ENG4052: Digital Communication 4 (2022-23)

Digital Modulation and Demodulation

Ran Shuai (2633609R)

Content Tables

BPSK

The modulation and demodulation

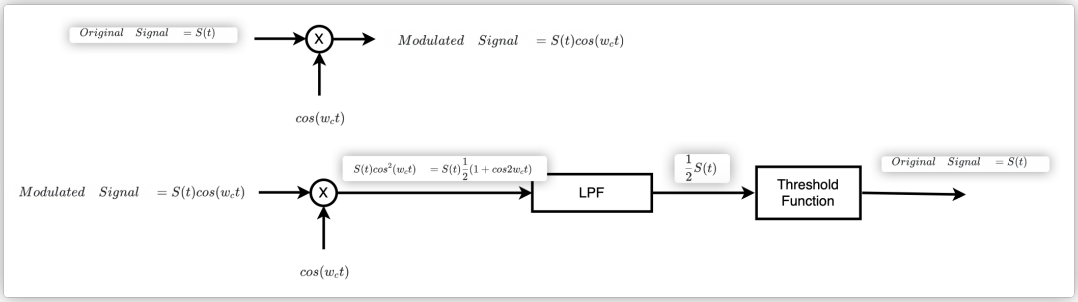

Figure 1.1(b): The modulation and demodulation Process

Basic Parameters

The parameter bit_len means that we take 16 sample points for one bit.
The parameter fc is the normalized carrier frequency.
When fc equals to 0.125 and bit_len = 8, it means two whole carrier wave periods only represent one bit.

Modulation

BPSK (Binary Phase Shift Keying) is a method of digital signal encoding that uses two different phases to represent two different information symbols. Typically, one phase is used to represent "0" and the other phase is used to represent "1".

What I have Done

To modulate the signal, the signal needs to multiply by a carrier wave. This is because the information we want to transfer is always low frequency but the low frequency is not easy to be transferred. The carrier frequency is used

here to transfer the signal. One of the main reasons is that by carrying the signal on a carrier wave, the signal is more resistant to interference and noise. By this formula in figure 1.1, we get the modulated signal.

