Project Report

Rajes Manna
2021BITE063

November 2023

1 Language and Libraries used:

Python:
1. Matplotlib
2. Numpy
3. Tkinter
4. Random and Timedata

2 Description:

In this project, a user-friendly interface (GUI) was developed for converting and encoding signals, including Pulse Code Modulation (PCM). The tool, created in Python, supports various signal encoding techniques like NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and PCM. The GUI, designed with Tkinter, enables users to visually observe signal representations using Matplotlib. An engaging feature of this project is the incorporation of animated visuals, enhancing the interactivity of the tool for users.

3 Functions are used in this project:

Line coding schemes :

1: Polar NRZ-L:

```
def polar_nrz_l(inp):
    inp1 = list(inp)
    inp1 = [-1 if i == 0 else 1 for i in inp1]
    return inp1
```

2: Polar NRZ-I:

```python
def polar_nrz_i(inp):
    inp2 = list(inp)
    flag = False
    for i in range(len(inp2)):
        if inp2[i] == 1 and not flag:
            flag = True
            continue
        if flag and inp2[i] == 1:
            if inp2[i-1] == 0:
                inp2[i] = 1
                continue
            else:
                inp2[i] = 0
                continue
        if flag:
            inp2[i] = inp2[i-1]
    inp2 = [-1 if i == 0 else 1 for i in inp2]
    return inp2
```

3: Manchester

```python
def manches(inp):
    inp1 = list(inp)
    manches = []
    for i in inp1:
        if i == 1:
            manches.append(-1)
            manches.append(1)
        else:
            manches.append(1)
            manches.append(-1)
    return manches
```

4: AMI

```python
def AMI(inp):
    inp1 = list(inp)
    flag = False
    for i in range(len(inp1)):
        if inp1[i] == 1 and not flag:
            flag = True
            continue
        elif flag and inp1[i] == 1:
            inp1[i] = -1
            flag = False
    return inp1
```

5: Differencial Manchester

```
def Diff_manchester(inp):
    li = []
    if inp[0] == 1:
        li.append(-1)
        li.append(1)
    else:
        li.append(1)
        li.append(-1)
    for i in range(1, len(inp[1:])):
        if li[-1] == 1:
            if(inp[i] == 1):
                li.append(-1)
                li.append(1)
            else:
                li.append(1)
                li.append(-1)
        else:
            if(inp[i] == 1):
                li.append(1)
                li.append(-1)
            else:
                li.append(-1)
                li.append(1)
    return li
```

6: B8ZS

```
def B8ZS(inpt):
    inp = inpt[0:]
    r = []
    prev = 1
    count = 0
    for i in range(len(inp)):
        if inp[i] == 0:
            count = 1
            for j in range(1, 8):
                if i+j < len(inp):
                    if inp[i+j] == 0:
                        count += 1
                    else:
                        break
                else:
                    break
            if count == 8:
                for j in range(1, 8):
                    inp[i+j] = -1
                r.append(0)
                r.append(0)
                r.append(0)
```

```
                r.append(prev)
                prev = prev * -1
                r.append(prev)
                r.append(0)
                r.append(prev)
                prev = prev * -1
                r.append(prev)
                count = 0
            else:
                r.append(inp[i])
        elif inp[i] == 1:
            prev = inp[i]
            r.append(inp[i])
        else:
            continue
    return r
```

7: HDB3

```
def hdb3(inpt):
    inp = inpt[0:]
    r = []
    prev = 1
    count = 0
    parity = 0
    for i in range(len(inp)):
        if inp[i] == 0:
            count = 1
            for j in range(1, 4):
                if i+j < len(inp):
                    if inp[i+j] == 0:
                        count += 1
                    else:
                        break
                else:
                    break
            if count == 4:
                for j in range(1, 4):
                    inp[i+j] = -1
                if parity % 2 == 1:
                    r.append(0)
                    r.append(0)
                    r.append(0)
                    r.append(prev)
                    parity += 1
                else:
                    prev = prev * -1
                    r.append(prev)
                    r.append(0)
                    r.append(0)
```

```
                r.append(prev)
            count = 0
        else:
            r.append(inp[i])
    elif inp[i] == 1:
        parity += 1
        prev = inp[i]
        r.append(inp[i])
    else:
        continue
return r
```

Pulse code modulation (PCM):

```
sampling_rate = 20
quantization_bits = 3  # Number of bits for quantization
amp=4 # frequency of input signal

# Step 1: Sampling
x_continuous = np.linspace(0, amp * np.pi, sampling_rate)
y_continuous = np.sin(x_continuous)

# Sample the sine wave
sampled_x = np.linspace(0, amp * np.pi, sampling_rate)
sampled_y = np.sin(sampled_x)

quantized_values = np.round((sampled_y + 1) * ((2**quantization_bits - 1) / 2)).astype(int)
x_binary = np.arange(len(quantized_values) * quantization_bits)
binary_output = [int(bit) for val in quantized_values for bit in format(val, f'0{quantization
```

4 Resources Used:

1. YouTube (Learning Libraries)
2. ChatGPT (Syntax and Error Assistance)
3. LaTeX (Project Report Writing)

Github repo:
Before running this code on your system, please consult the README.md file