Introduction

Quadrature signals are periodic waveforms that have a phase difference of one-quarter of their period.

A received waveform is decomposed into two signals. There is a signal that is in-phase (and so we call it I) with the received waveform, and one that is in quadrature (we call it Q).

\[
A(t)\sin[2\pi f t + \phi(t)] \quad \text{received waveform}
\]

\[
I(t) \equiv A(t)\cos[\phi(t)] \quad \text{in phase signal}
\]

\[
Q(t) \equiv A(t)\sin[\phi(t)] \quad \text{quadrature signal}
\]

When these signals are implemented in hardware, there are amplitude and phase differences between them. We want to eliminate these errors because they limit demodulation accuracy.

What sorts of errors are there? Phase errors are indicated by \(\phi\) and gain errors are indicated by \(A\).

Going from data generation to data reception,

- I/Q baseband skew error due to the modulator (\(\phi_{\text{MOD}}\))
- Local oscillator quadrature error (\(\phi_{\text{LO}}\))
- Residual DC voltage on received I and Q channels (\(A_{\text{DC}}\))
- I/Q gain imbalance error (\(A_{\text{IMBALANCE}}\))
- I/Q baseband skew due to the demodulator (\(\phi_{\text{DEMOD}}\))

The calibration procedure described here is a gradient descent to the proper phase and amplitude relationship between I and Q.

Consider a tone at frequency \(f\), where \(|f| < F_{\text{SAMPLE RATE}}/2\). After the tone is converted to I and Q signals, there should be no energy visible above the noise floor at \(-f\) if the balance is correct. The frequency constraint on \(f\) ensures that the tone is properly sampled.

If Q channel is not exactly 90 degrees offset from I, and the amplitude of Q channel is not exactly the same as I, you have imbalance, and a tone will be seen at \(-f\).

The average value of \(I^*Q\) should be zero for a tone since I and Q are orthogonal. This product is the basis for the phase offset detector.
Here is a series of diagrams showing what the correction algorithm is doing. The diagrams are in vector space, with I and Q represented in polar form. We start with a vector representation of our received I and Q signals. The received I is the horizontal signal at the bottom of Figure 1.

Received Q is expected to have a gain and a phase error when compared to an ideal Q, which is orthogonal to received I and has no gain mismatch. Transforming received Q to ideal Q is the purpose of this filter. We begin by taking a very small portion of the vector I, and add it to the received Q vector. This moves received Q towards ideal Q. The direction of movement will always be towards ideal Q.

In Figure 3, we see the new Q. This vector now has a reduced phase error. It is closer to ideal Q.

The correction algorithm is structured in a loop. Each time through the loop, there is a phase adjustment followed by a gain adjustment. The loop concludes when the adjusted Q and the received I are orthogonal and have the same magnitude.

The gain adjustment is accomplished by taking the difference between the magnitudes of new Q and I. A tiny bit of this difference in magnitudes is the amount we use to adjust the gain of Q. See Figure 4 for an illustration of this part of the procedure.

This concludes the first pass through the loop. With one phase and one gain adjustment, we have a Q with reduced phase and gain error.

An obvious question might be how we determine the value of “a tiny bit” of I. The amount of I used to adjust Q is calculated by the implementation of the gradient descent algorithm. The algorithm is given in the MATLAB/Octave model...
in the next section. The amount of adjustment during each pass of the loop is important, as it directly affects the algorithm’s stability and speed.

Figure 5 shows the adjusted Q after the first pass through the loop. Figure 6 shows the beginning of our second pass through the loop. Again, we add a little bit of received I to the adjusted Q to further reduce the gain error. We do the loop until the errors have been removed.

In other words, once I and Q have converged, you have balance. Figure 7 shows an equal adjusted Q and ideal Q.

Notice that the phase correction also does a portion of the gain correction. The received Q isn’t simply “tipped up” to correct phase. The received Q is shortened (or lengthened, depending on the particular type of gain mismatch) by projecting the received Q vector
onto the ideal Q vector during the phase correction portion of the loop. The gain correction portion of the loop corrects the remaining portion of the gain error. See Figures 8 and 9.

Below is a block diagram of where the IQ Correction filter fits into a receiver. The IQ Correction is the filter \( F \). Sampling detection is usually implemented with high speed switches. Side effects of switching is a common source of the amplitude and phase imbalance between I and Q.

\( F \) works as long as the DC component has been removed. This can be accomplished with series capacitance. Block diagrams of the IQ Correction filter are shown. The top level diagram below left shows the inputs and outputs of the filter. A high-level finite state diagram of the filter is on the right.
In this procedure, there is a false assumption. We assume that the band has frequency independent imbalance. Time delay errors, such as $\phi_{\text{DGEN}}$, $\phi_{\text{MOD}}$, and $\phi_{\text{DEMOD}}$, result in a frequency-dependent phase error which cannot be completely compensated for by a frequency-independent I/Q phase adjustment. This limits what the calibration procedure can do and provides a path for future improvements.

