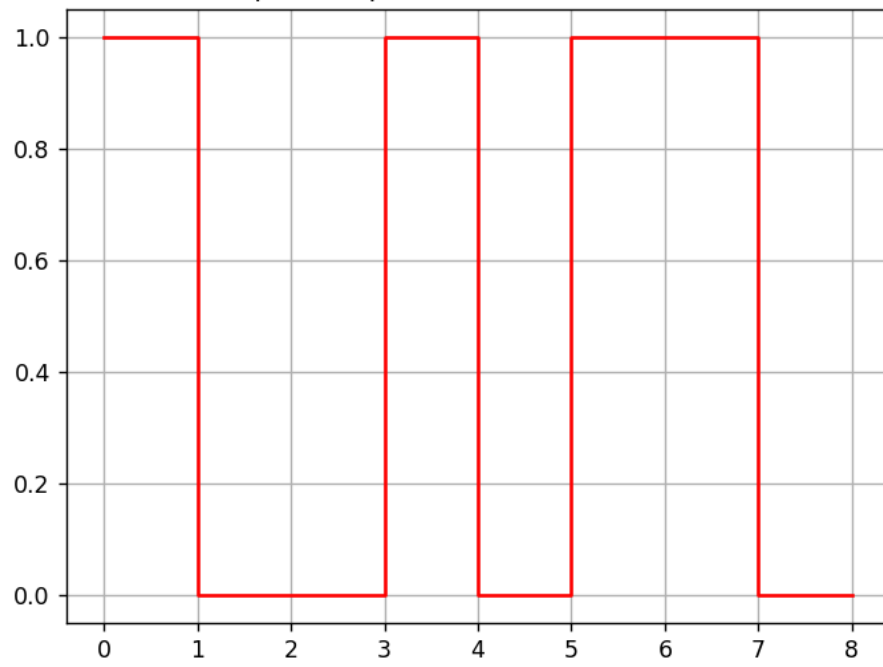
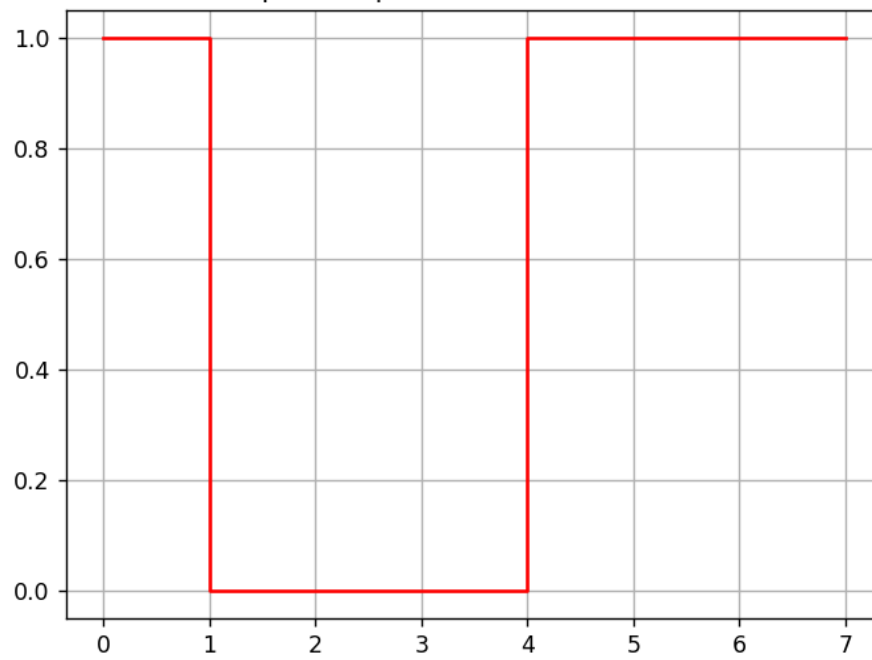


```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8  x.append(0)
9  y.append(1)
10
11 for i in inp_signal:
12     if i == '0':
13         if(y[-1] == 0):
14             y.append(0)
15         else:
16             x.append(x[-1])
17             y.append(0)
18             y.append(0)
19     else:
20         if(y[-1] == 1):
21             y.append(1)
22         else:
23             x.append(x[-1])
24             y.append(1)
25             y.append(1)
26
27     x.append(x[-1]+1)
28
29 plt.title('Unipolar Representation of : '+inp_signal)
30 plt.plot(x,y,'red')
31 plt.grid(True)
32 plt.show()
```

