# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(Autonomous Institute of Savitribai Phule Pune University)*



**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **CS TY B** |
| **Batch** | **3** |
| **GR no.** | **12320165** |
| **Roll no.** | **83** |
| **Name** | **Komal Mahadev Potdar** |

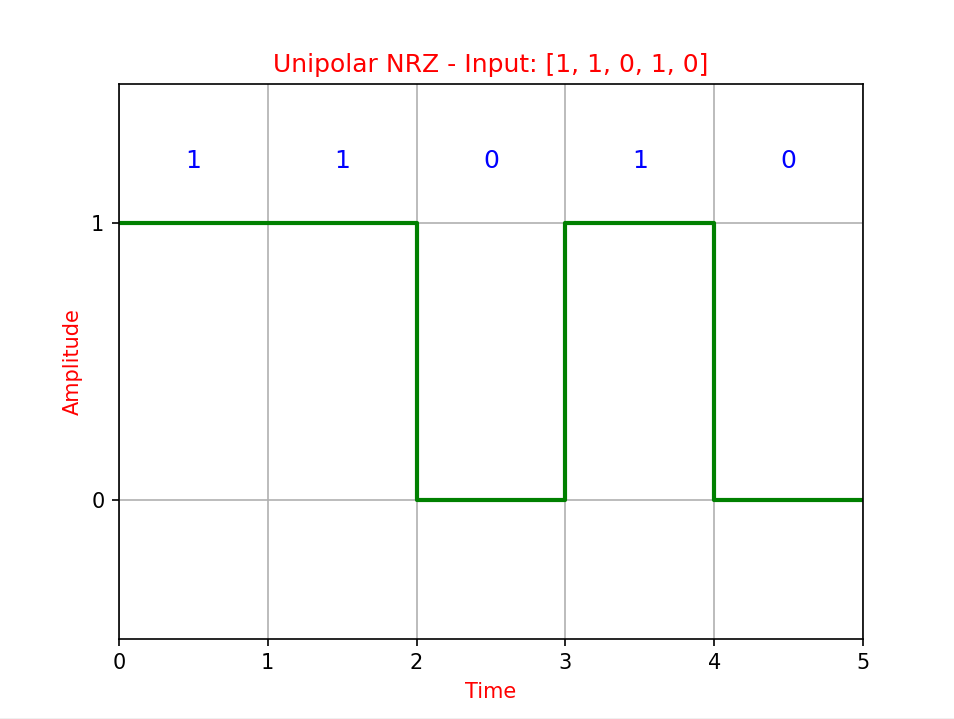
**Assignment No. 1**

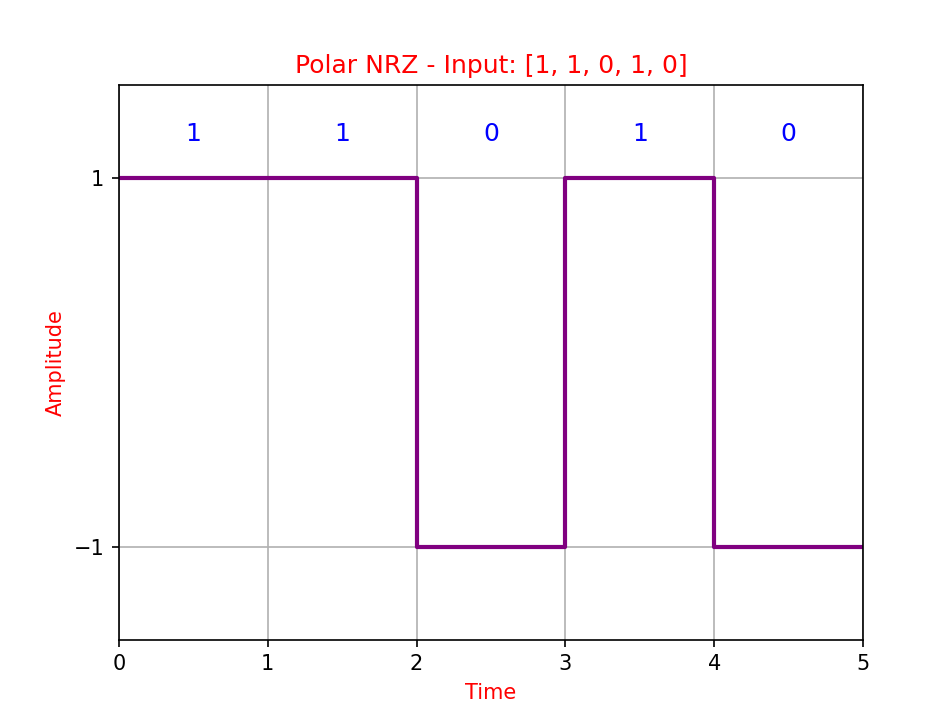
**Title:** Write a program to implement - Unipolar NRZ, Polar NRZ, NRZ Inverted, Bipolar Encoding, Manchester Encoding and Differential Manchester Encoding.

