****

ENG4052: Digital Communication 4 (2022-23)

Lab 2: Carrier Recovery using Costas Loop

**Ran Shuai (2633609R)**

Content Tables

[Introduction 3](#_Toc126788306)

[What is Carrier Frequency 3](#_Toc126788307)

[Why use carrier recovery 3](#_Toc126788308)

[The Modulated Part 4](#_Toc126788309)

[The code to modulate the signal 4](#_Toc126788310)

[The Demodulated Part 5](#_Toc126788311)

[What is Costas Loop 5](#_Toc126788312)

[VCO Cordic Digital Clock 6](#_Toc126788313)

[What is VCO used for 6](#_Toc126788314)

[The code for VCO 6](#_Toc126788315)

[The variable 6](#_Toc126788316)

[The variable 6](#_Toc126788317)

[The formula 7](#_Toc126788318)

[Random phase and frequency 7](#_Toc126788319)

[Differential Coding 7](#_Toc126788320)

[What is the differential Coding 7](#_Toc126788321)

[Why do we use Differential Coding 7](#_Toc126788322)

[The results 8](#_Toc126788323)

[Data input and output 8](#_Toc126788324)

[Clock Output with Reference Carrier 9](#_Toc126788325)

[Voltage driving the VCO 10](#_Toc126788326)

[Output data before thresholding 10](#_Toc126788327)

[The conclusion 11](#_Toc126788328)

[The difference between with and without differential coding 11](#_Toc126788329)

[Why we need to use carrier frequency 11](#_Toc126788330)

[Appendix 12](#_Toc126788331)

# Introduction

## What is Carrier Frequency

Carrier recovery is the process of recovering the carrier from the modulated signal.

## Why use carrier recovery

The purpose of carrier recovery is to correct the errors that may occur in the transmission process of the carrier. Therefore, the carrier recovered from the modulated signal can more accurately reflect the situation in the transmission process. In this way, the data in the modulated signal can 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 simulate the adapting signal which should be the same as the f\_noise after 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 modulated signal is not an important part of this lab. In this lab, the most important part is to recover the carrier frequency from the modulated signal.

## The code to modulate the signal

C:\Users\RanShy\AppData\Local\Temp\tianruoocr\截图_20230208202316.png

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. Later we need to use an adapting carrier frequency to demodulate the signal which is modulated by the f\_noise. This means the adapting signal should 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

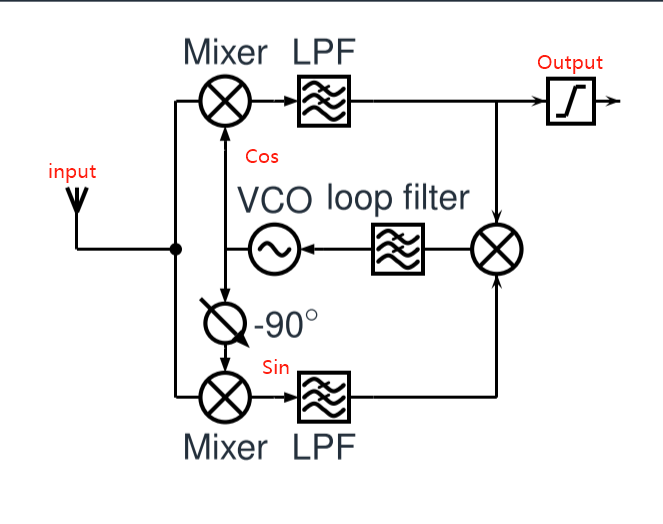


Figure 1.1The Costas Loop

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

The input is the modulated signal:

Cos and Sine is the signal produced by VCO.

After the LPFs, we get the signal:

Because we have already the high frequency .

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

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

In the VCO, 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 produce the phase reference signal. Costa Loop fine-tunes the VCO by comparing the phase of the demodulated signal and the phase reference signal produced by the VCO to recover the proper carrier signal as the frequency 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

 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 passed the Low pass filter.

