University of Glasgow | James Watt School of Engineering

ENG4052: Digital Communication 4 (2022-23)

Lab 2: Carrier Recovery using Costas Loop

Ran Shuai (2633609R)

Content Tables

Introduction

What is Carrier Frequency

Carrier recovery is the process of recovering the carrier from the modula
ted signal.

Why use carrier recovery

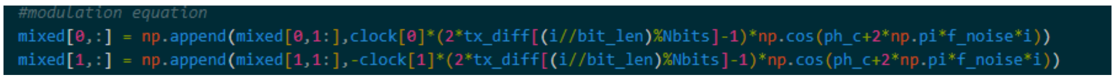
The purpose of carrier recovery is to correct the errors that may occur i
n the transmission process of the carrier. Therefore, the carrier recovere
d from the modulated signal can more accurately reflect the situation in t
he transmission process. In this way, the data in the modulated signal c
an be decoded more accurately.

In the code, I will use the f_noise to simulate the signal which has been
changed during the transmitting process. The variable dout is used to si
mulate the adapting signal which should be the same as the f_noise aft
er the Costas Loop. Finally, dout will be used to demodulate the signal.

The Modulated Part

In this Lab, we also use the BPSK to modulate the signal. But the mod ulated signal is not an important part of this lab. In this lab, the most im portant part is to recover the carrier frequency from the modulated signa l.

The code to modulate the signal



```
#modulation equation
mixed[0,:] = np.append(mixed[0,1:],clock[0]*(2*tx_diff[(i//bit_len)%Nbits]-1)*np.cos(ph_c+2*np.pi*f_noise*i))
mixed[1,:] = np.append(mixed[1,1:],-clock[1]*(2*tx_diff[(i//bit_len)%Nbits]-1)*np.cos(ph_c+2*np.pi*f_noise*i))
```

This figure shows that f_noise is used here to modulate the signal. And f_noise is the carrier frequency to which we add some random noise. La ter we need to use an adapting carrier frequency to demodulate the sign al which is modulated by the f_noise. This means the adapting signal sh ould be the same as the f_noise to get the correct demodulated signal by the adapting signal.

The Demodulated Part

To get the correct demodulated signal we should first recover the correct carrier frequency from the modulated signal. So recovering the carrier frequency is the most important part of the demodulation.

What is Costas Loop

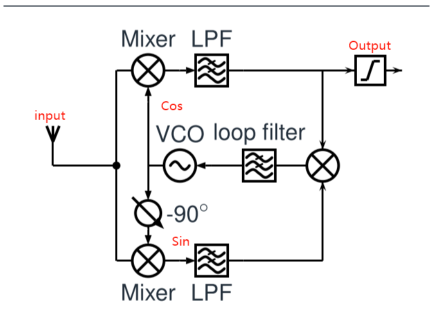


Figure 1.1The Costas Loop

This figure shows the process of the recovery frequency with Costas Loop.

The input is the modulated signal:

$$s(t)\cos(w_{ref}t+\theta_{ref})$$

Cos and Sine is the signal produced by VCO.

After the LPFs, we get the signal:

$$\frac{1}{2}s(t)\cos(\Delta wt+\Delta\theta)$$

$$-\frac{1}{2}s(t)\sin(\Delta wt+\Delta\theta)$$

Because we have already the high frequency $w_{ref}+w_i$.

Then we multiply these two signals, and get the signal:

$$-\frac{1}{8}s^2(t)\sin 2(\Delta wt+\Delta\theta)$$

When the adapting and the reference signal has the same frequency, we will get a signal:

$$\mathrm{Sin}(2\Delta\theta)$$

In the VCO, $Sin(2\Delta\theta)$ will be used to drive VCO.

VCO Cordic Digital Clock

What is VCO used for

Typically, the Costas Loop uses a VCO (variable oscillation circuit) to pro duce the phase reference signal. Costa Loop fine-tunes the VCO by co mparing the phase of the demodulated signal and the phase reference si gnal produced by the VCO to recover the proper carrier signal as the fr equency and phase of the VCO change over time.

The code for VCO

```
    c = np.cos(2*np.pi*f_ideal*(1.+0.25*volt))

    s = np.sin(2*np.pi*f_ideal*(1.+0.25*volt))

    clock = np.matmul(np.array([[c, -s], [s, c]]), clock)
```

The code here is used to produce the cos and sine part of VCO.

The variable v

```
lpmixed = [np.sum(b1*mixed[j,:]) for j in range(2)]

    volt = lpmixed[0]*lpmixed[1]
```

This is the code to get v for the VCO. In my code v is represented by a volt. This operation multiplies the sin and cos signals which have pass ed the Low pass filter.

