EXP 3: IMPLEMENTATION OF LINE ENCODING SCHEMES

AIM

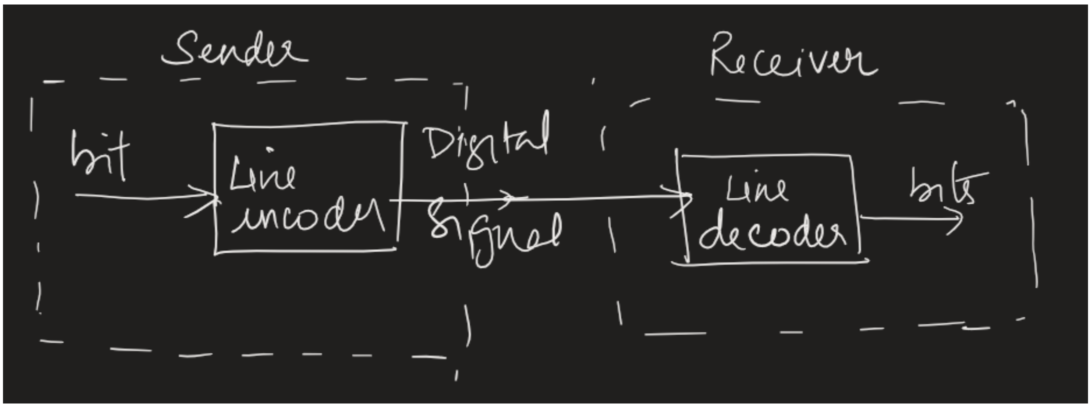
To implement the various line encoding schemes and to create a command line program and a GUI based web application for the same.

SOFTWARE USED

- **Command line program**
  - Python 3.2
  - Visual Studio Code
- **Web Application**
  - HTML
  - CSS
  - Javascript

THEORY

- Line coding is the process of converting digital data to digital signals.
- By this technique we converts a sequence of bits to a digital signal.
- At the sender side digital data are encoded into a digital signal and at the receiver side the digital data are recreated by decoding the digital signal



- Line Encoding Schemes can be divided into the following categories namely
  - Unipolar – NRZ
  - Polar
    - NRZ-L
    - NRZ-I
    - Bi-phase Manchester
    - Differential Manchester
  - Bipolar
    - AMI
    - Pseudoternary

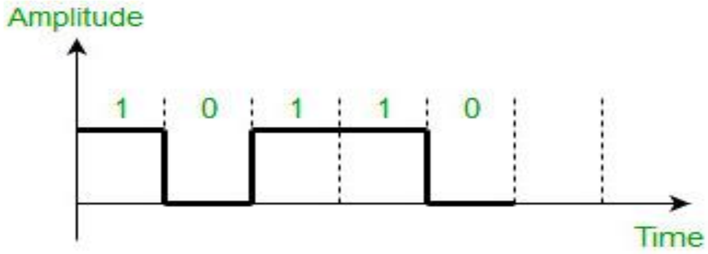
- The following are the desired characteristics of line encoding schemes
  - There should be **self-synchronizing** i.e., both receiver and sender clock should be synchronized
  - There should have some error-detecting capability
  - There should be immunity to noise and interference
  - There should be less complexity
  - There should be no low frequency component (**DC-component**) as long distance transfer is not feasible for low frequency component signal
  - There should be less base line wandering

UNIPOLAR-NRZ

It is unipolar line coding scheme in which positive voltage defines bit 1 and the zero voltage defines bit 0.

Signal does not return to zero at the middle of the bit thus it is called NRZ.

For example: Data = 10110.



But this scheme uses more power as compared to polar scheme to send one bit per unit line resistance. Moreover for continuous set of zeros or ones there will be self-synchronization and base line wandering problem.

POLAR-SCHEMES

In polar schemes, the voltages are on the both sides of the axis.

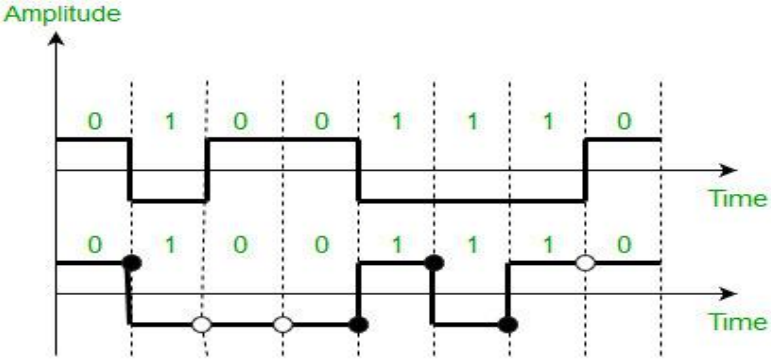
- **NRZ-L and NRZ-I**

  These are somewhat similar to unipolar NRZ scheme but here we use two levels of amplitude (voltages).

  For **NRZ-L (NRZ-Level)**, the level of the voltage determines the value of the bit, typically binary 1 maps to logic-level high, and binary 0 maps to logic-level low.

  For **NRZ-I (NRZ-Invert)**, two-level signal has a transition at a boundary if the next bit that we are going to transmit is a logical 1, and does not have a transition if the next bit that we are going to transmit is a logical 0.

Note – *For NRZ-I we are assuming in the example that previous signal before starting of data set “01001110” was positive. Therefore, there is no transition at the beginning and first bit “0” in current data set “01001110” is starting from +V. Example: Data = 01001110.*

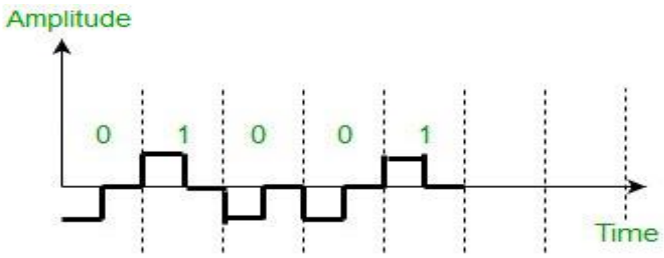


Comparison between NRZ-L and NRZ-I:

- Baseline wandering is a problem for both of them, but for NRZ-L it is twice as bad as compared to NRZ-I.
- This is because of transition at the boundary for NRZ-I (if the next bit that we are going to transmit is a logical 1).
- Similarly self-synchronization problem is similar in both for long sequence of 0’s, but for long sequence of 1’s it is more severe in NRZ-L.

- **Return to zero (RZ)**
  One solution to NRZ problem is the RZ scheme, which uses three values positive, negative, and zero. In this scheme signal goes to 0 in the middle of each bit.

