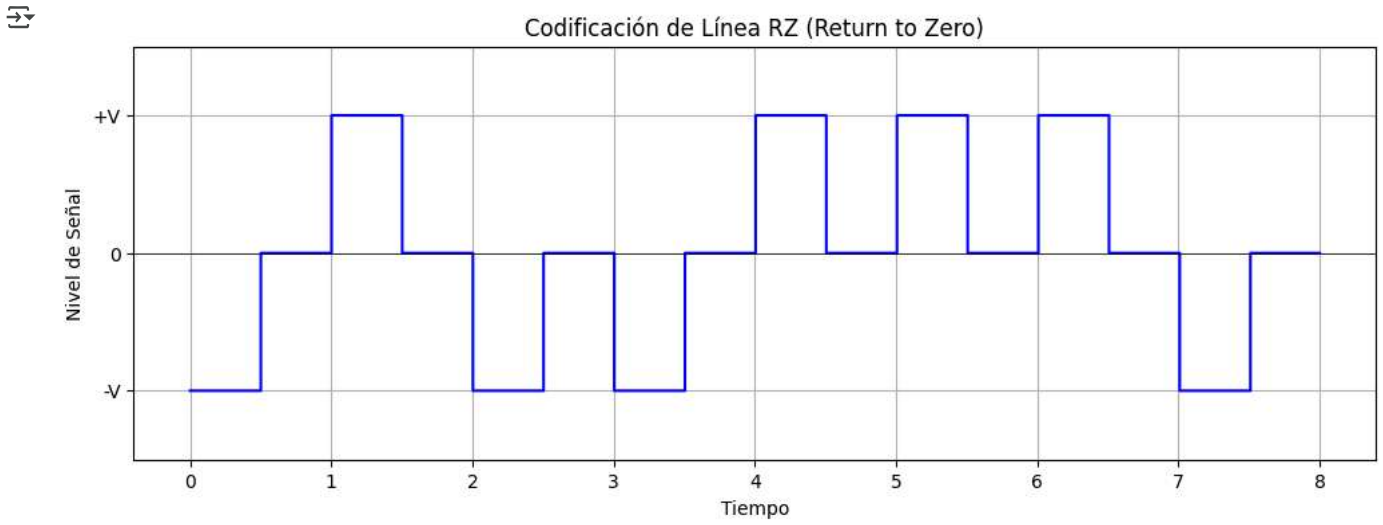
RZ (Return to Zero)

```
import matplotlib.pyplot as plt
import numpy as np

# Datos binarios de ejemplo (puedes cambiarlos)
bits = [0,1,0,0,1,1,1,0]

# Parámetros de la señal
bit_duration = 1  # duración de cada bit (en segundos o unidades arbitrarias)
samples_per_bit = 100  # resolución de la señal
t = []
signal = []

for bit in bits:
    # Tiempo para medio bit (ida)
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    # Señal para RZ
    if bit == 1:
        s = np.concatenate([np.ones(samples_per_bit // 2), np.zeros(samples_per_bit // 2)])
    else:
        s = np.concatenate([-1 * np.ones(samples_per_bit // 2), np.zeros(samples_per_bit // 2)])

    signal.extend(s)
    t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))

# Ajustar el eje de tiempo total
t = np.linspace(0, len(bits) * bit_duration, len(signal))

# Graficar la señal
plt.figure(figsize=(12, 4))
plt.plot(t, signal, drawstyle='steps-post', color='blue')
plt.title("Codificación de Línea RZ (Return to Zero)")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-1.5, 1.5)
plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```



NRZ (versión NRZ-L: Level)

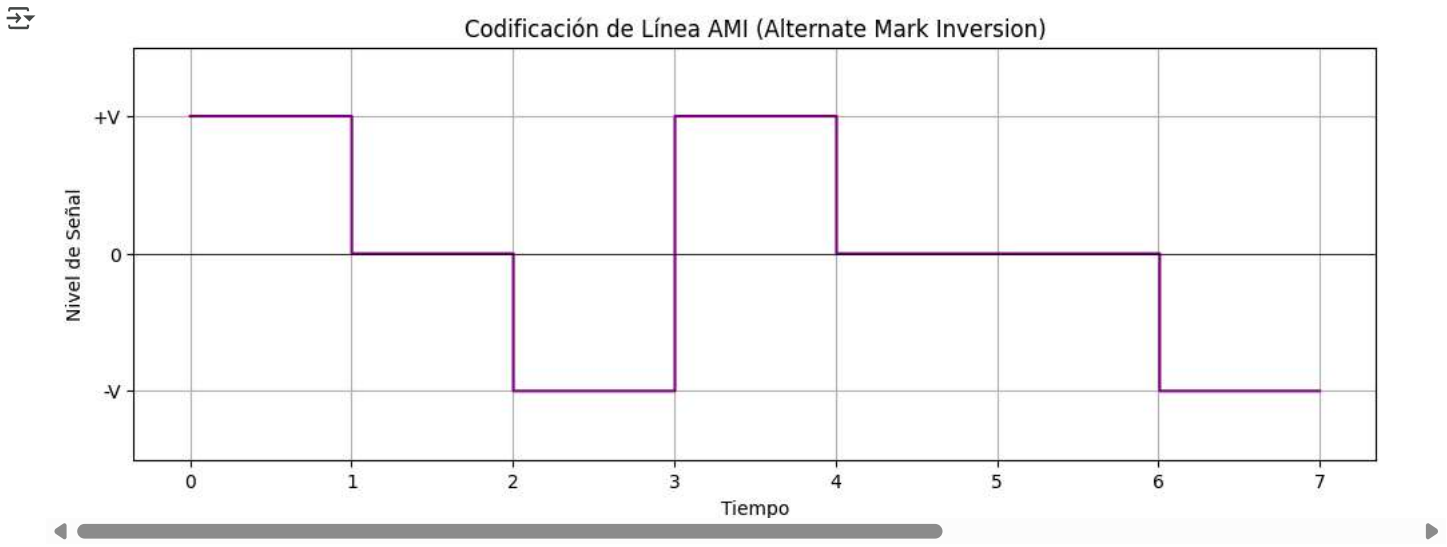
```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia binaria de ejemplo (puedes cambiarla)
bits = [1, 0, 1, 1, 0, 0, 1]
```

