Transmisión de datos mediante TDM

Universidad Nacional de San Antonio Abad del Cusco
Escuela profesional de Ingeniería Electrónica
Telecomunicaciones II

Alexander Palomino Lopez
Ingeniero Electrónico
Cusco, Perú
alexander.palomino@unsaac.edu.pe

Davis Bremdow Salazar Roa - 200353
Estudiante de Ingeniería Electrónica
Cusco, Perú
200353@unsaac.edu.pe

***Abstract*—Con la creciente demanda de recursos en espectro y físicos para la transmisión de datos explorar técnicas que permitan mitigar este creciente cambio es de vital importancia para la sostenibilidad en las comunicaciones y la implementación de sistemas de orden superior que integren en el mismo un tráfico de datos elevados sin que esto considere para su implementación un exceso uso de recursos. Una de las técnicas de optimización se centra en multiplexar las diferentes fuentes de información en intervalos de tiempo especifico para transmitir un bloque o slot de datos para luego en la etapa del receptor aplicar el proceso inverso (demux) para la recuperación de las señales.**

***Index Terms*—Transmisión de datos, Ancho de banda, PCM, TDM, sincronización de canal, Multiplexación de información**

I. Introducción

La Multiplexación por División de Tiempo (TDM, por sus siglas en inglés) es una técnica fundamental en los sistemas de telecomunicaciones modernos, ya que permite la transmisión eficiente de múltiples señales digitales o analógicas a través de un mismo canal físico. Su importancia radica en la optimización del uso del ancho de banda disponible, reduciendo costos de infraestructura y aumentando la capacidad de los sistemas de comunicación.

El principio de la TDM se basa en asignar intervalos de tiempo específicos a cada señal dentro de un marco de transmisión, garantizando que todas compartan el mismo medio sin interferencias. Esta característica la convierte en una herramienta esencial en redes de telefonía, sistemas satelitales, transmisión de datos en fibra óptica y en aplicaciones de comunicación digital como la televisión digital y los servicios de internet de alta velocidad.

Gracias a la TDM, es posible integrar diferentes tipos de información —voz, video y datos— en un mismo canal de forma organizada, manteniendo la calidad del servicio y facilitando la escalabilidad de las redes. En consecuencia, la TDM no solo representa una solución técnica eficiente, sino también una base clave para el desarrollo de sistemas de comunicación modernos que requieren alta velocidad, confiabilidad y capacidad de integración.

II. Esquema general

La Multiplexación por División de Tiempo permite la optimización en la cantidad de canales a usar para la trasmisión de datos digitales optimizando así el ancho de banda necesario para transmitir información de diferentes fuentes, en la figura 1 se muestra un esquema general para la transmisión de datos mediante este esquema.

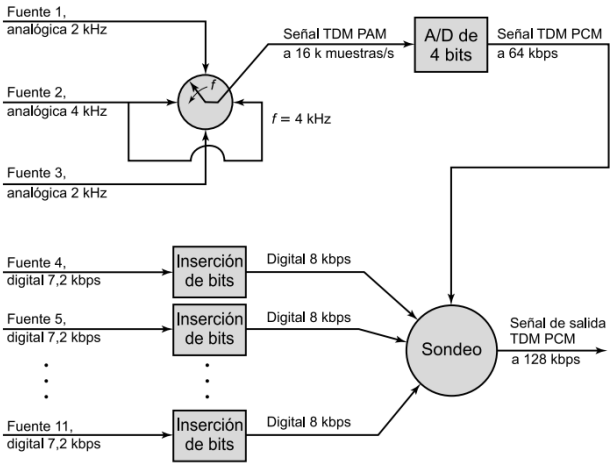

Fig. 1: Diagrama de bloques - TDM

Y en la cual se pueden apreciar 6 canales siendo 3 ellos de origen analógico y los restantes digitales, destacando que la transmisión se realiza de forma digital, en esta primera etapa es necesario transformar cada señal continua en una PCM ó codificación de los pulsos discretos haciendo uso de la frecuencia de muestreo adecuada para cada canal (para definir la frecuencia el teorema de Nyquist puede servir como punto de partida), con cada canal ya digitalizado el bloque de **SONDEO** [1] se encarga de realizar la mezcla de todas las fuente de información para su transmisión dividiendo este resultado en intervalos de tiempo correspondiente a cada fuente mientras que en el lado del receptor para recuperar cada señal se realiza el proceso inverso.