The variable α

The variable α here, I make it equal to 0.25

The formula $\boldsymbol{f_0 = f_i + \alpha v}$

In this formula, f_0 means the adapting frequency which I use the variable cout to represent this signal. Finally, f_0 should be the same as the reference signal.

Random phase and frequency

In this part, the ideal frequency is added by some Random phase and frequency. And we call this frequency reference frequency. But in the code, I use f_noise to represent it.

```
f_noise =   f_ideal*(1.+0.02*(random.rand()-0.5))
# Add some phase error to the ideal carrier frequency to simultae the error carrier
frequency
ph_c = 2*np.pi*random.rand()
```

Later the c_out signal should be the same as f_noise

Differential Coding

What is the differential Coding

It is a digital signal processing technology, which reduces data redundancy and error by differential coding of adjacent digital signals.

Why do we use Differential Coding

Differential coding is usually used to reduce data redundancy and error and to improve the stability and reliability of data transmission or storage.

The results

Data input and output

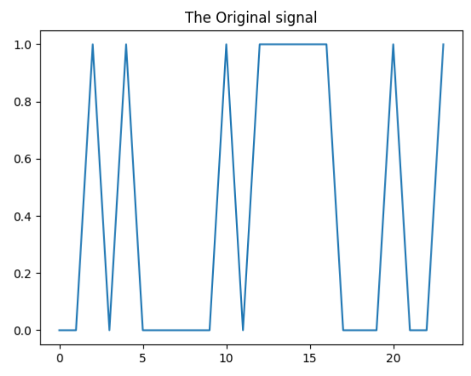


Figure 2.1: The original signal

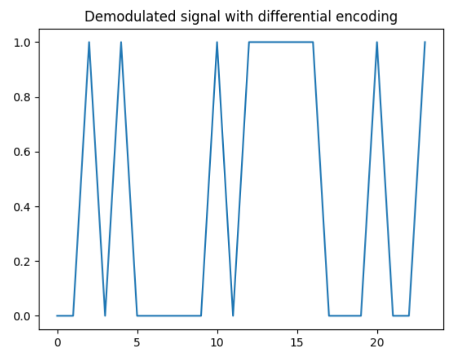


Figure 2.2: The Demodulated Signal

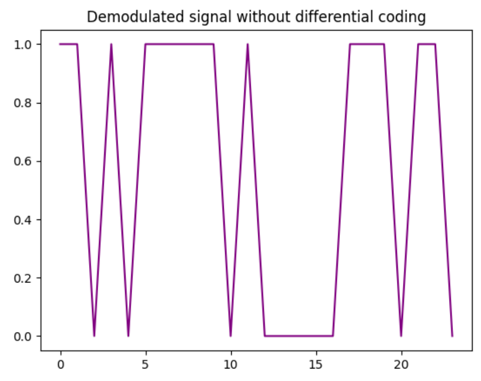


Figure 2.3: The demodulated signal without Differential Coding

Clock Output with Reference Carrier

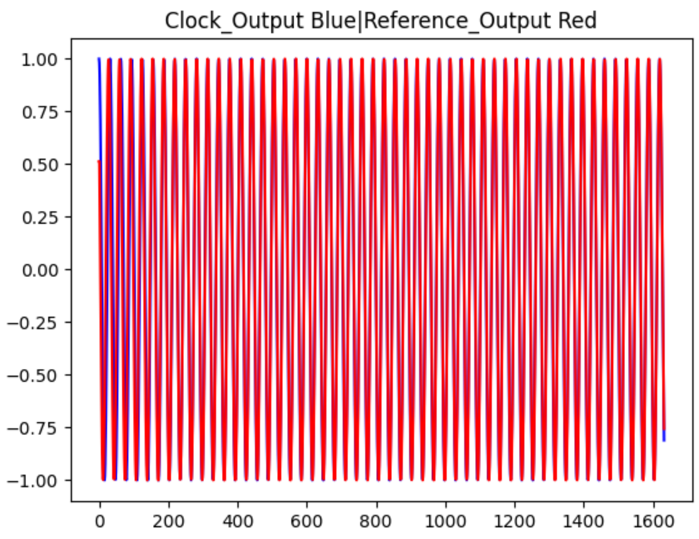


Figure 2.4: Clock Output with Reference Carrier

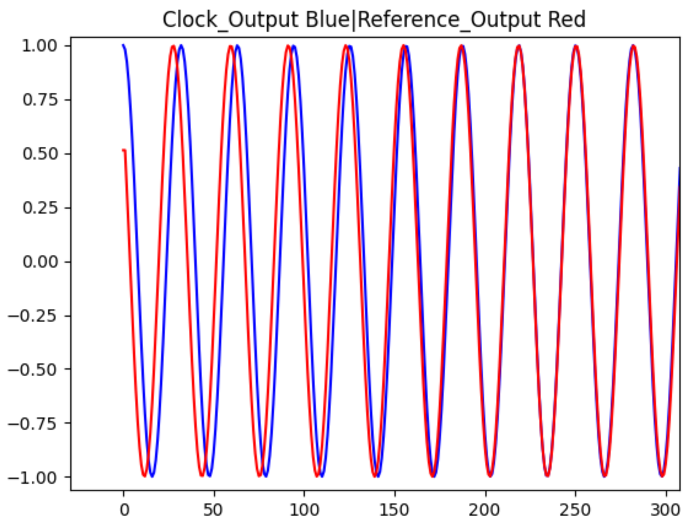


Figure 2.5: Clock Output with Reference Carrier which is zoomed in

Voltage driving the VCO