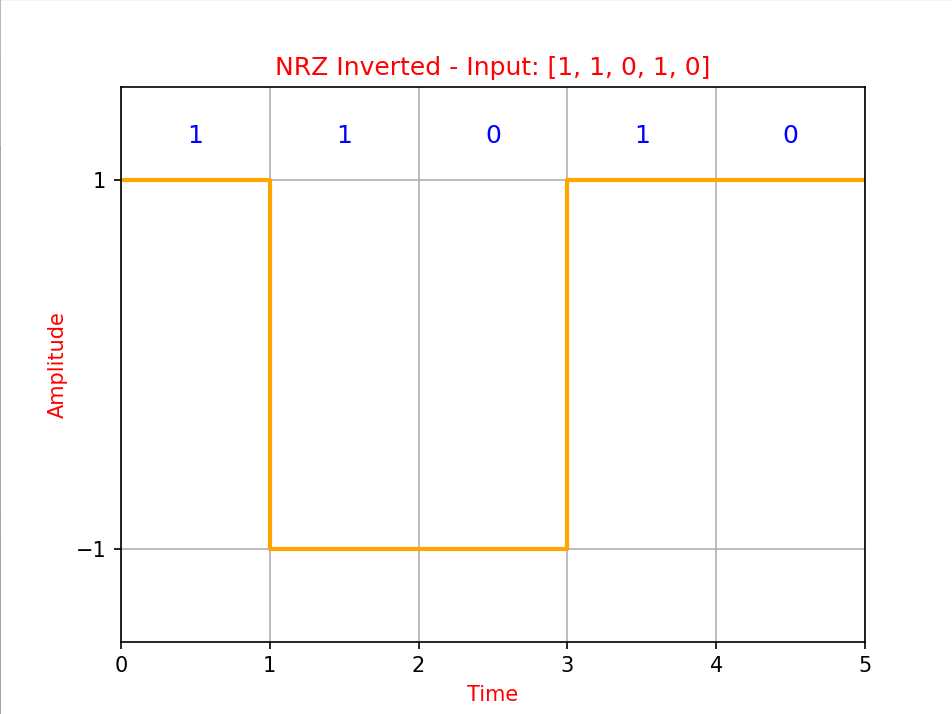
**Code:**

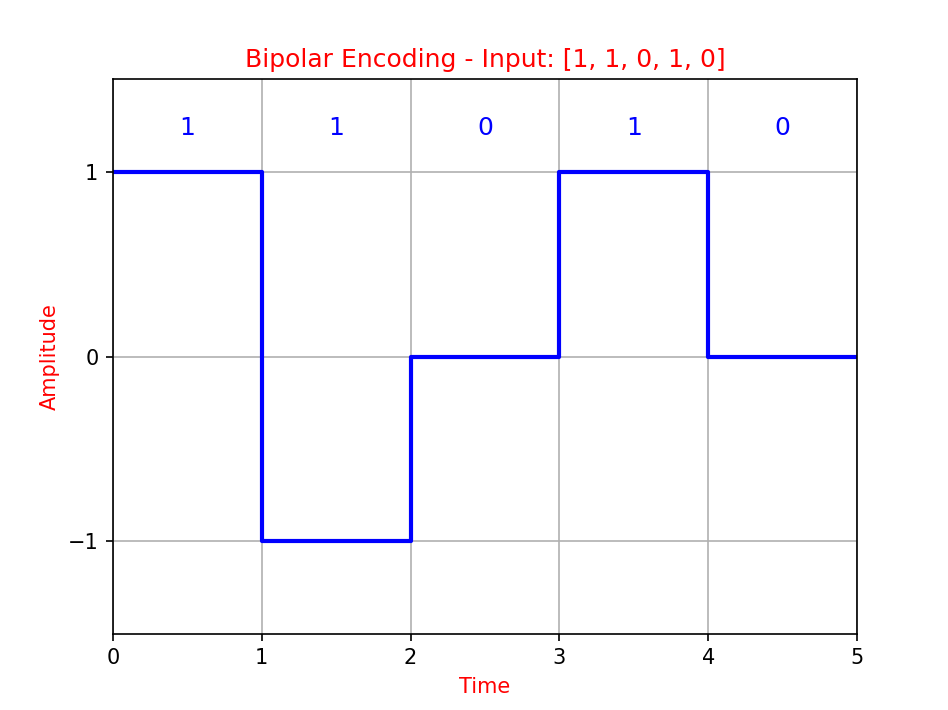
|  |
| --- |
| import matplotlib.pyplot as plt  def annotate\_input(data, t, d):      for i, bit in enumerate(data):          plt.annotate(f'{bit}', (i + 0.5, 1.2), color='blue', fontsize=12, ha='center')          if max(d) > 1:              plt.annotate(f'{bit}', (i + 0.5, -1.2), color='blue', fontsize=12, ha='center')  def unipolarNRZ(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 1)      d = []      for bit in data:          d.append(bit)          d.append(bit)      plt.step(t, d, where='post', color='green', linewidth=2)      plt.ylim(-0.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([0, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'Unipolar NRZ - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('unipolar\_nrz.png')      plt.show()  def polarNRZ(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 1)      d = []      for bit in data:          d.append(1 if bit == 1 else -1)          d.append(1 if bit == 1 else -1)      plt.step(t, d, where='post', color='purple', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'Polar NRZ - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('polar\_nrz.png')      plt.show()  def nrzInverted(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 1)      d = []      current\_level = -1      for bit in data:          if bit == 1:              current\_level \*= -1          d.append(current\_level)          d.append(current\_level)      plt.step(t, d, where='post', color='orange', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'NRZ Inverted - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('nrz\_inverted.png')      plt.show()  def bipolarEncoding(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 1)      d = []      current\_level = 1      for bit in data:          if bit == 1:              d.append(current\_level)              d.append(current\_level)              current\_level \*= -1          else:              d.append(0)              d.append(0)      plt.step(t, d, where='post', color='blue', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 0, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'Bipolar Encoding - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('bipolar\_encoding.png')      plt.show()  def manchesterEncoding(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 0.5)          t.append(i + 0.5)          t.append(i + 1)      d = []      for bit in data:          if bit == 1:              d.append(1)              d.append(1)              d.append(-1)              d.append(-1)          else:              d.append(-1)              d.append(-1)              d.append(1)              