III. Sincronización

La optimización de un canal para el transmisión de información requiere ciertos parámetros de control entre los cuales se puede destacar por su importancia la señal de conmutación o de sincronización la cual es la encargada de generar los bloques de información o slots conformados a partir de las diferentes fuentes que

conforman el bloque general TDM que consiste principalmente en la generación de paquetes de información espaciados temporalmente y reconstruidos en función al mismo principio de división en tiempo.

Por lo tanto al requerirse en ambas etapa (envío y recepción) es necesario considerar que la señal de sincronización que actúa al mismo tiempo de frecuencia de muestreo (en caso de contar con señales analógicas) debe ser la misma en ambos extremos de un enlace de comunicación digital por lo tanto para su despliegue se desarrollando 2 formas de poder **sincronizar** este recurso tan relevante como se menciona en [2] y se ejemplifica de forma gráfica en la figura 2

1) Sincronización mediante canal externo
2) Sincronización por dígito agregado

Siendo para el presente caso el empleo del primer método, en la cual se hace de un canal externo para sincronizar las señales de conmutación en ambos extremos del enlace, permitiendo recuperar de forma efectiva las señales multiplexadas en el tiempo requiriendo sin embargo una canal adicional para sincronizar correctamente la frecuencia de muestreo desde el emisor en el receptor menguando el propósito de una comunicación TDM que busca reducir el uso de canales para la transmisión de información ubicándose en las aplicaciones práctica el empleo del segundo método en la cual se hace uso de una cantidad de bits en la parte final de una trama para la sincronización de la señales.

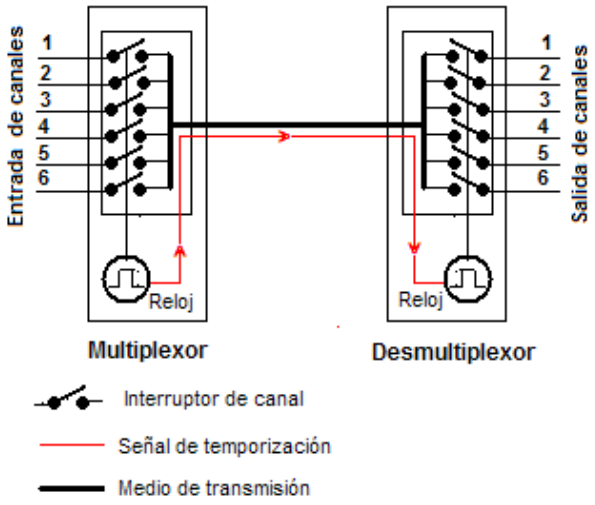

Fig. 2: Sincronización en la señal de multiplexación en una TDM

IV. Simulación

Para la emulación de una señal TDM el proceso guía estuvo conformado por varías etapas entre las cuales se destacan:

1) Elección del software a implementar
2) Generación de las señales o información para la transmisión
3) El multiplexado por división de tiempo
4) El proceso inverso y/o recuperación de la señal

A. *Lenguaje de programación*

Durante la experiencia para la emulación de una transmisión mediante TDM se consideraron en primera instancia las diferentes herramientas de programación existentes ubicándose entre todas ellas por ejemplo

1) Python
2) Javascript
3) MATLAB

Cabe mencionar que el presente experimento al tratarse de una simulación a nivel digital cualquier lenguaje de programación es apto para su desarrollo, sin embargo lenguajes como Python o MATLAB brinda cierta facilidad con herramientas integradas para la generación de interfaces gráficas o figuras para mostrar los resultados así como librerías que facilitan el grabado y reproducción de audio en tiempo real.

Dentro de las virtudes observadas para cada lenguaje de programación listado se decidió finalmente por MATLAB por su integración de la librería *audiorecord* y su fortaleza para la manipulación de matrices fila, columna o multidimensionales otorgando una gran ventaja adicional gracias a su IDE (figura 3) o entorno de programación que permite administrar las variables creadas así como sus dimensiones

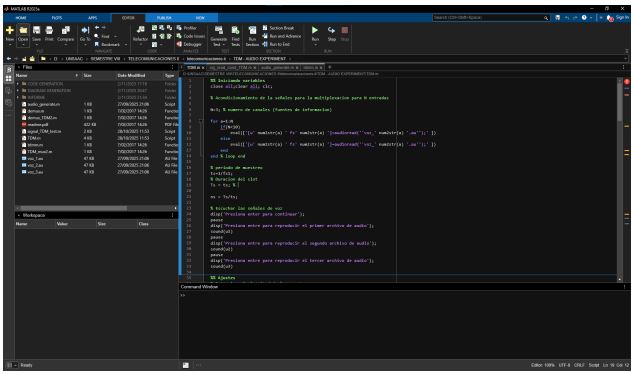

Fig. 3: Interfaz gráfica de MATLAB

B. *Generación de las señales*

En esta segunda etapa del desarrollo emulado se considero un archivo **.m** en el cual se hace uso del objeto `audiorecorder` para la generación de archivos de audio con una duración de 3 segundos cada uno para limitar la cantidad de muestras generadas potenciando esta función mediante variables de estado registrando eficazmente según lo requerido.