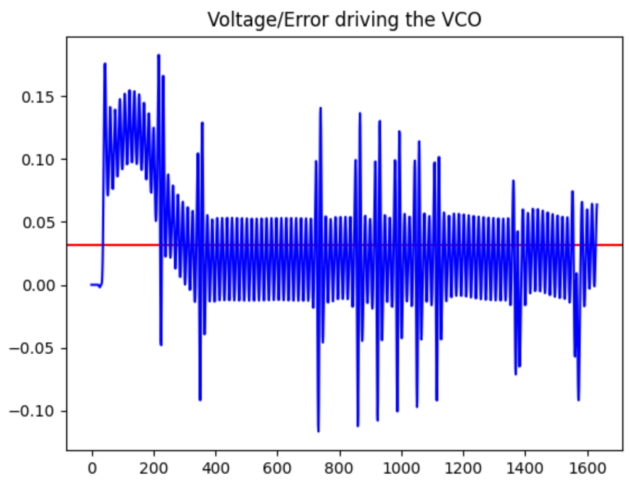


Figure 2.6: The voltage driving the VCO

Output data before thresholding

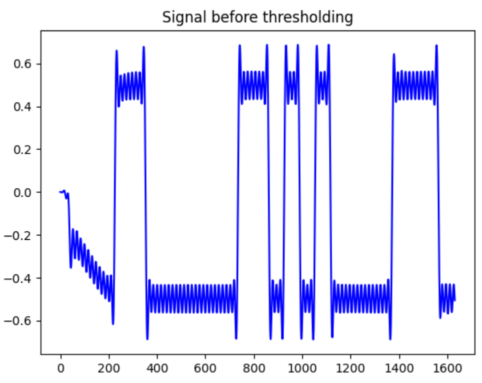


Figure 2.7: Output data before thresholding

The conclusion

The difference between with and without differential coding

From figures 2.1, 2.2, and 2.3, we can see that the code with differential coding can get the correct demodulated signal. However, the code without differential coding gets the wrong demodulated signal which is reversed compared with the original signal.
I think this error appears because in the process of digital signal transmission, noise and error may occur, resulting in changes in the original signal at the receiving end. If differential coding is not used, the receiver will not be able to accurately determine the value of the digital signal and may cause the signal to reverse.

Why we need to use carrier frequency

Carrier recovery is used in digital communication to correct carrier phase and frequency errors to ensure the accuracy of digital signals in the transmission process.

Digital signals are sent via carrier technology, which also includes carrier frequency and phase. However, the carrier may be impacted by several elements during the digital signal transmission process, such as jitter, distortion, interference, etc., leading to carrier phase and frequency problems.

To fix the error and guarantee the accuracy of the digital signal during transmission, it is crucial to restore the carrier at the receiving end. The receiver might not be able to correctly decode the digital signal, leading to transmission error, if the carrier is not recovered.