d.append(1)      plt.step(t, d, where='post', color='brown', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'Manchester Encoding - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('manchester\_encoding.png')      plt.show()  def differentialManchesterEncoding(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 0.5)          t.append(i + 0.5)          t.append(i + 1)      d = []      current\_level = -1      for bit in data:          if bit == 0:              d.append(current\_level)              d.append(current\_level)              current\_level \*= -1              d.append(current\_level)              d.append(current\_level)          else:              d.append(current\_level)              d.append(current\_level \* -1)              d.append(current\_level \* -1)              d.append(current\_level)      plt.step(t, d, where='post', color='magenta', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'Differential Manchester Encoding - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('differential\_manchester\_encoding.png')      plt.show()  def ieeeManchesterEncoding(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 0.5)          t.append(i + 0.5)          t.append(i + 1)        d =[]      for bit in data:          if bit == 1:              d.append(-1)              d.append(-1)              d.append(1)              d.append(1)          else:              d.append(1)              d.append(1)              d.append(-1)              d.append(-1)      plt.step(t, d, where='post', color='cyan', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'IEEE Manchester Encoding - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('ieee\_manchester\_encoding.png')      plt.show()  def nrzI(data):      t = []      for i in range(len(data)):          t.append(i)          t.append(i + 1)      d = []      current\_level = 1      for bit in data:          if bit == 1:              current\_level \*= -1          d.append(current\_level)          d.append(current\_level)      plt.step(t, d, where='post', color='orange', linewidth=2)      plt.ylim(-1.5, 1.5)      plt.xlim(0, len(data))      plt.yticks([-1, 1])      plt.xticks(range(len(data) + 1))      plt.xlabel('Time', color='red')      plt.ylabel('Amplitude', color='red')      plt.title(f'NRZ-I - Input: {data}', color='red')      plt.grid(True)      annotate\_input(data, t, d)      plt.savefig('nrz\_i.png')      plt.show()  binary\_data = [1, 1, 0, 1, 0]  unipolarNRZ(binary\_data)  polarNRZ(binary\_data)  nrzInverted(binary\_data)  bipolarEncoding(binary\_data)  manchesterEncoding(binary\_data)  differentialManchesterEncoding(binary\_data)  ieeeManchesterEncoding(binary\_data)  nrzI(binary\_data) |