```
# Parámetros de la señal
bit_duration = 1
samples_per_bit = 100

# Tiempo y señal
t = []
signal = []

for bit in bits:
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    # Codificación NRZ-L (1 = +V, 0 = -V)
    if bit == 1:
        s = np.ones(samples_per_bit)  # nivel alto
    else:
        s = -1 * np.ones(samples_per_bit)  # nivel bajo

    signal.extend(s)
    t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))

# Ajustar eje de tiempo completo
t = np.linspace(0, len(bits) * bit_duration, len(signal))

# Graficar la señal
plt.figure(figsize=(12, 4))
plt.plot(t, signal, drawstyle='steps-post', color='green')
plt.title("Codificación de Línea NRZ (Non Return to Zero) - Nivel")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-1.5, 1.5)
plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```

AMI (Alternate Mark Inversion)

```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia binaria de ejemplo
bits = [1, 0, 1, 1, 0, 0, 1]

# Parámetros
bit_duration = 1
samples_per_bit = 100
t = []
signal = []

# Control de polaridad alterna para los unos
last_polarity = -1  # comienza con -1 para que el primer 1 sea +1
```

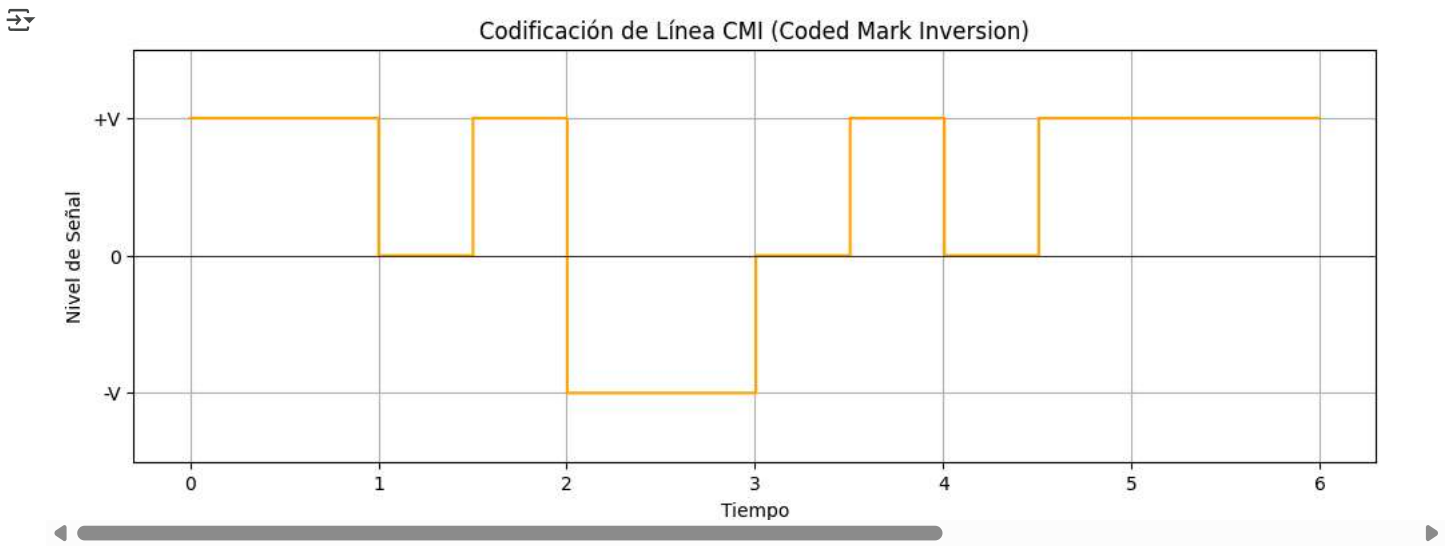
```
for bit in bits:
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    if bit == 1:
        # Alternar la polaridad
        last_polarity *= -1