Note – *The logic we are using here to represent data is that for bit 1 half of the signal is represented by +V and half by zero voltage and for bit 0 half of the signal is represented by -V and half by zero voltage. Example: Data = 01001.*



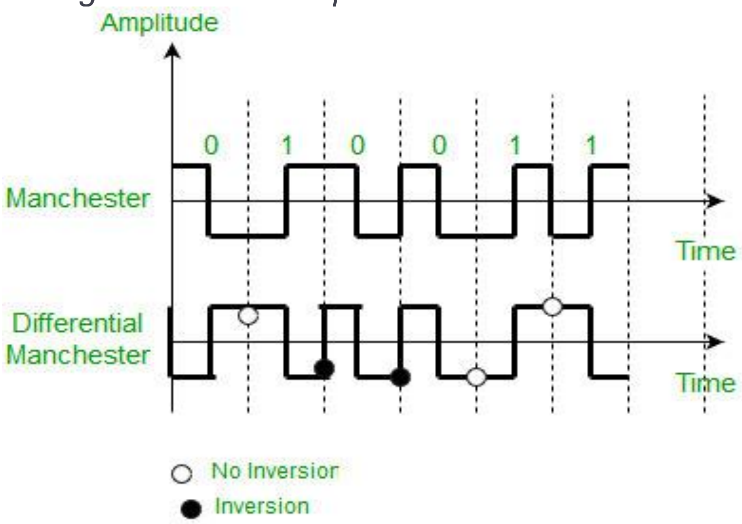
Disadvantages

- Main disadvantage of RZ encoding is that it requires greater bandwidth. Another problem is the complexity as it uses three levels of voltage.
- As a result of all these deficiencies, this scheme is not used today.
- Instead, it has been replaced by the better-performing Manchester and differential Manchester schemes.

- **Biphase (Manchester and Differential Manchester)**

  - **Manchester encoding** is somewhat combination of the RZ (transition at the middle of the bit) and NRZ-L schemes.
  - The duration of the bit is divided into two halves.
  - The voltage remains at one level during the first half and moves to the other level in the second half.
  - The transition at the middle of the bit provides synchronization.
  - **Differential Manchester** is somewhat combination of the RZ and NRZ-I schemes.
  - There is always a transition at the middle of the bit but the bit values are determined at the beginning of the bit.
  - If the next bit is 0, there is a transition, if the next bit is 1, there is no transition.

Note **–**
1. *The logic we are using here to represent data using Manchester is that for bit 1 there is transition form -V to +V volts in the middle of the bit and for bit 0 there is transition from +V to -V volts in the middle of the bit.*
2. *For differential Manchester we are assuming in the example that previous signal before starting of data set “010011” was positive. Therefore there is transition at the beginning and first bit “0” in current data set “010011” is starting from -V. Example: Data = 010011.*



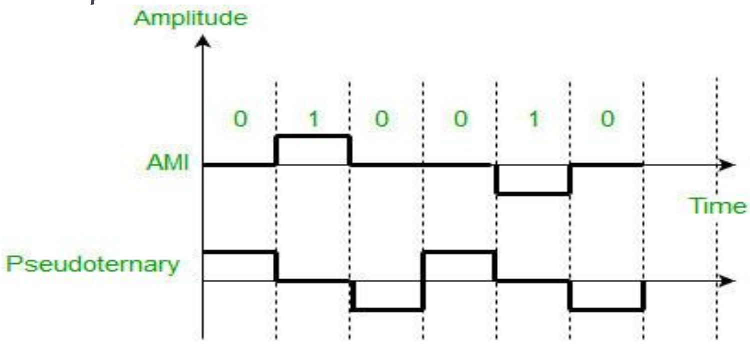
Problems solved

- The Manchester scheme overcomes several problems associated with NRZ-L, and differential Manchester overcomes several problems associated with NRZ-I as there is no baseline wandering and no DC component because each bit has a positive and negative voltage contribution.
- Only limitation is that the minimum bandwidth of Manchester and differential Manchester is twice that of NRZ.

Bipolar-schemes

In this scheme there are three voltage levels positive, negative, and zero. The voltage level for one data element is at zero, while the voltage level for the other element alternates between positive and negative.

- **Alternate Mark Inversion (AMI) –** A neutral zero voltage represents binary 0. Binary 1's are represented by alternating positive and negative voltages.
- **Pseudoternary –** Bit 1 is encoded as a zero voltage and the bit 0 is encoded as alternating positive and negative voltages i.e., opposite of AMI scheme.
- *Example:Data=010010*.



Note

The bipolar scheme is an alternative to NRZ.This scheme has the same signal rate as NRZ,but there is no DC component as one bit is represented by voltage zero and other alternates every time

IMPLEMENTATION *IN PYTHON*

UNIPOLAR – NRZ

Function definition

```
def unipolar(inp):
    inp1=list(inp)
    inp1.insert(0,0)
    return inp1
```

POLAR NRZ-L

Function definition

```
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
```

POLAR NRZ-I

Function definition

```
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
```

POLAR RZ

Function definition

```
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
```

BIPHASE MANCHESTER

Function Definition

```
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
```

DIFFERENTIAL MANCHESTER

Function Definition

```
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
```

ALERNATE MARK INVERSION

Function definition

```
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
```

PSEUDOTERNARY

Function definition

```
def pseudoternary(inp):
    inp1=list(inp)
    inp1.insert(0,0)
    flag=False
    for i in range(len(inp1)):
        if inp1[i]==0 and not flag:
            inp1[i]=-1
            flag=True
            continue
        elif flag and inp1[i]==0:
            inp1[i]=1
            flag=False
        elif inp1[i]==1 and not flag:
            inp1[i]=0
    return inp1
```

SOURCE CODE WITH DRIVER PROGRAM

```
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 plotall(li):
    plt.subplot(7,1,1)
    plt.ylabel("Unipolar-NRZ")
    plt.title("Unipolar -NRZ")
    plt.plot(unipolar(li),color='red',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,2)
    plt.ylabel("P-NRZ-L")
    plt.title("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.title("NRZ-I")
    plt.plot(polar_nrz_i(li),color='green',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,4)
    plt.ylabel("Polar-RZ")
    plt.title("Polar RZ")
    plt.plot(polar_rz(li),color='red',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,5)
    plt.ylabel("B_Man")
    plt.title("Manchester")
    plt.plot(Biphase_manchester(li),color='violet',drawstyle='steps-
pre',marker='>')
    plt.subplot(7,1,6)
    plt.ylabel("Dif_Man")
    plt.title("Differential Manchester")
    plt.plot(Differential_manchester(li),color='red',drawstyle='steps-
pre',marker='>')
    plt.subplot(7,1,7)
    plt.ylabel("A-M-I")
    plt.title("Alternate Mark Inversion")
    plt.plot(AMI(li),color='blue',drawstyle='steps-pre',marker='>')

    plt.show()

def plotunrz(li):
    plt.ylabel("Unipolar-NRZ")
    plt.plot(unipolar(li),color='red',drawstyle='steps-pre',marker='>')
```