Unipolar Representation of : 10010110

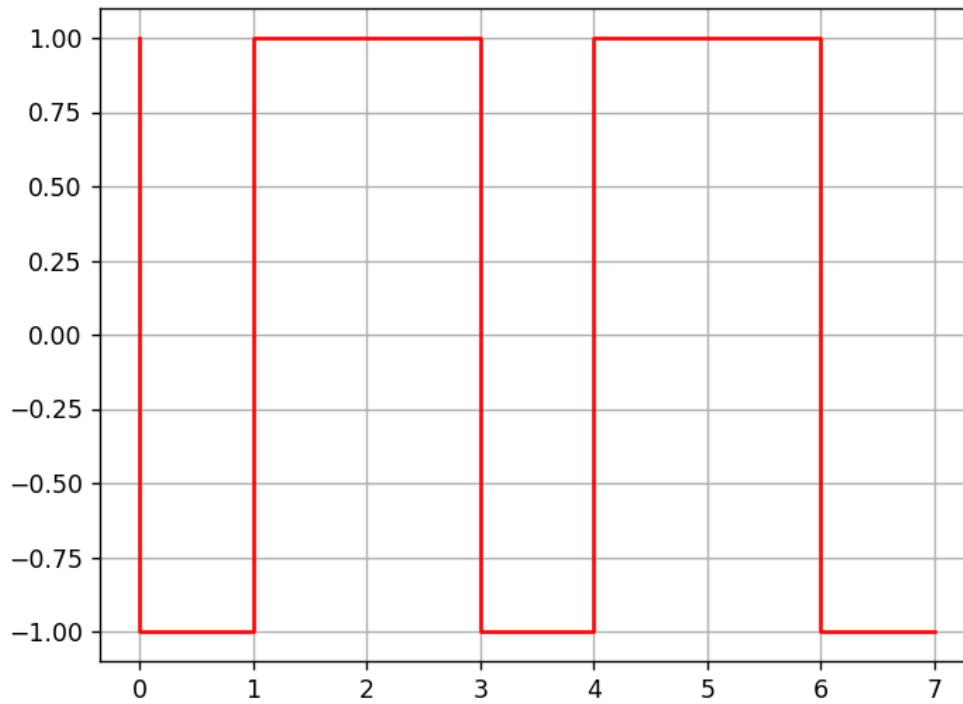


Unipolar Representation of : 1000111

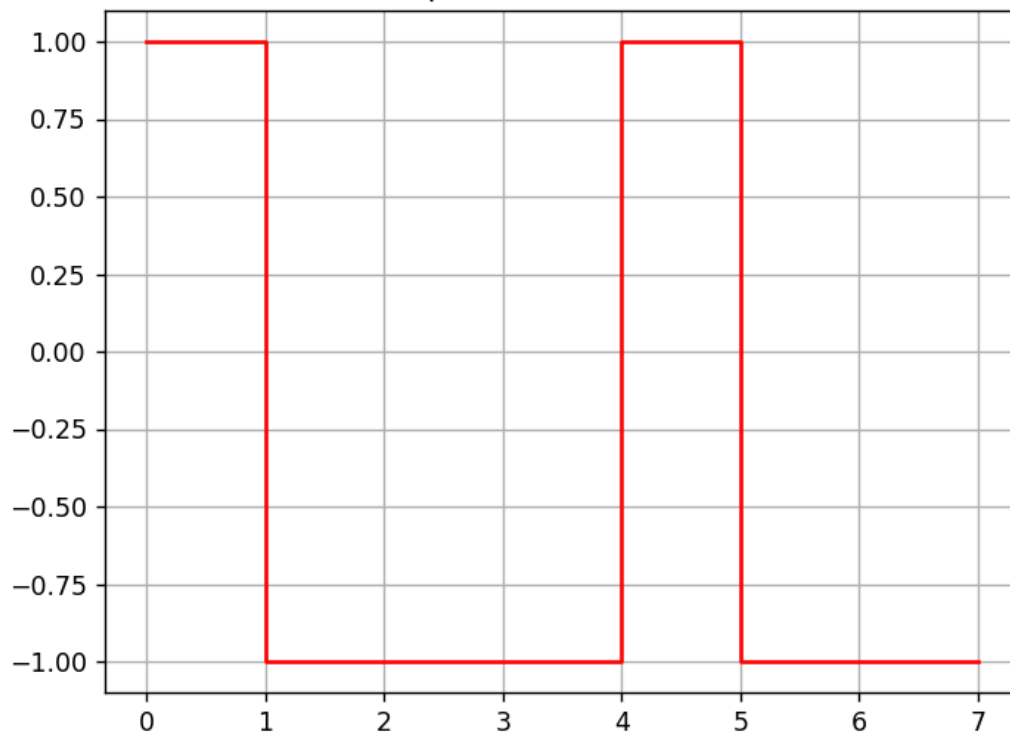


```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8  x.append(0)
9  y.append(1)
10
11  for i in inp_signal:
12      if i == '0':
13          y.append(y[-1])
14      else:
15          x.append(x[-1])
16          y.append(-1*y[-1])
17          y.append(y[-1])
18          x.append(x[-1]+1)
19
20  plt.title('NRZ I Representation of : '+inp_signal)
21  plt.plot(x,y,'red')
22  plt.grid(True)
23  plt.show()
```

NRZ I Representation of : 1101101

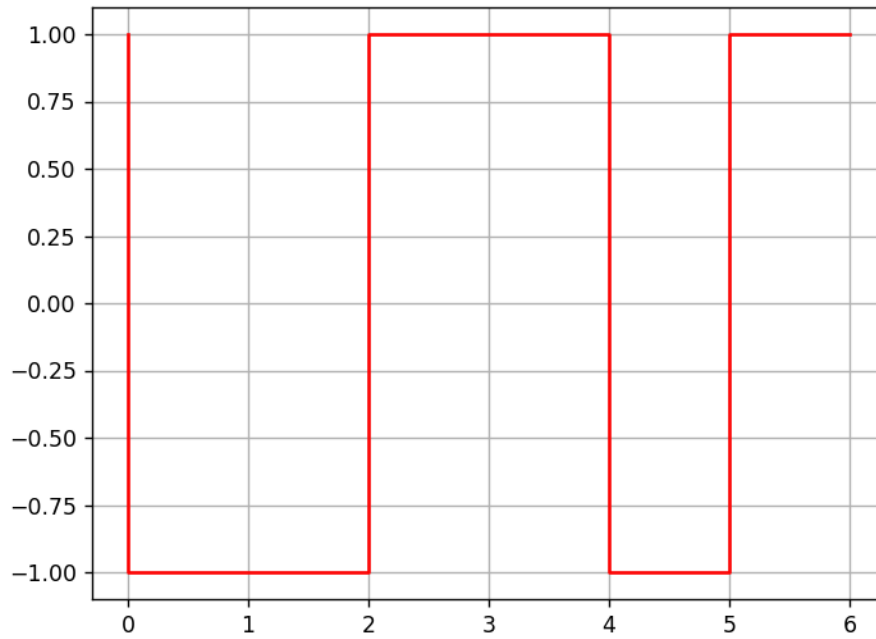


NRZ I Representation of : 0100110

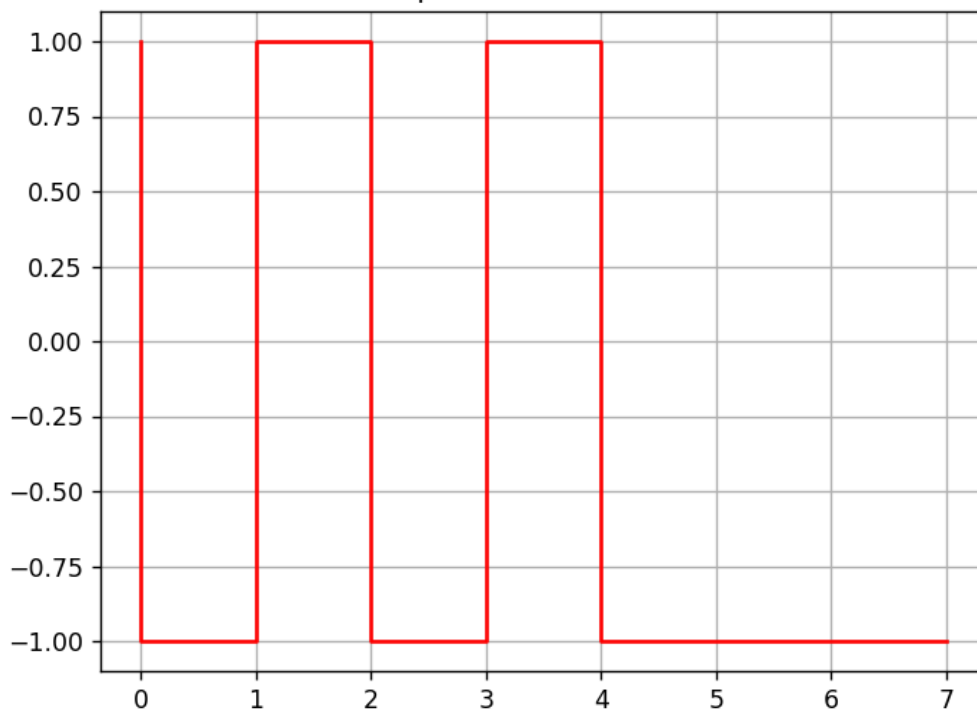


```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8  x.append(0)
9  y.append(1)
10
11 for i in inp_signal:
12     if i == '0':
13         if(y[-1] == 1):
14             y.append(1)
15         else:
16             x.append(x[-1])
17             y.append(1)
18             y.append(1)
19     else:
20         if(y[-1] == 1):
21             x.append(x[-1])
22             y.append(-1)
23             y.append(-1)
24         else:
25             y.append(-1)
26     x.append(x[-1]+1)
27
28 plt.title('NRZ L Representation of : '+inp_signal)
29 plt.plot(x,y,'red')
30 plt.grid(True)
31 plt.show()
```

