Multiplexing (TDM) audio transfer

Prepared by:

Abhijit Das (Abhi JD)

Multiplexing (TDM) Audio Transfer

1 Introduction

In telecommunications and computer networks we use a method called multiplexing, which combines multiple digital or analogue signals into one signal stream which then can be transmitted through a medium. The advantage is that, it can reduce the overall expense of transmission by sharing a medium.

TDM (Time **D**ivision **M**ultiplexing) is a type of multiplexing method which uses the time division sharing. It works by switching to collect a sequence of bits from several inputs one after the other and sending it to the appropriate receiver. A small bit of information is taken from each input in sequence to form a stream of data with chunks of each bit of information to be transmitted which is then again distributed to the appropriate receiver. There are several problems related to TDM system, one being delay, but if done fast enough, the receiver will not detect significant delay.

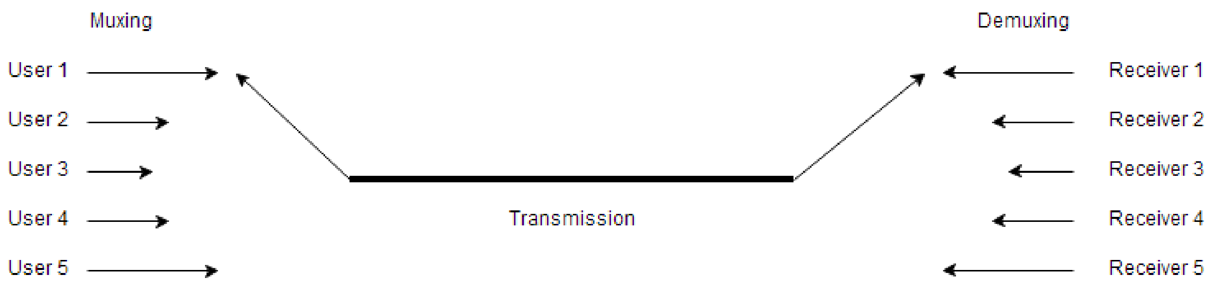

Figure 01: TDM concept diagram

Matlab Code Used in the Experiment

```
%% cleanup
close all;clear all;clc;
```

```
%signal conditioning and muxing for N input(s)

N=3; % number of users to add

for a=1:N
    if(N<10)
        eval(['[u' num2str(a) ' fs' num2str(a) ']=auread(''user0' 
num2str(a) '.au'');' ])
    else
        eval(['[u' num2str(a) ' fs' num2str(a) ']=auread(''user' num2str(a) 
'.au'');' ])
    end
end % loop end

ts=1/fs1; % sampling period
Ts = ts; % Slot time interval

ns = Ts/ts; %Ts/ts;

% disp('before conditioning');
% pause
% sound(u1)
% pause
% sound(u2)
% pause
% sound(u3)

%% adjustments
for s=1:N    %TO find LENGTH
    eval(['zz'  '=u' num2str(s) ';'])
    eval(['l' num2str(s)  '=length(zz);'])
end


mm=0;

for s=1:N    % FOR MAX LENGTH
    if eval(['l' num2str(s)])>mm
        mm=eval(['l' num2str(s) ';']);
    end

end
m=mm;

% calculate time axis data
tm=ts*(m-1);
tx=0:ts:tm;

for s=1:N    %TO MAKE EQUAL LENGTH

    if eval(['l' num2str(s) ' <m'])
        eval(['ll' '=l' num2str(s) ';' ]);
        eval(['zz' '=u' num2str(s) ';'  ])
        eval(['uu' num2str(s)  '=zz;'])  ;
        eval(['uu' num2str(s)  '(ll+1:m)=0;']);
    else
        eval(['zz' '=u' num2str(s) ';']);
        eval(['uu' num2str(s)  '=zz;']) ;
    end
end

% disp('after conditioning');
% pause
% sound(uu1)
% sound(uu2)
% sound(uu3)

%% Combine into one matrix
for b=1:N
    eval(['u(' num2str(b) ',:)=uu' num2str(b) ';' ])
end % loop end


% Plotting: input signal
figure(1)

for i=1:N
    subplot(N,1,i)
    plot(tx,u(i,:));
    xlabel('Time(s)');
    ylabel('Amplitude');
    if i==1
        title('User Voice Data')
    end
end

%% Processing
% Input -> [TDM] -> um
um=tdmm(u,ns);

% add noise: use either awgn or rand
umn=awgn(um,25);
% umn=um+rand(1,m);

% Plotting: TDM signal with and w/o noise
figure(2)
subplot(2,1,1)

plot(tx,um);
xlabel('Time(s)');
ylabel('Amplitude');
title('TDM signal')
subplot(2,1,2)

plot(tx,umn);
xlabel('Time(s)');
ylabel('Amplitude');
title('Signal + Noise')


disp('Hit [Enter}]to listen to the muxed sound');
pause
sound(um)
```

```
% Processing
% um -> [Demux] -> ud
ud=demux(um,ns,N);

% split demuxed signals into individual user signal
for b=1:N
    eval(['ux' num2str(b) '=ud(' num2str(b) ',:);' ]);
end % loop end

% Play the demuxed audio signals to observe changes and effect of
% parameters

% disp('Playing the demuxed user voice stream sequentially with a delay of
1second');
%
display('press enter to hear the demuxed signal')
pause
for b=1:N

    eval(['sound(ux' num2str(b) ');' ])
    display('press enter to hear the demuxed signal')
    pause

end % loop end

% Plotting: Demuxed voice signal
figure(3)
for a=1:N
    subplot(N,1,a)
    plot(tx,ud(a,:));
    xlabel('Time(s)');
    ylabel('Amplitude');
    if a==1
        title('Demuxed Signal')
    end
end
```