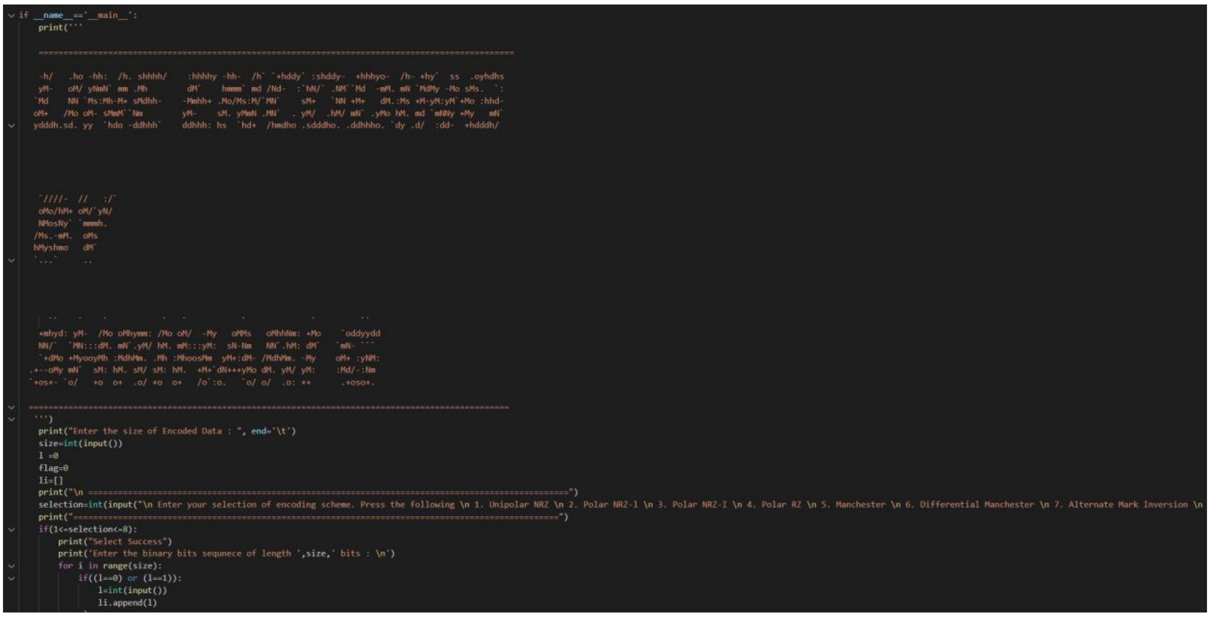
```
    plt.title("Unipolar -NRZ")
    plt.show()

def plotpnrzl(li):
    plt.ylabel("P-NRZ-L")
    plt.plot(polar_nrz_l(li),color='blue',drawstyle='steps-pre',marker='>')
    plt.title("NRZ-L")
    plt.show()
def plotnrzi(li):
    plt.ylabel("P-NRZ-I")
    plt.plot(polar_nrz_i(li),color='green',drawstyle='steps-pre',marker='>')
    plt.title("NRZ-I")
    plt.show()
def plotprz(li):
    plt.ylabel("Polar-RZ")
    plt.plot(polar_rz(li),color='red',drawstyle='steps-pre',marker='>')
    plt.title("Polar RZ")
    plt.show()
def plotbman(li):
    plt.ylabel("B_Man")
    plt.plot(Biphase_manchester(li),color='violet',drawstyle='steps-
pre',marker='>')
    plt.title("Manchester")
    plt.show()
def plotdifman(li):
    plt.ylabel("Dif_Man")
    plt.plot(Differential_manchester(li),color='red',drawstyle='steps-
pre',marker='>')
    plt.title("Differential Manchester")
    plt.show()
def plotami(li):
    plt.ylabel("A-M-I")
    plt.plot(AMI(li),color='blue',drawstyle='steps-pre',marker='>')
    plt.title("Alternate Mark Inversion")
    plt.show()









if __name__=='__main__':
    print('''

    ==========================================================================
======================
```

```
    -h/    .ho -hh:   /h. shhhh/     :hhhhy -
hh-   /h` `+hddy` :shddy-  +hhhyo-   /h- +hy`   ss   .oyhdhs
    yM-    oM/ yNmN` mm .Mh         dM`     hmmm` md /Nd-   :`hN/` .NM``Md   -
mM. mN `MdMy -Mo sMs.   `:
   `Md    NN `Ms:Mh-M+ sMdhh-     -
Mmhh+ .Mo/Ms:M/`MN`     sM+    `NN +M+    dM.:Ms +M-yM:yM`+Mo :hhd-
   oM+    /Mo oM- sMmM``Nm         yM-     sM. yMmN .MN`   . yM/   .hM/ mN` .yMo hM
. md `mNNy +My    mN`
   ydddh.sd. yy   `hdo -
ddhhh`     ddhhh: hs   `hd+  /hmdho .sdddho. .ddhhho. `dy .d/   :dd-   +hdddh/




    `////-  //    :/`

    oMo/hM+ oM/`yN/

    NMosNy` `mmmh.

   /Ms.-
mM.  oMs

   hMyshmo    dM`

   `...`      ..



      ``    `    `           `   `           `             `         ``

    +mhyd: yM-   /Mo oMhymm: /Mo oM/   -
My    oMMs    oMhhNm: +Mo     `oddyydd
```

```
    NN/`   `MN:::dM. mN`.yM/ hM. mM:::yM:  sN-
Nm   NN`.hM: dM`    `mN- ```
    `+dMo +MyooyMh :MdhMm. .Mh :MhoosMm  yM+:dM- /MdhMm. -
My    oM+ :yNM:
  .+--oMy mN`  sM: hM. sM/ sM: hM.  +M+`dN+++yMo dM. yM/ yM:    :Md/-
:Nm
  `+os+- `o/   +o  o+  .o/ +o  o+   /o`:o.   `o/ o/  .o: ++      .+oso+.

  ============================================================================
=====================
   ''')
    print("Enter the size of Encoded Data : ", end='\t')
    size=int(input())
    l =0
    flag=0
    li=[]
    print("\n ================================================================
=================================")
    selection=int(input("\n Enter your selection of encoding scheme. Press the
 following \n 1. Unipolar NRZ \n 2. Polar NRZ-l \n 3. Polar NRZ-
I \n 4. Polar RZ \n 5. Manchester \n 6. Differential Manchester \n 7. Alternat
e Mark Inversion \n 8. All \n===================================================
============================================= \n Enter your selection : "))
    print("===================================================================
===============================")
    if(1<=selection<=8):
        print("Select Success")
        print('Enter the binary bits sequnece of length ',size,' bits : \n')
        for i in range(size):
            if((l==0) or (l==1)):
                l=int(input())
                li.append(l)
            else:
                print("\n Invalid Input")
                flag=1
                break
        if(flag==0):
            if(selection==1):
                plotunrz(li)
            elif(selection==2):
                plotpnrzl(li)
            elif(selection==3):
                plotnrzi(li)
            elif(selection==4):
                plotprz(li)
            elif(selection==5):
                plotbman(li)
            elif(selection==6):
```

```
                plotdifman(li)
            elif(selection==7):
                plotami(li)
            elif(selection==8):
                plotall(li)
            print("\n Encoding Success")
        else:
            print("\n Enter only binary inputs. Try Again!")
    else:
        print("\n Enter a valid selection")
```

