The Costas Loop – An Introduction

by Eric Hagemann

My previous columns (check them out in the ChipCenter online archives) examined efficient ways to generate sine waves using table-driven techniques. Let's continue the saga and examine a typical application for these waveforms. One interesting and often-studied application is the phase-locked loop (PLL). Designers use this device to 'lock' a locally generated waveform onto both the phase and frequency of a received waveform. The PLL is crucial to many demodulation algorithms for both carrier and baud synchronization; another interesting use is in tracking Doppler shifts (frequency deviation due to motion of the transmitter or receiver).

You can find many references for PLLs, but they typically focus on hardware implementations of the algorithm. In contrast, very little literature exists describing a software-based approach suited for an all-software demodulator. Thus this column explores a PLL structure suitable for such a software implementation. The structure in question is the Costas loop, named for J. P. Costas, a pioneer in synchronous communications.

The phase-locked loop

Fig 1 depicts the basic structure of a PLL. It mixes a reference waveform with a received waveform to form an error signal. By filtering the error signal it generates a waveform suitable for adjusting the oscillator into 'lock.' When the reference waveform and the

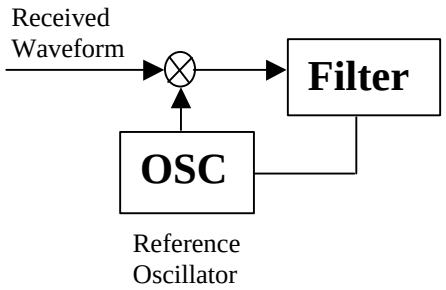

Fig 1 -- In the basic phase-locked loop structure the system relies on feedback formed by mixing a reference signal with a received waveform.

received signal are equal in phase and frequency they are locked in sync.

In hardware implementations the oscillator is generally a VCO (voltage controlled oscillator) operating at a nominal frequency that the system adjusts with the feedback term (a voltage). The filter controls loop operation for parameters such as how fast it takes to acquire lock and the range over which it can acquire lock. This implementation of a PLL is perfect for a hardware-only solution because VCOs are readily available and the filter is simple to construct.

Using software to process a sampled stream opens up some alternatives, namely the Costas loop, whose basic form appears in Fig 2.

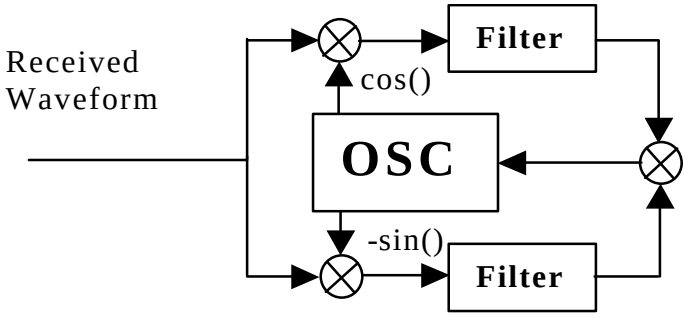

Fig 2 -- A basic Costas loop uses two filters, and matching them is far easier in software

Although the Costas loop looks a bit more intimidating, the implementation is quit simple and the structure is very powerful and useful in many situations. In the software loop, a table-driven sine wave (discussed in previous columns) can function as the oscillator. Using two filters, one on each arm, limits the hardware implementation of this structure because making identical filters is difficult with discrete components. In a sample-based approach, matching the two filters is quite simple.

One structure, many uses

The Costas loop proves handy in many situations. For the moment let's focus on using the loop to track a single tone in noise, although other applications include FM and BPSK demodulation and with slight modification QPSK and QAM demodulation.

Before exploring these uses, let's spotlight the basics of loop operation. Any PLL structure involves feedback. By comparing two waveforms it derives an error value. The

goal over time is to reduce the error term to zero with feedback. As you probably know, feedback is a tricky thing: too much and the structure goes out of control; too little and it remains stable but never locks. (The author plans to leave a detailed discussion of feedback for a later column.)

The Costas loop

Fig 3 shows a breakdown of the loop where the input is a simple sine wave. This design employs an NCO (numerically controlled oscillator) that the designer set to generate a nominal frequency close to the range of the incoming frequency. Now follow the circled numbers and see how this design really works.