%functions Used in the code:

%function “tdmm”

```
function [y]=tdmm(x,ns)
if(ns==0)
    ns=1;
else
    ns=round(ns);
end % end if condition

[r c]=size(x); % determine dimension
y=zeros(1,c); % preallocate matrix
i=1; % initialize var i
c1=c-ns; % adjustment for the last data
for j=1:ns:c1
    y(j:j+ns-1)=x(i,j:j+ns-1);
    i=i+1;

    if i>r
```

```
        i=1;
    end % end of if condition
end % end of for loop
end % end of function
```

```
%functions "tdmm"

function y=demux(x,ns,N)

if(ns==0)
    ns=1;
else
    ns=round(ns);
end % end if condition

r=length(x); % length of the data stream
y=zeros(N,r); % preallocation
b=1; % initialize var b
r=r-ns; % adjustment for the last data
for a=1:ns:r
        y(b,a:a+ns-1)=x(a:a+ns-1);
        b=b+1;

        if b>N
            b=1;
        end % end of condition
end % end of for loop
end % end of function
```

2.2 Alternate Matlab Code Used in the Experiment

```
clc
clear all
%signal conditioning and muxing for 3 input

[u1 fs1 b1]=auread('guitar1.au');
[u2 fs2 b2]=auread('speech2.au');
[u3 fs3 b3]=auread('guitar2.au');
N=3;
ts=1/fs1;
Ts = ts; % Slot time interval
ns = Ts/ts;

disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(u1)
disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(u2)
disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(u3)

% disp('Press ENTER to hear the audio signals before conditioning');
%pause
```

```
%sound(u4)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(u5)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(u6)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(u7)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(u8)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(u9)

for s=1:N    %TO find LENGTH
    eval(['zz'  '=u' num2str(s)])
    eval(['l' num2str(s)  '=length(zz)'])
end

% m1=max(l1,l2);
% m=max(m1,l3);
mm=0;

for s=1:N    % FOR MAX LENGTH
    if eval(['l' num2str(s)])>mm
        mm=eval(['l' num2str(s)]);
    end

end
m=mm;

for s=1:N    %TO MAKE EQUAL LENGTH

    if eval(['l' num2str(s) ' <m;'])
        eval(['ll' '=l' num2str(s) ])

        eval(['zz' '=u' num2str(s)  ])
        eval(['uu' num2str(s)  '=zz;'])
        for i=ll+1:m
            eval(['uu' num2str(s)  '(i)=0;'])
        end
    else
        eval(['zz' '=u' num2str(s)  ])
        eval(['uu' num2str(s)  '=zz;'])
    end
end
clc

disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(uu1)
disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(uu2)
disp('Press ENTER to hear the audio signals before conditioning');
pause
sound(uu3)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu4)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu5)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu6)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu7)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu8)
% disp('Press ENTER to hear the audio signals before conditioning');
% sound(uu9)

for b=1:N
    eval(['u(b,:)=uu' num2str(b) ';' ])
end


uuu=TDM_mux2(u,ns);

disp('Hit [Enter}]to listen to the muxed sound');
pause
sound(uuu)

y=demux_TDM2(uuu,N,ns);
```

