK_d(1) sn],
in_BPSK_d);
```

Then, index of the maximum value was obtained for the removing the unnecessary part of the signal while detector part.

```
% Obtaining data part
[val, idx] = max(buff_BPSK_d);
```

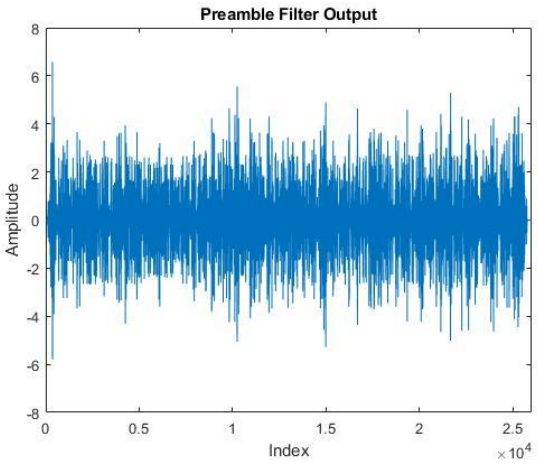

Figure 18: Output of the preamble filter

The peak of the preamble matching filter output was observed at the 361st index and it can be seen in the Figure 19 below.

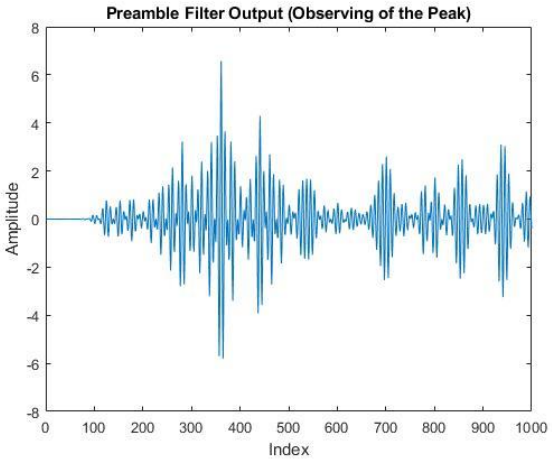

Figure 19: Peak of the Preamble MF Filter output

In order to downconvert the received signal, the data must be multiplied with cosine again. The resulting signal was in $m(t)+ m(t)cos(4\pi f_c\ t)$ form which can be seen in the Figure 20.

```
% Multiplication with carrier(Cosine)
pre_out_BPSK_d = out_BPSK_m;
mult_out_BPSK_d = carr.*pre_out_BPSK_d;
```

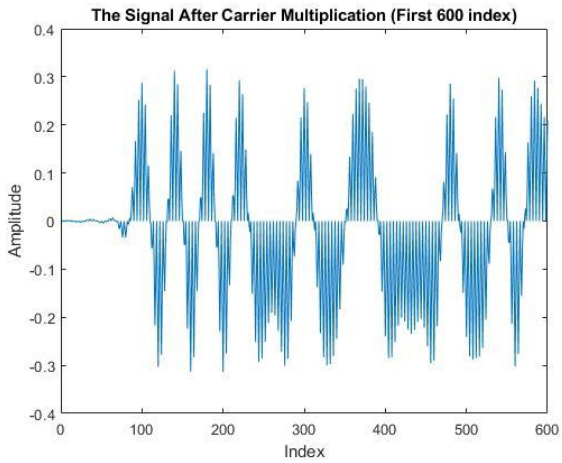

Figure 20: The signal after carrier multiplication

To remove the component around $2f_c$, a low pass filter was used.