        s = last_polarity * np.ones(samples_per_bit)
    else:
        # Cero lógico se representa como 0 V
        s = np.zeros(samples_per_bit)

    signal.extend(s)
    t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))

# Tiempo total
t = np.linspace(0, len(bits) * bit_duration, len(signal))

# Graficar
plt.figure(figsize=(12, 4))
plt.plot(t, signal, drawstyle='steps-post', color='purple')
plt.title("Codificación de Línea AMI (Alternate Mark Inversion)")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-1.5, 1.5)
plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```



CMI (Coded Mark Inversion)

```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia de bits de ejemplo
bits = [1, 0, 1, 0, 0, 1]

# Parámetros de la señal
bit_duration = 1
samples_per_bit = 100
t = []
signal = []

# Alternancia para bits 1
last_polarity = -1

for bit in bits:
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    if bit == 0:
        # Un '0' es una transición: mitad baja (0), mitad alta (+1)
        s = np.concatenate([np.zeros(samples_per_bit // 2), np.ones(samples_per_bit // 2)])
```

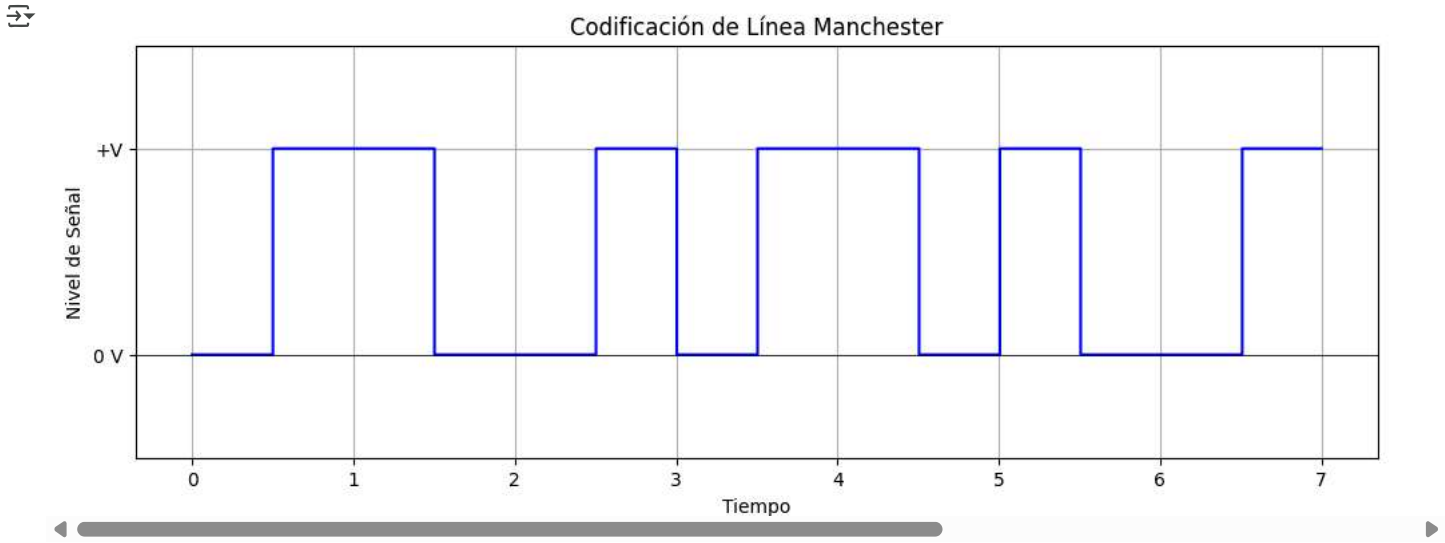
```
        else:
            # Un '1' es una señal constante, alternando entre +1 y -1
            last_polarity *= -1
            s = last_polarity * np.ones(samples_per_bit)