%functions “TDM_mux2”

```
function [y]=TDM_mux2(x,ns)

i=1;
[r c]=size(x);
k2=1;
k=1;
for j=1:floor(c/ns)

    for kk=1:ns
    y(k)=x(k2,k);
    k=k+1;

    end
    k2=k2+1;

    if k2==r+1
        k2=1;
    end
end
```

%functions “TDM_mux2”

```
function [y]=demux_TDM2(x,N,ns)
L=length(x);
Ls=ceil(L/N);



k=1;
k2=1;
```

```
y=zeros(N,Ls);
for i=1:L


    for kk=1:ns
        y(k2,k)=x(k);
        k=k+1;
        if k==L
            break
        end


    end
    k2=k2+1;

    if k2==N+1
        k2=1;
    end

%      eval(['ud' num2str(i)  '=y(i,:)'])
    if k==L
        break
    end

end

for i=1:N
    eval(['ud' num2str(i)  '=y(i,:)'])
end


for i=1:N
    z=y(i,:);
    disp('Hit [Enter}]to listen to the demuxed signal ');
    pause
    sound(z)
end
```

```
% code to find the frequency of a signal
```

```
[Y,FS,NBITS]=auread('guitar1.au') ;
L = length(Y);

NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(Y,NFFT)/L;
f = FS/2*linspace(0,1,NFFT/2);

% Plot single-sided amplitude spectrum.
figure
plot(f,2*abs(Y(1:NFFT/2)))
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')
```

2 Project Brief

Several audio files to represent user voice conversation. These voice data will be multiplexed using the TDM method to simulate a real time TDM of voice conversation in telephone system. To simulate the effects of transmission medium we will add noise to the multiplexed signals before performing a de-multiplexing on the signal. Finally we will de-mutiplex this noise added signal to simulate a de-multiplexing in telecommunication system on the receiving end. [1]

We will change few of the parameters of input to observe any effect on our signal and the overall TDM process.

We will basically try to answer the following questions:

- What is the effect of noise on the simulation?
- How many signals can be multiplexed and de-mutiplexed with satisfactory accuracy?
- What are the parameters that affect the accuracy?
- How can we increase the efficiency of the transfer to ensure maximum utilization of any medium?

3 Project Work

3.1 Voice data input

First, we will take few audio files as voice data input, which are to be multiplexed later.
We will start with two voice data files for our initial simulation.

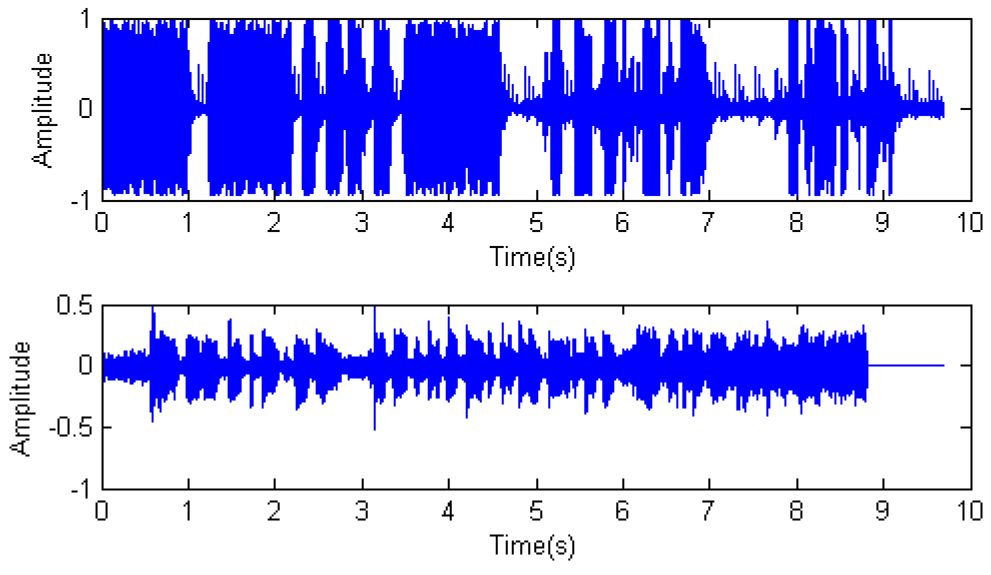

Figure 02: Two voice data

3.2 TDM

We will use these two-voice data to perform TDM. Our TDM system will assume two user conversations and perform the TDM accordingly.