```
% Low pass filter
[B,A] = butter(6, 0.6);
flt_out_BPSK_d = filter(B, A, mult_out_BPSK_d);
```

Frequency response of this filter can be seen in the Figure 21. Also, the output of this low pass filter can be seen in the Figure 22.

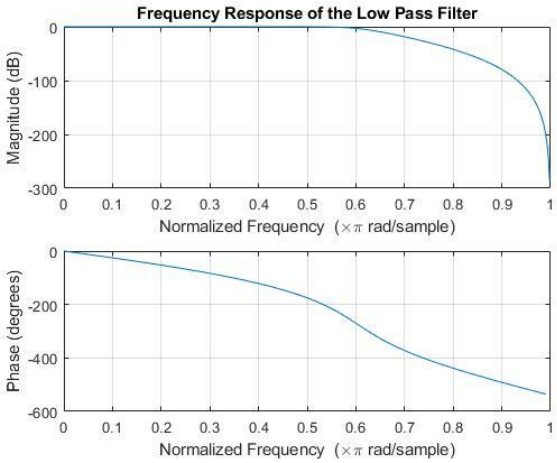

Figure 21: Frequency response of the low pass filter

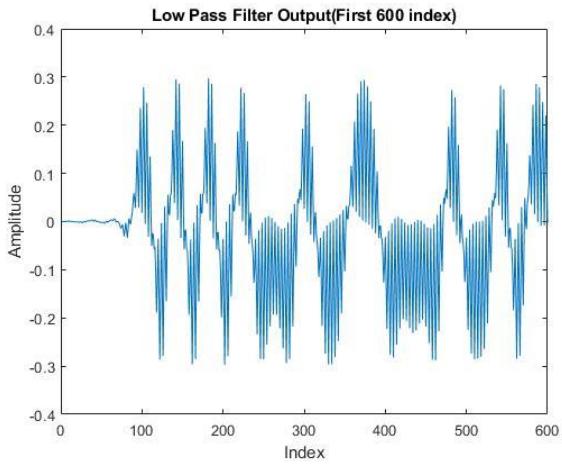

Figure 22: Output of the low pass filter

This signal must be filtered again with matched filter. As stated in the explanations, matched filter can be obtained with “fliplr” command

```
%Rx Matched Filter (RRC)
RxFilter = fliplr(TxFilter);
RRC_r_out = conv(RxFilter, flt_out_BPSK_d);
```

Output of this matched filter can be seen in the Figure 23 below.

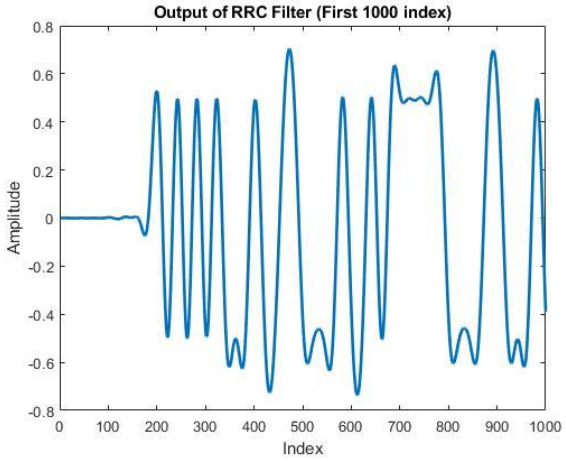

Figure 23: Output of the matched filter

Then the signal was downsampled with L=20. The resulting signal can be seen in the Figure 24.

```
% Downsampling
down_out_BPSK_d = downsample(RRC_r_out, L);
```

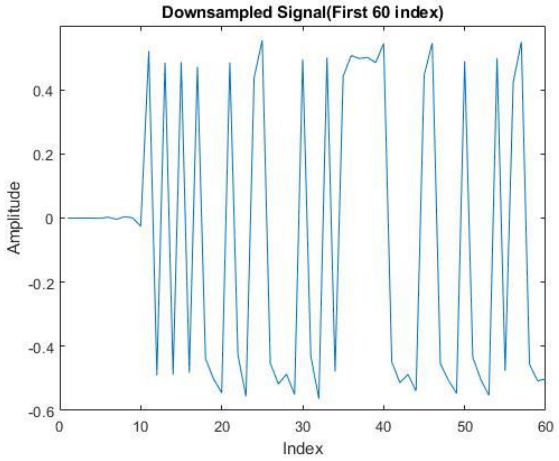

Figure 24: Representation of the downsampled signal

After downsampling, samples were compared with zero and if they were greater than zero, they were accepted as 1 and if they are less than zero, they were accepted as 0.

```
% Detector
det_out = down_out_BPSK_d(round((idx)/L) :
length(down_out_BPSK_d)-SPAN);
det_out(det_out < 0) = 0;
det_out(det_out > 0) = 1;
y_BPSK_d = det_out(2:length(det_out));
```

The output of the detector can be seen in the Figure 25.

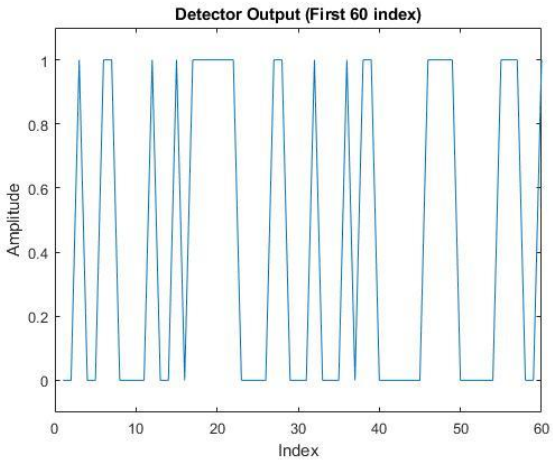

Figure 25: Output of the detector

By using LPC Decoder and “audioplayer” function of MATLAB, audio signal was obtained and listened. The resulting audio signal can be seen in the Figure 26.

