BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India

DEPARTMENT OF COMPUTER Science ENGINEERING
SUBJECT: MDM Lab 1

Name	Pratham Masurkar
UID no.	2023800056
class and branch	TE Computer Science and Engineering - Batch C
Lab	3
Aim	To understand and simulate various line coding schemes used in digital communication

Implementation	see code below

Implementation

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

# --- Step 1: ASCII to Binary ---
def str_to_bin(s):
    return ''.join([format(ord(c), '08b') for c in s])

data = "pra"
bin_data = str_to_bin(data)
print(f"Binary of '{data}': {bin_data}")

# --- Step 2: Polar NRZ and Manchester ---
def polar_nrz(bits):
    # 1 -> +1, 0 -> -1
    return np.array([1 if b == '1' else -1 for b in bits])

def manchester(bits):
    # 1 bit = 2 samples
    # 0 -> high to low (1 then -1)
    # 1 -> low to high (-1 then 1)
    manch = []
    for b in bits:
        if b == '0':
            manch.extend([1, -1])
        else:
            manch.extend([-1, 1])
    return np.array(manch)

# --- Step 3: Insert 8 zeros randomly, then AMI and B8ZS
coding ---
def insert_zeros(bits, n_zeros, positions=1):
    bits = list(bits)
    for _ in range(positions):
        pos = random.randint(0, len(bits))
        for _ in range(n_zeros):
            bits.insert(pos, '0')
    return ''.join(bits)

def ami(bits):
    # 1 alternates +1, -1; 0 = 0
    output = []
    last_pulse = -1
    for b in bits:
        if b == '1':
            last_pulse *= -1
            output.append(last_pulse)
        else:
            output.append(0)
    return np.array(output)

def b8zs(bits):
    # AMI first
    ami_code = list(ami(bits))
    # B8ZS substitutes any 8 consecutive zeros with the
pattern: 000VB0VB
    i = 0
    last_pulse = 1 if 1 in ami_code else -1
    while i <= len(ami_code) - 8:
        if all(x == 0 for x in ami_code[i:i+8]):
            V = last_pulse
            B = -last_pulse
            pattern = [0, 0, 0, V, B, 0, V, B]
            for j in range(8):
                ami_code[i+j] = pattern[j]
                last_pulse = ami_code[i+7]
                i += 8
            else:
                i += 1
    return np.array(ami_code)

# --- Step 4: Insert 4 zeros twice, then AMI and HDB3 ---
def hdb3(bits):
    # AMI first
    ami_code = list(ami(bits))
    i = 0
    last_pulse = 1
    pulse_count = 0  # count pulses since last substitution
    while i <= len(ami_code) - 4:
        if all(x == 0 for x in ami_code[i:i+4]):
            if pulse_count % 2 == 0:
                # substitution pattern: 000V (violation
pulse)
                V = last_pulse
                pattern = [0, 0, 0, V]
            else:
                # substitution pattern: B00V (bipolar pulse,
2 zeros, violation)
                V = last_pulse
                B = -last_pulse
                pattern = [B, 0, 0, V]
            for j in range(4):
                ami_code[i+j] = pattern[j]
            last_pulse = ami_code[i+3]
            pulse_count = 0
            i += 4
        else:
            if ami_code[i] != 0:
                pulse_count += 1
                last_pulse = ami_code[i]
            i += 1
    return np.array(ami_code)