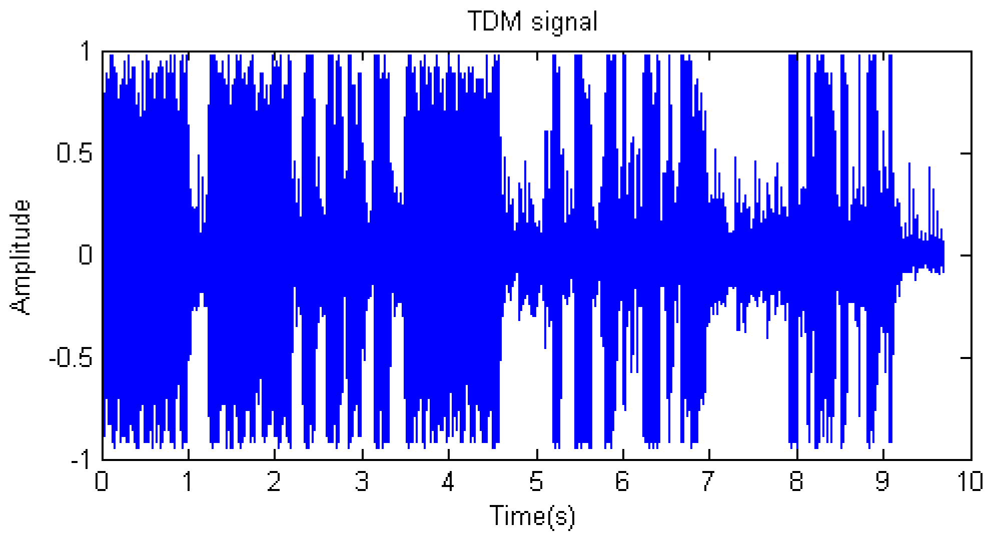

Figure 03: TDM signal for two user voice data

3.3 Noise

We added noise with the TDM signal to simulate a transmission through a medium.

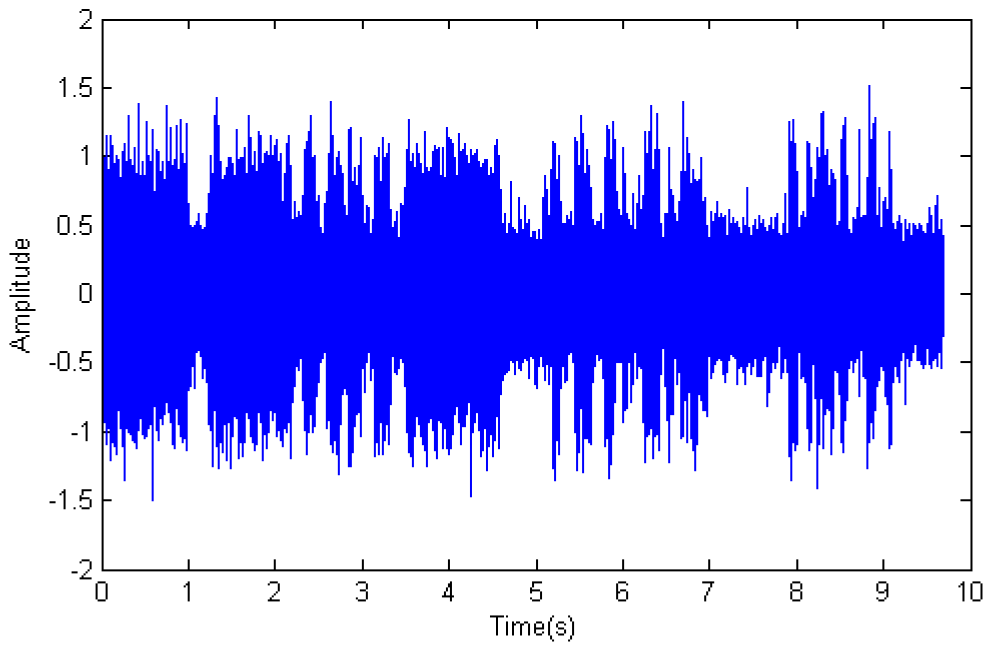

Figure 04: TDM signal with noise

From the signal in Figure 04, we see that the density of signal at certain points has increased as well as the amplitude. These can happen due to the repeaters on the line which amplifies the weakened signals through the transmission along with the noise in the signal.

3.4 De-multiplexing

We sent this signal to a de-multiplexing function to retrieve the final signal that the receivers will be getting at their end.