        signal.extend(s)
        t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))

    # Ajustar tiempo
    t = np.linspace(0, len(bits) * bit_duration, len(signal))

    # Graficar
    plt.figure(figsize=(12, 4))
    plt.plot(t, signal, drawstyle='steps-post', color='orange')
    plt.title("Codificación de Línea CMI (Coded Mark Inversion)")
    plt.xlabel("Tiempo")
    plt.ylabel("Nivel de Señal")
    plt.grid(True)
    plt.ylim(-1.5, 1.5)
    plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
    plt.xticks(np.arange(0, len(bits) + 1, 1))
    plt.axhline(0, color='black', linewidth=0.5)
    plt.show()
```



Manchester

```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia binaria de ejemplo
bits = [1, 0, 1, 1, 0, 0, 1]

# Parámetros
bit_duration = 1
samples_per_bit = 100
half = samples_per_bit // 2

t = []
signal = []

for bit in bits:
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    if bit == 1:
        # 1 = baja -> alta
        s = np.concatenate([np.zeros(half), np.ones(half)])
    else:
        # 0 = alta -> baja
        s = np.concatenate([np.ones(half), np.zeros(half)])

    signal.extend(s)
    t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))
```

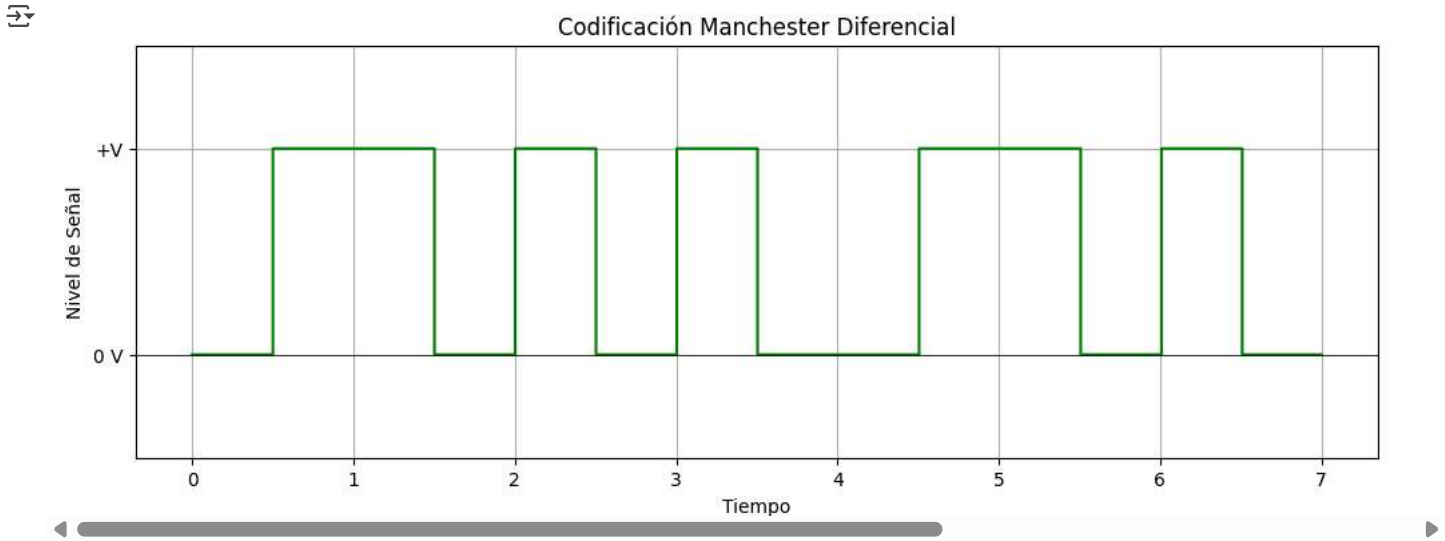
```
# Tiempo total
t = np.linspace(0, len(bits) * bit_duration, len(signal))