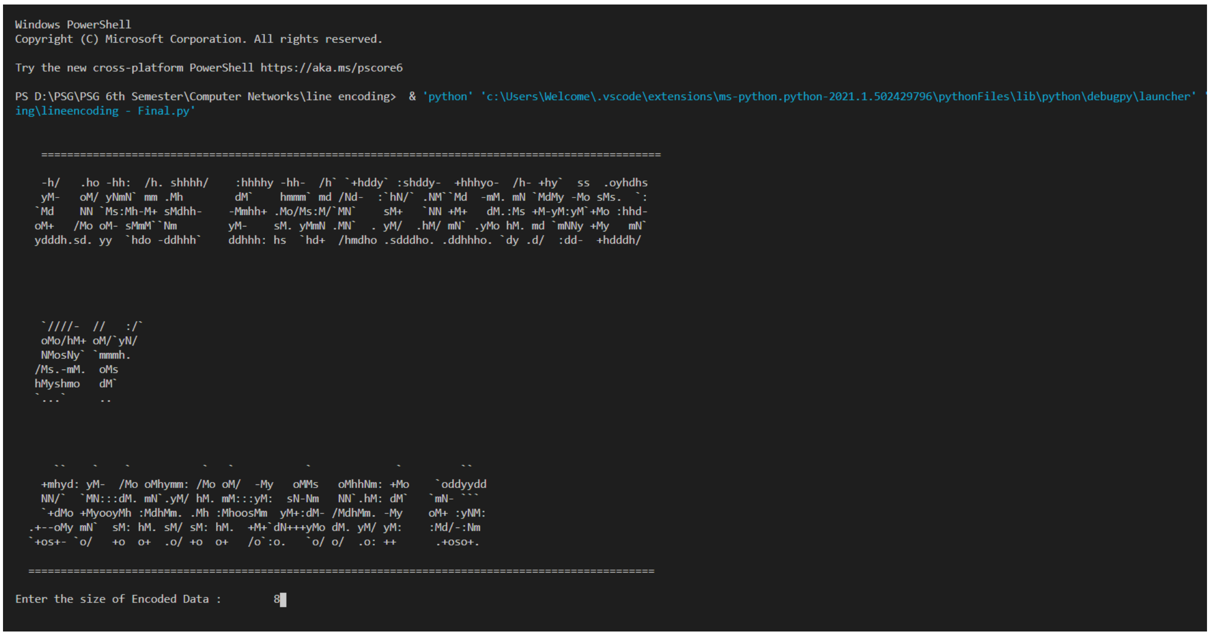
Screenshot of formatted test code



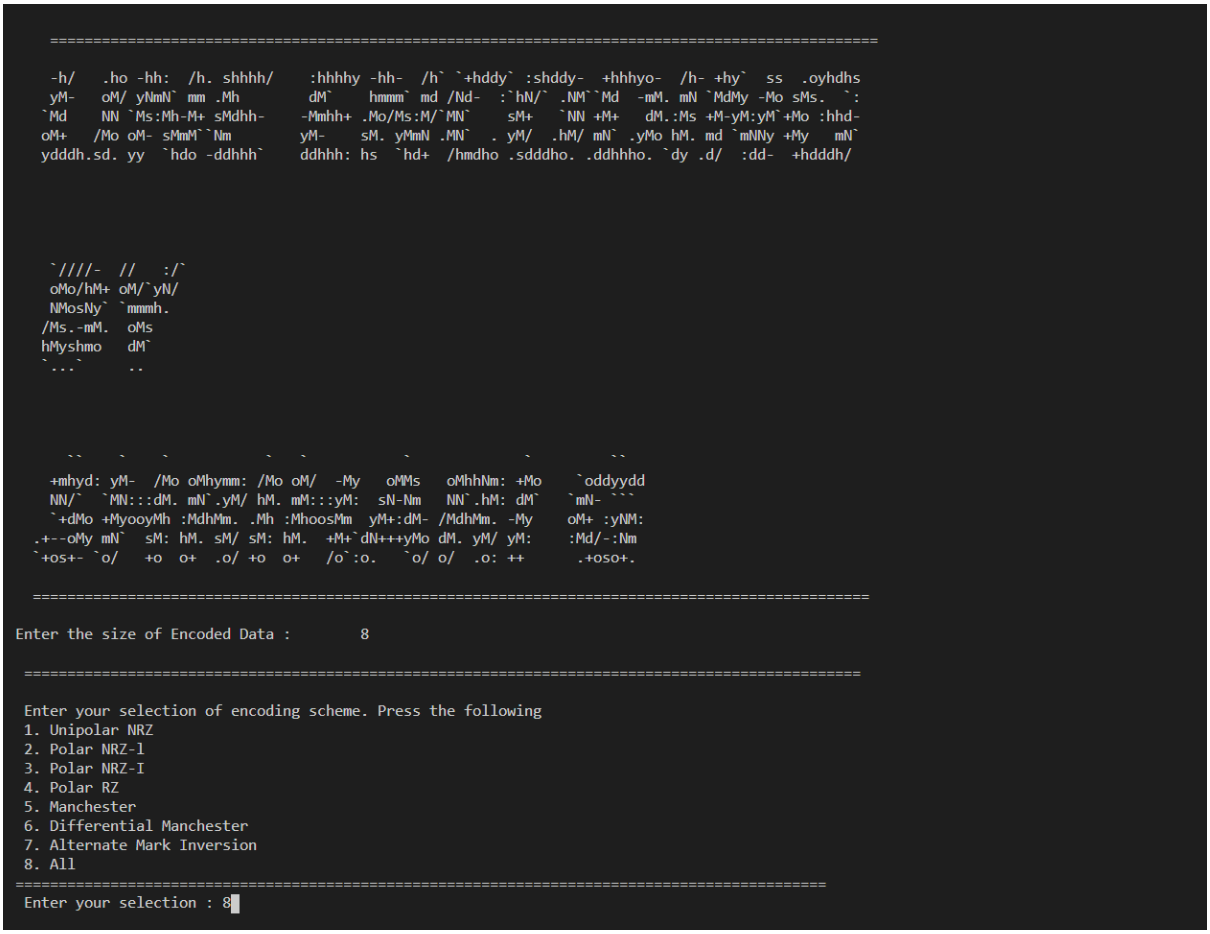
```
if __name__=='__main__':
    print('''
''')
    print("Enter the size of Encoded Data : ", end='\t')
    size=int(input())
    l =0
    flag=0
    li=[]
    print("\n ====================================================================================================")
    selection=int(input("\n Enter your selection of encoding scheme. Press the following \n 1. Unipolar NRZ \n 2. Polar NRZ-l \n 3. Polar NRZ-I \n 4. Polar RZ \n 5. Manchester \n 6. Differential Manchester \n 7. Alternate Mark Inversion \n
    print("====================================================================================================")
    if(1<=selection<=8):
        print("Select Success")
        print('Enter the binary bits sequnece of length ',size,' bits : \n')
        for i in range(size):
            if((l==0) or (l==1)):
                l=int(input())
                li.append(l)
```