NRZ L Representation of : 110010



NRZ L Representation of : 1010111

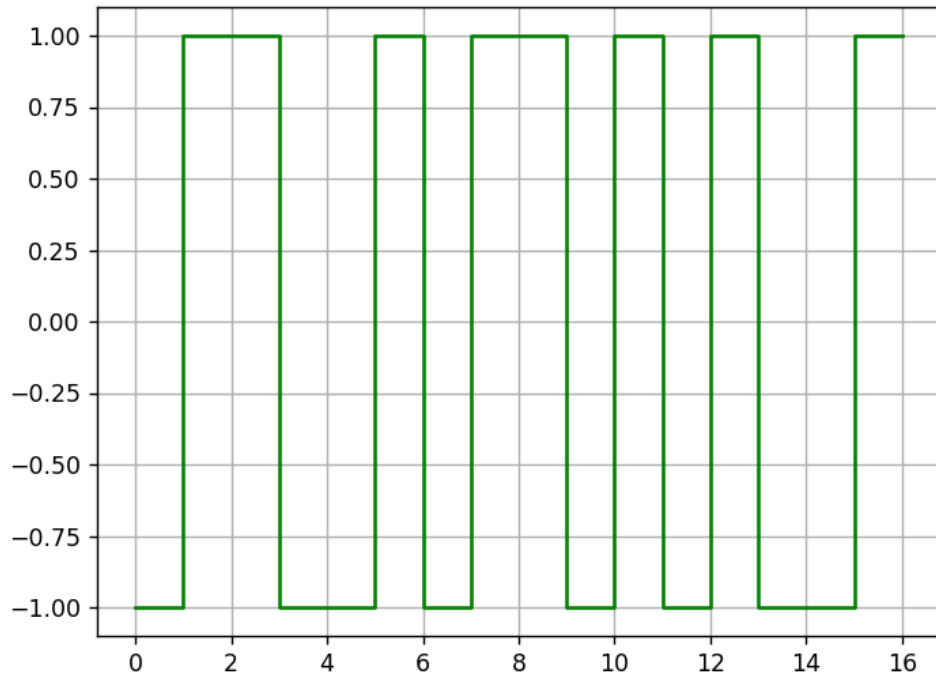


---

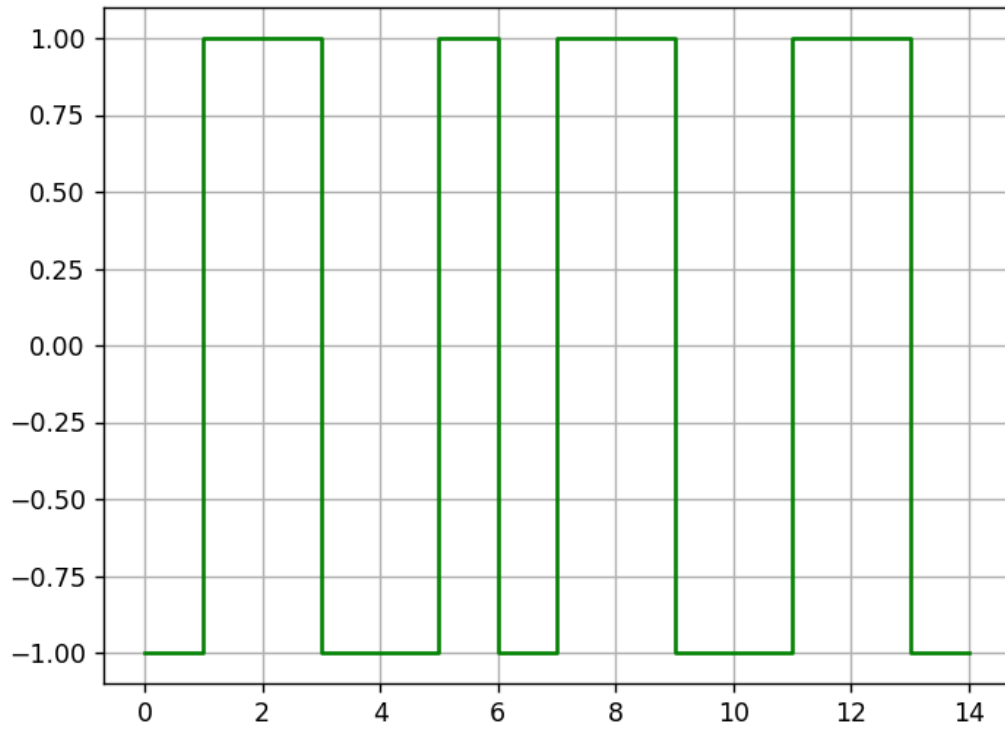
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8
9  for i in inp_signal:
10     if i == '0':
11         y.append(1)
12         y.append(-1)
13     else:
14         y.append(-1)
15         y.append(1)
16
17  y.append(y[-1])
18  print('y : ',y)
19  plt.title('Manchester Representation of : '+inp_signal)
20  #plt.plot(x,y,'red')
21  x = np.arange(0,len(y))
22  plt.step(x,y,color='green',where='post')
23  plt.grid(True)
24  plt.show()
```

---

Manchester Representation of : 10110001



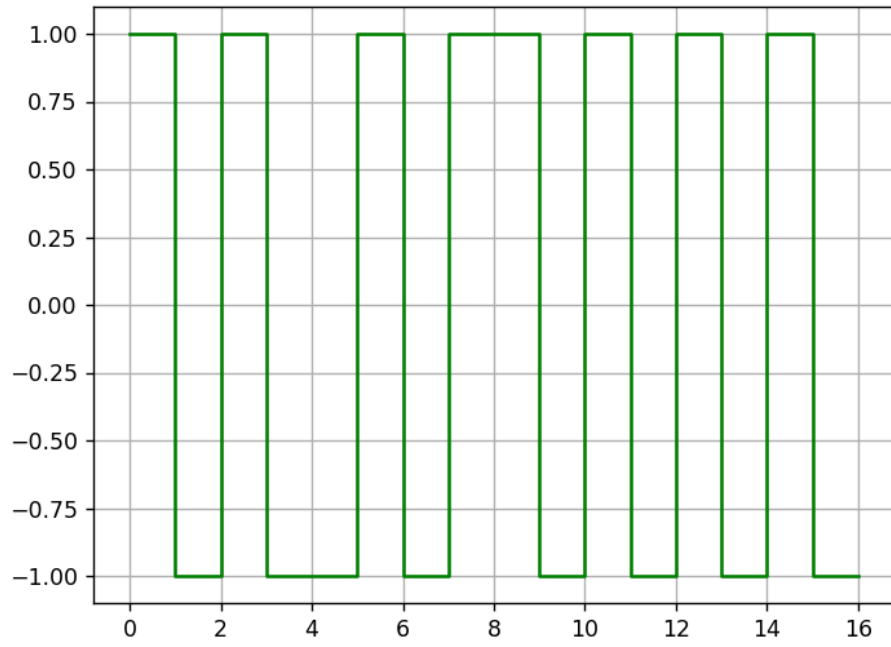
Manchester Representation of : 1011010



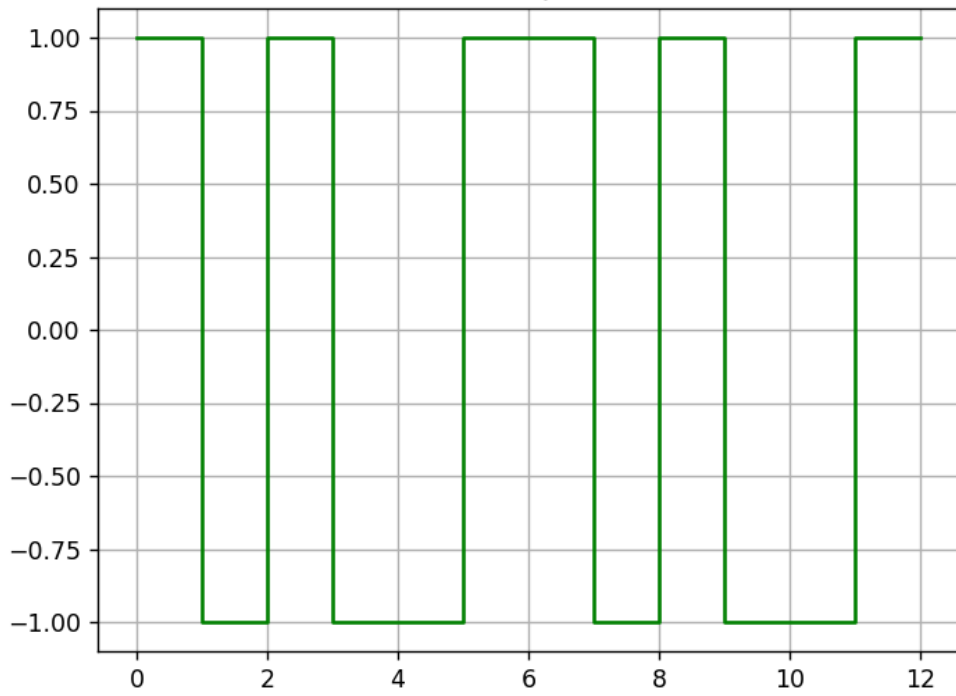