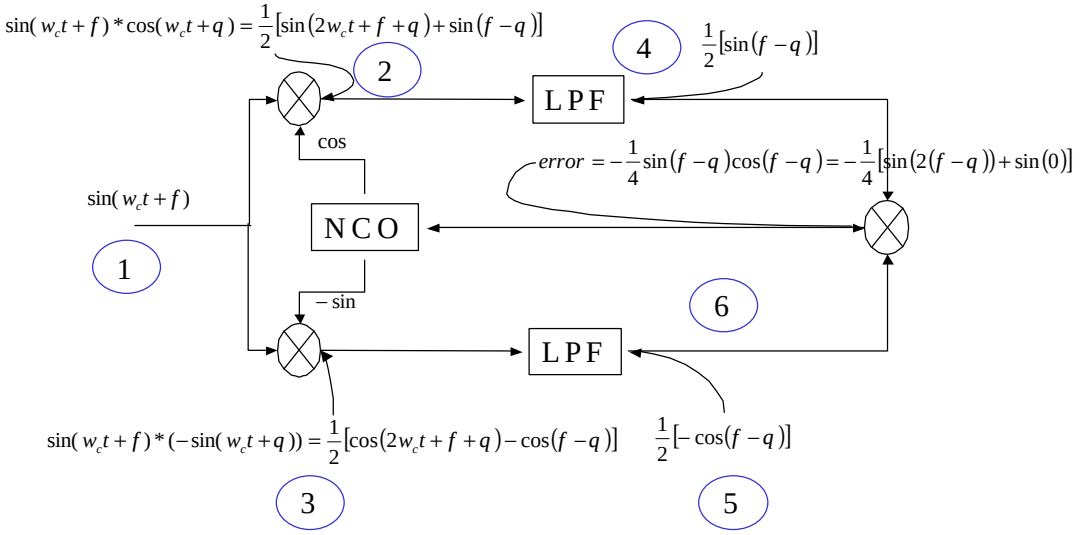

Fig 3 Although a bit complex, the operation of the Costas loop can be demonstrated with some basic trigonometry

Circle 1

We assume the incoming waveform is a tone (sine wave) with an unknown frequency and phase but whose frequency is bounded within a tolerable range. A real-world signal carrying the tone would likely also include some noise, but let's put that issue aside for the moment. To further simply this study, assume the frequency of the reference waveform is matched identically to that of the received waveform.

Circles 2 and 3

At these points the system mixes (multiplies) the incoming waveform independently with a cosine and sine wave it gets from the NCO. Notice the sign change on the sine wave; essentially the system is performing a down-conversion and thus multiplying by e^{-jw}. Recalling some basic trigonometry, note that you're left with two terms: a sum and a difference (considering frequency and phase). Exercise care when dealing with the sum term; the sum term will create a component at double the input frequency. You need to make sure that the sample rate of the system is high enough so that this new tone does not alias back into your operating frequency range (within the bandwidth of the arm filters). The simple fix is to make sure your system sample rate exceeds the input frequency by at least a factor of two.

Circles 4 and 5

Tracking the signal following each lowpass filters (LPF), it's easy to see that only the difference term makes it through. Knowing that no LPF is ideal, a residual of the sum term always exists, but hopefully it's sufficiently attenuated. For a clear understanding of the loop's operation let's assume that the filter has totally eliminated any traces of that residual term. Notice the upper arm has the sine of the difference between the instantaneous angle of the received and the reference waveforms whereas the lower arm contains the cosine of the angle difference.

Circle 6

Here the system mixes the signal from both arms, and we have to delve back into trigonometric identities to find the result. Again you have both a sum term and a difference term. In this case, though, the difference term is zero (sin(0) = 0); thus you have only an error term that is a function of the difference of instantaneous angles.

Employ the error term

The critical feature of this error term is the relationship between the angle difference and the sign of the angle difference. If you hope to bring the loop into lock, the error term must reflect a value that pushes the loop in a direction opposite to the error. When the instantaneous angle of the incoming waveform leads the reference, then the feedback mechanism must retard the reference angle. Did you notice the minus sign? That negation, combined with the asymmetric nature of the sine function about 0, forms a usable feedback waveform. Ideally you would like to work with a linear relationship between angle difference and error. Using approximations valid for small angles, it's

generally accepted that the sine of angles less than 45^0 is close enough to linear for the feedback to work.

Next steps

Now that you have a working model for the loop, what the next step? In order for the loop to lock, you must devote time to understanding feedback and how to accomplish altering the reference waveform in suitable fashion so as to maintain a stable system. Once we get the loop to perform a lock we'll take a look at what happens when the frequency isn't matched and over what frequency ranges you can expect the loop to lock.

Author's biography

Eric Hagemann (ehagemann@home.com) is an electrical engineer who has been programming DSPs for fifteen years. When not writing code, he spends time with his wife and two cats endlessly remodeling their house.