The Model

Here is the MATLAB model for the correction algorithm. This model is what the VHDL implementation is based upon.

```matlab
close all;
clear all;
e1 = 0.1; %gain error

%select one of the following phase errors or create your own
a1 = -10*pi/180; %provides negative phase error

%provide noise
sgma = 0.01; %noise standard deviation

%original sample length
%n_dat = 250000; % number of samples
n_dat = 50000; % number of samples
freq = 0.03; % provides enough points for pretty figures
amplitude = 1;

%in phase signal, or I
x1=amplitude*cos(2*pi*(0:n_dat-1)*freq)+sgma*randn(1,n_dat);

%quadrature signal, or Q
y1=amplitude*(1+e1)*sin(2*pi*(0:n_dat-1)*freq+a1) + sgma*randn(1,n_dat);

reg_1 = 0;
reg_2 = 1;

%original mu
mu_1 = 0.0002;
mu_2 = 0.0001;

%mu adjusted to closest powers of two in order to use shifts in VHDL.
mu_1 = 0.000244;
mu_2 = 0.000122;
y2=zeros(1,n_dat);
y3=zeros(1,n_dat);

reg_1_sv = zeros(1,n_dat); % used to create phase error estimate graph
reg_2_sv = zeros(1,n_dat); % used to create gain error estimate graph
```
%phase and gain correction loop
for nn=1:n_dat

%adjust phase of Q
y2(nn) = y1(nn)-reg_1*x1(nn);
reg_1_sv(nn) = reg_1;
reg_1 = reg_1 + mu_1*x1(nn)*y2(nn);

%adjust gain of Q
y3(nn) = y2(nn)*reg_2;
reg_2_sv(nn) = reg_2;

%original reg_2
%reg_2 = reg_2+mu_2*(abs(x1(nn))^2 - abs(y3(nn))^2);

%proposed reg_2 - remove absolute value because we’re squaring
reg_2 = reg_2+mu_2*((x1(nn))^2 - (y3(nn))^2);
end
The MATLAB model continues with commands that produce several types of graphs. These graphs show the output of the model. The first set, on the previous page, is Phase Error Estimate Trajectory (in radians) and Gain Error Estimate Trajectory (in proportion of adjustment). The phase error estimate converges at approximately 50,000 samples. The gain error estimate converges at approximately 25,000 samples. Above, Lissajous figures and, below, the spectra of I and Q are shown.
An entity in VHDL is the interface to the outside world. It’s equivalent to the list of pins of an IC that you might want to use in a project.

Here, there are three inputs, the I and Q signals and a clock, and six outputs, the gain error and the phase error between the two signals, the corrected signals, and a gain and phase quality indicator.

The “generic” keyword allows parameters in VHDL to be set. You can see that there’s an input width and an output width. It allows flexibility and reuse in VHDL.

Test Bench Requirements

The test bench is required to provide I and Q samples with amplitude of 1. Noise will be added from a mean zero random normal distribution to both I and Q. The standard deviation of the noise will be 0.01.

The testbench signal Q will have a phase error of 10 degrees and a gain error of 0.1. The testbench will have signals corresponding to all the outputs of the IQ Correction block. The corrected I and corrected Q will be connected to observable signals. Phase and gain lock will be monitored, and phase and gain error estimates will be monitored.

The device under test must converge under these error conditions.
**VHDL Implementation**

Most recent source code can be found at http://opencores.org/project,iqcorrection

---

**Preliminary Results**

Above is a graph of Q data in constellation space. This means that the Q input and Q data output from the VHDL implementation are presented in rectangular coordinates, with the Q data sets plotted against I data output. In this space, phase errors introduce an oval shape, and gain errors affect the radius of the circle.

The adjustments made in the implementation can be seen in the change of the shape and size of the circle formed by graphing the quadrature samples. Comparing this to the Lissajous figures from the MATLAB model, it can be seen that an adjustment is being applied. However, instead of the outer blue (uncorrected data) oval being corrected to a inner red (corrected data) circle, the inner red set of samples is even more squashed (more oval) than the outer blue set of samples.

This error in adjustment is because of an error in the implementation. It was easily caught by graphing the data and comparing with an expected output.
Results

Above is a graph of Q data in constellation space. As in the previous graph from the preliminary results, the Q input and Q data output from the VHDL implementation are plotted against I data output in constellation space. The outer blue (uncorrected data) oval is now adjusted to a inner red (corrected data) circle.

This result confirms that the implementation with real variables works as intended. The next step is to complete the implementation with signed registers.
Synthesizeable Implementation

The first implementation of the IQ Correction filter was completed by using variables. This closely matched the MATLAB implementation, but cannot be easily synthesized. Synthesis is the process where the logic represented by the VHDL is compiled and mapped into an FPGA or ASIC.

In order to synthesize, there are limits on what data types can be used in VHDL. In general, variables are not supported. The calculations for the IQ Correction filter must be done within registers. The values within the registers are expressed as signed data types.

Incorrect Initial Results

During the initial implementation of a registered IQ Correction filter, the output was incorrect. The problem was due to scaling. With the registered implementation, the multiplication of a sample times a ratio means that there was a factor of two error in the gain error estimation. Shifting the selection to the right by one bit solved this particular problem. However, another problem was revealed. The step size for the gain error estimation also needed to be adjusted by a factor of two. Decreasing the number of shifts to the right in order to increase the step size solved this problem.
The hardware used in the modeling and coding of IQ Correct consisted of the following.

Windows and Macintosh personal computers.