```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8  y.append(1)
9  for i in inp_signal:
10     if i == '0':
11         y.append(-1*y[-1])
12         y.append(-1*y[-1])
13     else:
14         y.append(y[-1])
15         y.append(-1*y[-1])
16
17  print('y : ',y)
18  plt.title('Differential Manchester Representation of : '+inp_signal)
19  #plt.plot(x,y,'red')
20  x = np.arange(0,len(y))
21  plt.step(x,y,color='green',where='pre')
22  plt.grid(True)
23  plt.show()
```

Differential Manchester Representation of : 10101000



Differential Manchester Representation of : 101101

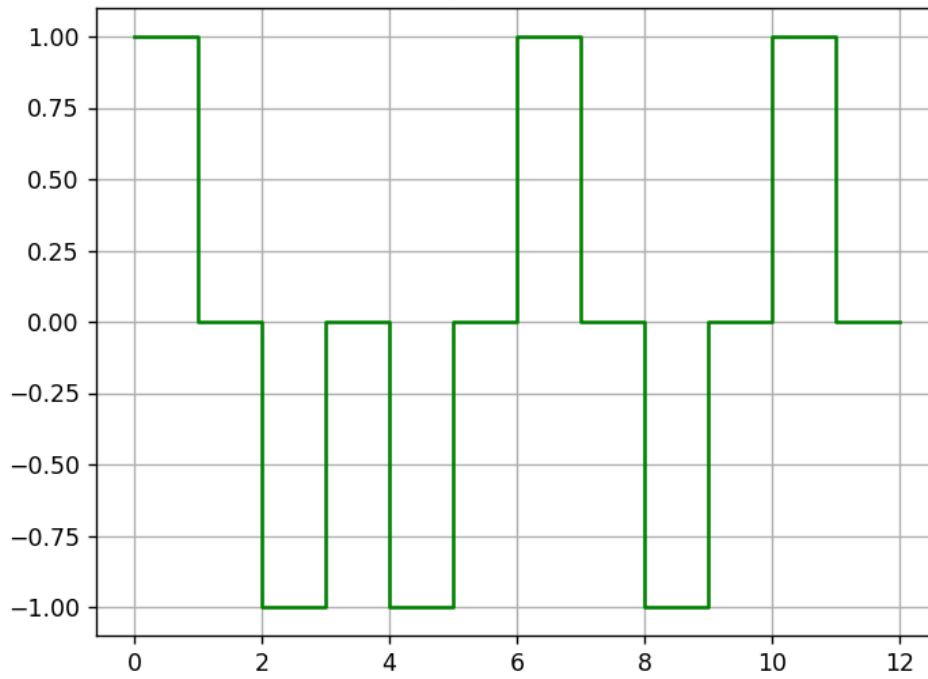


---

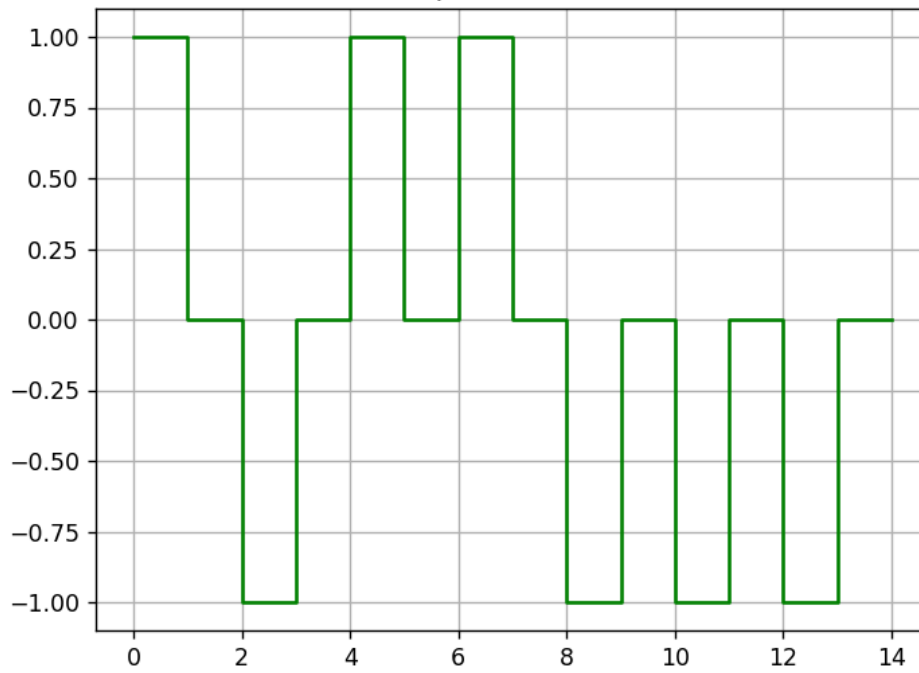
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  x = list()
7  y = list()
8  for i in inp_signal:
9      if i == '0':
10         y.append(-1)
11         y.append(0)
12     else:
13         y.append(1)
14         y.append(0)
15 y.append(y[-1])
16 print('y : ',y)
17 plt.title('Return Zero Representation of : '+inp_signal)
18 #plt.plot(x,y,'red')
19 x = np.arange(0,len(y))
20 plt.step(x,y,color='green',where='post')
21 plt.grid(True)
22 plt.show()
```