$$(2Signal - 1)Cos(w_c t)$$

Figure 1.1(b): The modulation Formula

The Operation in the code:

```
(2*s[i]-1)*np.cos(2*np.pi*fc*(i*bit_len+j))
```

The results

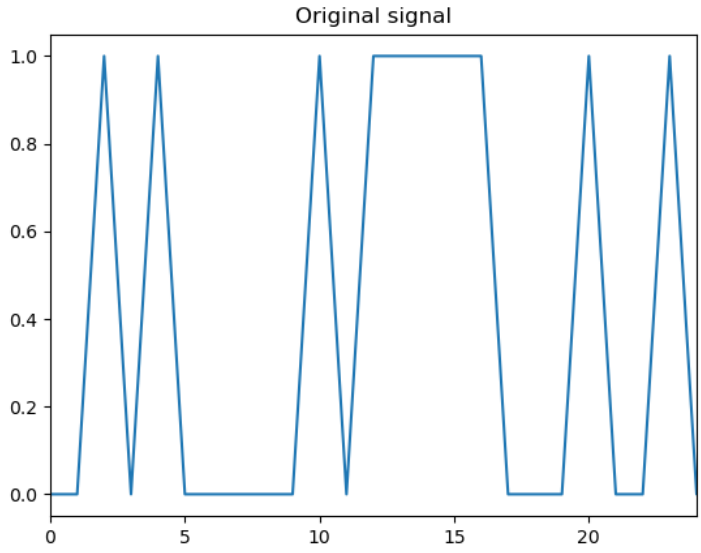

Figure 1.2: The Original Signal

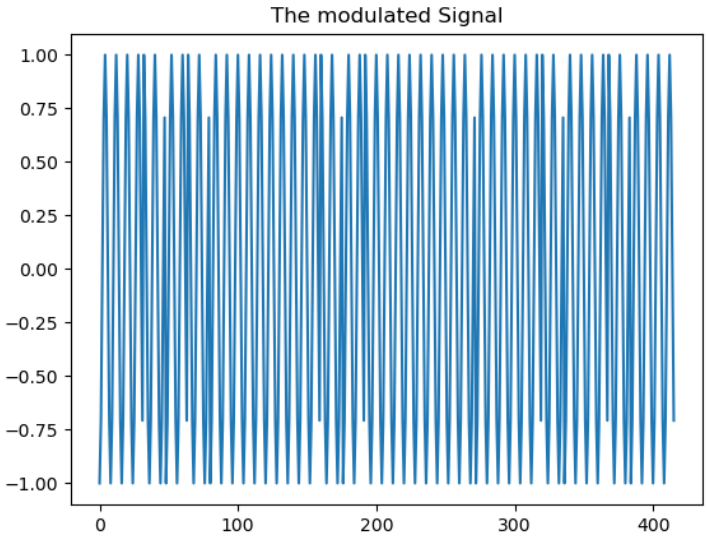

Figure 1.2: The modulated Signal

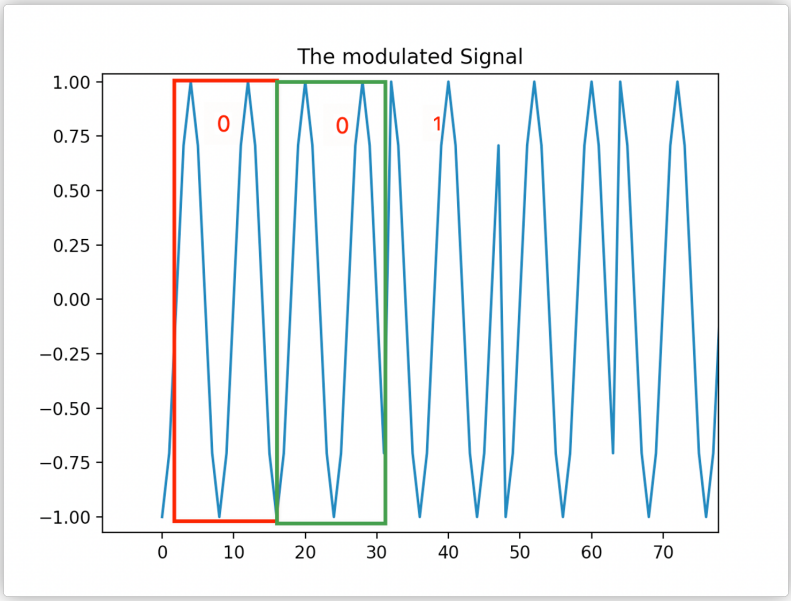

Figure 1.3: The modulated signal which is zoomed in

The first two elements of the 24-bit binary array of my student number are false. From Figure 1.3, it is implied that A cosine function with a phase of 180 represents 0, while a cosine function with a phase of 0 represents 1.

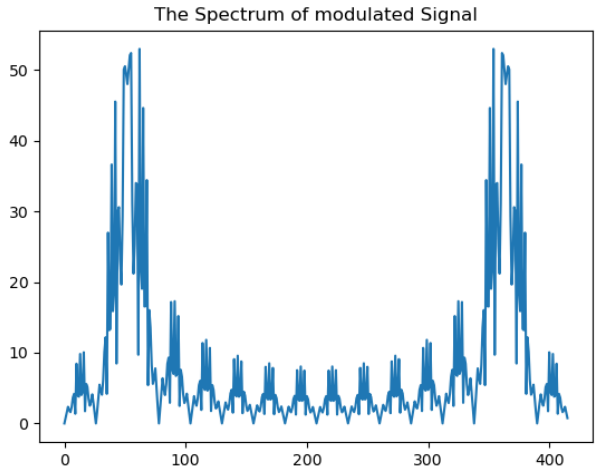

Figure 1.4: The spectrum of Modulated Signal

From figure 1.4, it is implied that carrier frequency has higher power, but sidebands have lower power. It is not effective, because the carrier only carries small information. To be more efficient, the sidebands should have more power.

Demodulation

This process is to convert the modulated signal to the original signal.

