به نام خدا

دانشگاه صنعتی شریف
دانشکده مهندسی برق

گزارش پروژه درس

سیستم‌های مخابراتی

استاد مربوطه: دکتر هادی

دانشجو: زهرا مجتهدین ۹۹۱۰۲۱۶۷
طاهر ابدی ۹۹۱۰۱۰۳۲

فهرست مطالب

فهرست مطالب ۳

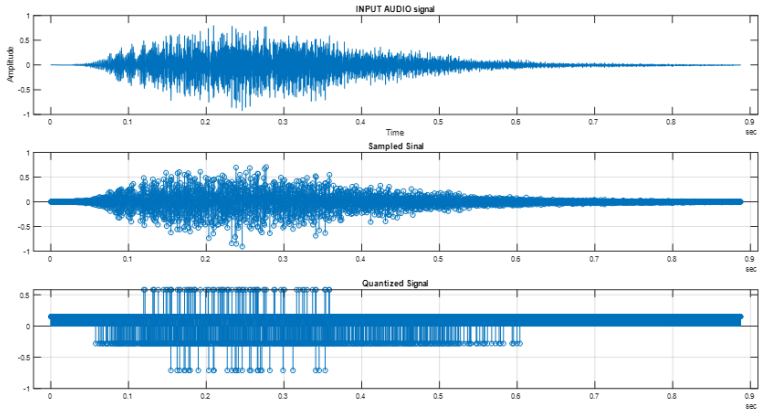
هدف از این پروژه شبیه‌سازی یک سیستم مخابراتی دیجیتال باند پایه است. با توجه به مطالب بیان‌شده در فایل مربوط به بررسی مختصر این شبیه‌سازی می‌پردازیم.

۰-۱ بلوک‌های سیستم دیجیتال باند پایه

در این بخش به معرفی و نحوه پیاده‌سازی هرکدام از بلوک‌های سیستم مورد نظر پرداخته خواهد شد.

۰-۱-۱ ADC

عمل تبدیل سیگنال آنالوگ به سیگنال دیجیتال توسط تابع pcm function انجام می‌شود. این تابع سیگنال صدا را که با استفاده از تابع audioread خوانده شده به همراه فرکانس نمونه‌برداری و تعداد سطوح کوانتیزایسون دریافت می‌کند. بعد از دریافت سیگنال ورودی با استفاده از تعداد سطوح کوانتیزاسیون سطوح خروجی را تعیین و سپس با کمک توابع متلب سیگنال را کوانتیزه می‌کنیم (شکل ۱). در این مرحله سیگنال پیوسته ما به یک سیگنال نمونه‌برداری شده با سطوح مشخص تبدیل شد. به عنوان گام انتهایی برای این بلوک به سطوح کوانتیزه مطابق با یک pcm اعداد باینری نسبت می‌دهیم. در حالی که شکل‌های ۱ و ۲ به ترتیب سیگنال اصلی، نمونه‌برداری شده و کوانتیزه شده را برای $f_s = 4000$ و $\nu = 4$ به ازای کل و بخشی از صوت نشان می‌دهد، شکل ۳ نشان‌دهنده تعدادی از بیت‌های خروجی است.



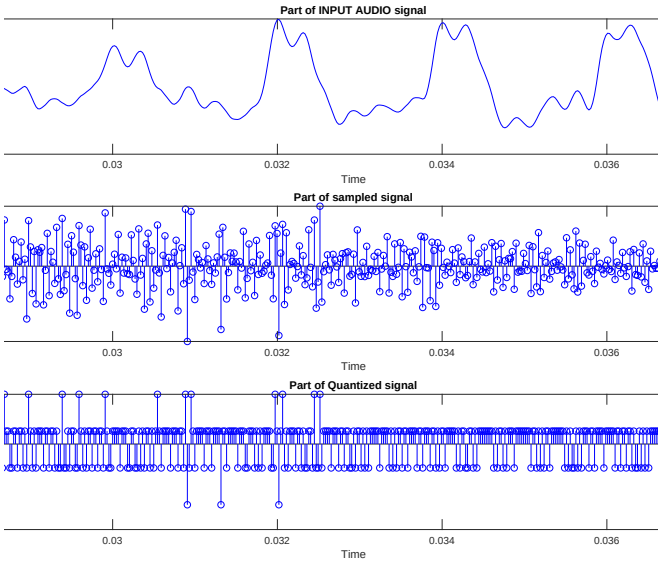
شکل ۱: نمودار سیگنال اصلی، نمونه‌برداری شده و کوانتیزه شده به ازای $f_s = 4000$ و $\nu = 4$

۰-۱-۲ linecoder

عمل linecoder توسط تابع line code funcyion انجام می‌شود. در این تابع متناسب با خواسته سوال ابتدا یک سیگنال Nyquist با پارامترهای r و β ساخته و سپس به بیت‌های خروجی تابع قبل این شکل موج تخصیص داده می‌شود. به عبارت دیگر بعد از ساخت سیگنال Nyquist با هر بیت صفر و یک به ترتیب یک سیگنال Nyquist با دامنه $A-$ و A به فاصله D ارسال خواهد شد. شکل ۴ نمودار تعدادی از بیت‌های ارسالی و شکل موج خروجی linecoder را نشان می‌دهد.