DEMO

Entering size of data as 8 bits



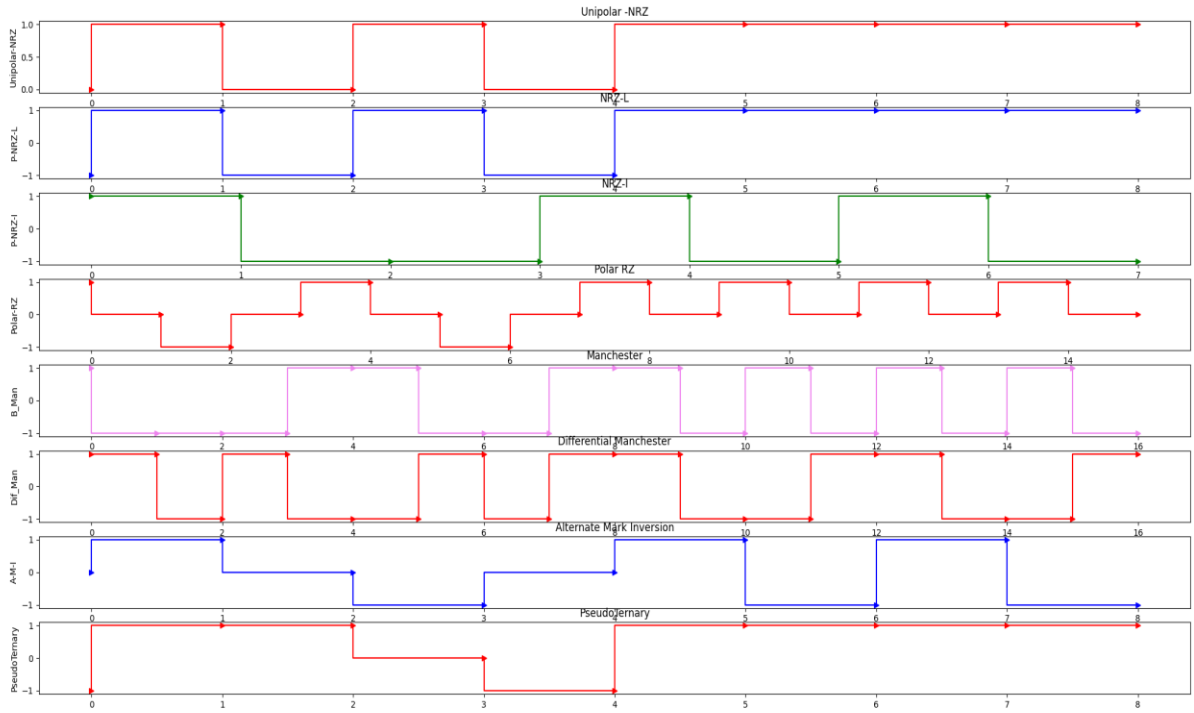
Selecting for option 8 – All encoding schemes



Entering data

```
     ``    `    `           `   `           `              `          ``
    +mhyd: yM-  /Mo oMhymm: /Mo oM/  -My   oMMs   oMhhNm: +Mo    `oddyydd
    NN/`   `MN:::dM. mN`.yM/ hM. mM:::yM:  sN-Nm   NN`.hM: dM`   `mN- ```
    `+dMo +MyooyMh :MdhMm. .Mh :MhoosMm  yM+:dM- /MdhMm. -My    oM+ :yNM:
  .+--oMy mN`   sM: hM. sM/ sM: hM.  +M+`dN+++yMo dM. yM/ yM:    :Md/-:Nm
  `+os+- `o/    +o  o+  .o/ +o  o+   /o`:o.   `o/ o/  .o: ++      .+oso+.

  ===================================================================================================

 Enter the size of Encoded Data :        8

  ===================================================================================================

  Enter your selection of encoding scheme. Press the following
  1. Unipolar NRZ
  2. Polar NRZ-l
  3. Polar NRZ-I
  4. Polar RZ
  5. Manchester
  6. Differential Manchester
  7. Alternate Mark Inversion
  8. All
 ===============================================================================================
  Enter your selection : 8
 ===================================================================================================
 Select Success
 Enter the binary bits sequnece of length  8  bits :

 1
 0
 1
 0
 1
 1
 1
 1
```

Output



EXCEPTION HANDLING

Case 1: Size of data stream is not an integer

```python
try:
    print("Enter the size of Encoded Data : ", end='\t')
    size=int(input())
except:
    print("The size must be an integer value")
    exit()
```

```
  ==================================================================================================

   -h/   .ho -hh:  /h. shhhh/    :hhhhy -hh-  /h` `+hddy` :shddy-  +hhhyo-  /h- +hy`  ss  .oyhdhs
   yM-   oM/ yNmN` mm .Mh        dM`    hmmm` md /Nd-  :`hN/` .NM``Md  -mM. mN `MdMy -Mo sMs.  `:
  `Md    NN `Ms:Mh-M+ sMdhh-    -Mmhh+ .Mo/Ms:M/`MN`    sM+   `NN +M+   dM.:Ms +M-yM:yM`+Mo :hhd-
  oM+   /Mo oM- sMmM``Nm        yM-    sM. yMmN .MN`  . yM/  .hM/ mN` .yMo hM. md `mNNy +My   mN`
  ydddh.sd. yy  `hdo -ddhhh`    ddhhh: hs  `hd+  /hmdho .sdddho. .ddhhho. `dy .d/  :dd-  +hdddh/



   `////-  //   :/`
   oMo/hM+ oM/`yN/
   NMosNy` `mmmh.
  /Ms.-mM.  oMs
  hMyshmo   dM`
  `...`     ..



     ``   `   `        `  `        `          `     ``
   +mhyd: yM-  /Mo oMhymm: /Mo oM/  -My   oMMs   oMhhNm: +Mo    `oddyydd
   NN/`  `MN:::dM. mN`.yM/ hM. mM:::yM:  sN-Nm   NN`.hM: dM`   `mN- ```
   `+dMo +MyooyMh :MdhMm. .Mh :MhoosMm  yM+:dM- /MdhMm. -My    oM+ :yNM:
 .+--oMy mN`  sM: hM. sM/ sM: hM.  +M+`dN+++yMo dM. yM/ yM:    :Md/-:Nm
 `+os+- `o/   +o  o+  .o/ +o  o+   /o`:o.   `o/ o/  .o: ++      .+oso+.

  ==================================================================================================

Enter the size of Encoded Data :        a
The size must be an integer value
PS D:\PSG\PSG 6th Semester\Computer Networks\line encoding>
```