Appendix

```
'''
Date         : 2023/02/07
LastEditors  : RanShuai
'''
import numpy as np
from matplotlib import pyplot as plt
from scipy import fft
from scipy import signal
from numpy import random

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(bool)
# tranfer student number to 24 bit ninary data
id_num = 2633609
Nbits = 24
tx_bin = bin_array(id_num, Nbits) # the original data

# The Original signal
plt.figure()
plt.title('The Original signal')
plt.plot(tx_bin)
plt.show()

s = tx_bin # copy the original signal for Differential coding
# carrier frequency
f_ideal = 1/32 # the normalised ideal frequency
# bit length
bit_len = 64

# turn on or off the  differential coding operation
flagDiffCoding = True
if (flagDiffCoding):
    #Differential Coding of tx_bin
    tx_diff = np.zeros(1, dtype='bool')
    for i in range(Nbits):
        tx_diff = np.append(tx_diff, tx_diff[i]^s[i])
    Nbits = Nbits+1
else:
    tx_diff = tx_bin
```

```
# low-pass filter
numtaps = 64 # taps
b1 = np.flip(signal.firwin(numtaps, 0.1))    # the filter coefficients


clock = np.array([1.0,0.0]) # the original value of  VCO Cosoutput and
Sine_output
# Add some randome noise to the ideal carrier frequency
f_noise =  f_ideal*(1.+0.02*(random.rand()-0.5))
# Add some phase error to the ideal carrier frequency to simultae the
error carrier frequency
ph_c = 2*np.pi*random.rand()
volt = 0.0 # The error to drive the VCO
vout = np.array(volt)# The error to drive the VCO
# out_array of clock(recovery clock)

cout = clock[0] # the cos part which is not passed LPF
# out_array of reference clock
rout = np.cos(ph_c)
# Data outpur of c_output
dout = np.empty(0)

mixed = np.zeros((2,numtaps))
for i in range(0, bit_len*Nbits + numtaps//2):
    #modulation equation
    mixed[0,:] =
np.append(mixed[0,1:],clock[0]*(2*tx_diff[(i//bit_len)%Nbits]-
1)*np.cos(ph_c+2*np.pi*f_noise*i))
    mixed[1,:] = np.append(mixed[1,1:],-
clock[1]*(2*tx_diff[(i//bit_len)%Nbits]-
1)*np.cos(ph_c+2*np.pi*f_noise*i))
    # Filtering of mixed data
    lpmixed = [np.sum(b1*mixed[j,:]) for j in range(2)]
    volt = lpmixed[0]*lpmixed[1]

    c = np.cos(2*np.pi*f_ideal*(1.+0.25*volt))
    s = np.sin(2*np.pi*f_ideal*(1.+0.25*volt))
    clock = np.matmul(np.array([[c, -s], [s, c]]), clock)

    vout = np.append(vout, volt)
    cout = np.append(cout, clock[0])
    rout = np.append(rout, np.cos(ph_c+2*np.pi*f_noise*i)) # the noise
carrier frequncy red
```

```python
    dout = np.append(dout, lpmixed[0]) # the adapting frequency which
should be exactly the same as the noise carrier frequency  blue

plt.figure()
plt.title('Voltage/Error driving the VCO')
plt.axhline(f_noise,color='r')
plt.plot(vout, color='b')
plt.show()

plt.figure()
plt.title('Clock_Output Blue|Reference_Output Red')
plt.plot(cout,color='b')
plt.plot(rout, color='r')
plt.show()

plt.figure()
plt.title('Signal before thresholding')
plt.plot(dout,color='b')
plt.show()

# two parts which do or not do the differential coding
# This will cause the reverse demodulated signal
if (flagDiffCoding):
    rx_diff = np.empty(0)
    for i in range(Nbits):
        #select an appropriate sample point
        k = (2*i+1)*bit_len//2 +numtaps//2
        rx_diff= np.append(rx_diff, np.heaviside(dout[k],0))

    rx_bin = np.empty(0, dtype='bool')
    Nbits = Nbits-1
    for i in range(Nbits):
        rx_bin = np.append(rx_bin,
rx_diff[i].astype(bool)^rx_diff[i+1].astype(bool))

    # print(rx_bin)
    plt.figure()
    plt.title('Demodulated signal with differential encoding')
    plt.plot(rx_bin)
    plt.show()
else:
    rx_bin = np.empty(0, dtype='bool')
    for i in range(0,Nbits):
        t = (2*i+1)*bit_len//2 +numtaps//2
        rx_bin = np.append(rx_bin, np.heaviside(dout[t],0))

# judge if the demodulated signal is the same as the original signal
# if not draw the orange  wrong pic
    if ((rx_bin != tx_bin).any()):
        plt.figure()
        plt.title('Demodulated signal without differential coding')
        plt.plot(rx_bin, color='purple')
        plt.show()
    else:
        plt.figure()
        plt.title('Demodulated signal without differential coding')
        plt.plot(rx_bin)
        plt.show()
```