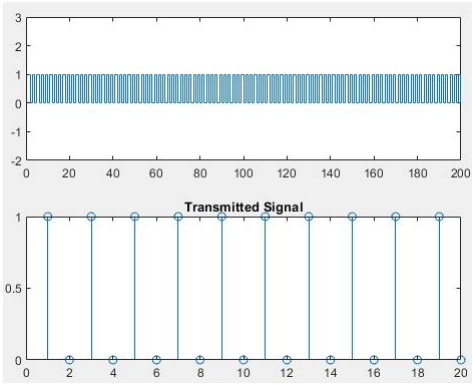
۰-۱-۳ channel

سیگنال خروجی مرحله قبل حال وارد کانال خواهد شد. کانال ما اینجا یک کانال AWGN باند محدود خواهد بود. برای پیاده سازی این کانال یک فیلتر پایین‌گذر با پهنای باند B تولید کرده و سپس سیگنال ارسالی نویزی

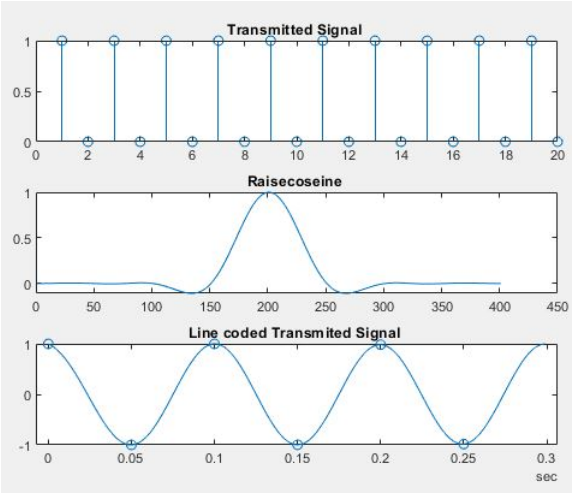


شکل ۲: نمودار بخشی از سیگنال اصلی، نمونه‌برداری شده و کوانتیزه شده به ازای $f_s = 4000$ و $\nu = 4$

یا نویز سفید گوسی با $\sigma^2 = N_0 B$ را از آن عبور می‌دهیم. لازم به ذکر است که در فایل‌های متلب این سیستم برای پیاده‌سازی فیلتر پایین‌گذر و نویز از دستور sinc سیگنال رندم استفاده شده است. شکل ۵ خروجی کانال را به ازای بیت‌های ارسالی برای کانالی با پهنای باند$B = 12 \leq r = 20 \leq 2B = 24$ و $S/N = 1000$ نشان می‌دهد.



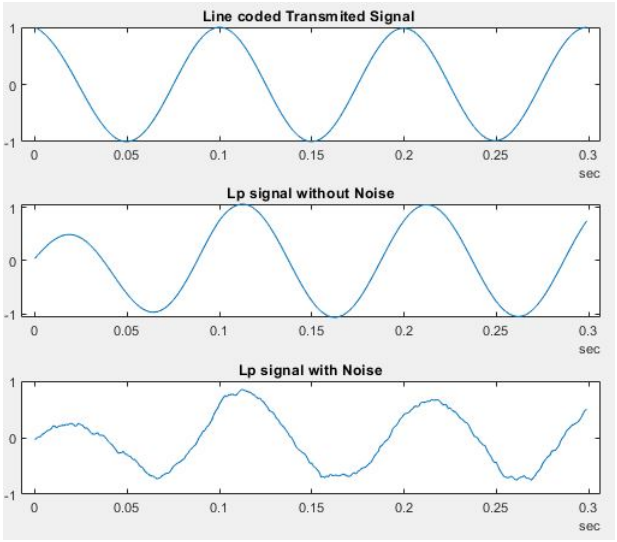
شکل ۳: نمودار تعدادی از بیت‌های خروجی از بلوک ADC



شکل ۴: نمودار تعدادی از بیت‌های ارسالی و شکل موج خروجی linecoder به ازای $r = 20$ و $\beta = 6$