---

Return Zero Representation of : 100101



Return Zero Representation of : 1011000

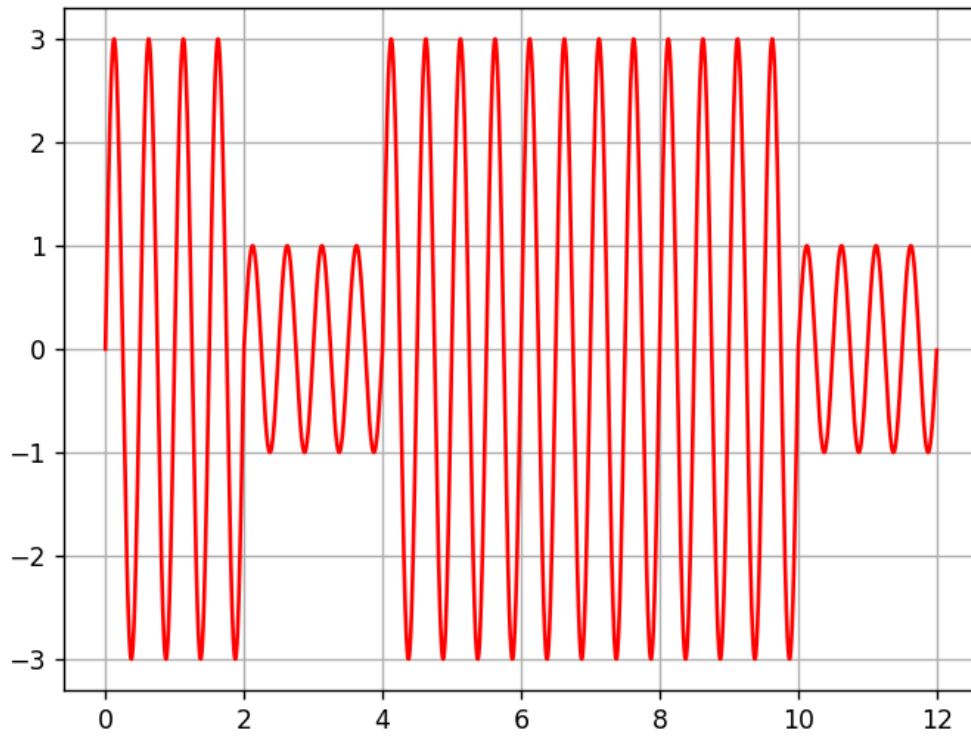


---

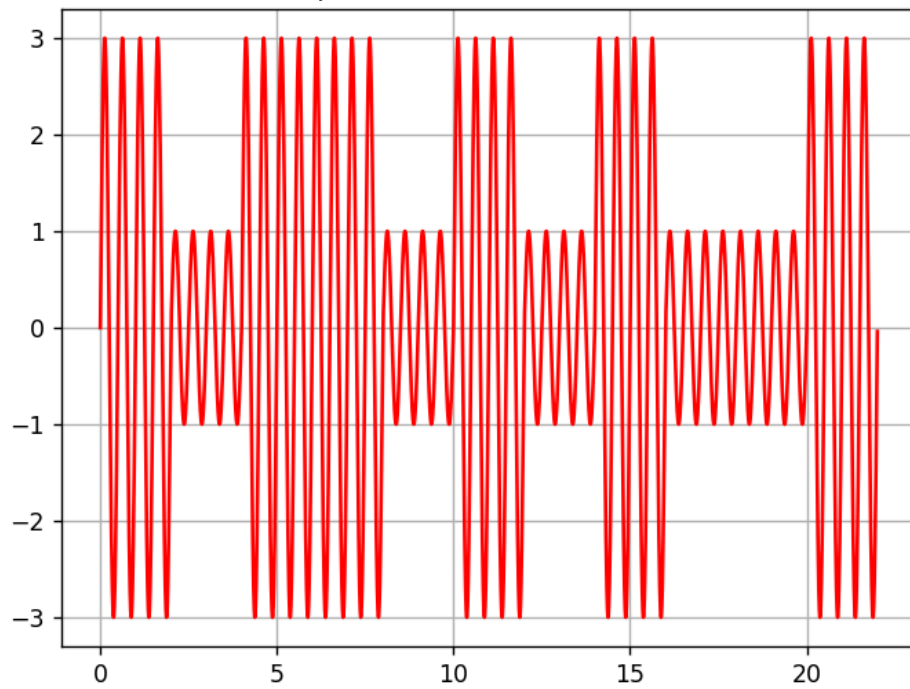
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  time = np.array(list())
7  amplitude = np.array(list())
8  step = 0
9  Low_amplitude = 1
10 High_amplitude = 3
11
12 def getSignal(A,time):
13     frequency = 2
14     phase = 0
15     return A*np.sin(2*np.pi*frequency*time+phase)
16
17 for i in inp_signal:
18     temp_time = np.arange(step,step+2,0.001)
19     time = np.append(time,temp_time)
20     if i == '0':
21         amplitude = np.append(amplitude,getSignal(Low_amplitude,temp_time))
22     else:
23         amplitude = np.append(amplitude,getSignal(High_amplitude,temp_time))
24     step = step + 2
25
26 plt.title('ASK Representation of : '+inp_signal)
27
28 plt.plot(time,amplitude,color='red')
29 plt.grid(True)
30 plt.show()
```

---

ASK Representation of : 101110



ASK Representation of : 10110101001

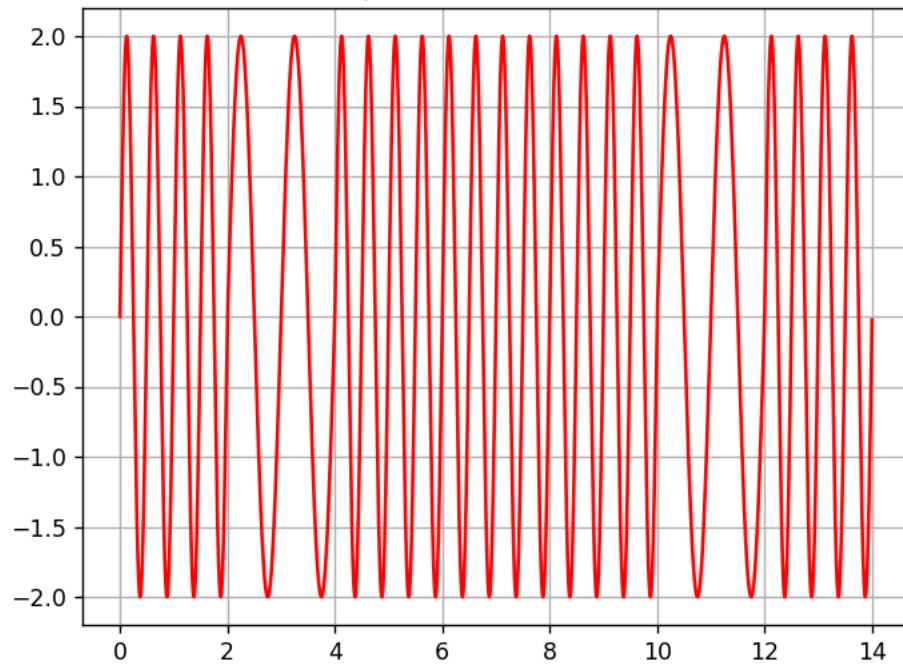


---

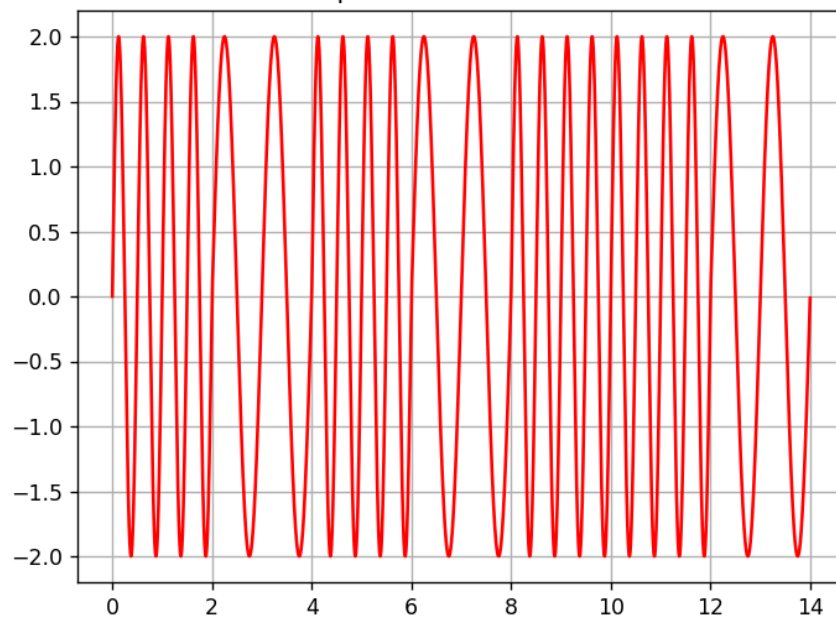
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  time = np.array(list())
7  amplitude = np.array(list())
8  step = 0
9  Low_frequency = 1
10 High_frequency = 2
11
12 def getSignal(f,time):
13     A = 2
14     phase = 0
15     return A*np.sin(2*np.pi*f*time+phase)
16
17 for i in inp_signal:
18     temp_time = np.arange(step,step+2,0.001)
19     time = np.append(time,temp_time)
20     if i == '0':
21         amplitude = np.append(amplitude,getSignal(Low_frequency,temp_time))
22     else:
23         amplitude = np.append(amplitude,getSignal(High_frequency,temp_time))
24     step = step + 2
25
26 plt.title('FSK Representation of : '+inp_signal)
27
28 plt.plot(time,amplitude,color='red')
29 plt.grid(True)
30 plt.show()
```