```
%LPC Decoder - Audioplay
synth_speech_agr=LPC_rx_s(y_BPSK_d.');

p = audioplayer(synth_speech_agr, Fs);
play(p);
```

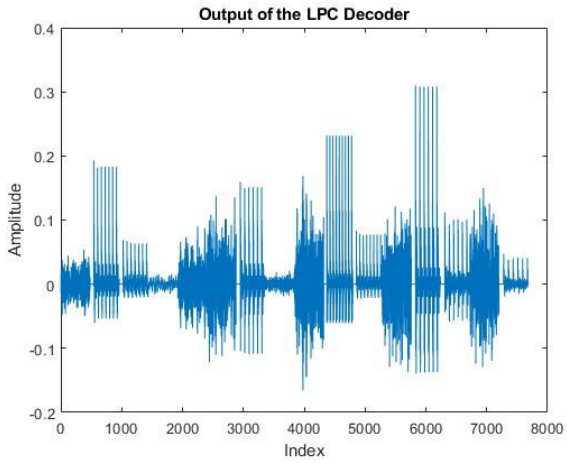

Figure 26: Resulting audio signal

C. Quadrature Phase-Shift Keying (QPSK)

The Quadrature Phase Shift Keying, QPSK, is a variation of BPSK, and it is also a Double Side Band Suppressed Carrier DSBSC modulation scheme. In QPSK modulation, the digital bits are converted into bit pairs instead of converting them into digital stream, which enables us to decrease the data bit rate into half. Overall operation for QPSK modulation is summarized in the block diagram which is provided in Figure 27. In sections to follow, the developed code for QPSK implementation will be demonstrated.

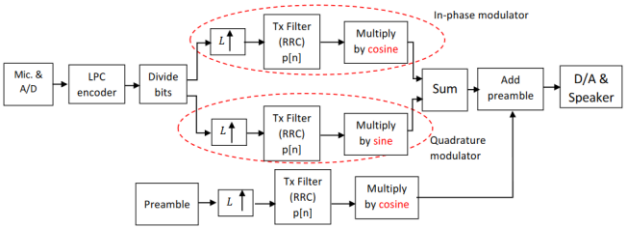

Figure 27: Block diagram of QPSK Modulation

```
% Initializations

% Sampling Frequency : 8kHz
Fs=8*1e3;

% Up/Downsample Rate
L = 20;

%RRC filter (Tx)
BETA=0.5; SPAN=10;
TxFilter = rcosdesign(BETA, SPAN, L);

% RRC filter (Rx)
RxFilter = fliplr(TxFilter);

% Carier Sinusodial
t_step = 1/ (1200 * L);
carr_amp = 1;                % Gain Amplification

% Low pass filter (Butterworth)
[b,a] = butter(6, 0.6);

% Input                                  .wav --> X
[X, Fs]=audioread('430_dnm.wav');
% Loading sound file

% LPC Output                           X-->y_LPC_tx
LPC_out = transpose(LPC_tx_s(X));
```

Here, parameters that will be used throughout the code are initialized. Notice that the low-pass filter implemented here is the same Butterworth Filter as the one used in the BPSK Modulation. Please refer to *Section B* for the specifications and the characteristics of this filter.

```
% QPSK Modulation

% Receive Data
LPC_out --> x_QPSK_m
x_QPSK_m = int16(LPC_out);
figure, stem(x_QPSK_m, 'LineWidth',2);
xlim([1 50]), ylim([-0.1 1.1]), title('Received
Data');
```

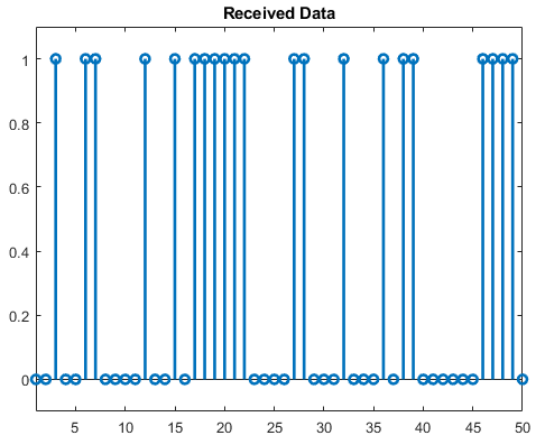

Figure 28: Represenatite characterşstics of received data