۴-۱-۰ Line decoder

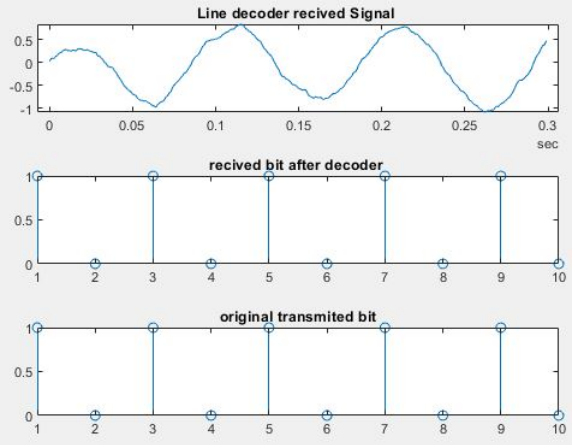
در این بلوک با فرض وجود یک مدار synchronization دقیق، به بازیابی بیت‌های ارسالی می‌پردازیم. به عبارت دیگر با فرض synchronization بعد از تشخیص اولین مکان برای نمونه‌برداری به فاصله D از سیگنال نمونه‌برداری کرده و بیت ارسالی را با مقایسه دامنه سیگنال با صفر بازیابی می‌کنیم. شکل ۶ تعدادی از بیت‌های بازیابی شده توسطLine decoder را نشان می‌دهد.



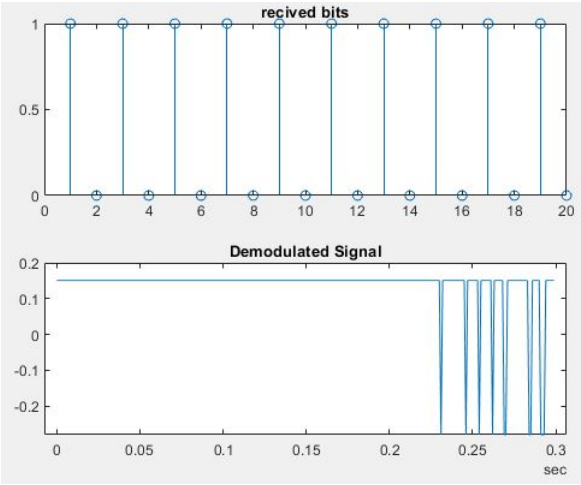
شکل ۵: نمودار خروجی کانال در حضور و عدم حضور نویز به ازای سیگنال ورودی مشخص شده با پارامترهای پهنای باند$B = 12 \leq r = 20 \leq 2B = 24$ و $S/N = 1000$

DAC ۵-۱-۰

این بلوک کاری عکس بلوک اول را انجام می‌دهد، به این صورت که بیت‌های خروجی Line decoder را دریافت کرده و آن‌ها را به اعداد دیجیتال تبدیل می‌کند. اعداد حاصل‌شده اعدادی با دامنه مشخص با تعداد نمونه کم خواهد بود(عکس عمل کوانتیزاسیون) که گرچه دامنه آن از حالت باینری به دیجیتال تبدیل شده ولی بدون عمل upsample و interpolation نمی‌توان آن را سیگنال پیوسته فرض کرد. در نتیجه به عنوان آخرین گام عمل upsample و interpolation را انجام می‌دهیم.



شکل ۶: نمودار بخشی از سیگنال ورودی و خروجی Line decoder و بیت‌های ارسال



شکل ۷: نموداربخشی از سیگنال دیجیتال و آنالوگ قبل از upsample و interpolation برای $\mu = 4$.

۲-۰ تست بلوک‌ها و نحوه عملکرد آن‌ها

اگرچه نحوه عملکرد مدار در بخش قبل به صورت مرحله‌ای انجام شده و خروجی آن نمایش داده شد. در ادامه می‌خواهیم خروجی مدار را به ازای پارامترهای مختلف چک کرده و تفاوت آن را بررسی کنیم.

در اجرای کد باید به این نکته توجه داشت که اگر ما با فرکانس‌های در حدود ۴۰۰۰ که کمتر از نرخ

نایکوییست از نمونه‌برداری کنیم صدایی که از سیگنال به گوش می‌رسد کمی با حالت اصلی متفاوت‌تر بوده همچنین کوانتیزه کردن آن خود باعث اعمال ناهنجاری در صوت خواهد شد.

مطابق با مطالب بیان شده می‌دانیم که یکی از ویژگی‌های سیگنال Nyquist حذف ISI خواهد بود، اما برای این امر باید پارامترهای این سیگنال و پهنای باند کانال ما شرایطی را داشته باشند به عبارت دیگر باید $B = \frac{r}{2} + \beta$ بوده و شروط لازم $B \leq r \leq 2B$ و $0 \leq \beta \leq \frac{r}{2}$ نیز برقرار باشد [1].