# --- Step 5: Decode B8ZS ---
```

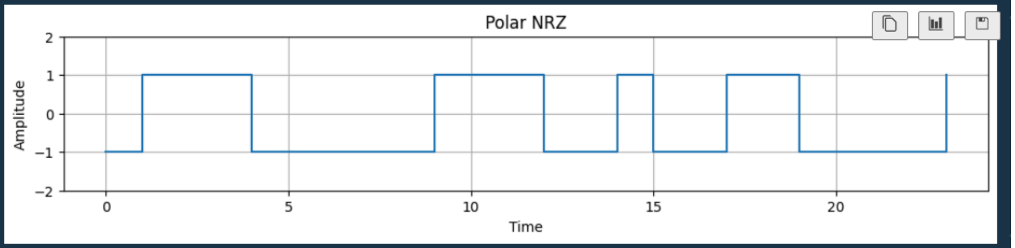
```
def decode_b8zs(signal):
    # Replace violation patterns with zeros
    i = 0
    decoded = signal.copy()
    while i <= len(signal) - 8:
        segment = signal[i:i+8]
        if (segment[0] == 0 and segment[1] == 0 and
segment[2] == 0 and
            segment[3] != 0 and segment[4] != 0 and
segment[5] == 0 and
            segment[6] != 0 and segment[7] != 0):
            decoded[i:i+8] = [0]*8
            i += 8
        else:
            i += 1
    return decoded

# --- Plotting ---
def plot_signal(signal, title, samples_per_bit=1):
    plt.figure(figsize=(12, 2))
    time = np.arange(len(signal)) / samples_per_bit
    plt.step(time, signal, where='post')
    plt.ylim(-2, 2)
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

# Helper: plot binary data (digital signal)
def plot_digital(bits, title):
    plt.figure(figsize=(12, 2))
    signal = [int(b) for b in bits]
    time = np.arange(len(signal))
    plt.step(time, signal, where='post')
    plt.ylim(-0.5, 1.5)
    plt.title(title)
    plt.xlabel('Bit index')
    plt.ylabel('Bit value')
```

```python
    plt.grid(True)
    plt.show()

# --- Execution ---
print("\n--- Polar NRZ ---")
polar = polar_nrz(bin_data)
plot_signal(polar, "Polar NRZ")

print("\n--- Manchester ---")
manch = manchester(bin_data)
plot_signal(manch, "Manchester", samples_per_bit=2)

# Insert 8 zeros randomly once
bin_data_8zero = insert_zeros(bin_data, 8, positions=1)
print(f"\nBinary with 8 zeros inserted once (random
position):\n{bin_data_8zero}")

ami_8zero = ami(bin_data_8zero)
plot_signal(ami_8zero, "AMI with 8 zeros inserted")

b8zs_code = b8zs(bin_data_8zero)
plot_signal(b8zs_code, "B8ZS coding with 8 zeros inserted")

# Insert 4 zeros twice randomly
bin_data_4zero = insert_zeros(bin_data, 4, positions=2)
print(f"\nBinary with 4 zeros inserted twice (random
positions):\n{bin_data_4zero}")

ami_4zero = ami(bin_data_4zero)
plot_signal(ami_4zero, "AMI with 4 zeros inserted twice")

hdb3_code = hdb3(bin_data_4zero)
plot_signal(hdb3_code, "HDB3 coding with 4 zeros inserted
twice")

# Decode B8ZS
decoded_b8zs = decode_b8zs(b8zs_code)
plot_signal(decoded_b8zs, "Decoded B8ZS signal")
```

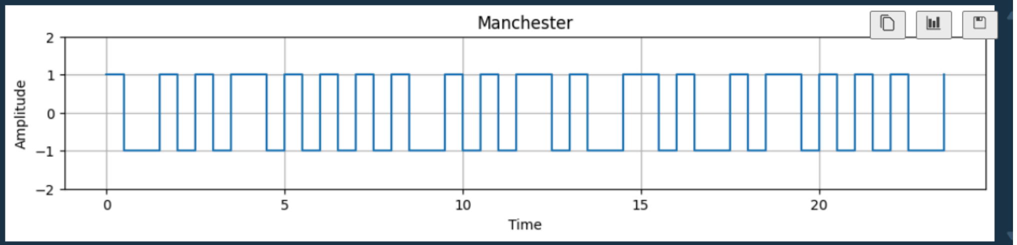
```python
# Compare original AMI and decoded B8ZS for error detection
print("\nDecoded B8ZS equals AMI before B8ZS substitution? ",
np.array_equal(decoded_b8zs, ami_8zero))