El objeto *audiorecorder* por defecto cuenta con una frecuencia de muestreo 8K [Hz] obteniendo así por cada señal de audio 24000 muestras o datos digitales las cuales se muestran en la figura 4.

Además en esta misma se puede apreciar la duración y amplitud siendo la más característica la señal 3 que cuenta con un mayor rango de valores discretizados, viéndose reflejado este proceso en el bloque de código de código 1

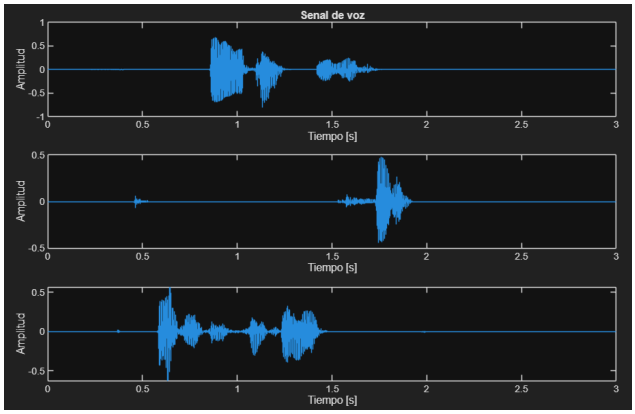

Fig. 4: Señales de voz de entrada

C. *Organización de archivos para la simulación*

El desarrollo de la simulación cuenta con diferentes etapas de multiplexación desde la lectura de las fuentes de información y los procesos inversos correspondientes estableciendo así varias operaciones repetitivas las cuales se pueden optimizar mediante el empleo de funciones estableciendo de esta forma una jerarquía de archivos dentro del esquema de emulación, considerando en el esquema mostrado en la figura 5

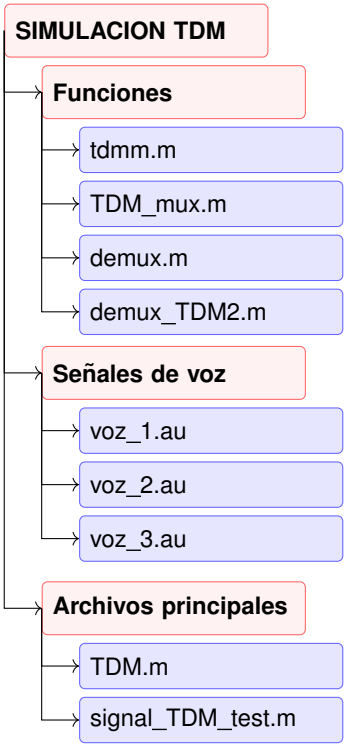

Fig. 5: Jerarquía de archivos - Simulación TDM

El cual representa la estructura de trabajo para facilitar la comprensión del mismo para el llamado de funciones.

D. *Multiplexación señal*

Para el proceso de multiplexación el flujo del algoritmo a considerar a nivel de secuencia de pasos se describe en el diagrama 6 utilizando para cada bloque los archivos de función en MATLAB para la ejecución de cada uno.

En el TDM la multiplexación de la señal se considera como el elemento crítico para su implementación y que consiste de manera técnica en la fusión de las diferentes fuentes asignando un intervalo de tiempo para cada una de ellas en función a una señal de conmutación y sincronización mencionadas previamente.

