GSM®

GLOBAL SYSTEM FOR
MOBILE COMMUNICATIONS

GSM Matlab Project Report

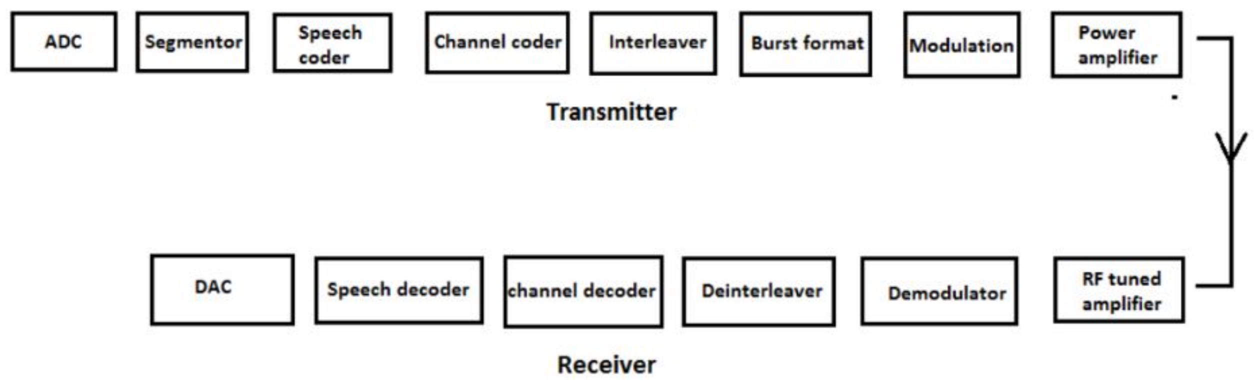
By

Asmaa Mohammed Salah Eldine Ismail	44
Asmaa Mostafa Saad Abdelhamid Ouf	45
Shahd Ahmed Mohammed Elkabbary	94
Mohammed Wagdy Nomeir	176
Marwan Nabil Mohammed Basiouny	189
Moustafa Raafat Mostafa Mohammed	193

Overview

The goal of this code is to simulate a simplified GSM mobile communication system using matlab.

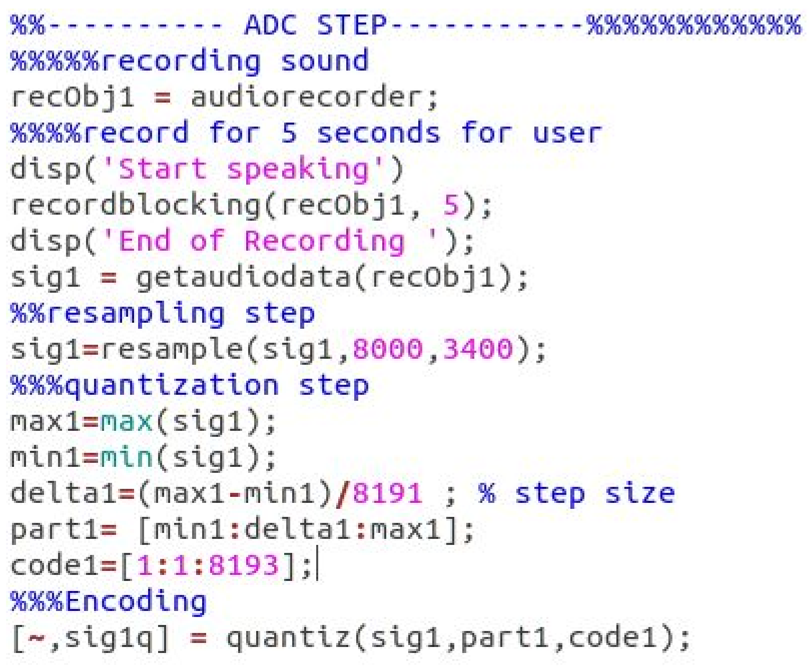
The code is separated into 2 matlab source code files (GSM_Tx.m and GSM_Rx.m), and is logically separated into sequential modules corresponding to each of the following blocks in this block diagram:



Following, is a step by step discussion of each of these modules, along with screenshots and comments.

We begin by examining the transmitter.

Analog to Digital Converter



```
%%---------- ADC STEP----------%%%%%%%%%%%
%%%%%recording sound
recObj1 = audiorecorder;
%%%%record for 5 seconds for user
disp('Start speaking')
recordblocking(recObj1, 5);
disp('End of Recording ');
sig1 = getaudiodata(recObj1);
%%resampling step
sig1=resample(sig1,8000,3400);
%%%quantization step
max1=max(sig1);
min1=min(sig1);
delta1=(max1-min1)/8191 ; % step size
part1= [min1:delta1:max1];
code1=[1:1:8193];
%%%Encoding
[~,sig1q] = quantiz(sig1,part1,code1);
```