What I have Done

The achieve the original signal, the modulated signal should multiply the carrier wave again.

The formula to translate this operation

$$S(t)cos^2 w_c t = S(t)\frac{1}{2}(1 + cos(2w_c t))$$

Figure 1.5: The demodulation Formula

This formula shows that after multiplying the modulated signal by the carrier signal again. We get half of the original signal and a signal which has twice the frequency of the carrier signal.

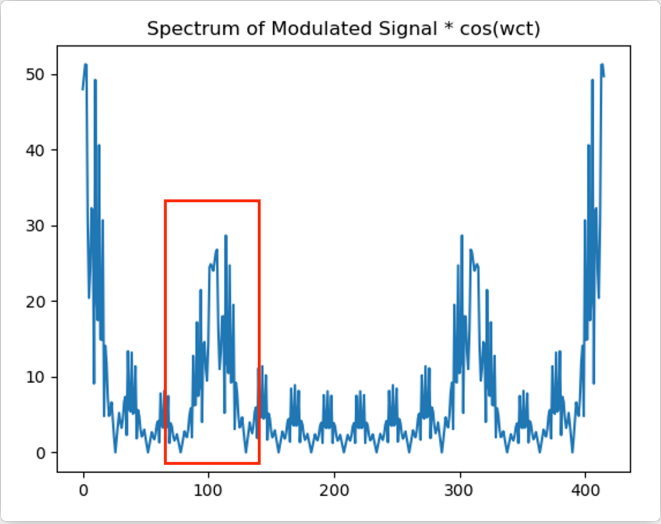

Figure 1.6: Spectrum of Modulated signal Multiplied by Carrier signal again

From the figure, we can see a signal with a frequency twice that of the carrier frequency. To get the original signal, we should filter it out. So a lowpass filter is used here to achieve this aim.

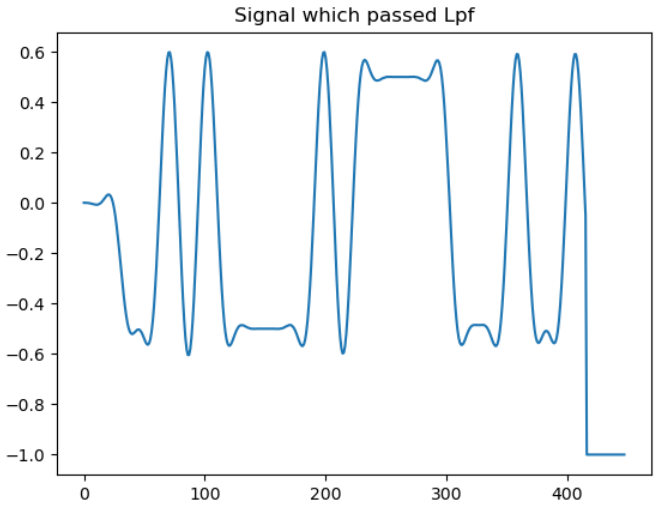

Figure 1.7: Signal which passed LPF

This is the signal which has already passed the LPF. But it looks like an analogue signal, and not like the original signal. We can use a threshold function to convert it back to the original signal.

The results

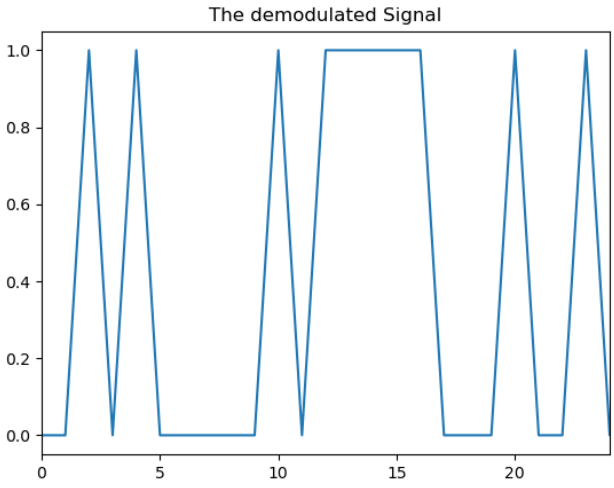

Figure 1.8: The demodulated Signal

This is the demodulated signal I finally get. There is hardly any delay in the demodulated signal compared to the original signal because I add numtaps//2 dummy samples to the end of my bit array.