### The variable

The variable here, I make it equal to 0.25

### The formula

In this formula, means the adapting frequency which I use the variable cout to represent this signal. Finally, 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

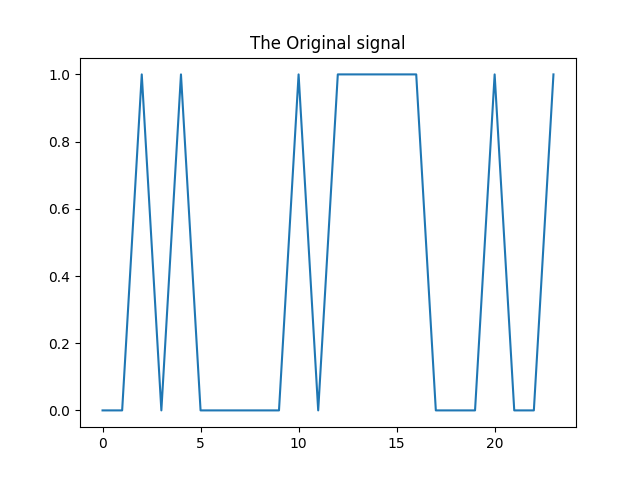


Figure 2.1: The original signal

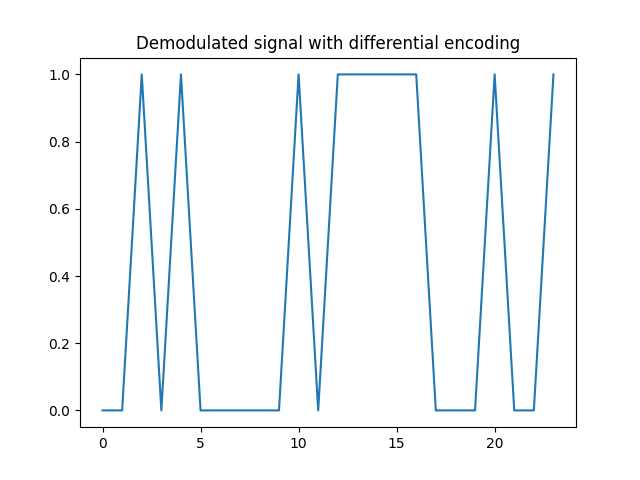


Figure 2.2: The Demodulated Signal

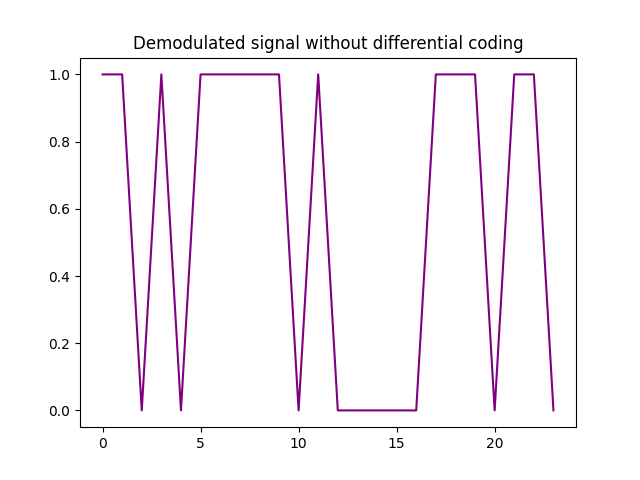


Figure 2.3: The demodulated signal without Differential Coding

## Clock Output with Reference Carrier

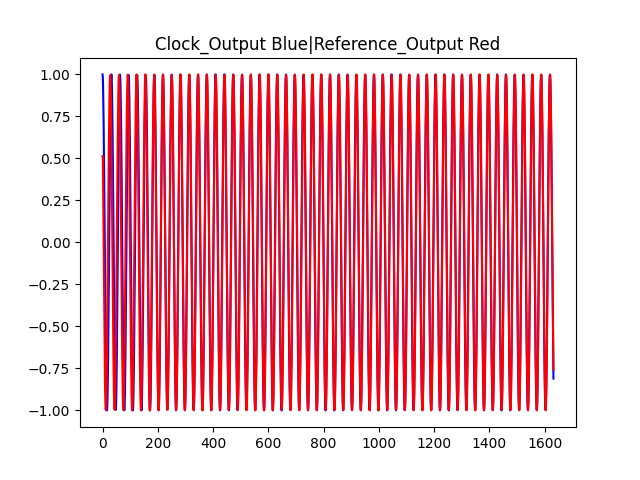