# Plot the binary data with inserted zeros
plot_digital(bin_data_8zero, "Binary Data with 8 zeros
inserted")

# Convert decoded B8ZS waveform back to bits (pulse -> 1,
zero -> 0)
decoded_bits = ''.join(['1' if abs(x) > 0 else '0' for x in
decoded_b8zs])

# Plot decoded B8ZS bits
plot_digital(decoded_bits, "Decoded B8ZS Data")

# Check if equal
print("Decoded B8ZS bits == Binary data with inserted zeros?
", decoded_bits == bin_data_8zero)
```

Output :

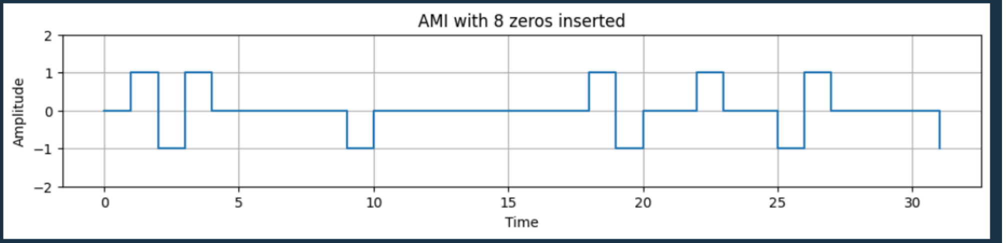
```
Binary of 'pra': 011100000111001001100001

