Simulación en MATLAB de Códigos de Línea y Comparación de sus PSD

Nombre: Santiago Valencia García. Código: A00395902.

Adapted from: https://drmoazzam.com/matlab-simulation-of-line-codes-and-their-psd

Para transmitir los bits sobre un canal físico, deben transformarse en una forma de onda física. Un codificador de línea o transmisor binario en banda base convierte una secuencia de bits en una forma de onda física adecuada para la transmisión a través de un canal. Los codificadores de línea utilizan el término "marca" para indicar un “1” y "espacio" para indicar “0”. En sistemas de banda base, los datos binarios pueden transmitirse utilizando varios tipos de pulsos. Estos diferentes tipos de formas de onda están diseñados para satisfacer distintos criterios de rendimiento en diversos escenarios. Cada tipo de código de línea tiene sus propias ventajas y desventajas. La elección de la forma de onda de codificación de línea depende de las características operativas de un sistema, tales como:

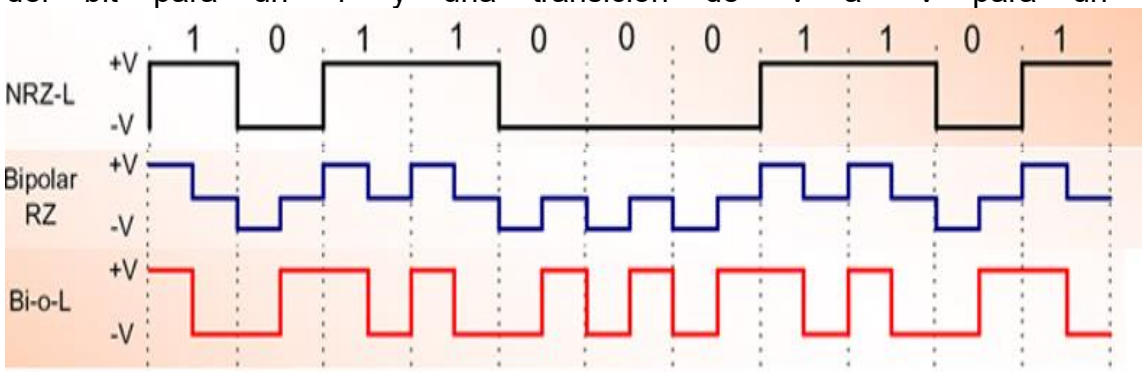
Requerimiento de ancho de banda: Los códigos de línea como NRZ-L requieren menos ancho de banda, pero tienen la desventaja de tener un nivel de DC y una mala sincronización.

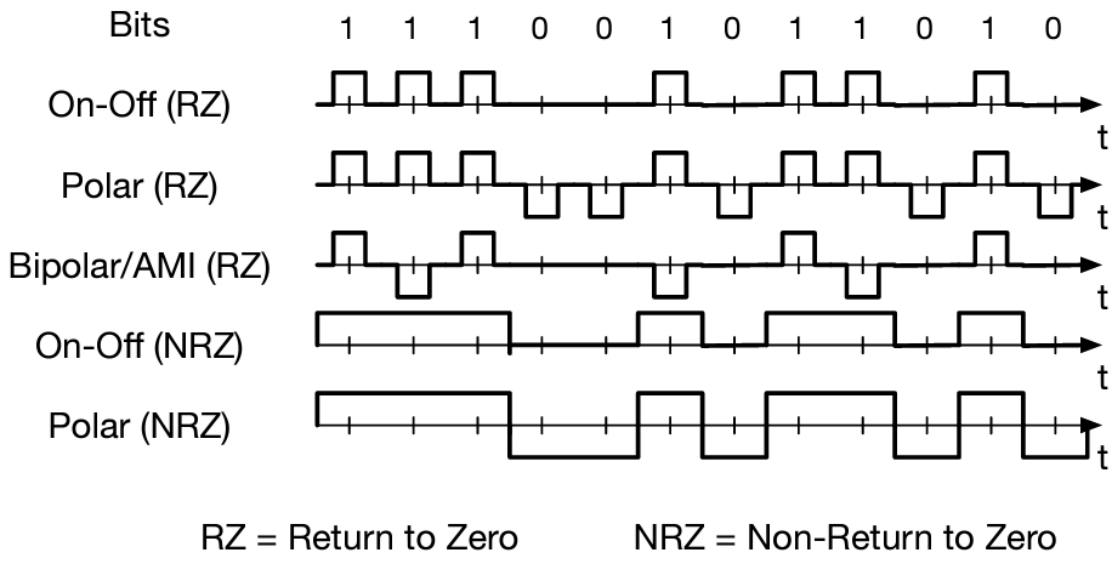
Requerimiento de sincronización: Un esquema de codificación de línea como Manchester tiene buena sincronización, pero requiere un ancho de banda significativamente mayor.

Complejidad del receptor: Los esquemas complejos obviamente requieren un receptor más complejo, lo que aumenta el costo.