Case 2: Invalid selection for encoding scheme

```
if(1<=selection<=8):
    print("Select Success")
    print('Enter the binary bits sequnece of length ',size,' bits : \n')
    for i in range(size):
        if((l==0) or (l==1)):
            l=int(input())
            li.append(l)
        else:
            print("\n Invalid Input")
            flag=1
            break
    if(flag==0):
        if(selection==1):
            plotunrz(li)
        elif(selection==2):
            plotpnrzl(li)
        elif(selection==3):
            plotnrzi(li)
        elif(selection==4):
            plotprz(li)
        elif(selection==5):
            plotbman(li)
        elif(selection==6):
            plotdifman(li)
        elif(selection==7):
            plotami(li)
        elif(selection==8):
            plotall(li)
        print("\n Encoding Success")
    else:
        print("\n Enter only binary inputs. Try Again!")
else:
    print("\n Enter a valid selection")
```

```
 ==================================================================================================

Enter the size of Encoded Data :        4

 ==================================================================================================

 Enter your selection of encoding scheme. Press the following
 1. Unipolar NRZ
 2. Polar NRZ-l
 3. Polar NRZ-I
 4. Polar RZ
 5. Manchester
 6. Differential Manchester
 7. Alternate Mark Inversion
 8. All
==============================================================================================
 Enter your selection : 34
==================================================================================================

 Enter a valid selection
PS D:\PSG\PSG 6th Semester\Computer Networks\line encoding>
```

Case 3: Input data is not binary

```python
for i in range(size):
    if((l==0) or (l==1)):
        try:
            l=int(input())
            li.append(l)
        except:
            print("\n Data stream must have only binary values")
    else:
```

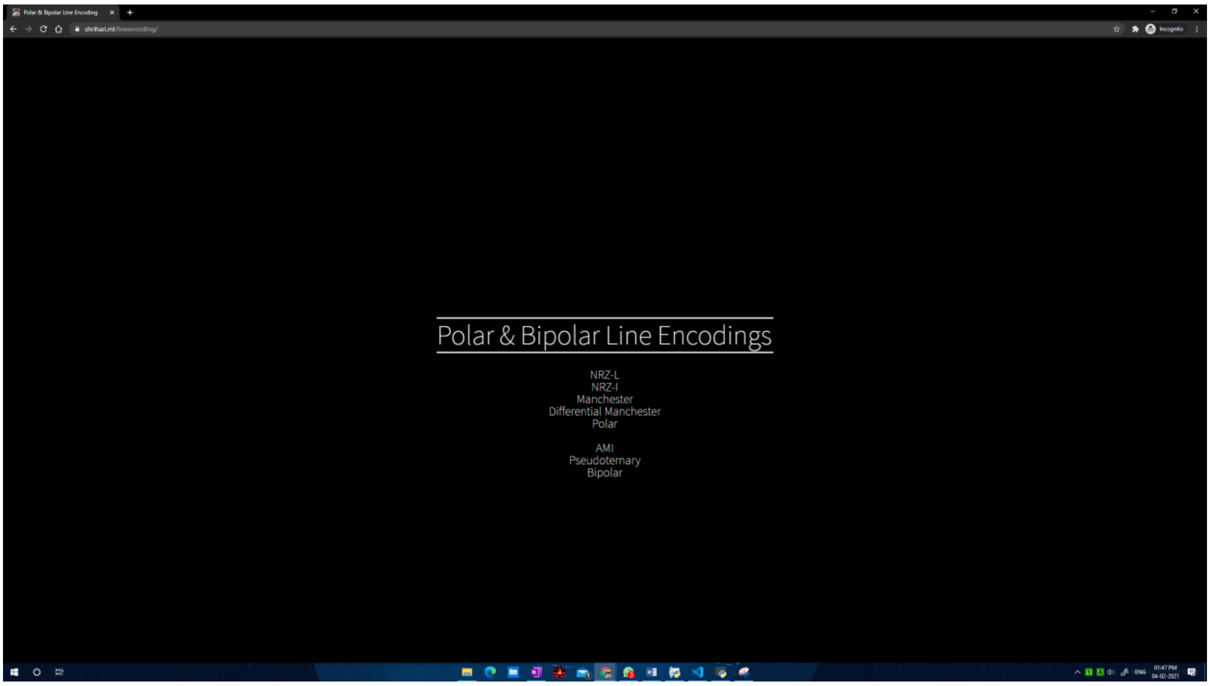
```
Enter the size of Encoded Data :        8

 ==================================================================================================

 Enter your selection of encoding scheme. Press the following
 1. Unipolar NRZ
 2. Polar NRZ-l
 3. Polar NRZ-I
 4. Polar RZ
 5. Manchester
 6. Differential Manchester
 7. Alternate Mark Inversion
 8. All
==============================================================================================
 Enter your selection : 8
==================================================================================================
Select Success
Enter the binary bits sequnece of length  8  bits :

1
a

 Data stream must have only binary values
```