---

FSK Representation of : 1011101



FSK Representation of : 1010110



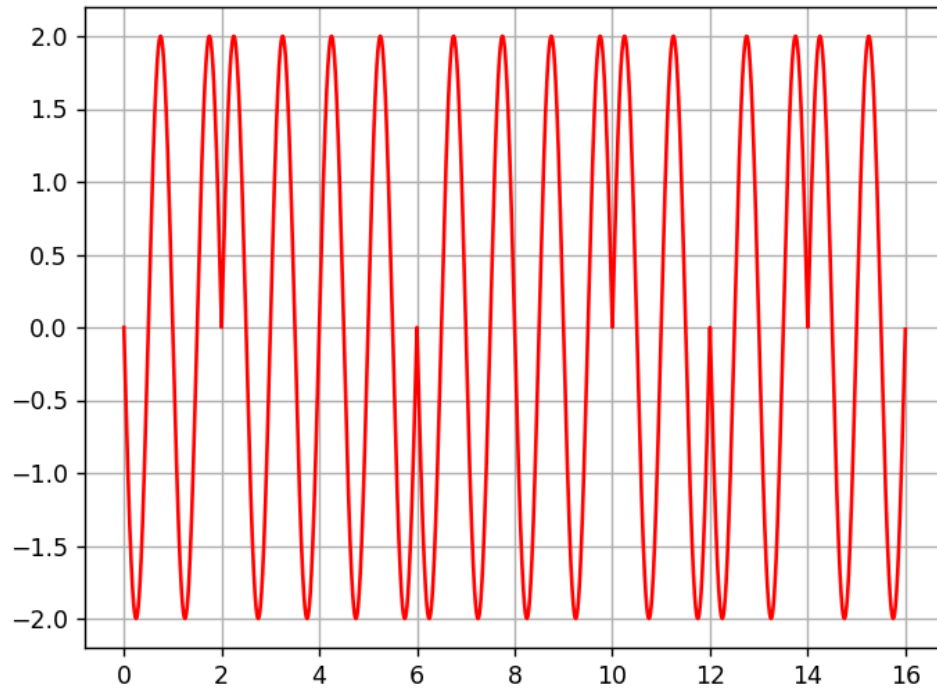


---

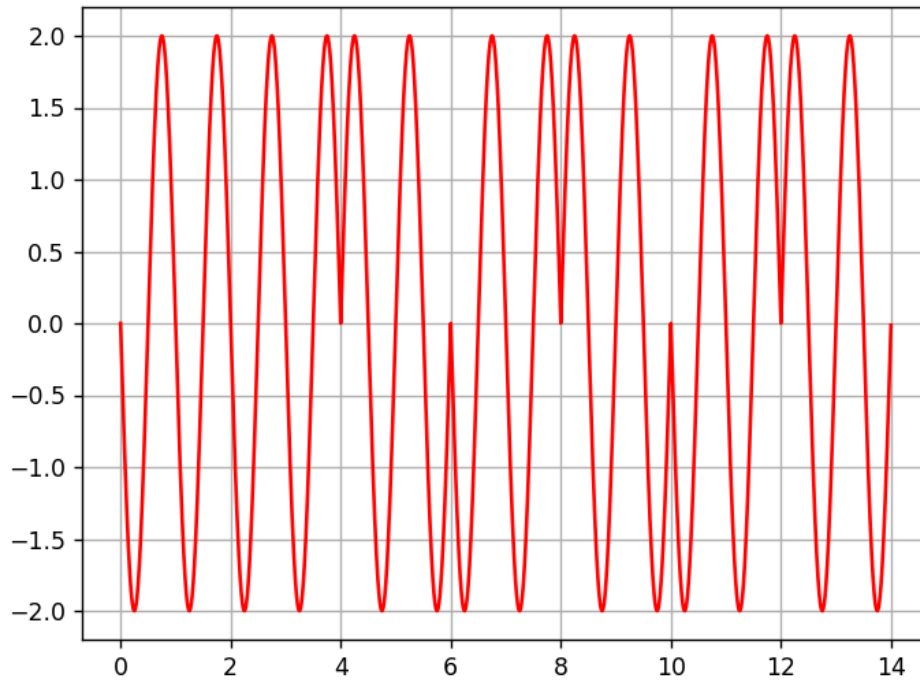
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  inp_signal = input('Enter binary bit: ')
5
6  time = np.array(list())
7  amplitude = np.array(list())
8  step = 0
9
10
11 def getSignal(phase,time):
12     A = 2
13     f = 1
14     return A*np.sin(2*np.pi*f*time+phase)
15
16 for i in inp_signal:
17     temp_time = np.arange(step,step+2,0.001)
18     time = np.append(time,temp_time)
19     if i == '0':
20         amplitude = np.append(amplitude,getSignal(0,temp_time))
21     else:
22         amplitude = np.append(amplitude,getSignal(np.pi,temp_time))
23     step = step + 2
24
25 plt.title('PSK Representation of : '+inp_signal)
26 plt.plot(time,amplitude,color='red')
27 plt.grid(True)
28 plt.show()
```

---

PSK Representation of : 10011010



PSK Representation of : 1101010

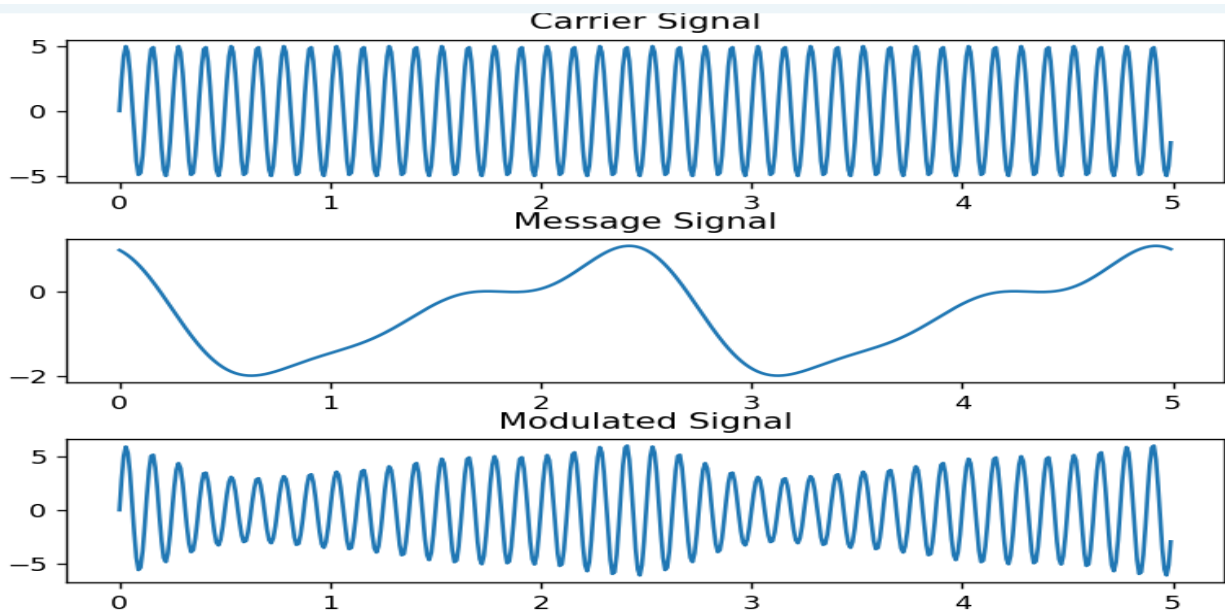