La entrada del codificador de línea es la secuencia de bits digitales que se transmitirá por el canal, y la salida es una secuencia de formas de onda en función de los bits de datos de entrada. Aquí vamos a presentar la simulación en MATLAB de los esquemas On-Off NRZ, Bipolar NRZ-L, Bipolar-RZ y Manchester-L. En la siguiente figura, podemos ver cómo una secuencia de bits ‘10110001101’ es codificada en formas de onda físicas por cada uno de estos esquemas. Para NRZ-L, un ‘1’ se representa como voltaje positivo +V durante toda la duración del bit, mientras que un ‘0’ se representa como voltaje negativo -V durante toda la duración del bit. En comparación, para Bipolar-RZ, vemos que, en el medio del bit, +V y -V regresan a cero para ‘1’ y ‘0’, respectivamente. Bi-phase-L es otro nombre para el esquema Manchester-L, donde observamos una transición de +V a -V en el medio

del bit para un ‘1’ y una transición de -V a +V para un ‘0’.





RZ = Return to Zero NRZ = Non-Return to Zero

En comunicaciones digitales, la Densidad Espectral de Potencia (PSD, por sus siglas en inglés) es un concepto fundamental para el análisis y diseño de sistemas de transmisión. La PSD describe cómo se distribuye la potencia de una señal en función de la frecuencia, proporcionando una representación del contenido de frecuencia de las formas de onda utilizadas en la transmisión de datos. Conocer la PSD es crucial, ya que permite evaluar el ancho de banda necesario para transmitir una señal y prever posibles interferencias en el canal. Además, la PSD ayuda a determinar la eficiencia espectral de diferentes esquemas de codificación de línea y permite identificar la presencia de componentes de baja frecuencia (DC) que podrían afectar la estabilidad y sincronización de la señal en sistemas de banda base.

Para simular estas formas de onda en MATLAB y comparar sus Densidades Espectrales de Potencia (PSD), necesitamos representar estas formas de onda como valores o muestras discretas. En nuestra simulación, estamos representando la forma de onda correspondiente a cada bit mediante 10 muestras. Por ejemplo, para el código Bipolar RZ-L, si el bit de entrada es ‘1’, generaremos la forma de

onda muestreada “V V V V V 0 0 0 0 0” en la salida, y en el caso de que el bit sea ‘0’, la salida será “-V -V -V -V -V 0 0 0 0 0”.

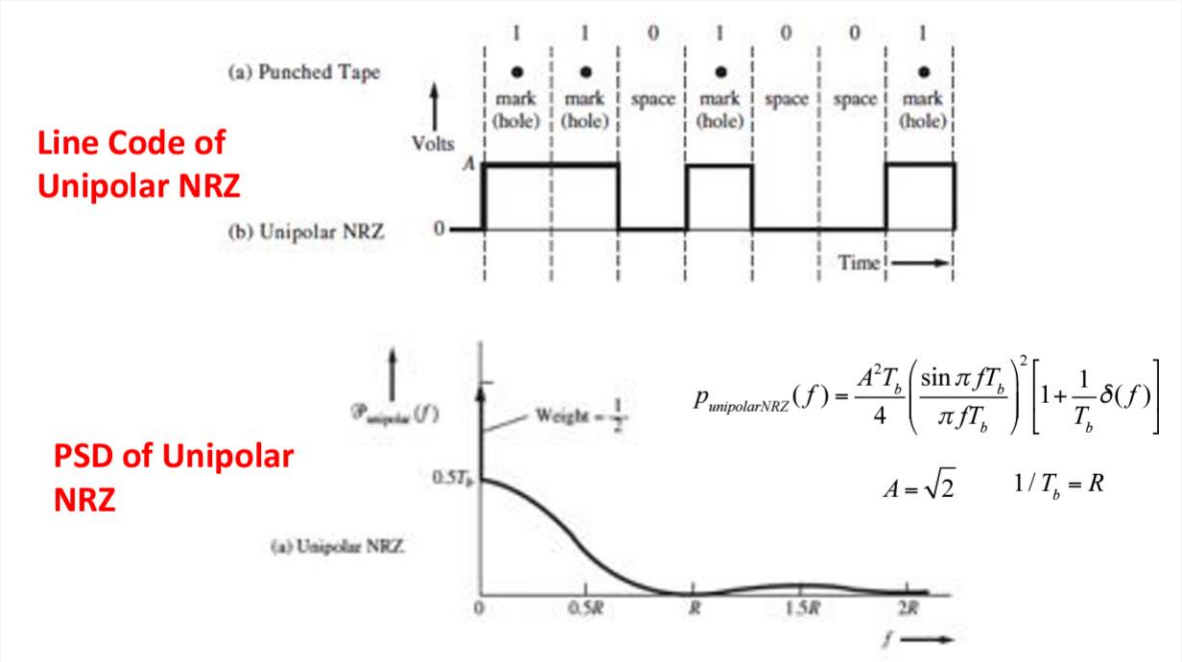
A continuación se presenta el código fuente completo:

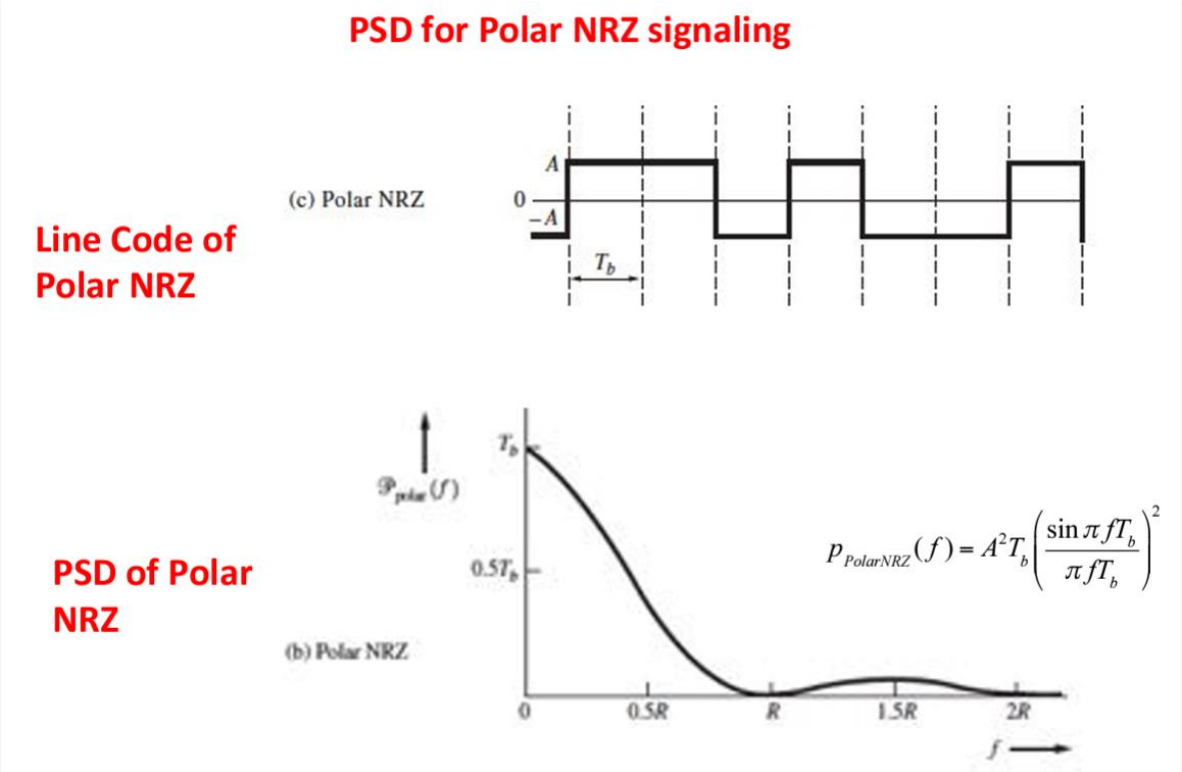
```
clear all;
close all;
%Nb is the number of bits to be transmitted
Nb=10000;
% Generate Nb bits randomly
b=rand(1,Nb)>0.5;
%Rb is the bit rate in bits/second
Rb=4;
%Since each waveform is represented by 10 samples so sampling
%frequency is 10 times the bit rate
fs=10*Rb;
NRZ_out=[];
RZ_out=[];
Manchester_out=[];
NRZno_polar=[];
%Vp is the peak voltage +v of the waveform
Vp=5;
%Here we encode input bitstream as On-Off (NRZ) waveform
for index=1:size(b,2)
 if b(index)==1
 NRZno_polar=[NRZno_polar [1 1 1 1 1 1 1 1 1 1]*Vp];
 elseif b(index)==0
 NRZno_polar=[NRZno_polar [0 0 0 0 0 0 0 0 0 0]*0];
 end
end
%Now we draw the PSD spectrum of On-Off (NRZ) using Welch PSD
%estimation method
h = spectrum.welch;
Hpsd=psd(h,NRZno_polar,'Fs',fs);
figure;
hold on;
handle1=plot(Hpsd);
set(handle1,'LineWidth',2.5,'Color','g')
%Here we encode input bitstream as Bipolar NRZ-L waveform
for index=1:size(b,2)
 if b(index)==1
 NRZ_out=[NRZ_out [1 1 1 1 1 1 1 1 1 1]*Vp];
```

```
 elseif b(index)==0
 NRZ_out=[NRZ_out [1 1 1 1 1 1 1 1 1 1]*(-Vp)];
 end
end
%Now we draw the PSD spectrum of Bipolar NRZ-L using Welch PSD
%estimation method
h = spectrum.welch;
Hpsd=psd(h,NRZ_out,'Fs',fs);
handle2=plot(Hpsd)
set(handle2,'LineWidth',2.5,'Color','r')

%Here we encode input bitstream as Bipolar RZ waveform
for index=1:size(b,2)
 if b(index)==1
 RZ_out=[RZ_out [1 1 1 1 1 0 0 0 0 0]*Vp];
 elseif b(index)==0
 RZ_out=[RZ_out [1 1 1 1 1 0 0 0 0 0]*(-Vp)];
 end
end

%Now we draw the PSD spectrum of Bipolar RZ using Welch PSD
%estimation method
h = spectrum.welch;
Hpsd=psd(h,RZ_out,'Fs',fs);
handle3=plot(Hpsd);
set(handle3,'LineWidth',2.5,'Color','b')
 %Here we encode input bitstream as Manchester-L waveform
for index=1:size(b,2)
 if b(index)==1
 Manchester_out=[Manchester_out [1 1 1 1 1 -1 -1 -1 -1 -1]*Vp];
 elseif b(index)==0
 Manchester_out=[Manchester_out [1 1 1 1 1 -1 -1 -1 -1 -1]*(-Vp)];
 end
end
 %Now we draw the PSD spectrum of Manchester-L using Welch PSD
%estimation method
h = spectrum.welch;
Hpsd=psd(h,Manchester_out,'Fs',fs);
handle4=plot(Hpsd)
set(handle4,'LineWidth',2.5,'Color','k')
legend('On-Off (NRZ)','Bipolar NRZ-L','Bipolar-RZ','Manchester-L');
hold off;

figure;
plot((1:Nb*10)/10,NRZno_polar);
ylim([-Vp*1.1  Vp*1.1 ])
xlabel('bits-->');
ylabel('Amplitude (volts)-->');
```