```
% 0 to -1 Conversion
x_QPSK_m(x_QPSK_m == 0) = -1;
figure, stem(x_QPSK_m, 'LineWidth',2);
xlim([1 50]), ylim([-1.1 1.1]), title('0 to -1
Converted Received Data');
```

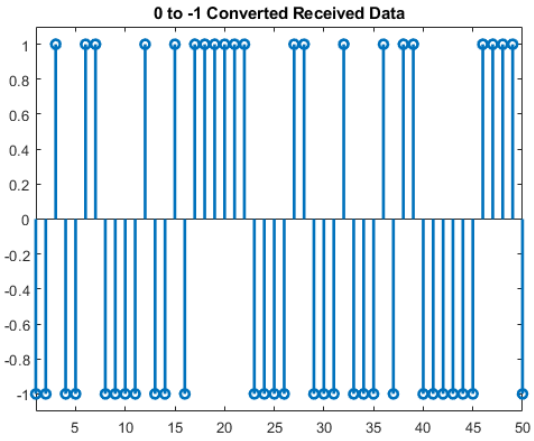

Figure 29: 0 to -1 conversion

Input signal to the QPSK Modulation unit is taken as the direct output of the LPC_tx rather than including a transmission medium for the sake of simplicity. The representative characteristics of the input is demonstrated in Figure 28 for a small interval. Notice that, the signal is composed of 0/1 which will be the case when the microphone output is digitized at the A/D converter. In order to proceed with the QPSK Modulation, 0' s shall be converted into -1' s, which is demonstrated in Figure 29.

```
% Bit division                          x_QPSK_m -->
div1_QPSK_m , div2_QPSK_m
div1_QPSK_m = downsample(x_QPSK_m, 2);
% In Phase component
```

```
div2_QPSK_m = downsample(x_QPSK_m(2:end), 2);
% Quadrature Phase component
figure, stem(div1_QPSK_m, 'LineWidth',2);
xlim([1 25]), ylim([-1.1 1.1]);
title('In Phase component of the Received
Data');
```

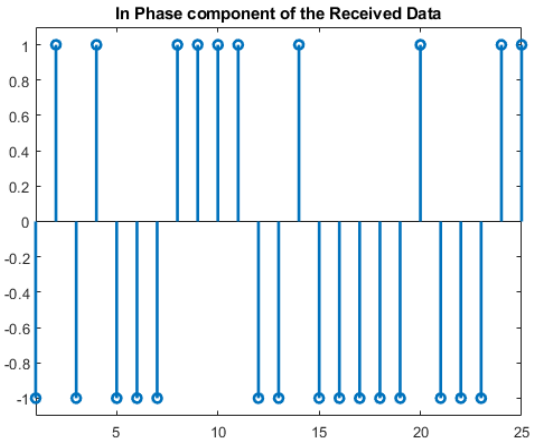

Figure 30: In-Phase component

```
figure, stem(div2_QPSK_m, 'LineWidth',2);
xlim([1 25]), ylim([-1.1 1.1]);
title('Quadrature Phase component of the
Received Data');
```

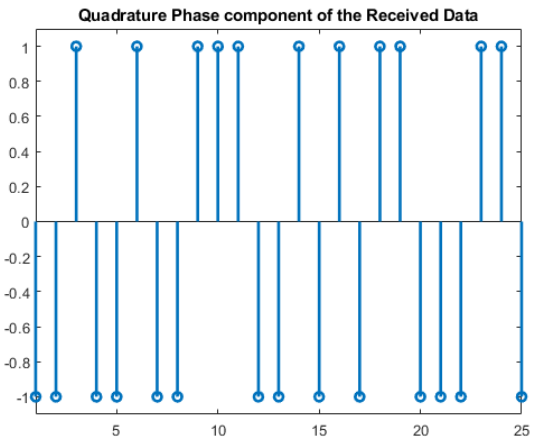

Figure 31: Quadrature component

QPSK Modulation is based on two orthogonal basis functions. Cosine and Sine modulation are suitable orthogonal carriers for QPSK modulation. In the QPSK transmitter, the ingredients of the input signal are to be dived into two so that the divided bits can be transmitted by the cosine and sine carriers. Referring to Figure 28, the even and odd indices of the provided signal divided into two whose are demonstrated in Figure 30 and Figure 31 respectively.