QPSK

Modulation

QPSK encoding method uses changes in phase to transmit information, with each code symbol consisting of two binary information bits, resulting in four possible states.

What I have Done

```
for i in range(0,Nbits,2):
    for j in range(bit_len):
        tx_mod = np.append(tx_mod,(2*s[i]-1)*np.cos(2*np.pi*fc*(i*bit_len+j))+(2*s[i+1]-1)*np.sin(2*np.pi*fc*(i*bit_len+j)))
```

Figure 2.1: The modulation process

The cosine and sine functions are used here to implement phase shifting, where fc is the carrier frequency and i*bit_len+j is the time point. Through this method, the phase of the signal changes between 0 degrees, 90 degrees, 180 degrees and 270 degrees according to the values of s[i] and s[i+1], achieving QPSK encoding.

The results

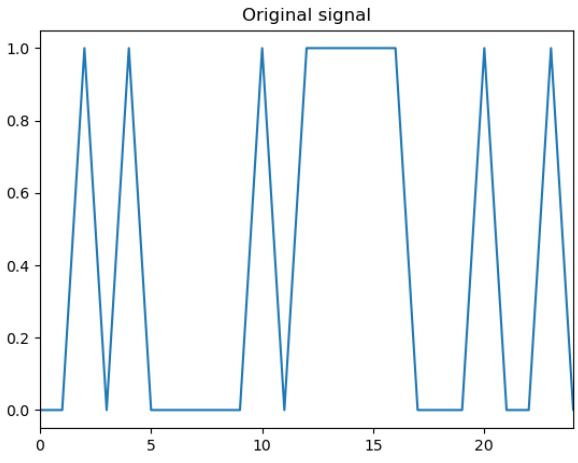

Figure 2.2: The original signal

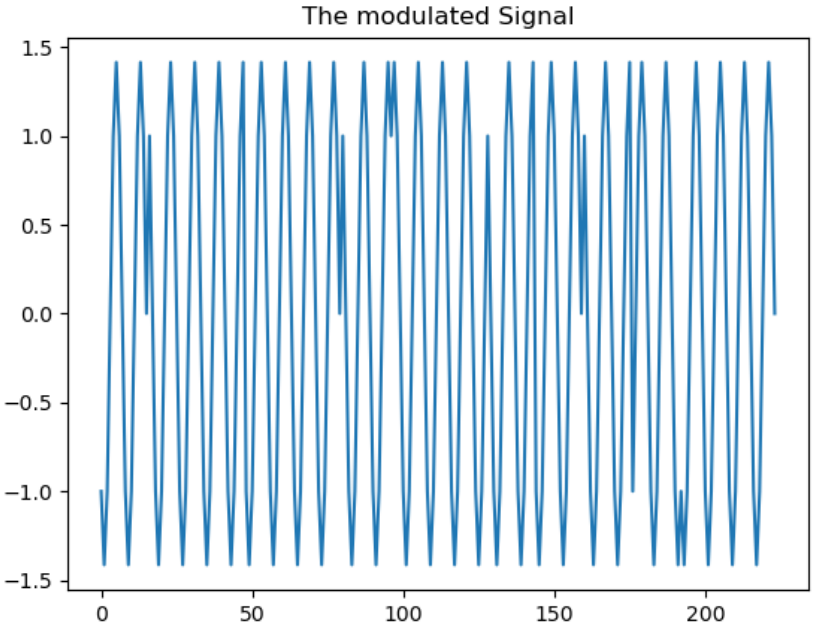

Figure 2.3: The modulated signal

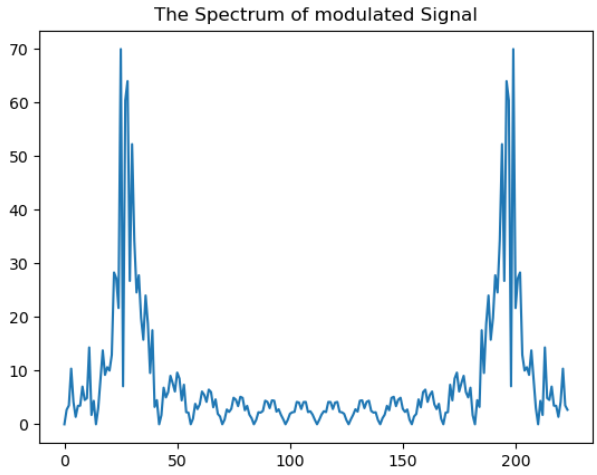

Figure 2.4: The spectrum of modulated Signal

Demodulation

Just like BPSK, but in QPSK we need to demodulate the cos carrier wave and sin carrier wave separately.

What I have Done

```
for i in range(0,Nbits,2):
    for j in range(bit_len):
        rx_demod_cos = np.append(rx_demod_cos,tx_mod[t]*np.cos(2*np.pi*fc*t))