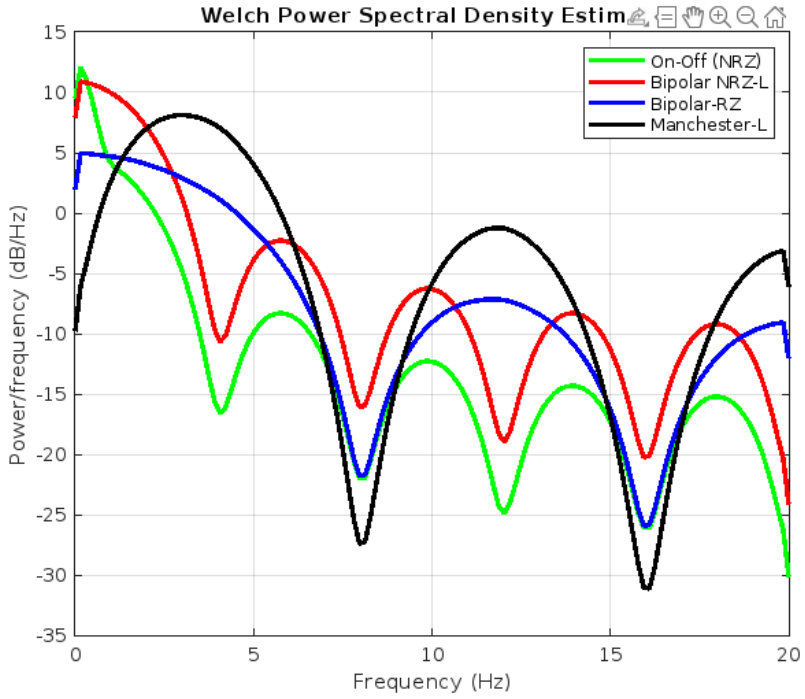
```
title('ON OFF NRZ encoded bit stream');
figure;
plot((1:Nb*10)/10,NRZ_out);
ylim([-Vp*1.1  Vp*1.1 ])
xlabel('bits-->');
ylabel('Amplitude (volts)-->');
title('Bipolar NRZ-L encoded bit stream');
figure;
plot((1:Nb*10)/10,RZ_out);
ylim([-Vp*1.1  Vp*1.1 ])
xlabel('bits-->');
ylabel('Amplitude (volts)-->');
title('RZ-L encoded bit stream');
figure;
plot((1:Nb*10)/10,Manchester_out);
ylim([-Vp*1.1  Vp*1.1 ])
xlabel('bits-->');
ylabel('Amplitude (volts)-->');
title('Manchester-L encoded bit stream');
```

Al comparar la PSD de diferentes códigos de línea, los ingenieros Telemáticos pueden seleccionar el esquema de codificación más adecuado para una aplicación específica, optimizando así el rendimiento del sistema en términos de ancho de banda, consumo de energía y complejidad de implementación. En un mundo donde la eficiencia espectral y la robustez de la comunicación son cada vez más demandadas, entender y analizar la PSD se vuelve indispensable para el desarrollo de soluciones de comunicación confiables y eficientes. A continuación se muestra la PSD teórica para dos esquemas de codificación de línea (tomado de https://www.siue.edu/~yadwang/ECE375_Lec8.pdf)

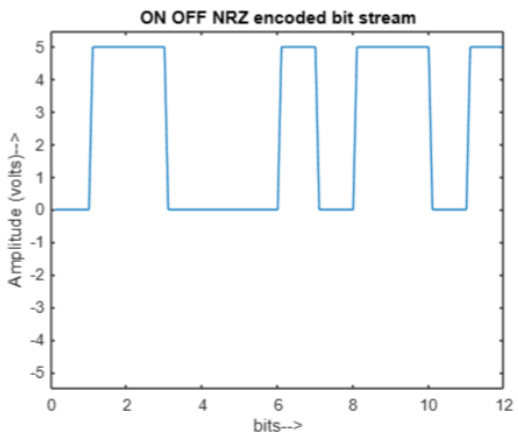


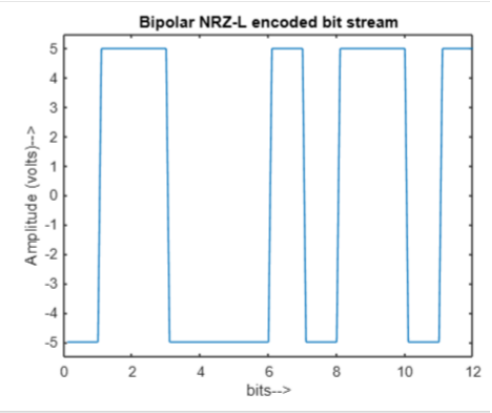


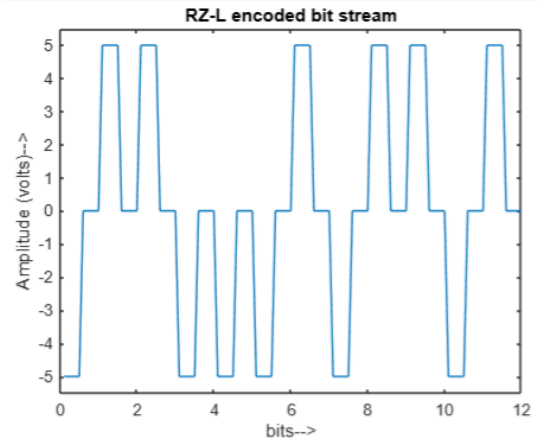
La comparación de la PSD de estos esquemas de codificación de línea es la siguiente:

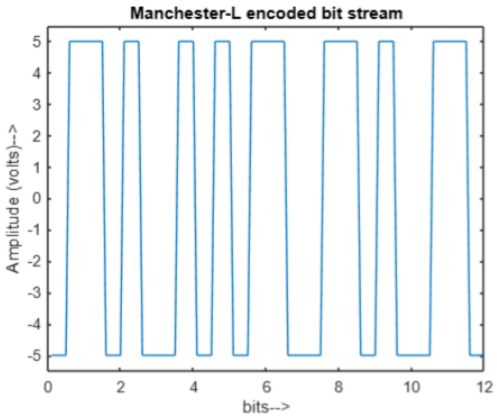


La salida de los diferentes codificadores de línea para la secuencia de bits 0 1 1 0 0 0 1 0 1 1 0 1 se grafica de la siguiente manera:









Preguntas y Ejercicios Basados en el Código de Simulación:

Parte 1: Interpretación de Resultados

Observando las gráficas de PSD (densidad espectral de potencia) para los diferentes esquemas de códigos de línea, ¿qué código de línea tiene el mayor ancho de banda y cuál el menor? ¿Cómo se explica esto con base en la teoría?

R/

Con base en la gráfica de Welch Power Spectral Density, el Manchester-L (Bipolar RZ) muestra el mayor ancho de banda, ya que se espectro se extiende más en la frecuencia y decae más lentamente. El On-Off NRZ muestra el menor ancho de banda, con una concentración de potencia en las frecuencias más bajas. Esto se explica por la teoría porque el código Manchester requiere una transición obligatoria en la mitad de cada intervalo de bit, lo que significa que tiene el doble de transiciones que otros códigos. Más transiciones implican cambios más rápidos en la señal, lo que se traduce en componentes de frecuencia más altas y por tanto mayor ancho de banda. El NRZ, por otro lado, solo cambia cuando hay un cambio en el valor del

bit, lo que resulta en menos transiciones y por tanto requiere menos ancho de banda.

Describe las características de sincronización los códigos en la presencia de a una cadena larga de unos o de ceros. Por ejemplo, compara el comportamiento del código Manchester (que tiene una transición en cada bit) con el On-Off NRZ Explique ¿por qué Manchester proporciona mejor sincronización?

R/

Comparando el código de Manchester, tiene una transición garantizada en la mitad de cada intervalo de bit, para un ‘1’ la transición es de +V a -V, para un ‘0’ la transición es de -V a +V. Esto significa que incluso con una larga secuencia de unos o ceros, siempre habrá transiciones regulares.

En el On-Off NRZ solo hay transición cuando hay un cambio en el valor del bit, una larga secuencia de unos o ceros resulta en un nivel constante sin transiciones. Esto hace más difícil para el receptor mantener la sincronización.

El código de Manchester proporciona una mejor sincronización porque las transiciones regulares y garantizadas permiten al receptor mantener su reloj interno sincronizado con la señal entrante. El receptor puede usar estas transiciones para determinar con precisión el inicio y el fin de cada bit. Además, la ausencia de componente DC (debido a las transiciones simétricas) hace más fácil la detención de las transiciones.

Parte 2: Comparación y Simulación con Comunicaciones (RS-232 y UART)

1. Considerando los códigos de línea implementados en clase, ¿cuál esquema sería adecuado para la comunicación RS-232? ¿cuál esquema sería adecuado para la comunicación UART?

R/

En el análisis de los protocolos de comunicación RS-232 y UART, la selección del código de línea más apropiado resulta fundamental para garantizar una comunicación eficiente.

Para el protocolo RS-232, el código de línea NRZ polar (Bipolar NRZ-L) se presenta como la opción más adecuada. Esta elección se fundamenta en las características eléctricas del protocolo, que opera con niveles de voltaje bipolares (+3V a +15V para '0' lógico, -3V a -15V para '1' lógico). El NRZ polar se adapta naturalmente a esta especificación debido a sus dos niveles de voltaje opuestos y presenta un espectro de frecuencia concentrado que beneficia las transmisiones a largas

distancias. La separación amplia entre niveles lógicos proporciona mejor inmunidad al ruido, crucial para las aplicaciones típicas de RS-232.

Para UART, el código NRZ On-Off representa la mejor alternativa. Este protocolo asíncrono, que opera con niveles lógicos TTL/CMOS (típicamente 0V y 5V o 3.3V), requiere un esquema de codificación compatible con estas características. El NRZ On-Off resulta ideal por su compatibilidad directa con estos niveles lógicos y su simplicidad de implementación. Se puede observar que genera formas de onda claras y definidas, mientras que posee el menor ancho de banda entre las opciones analizadas, ideal para las comunicaciones a corta distancia típicas de UART. La sincronización se gestiona eficientemente mediante el formato 8N1, mitigando las potenciales desventajas del NRZ On-Off en este aspecto.

