Ex2: analog to digital

Code:

```
import numpy as np
import matplotlib.pyplot as plt
def calculate_bitrate(bits, time):
    return bits / time
print("Signal Types")
print("1. Analog Signal")
print("2. Digital Signal")
choice = int(input("Enter choice (1 or 2): "))
bits = int(input("Enter number of bits: "))
time = float(input("Enter time in seconds: "))
bitrate = calculate_bitrate(bits, time)
if choice == 1:
    signal = "Analog Signal"
    x = np.linspace(0, 2*np.pi, 500)
    y = np.sin(2*x)
    plt.plot(x, y)
    plt.title("Analog Signal")
    plt.xlabel("Time")
    plt.ylabel("Amplitude")
    plt.grid()
elif choice == 2:
    signal = "Digital Signal"
    x = np.arange(8)
    y = [1, 0, 1, 1, 0, 1, 0, 1]
    plt.step(x, y, where="mid")
    plt.title("Digital Signal")
```

```
    plt.xlabel("Time")
    plt.ylabel("Bit")
    plt.ylim(-0.2, 1.2)
    plt.grid()
else:
    signal = "Unknown"
    print("Invalid Choice")
    exit()
plt.show()
print("\nSignal Type:", signal)
print("Bit Rate =", bitrate, "bps")
```

Example Output

Input:

```
Signal Types
1. Analog Signal
2. Digital Signal
Enter choice (1 or 2): 2
Enter number of bits: 1000
Enter time in seconds: 2
```

Graph:
(Digital step signal graph appears)

Console Output:

```
Signal Type: Digital Signal
Bit Rate = 500.0 bps
```

ex4: **:** Parity Checking

code:

```
def parity(data, t):
    ones = data.count('1')
    return '0' if (ones % 2 == 0 and t=="even") or (ones % 2 != 0 and t=="odd") else '1'
```

```
def check(data, t):
    ones = data.count('1')
    return ones % 2 == 0 if t=="even" else ones % 2 != 0

data = "10110101"

for t in ["even", "odd"]:
    p = parity(data, t)
    send = data + p

    print("\n", t.upper(), "PARITY")
    print("Original:", data)
    print("Parity Bit:", p)
    print("Transmit:", send)

    print("No Error:", check(send, t))

    r = list(send)
    r[3] = '1' if r[3]=='0' else '0'
    r = ''.join(r)

    print("With Error:", r)
    print("Check:", check(r, t))
```

output

EVEN PARITY

Original: 10110101

Parity Bit: 1

Transmit: 101101011

No Error: True

With Error: 101001011

Check: False

ODD PARITY

Original: 10110101

Parity Bit: 0

Transmit: 101101010

No Error: True

With Error: 101001010

Check: False

Cyclic Redundancy Check (CRC)

Code:

```
def crc(data, gen):
    d = list(data + '0'*(len(gen)-1))

    for i in range(len(data)):
        if d[i] == '1':
            for j in range(len(gen)):
                d[i+j] = str(int(d[i+j] != gen[j]))

    return ''.join(d[-(len(gen)-1):])

data = "1101001110"
gen = "1011"

rem = crc(data, gen)
send = data + rem
```

```
print("Original Data:", data)
print("Generator:", gen)
print("CRC:", rem)
print("Transmit:", send)

print("\nNo Error:", send)

err = list(send)
err[4] = '1' if err[4]=='0' else '0'
err = ''.join(err)

print("Single Error:", err)

err2 = list(send)
err2[2] = '1' if err2[2]=='0' else '0'
err2[7] = '1' if err2[7]=='0' else '0'
err2 = ''.join(err2)

print("Multiple Error:", err2)
```

Example Output

```
Original Data: 1101001110
Generator: 1011
CRC: 001
Transmit: 1101001110001
No Error: 1101001110001
Single Error: 1101101110001
Multiple Error: 1111001010001
```

Hamming Code

Code:

```
def ham(data):
    h = [0]*7
    h[2], h[4], h[5], h[6] = map(int, data)

    h[0]=(h[2]+h[4]+h[6])%2
    h[1]=(h[2]+h[5]+h[6])%2
    h[3]=(h[4]+h[5]+h[6])%2

    return ''.join(map(str,h))

data="1011"
code=ham(data)

print("Original:",data)
print("Codeword:",code)

print("\nNo Error:",code)

err=list(code)
err[2]='1' if err[2]=='0' else '0'
err=''.join(err)

print("Single Error:",err)
print("Corrected:",code)

err2=list(code)
err2[1]='1' if err2[1]=='0' else '0'
err2[5]='1' if err2[5]=='0' else '0'
err2=''.join(err2)
```

```
print("Multiple Error:",err2)
```

