DATA COMMUNICATION ASSIGNMENT

ASSIGNMENT GROUP INTORDUCTION

We are a group of 3 students

2021BITE038-YARRAMSETTI KUSUMA

2021BITE071-RAKHI SINGH

2021BITE072-KANCHAN MORE

BRANCH:- IT

BATCH:-2021-2025

SEMESTER:-5TH

ASSIGNMENT:- Line Encoding techniques Implementation

SUBMITTED TO:- DR. Iqra Altaf Gillani

LINE ENCODING TECHNIQUES IMPLMENTATION:

In this assignment we have implemented a line coding encoder, decoder and scrambler with a digital data generator that generates completely random data sequences and subsequences containing consecutive 4 or 8 zeros. The code is written in PYTHON JUPYTER.

IMPLEMENTAION DETAILS:

LANGUAGE USED:python jupyter.

CODE:

FOR NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI and scrambling schemes: B8ZS, HDB3.

FOR PCM AND DM.

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

def nrz_l(data):
    # NRZ-L encoding logic
    encoded_signal = [1 if bit == '1' else -1 for bit in data]
    return encoded_signal

def nrz_i(data):
    # NRZ-I encoding logic
    encoded_signal = [1]
    prev = 1
    for bit in data:
        if bit == '0':
            prev *= -1
        encoded_signal.append(prev)
    return encoded_signal

def manchester(data):
    # Manchester encoding logic
    encoded_signal = []
    for bit in data:
        if bit == '0':
            encoded_signal.extend([1, -1])
        else:
            encoded_signal.extend([-1, 1])
    return encoded_signal

def differential_manchester(data):
    # Differential Manchester encoding logic
```

```
    encoded_signal = []
    prev = 1
    for bit in data:
        if bit == '0':
            encoded_signal.extend([prev * -1, prev])
        else:
            prev *= -1
            encoded_signal.extend([prev, prev * -1])
    return encoded_signal

def ami(data):
    # AMI encoding logic
    encoded_signal = []
    prev = 0
    for bit in data:
        if bit == '0':
            encoded_signal.append(0)
        else:
            prev = -prev
            encoded_signal.append(prev)
    return encoded_signal

def b8zs(data):
    # B8ZS scrambling logic
    scrambled_data = []
    ones_count = 0
    for bit in data:
        if bit == '1':
            ones_count += 1
        else:
```

```
            ones_count = 0
        if ones_count >= 8:
            scrambled_data.extend([-1, -1, -1, 0, 0, 0, -1, 0])  # Replace eight consecutive ones
            ones_count = 0
        else:
            scrambled_data.append(int(bit))
    return scrambled_data

def hdb3(data):
    # HDB3 scrambling logic
    # Placeholder for HDB3 scrambling logic
    encoded_signal = []
    polarity = 0
    count = 0  # Counter for consecutive zeros

    for bit in data:
        if bit == '1':
            count = 0
            if polarity == 0:
                encoded_signal.extend([1, 0, 0, -1])  # HDB3 encoding for '1'
                polarity = 1
            else:
                encoded_signal.extend([-1 * polarity])  # Non-zero bit with the current polarity
        else:  # bit is '0'
            count += 1
            if count == 4:  # If four consecutive zeros are encountered
                if polarity != 0:  # If the previous substitution was not violated
                    encoded_signal[-4] = 0  # Replace the last 000V pattern with '0'
                    count = 0
                else:  # Violation
```

```
            encoded_signal.extend([0, 0, 0, 0])  # Insert '000V' to solve the violation
            count = 0
        else:
          encoded_signal.append(0)  # Encode '0' as '0'

  return encoded_signal



# Implement other encoding schemes


def apply_scrambling(data, scrambling_type):
  if scrambling_type == 'HDB3':
    return hdb3(data)
  else:
    return data


def pcm(analog_input, quantization_bits=8):
  # PCM logic
  max_value = 2 ** quantization_bits / 2 - 1
  step_size = max_value / max(abs(analog_input.min()), abs(analog_input.max()))

  quantized_signal = np.round(analog_input * step_size)
  return quantized_signal


def dm(analog_input, step_size=1):
  # DM logic
  delta_modulated = np.zeros(len(analog_input))
  delta_modulated[0] = 0  # Set initial delta modulated value

