**Name:** Mitrajeet Golsangi

**Roll No:** 01

**PRN:** 12010484

**Division:** Ty-CS-B

**Batch:** B1

**TY. B. Tech.**

**CS3052: Computer Networks**

**Lab No: 3**

**Write a program in C++/JAVA to implement - Unipolar NRZ, Polar NRZ, NRZ Inverted, Bipolar Encoding, Manchester Encoding, Differential Manchester Encoding.**

**Code:**

*import* matplotlib.pyplot *as* plt

def unipolar(inp):

    inp1=list(inp)

    inp1.insert(0,0)

*return* inp1

def polar\_nrz\_l(inp):

    inp1=list(inp)

    inp1.insert(0,0)

    inp1=[-1 *if* i==0 *else* 1 *for* i *in* inp1]

*return* inp1

def polar\_nrz\_i(inp):

    inp2=list(inp)

    lock=False

*for* i *in* range(len(inp2)):

*if* inp2[i]==1 and not lock:

            lock=True

*continue*

*if* lock and inp2[i]==1:

*if* inp2[i-1]==0:

                inp2[i]=1

*continue*

*else* :

                inp2[i]=0

*continue*

*if* lock:

            inp2[i]=inp2[i-1]

    inp2=[-1 *if* i==0 *else* 1 *for* i *in* inp2]

*return* inp2

def polar\_rz(inp):

    inp1=list(inp)

    inp1=[-1 *if* i==0 *else* 1 *for* i *in* inp1]

    li=[]

*for* i *in* range(len(inp1)):

        li.append(inp1[i])

        li.append(0)

*return* li

def Biphase\_manchester(inp):

    inp1=list(inp)

    li,init=[],False

*for* i *in* range(len(inp1)):

*if* inp1[i]==0:

            li.append(-1)

*if* not init:

                li.append(-1)

                init=True

            li.append(1)

*elif* inp1[i]==1 :

            li.append(1)

            li.append(-1)

*return* li

def Differential\_manchester(inp):

    inp1=list(inp)

    li,lock,pre=[],False,''

*for* i *in* range(len(inp1)):

*if* inp1[i]==0 and not lock:

            li.append(-1)

            li.append(-1)

            li.append(1)

            lock=True

            pre='S'

*elif* inp1[i]==1 and not lock :

            li.append(1)

            li.append(1)

            li.append(-1)

            lock=True

            pre='Z'

*else*:

*if* inp1[i]==0:

*if* pre=='S':

                    li.append(-1);li.append(1)

*else*:

                    li.append(1);li.append(-1)

*else*:

*if* pre=='Z':

                    pre='S'

                    li.append(-1);li.append(1)

*else*:

                    pre='Z'

                    li.append(1);li.append(-1)

*return* li

def AMI(inp):

    inp1=list(inp)

    inp1.insert(0,0)

    lock=False

*for* i *in* range(len(inp1)):

*if* inp1[i]==1 and not lock:

            lock=True

*continue*

*elif* lock and inp1[i]==1:

            inp1[i]=-1

            lock=False

*return* inp1

def plot(li):

    plt.subplot(7,1,1)

    plt.ylabel("Unipolar-NRZ")

    plt.plot(unipolar(li),color='red',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,2)

    plt.ylabel("P-NRZ-L")

    plt.plot(polar\_nrz\_l(li),color='blue',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,3)

    plt.ylabel("P-NRZ-I")

    plt.plot(polar\_nrz\_i(li),color='green',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,4)

    plt.ylabel("Polar-RZ")

    plt.plot(polar\_rz(li),color='red',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,5)

    plt.ylabel("B\_Man")

    plt.plot(Biphase\_manchester(li),color='violet',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,6)

    plt.ylabel("Dif\_Man")

    plt.plot(Differential\_manchester(li),color='red',drawstyle='steps-pre',marker='>')

    plt.subplot(7,1,7)

    plt.ylabel("A-M-I")

    plt.plot(AMI(li),color='blue',drawstyle='steps-pre',marker='>')

    plt.show()

*if* \_\_name\_\_=='\_\_main\_\_':

    print("Enter the size of Encoded Data : ")

    size=int(input())

    li=[]