Estas elecciones se basan en un balance entre complejidad de implementación, requerimientos eléctricos, resistencia al ruido y eficiencia espectral, respaldadas por las características espectrales y temporales mostradas en las imágenes proporcionadas.

2. Ajustes para Simular Comunicación RS-232 (9600 8N1)

En RS-232 a 9600 baudios, con un formato de 8 bits de datos, sin paridad y 1 bit de parada (8N1), se pide realizar ajustes sobre el código de Matlab para simular la transmisión de una cadena de 100 caracteres aleatorios. Muestre la gráfica de PSD.

Además, realice la simulación en Matlab de una comunicación UART (9600 8N1) entre dos tarjetas ESP32. En este caso, se le pide simular el flujo de datos de 100 caracteres aleatorios. Muestre la gráfica de PSD.

Luego muestre el bitstream al trasmitir juntos los caracteres A y Z para cada uno de los esquemas de códigos de línea.

Explique que cambios se puede ver en la PSD si para la misma secuencia de bits, se multiplica el Rb por 100?

R/

Al aumentar la tasa de baudios Rb por un factor de 100, la señal transmite bits más rápido. Esto provoca que los intervalos entre bits se reduzcan, concentrando la energía de la señal en frecuencias más altas. Como resultado, la PSD se desplaza hacia frecuencias mayores, y la distribución de la potencia se vuelve más concentrada. Además, la resolución de la PSD mejora debido a la mayor frecuencia de muestreo, pero la potencia en frecuencias bajas disminuye.

Aunque en la práctica el UART se usa para la comunicación entre microcontroladores como el ESP32, en la simulación de MATLAB no se modifica el protocolo para estos dispositivos. El código simula la transmisión de datos usando

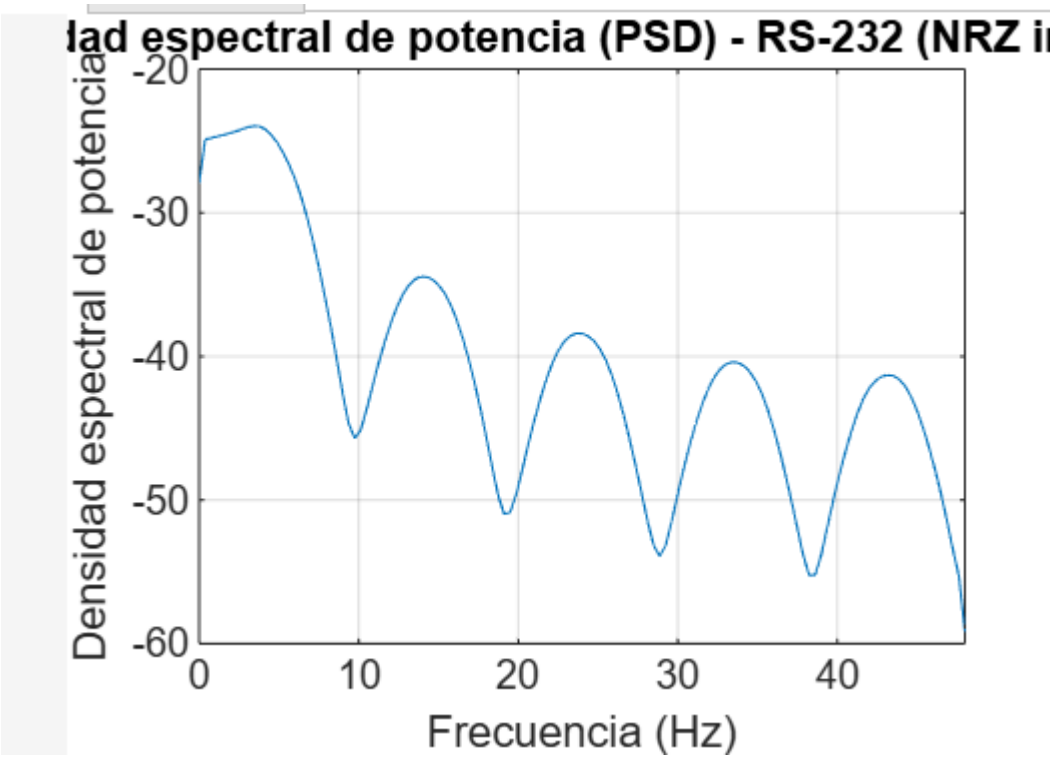
los protocolos **RS-232** y **UART**, ambos con la misma configuración: 9600 baudios, 8 bits de datos, sin paridad y 1 bit de parada.