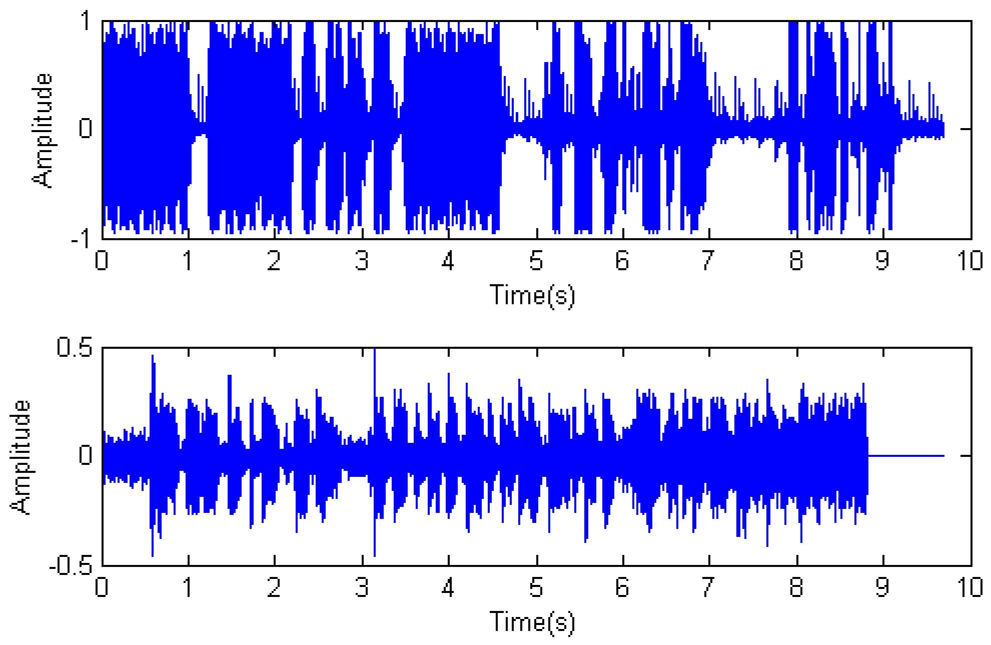

Figure 05: Two de-multiplexed signal at the receiving end

We see that there is a slight difference between the initial signals from the transmission end the de-multiplexed signal at receiving. Due the fast switching rate and small time slot the overall loss is not significantly visible here.

4 Experiment

To answer our questions stated previously we repeated the experiment with different parameters. First of all, we observed the effect of noise on the TDM signal.

a) What is the effect of noise on the simulation?

To understand the effect(s) of noise on simulation we repeated the experiment with different parameters each time. The signal usually showed two common effects which we can observe from the figures provided below:

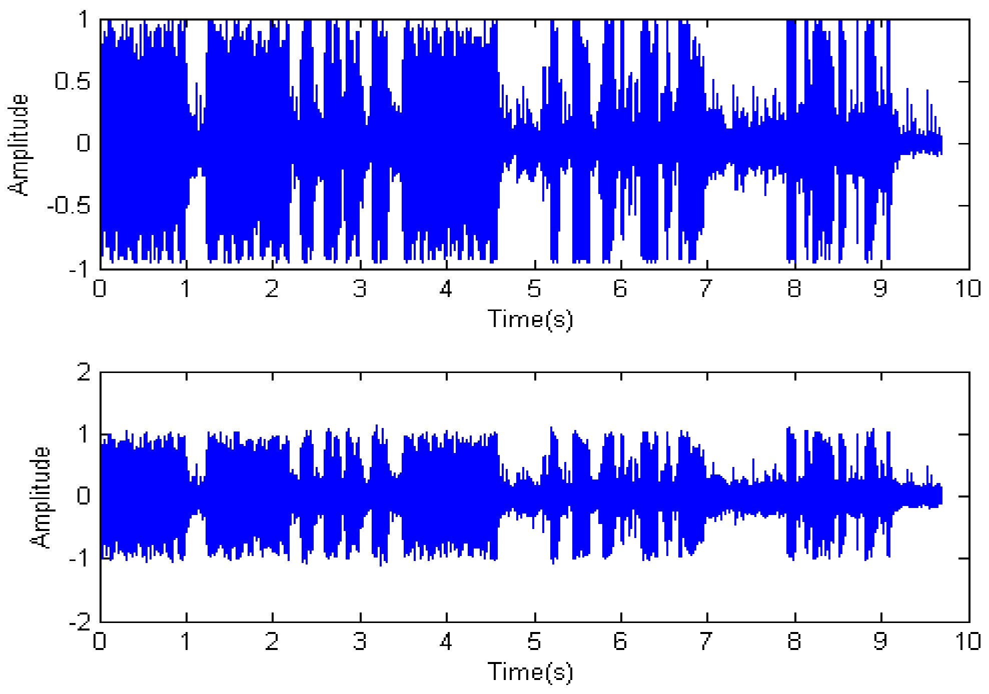

Figure 06: TDM(top), and TDM with Noise(bottom)