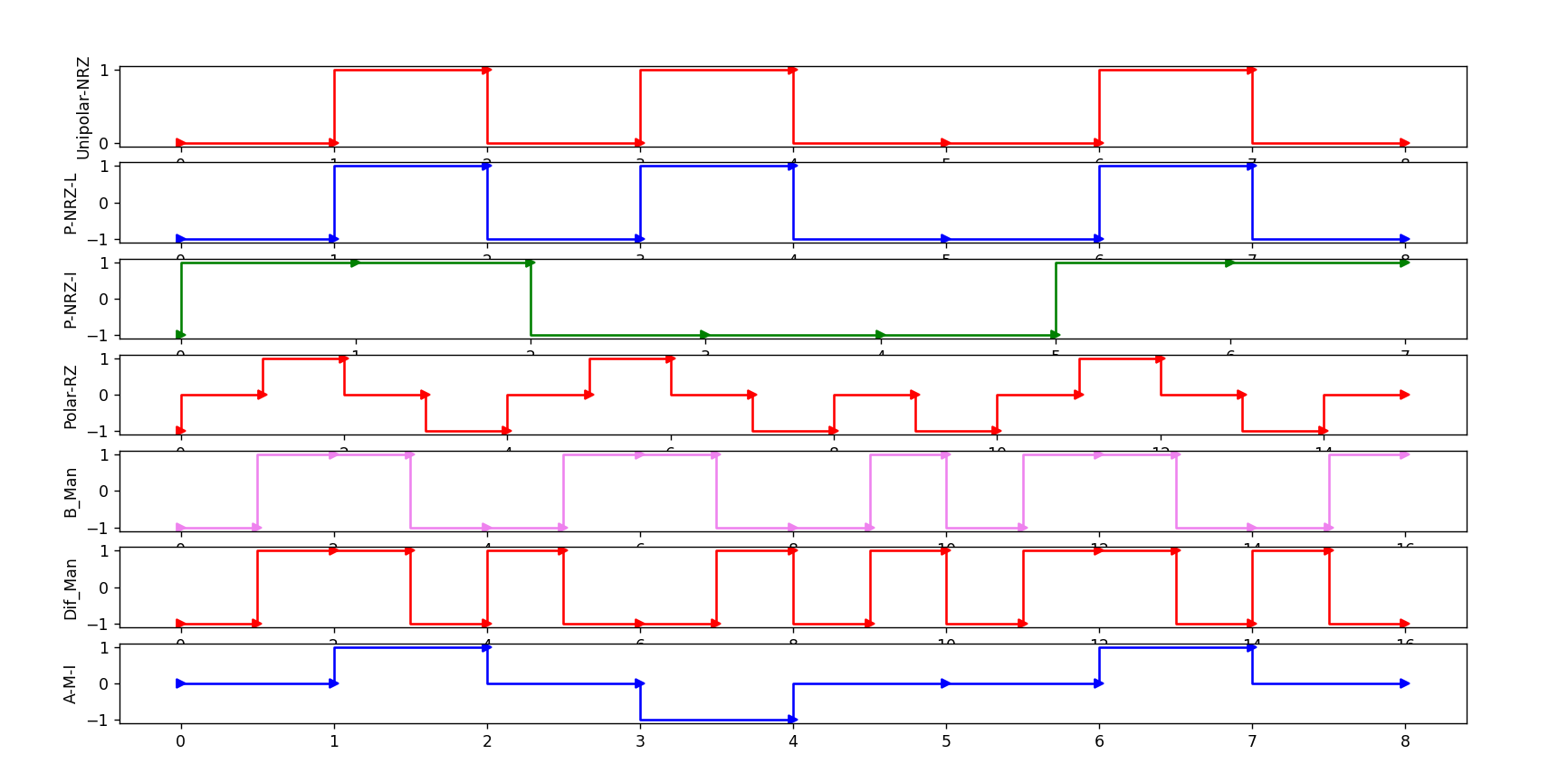
    print('Enter the binary bits sequnce of length ',size,' bits : \n')

*for* i *in* range(size):

        li.append(int(input()))

    plot(li)

**Output:**

****