Encoding (Unipolar, NRZ-L, NRZ-I, RZ, Manchester, Differential Manchester)

```
import matplotlib.pyplot as plt

def plot_encoding(encoding, title):
    plt.step(range(len(encoding)), encoding, where='post', linewidth=2)
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('Voltage')
    plt.ylim(-1.5, 1.5)
    plt.grid()
    # plt.show()


#Unipolar 1 - Positive Voltage &  0 - Negative Voltage
def unipolar(bits):
    encoding = []
    for bit in bits:
        if bit == '1':
            encoding.extend([1])
        else:
            encoding.extend([0])
    return encoding


# 1 -Transition & 0 -No transition
```

```
def nrz_i(bits):
    encoding = []
    voltage = 1
    for bit in bits:
        if bit == '1':
            voltage *= -1
        encoding.extend([voltage]*2)
    return encoding

# 1 - negative voltage & 0 - Positive Voltage
def nrz_l(bits):
    encoding = []
    voltage = 1
    for bit in bits:
        if bit == '1':
            voltage = -1
        else:
            voltage = 1
        encoding.extend([voltage])
    return encoding

# 1 - positive to zero & 0 - negative to zero
def rz(bits):
    encoding = []
```

```
    for bit in bits:
        if bit == '0':
            encoding.extend([-1,-1,0,0])
        else:
            encoding.extend([1,1,0,0])
    return encoding

# 1 - Positive to negative & 0 - Negative to positive (Dr. Thomas)
def manchester(bits):
    encoding = []
    for bit in bits:
        if bit == '1':
            encoding.extend([1,-1])
        else:
            encoding.extend([-1,1])
    return encoding

# 1 - No transition & 0 - Transition
def differential_manchester(bits):
    encoding = []
    voltage = 1
    for bit in bits:
        if bit == '1':
            voltage *= -1
```

```
            encoding.extend([-voltage, voltage])
        else:
            encoding.extend([voltage, -voltage])
    return encoding



if __name__ == "__main__":
    bits = input("Enter bit: ")

    plt.subplot(2,3,1)
    unipolar_encoding = unipolar(bits)
    plot_encoding(unipolar_encoding,"Unipolar Encoding")

    plt.subplot(2,3,2)
    nrz_i_encoding = nrz_i(bits)
    plot_encoding(nrz_i_encoding, "Polar NRZ-I Encoding")

    plt.subplot(2,3,3)
    nrz_l_encoding = nrz_l(bits)
    plot_encoding(nrz_l_encoding, "Polar NRZ-L Encoding")

    plt.subplot(2,3,4)
    rz_encoding = rz(bits)
    plot_encoding(rz_encoding, "Polar RZ Encoding")
```

```
plt.subplot(2,3,5)
manchester_encoding = manchester(bits)
plot_encoding(manchester_encoding, "Manchester Encoding")

plt.subplot(2,3,6)
diff_manchester_encoding = differential_manchester(bits)
plot_encoding(diff_manchester_encoding, "Differential Manchester Encoding")

plt.tight_layout()
plt.show()
```

Modulating (AM, FM, PM)

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

def am_modulation(message, carrier_frequency, modulation_index, time):
    message_signal = np.sin(2 * np.pi * message * time)
    carrier_signal = np.sin(2 * np.pi * carrier_frequency * time)
    am_signal = (1 + modulation_index * message_signal) * carrier_signal
    return am_signal

def pm_modulation(message, carrier_frequency, modulation_index, time):
    message_signal = np.sin(2 * np.pi * message * time)
    pm_signal = np.sin(2 * np.pi * carrier_frequency * time + modulation_index * message_signal)
    return pm_signal

def fm_modulation(message, carrier_frequency, modulation_index, time):
    message_signal = np.sin(2 * np.pi * message * time)
    fm_signal = np.sin(2 * np.pi * carrier_frequency * time + 2 * np.pi * modulation_index * np.cumsum(message_signal) / len(time))
    return fm_signal

