University of Glasgow | James Watt School of Engineering

1 Digital Communications 4: Carrier Recovery using Costas Loop

1.1 Introduction

This coding project will introduce you to Carrier Recovery. We will be coding in the python programming language in order to make use of a bespoke library for later projects. If you are using your own computer, make sure the Python libraries `scipy`, `numpy`, and `matplotlib` are installed. It is recommended that you use a suitable IDE for your project, such as Spyder, PyCharm or Visual Studio. Tip: if you are using Spyder, you can display, manipulate and save graphics in separate windows by setting

```
Tools>Preferences>IPython console>Graphics>Backend
```

from `Inline` to `Automatic`.

Each project will be scheduled over a two week period, within which there will be 2 scheduled online consultation sessions where you will be able to ask teaching staff for guidance. The project should be written up as a short report describing what you've done and the results you have taken along with any conclusions that you draw. Include your python code(s) in the appendices. Make sure your name and student ID number is on the report. The report should be uploaded to the Moodle assignment by the stated deadline, either using Moodle's inbuilt html editor, or as a single PDF file.

1.2 VCO Cordic Digital Clock

Code an algorithm for a voltage controlled digital clock. The current state of the clock is signified by 2 floating point numbers representing the in-phase and quadrature components sampled values. Then each successive state is obtained by the Cordic transformation using $\cos f_0$ and $\sin f_0$, where $f_0 = f_i + \alpha v$ is the scaled voltage dependent frequency, including a linear applied voltage dependence. Take your previous value for the carrier frequency as the initial value f_i. Check that your clock code is functional by inspecting a plot of the sampled in-phase and quadrature components with sample number.

1.3 Demodulation with Carrier Recovery

It is recommended that you take as your starting point for remainder of this project a copy of your first laboratory BPSK Modulation/Demodulation code. Your input data will remain as the 24 bit representation of your student ID number. Perform modulation in your nested loops as before. The Costas loop involves mixing the received modulated waveform with the VCO outputs, and then low-pass filtering the results. This does require a bit more thought than in the simple demodulation examples, as the VCO is being dynamically driven whilst the digital filtering is taking place, so the `lfilter` function cannot be simply deployed as in the previous

University of Glasgow | 詹姆斯*瓦特工程学院

1 数字通信4：使用Costas的载波恢复 Loop

1.1 Introduction

这个编码项目将向您介绍载波恢复.我们将使用python编程语言进行编码，以便为以后的项目使用定制的库。如果您使用的是自己的计算机，请确保安装了Python库scipy，numpy和matplotlib。建议您为项目使用合适的IDE，例如Spyder，PyCharm或VisualStudio。提示：如果您使用的是Spyder，则可以通过设置在单独的窗口中显示，操作和保存图形

```
Tools>Preferences>IPython console>Graphics>Backend
```

从内联到自动。
每个项目将安排在两个星期内，在这两个星期内将有两个预定的在线咨询会议，你将能够要求教师指导。这个项目应该写成一份简短的报告，描述你所做的事情和你所取得的结果以及你得出的任何结论。在附录中包含您的python代码。确保您的姓名和学生证号在报告上。报告应该在规定的截止日期之前上传到Moodle分配，或者使用moodle的内置html编辑器，或者作为单个PDF文件。

1.2 VcoCordic数字时钟

码电压控制数字时钟的算法。时钟的当前状态由2个浮点数表示，代表同相和正交分量采样值。然后通过Cordic变换使用cos∈0和sin∈0获得每个连续状态，其中∈0=∞+∞是缩放的电压依赖性频率，包括线性施加的电压依赖性。将载波频率的先前值作为初始值 。通过检查采样同相和正交分量的图以及样本编号，检查时钟代码是否正常运行。

1.3 载波恢复解调

建议您将第一个实验室BPSK调制解调代码的副本作为本项目剩余部分的起点。您的输入数据将保留为您的学生证号码的24位表示.像以前一样在嵌套循环中执行调制。Costas环路包括将接收到的调制波形与VCO输出混合，然后对结果进行低通滤波。这确实需要比简单解调示例更多的考虑，因为VCO是动态驱动的，而数字滤波正在进行，所以lfilter功能不能像以前那样简单地部署