۰-۲-۱ اجرای کد و ذخیره فایل‌های صوتی به ازای $f_s = 4000$ ، $\nu = 4$، $r = 20$، $\beta = 6$ ، $B = 12$

در این حالت فایل برنامه را به ازای نویزهای مختلف اجرا کرده و خروجی صدا را در یک پوشه مجزا قرار داده‌ایم. همانگونه که انتظار می‌رود با افزایش نویز صدای ما از حالت اصلی فاصله می‌گیرد، به نحوی که برای نسبت توان سیگنال به نویز ۱۰ صدای اصلی گم خواهد شد. انتظار می‌رود که با افزایش سطح کوانتیزاسیون بخشی از این ناهنجاری که به دلیل کوانتیزه شدن سیگنال اضافه شده است کم شود اما صفر نخواهد شد.

یکی دیگر راهکارها برای نزدیک‌شدن به صدای اصلی طراحی کانال برای حذف ISI خواهد بود.

[1] به دلیل کوتاه شدن حجم گزارش از تکرار نمودارهای خروجی پرهیز شده است درحالی که تمامی نمودارها با اجرای کد قابل حصول خواهد بود

۰-۲-۲ اجرای کد و ذخیره فایل‌های صوتی به ازای $r = 20$، $\nu = 4$ ، $f_s = 4000$ $B = 16$ ، $\beta = 6$،

مشابه قبل در این حالت فایل برنامه را به ازای نویزهای مختلف اجرا کرده و خروجی صدا را در یک پوشه مجزا قرار داده‌ایم. همانگونه که انتظار می‌رود با افزایش نویز صدای ما از حالت اصلی فاصله می‌گیرد، اما کیفیت صدا و نزدیک بودن آن به صدای اصلی نسبت به حالت قبل به دلیل عدم حضور ISI بهتر خواهد بود.

۰-۲-۳ اجرای کد و ذخیره فایل‌های صوتی به ازای $r = 20$، $\nu = 4$ ، $f_s = 4000$ $A = 1$ ، $B = 16$ ، $\beta = 3$،

مشابه قبل در این حالت فایل برنامه را به ازای نویزهای مختلف اجرا کرده و خروجی صدا را در یک پوشه مجزا قرار داده‌ایم. همانگونه که انتظار می‌رود با افزایش نویز صدای ما از حالت اصلی فاصله می‌گیرد، اما از آنجا که کاهش β باعث نزدیک‌تر شدن پالس خروجی به پالس ایده‌آل می‌شود خروجی نیز بهتر خواهد بود، اما میزان این بهبود به دلیل وجود ISI نسبت به حالت $B = 16$ ، $\beta = 6$ مشهود نیست.

۰-۲-۴ اجرای کد و ذخیره فایل‌های صوتی به ازای $r = 20$، $\nu = 4$ ، $f_s = 4000$ $A = 5$ ، $B = 16$ ، $\beta = 3$،

افزایش توان سیگنال باعث تقویت سیگنال اصلی شده و سیگنال خروجی با نویزی برابر با حالت مشابه دارای کیفیت صدای بهتری خواهد بود.

۰-۲-۵ اجرای کد و ذخیره فایل‌های صوتی به ازای $r = 20$، $\nu = 4$ ، $f_s = 44100$ $B = 16$ ، $\beta = 6$،

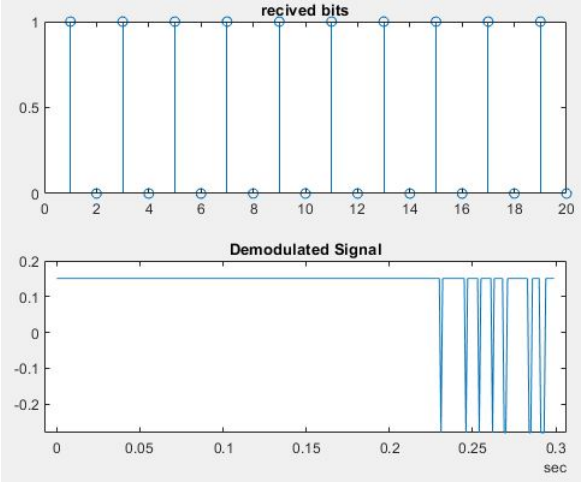
در این حالت به دلیل زیاد بودن تعداد نمونه‌ها و طولانی شدن فرآیند محاسبات تنها به اجرای کد برای یک حالت نویز اکتفا می‌کنیم (نسبت توان سیگنال به نویز ۱۰۰۰). اما همانگونه که انتظار می‌رود به دلیل افزایش تعداد نمونه‌ها صدای شنیده شده بعد از کوانتازیسیون نسبت به فرکانس قبل بهتر بوده و خروجی نیز بهتر خواهد بود.

۰-۲-۶ اجرای کد و ذخیره فایل‌های صوتی به ازای $r = 20$، $\nu = 2^8$ ، $f_s = 4000$ $B = 16$ ، $\beta = 6$،

در این حالت به دلیل زیاد بودن تعداد نمونه‌ها و طولانی شدن فرآیند محاسبات تنها به اجرای کد برای یک حالت نویز اکتفا می‌کنیم (نسبت توان سیگنال به نویز ۱۰۰۰). اما همانگونه که انتظار می‌رود به دلیل افزایش تعداد سطوح کوانتیزه‌شده صدای شنیده شده بعد از کوانتازیسیون و خروجی نسبت به حالت مشابه قبل بسیار بهتر بوده است.