Comments and plots

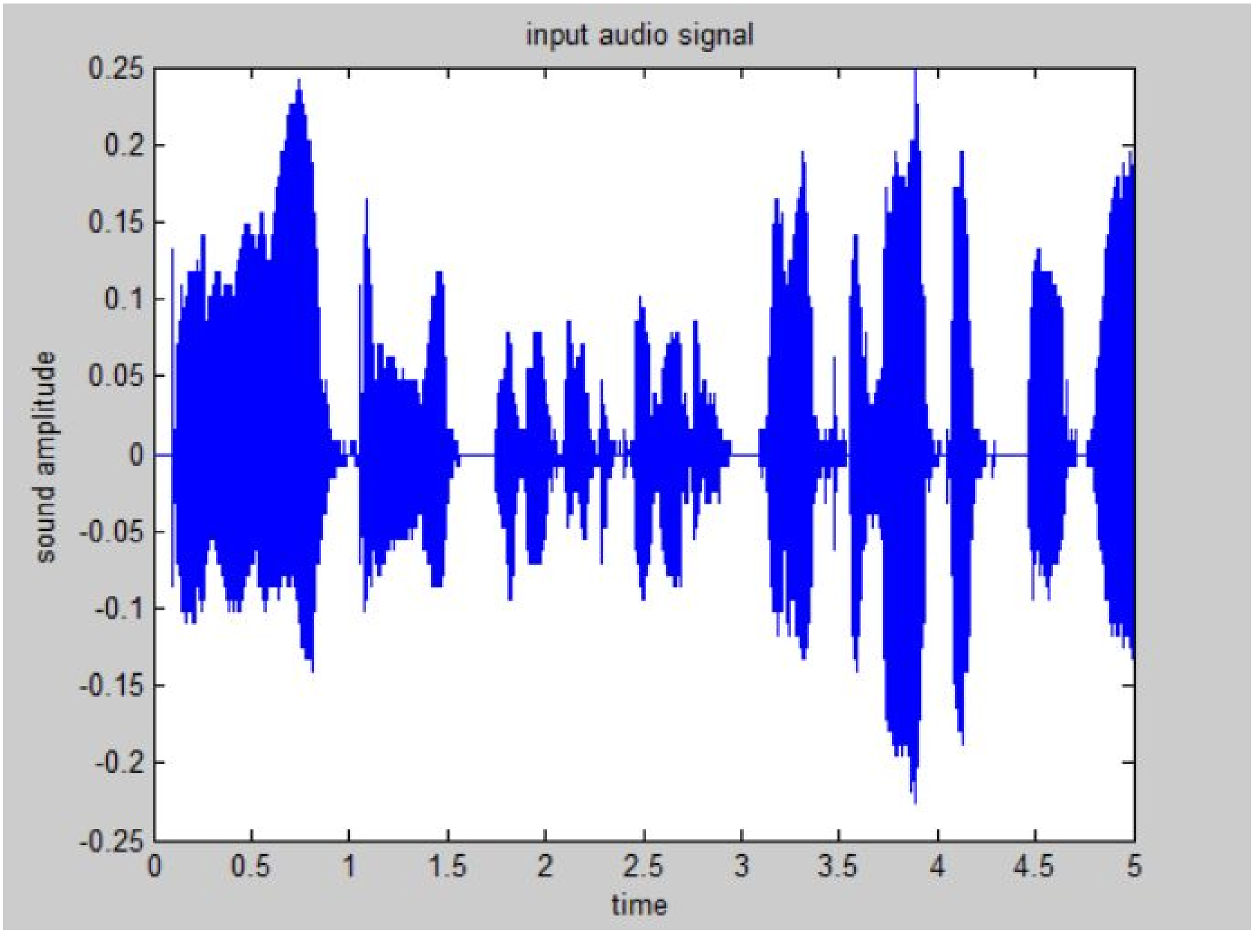
We use a built-in matlab helper object to record the user's voice (5 seconds of it), this recording uses the computer's built-in ADC to produce a digital form of the voice, but this format uses a much higher sampling rate than needed (depends on the audio hardware of the computer), so a resampling step is essential.

We move on to the quantization step, we first determine the step size according to the minimum and maximum amplitudes of the speech content.

We use this delta to construct the partitions vector, which delimits the absolute start and end of each partition.

The quantiz() call produces the encoded digital signal (performs both quantization and encoding), it requires both the partitions vector and the codes vector (codes corresponding to these partitions).

[a plot of original speech signal]