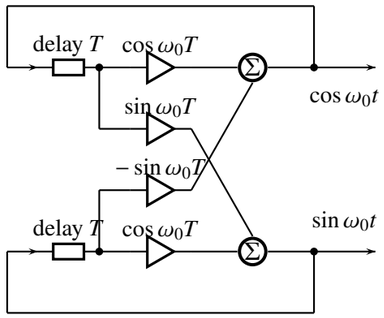

Figure 1: Digital Clock by Cordic Algorithm

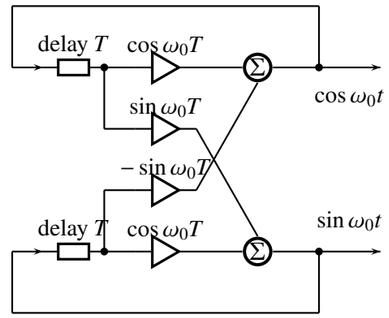

图1：数字时钟通过Cordic算法

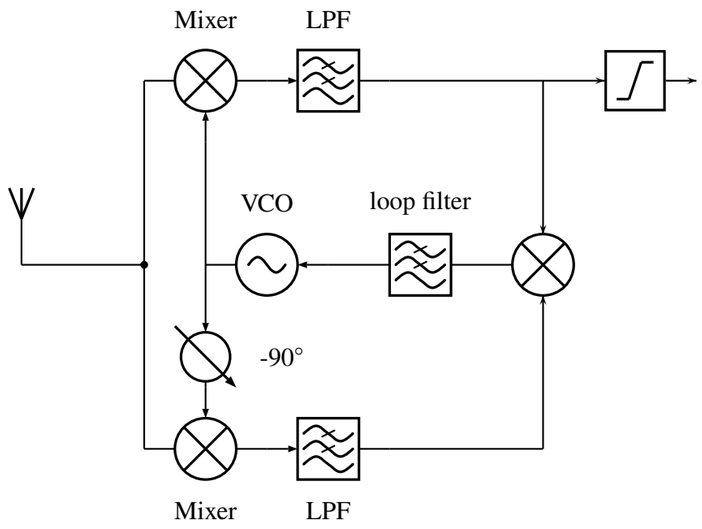

Figure 2: Block diagram of a Classical Costas Loop

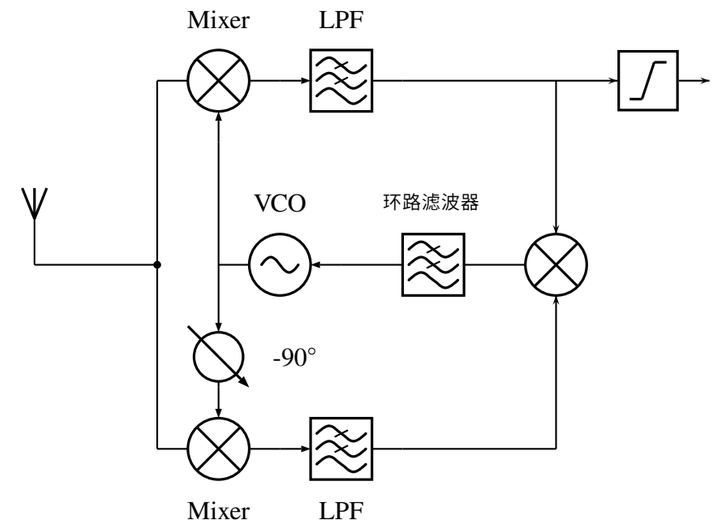

图2：经典Costas环路的框图

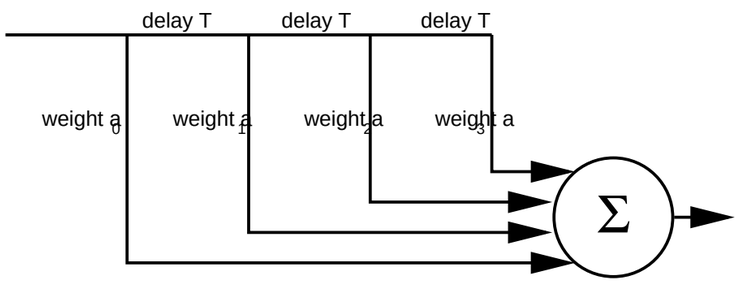

Figure 3: FIR Digital Filter

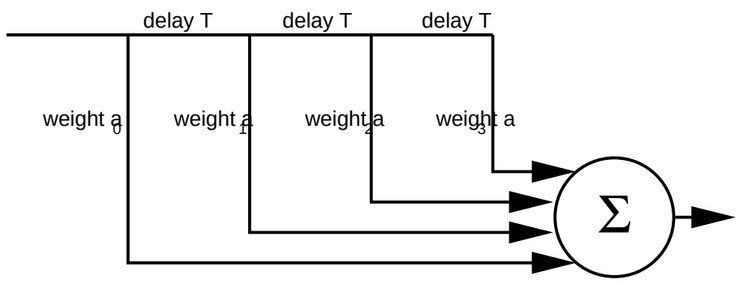

图3：FIR数字滤波器

exercise. Therefore we should consider the discrete convolution that underlies digital filtering using an FIR.