Output

Original: 1011

Codeword: 0110011

No Error: 0110011

Single Error: 0100011

Corrected: 0110011

Multiple Error: 0010001

Ex5:

Stop and wait

Code:

```
import random
def stop_wait(data):
    seq = 0
    for f in data:
        print("\nSend:", f, "Seq:", seq)
        if random.random() < 0.3:
            print("Frame Lost → Retransmit")
            continue
        print("Receiver: Received", f)
        if random.random() < 0.2:
            print("ACK Lost")
            continue
        print("ACK", seq, "Received")
        seq = 1 - seq
    print("\nTransmission Complete")
frames = ['A','B','C','D']
stop_wait(frames)
```

Example Output

Send: A Seq: 0

Receiver: Received A

ACK 0 Received

Send: B Seq: 1

Frame Lost → Retransmit

Send: C Seq: 1

Receiver: Received C

ACK Lost

Send: D Seq: 1

Receiver: Received D

ACK 1 Received

Transmission Complete

Sliding Window Protocol (Flow Control)

Code:

```
import random
import time
def sliding_window(frames, window):
    i = 0
    while i < len(frames):
        print("\nWindow:", frames[i:i+window])
        for j in range(i, min(i+window, len(frames))):
            print("Sender -> Frame", j, ":", frames[j])
            if random.random() < 0.3:
                print("Frame Lost → Retransmit Window")
                break
            if random.random() < 0.1:
                print("Frame Corrupted")
                break
            print("Receiver -> ACK", j)
            time.sleep(0.5)
```

```
        else:
            i += window
            continue
        print("Timeout... Sending Again")
    print("\nTransmission Successful")
data = ['P','Q','R','S','T','U']
sliding_window(data, 3)
```

Example Output

```
Window: ['P', 'Q', 'R']
Sender -> Frame 0 : P
Receiver -> ACK 0
Sender -> Frame 1 : Q
Receiver -> ACK 1
Sender -> Frame 2 : R
Frame Lost → Retransmit Window
Timeout... Sending Again
Window: ['P', 'Q', 'R']
Sender -> Frame 0 : P
Receiver -> ACK 0
Sender -> Frame 1 : Q
Receiver -> ACK 1
Sender -> Frame 2 : R
Receiver -> ACK 2
Window: ['S', 'T', 'U']
...
Transmission Successful
```

Bit Stuffing

Code

```
def stuff(data):
    s = ""
    c = 0
```

```
    for i in data:
        s += i
        if i == '1':
            c += 1
            if c == 5:
                s += '0'
                c = 0
        else:
            c = 0
    return "01111110" + s + "01111110"
def unstuff(data):
    data = data[8:-8]
    return data.replace("111110", "11111")
data = "011111111110011111101"
print("Original :", data)
sf = stuff(data)
print("Stuffed  :", sf)
us = unstuff(sf)
print("Unstuffed:", us)
print("Match    :", data == us)
```

Example Output

```
Original : 011111111110011111101
Stuffed                                                                    :
01111110011111011111000111110101011111110
Unstuffed:
011111111110011111101
Match :
True
```

Character Stuffing (Byte Stuffing)

Code:

```
def stuff(data):
    out = []
    for i in data:
        if i == "DLE" or i == "DLESTX":
            out.append("DLE")
        out.append(i)
    return ["DLESTX"] + out + ["DLESTX"]
def unstuff(data):
    data = data[1:-1]
    out = []
    i = 0
    while i < len(data):
        if data[i] == "DLE":
            i += 1
        out.append(data[i])
        i += 1
    return out
data = ['A','B','DLE','C','DLESTX','D']
print("Original :", data)
sf = stuff(data)
print("Stuffed  :", sf)
uf = unstuff(sf)
print("Unstuffed:", uf)
print("Match    :", data == uf)
```

Example Output

Original :

['A', 'B', 'DLE', 'C', 'DLESTX', 'D']
Stuffed :

['DLESTX', 'A', 'B', 'DLE', 'DLE', 'C',

'DLE', 'DLESTX', 'D', 'DLESTX']

Unstuffed :

['A', 'B', 'DLE', 'C', 'DLESTX', 'D']

Match :

True