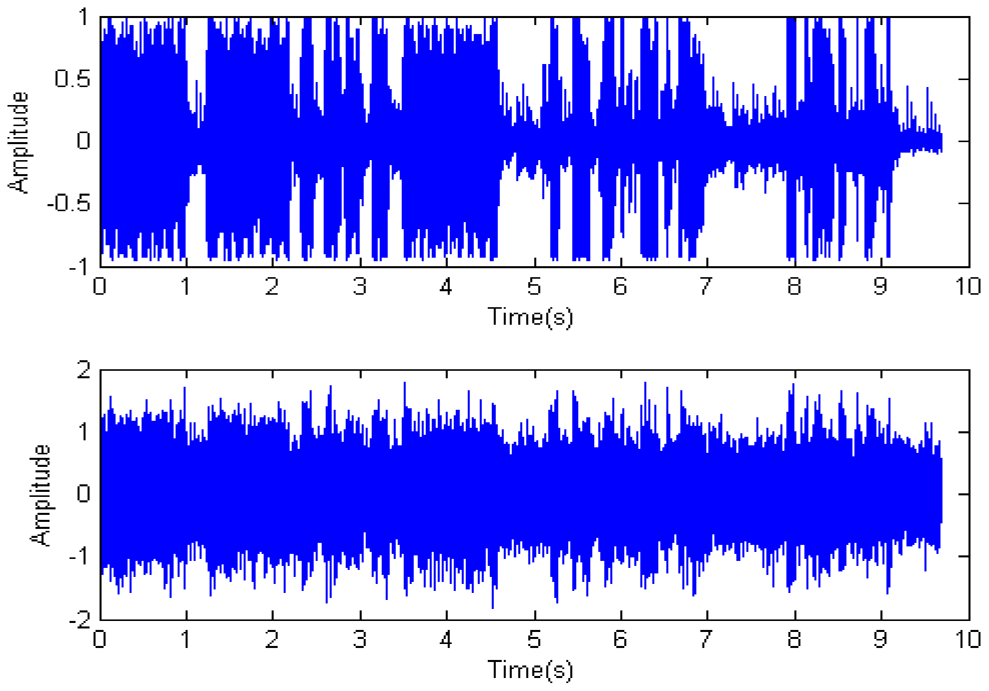

Figure 07: TDM(top), and TDM with Noise (bottom)

From the Figure 06, we have applied a small signal to noise ratio which had the effect of increasing the signal amplitude, and in Figure 07, we used a larger signal to noise ratio which shows a severely distorted signal with a lot of extra information which renders the signal almost unusable unless we use proper filters at the receiving end. The common effects we observed are:

- Change in signal amplitude
- Change in information being transmitted

b) How many signals can be multiplexed and demultiplexed with satisfactory result?

We started with 2 signals as shown in the figure below:

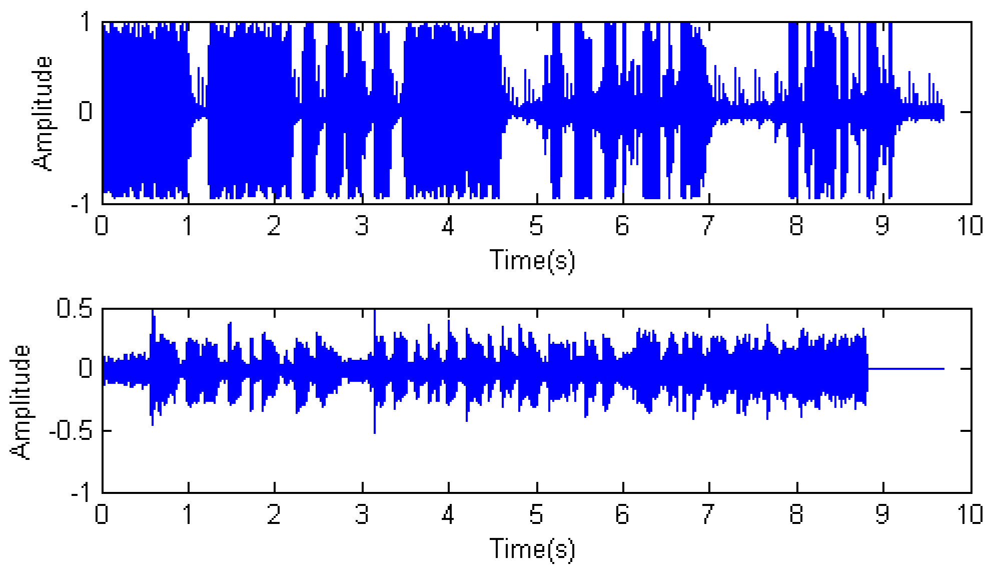

Figure 08: Two voice data

We increased the signal gradually to 4 voice data:

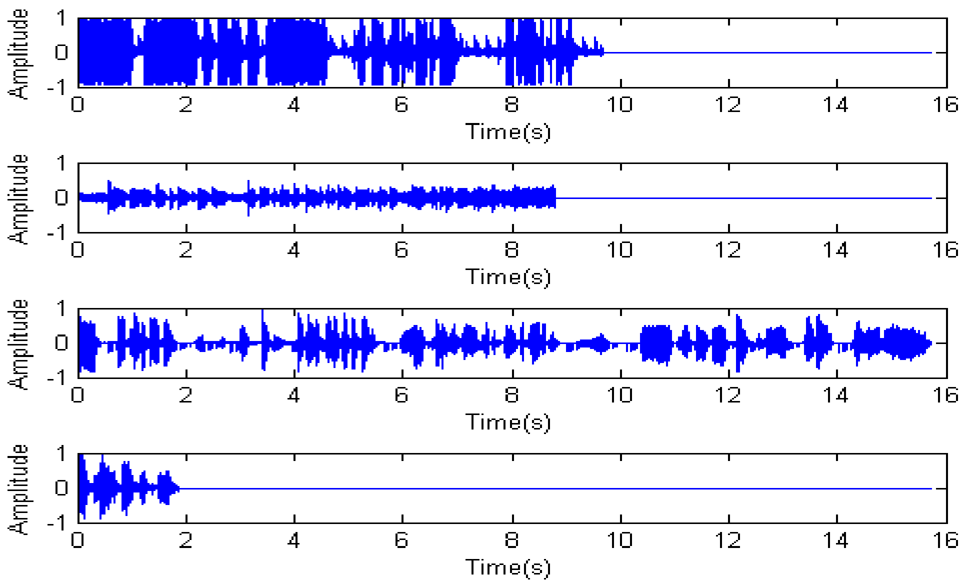

Figure 09: Four user voice data

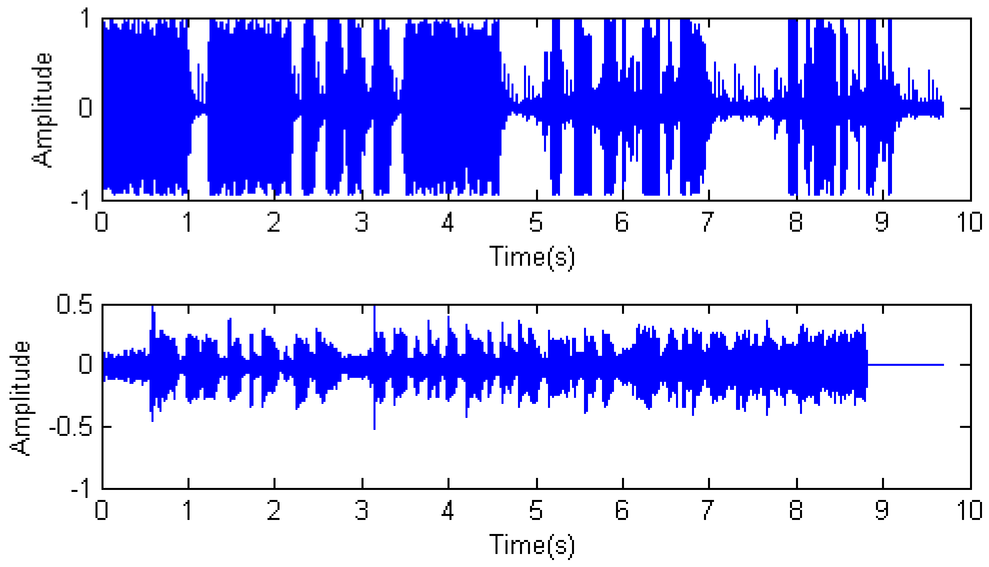

Figure10: TDM(top), TDM with Noise(bottom)