Figure 2.4: Clock Output with Reference Carrier

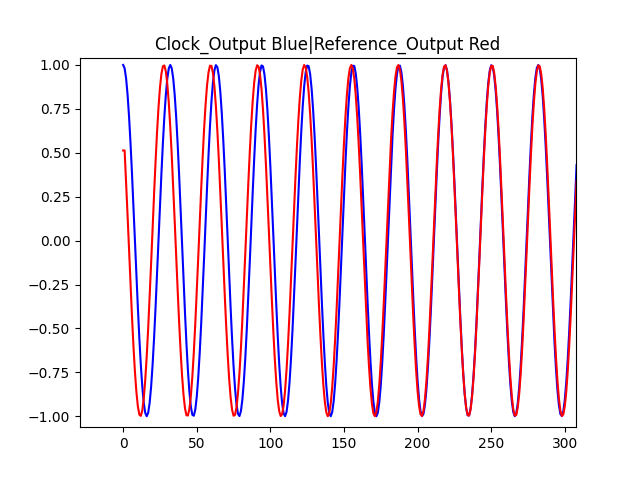


Figure 2.5: Clock Output with Reference Carrier which is zoomed in

## Voltage driving the VCO

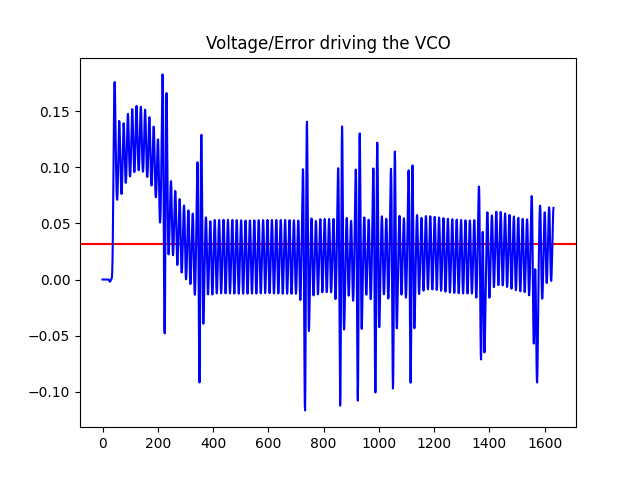


Figure 2.6: The voltage driving the VCO

## Output data before thresholding

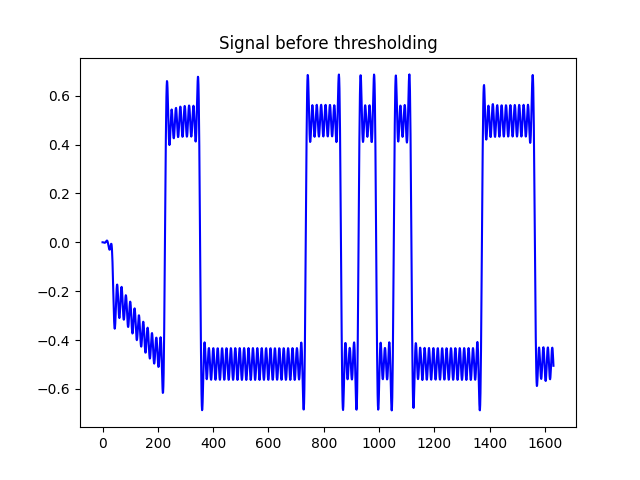


Figure 2.7: Output data before thresholding

# 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*

    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()