$$[f \otimes a](n) = \sum_{i=0}^{N-1} f(n-i)a(i)$$

So to get the nth sampled filter result we need the $f(n)$ to $f(n+1-N)$ filter inputs and the $a(0)$ to $a(N-1)$ tap weights, with N being the number of filter taps. Therefore we need an array (for each branch) which stores the last N LPF input samples, and this array should be initialised before the demodulation loops.

```
c_mixed = np.zeros(numtaps)
for ...
    for ...
        c_mixed[:] = np.append(c_mixed[1:], c_lpf_input)
        c_lpf_output = np.sum(b1*c_mixed)
```

We will use low-pass FIR filters in the Costas loop, so obtain the tap weights in the initialisation as previously, but with a flip in the ordering to account that it is the last term in `c_mixed` defined above which corresponds to zero delay, e.g.

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

The Costas Loop block diagram in figure **??** shows that the two LPF outputs are multiplied at a mixer,

```
volt = c_lpf_output*s_lpf_output
```

and then passed through a loop filter. As we have already employed low pass filters with a high rolloff-rate, it is normally not necessary to include a further frequency selective element, but we we should ensure that we set the gain/attenuation appropriately through the VCO α parameter, so you may need to experiment till you find a suitable value.

The final step in the demodulation of digital data is to select an appropriate sample point in `c_lpf_output`, and then use a thresholding function to convert floating point into boolean. We shall again use the most obvious sample point by taking the midpoint of the bit, i.e. `i*bit_len+bit_len//2`. Therefore construct a for loop over `Nbits` and applying a threshold to the bit midpoint. In this case an appropriate threshold function is the heaviside (step) function

```
rx_bin[i] = np.heaviside(c_lpf_output[i], 0)
```

Finally, compare your output binary data with your input data. Do you notice any issues?

You should observe a delay in the output data when compared to the input data. This is because the digital filter comprises delays within each tap, and the filter designs resulting from `signal.firwin` normally have an overall delay of `numtaps//2`. Account for this delay in your code; you will need to add at least `numtaps//2` dummy samples to the end of your mixed signal data prior to filtering.

You will also find that it takes a number of symbols for the Costas loop to lock on the input. Therefore you will also need to preface your digital data with a number of dummy symbols to cover this locking delay. You may want to include a unique pattern so that your data can be identified, say a number of zero bits followed by a single one bit.

运动。因此，我们应该考虑使用FIR进行数字滤波的离散卷积。

$$[f \otimes a](n) = \sum_{i=0}^{N-1} f(n-i)a(i)$$

因此，获得 个取样过滤器的结果是我们需要 ( )以 (+1)筛选输入和 (0) (1)挖掘的权的数量过滤器的水龙头。因此，我们需要一个数组（对于每个分支），它存储最后一个 LPF输入样本，并且该数组应该在解调循环之前初始化。

```
c_mixed = np.zeros(numtaps)
for ...
    for ...
        c_mixed[:] = np.append(c_mixed[1:], c_lpf_input)
        c_lpf_output = np.sum(b1*c_mixed)
```

我们将在Costas循环中使用低通FIR滤波器，因此像以前一样在初始化中获取抽头权重，但在排序中进行翻转以考虑它是上面定义的c_mixed中的最后一项，对应

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

图中的Costas循环框图??显示两个LPF输出在混频器处相乘

```
volt = c_lpf_output*s_lpf_output
```

然后通过环路滤波器。由于我们已经使用了具有高滚降率的低通滤波器，因此通常不需要包含更多的频率选择元件，但我们应该确保通过VCO 参数适当地设置增益衰减，因此您可能需要进行实验，直到找到合适的值。