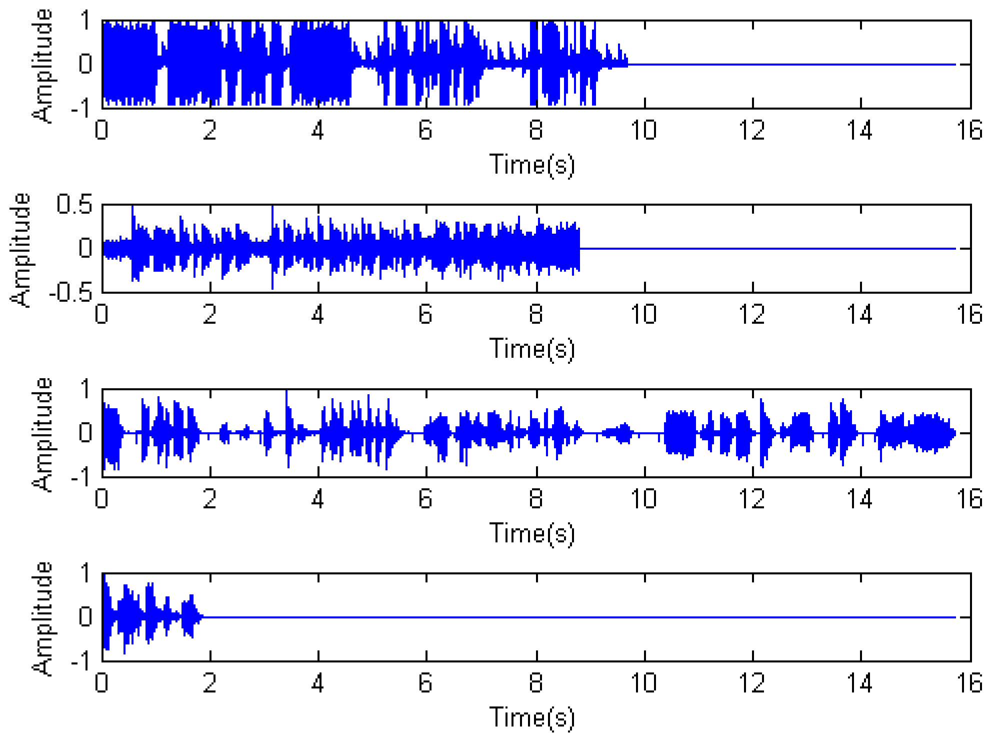

Figure11: Demultiplexed voice data at receiving end

We performed a TDM of 7 voice data properly, and theoretically we can perform TDM for few hundreds or more voice data. Since this is a simulation we are not facing the practical limitation of the electrical switching circuit and other limitation. Our simulation is limited by the computing power only.

c) **What are the parameters that affect the accuracy?**

We repeated the experiment changing several parameters of the TDM such as

- Time slot length
- Frequency of input signal
- Amplitude of the input signal
- Number of input signals (voice data)

The time slot length did not affect the TDM process but increasing the length resulted in much more data loss and increment of delay at the receiving end.

Frequency of the input varied to a wide region due to usage of voice data. We observed some slight changes in the TDM signal.

Amplitude of the input signal did not show any change in the TDM process.

Number of input signals showed similar effect at the receiving end, when the number of users increased to larger extent the delay is noticeable and there is larger data loss.

d) **How can we increase the efficiency of the transfer medium to ensure maximum utilization of any medium?**

To increase the efficiency of the transfer medium we need to observe certain characteristics of the transmission medium. We need to check the attenuation, delay, data rate, bandwidth, repeater spacing, cross talk etc.

We have to make sure that the bit rate is large enough to handle the average desired number of users' load. The repeater length should be longer to avoid the noise effects as voice data can severely deteriorate due to noise. Since we are using a single medium or a shared medium to transmit we don't have to consider the cross-talk characteristics but we should be aware of the EMI of the wire to avoid unwanted changes in the transmission signal.

5 Conclusion

We observed that TDM saves the overall transmission expense by sharing a single medium as well as sharing small number of channels to several users. But TDM method has some major draw backs as well.

Depending on the switching speed of the circuit and other limitation TDM can have large data loss as well as delay. The electrical switch might not be switching properly, and a slight error or delay in switching properly in one end will cause a large delay on the other end as well as sending the wrong information to the wrong user is the both end (transmission and reception) switching is slightly out of synchronization. The saving of resource in one end setbacks the delay and data loss on the other end. TDM can be suitable if the communication is slow and limited to small number of users but for a huge number of users this is not the best choice.