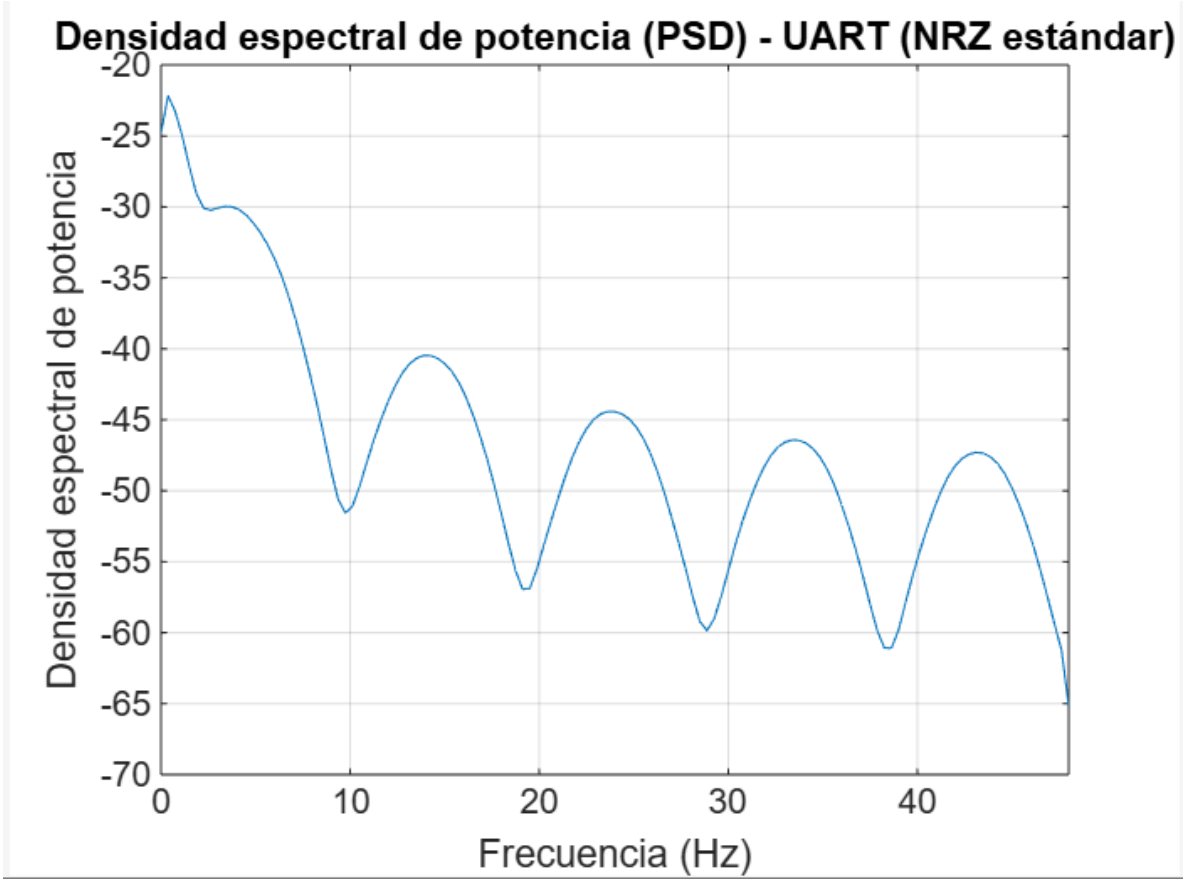
El código genera las representaciones binarias de los caracteres 'A' y 'Z', añadiendo bits de **inicio** y **parada**. Luego, se aplica cada tipo de codificación de línea:

- **NRZ no polar**: "1" se transmite como voltaje positivo, "0" como cero.
- **NRZ-L**: "1" como positivo y "0" como negativo.
- **RZ**: "1" es un pulso de voltaje seguido de cero, y "0" es simplemente cero.
- **Manchester**: "1" y "0" se representan por transiciones de voltaje.

El código calcula la PSD para cada tipo de codificación. Al cambiar la tasa de baudios, la PSD se concentra en frecuencias más altas, lo que se ve claramente en las gráficas. Con una tasa de baudios más alta, el espectro se desplaza hacia frecuencias más altas, reduciendo el ancho de banda de la señal.