  for i in range(1, len(analog_input)):
```

```
        delta_modulated[i] = 1 if analog_input[i] > delta_modulated[i - 1] else -1

    return delta_modulated
def signal_generator():
    user_input = input("Enter 'analog' or 'digital' for input: ")

    if user_input == 'digital':
        encoding_choice = input("Choose encoding technique (NRZ-L, NRZ-I, Manchester, Differential Manchester,
AMI, HDB3,B8ZS): ")

        if encoding_choice == 'NRZ-L':
            digital_data = input("Enter digital data: ")
            encoded_signal = nrz_l(digital_data)
            print("NRZ-L Encoded Signal:", encoded_signal)

        elif encoding_choice == 'NRZ-I':
            digital_data = input("Enter digital data: ")
            encoded_signal = nrz_i(digital_data)
            print("NRZ-I Encoded Signal:", encoded_signal)

        elif encoding_choice == 'Manchester':
            digital_data = input("Enter digital data for Manchester encoding: ")
            encoded_signal = manchester(digital_data)
            print("Manchester Encoded Signal:", encoded_signal)

        elif encoding_choice == 'Differential Manchester':
            digital_data = input("Enter digital data for Differential Manchester encoding: ")
            encoded_signal = differential_manchester(digital_data)
            print("Differential Manchester Encoded Signal:", encoded_signal)
```

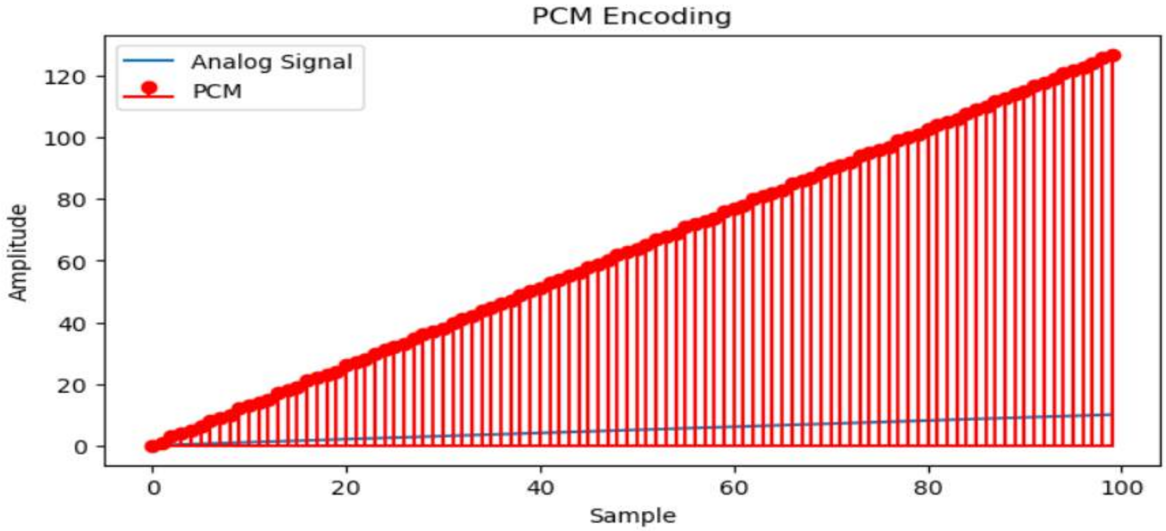
```
    elif encoding_choice == 'AMI':
        digital_data = input("Enter digital data for AMI encoding: ")
        encoded_signal = ami(digital_data)
        print("AMI Encoded Signal:", encoded_signal)

    elif encoding_choice in ['B8ZS', 'HDB3']:
        data = input(f"Enter data for {encoding_choice} scrambling: ")
        scrambled_data = apply_scrambling(data, encoding_choice)
        print(f"{encoding_choice} Scrambled Data:", scrambled_data)