۰-۳ محاسبه BER

به منظور محاسبه بهتر BER نیاز است تا مشابه قبل فایل‌ها را برای حالت‌های مختلف بالا با اعمال حلقه مونتیکارلو یر روی میزان نویز (کانال) اجرا کرده و سپس متوسط خطای بیت خروجی را برای هر حالت به



شکل ۸: نمودار متوسط خطای بازیابی به ازای توان نویزهای مختلف

صورت مجزا به عنوان نتیجه نهایی اعلام کرد.

از آنجا که کلیت و کیفیت خروجی در بخش قبل توضیح داده‌شد و به منظور پرهیز از تکرار مکررات نمودار BER را تنها برای یکی از حالت‌های قبل گزارش شده است؛ در حالی که می‌توان فایل BER pr را به ازای تمام حالت‌های قبل اجرا کرد.

۰-۳-۱ اجرای کد و ذخیره فایل‌های صوتی به ازای $f_s = 4000$ ، $\nu = 4$، $r = 20$، $\beta = 6$، $B = 12$

شکل ۸ نمودار متوسط خطای بازیابی به ازای توان نویزهای مختلف را نشان می‌دهد. همانگونه که واضح است افزایش نویز باعث کاهش دقت خواهد شد.

۰-۴ پیاده‌سازی مدار به صورت realtime

برای این امر یک object را تعریف می‌کنیم، تا صدای ما را به اندازه ۳ ثانیه دریافت کند، حال صدای ما به عنوان ورودی برای تابع اصلی عمل خواهد کرد، مابقی مراحل مشابه قبل خواهد بود. اگرچه این فرآیند کاملا realtime نیست ولی به نظر می‌رسد اگر طول سیگنال را کوتاه کنیم بتوان به صورت سریع و real time فرآیند پردازش را انجام داد.

۰-۵ کدهای مربوطه

در این بخش، کدهای اجراشده در طول پروژه قرار داده شده اند.

کدهای مربوط به تابع Main

```
%Main
clc;
clear all;
close all;
%%%%ADC_Function Input
n=input('Enter for n-bit PCM system : '); %Encodebook Bit Length
fs=input('Enter Sampling Frequency : '); %Sampling Frequency
signal= input('Enter voice signal: '); % Inpit Signal

%%LineCode_function_Input
rolloff=input('Enter rolloff factor : '); % rolloff factor
r =input('Enter baud rate r: '); % baud rate
A = input('Enter amplitude A: '); % amplitude

%channel Input
B = input('Enter channel bandwidth : ') ;
N0 = input('Enter noise power spectral density : ') ;
A_C = input('Enter amplitude A_Channel: '); % amplitude

% n=2; %Encodebook Bit Length
% fs=4000; %Sampling Frequency
% signal=  'voice.wav'; % Inpit Signal
% % rolloff factor
% r =20; % baud rate
% rolloff=6; % rolloff factor
% A = 1; % amplitude
% B = 12 ;
%
% A_C = 1;
t=0:0.001:0.1;
[y,Fs] = audioread(signal); % audio file
info = audioinfo(signal); % Information about audio file
[SerialCode,q ,Vmax,Vmin,len_t,len_ts] = PCM_function(signal , n ...
    , fs);
[Pulse, Power,s,len_p] = Linecode_function(rolloff,r, A , ...
    SerialCode);

Pulse_output_channel = Channel(B,N0*Power,A_C,Pulse);
 r_bit = Line_Decoder(Pulse_output_channel,SerialCode,r,s);
 sound_out = DAC_function(r_bit,n,Vmax,Vmin,len_p,len_t,len_ts)
sound(sound_out,Fs)
%%Write .wav
formatSpec = 'output_file_%d_.wav'
output_file = sprintf(formatSpec,N0/Power);
audiowrite(output_file,sound_out,Fs);
```