```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 figure, axis = plt.subplots(3)
5 plt.tight_layout()
6
7 #carrier Signal
8 carrier_time = np.arange(0,5,0.01)
9 carrier_A = 5
10 carrier_f = 8
11 carrier_phase = 0
12 carrier_amplitude = carrier_A*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
13 axis[0].plot(carrier_time,carrier_amplitude)
14 axis[0].set_title('Carrier Signal')
15
16 #Message Signal
17 message_time = np.arange(0,5,0.01)
18 message_A = 1
19 message_f = 0.4
20 message_phase = (np.pi/2)
21 thita = 2*np.pi*message_f*message_time+message_phase
22 message_amplitude = message_A*(np.sin(thita)**3-np.cos(thita)**2+np.cos(thita))
23 axis[1].plot(message_time,message_amplitude)
24 axis[1].set_title('Message Signal')
25
26 #Modulated Signal
27 modulated_time = np.arange(0,5,0.01)
28 modulated_amplitude = (carrier_A + message_amplitude)*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
29 axis[2].plot(modulated_time,modulated_amplitude)
30 axis[2].set_title('Modulated Signal')
31
32 plt.show()

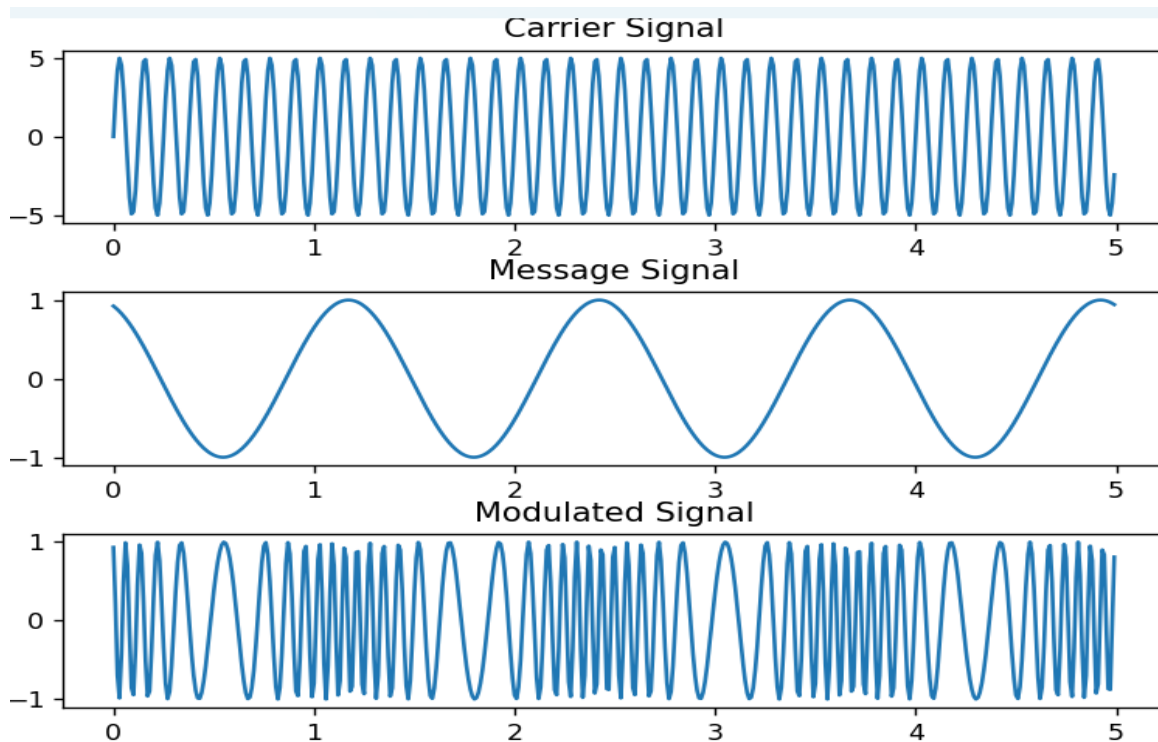
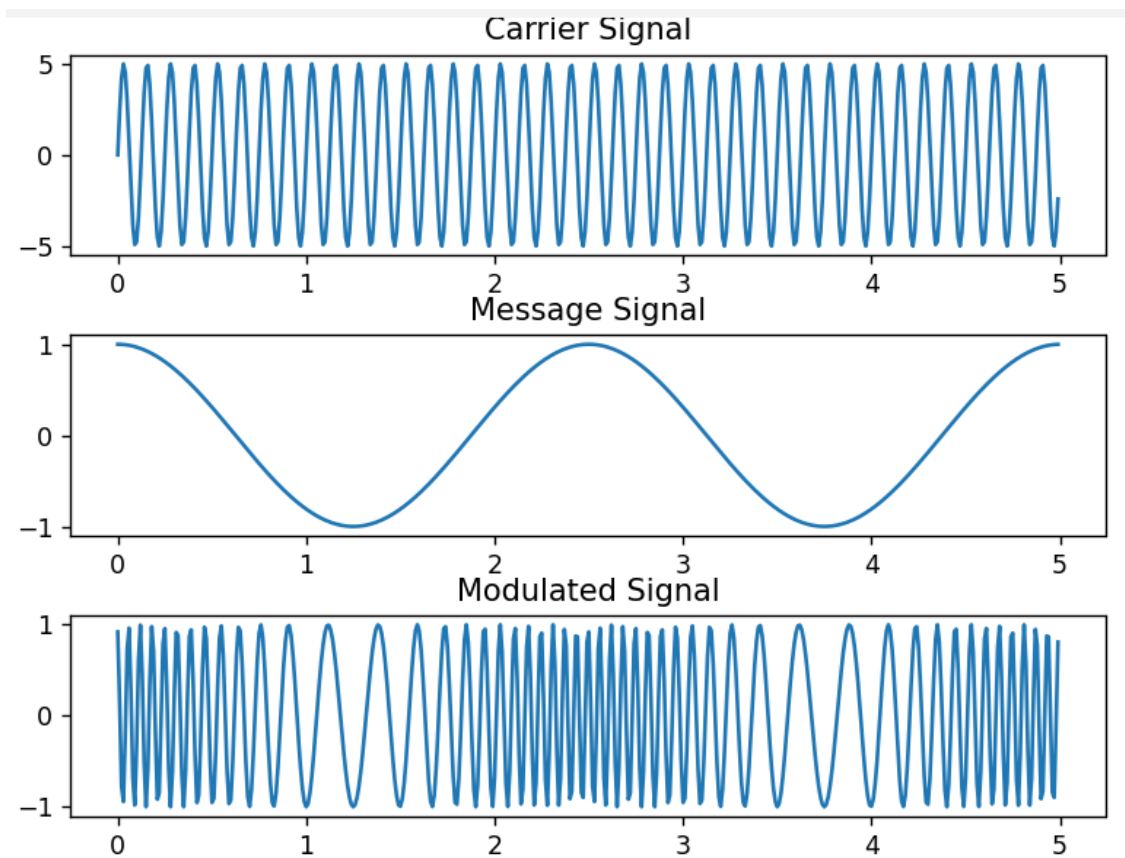
```



---

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 figure, axis = plt.subplots(3)
5 plt.tight_layout()
6
7 #carrier Signal
8 carrier_time = np.arange(0,5,0.01)
9 carrier_A = 5
10 carrier_f = 8
11 carrier_phase = 0
12 carrier_amplitude = carrier_A*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
13 axis[0].plot(carrier_time,carrier_amplitude)
14 axis[0].set_title('Carrier Signal')
15
16 #Message Signal
17 message_time = np.arange(0,5,0.01)
18 message_A = 1
19 message_f = 0.4
20 message_phase = (np.pi/2)
21 thita = 2*np.pi*message_f*message_time+message_phase
22 message_amplitude = message_A*np.sin(thita)
23 axis[1].plot(message_time,message_amplitude)
24 axis[1].set_title('Message Signal')
25
26 #Modulated Signal
27 modulated_time = np.arange(0,5,0.01)
28 modulated_frequency = 10
29 k = 0.4 #sensitivity
30 phi = 2*np.pi*modulated_frequency*modulated_time + k*np.cumsum(message_amplitude)
31 modulated_amplitude = np.cos(phi)
32 axis[2].plot(modulated_time,modulated_amplitude)
33 axis[2].set_title('Modulated Signal')
34
35 plt.show()
```