```
% In Phase Modulator
div1_QPSK_m --> mult_div1_QPSK
```

```
up_div1 = upsample(div1_QPSK_m, L);
figure, stem(up_div1, 'LineWidth',2);
xlim([1 25]), ylim([-1.1 1.1]), title('Upsampled
In-Phase component');
```

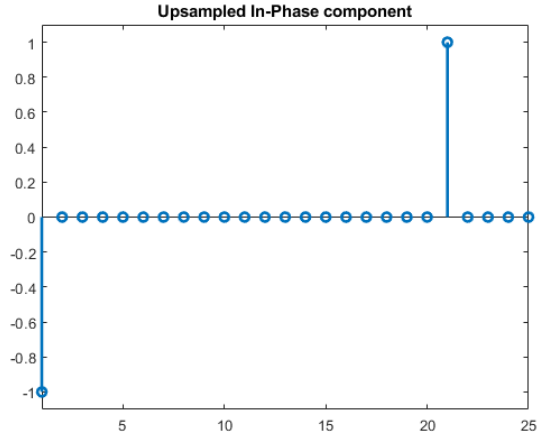

Figure 32: Representative upsampling of In-Phase component

```
%RRC filter
RRC_out_div1_QPSK = conv(up_div1, TxFilter);
figure, plot(RRC_out_div1_QPSK, '.',
'LineWidth',1.25);
xlim([50 750]), ylim([-1.1 1.1]),
title('Filtered In-Phase component');
```

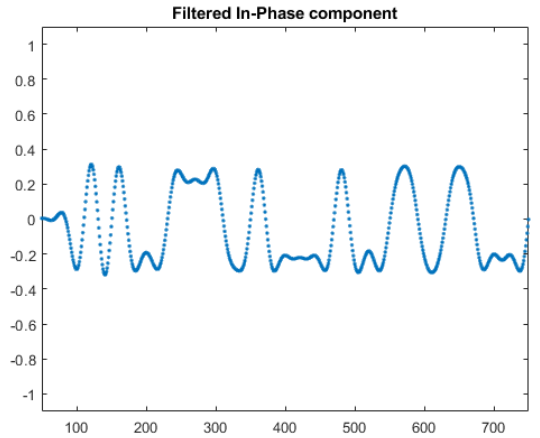

Figure 33: Representation of filtered in-phase component

```
% Cosine Multiplication
cos_freq = 6*1e3;
% Frequency
cos_t = t_step : t_step :
length(RRC_out_div1_QPSK)*t_step;
% time
cos_div = cos(2*pi*cos_freq*cos_t);
mult_div1_QPSK = cos_div .* RRC_out_div1_QPSK;
figure, plot(mult_div1_QPSK, 'LineWidth',1.5);
xlim([50 750]), ylim([-1.1 1.1]);
title('Cosine Multiplied In-Phase component');
```

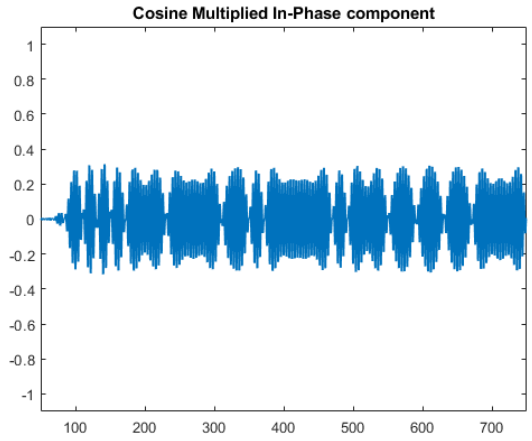

Figure 34: Output signal of Cosine multiplication

Having divided the input signal into two sequences, one of the resultants is to be modulated by the first orthogonal carrier. Here, the modulation of the sequence that is composed of even indices of the input signal with a Cosine is studied. The signal provided in Figure 30 is upsampled with L, 20, which can be seen in Figure 32. The basic idea here is to zero pad between two consecutive indices with (L-1). Then, the resultant signal is to be filtered out with an RRC filter whose output characteristics is illustrated in Figure 33. The signal at the output end of the RRC filter is now ready to be multiplied with a predefined Cosine. Figure 34 demonstrates the overall output of the In-Phase Modulation.

```
% Quadrature Phase Modulator
up_div2 = upsample(div2_QPSK_m, L);

%RRC filter
RRC_out_div2_QPSK = conv(up_div2, TxFilter);

% Sine Multiplication
sin_freq = 6*1e3;
% Frequency
sin_t = 1/(20*1200):1/(20*1200)
:length(RRC_out_div2_QPSK)*(1/(20*1200)));
% time
sin_div = sin(2*pi*cos_freq*sin_t);
mult_div2_QPSK = sin_div .* RRC_out_div2_QPSK;
figure, plot(mult_div1_QPSK, 'LineWidth',1.5);
xlim([50 750]), ylim([-1.1 1.1]);
title('Sine Multiplied Quadrature-Phase
component');
```

A similar approach as in the In-Phase modulation is applied while modulating the quadrature part of the input signal. The sequence is upsampled with L, filtered out with an RRC filter and multiplied with a sinusoidal. It should be noted however that, the multiplying sinusoidal is a Sine rather than a Cosine.