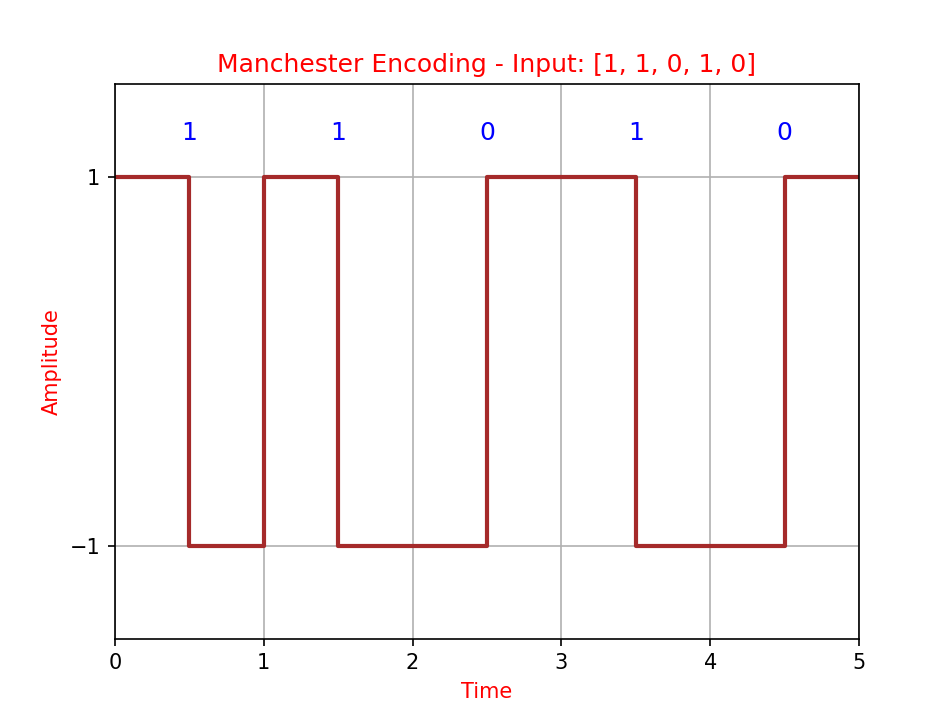
**Output:**

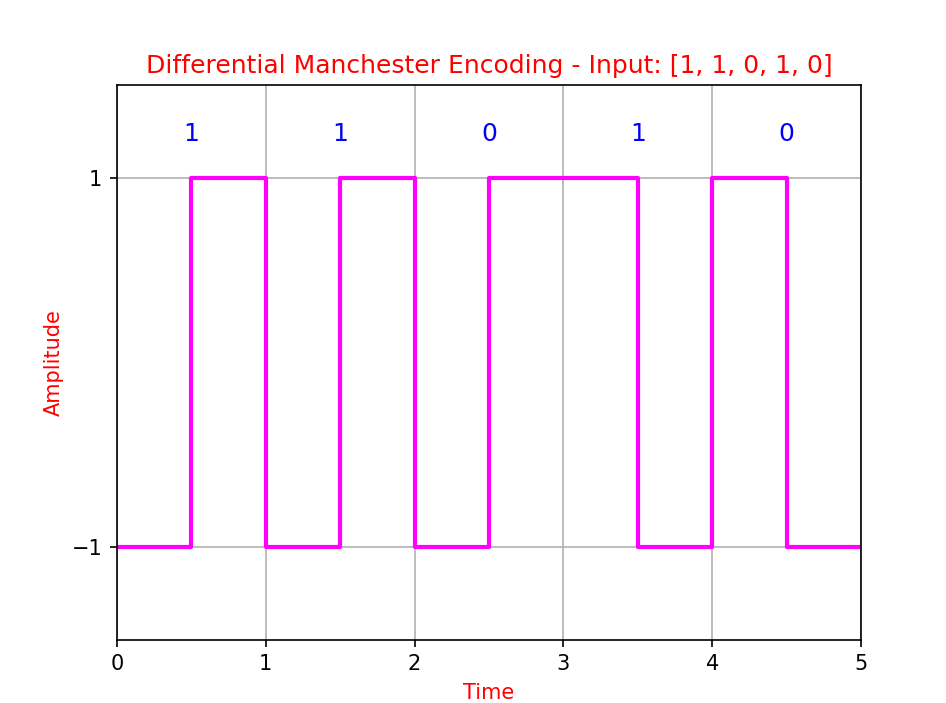