Cuando se multiplica Rb por 100, la señal se transmite más rápido, lo que comprime la energía de la señal en frecuencias más altas. Esto provoca que la PSD se concentre en un rango de frecuencias más estrecho, y el ancho de banda de la señal se reduzca, concentrándose más en las frecuencias altas.






```
clear all;
close all;

% Integrante: Santiago Valencia García - A00395902

% Parámetros de la simulación
num_chars = 100; % Número de caracteres a transmitir
baud_rate = 9600; % Tasa de baudios (9600 baudios)
sampling_freq = 10 * baud_rate; % Frecuencia de muestreo (10 veces la tasa de
baudios)
peak_voltage = 5; % Voltaje máximo (5V)

% Generación de 100 caracteres ASCII aleatorios (del 32 al 126)
random_chars = randi([32, 126], 1, num_chars);

% Inicialización del flujo de bits
bitstream = [];

% Generación del bitstream con bits de inicio (0) y parada (1)
for i = 1:num_chars
    % Conversión de cada carácter a su representación binaria (8 bits)
    char_bits = de2bi(random_chars(i), 8, 'right-msb');
    % Añadir los bits de inicio (0) y parada (1)
    frame = [0 char_bits 1];
    bitstream = [bitstream frame];
end
```

```
% Codificación NRZ con polaridad invertida (RS-232)
NRZ_RS232_signal = [];
for bit = bitstream
    if bit == 1
        NRZ_RS232_signal = [NRZ_RS232_signal ones(1, sampling_freq/baud_rate)*(-
peak_voltage)]; % Lógico '1' = voltaje negativo
    else
        NRZ_RS232_signal = [NRZ_RS232_signal ones(1,
sampling_freq/baud_rate)*(peak_voltage)]; % Lógico '0' = voltaje positivo
    end
end

% Cálculo de la PSD utilizando el método de Welch
psd_estimation = spectrum.welch;
psd_RS232 = psd(psd_estimation, NRZ_RS232_signal, 'Fs', sampling_freq);
figure;
plot(psd_RS232);
title('Densidad espectral de potencia (PSD) - RS-232 (NRZ invertido)');
xlabel('Frecuencia (Hz)');
ylabel('Densidad espectral de potencia');

% Codificación NRZ estándar para UART
NRZ_UART_signal = [];
for bit = bitstream
    if bit == 1
        NRZ_UART_signal = [NRZ_UART_signal ones(1,
sampling_freq/baud_rate)*(peak_voltage)]; % Lógico '1' = voltaje positivo
    else
        NRZ_UART_signal = [NRZ_UART_signal ones(1, sampling_freq/baud_rate)*0];
% Lógico '0' = voltaje cero
    end
end

% Cálculo de la PSD para la señal UART
psd_UART = psd(psd_estimation, NRZ_UART_signal, 'Fs', sampling_freq);
figure;
plot(psd_UART);
title('Densidad espectral de potencia (PSD) - UART (NRZ estándar)');
xlabel('Frecuencia (Hz)');
ylabel('Densidad espectral de potencia');

%% Codificación de los caracteres 'A' y 'Z'
chars_to_transmit = ['A', 'Z'];
bitstream_chars = [];
for i = 1:length(chars_to_transmit)
    % Conversión de cada carácter a binario (8 bits)
    char_bits = de2bi(chars_to_transmit(i), 8, 'right-msb');
    % Añadir bits de inicio (0) y parada (1)
    frame = [0 char_bits 1];
    bitstream_chars = [bitstream_chars frame];
end

% Codificación NRZ no polar
NRZ_no_polar = [];
```

```
for bit = bitstream_chars
    if bit == 1
        NRZ_no_polar = [NRZ_no_polar ones(1,
sampling_freq/baud_rate)*(peak_voltage)];
    else
        NRZ_no_polar = [NRZ_no_polar ones(1, sampling_freq/baud_rate)*0];
    end
end

% Codificación NRZ-L
NRZ_L_signal = [];
for bit = bitstream_chars
    if bit == 1
        NRZ_L_signal = [NRZ_L_signal ones(1,
sampling_freq/baud_rate)*(peak_voltage)];
    else
        NRZ_L_signal = [NRZ_L_signal ones(1, sampling_freq/baud_rate)*(-
peak_voltage)];
    end
end

% Codificación RZ
RZ_signal = [];
current_level = peak_voltage;
for bit = bitstream_chars
    if bit == 1
        pulse = [ones(1, sampling_freq/(2*baud_rate))*(current_level) zeros(1,
sampling_freq/(2*baud_rate))];
        RZ_signal = [RZ_signal pulse];
        current_level = -current_level; % Invertir el nivel de voltaje
    else
        RZ_signal = [RZ_signal zeros(1, sampling_freq/baud_rate)];
    end
end

% Codificación Manchester
Manchester_signal = [];
for bit = bitstream_chars
    if bit == 1
        symbol = [ones(1, sampling_freq/(2*baud_rate))*(peak_voltage) ones(1,
sampling_freq/(2*baud_rate))*(-peak_voltage)];
    else
        symbol = [ones(1, sampling_freq/(2*baud_rate))*(-peak_voltage) ones(1,
sampling_freq/(2*baud_rate))*(peak_voltage)];
    end
    Manchester_signal = [Manchester_signal symbol];
end

% Tiempo de simulación para graficar las señales
time = (0:length(NRZ_no_polar)-1)/sampling_freq;

% Crear una figura para visualizar los bitstreams
figure;
subplot(2,2,1);
plot(time, NRZ_no_polar);
```

```
title('NRZ no polar (A y Z)');
xlabel('Tiempo (s)');
ylabel('Amplitud (V)');
ylim([-peak_voltage*1.1 peak_voltage*1.1]);

subplot(2,2,2);
plot(time, NRZ_L_signal);
title('NRZ-L (A y Z)');
xlabel('Tiempo (s)');
ylabel('Amplitud (V)');
ylim([-peak_voltage*1.1 peak_voltage*1.1]);

subplot(2,2,3);
plot(time, RZ_signal);
title('RZ (A y Z)');
xlabel('Tiempo (s)');
ylabel('Amplitud (V)');
ylim([-peak_voltage*1.1 peak_voltage*1.1]);

subplot(2,2,4);
plot(time, Manchester_signal);
title('Manchester (A y Z)');
xlabel('Tiempo (s)');
ylabel('Amplitud (V)');
ylim([-peak_voltage*1.1 peak_voltage*1.1]);

%% Aumento de la tasa de baudios (Rb multiplicado por 100)
new_baud_rate = baud_rate * 100; % Aumento de Rb por un factor de 100
new_sampling_freq = 10 * new_baud_rate; % Frecuencia de muestreo ajustada

% Recalculamos la señal UART con la nueva tasa de baudios
NRZ_UART_high = [];
for bit = bitstream
    if bit == 1
        NRZ_UART_high = [NRZ_UART_high ones(1,
new_sampling_freq/new_baud_rate)*(peak_voltage)];
    else
        NRZ_UART_high = [NRZ_UART_high ones(1,
new_sampling_freq/new_baud_rate)*0];
    end
end

% Cálculo de la PSD para la nueva señal UART con tasa de baudios alta
psd_high = psd(psd_estimation, NRZ_UART_high, 'Fs', new_sampling_freq);
figure;
plot(psd_high);
title('PSD de la señal UART con Rb multiplicado por 100');
xlabel('Frecuencia (Hz)');
ylabel('Densidad espectral de potencia');
```