

Signal Encoding Scheme Simulations

Introduction

In modern communication systems, transferring data from one point to another requires encoding techniques to represent digital information accurately. This project focuses on implementing various encoding schemes: NRZ-L, NRZ-I, Bipolar AMI, Manchester, and Differential Manchester.

Project overview

This project consists of a Python script capable of encoding input data using different encoding schemes. The Implemented encoding functions ensure the conversion of binary data to corresponding electrical signals, reflecting distinctive voltage patterns characteristics for each scheme.

Implemented Encoding Schemes

- **NRZ-L:** This scheme encodes binary data by maintaining a constant voltage level for each bit, representing '0' as a low voltage level and '1' as a high voltage level.

```
# Function to encode data using NRZ-L encoding scheme
def encode_NRZ_L(data):
    encoded_signal = ""
    for bit in data:
        if bit == '0':
            encoded_signal += '|-V'
        else:
            encoded_signal += '|+V'
    return encoded_signal + '|'
```

- **NRZ-I:** This scheme encodes data by toggling the voltage level whenever a '1' bit is encountered, while maintaining the voltage level for '0' bits.

```
# Function to encode data using NRZ-I encoding scheme
def encode_NRZ_I(data):
    encoded_signal = ""
    prev_voltage = '-V'
    for bit in data:
        if bit == '0':
            encoded_signal += '|' + prev_voltage
        else:
            prev_voltage = '-V' if prev_voltage == '+V' else '+V'
            encoded_signal += '|' + prev_voltage
    return encoded_signal + '|'
```

- **Bipolar AMI:** In this scheme, '0' bits are represented by zero voltage, while '1' bits are alternately represented by positive and negative voltages, ensuring synchronization and error detection capabilities.

```
# Function to encode data using Bipolar AMI encoding scheme
def encode_Bipolar_AMI(data):
    encoded_signal = ""
    voltage = '+V'
    for bit in data:
        if bit == '0':
            encoded_signal += '|0V'
        else:
            encoded_signal += '|' + voltage
            voltage = '-V' if voltage == '+V' else '+V'
    return encoded_signal + '|'
```

- **Manchester Encoding:** Manchester encoding involves dividing each bit period into two halves and representing '0' bits with a high-to-low voltage transition and '1' bits with a low-to-high transition, ensuring frequent level changes for clock synchronization.

```
# Function to encode data using Manchester encoding scheme
def encode_Manchester(data):
    encoded_signal = ""
    for bit in data:
        if bit == '0':
            encoded_signal += '|+V|-V'
        else:
            encoded_signal += '|-V|+V'
    return encoded_signal
```

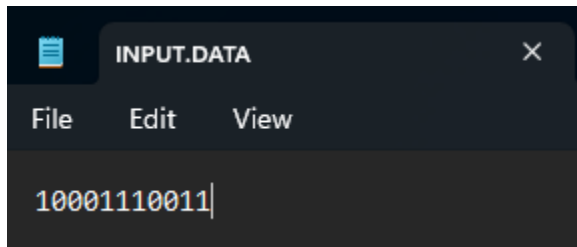
- **Differential Manchester Encoding:** This scheme ensures synchronization by using transitions in the middle of each bit period. '0' bits are represented by a transition at the beginning of the period, while '1' bits are indicated by no transition at the beginning but a transition in the middle.

```
# Function to encode data using Differential Manchester encoding scheme
def encode_Differential_Manchester(data):
    encoded_signal = ""
    prev_state = '+V'
    for bit in data:
        if bit == '0':
            encoded_signal += prev_state + '|'
            prev_state = '-V' if prev_state == '+V' else '+V'
        else:
            encoded_signal += '|' + prev_state
    return encoded_signal
```

Project Execution

NRZ-L:

```
C:\Users\johnn\Desktop>python encode.py INPUT.DATA 0  
Encoded signal saved to OUTPUT.SIGNAL
```



INPUT.DATA

File Edit View

10001110011|



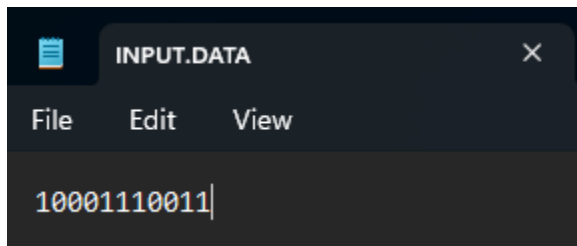
OUTPUT.SIGNAL

File Edit View

||+V|-V|-V|-V|+V|+V|+V|-V|-V|+V|+V|

NRZ-I:

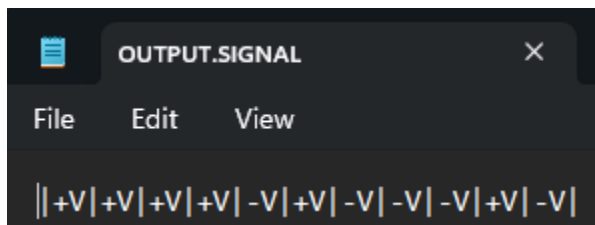
```
C:\Users\johnn\Desktop>python encode.py INPUT.DATA 1  
Encoded signal saved to OUTPUT.SIGNAL
```