# Gráfico
plt.figure(figsize=(12, 4))
plt.plot(t, signal, drawstyle='steps-post', color='blue')
plt.title("Codificación de Línea Manchester")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-0.5, 1.5)
plt.yticks([0, 1], ['0 V', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```



Manchester Diferencial

```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia binaria de ejemplo
bits = [1, 0, 1, 1, 0, 0, 1]

# Parámetros
bit_duration = 1
samples_per_bit = 100
half = samples_per_bit // 2

# Inicialización
signal = []
t = []
current_level = 1  # comienza en nivel alto

for bit in bits:
    t_bit = np.linspace(0, bit_duration, samples_per_bit, endpoint=False)

    if bit == 0:
        # No transición al inicio
        first_half = [current_level] * half
        current_level = 1 - current_level  # Cambio en el medio
        second_half = [current_level] * half
    else:
        # Transición al inicio
        current_level = 1 - current_level
        first_half = [current_level] * half
        current_level = 1 - current_level  # Cambio en el medio
        second_half = [current_level] * half

    s = first_half + second_half
    signal.extend(s)
    t.extend(t_bit + len(t) * (bit_duration / samples_per_bit))
```

```
# Tiempo total
t = np.linspace(0, len(bits) * bit_duration, len(signal))

# Graficar
plt.figure(figsize=(12, 4))
plt.plot(t, signal, drawstyle='steps-post', color='green')
plt.title("Codificación Manchester Diferencial")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-0.5, 1.5)
plt.yticks([0, 1], ['0 V', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```

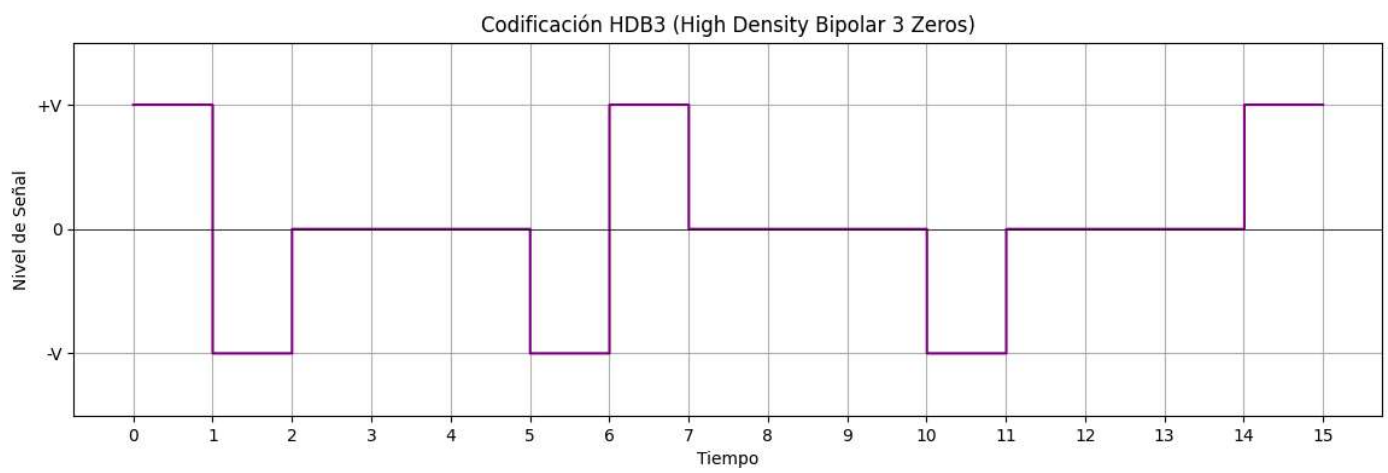


HDB3 (High-Density Bipolar 3 zeros)

> Añadir blockquote