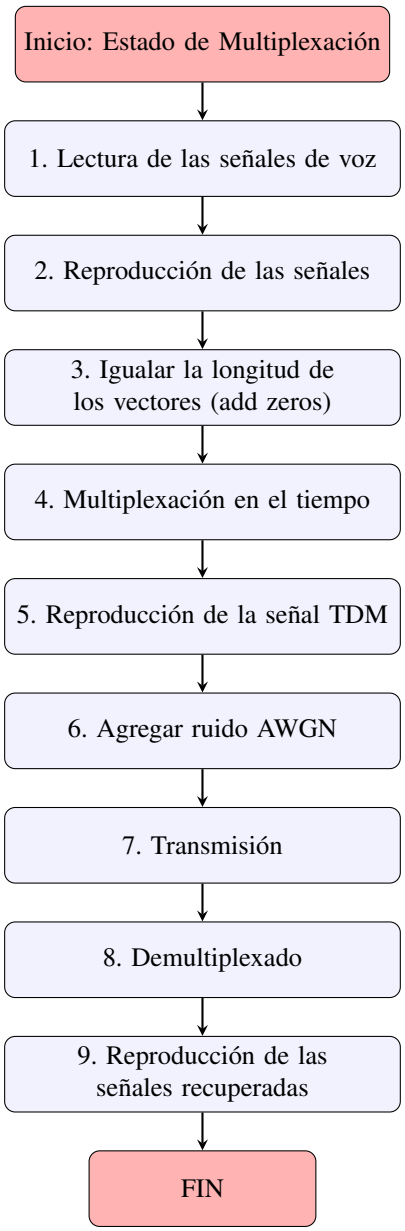

Fig. 6: Diagrama de flujo del estado de Multiplexación de la señal.

V. Resultados

Una vez realizado el proceso de multiplexación para las diferentes fuentes mostradas en la figura 4 como salida de la simulación realizada se observan una figura intermedia y otra de salida como resultado siendo 7 las señales de audio considerando el agregado de **ruido AWGN** con la finalidad de ver los efectos de este fenómeno no deseado en este tipo de multiplexación y el cual se muestra en la figura 8 la señal de salida resultante demultiplexada al aplicar el proceso inverso para la recuperación de la señal.

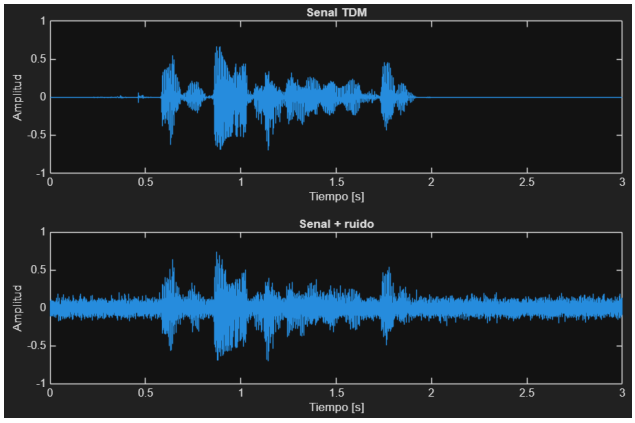

Fig. 7: Señal TDM - Señal TDM con ruido (AWGN agregado)

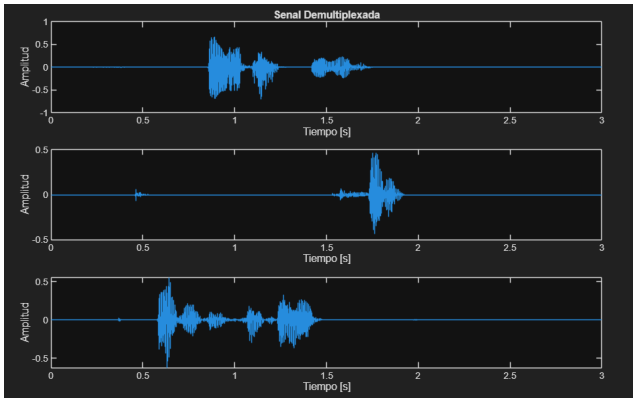

Fig. 8: Señal TDM + Noise demultiplexada

VI. Conclusiones

- Durante la simulación se pudo generar y transmitir diferentes fuentes de información partiendo desde su digitalización pasando por el proceso de multiplexación en el tiempo y/o TDM para finalizar con el proceso inverso para recuperar las señales analizando dentro de este proceso el agregado de ruido aleatorio, siendo poco perceptible o con poca influencia al momento de reproducir el audio grabado.
- Además también en la ejecución del algoritmo propuesto se pudo validar la definición teórica mediante la puesta en práctica del modelo TDM considerando tres fuentes de audio haciendo uso para esto el software de MATLAB y una sincronización de canal externo para cada etapa del proceso, expandiendo de esta forma el análisis mediante su aplicación directa.
- Finalmente como consideración en la TDM se manipularon diferentes etapas de trabajo a nivel de software, sin embargo el experimento provisto no es extendible y fue ejecutado a nivel de implementación y para lo cual se puede hacer uso de micro controlador como Arduino siendo un uso de aplicación o ejemplo el envío o transmisión de medidas a partir de varios sensores como se menciona en [3] y el cual se puede usar de guía para otra posible aplicación.