IMPLEMENTATION WITH GUI AS WEB APPLICATION

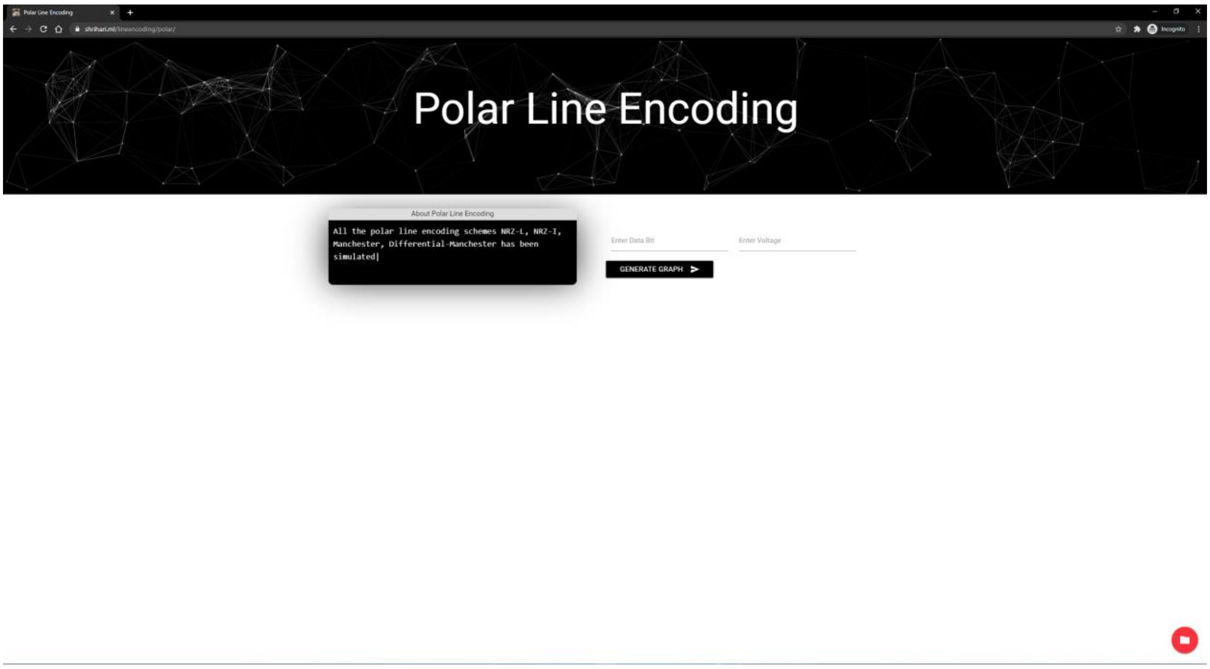
- *The line encoding schemes are implemented in the form of a GUI with a web interface*
- *The front end used is HTML and CSS (Bootstrap for themes)*
- *The back end used Is Javascript*
- *The web-application has been deployed on my web-server*

- *The source codes and the application can be found at*
    - ***https://shrihari.ml/lineencoding***