        rx_demod_sin = np.append(rx_demod_sin,tx_mod[t]*np.sin(2*np.pi*fc*t))
        t += 1
```

Figure 2.5: The QPSK Demodulation Processing

In this piece of code, rx_demod_cos and rx_demod_sin are variables used to store the demodulated signal. In the loop, demodulation is achieved by multiplying the modulated signal with the cosine and sine functions.
And then the two signals pass the LPF, so we get the original signal. But we still need to use a threshold function to deal with the original signal.

The results

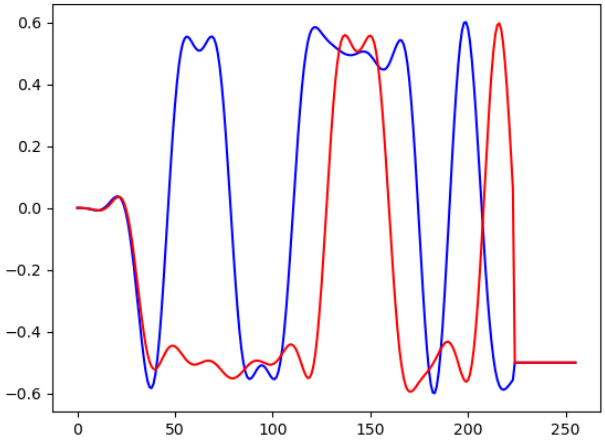

Figure 2.6: Modulated signal * coswct passed LPF

The threshold function will be applied here to make this signal more like a digital signal.

```
# The threshold function
rx_bin = np.empty(0)
for i in range(0,Nbits,2):
    t = (i+1)*bit_len//2 + numtaps//2
    rx_bin = np.append(rx_bin,rx_filt_cos[t] > 0.0)
    rx_bin = np.append(rx_bin,rx_filt_sin[t] > 0.0)
```

Figure 2.7: The threshold function

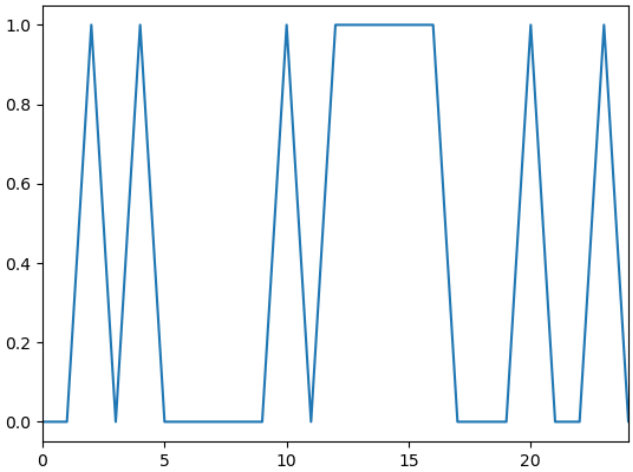

Figure 2.8: The demodulated signal

Conclusion

The problem I met

When my binary sequence is 24-bit binary, there is always a problem when demodulating the last bit. So I added some zeros to the original sequence at the end.