References

[1] W. Stallings, *Comunicaciones y Redes de Computadores*, 7th ed. Madrid, España: Pearson Prentice Hall, 2004, traducción de la 7ª edición en inglés.

[2] L. W. Couch, *Sistemas de comunicación digitales y analógicos*, 7th ed. México: Pearson Educación, 2012.

[3] Rajkumar2506, "Interfacing sensors with arduino using time division multiplexing," Instructables, 2025, accessed: 2025-11-13. [Online]. Available: https://www.instructables.com/Interfacing-Sensors-With-Arduino-Using-TIME-DIVISI/

Generación de audio

```
clc;
state = 1;
% Solo realiza la grabación cuando el
↪  estado es 0
if state == 1
clear;
% Objeto de grabacion
recObj = audiorecorder;

% Iniciar una grabacion (solo es
↪  necesario ejecutarla una vez por
↪  audio)
disp("Iniciando con la grabacion");
recordblocking(recObj, 3);
end

% Reproducir el audio grabado
play(recObj)
y = getaudiodata(recObj);

save_voice = 1;

if save_voice == 1
% guardar la señal de voz
fs = 8000;
audiowrite('voz_3.wav', y, fs);
end
```

Listing 1: Generación de audio

Simulación TDM - I

```
%% Iniciando variables
close all;clear all; clc;
N=3;
for a=1:N
if(N<10)
eval(['[u' num2str(a) ' fs' num2str(a)
↪  ']=audioread(''voz_' num2str(a)
↪  '.au'');' ])
else
eval(['[u' num2str(a) ' fs' num2str(a)
↪  ']=audioread(''voz_' num2str(a)
↪  '.au'');' ])
end
end % loop end

% periodo de muestreo
ts=1/fs1;
```

Listing 2: Simulación TDM

Simulación TDM - II

```
% Duracion del slot
Ts = ts; %

ns = Ts/ts;
% Escuchar las señales de voz
disp('Presiona enter para continuar');
pause
disp('Press enter to play the firsts
↪  audio');
sound(u1)
pause
disp('Press enter to play the second
↪  audio');
pause
disp('Press enter to play the third
↪  audio');
sound(u3)

%% Ajustes
% Determinar la longitud de las
↪  muestras
for s=1:N
eval(['zz'  '=u' num2str(s) ';'])
eval(['l' num2str(s)  '=length(zz);'])
end

mm=0;

% Determinar la máxima longitud
for s=1:N
if eval(['l' num2str(s)])>mm
mm=eval(['l' num2str(s) ';']);
end
end

m=mm;

% Estimacion de tiempo
tm=ts*(m-1);
tx=0:ts:tm;

% Igualando la longitud de los vectores
for s=1:N
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
```

Simulación TDM III

```
%% Multiplexando (union en una matriz)
for b=1:N
eval(['u(' num2str(b) ',:)=uu'
↪  num2str(b) ';' ])
end

% Graficando las senales
figure(1)

for i=1:N
subplot(N,1,i)
stairs(tx,u(i,:));
xlabel('Tiempo [s]');
ylabel('Amplitud');
if i==1
title('Senal de voz')
end
end

%% Procesamiento
% Multiplexando la senal

um=tdmm(u,ns);
% Agregando AWGN (ruido gaussiano
↪  aditivo aleatorio)
umn=awgn(um,25);

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

stairs(tx,um);
xlabel('Tiempo [s]');
ylabel('Amplitud');
title('Senal TDM')

subplot(2,1,2)
stairs(tx,umn);
xlabel('Tiempo [s]');
ylabel('Amplitud');
title('Senal + ruido')

disp('Presiona enter para escuchar la
↪  señal multiplexada: ');
pause
sound(um)

% Procesando (operacion inversa) -
↪  Demultiplexando
% um -> [Demux] -> ud
ud=demux(um,ns,N);

% Dividiendo las senales
↪  demultiplexadas
for b=1:N
eval(['ux' num2str(b) '=ud(' num2str(b)
↪  ',:);' ]);
end
```

Simulación TDM - IV

```
% Reproduciendo las senales
↪  demultiplexadas
display('Presiona enter para escuchar
↪  las senales demultiplexadas')
pause
for b=1:N

eval(['sound(ux' num2str(b) ');' ])
display('Presionar enter para
↪  reproducir la senal
↪  demultiplexada')
pause

end % loop end

% Graficando las senales
↪  demultiplexadas
figure(3)
for a=1:N
    subplot(N,1,a)
    stairs(tx,ud(a,:));
    xlabel('Tiempo [s]');
    ylabel('Amplitud');
    if a==1
        title('Senal Demultiplexada')
    end
end
```