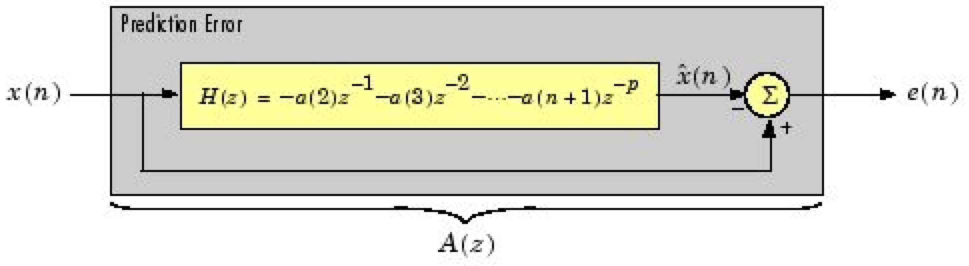


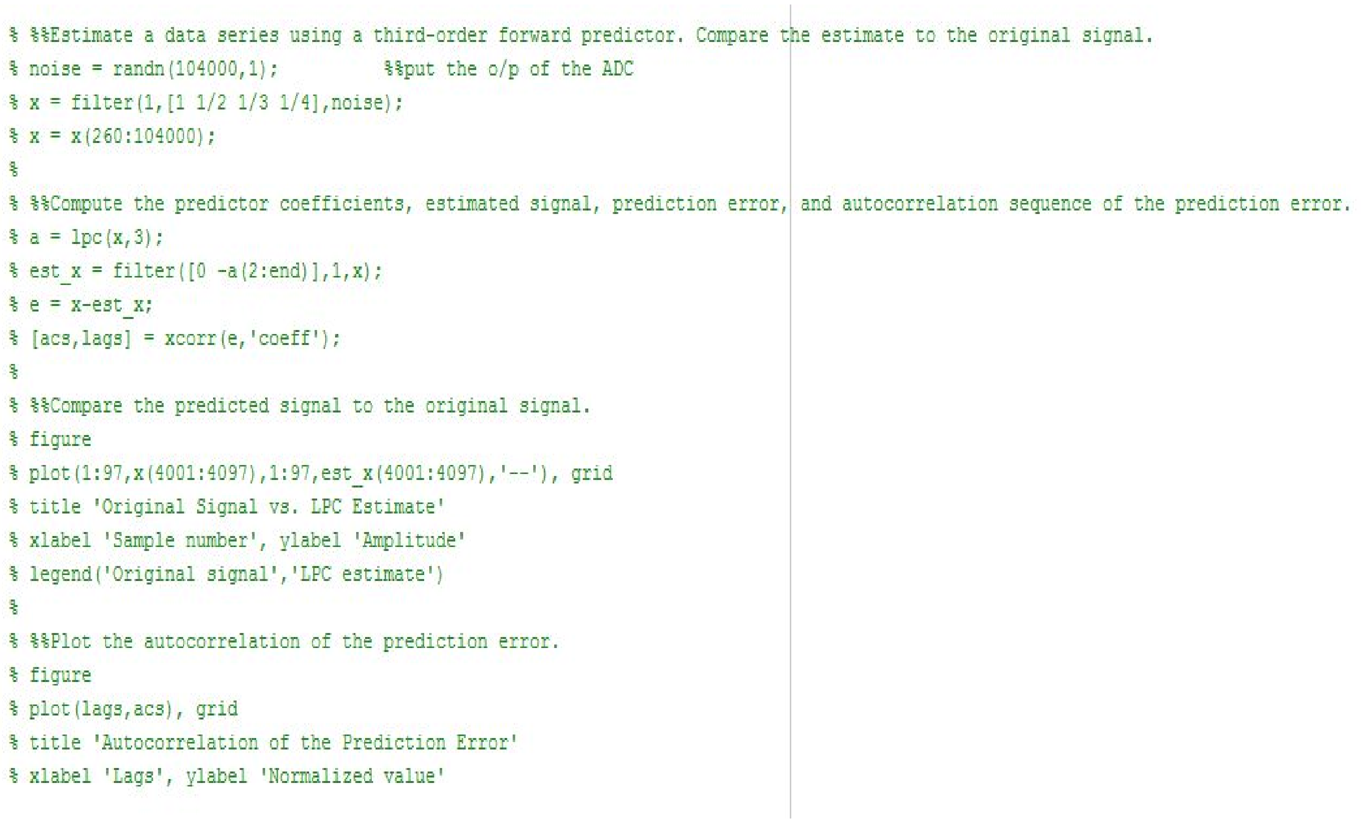
SEGMENTATION AND SPEECH CODING

```
%%&&&&&&&&&&&&&&&&&&&&&&&&&&&& Segmentation and Speech Coding Part &&&&&&&&&&&&&&&&&&&&&&&&&

%%%%%%%%%%%%%%%%%%%%%%%% Segmentation Part &&&&&&&&&&&&&&&&&&&&&&&&&
%%%% we will use downsampling concept to reduce data rate
% Downsample the quantized_samples (ADC signal o/p) as required to have 260 bits/segment
down_sampled_signal = downsample(sig1q,8);
%%appending zero samples in order to have chunks of 20 samples
down_sampled_signal = [down_sampled_signal  zeros(20-mod(length(down_sampled_signal),20),1)'];
output_stream = de2bi(down_sampled_signal,13); %% encoding the results
%%%%%%%%%%%%%%%% Concatinating samples belong to the same segment (20 ms)
temporary_block=[];
blocks_before_channel_coding=[];
for k=1:20:length(output_stream) % concatenate every 20 samples to have 20*13 bits for every 20ms
    temporary_block=[];
    for m=k:k+19
        temporary_block=[temporary_block output_stream(m,:)];
    end
    blocks_before_channel_coding=[blocks_before_channel_coding ; temporary_block];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Comments and plots

We use the concept of Down sampling to reduce the data rate of ADC output , however the most correct way is to use LPC .



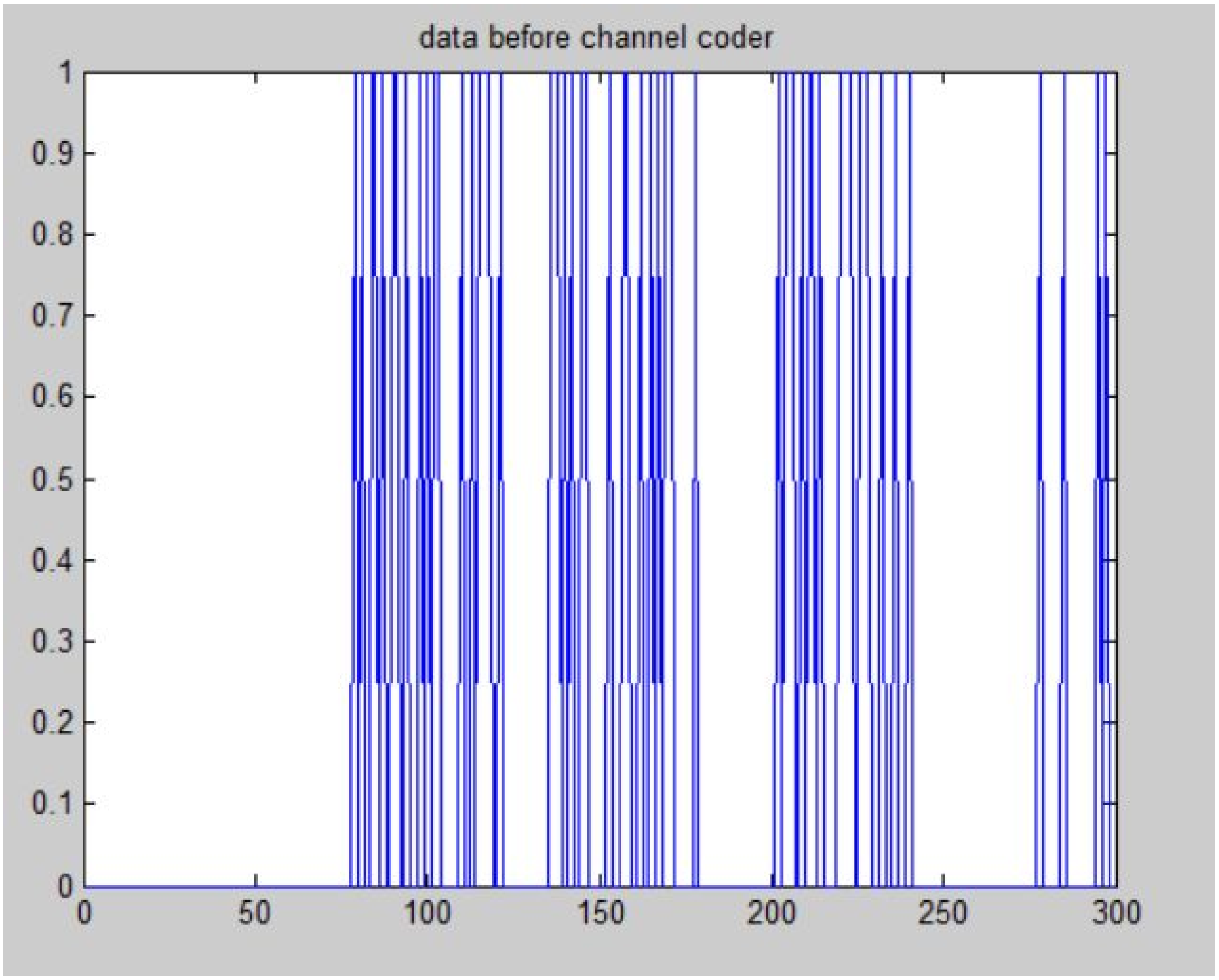