```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia de bits original
bits = [1,1,0,0,0,0,1,0,0,0,0,0,0,0,0]

# Parámetros
bit_duration = 1
samples_per_bit = 100
t = []
signal = []

# Variables internas
pulses_since_last_violation = 0
last_polarity = -1  # Para alternancia

# Construir señal codificada
i = 0
while i < len(bits):
    if bits[i] == 1:
        # Alterna polaridad
        last_polarity *= -1
        pulses_since_last_violation += 1
        s = [last_polarity] * samples_per_bit
        i += 1
    elif i + 3 < len(bits) and bits[i:i+4] == [0, 0, 0, 0]:
        # Detecta 0000 y reemplaza según paridad
        if pulses_since_last_violation % 2 == 0:
            # Reemplazo tipo B00V
            v = last_polarity
            s = (
                [0] * samples_per_bit +  # B
```

```
                [0] * samples_per_bit +  # 0
                [0] * samples_per_bit +  # 0
                [v] * samples_per_bit    # V
            )
            last_polarity = v
        else:
            # Reemplazo tipo 000V
            v = -last_polarity
            s = (
                [0] * samples_per_bit +  # 0
                [0] * samples_per_bit +  # 0
                [0] * samples_per_bit +  # 0
                [v] * samples_per_bit    # V
            )
        pulses_since_last_violation = 0
        signal.extend(s)
        i += 4
        continue
    else:
        # Es un cero normal
        s = [0] * samples_per_bit
        i += 1

    signal.extend(s)

# Tiempo total
t = np.linspace(0, len(signal) / samples_per_bit, len(signal))

# Graficar
plt.figure(figsize=(14, 4))
plt.plot(t, signal, drawstyle='steps-post', color='purple')
plt.title("Codificación HDB3 (High Density Bipolar 3 Zeros)")
plt.xlabel("Tiempo")
plt.ylabel("Nivel de Señal")
plt.grid(True)
plt.ylim(-1.5, 1.5)
plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
plt.xticks(np.arange(0, len(bits) + 1, 1))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

# Secuencia de bits original
bits = [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

# Parámetros
samples_per_bit = 100

# Variables internas para codificación
last_polarity = -1        # Última polaridad usada (alternar entre +1 y -1)
pulse_count = 0           # Conteo de pulsos desde la última violación
signal = []               # Señal codificada
```

```
i = 0
while i < len(bits):
    if bits[i] == 1:
        # Pulsos de 1 alternando polaridad
        last_polarity *= -1
        pulse_count += 1
        signal.extend([last_polarity] * samples_per_bit)
        i += 1
    else:
        # Detectar bloque de cuatro ceros consecutivos
        if i + 3 < len(bits) and bits[i:i+4] == [0, 0, 0, 0]:
            if pulse_count % 2 == 0:
                # Patrón B00V: B con polaridad opuesta (como si fuera un "1"), V con misma polaridad
                last_polarity *= -1
                signal.extend([last_polarity] * samples_per_bit)   # B
                signal.extend([0] * samples_per_bit * 2)           # 0 0
                signal.extend([last_polarity] * samples_per_bit)   # V
            else:
                # Patrón 000V: V con polaridad opuesta (normal alternancia)
                signal.extend([0] * samples_per_bit * 3)           # 0 0 0
                last_polarity *= -1
                signal.extend([last_polarity] * samples_per_bit)   # V
            pulse_count = 0
            i += 4
        else:
            # Cero normal
            signal.extend([0] * samples_per_bit)
            i += 1

# Vector de tiempo para la gráfica
t = np.linspace(0, len(bits), len(signal))

# Gráfica
plt.figure(figsize=(15, 4))
plt.plot(t, signal, drawstyle='steps-post', color='purple', linewidth=2)
plt.title("Codificación HDB3 (High Density Bipolar of order 3)")
plt.xlabel("Tiempo (bits)")
plt.ylabel("Nivel de señal")
plt.ylim(-1.5, 1.5)
plt.yticks([-1, 0, 1], ['-V', '0', '+V'])
plt.grid(True)

# Añadir líneas verticales para los bits
for x in range(len(bits) + 1):
    plt.axvline(x=x, color='gray', linestyle='--', linewidth=0.5)

# Línea base
plt.axhline(y=0, color='black', linewidth=0.7)

# Mostrar
plt.show()
```