# Example parameters
message_frequency = 2  # Hz
```

```
carrier_frequency = 20  # Hz
modulation_index_am = 0.5
modulation_index_pm = 1
modulation_index_fm = 5

# Time values
time = np.arange(0, 1, 0.001)

# Modulate the signals
am_signal = am_modulation(message_frequency, carrier_frequency,
modulation_index_am, time)
pm_signal = pm_modulation(message_frequency, carrier_frequency,
modulation_index_pm, time)
fm_signal = fm_modulation(message_frequency, carrier_frequency,
modulation_index_fm, time)

# Plotting
plt.figure(figsize=(12, 8))

# AM Modulation
plt.subplot(3, 1, 1)
plt.plot(time, am_signal)
plt.title('AM Modulation')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

```
# PM Modulation
plt.subplot(3, 1, 2)
plt.plot(time, pm_signal)
plt.title('PM Modulation')
plt.xlabel('Time')
plt.ylabel('Amplitude')

# FM Modulation
plt.subplot(3, 1, 3)
plt.plot(time, fm_signal)
plt.title('FM Modulation')
plt.xlabel('Time')
plt.ylabel('Amplitude')

plt.tight_layout()
plt.show()
```

CRC (Cyclic Redundancy Check)

```
def xor1(a, b):
    result = ""
    n = len(b)
    for i in range(1, n):
        if a[i] == b[i]:
            result += "0"
        else:
            result += "1"
    return result


def mod2div(dividend, divisor):
    pick = len(divisor)
    tmp = dividend[:pick]
    n = len(dividend)


    while pick < n:
        if tmp[0] == '1':
            tmp = xor1(divisor, tmp) + dividend[pick]
        else:
            tmp = xor1('0' * pick, tmp) + dividend[pick]
```

```
        pick += 1

    if tmp[0] == '1':
        tmp = xor1(divisor, tmp)
    else:
        tmp = xor1('0' * pick, tmp)

    return tmp

def encode_data(data, key):
    l_key = len(key)
    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)
    codeword = data + remainder
    print("Remainder : ", remainder)
    print("Encoded Data (Data + Remainder): ", codeword)

def receiver(data, key):
    curr_xor = mod2div(data[:len(key)], key)
    curr = len(key)

    while curr != len(data):
        if len(curr_xor) != len(key):
            curr_xor += data[curr]
```

```
            curr += 1
        else:
            curr_xor = mod2div(curr_xor, key)

    if len(curr_xor) == len(key):
        curr_xor = mod2div(curr_xor, key)

    if '1' in curr_xor:
        print("There is some error in data")
    else:
        print("Correct message received")

# Driver code
data = input("Inter Data: ")
key = "1101"
print("Sender side...")
encode_data(data, key)

print("\nReceiver side...")
receiver(data + mod2div(data + '0' * (len(key) - 1), key), key)
```

Hamming Code

```
def hamming_error_check(data):
    c1 = 0
    c2 = 0
    c3 = 0
    c4 = 0
    idx = 1
    ok = False
    for i in range(len(receve_data)):
        if i%2 == 0:
            c1 ^= receve_data[i]
        if i == idx:
            c2 ^= receve_data[i]
            if ok == False:
                idx += 1
                ok = True
            else:
                idx +=3
                ok = False
        if i >= 3:
            c3 ^= receve_data[i]
        if i >= 7:
```

```
        c4 ^= receve_data[i]

    c = c1 + 2*c2 + 4*c3 + 8*c4
    if c == 0:
        print("There is no error")
    else:
        print("Error detected at position:", len(receve_data)-c+1)
        print("Which is:", data[c-1],"\nIt should be:", 1 if data[c-1] == '0' else 0)

a = input("Enter bit transmitted data: ")
receve_data = []
for i in a:
    receve_data.append(int(i))

receve_data.reverse()
hamming_error_check(receve_data)
```