INPUT.DATA

File Edit View

10001110011|



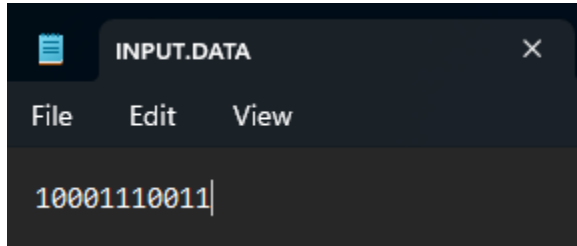
OUTPUT.SIGNAL

File Edit View

||+V|+V|+V|+V|-V|+V|-V|-V|-V|+V|-V|

B-AMI:

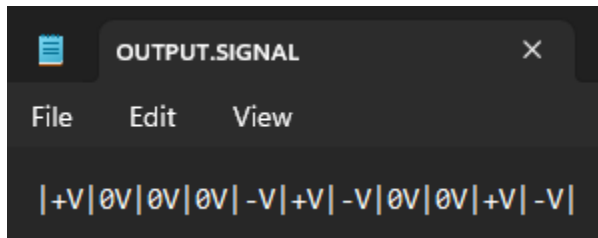
```
C:\Users\johnn\Desktop>python encode.py INPUT.DATA 2  
Encoded signal saved to OUTPUT.SIGNAL
```



INPUT.DATA

File Edit View

10001110011|



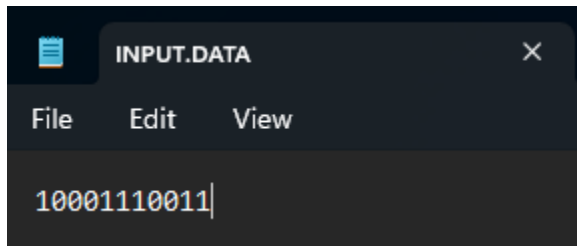
OUTPUT.SIGNAL

File Edit View

|+V|0V|0V|0V| -V|+V| -V|0V|0V|+V| -V|

Manchester:

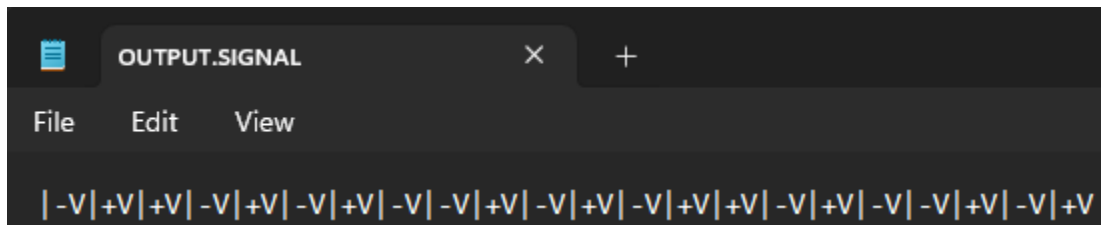
```
C:\Users\johnn\Desktop>python encode.py INPUT.DATA 3  
Encoded signal saved to OUTPUT.SIGNAL
```



INPUT.DATA

File Edit View

10001110011|



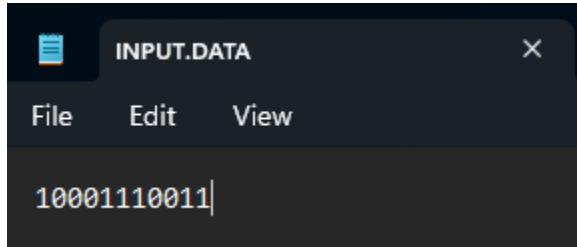
OUTPUT.SIGNAL

File Edit View

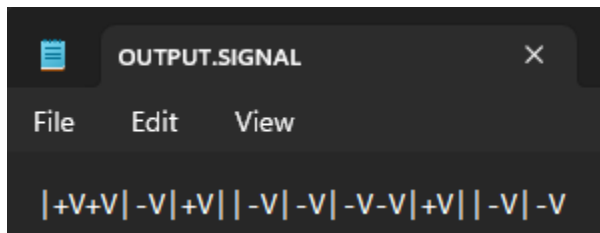
| -V|+V|+V| -V|+V| -V|+V| -V| -V|+V| -V|+V| -V|+V|+V| -V|+V| -V| -V|+V| -V|+V|

D-Manchester:

```
C:\Users\johnn\Desktop>python encode.py INPUT.DATA 4  
Encoded signal saved to OUTPUT.SIGNAL
```



A screenshot of a text editor window titled "INPUT.DATA". The window has a menu bar with "File", "Edit", and "View". The text area contains the binary sequence "10001110011" followed by a cursor.



A screenshot of a text editor window titled "OUTPUT.SIGNAL". The window has a menu bar with "File", "Edit", and "View". The text area contains the encoded signal sequence "|+V+V| -V|+V| | -V| -V| -V-V|+V| | -V| -V".

Conclusion

In conclusion, the successful implementation and simulation of various encoding schemes, including NRZ-L, NRZ-I, Bipolar AMI, Manchester, and Differential Manchester, have been achieved in this project. Through Python scripting, each scheme's distinctive characteristics in converting binary data into electrical signals have been demonstrated. This project not only enhances understanding of signal encoding techniques but also lays a foundation for further exploration and application in modern communication systems.