数字数据解调的最后一步是在c_lpf_output中选择一个合适的采样点，然后使用阈值函数将浮点转换为布尔值。我们将再次通过取位的中点来使用最明显的样本点，即i*bit_len+bit_len2。因此，在Nbits上构建for循环并将阈值应用于位中点。在这种情况下，适当的阈值函数是heaviside（step）函数

```
rx_bin[i] = np.heaviside(c_lpf_output[i], 0)
```

最后，将输出二进制数据与输入数据进行比较。你注意到什么问题了吗？
与输入数据相比，您应该观察到输出数据中的延迟。这是因为数字滤波器包括每个抽头内的延迟，以及由信号产生的滤波器设计。firwin通常具有numtaps2的整体延迟。考虑到代码中的这种延迟；在过滤之前，您需要在混合信号数据的末尾添加至少numtaps2个虚拟样本。

您还会发现Costas循环需要许多符号才能锁定输入。
因此，您还需要在数字数据的前面加上一些虚拟符号来复盖这个锁定延迟。您可能希望包含一个唯一的模式，以便可以识别您的数据，例如一些零位后跟一个位。

1.4 Random phase and frequency

Once you have obtained satisfactory locking, then try locking to a carrier with random phase and frequency offset. Ensure the `numpy random` library is loaded.

```
from numpy import random
```

Then modify the carrier phase and frequency (up to 1% deviation) as follows

```
f_c = f_i*(1.+0.02*(random.rand()-0.5))
ph_c = 2*np.pi*random.rand()
```

Run this code a few times and adjust parameters to obtain locking of the costas loop. Examine the demodulated binary data and compare with the original data. You should find that in some instances your data is recovered, and sometimes the inverse.

1.5 Differential Coding

Finally, we shall implement differential coding to deal with the ambiguity in the recovered data.

```
tx_diff = np.zeros(1, dtype='bool')
for i in range(Nbits):
    tx_diff = np.append(tx_diff, tx_diff[i]^tx_bin[i])
Nbits = Nbits+1

... rest of code ...

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

Confirm that this change permits error-free transmission of the original digital data with clock recovery.

Your report should demonstrate, ideally in figures appropriately labelled and captioned:

- data input and output, both for without and with differential encoding
- clock output with reference carrier
- voltage driving the vco
- output data before thresholding

1.6 Documentation

python 3 `https://docs.python.org/3/`

numpy and scipy `https://docs.scipy.org/doc/`

matplotlib `https://matplotlib.org/contents.html`

spyder `https://docs.spyder-ide.org/`

1.4 随机相位和频率

一旦获得了令人满意的锁定，则尝试锁定到具有随机相位和频率偏移的载波。确保numpy随机库已加载。

从numpy导入随机

然后修改载波相位和频率（高达1%的偏差），如下所示

```
f_c = f_i*(1.+0.02*(random.rand()-0.5))
ph_c = 2*np.pi*random.rand()
```

运行此代码几次并调整参数以获得costas循环的锁定。检查解调的二进制数据并与原始数据进行比较。您应该发现，在某些情况下，您的数据被恢复，有时是相反的。

1.5 差分编码

最后，我们将实施差分编码来处理恢复数据中的模糊性。

tx_diff=np。范围内i的零（1，dtype='bool'）
（Nbits）：

```
    tx_diff = np.append(tx_diff, tx_diff[i]^tx_bin[i])
Nbits = Nbits+1
```

…代码的其余部分。。

rx_bin=np。empty(0 dtype='bool')Nbits
=nbits−1foriinrange(Nbits):

```
    rx_bin = np.append(rx_bin, rx_diff[i]^rx_diff[i+1]).astype(bool)
```

确认此更改允许在时钟恢复的情况下无错误地传输原始数字数据。

你的报告应该证明，最好是在适当的标签和标题的数字:

*数据输入和输出，无论是无和差分编码

*带参考载波的时钟输出

*电压驱动vco

*脱粒前输出数据

1.6 Documentation

python 3 `https://docs.python.org/3/`

numpy and scipy `https://docs.scipy.org/doc/`

matplotlib `https://matplotlib.org/contents.html`

spyder `https://docs.spyder-ide.org/`

Getting the python libraries

If you are using your own computer, make sure the Python libraries `scipy`, `numpy` and `matplotlib` are installed. These libraries are installed by default with the Anaconda python distribution. It is recommended that you use a suitable IDE for your project, such as Spyder.

获取python库

如果您使用自己的计算机，请确保安装了Python库scipy，numpy和matplotlib。这些库默认与Anacondapython发行版一起安装。建议您为项目使用合适的IDE，例如Spyder。