    # Implement other encoding schemes

    # Add PCM/DM logic
else:
    # Apply PCM/DM on analog input
    analog_input = np.linspace(0, 10, 100)  # Sample analog signal (replace with your data)

    pcm_dm_choice = input("Choose PCM or DM: ")
    if pcm_dm_choice.lower() == 'pcm':
        quantized_signal = pcm(analog_input)
        plt.figure(figsize=(8, 4))
        plt.plot(analog_input, label='Analog Signal')
        plt.stem(quantized_signal, linefmt='r-', markerfmt='ro', basefmt='r-', label='PCM')
        plt.title('PCM Encoding')
        plt.xlabel('Sample')
        plt.ylabel('Amplitude')
        plt.legend()
        plt.show()

    elif pcm_dm_choice.lower() == 'dm':
```

```
        delta_modulated = dm(analog_input)
        plt.figure(figsize=(8, 4))
        plt.plot(analog_input, label='Analog Signal')
        plt.step(range(len(delta_modulated)), delta_modulated, label='DM', where='mid')
        plt.title('Delta Modulation (DM)')
        plt.xlabel('Sample')
        plt.ylabel('Amplitude')
        plt.legend()
        plt.show()

    else:
        print("Invalid choice for PCM/DM")

signal_generator()
```

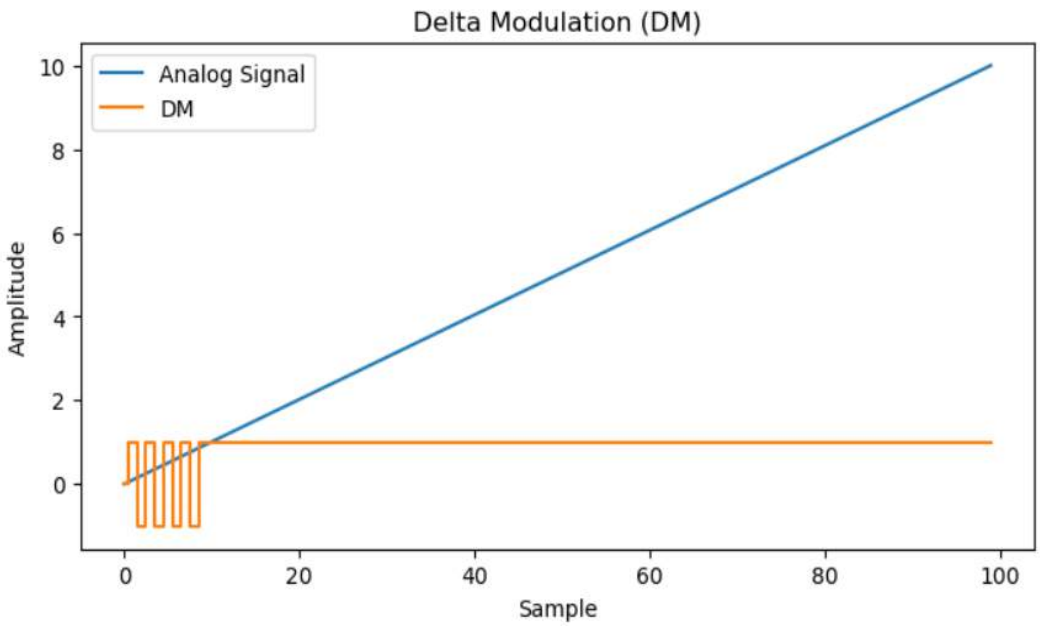
OUTPUT:

```
Enter 'analog' or 'digital' for input: digital
Choose encoding technique (NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, HDB3,B8ZS): NRZ-L
Enter digital data: 101000011
NRZ-L Encoded Signal: [1, -1, 1, -1, -1, -1, -1, 1, 1]

Enter 'analog' or 'digital' for input: digital
Choose encoding technique (NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, HDB3,B8ZS): NRZ-I
Enter digital data: 10111100
NRZ-I Encoded Signal: [1, 1, -1, -1, -1, -1, -1, 1, -1]
```

```
Enter 'analog' or 'digital' for input: analog
Choose PCM or DM: PCM
```




```
Enter 'analog' or 'digital' for input: analog
Choose PCM or DM: DM
```



```
Enter 'analog' or 'digital' for input: digital
Choose encoding technique (NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, HDB3): HDB3
Enter data for HDB3 scrambling: 101100011
HDB3 Scrambled Data: [1, 0, 0, -1, 0, -1, -1, 0, 0, 0, -1, -1]
```

```
Enter 'analog' or 'digital' for input: digital
Choose encoding technique (NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, HDB3,B8ZS): AMI
Enter digital data for AMI encoding: 101111000
AMI Encoded Signal: [0, 0, 0, 0, 0, 0, 0, 0, 0]
```