```
% Sumation of divided bits
sum_out_QPSK = mult_div1_QPSK + mult_div2_QPSK;
figure, plot(sum_out_QPSK, 'LineWidth',1.5);
xlim([50 750]), ylim([-1.1 1.1]), title('Summed
Bits');
```

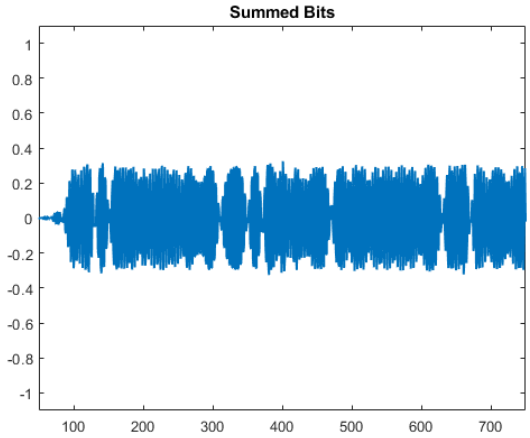

Figure 35: Output signal of summation block

In phase and Quadrature Phase modulated sequences are to be summed up before proceeding to adding a preamble to the signal that is to be transmitted. Output of the summation block is provided in Figure 35.

```
% Preamble INI
seq_QPSK = [1 -1 1 -1 1 -1 1 -1];
up_seq_QPSK = upsample(seq_QPSK, L);
% Upsampling
RRC_out_seq = conv(up_seq_QPSK, TxFilter);
% RRC filtering
cos_time = t_step : t_step :
length(RRC_out_seq)*t_step;              % time
cos_seq = cos(2*pi*cos_freq*cos_time);
mult_out_seq = cos_seq .* RRC_out_seq;
% Cosine Multiplication
figure, plot(mult_out_seq, 'LineWidth',1.5);
title('Preamble Characteristics');
```

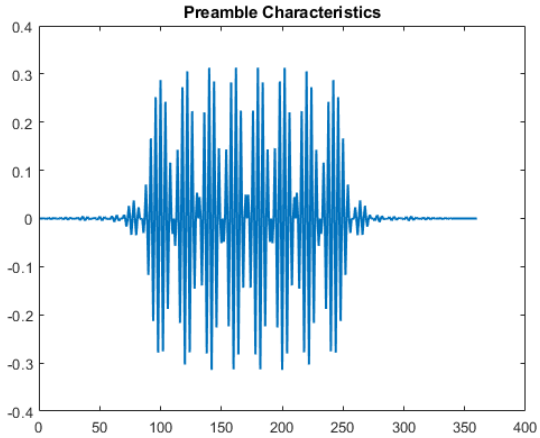

Figure 36: Predefined preamble sequence characteristics

Having modulated the signals with two orthogonal basis function, a preamble sequence should be included in order to alert the receiver part of the communication couple that a logical data transfer is about occur. The selected preamble sequence is modulated in a similar fashion with

the In-phase and Quadrature Phase components. The reason why we are modulating the preamble sequence separately from the transmission data is that, we are modulating the input sequence with two different multipliers. Remember that, unlike QPSK modulation, the preamble sequence was modulated alongside with the transmission data when the modulation technique is BPSK. In QPSK modulation, the preamble sequence should be modulated with one of the orthogonal multipliers. A cosine modulation is selected in our case.
The sequence characteristics of the preamble is stored in *mult_out_seq*, which is illustrated in Figure 36, in order to have a proper filter in Demodulation part which will detect the presence of the preamble.