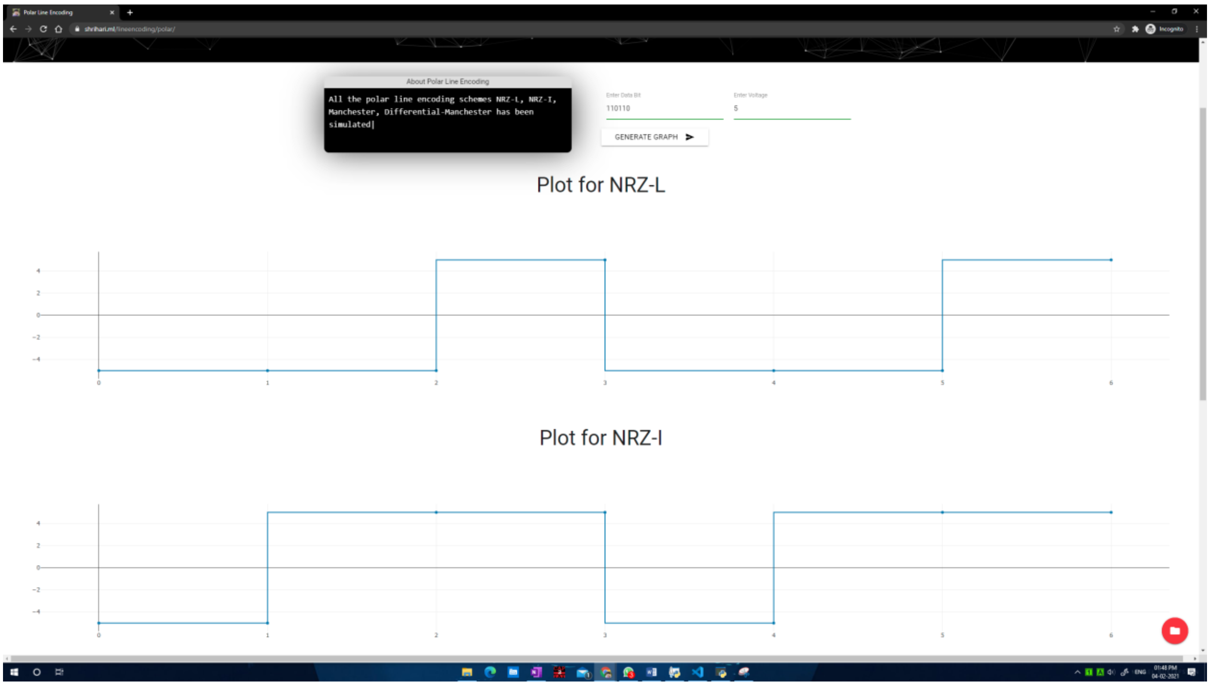
HOMEPAGE

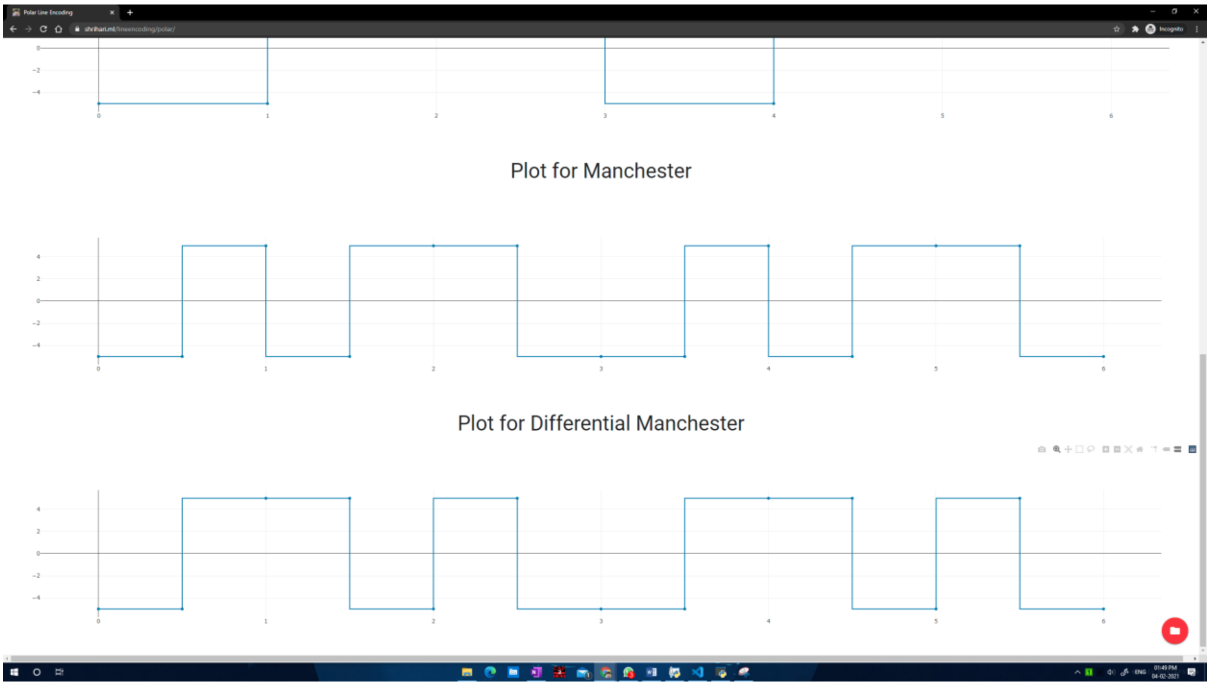


SELECTING FOR POLAR ENCODING SCHEMES

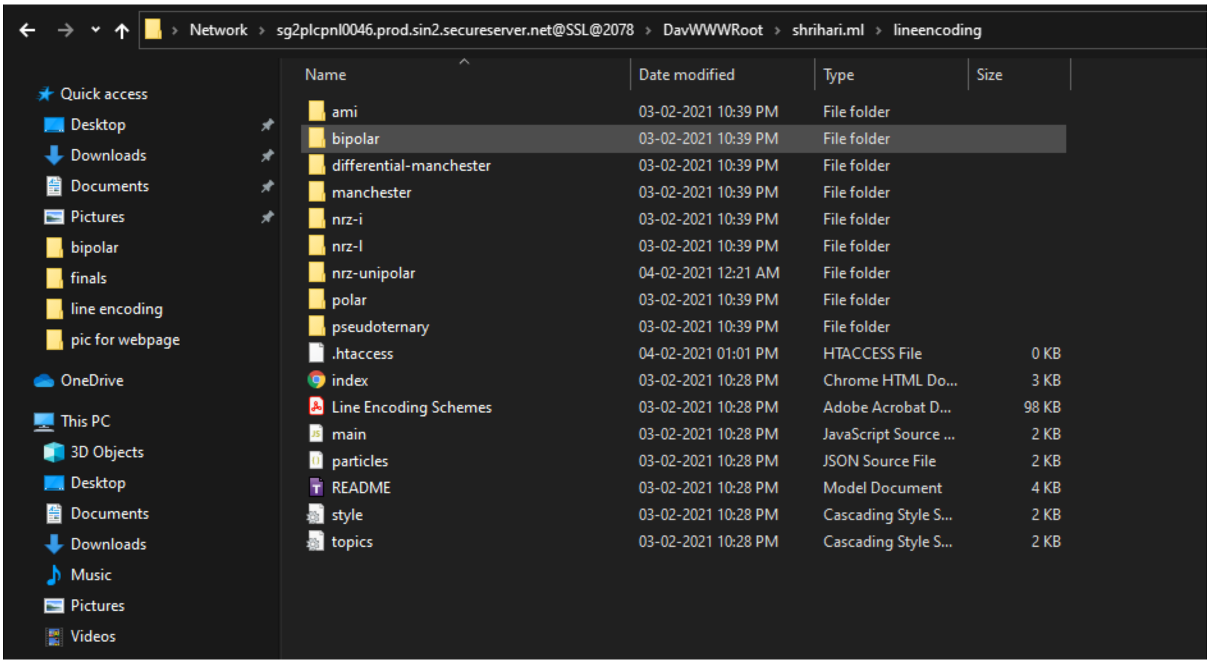


OUTPUT

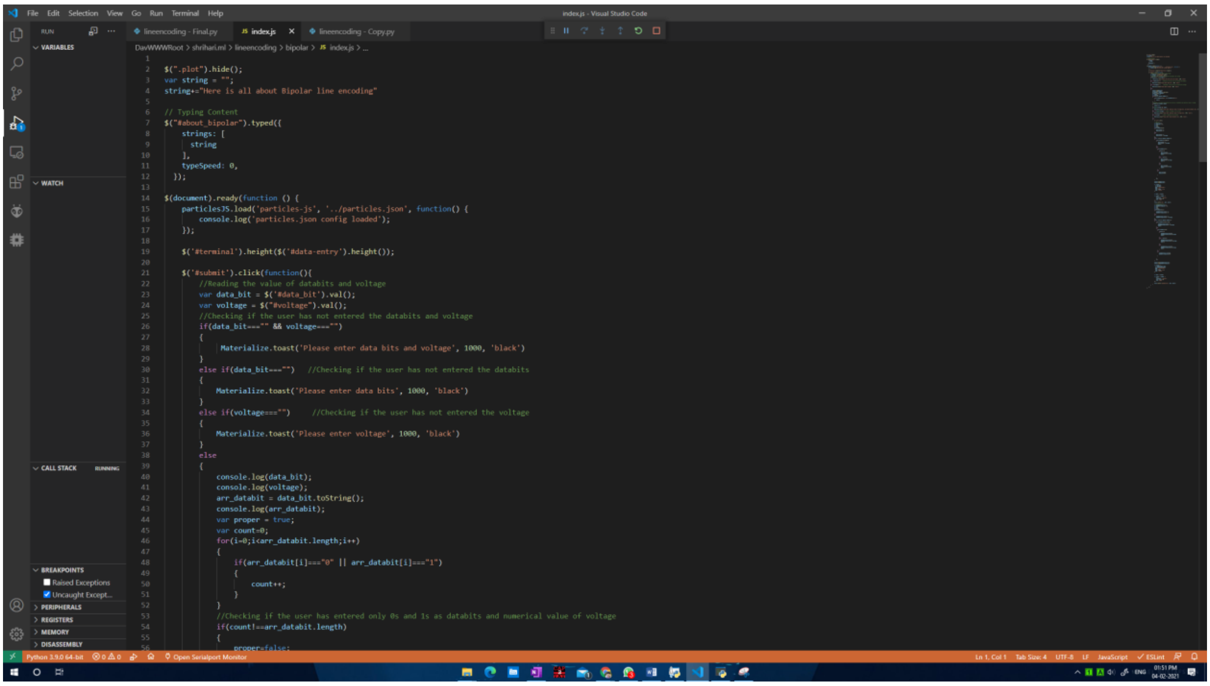




DIRECTORY STRUCTURE



SOURCE CODE – JAVASCRIPT – FOR BIPOLAR ENCODING



Due to the size of the .js, .css and .html files, source code is available in

Shrihari.ml/lineencoding (Ctrl+Shift+I in Google Chrome)

Result

The various line encoding schemes and to create a command line program and a GUI based web application for the same

- **Web application : https://shrihari.ml/lineencoding**