کدهای مربوط به تابع pcm.function

```
function [SerialCode,q,Vmax,Vmin,len_t,len_ts] = ...
    PCM_function(signal , n , fs )


%% sounf file
 %signal = 'alaw08m.wav'
[y,Fs] = audioread(signal); % audio file
info = audioinfo(signal); % Information about audio file
if info.NumChannels==2
   y= sum(y, 2) / size(y, 2);
end
% Time domain analysis
t=0:seconds (1/Fs):seconds(info.Duration);  % Time array
t=t(1:end-1); %Time index adjustment


L = 2^n; %Number of Quantisation Levels

%% Here we pllot the Analog Signal and its Sampled form
Vmax = max(y);
ActualSignl=y; %Actual input



t_sampled=0:seconds (1/fs):seconds(info.Duration);
len_t = length(t)
len_ts = length(t_sampled);
sampl = linspace(1 , length(y) , seconds(info.Duration) / ...
    seconds (1/fs));
Sampled_signal = ActualSignl(ceil(sampl));

% subplot(3,1,1);
% plot(t,y');
% xlabel('Time');
% ylabel('Amplitude');
% title('INPUT AUDIO signal');
% sound(y , Fs);
% keyboard
%  subplot(3,1,2); %Sampled Version
% stem(t_sampled(1:end-1), Sampled_signal);grid on; ...
    title('Sampled Sinal');
% sound(Sampled_signal,fs)
% keyboard

%% Now perform the Quantization Process
Vmin=min(y); %Since the Signal is sine
StepSize=(Vmax-Vmin)/L; % Diference between each quantisation level
```

```
45 QuantizationLevels=Vmin:StepSize:Vmax; % Quantisation Levels - ...
       For comparison
46 codebook=Vmin-(StepSize/2):StepSize:Vmax+(StepSize/2); % ...
       Quantisation Values - As Final Output of qunatiz
47 [ind,q]=quantiz(Sampled_signal,QuantizationLevels,codebook); % ...
       Quantization pprocess
48 NonZeroInd = find(ind ~= 0);
49 ind(NonZeroInd) = ind(NonZeroInd) - 1;
50 % MATLAB gives indexing from 1 to N. But we need indexing from 0, ...
       to convert it into binary codebook
51 BelowVminInd = find(q == Vmin-(StepSize/2));
52 q(BelowVminInd) = Vmin+(StepSize/2);
53 %This is for correction, as signal values cannot go beyond Vmin
54 %But quantiz may suggest it, since it return the Values lower ...
       than Actual
55
56 %Signal Value
57 % subplot(3,1,3);
58 % stem(t_sampled(1:end-1),q);
59 % grid on; % Display the Quantize values
60 % title('Quantized Signal');
61 % sound(q,fs)
62 % keyboard
63
64
65 % zoom=1500:1550+size(t,2)/120;
66 % figure(2);
67 % subplot(3,1,1);
68 % plot(t(zoom),ActualSignl(zoom),'b');
69 % title('Part of INPUT AUDIO signal');
70 % xlabel('Time');
71 % ylabel('Amplitude');
72 % subplot(3,1,2);
73 % stem(t(zoom),Sampled_signal(zoom),'b');
74 % title('Part of sampled signal');
75 % xlabel('Time');
76 % ylabel('Amplitude');
77 % subplot(3,1,3);
78 % stem(t(zoom),q(zoom),'b');
79 % title('Part of Quantized signal');
80 % xlabel('Time');
81 % ylabel('Amplitude');
82
83 %% Having Quantised the values, we perform the Encoding Process
84 TransmittedSig = de2bi(ind,'left-msb'); % Encode the Quantisation ...
       Level
85 SerialCode = reshape(TransmittedSig',[1 ...
       size(TransmittedSig,1)*size(TransmittedSig,2)]);
86 % TransmittedSig1 = de2bi(ind,'left-msb'); % Encode the ...
       Quantisation Level
```

```
% SerialCode1 = reshape(TransmittedSig1',[1 ...
    size(TransmittedSig1,1)*size(TransmittedSig1,2)]);
% figure()
% grid on;
% subplot(2,1,1)
% stairs(SerialCode); % Display the SerialCode Bit Stream
% axis([0 200 -2 3]);
% subplot(2,1,2)
% stem(SerialCode(1:20)); % Display the SerialCode Bit Stream
% title('Transmitted Signal');
end
```

کدهای مربوط به تابع Linecode.function

```
function [Pulse_out,Power,Len_R,len_p] = ...
    Linecode_function(rolloff,r,A , SerialCode)
%Raisecosine Signal
t = 0: 0.001:0.2;
Raise = A.*(cos(2*pi*rolloff*t)./((1-(4*rolloff*t).^2))) ...
    .*sinc(r*t);
Raise = [Raise(end:-1:2) , Raise];
Power = norm(Raise,2)^2;
Len_R = length(Raise);
Pulse = [];



for k=1:length(SerialCode)
    s =find(t==(1/r))-1;

    if k ==1
        T=  (cos(2*pi*rolloff*t)./((1-(4*rolloff*t).^2))) ...
            .*sinc(r*t);
        if SerialCode(k)==1


            Pulse(1, 1:length(T)) = T;

        end

        if SerialCode(k)==0


            Pulse(1, 1:length(T))=  -T;

        end

    else


    if SerialCode(k)==1

        if s*(k-1)+1-length(T)≤0
            Pulse(2, 1:s*(k-1)+length(T)) =  ...
                Raise(abs(s*(k-1)+1-length(T))+1:end);


        else
            Pulse(2, s*(k-1)+1-length(T):s*(k-1)+length(T)-1) =  ...
                Raise;