```
% %%Estimate a data series using a third-order forward predictor. Compare the estimate to the original signal.
% noise = randn(104000,1);          %%put the o/p of the ADC
% x = filter(1,[1 1/2 1/3 1/4],noise);
% x = x(260:104000);
%
% %%Compute the predictor coefficients, estimated signal, prediction error, and autocorrelation sequence of the prediction error.
% a = lpc(x,3);
% est_x = filter([0 -a(2:end)],1,x);
% e = x-est_x;
% [acs,lags] = xcorr(e,'coeff');
%
% %%Compare the predicted signal to the original signal.
% figure
% plot(1:97,x(4001:4097),1:97,est_x(4001:4097),'--'), grid
% title 'Original Signal vs. LPC Estimate'
% xlabel 'Sample number', ylabel 'Amplitude'
% legend('Original signal','LPC estimate')
%
% %%Plot the autocorrelation of the prediction error.
% figure
% plot(lags,acs), grid
% title 'Autocorrelation of the Prediction Error'
% xlabel 'Lags', ylabel 'Normalized value'
```

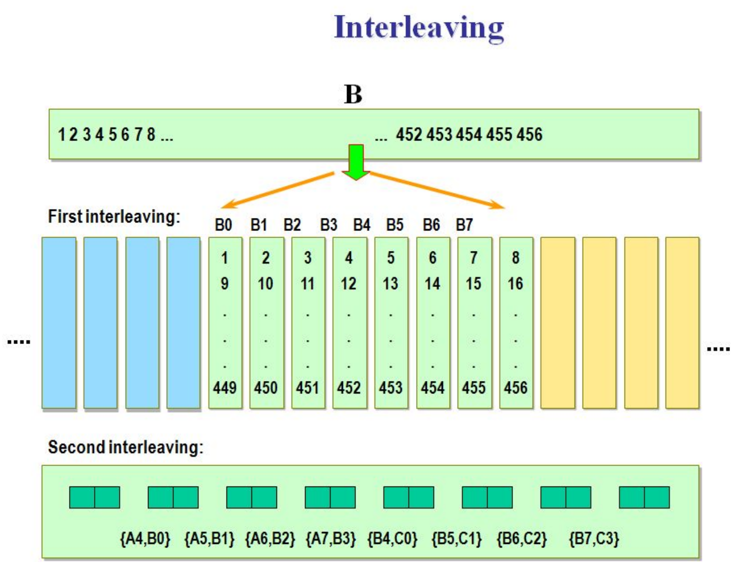
But we faced some difficulties while implementing this code , so we used the downsampling concept.

Channel coding

plots



Interleaving and De interleaving

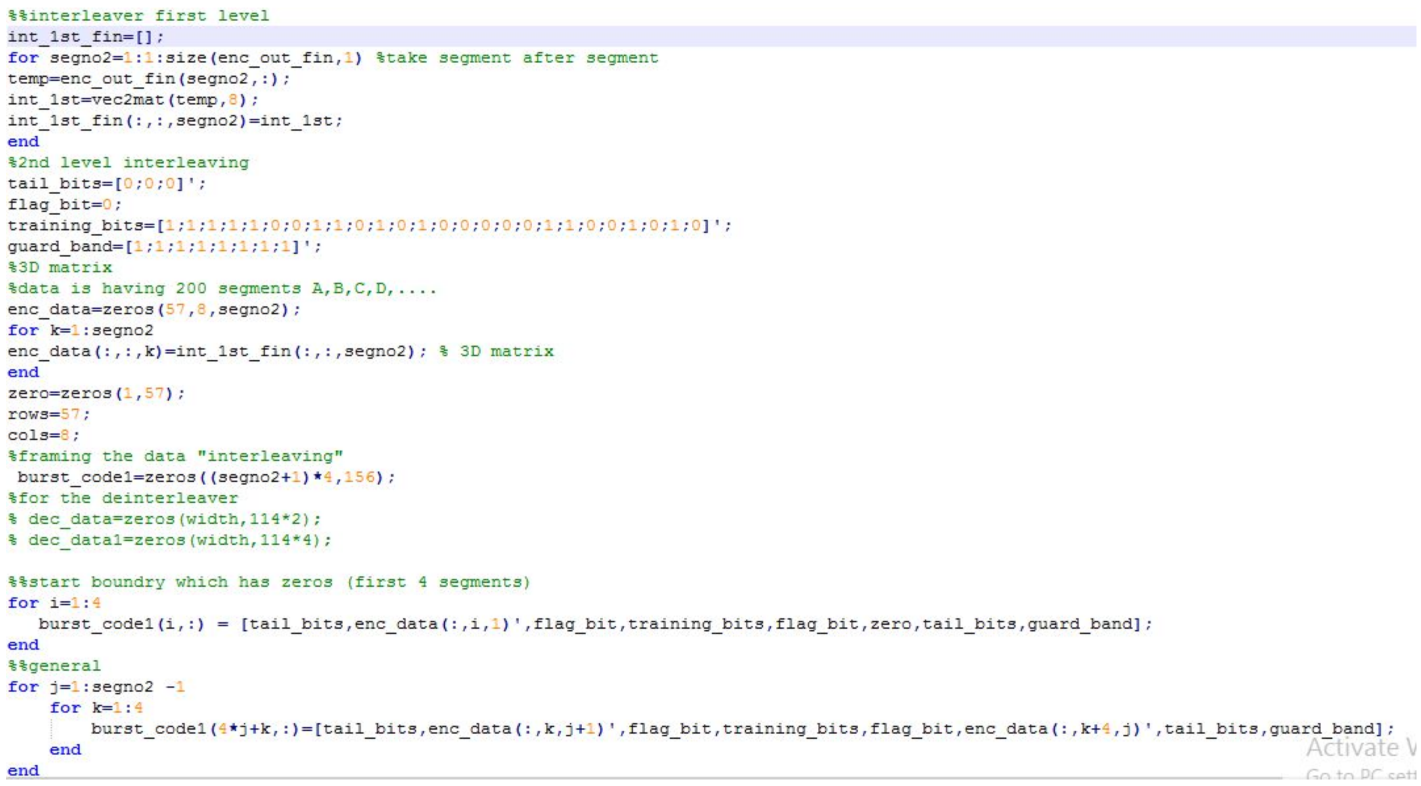