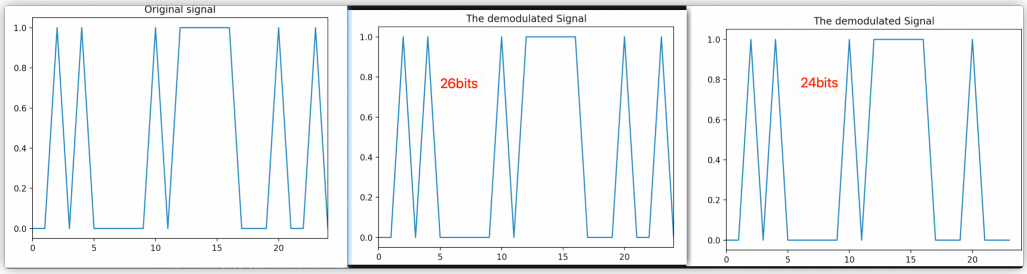

Figure 3.1: The problem

Which one is more efficient?

From figures 1.4 and 2.4, it is implied The sideband power of the signal after BPSK encoding is lower than the QPSK encoding. QPSK is more efficient because the sidebands carry more information than the carrier frequency.

Advantages and disadvantages of QPSK and BPSK

BPSK (Binary Phase Shift Keying):
Advantages: Simple, easy to implement, high bandwidth efficiency, only two possible symbol states, low error rate
Disadvantages: Low symbol rate, high signal-to-noise ratio requirements

QPSK (Quadrature Phase Shift Keying):
Advantages: High symbol rate, low signal-to-noise ratio requirements, more tolerant to multipath fading and interference
Disadvantages: Complex, difficult to implement, lower bandwidth efficiency

Appendix

BPSK

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 23 18:32:03 2023

@author: ranshuai
"""
# BPSK
import numpy as np
from matplotlib import pyplot as plt
from scipy import fft
from scipy import signal
def bin_array(num, m):
    """Convert a positive integer num into an m-bit bit vector"""
    return np.array(list(np.binary_repr(num).zfill(m))).astype(np.bool)
# import 24 bit digital data
id_num = 2633609
Nbits = 24
tx_bin = bin_array(id_num, Nbits) # the 24 bit binarry array
# When transmitting the last signal, errors always occur,
# so I add several zero values to the original signal, making the last signal the
third last signal.
# tx_bin = np.append(tx_bin,0)
# tx_bin = np.append(tx_bin,0)
# Nbits = Nbits + 2
##################################################
```

```
bit_len = 16 #
fc = 0.125 # normalized frequency
s = np.copy(tx_bin) # s = original signal
print(tx_bin)
tx_mod = np.empty(0) # The modulated signal
plt.figure()
plt.plot(tx_bin)
plt.title("Original signal")
plt.xlim(0,24)
plt.show()

################ modulation################
for i in range(0,Nbits):
        for j in range(bit_len):
                tx_mod = np.append(tx_mod,(2*s[i]-
1)*np.cos(2*np.pi*fc*(i*bit_len+j)))# signal * coswct carrier frequency

plt.figure()
plt.plot(tx_mod)
plt.title("The modulated Signal")
plt.show()

plt.figure()
plt.plot(np.abs(fft.fft(tx_mod))) # converting a complex number to its
magnitude.
plt.title("The Spectrum of modulated Signal")
plt.show()


##########demodulation###############
```

```
numtaps = 64
delays = np.arange(numtaps) #
b1 = signal.firwin(numtaps, 0.1) # 0.1 means the cut off frequency which is
normalised
rx_demod = np.empty(0)
for i in range(Nbits):
        for j in range(bit_len):
                rx_demod =
np.append(rx_demod,tx_mod[i*bit_len+j]*np.cos(2*np.pi*fc*(i*bit_len+j)))#
# the process of modulating process :s(t)*cos"*cos
plt.figure()
plt.plot(np.abs(fft.fft(rx_demod))) #因为是傅立叶变换是 complex quantity 复
数 方便在 spectrum analyser 上查看
plt.title("Spectrum of Modulated Signal * cos(wct)")
plt.show()
rx_filt = signal.lfilter(b1,1,rx_demod)
rx_filt = np.append(rx_filt,-np.ones(numtaps//2))