```

```
                    end

    end

    if SerialCode(k)==0


        if s*(k-1)+1-length(T)≤0
            Pulse(2, 1:s*(k-1)+length(T)) =  ...
                -Raise(abs(s*(k-1)+1-length(T))+1:end);



        else
        Pulse(2, s*(k-1)+1-length(T):s*(k-1)+length(T)-1) =  -Raise;


    end
    end
     Pulse(1,:) = sum(Pulse);
     Pulse(2,:) = [];
    end


end
Pulse_out = Pulse(1,:);
% tb=0:seconds (0.001):seconds(0.3);
% tb=tb(1:end-1);
% figure
%
% subplot(3,1,1)
% stem(SerialCode(1:20));
% title('Transmitted Signal ')
%
% subplot(3,1,2)
% plot(Raise);
% title('Raisecoseine ')
% marker = 1:s:numel(tb);
% subplot(3,1,3)
% plot(tb, Pulse_out(1,1:numel(tb)), 'o-','MarkerIndices',marker)
% title('Line coded Transmited Signal ')
Len_R =s;
len_p=k;
end
```

کدهای مربوط به تابع Channel

```
function [y] = Channel(B,N0,A,signal )
sigma = sqrt(N0*B);
t = 0: 0.001:0.1;
x = sigma*randn(1,length(signal));
mam_mod = max(signal);

noise_Signal = signal + x ;
chnlp_withnoise = conv(noise_Signal, ...
    abs(t(2)-t(1))*6*B*sinc(2*B*t));
chnlp_withoutnoise = conv(signal, abs(t(2)-t(1))*6*B*sinc(2*B*t));

chnlp_withoutnoise = ...
    chnlp_withoutnoise*mam_mod/max(chnlp_withoutnoise);
chnlp_withnoise = chnlp_withnoise*mam_mod/max(chnlp_withnoise);

mam_mod_noise = max(chnlp_withnoise);
mam_mod_withoutnoise = max(chnlp_withoutnoise);

chnlp_withnoise = ...
    chnlp_withnoise*mam_mod_noise/max(chnlp_withnoise);
chnlp_withnoise = chnlp_withnoise * A;

chnlp_withoutnoise = ...
    chnlp_withoutnoise*mam_mod_withoutnoise/max(chnlp_withoutnoise);

chnlp_withoutnoise = chnlp_withoutnoise * A;
%figure
% tb=0:seconds (0.001):seconds(0.3);
% tb=tb(1:end-1);
%
% subplot(3,1,1)
% plot(tb, signal(1:numel(tb)))
% title('Line coded Transmited Signal ')
% subplot(3,1,2)
% plot(tb, chnlp_withoutnoise(1:numel(tb)))
% title('Lp signal without Noise')
% subplot(3,1,3)
% plot(tb, chnlp_withnoise(1:numel(tb)));
% title('Lp signal with Noise')

y = chnlp_withnoise;
end
```

کدهای مربوط به تابع Line.Decoder

```
function [r_bit] = Line_Decoder(signal,serialcode,r,s)
r_bit=[];

for k =1:s:length(signal)
    if  signal(k) ≥0
       r_bit = [r_bit , 1] ;
    else
        r_bit = [r_bit , 0];
    end
end
% tb=0:seconds (0.001):seconds(0.3);
% tb=tb(1:end-1);
% figure
% subplot(3,1,1)
% plot(tb, signal(1,(1:numel(tb))))
% title('Line decoder recived Signal ')
% subplot(3,1,2)
% stem(r_bit)
% title('recived bit after decoder ')
% subplot(3,1,3)
% stem(serialcode)
% title('original transmited bit')
end
```

کدهای مربوط به تابع DAC.function

```
function [sound_out_interp] = ...
    DAC_function(recive_bit,n,Vmax,Vmin,len_p,len_t,len_ts)
%% Now we perform the Demodulation Of PCM signal
%RecievedCode=reshape(recive_bit,n,ceil(length(recive_bit)/n)); ...
    %Again Convert the SerialCode into Frames of 1 Byte
recive_bit = recive_bit(1:len_p);
RecievedCode=reshape(recive_bit,n,len_p/n);
index = bi2de(RecievedCode','left-msb'); %Binary to Decimal ...
    Conversion
StepSize = (Vmax-Vmin)/(2^n);
q = (StepSize*index); %Convert into Voltage Values
q = q + (Vmin+(StepSize/2)); % Above step gives a DC shfted ...
    version of Actual siganl
%Thus it is necessary to bring it to zero level
% figure()
% subplot(2,1,1)
% stem(recive_bit);
% title('resived bits');
% subplot(2,1,2);
% grid on;
% plot(q); % Plot Demodulated signal
%
% title('Demodulated Signal');

%upsampling

sound_out = q;
sound_out_interp = interp(q, ceil(len_t/len_ts));
end
```