If the voice signal is transmitted directly after channel coding, a deep of the fading will influence a successive string of bits and cause high bit error rate.To solve this problem, some technique to separate the successive bits are required.This method is called interleaving technique. The interleaving method is the most effective coding method for dispersion of bit errors.

Implementation of the interleaver on matlab is done by the following steps bellow:

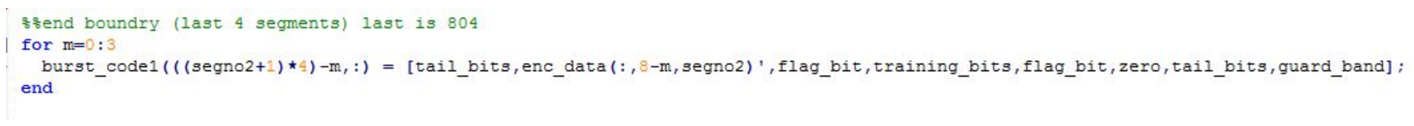
1) Dispersing the bits of a data burst into multiple bursts in a systematic format.
2) Putting this dispersed bits in a normal GSM burst format which is used for the standard communication between the base station and the mobile.

Interleaver Codes' screen shots:



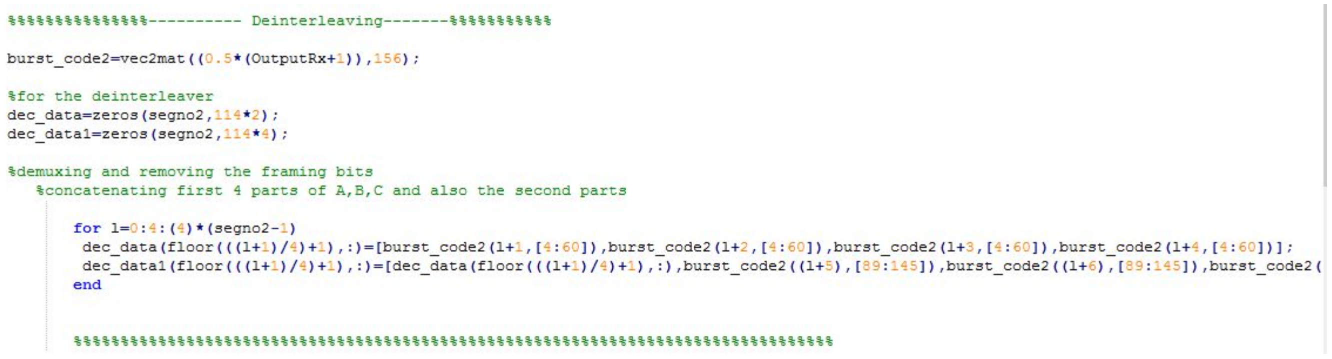
```
%%interleaver first level
int_1st_fin=[];
for segno2=1:1:size(enc_out_fin,1) %take segment after segment
temp=enc_out_fin(segno2,:);
int_1st=vec2mat(temp,8);
int_1st_fin(:,:,segno2)=int_1st;
end
%2nd level interleaving
tail_bits=[0;0;0]';
flag_bit=0;
training_bits=[1;1;1;1;1;0;0;1;1;0;1;0;1;0;0;0;0;0;1;1;0;0;1;0;1;0]';
guard_band=[1;1;1;1;1;1;1;1]';
%3D matrix
%data is having 200 segments A,B,C,D,....
enc_data=zeros(57,8,segno2);
for k=1:segno2
enc_data(:,:,k)=int_1st_fin(:,:,segno2); % 3D matrix
end
zero=zeros(1,57);
rows=57;
cols=8;
%framing the data "interleaving"
 burst_code1=zeros((segno2+1)*4,156);
%for the deinterleaver
% dec_data=zeros(width,114*2);
% dec_data1=zeros(width,114*4);