```
% Preamble Addition              mult_out_seq,
sum_out_QPSK --> out_QPSK_m
out_QPSK_m = [mult_out_seq sum_out_QPSK];
figure, plot(out_QPSK_m), title('Output of QPSK
Modulation');
```

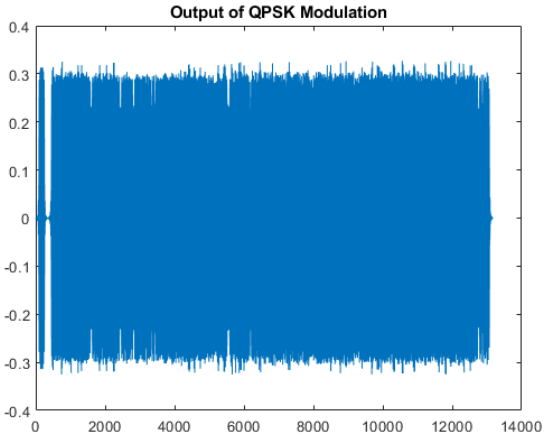

Figure 37: Signal at the output-end of Modulator

Referring to Figure 37, modulated preamble sequence and input sequences are combined before transmitting the signal.

```
% QPSK Demodulation
Received Data --> in_QPSK_d
in_QPSK_d = out_QPSK_m;
```

For the sake of simplicity, the Demodulation input is selected as the output of the Modulation, rather than having a transmission medium in between.

```
% Preamble
sn_QPSK =fliplr(mult_out_seq(2:360));
sn_QPSK = [sn_QPSK(1) sn_QPSK];

buff_QPSK_d = conv(sn_QPSK, in_QPSK_d);
figure, plot(buff_QPSK_d), title('Preamble
Detection');
```

```
% Obtaining data part
[VAL, IDX] = max(buff_QPSK_d);
% The maximum value is stored in IDX

in_QPSK_d = in_QPSK_d(IDX:end);
% The input signal is purified from preamble
```

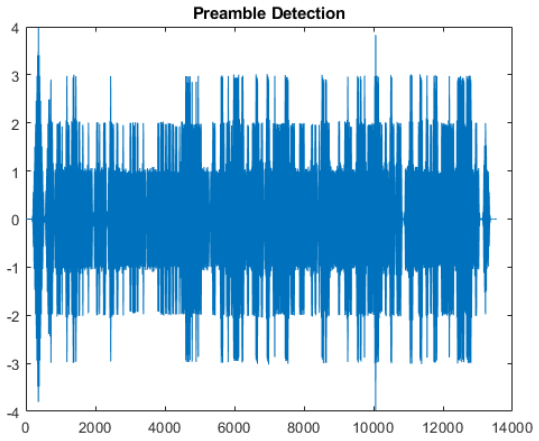

Figure 38: The output response of defined Preamble detector filter

Before the demodulation process starts, a filter that detects the presence of the preamble shall be installed. A suitable choice for this task is to select a filter with an impulse response which is nothing but circularly reversed preamble sequence. The reason behind such a selection is that, the correlation operation will give rise to a peak when the predefined preamble passes from this filter. In this block, the IDX value, which corresponds to the index of the preamble filter output peak. Notice that, as illustrated in Figure 38, the filter was able to detect the preamble presence; however, for a more reliable peak detection, it is possible to update the preamble sequence. After detecting the index of the peak valued component, the received signal should disregard the preamble and take the resultant sequence as logical communication data.

```
% In Phase Part
in_QPSK_d --> det_I_out

% Multiplication with carrier(Cosine)
cos_I_t = t_step : t_step
:length(in_QPSK_d)*t_step;
cos_I = cos(2*pi*cos_freq*cos_I_t);
mult_I_out_QPSK_d = cos_I .* in_QPSK_d;

% Low pass filter
flt_I_out_QPSK_d = filter(b, a,
mult_I_out_QPSK_d);

%Rx Matched Filter (RRC)
RRC_r_I_out_QPSK_d = conv(RxFilter,
flt_I_out_QPSK_d);

% Downsampling
down_out_I_QPSK_d =
```

```
downsample(RRC_r_I_out_QPSK_d, L);

% Detector
det_I_out(down_out_I_QPSK_d < 0) = 0;
det_I_out(down_out_I_QPSK_d > 0) = 1;
```

In this part, the orthogonally transmitted signals should be detected and recombined into one. This block is reversely analogous to the modulation block. The In-Phase part of the resultant signal after disregarding preamble, is demodulated by multiplying the sequence with the exact cosine multiplier in the modulation part. This results in the logical communication data alongside with a high frequency component. The high frequency component can be disregarded by applying this sequence to a low pass filter. The installed low-pass filter here is, again, the Butterworth filter with a cut-off frequency of 0.6 rad. Then the sequence is to pass through another RRC filter whose impulse response is time reversed version of the one that is used in the modulation part. Refer to *Initialization* for further details. The resulting sequence is then digitized into 0/1 in the detector unit.