کدهای مربوط به تابع BER

```
%Calculate BER
clc;
clear all;
close all;
n=input('Enter for n-bit PCM system : '); %Encodebook Bit Length
fs=input('Enter Sampling Frequency : '); %Sampling Frequency
signal= input('Enter voice signal: '); % Inpit Signal

%%LineCode_function_Input

rolloff=input('Enter rolloff factor : '); % rolloff factor
r =input('Enter baud rate r: '); % baud rate
A = input('Enter amplitude A: '); % amplitude

%channel Input
B = input('Enter channel bandwidth : ') ;
N0 = input('Enter noise power spectral density : ') ;
A_C = input('Enter amplitude A_Channel: '); % amplitude


% n=2; %Encodebook Bit Length
% fs=4000; %Sampling Frequency
% signal=  'voice.wav'; % Inpit Signal
% % rolloff factor
% r =20; % baud rate
% rolloff=6; % rolloff factor
% A = 1; % amplitude
% B = 12 ;
%
% A_C = 1;
t=0:0.001:0.1;
[y,Fs] = audioread(signal); % audio file
info = audioinfo(signal); % Information about audio file
[SerialCode,q ,Vmax,Vmin,len_t,len_ts] = PCM_function(signal , n ...
    , fs);
[Pulse, Power,s,len_p] = Linecode_function(rolloff,r, A , ...
    SerialCode);

BER = [];
error_norm_F = [];
for N0 = 0:0.001:0.5
     error_m = [];
     error_norm = [];
 for Monti=1:5
Pulse_output_channel = Channel(B,N0,A_C,Pulse);
 r_bit = Line_Decoder(Pulse_output_channel,SerialCode,r,s);
  sound_out = DAC_function(r_bit,n,Vmax,Vmin,len_p,len_t,len_ts)
```

```
% error_m =[error_m , ...
    sum(abs(r_bit(1:length(SerialCode))-SerialCode))];
error_m =[error_m , ...
    sum(abs(r_bit(1:length(len_p))-SerialCode(1:length(len_p))))];
error_norm = [error_norm , norm( sound_out' - interp(q, ...
    ceil(len_t/len_ts)), 2)]
 end
 BER = [BER , mean(error_m)];
 error_norm_F = [error_norm_F , mean(error_norm)/norm(interp(q, ...
     ceil(len_t/len_ts)))];
end
figure()
plot(0:0.001:0.5 , error_norm_F)
title('error recovery')
```

کدهای مربوط به تابع RealTime

```
%%RealTIME
clc;
clear all;
close all;

%%get audio
recObj = audiorecorder;
disp('Start speaking.')
recordblocking(recObj, 3);
disp('End of Recording.');
y = getaudiodata(recObj);
formatSpec = 'input_file_.wav'
output_file = sprintf(formatSpec);
audiowrite(output_file,y,8000);
figure
plot(y);
sound(y)
keyboard
%%%ADC_Function Input
n=input('Enter for n-bit PCM system : '); %Encodebook Bit Length
fs=input('Enter Sampling Frequency : '); %Sampling Frequency
signal= input('Enter voice signal: '); % Inpit Signal

%%LineCode_function_Input
rolloff=input('Enter rolloff factor : '); % rolloff factor
r =input('Enter baud rate r: '); % baud rate
A = input('Enter amplitude A: '); % amplitude

%channel Input
B = input('Enter channel bandwidth : ') ;
N0 = input('Enter noise power spectral density : ') ;
A_C = input('Enter amplitude A_Channel: '); % amplitude

% n=2; %Encodebook Bit Length
% fs=4000; %Sampling Frequency
%
% rolloff factor
% r =20; % baud rate
% rolloff=6; % rolloff factor
% A = 1; % amplitude
% B = 12 ;
%
% A_C = 1;
% t=0:0.001:0.1;
% [y,Fs] = audioread(signal); % audio file
% info = audioinfo(signal); % Information about audio file
% [SerialCode,q ,Vmax,Vmin,len_t,len_ts] = PCM_function(signal , ...
    n , fs);
```

```
% [Pulse, Power,s,len_p] = Linecode_function(rolloff,r, A , ...
    SerialCode);
% N0 = 1e-2*Power;
Pulse_output_channel = Channel(B,N0,A_C,Pulse);
 r_bit = Line_Decoder(Pulse_output_channel,SerialCode,r,s);
 sound_out = DAC_function(r_bit,n,Vmax,Vmin,len_p,len_t,len_ts)
sound(sound_out,Fs)
%%Write .wav
formatSpec = 'output_file_%d_.wav'
output_file = sprintf(formatSpec,N0/Power);
audiowrite(output_file,sound_out,Fs);
```