---



---

```
1 import numpy as np
2 inp_signal = input('Enter Message Signal : ')
3 m = len(inp_signal)
4 r = int(np.log2(m))
5 while(2**r < m+r+1):
6     r = r + 1
7 l = m+r
8 print('-> Message len :',m,' -> Redundant len :',r,' -> Total len :',l)
9 hammingCode = list()
10 sig_ind = 0
11 for i in range(1,l+1):
12     if (np.log2(i)-int(np.log2(i)) == 0.0):
13         hammingCode.append(0)
14     else:
15         hammingCode.append(ord(inp_signal[sig_ind])-48)
16         sig_ind = sig_ind + 1
17 print('Before Calculating Redundant Bit : ',hammingCode)
18 for i in range(1,l+1):
19     if (np.log2(i)-int(np.log2(i)) == 0.0):
20         print('\t',i,'th bit parity calculation')
21         parity = 0
22         ind = i - 1
23         while ind < l:
24             for j in range(i):
25                 if ind >= l:
26                     break
27                 parity = parity + hammingCode[ind]
28                 print('\t\t',ind+1,'th bit',hammingCode[ind],'total parity :',parity)
29                 ind = ind + 1
30             for j in range(i):
31                 if ind >= l:
32                     break
33                 ind = ind + 1
34             hammingCode[i-1] = parity%2
35             print('\t',i,'th Parity will :',hammingCode[i-1])
36 print('-----Generated Hamming code is-----')
37 print(hammingCode)
```

---

```

1 + Enter Message Signal : 1011101
2 + -> Message len : 7 -> Redundant len : 4 -> Total len : 11
3 + Before Calculating Redundant Bit : [0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1]
4 +      1 th bit parity calculation
5 +      1 th bit 0 total parity : 0
6 +      3 th bit 1 total parity : 1
7 +      5 th bit 0 total parity : 1
8 +      7 th bit 1 total parity : 2
9 +      9 th bit 1 total parity : 3
10 +      11 th bit 1 total parity : 4
11 +      1 th Parity will : 0
12 +      2 th bit parity calculation
13 +      2 th bit 0 total parity : 0
14 +      3 th bit 1 total parity : 1
15 +      6 th bit 1 total parity : 2
16 +      7 th bit 1 total parity : 3
17 +      10 th bit 0 total parity : 3
18 +      11 th bit 1 total parity : 4
19 +      2 th Parity will : 0
20 +      4 th bit parity calculation
21 +      4 th bit 0 total parity : 0
22 +      5 th bit 0 total parity : 0
23 +      6 th bit 1 total parity : 1
24 +      7 th bit 1 total parity : 2
25 +      4 th Parity will : 0
26 +      8 th bit parity calculation
27 +      8 th bit 0 total parity : 0
28 +      9 th bit 1 total parity : 1
29 +      10 th bit 0 total parity : 1
30 +      11 th bit 1 total parity : 2
31 +      8 th Parity will : 0
32 + -----Generated Hamming code is-----
33 + [0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1]

```



```

1  import numpy as np
2
3  inp_signal = input('Enter Hamming Code:')
4
5  hammingCode = list()
6  for i in inp_signal:
7      hammingCode.append(ord(i)-48)
8
9  print('Hamming code is :',hammingCode)
10 l = len(inp_signal)
11 parity_mismatch = list()
12 for i in range(1,l+1):
13     if (np.log2(i)-int(np.log2(i))) == 0.0:
14         print(i,'th bit parity calculation')
15         parity = 0
16         ind = i - 1
17         while ind < l:
18             for j in range(i):
19                 if ind >= l:
20                     break
21                 parity = parity + hammingCode[ind]
22                 print('\t',ind+1,'th bit',hammingCode[ind],'total parity :',parity)
23                 ind = ind + 1
24             for j in range(i):
25                 if ind >= l:
26                     break
27                 ind = ind + 1
28         parity_mismatch.append(parity%2)
29         print(i,'th Parity will :',parity_mismatch[-1])
30
31 print('-----Hamming Missmatch Parity code is-----')
32 print(parity_mismatch)

```

Enter Message Signal : 1011001

-> Message len : 7 -> Redundant len : 4 -> Total len : 11

Before Calculating Redundant Bit : [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1]

1 th bit parity calculation

1 th bit 0 total parity : 0

3 th bit 1 total parity : 1

5 th bit 0 total parity : 1

7 th bit 1 total parity : 2

9 th bit 0 total parity : 2

11 th bit 1 total parity : 3

1 th Parity will : 1

2 th bit parity calculation

2 th bit 0 total parity : 0

3 th bit 1 total parity : 1

6 th bit 1 total parity : 2

7 th bit 1 total parity : 3

10 th bit 0 total parity : 3

11 th bit 1 total parity : 4

2 th Parity will : 0

4 th bit parity calculation

4 th bit 0 total parity : 0

5 th bit 0 total parity : 0

7 th bit 1 total parity : 2

4 th Parity will : 0

8 th bit parity calculation

8 th bit 0 total parity : 0

9 th bit 0 total parity : 0

10 th bit 0 total parity : 0

11 th bit 1 total parity : 1

8 th Parity will : 1

-----Hamming Mismatch Parity code is-----

[0, 0, 0, 1]