****

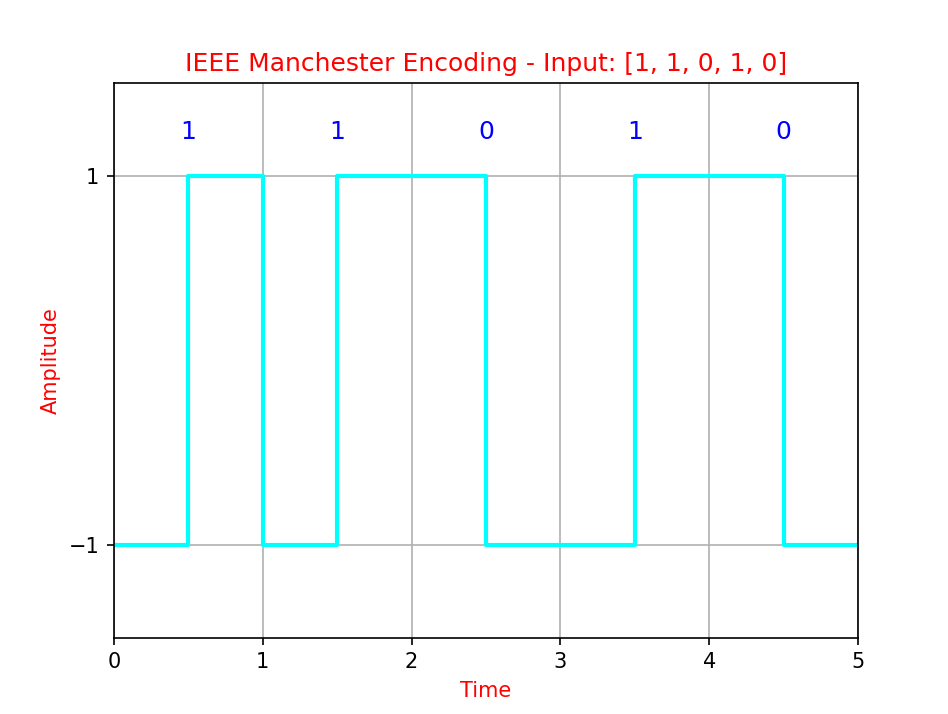
****

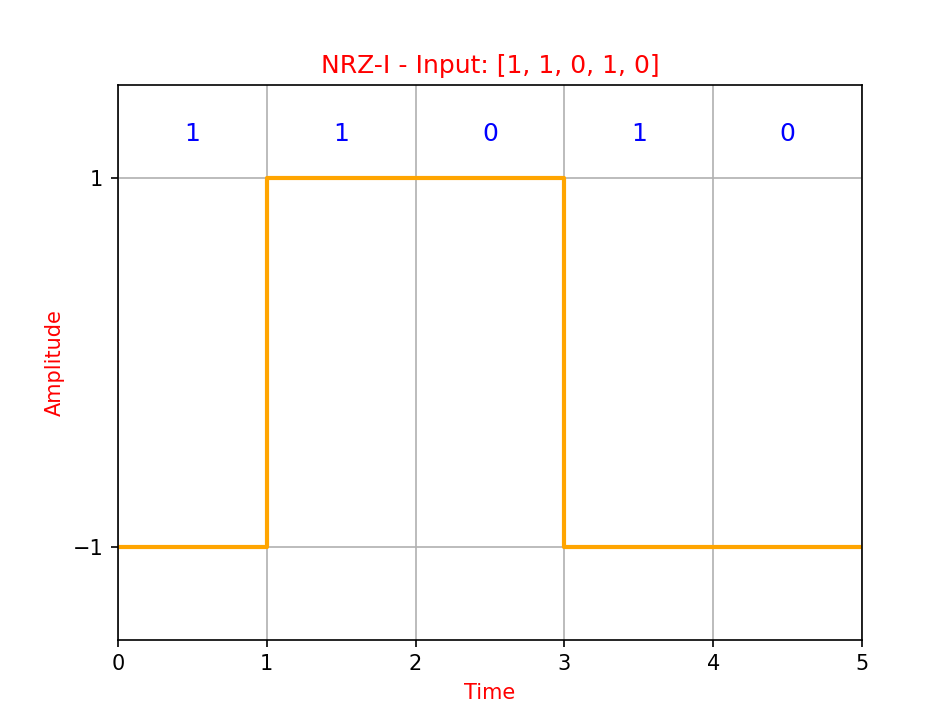
****

****

****

****

****

****