%%start boundry which has zeros (first 4 segments)
for i=1:4
   burst_code1(i,:) = [tail_bits,enc_data(:,i,1)',flag_bit,training_bits,flag_bit,zero,tail_bits,guard_band];
end
%%general
for j=1:segno2 -1
    for k=1:4
        burst_code1(4*j+k,:)=[tail_bits,enc_data(:,k,j+1)',flag_bit,training_bits,flag_bit,enc_data(:,k+4,j)',tail_bits,guard_band];
    end
end
```

Activate W
Go to PC sett



```
%%end boundry (last 4 segments) last is 804
for m=0:3
   burst_code1(((segno2+1)*4)-m,:) = [tail_bits,enc_data(:,8-m,segno2)',flag_bit,training_bits,flag_bit,zero,tail_bits,guard_band];
end
```

De interleaver Codes' screen shots:

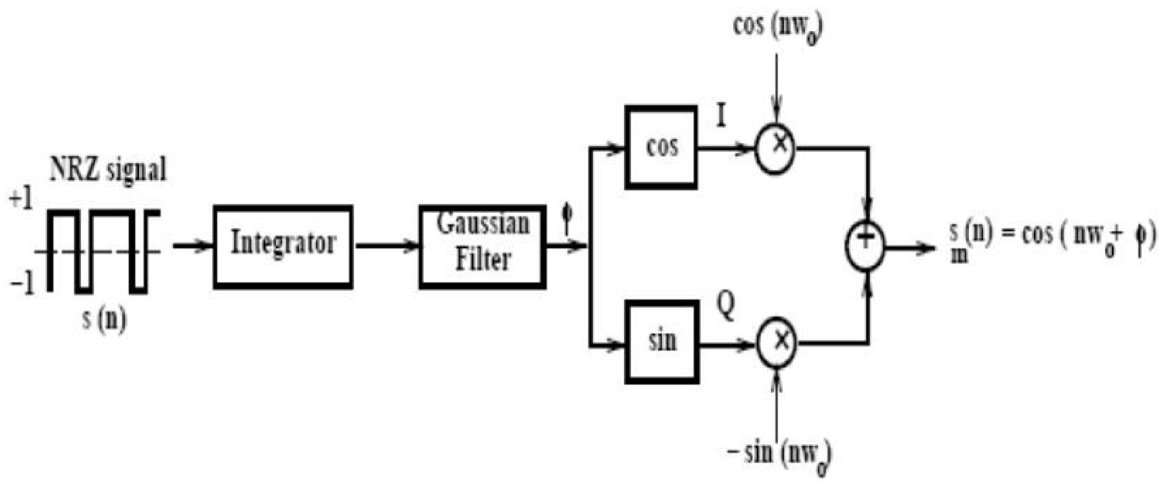


```
%%%%%%%%%%%%%%%----------- Deinterleaving-------%%%%%%%%%%

burst_code2=vec2mat((0.5*(OutputRx+1)),156);

%for the deinterleaver
dec_data=zeros(segno2,114*2);
dec_data1=zeros(segno2,114*4);

%demuxing and removing the framing bits
   %concatenating first 4 parts of A,B,C and also the second parts

      for l=0:4:(4)*(segno2-1)
       dec_data(floor(((l+1)/4)+1),:)=[burst_code2(l+1,[4:60]),burst_code2(l+2,[4:60]),burst_code2(l+3,[4:60]),burst_code2(l+4,[4:60])];
       dec_data1(floor(((l+1)/4)+1),:)=[dec_data(floor(((l+1)/4)+1),:),burst_code2((l+5),[89:145]),burst_code2((l+6),[89:145]),burst_code2(
      end


      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Modulation

The modulator type of the GSM is what is called Gaussian Minimum Shift Keying Modulation (GMSK).

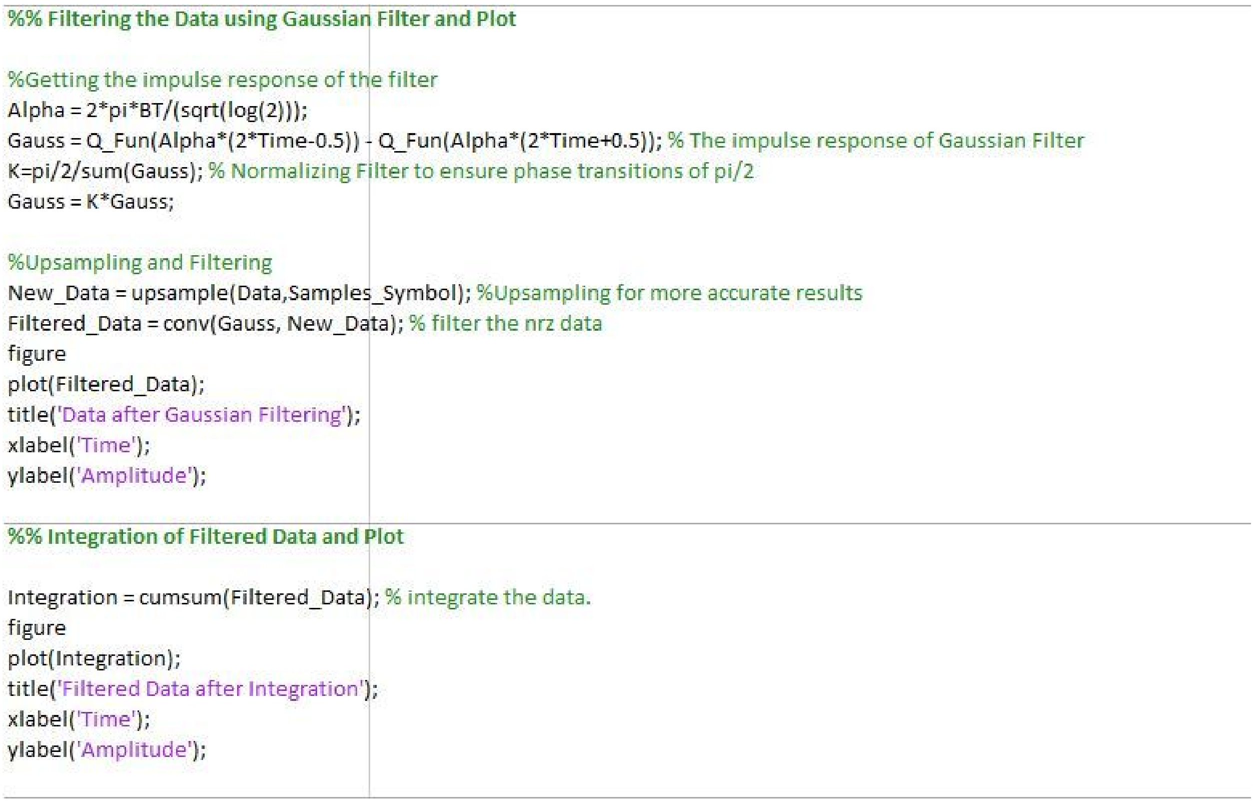
The modulator can be described by the following block diagram:



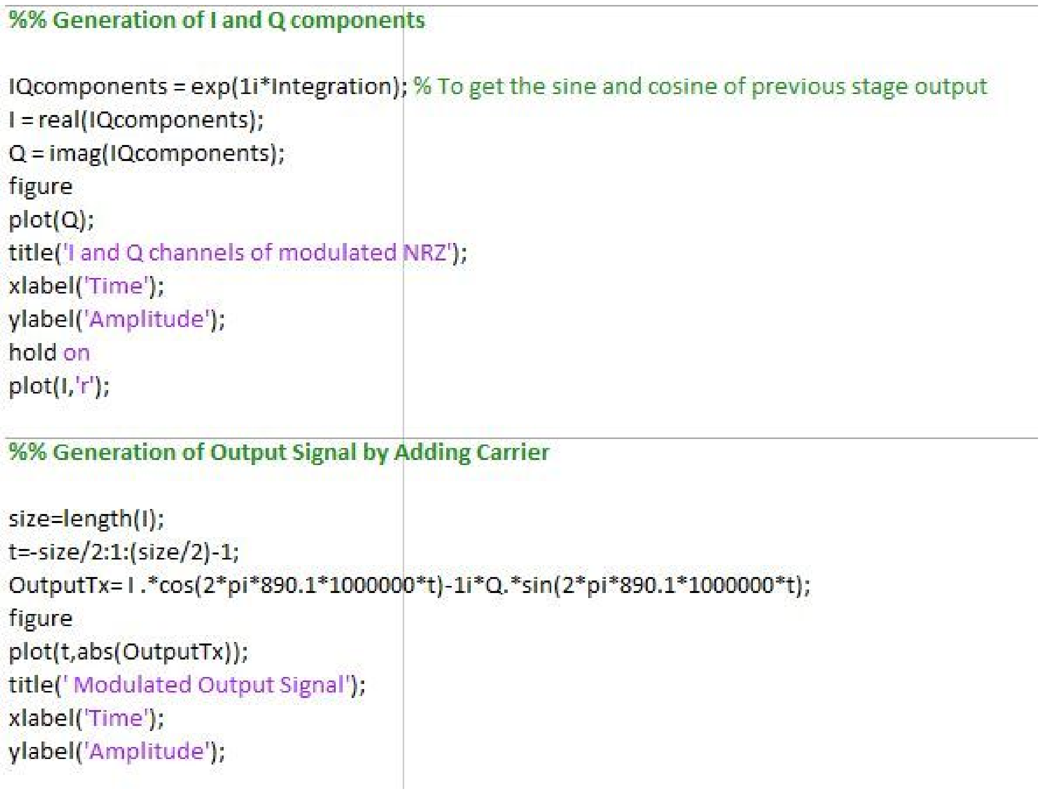
In the matlab implementation of the modulator, the code goes as follows:

- First, the impulse response of the Gaussian Filter is obtained.
- Next, the input data from the precedent stage is convolved with the filter.
- The convolution output is integrated.
- Then, I and Q components are obtained separately so they can get multiplied by sine/cosine in order to form the output modulated signal as depicted above.

The following are screenshots of the modulator code.



```
%% Filtering the Data using Gaussian Filter and Plot

%Getting the impulse response of the filter
Alpha = 2*pi*BT/(sqrt(log(2)));
Gauss = Q_Fun(Alpha*(2*Time-0.5)) - Q_Fun(Alpha*(2*Time+0.5)); % The impulse response of Gaussian Filter
K=pi/2/sum(Gauss); % Normalizing Filter to ensure phase transitions of pi/2
Gauss = K*Gauss;

%Upsampling and Filtering
New_Data = upsample(Data,Samples_Symbol); %Upsampling for more accurate results
Filtered_Data = conv(Gauss, New_Data); % filter the nrz data
figure
plot(Filtered_Data);
title('Data after Gaussian Filtering');
xlabel('Time');
ylabel('Amplitude');

%% Integration of Filtered Data and Plot

Integration = cumsum(Filtered_Data); % integrate the data.
figure
plot(Integration);
title('Filtered Data after Integration');
xlabel('Time');
ylabel('Amplitude');
```

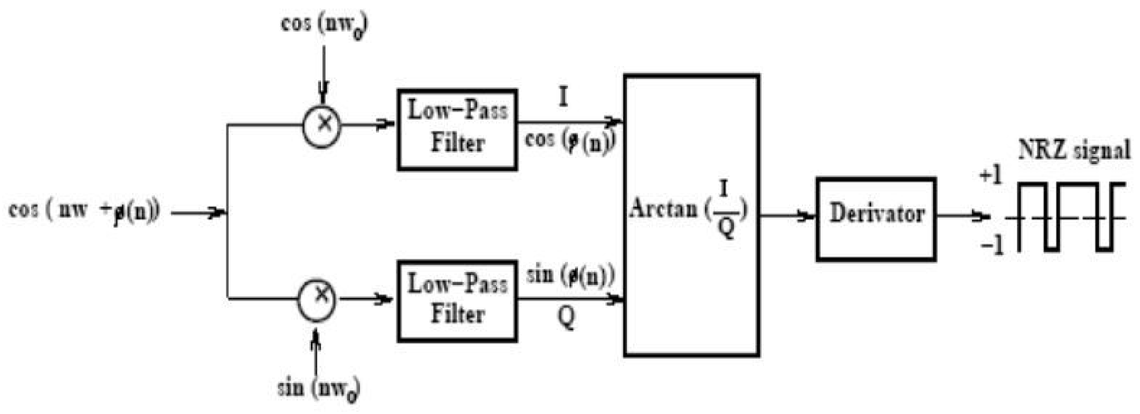


```
%% Generation of I and Q components

IQcomponents = exp(1i*Integration); % To get the sine and cosine of previous stage output
I = real(IQcomponents);
Q = imag(IQcomponents);
figure
plot(Q);
title('I and Q channels of modulated NRZ');
xlabel('Time');
ylabel('Amplitude');
hold on
plot(I,'r');

%% Generation of Output Signal by Adding Carrier

size=length(I);
t=-size/2:1:(size/2)-1;
OutputTx= I .*cos(2*pi*890.1*1000000*t)-1i*Q.*sin(2*pi*890.1*1000000*t);
figure
plot(t,abs(OutputTx));
title(' Modulated Output Signal');
xlabel('Time');
ylabel('Amplitude');
```

Next, we move on to examine the Receiver modules.