plt.figure()
plt.plot(rx_filt)
plt.title("Signal which passed Lpf")
plt.show()

demodulated_signal = np.empty(0) # modulated signal
for i in range(Nbits):
        t = (2*i+1)*bit_len//2 + numtaps//2
        demodulated_signal = np.append(demodulated_signal,rx_filt[t] > 0.0) #
threshold function

plt.figure()
```

```
plt.plot(demodulated_signal)
plt.title("The demodulated Signal")
plt.xlim(0,24)
plt.show()
```

QPSK

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 23 18:32:03 2023

@author: ranshuai
"""

# QPSK
import numpy as np
from matplotlib import pyplot as plt
from scipy import fft
from scipy import signal
def bin_array(num, m):
    """Convert a positive integer num into an m-bit bit vector"""
    return np.array(list(np.binary_repr(num).zfill(m))).astype(np.bool)
# import 24 bit digital data
```

```
id_num = 2633609
Nbits = 24
tx_bin = bin_array(id_num, Nbits) #
bit_len = 16 #
fc = 0.125 # normalized frequency
# When transmitting the last signal, errors always occur,
# so I add several zero values to the original signal, making the last signal the
fifth last signal.
tx_bin = np.append(tx_bin,0)
tx_bin = np.append(tx_bin,0)
tx_bin = np.append(tx_bin,0)
tx_bin = np.append(tx_bin,0)
Nbits = Nbits + 4
################################################


s = np.copy(tx_bin) # s = original signal
tx_mod = np.empty(0) # the modulated signal
plt.figure()
plt.plot(tx_bin)
plt.title("Original signal")
plt.xlim(0,24)
plt.show()


################ modulation################
for i in range(0,Nbits,2):
        for j in range(bit_len):
                tx_mod = np.append(tx_mod,(2*s[i]-
1)*np.cos(2*np.pi*fc*(i*bit_len+j))+(2*s[i+1]-1)*np.sin(2*np.pi*fc*(i*bit_len+j)))#
把信号追加到 tx_mod 中
```

```
plt.figure()
plt.plot(tx_mod)
plt.title("The modulated Signal")
plt.show()

plt.figure()
plt.plot(np.abs(fft.fft(tx_mod))) #converting a complex number to its
magnitude.
plt.title("The Spectrum of modulated Signal")
plt.show()


##########Low Pass Filter################
numtaps = 64
delays = np.arange(numtaps) # taos
b1 = signal.firwin(numtaps, 0.1) # coefficients of the LPF
#########demodulation#################
rx_demod_cos = np.empty(0)
rx_demod_sin = np.empty(0)
t = 0
## The demodulated processing
for i in range(0,Nbits,2):
      for j in range(bit_len):
            rx_demod_cos =
np.append(rx_demod_cos,tx_mod[t]*np.cos(2*np.pi*fc*t))
            rx_demod_sin =
np.append(rx_demod_sin,tx_mod[t]*np.sin(2*np.pi*fc*t))
            t += 1
# Low pass filter
rx_filt_cos = signal.lfilter(b1,1,rx_demod_cos)
```

```
rx_filt_cos = np.append(rx_filt_cos,-np.ones(numtaps//2)/2)
rx_filt_sin = signal.lfilter(b1,1,rx_demod_sin)
rx_filt_sin = np.append(rx_filt_sin,-np.ones(numtaps//2)/2)

plt.figure()
plt.plot(rx_filt_cos,color = "b")
plt.plot(rx_filt_sin,color = "r")
plt.show()

# The threshold function
rx_bin = np.empty(0)
for i in range(0,Nbits,2):
        t = (i+1)*bit_len//2 + numtaps//2
        rx_bin = np.append(rx_bin,rx_filt_cos[t] > 0.0)
        rx_bin = np.append(rx_bin,rx_filt_sin[t] > 0.0)

rx_bin = rx_bin[:-1]
plt.figure()
plt.plot(rx_bin)
plt.xlim(0,24)
plt.show()
```