--- Polar NRZ ---
```

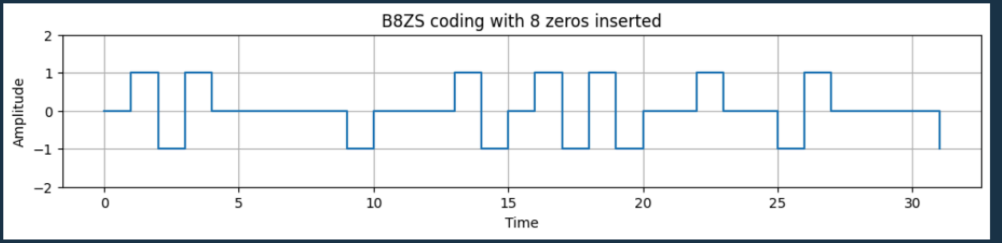


```
--- Manchester ---
```

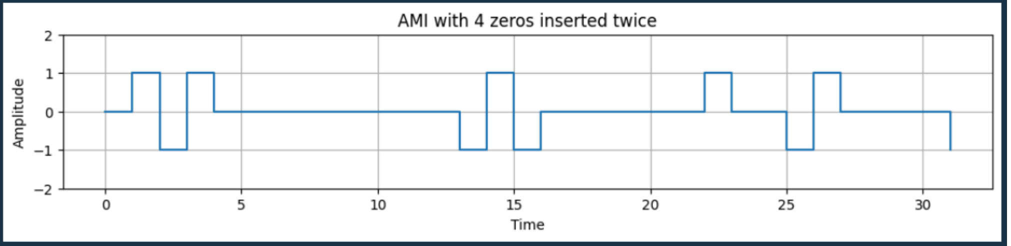


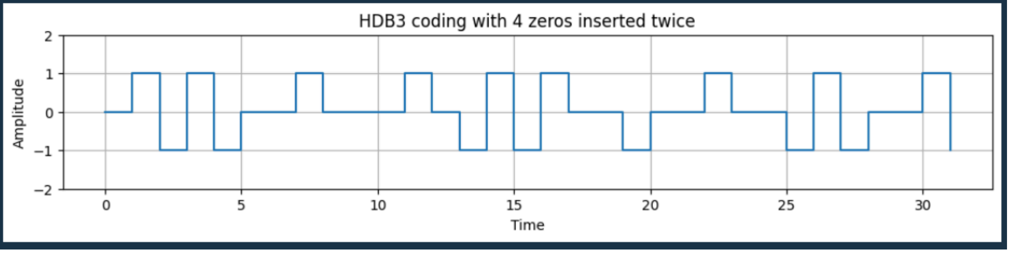
```
Binary with 8 zeros inserted once (random position):
01110000010000000011001001100001
```

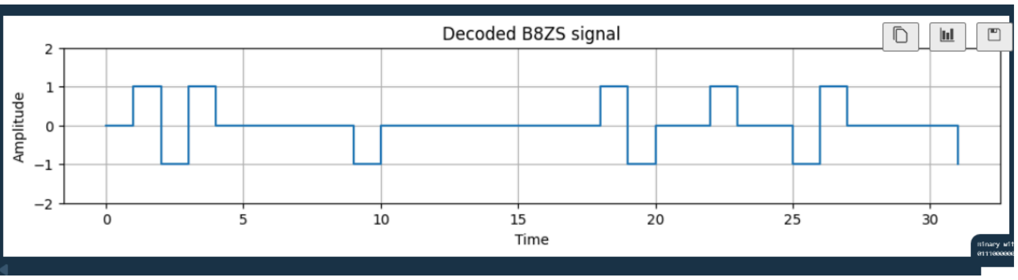


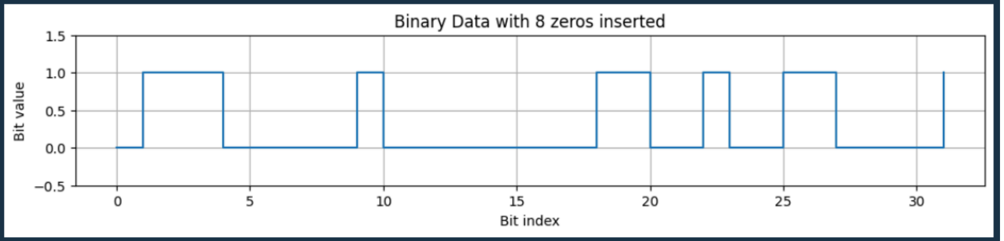
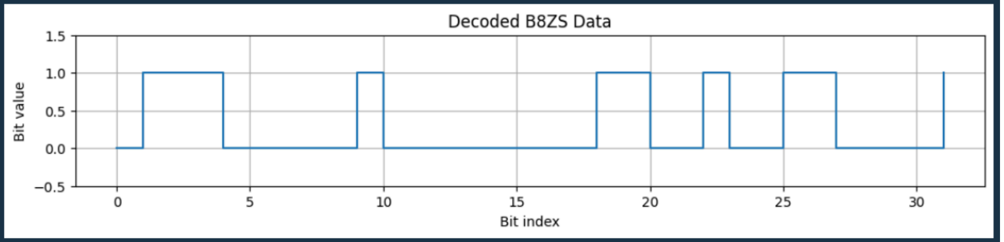


```
Binary with 4 zeros inserted twice (random positions):
01110000000001110000001001100001
```







	`Decoded B8ZS equals AMI before B8ZS substitution?  True`   `Decoded B8ZS bits == Binary data with inserted zeros?  True`
Conclusion :	Although basic line coding techniques such as Polar NRZ and Manchester are effective for translating binary data into electrical signals, they have notable limitations. The AMI scheme, for instance, can suffer from synchronization issues when long sequences of zeros occur, since a binary '0' is represented by zero voltage, resulting in no transitions in the signal. To address this, more advanced encoding methods like B8ZS and HDB3 were applied. These techniques enhance reliability by replacing long runs of zeros with specific pulse patterns, ensuring continuous signal transitions and maintaining synchronization.