Demodulation

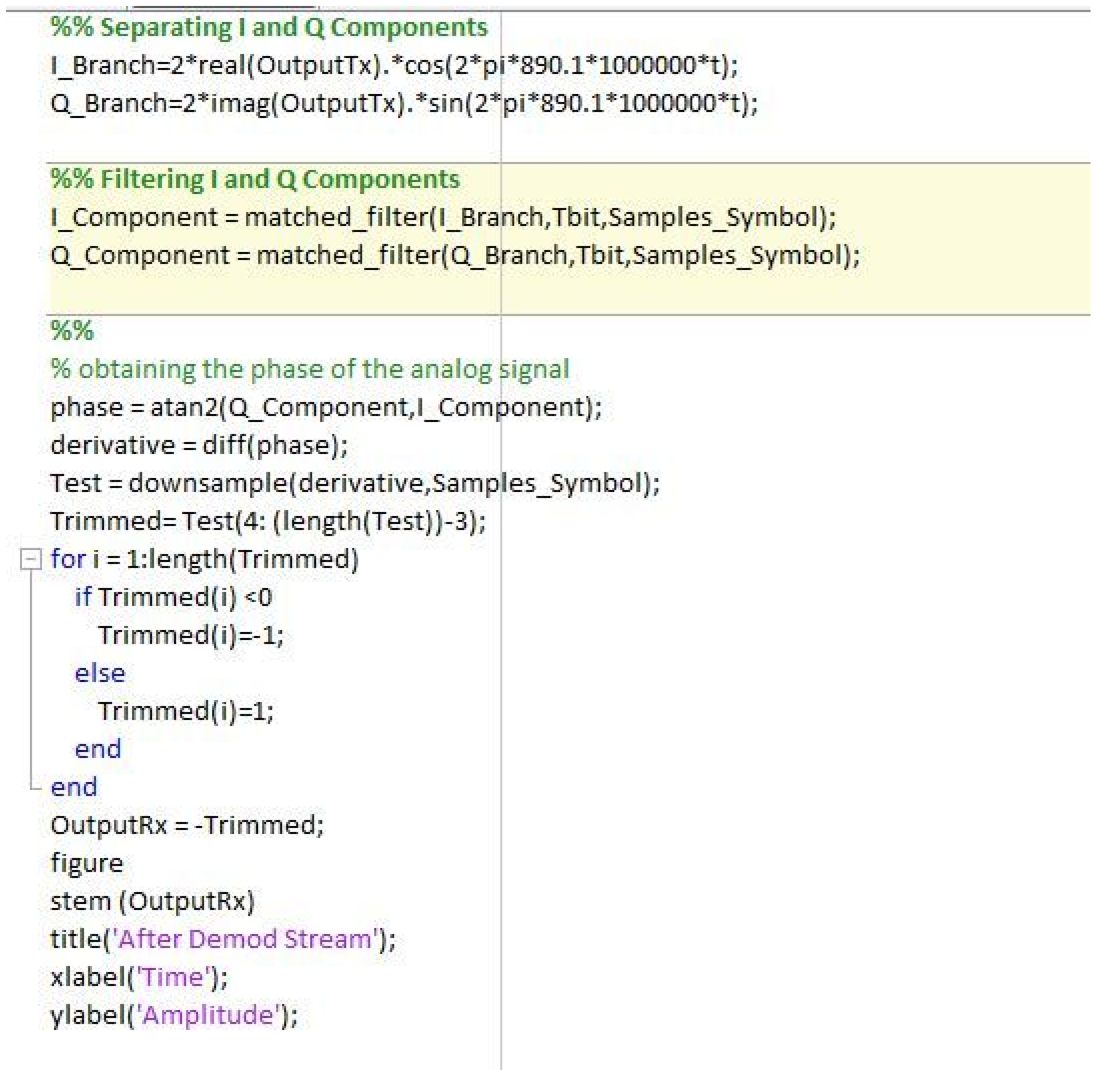
The demodulator can be described by the following block diagram:



In the matlab implementation of the modulator, the code goes as follows:

- The input modulated signal is multiplied by sine/cosine as a first step for obtaining I and Q components.
- The two outputs are then passed through matched filters to get the I and Q components.
- Inverse tan operation is performed of the ratio between both components.
- Finally, derivation is performed to get the final Non-return to Zero output representing the bit stream.

Next is the demodulator code screenshot.



```
%% Separating I and Q Components
I_Branch=2*real(OutputTx).*cos(2*pi*890.1*1000000*t);
Q_Branch=2*imag(OutputTx).*sin(2*pi*890.1*1000000*t);

%% Filtering I and Q Components
I_Component = matched_filter(I_Branch,Tbit,Samples_Symbol);
Q_Component = matched_filter(Q_Branch,Tbit,Samples_Symbol);

%%
% obtaining the phase of the analog signal
phase = atan2(Q_Component,I_Component);
derivative = diff(phase);
Test = downsample(derivative,Samples_Symbol);
Trimmed= Test(4: (length(Test))-3);
for i = 1:length(Trimmed)
  if Trimmed(i) <0
    Trimmed(i)=-1;
  else
    Trimmed(i)=1;
  end
end
OutputRx = -Trimmed;
figure
stem (OutputRx)
title('After Demod Stream');
xlabel('Time');
ylabel('Amplitude');
```

De Interleaver:

Code:

Comments and plots