```
% Quadrature Phase Part
in_QPSK_d → det_Q_out

% Multiplication with carrier(Cosine)
sin_Q_t = 1/(20*1200):1/(20*1200)
:length(in_QPSK_d)*(1/(20*1200)));
% Amplitude
sin_Q = sin(2*pi*cos_freq*sin_Q_t);
mult_Q_out_QPSK_d = sin_Q .* in_QPSK_d;

% Low pass filter
flt_Q_out_QPSK_d = filter(b, a,
mult_Q_out_QPSK_d);

%Rx Matched Filter (RRC)
RRC_r_Q_out_QPSK_d = conv(RxFilter,
flt_Q_out_QPSK_d);

% Downsampling
down_out_Q_QPSK_d =
downsample(RRC_r_Q_out_QPSK_d, L);

% Detector
det_Q_out(down_out_Q_QPSK_d < 0) = 0;
det_Q_out(down_out_Q_QPSK_d > 0) = 1;
```

A similar process is applied to the Quadrature-Phase component of the detected signal in this block.

```
% Combination of Bits
det_I_out, det_Q_out → y_QPSK_d
up_I_QPSK_d = upsample(det_I_out, 2);
% In Phase component
up_Q_QPSK_d = upsample(det_Q_out, 2);
% Quadrature Phase component
y_QPSK_d = [up_I_QPSK_d 0] + [0 up_Q_QPSK_d];
% Bits conversion
y_QPSK_d =
transpose(y_QPSK_d(L+1:length(y_QPSK_d)-(L+1)));
% Filtering logical components
```

The detected In-Phase and Quadrature-Phase sequences should be recombined into one. Keeping in mind that, the In-Phase component of the data was established by downsampling with two, whereas the Quadrature component of the data was established by downsampling with two after applying a 1 sample delay. In the recombination block, the operation orders are reversed in order to get the exact signal that is transmitted from the Modulator.

```
% LPC_rx
synth_speech_agr=LPC_rx_s(y_QPSK_d);
p = audioplayer(synth_speech_agr, Fs);
figure, plot(synth_speech_agr), title('Output of
LPCrx');
play(p);
```

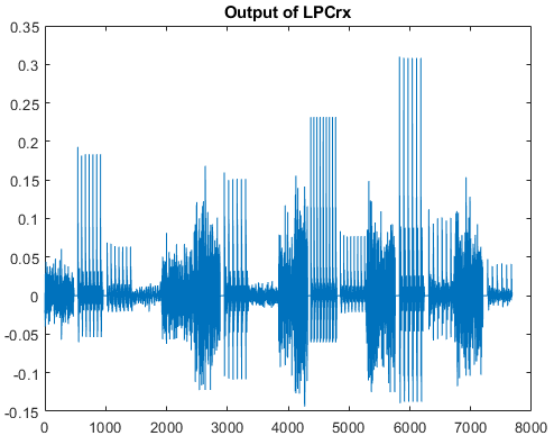

Figure 39: Output response of LPC_rx

The demodulated signal that will be transmitted to the speaker of the secondary computer is demonstrated in Figure 39 for convenience.

III. CONCLUSION

In the second stage of the project, it is desired to create a program using MATLAB to form a system which transfers acoustic waves from one terminal to another using different modulation topologies; namely, Frequency Shift Keying (FSK), Binary Phase Shift Keying (BPSK) and Quadrature Shift Keying. The milestones are upsampling/downsampling and performing operations on sensed sound data, filtering and modulation of the signal, transmitting the resulting signal and replay it from the destination platform. As the project is highly integrated with the digital signal processing, it is beneficial for the participating students in a way that it is possible to test their theoretical backgrounds and have a hands-on experience.

This report is devoted as to explain the developed for the transmission of acoustic waves. Although a perfect real time execution is not attempted here, the aim for this stage is to prepare an underlying platform that we will step on while we will try to converge a real time operation as much as possible for human hearing. The notices and conclusions of this report will illuminate the way of the actual implementation of real time process.

IV. REFERENCES

[1] Nordström, J. Real Time Digital Signal Processing, Uppsala Universitet, 2017

[2] Oppenheim, Alan V. and Schafer, Ronald W. Discrete Time Signal Processing, 3rd Edition, Prentice Hall Signal Processing Series, 2010

[3] Oppenheim, Alan V. and Willsky, Alan S. Signals and Systems, 2nd Edition, Prentice Hall Signal Processing Series, 2014

[4] Proakis, John G., and Masoud Salehi. Fundamentals of communication systems, 2nd Edition, Pearson Education, 2014

[5] Raza, Md. S. FSK modulation and demodulation, MATLAB https://www.mathworks.com/matlabcentral/fileexchange/44821-matlab-code-for-fsk-modulation-and-demodulation