Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Low Latency Demodulation for Atomic Force Microscopes, Part II: Efficient Calculation of Magnitude and Phase

Daniel Y. Abramovitch *

* *Molecular Detection Lab, Agilent Laboratories, 5301 Stevens Creek Blvd., M/S: 4U-SB, Santa Clara, CA 95051 USA,* danny@agilent.com

Abstract: This paper describes methods for doing high-speed, low latency, coherent demodulation of signals for dynamic or AC mode in Atomic Force Microscopes (AFMs), Abramovitch (2010). These demodulation methods allow the system to extract signal information in as little as one cycle of the fundamental oscillation frequency. By having so little latency, the demodulator minimizes the time delay in the servo loop for an AC mode AFM. This in turn minimizes the negative phase effects of the demodulation allowing for higher speed scanning. Part I, Abramovitch (2011), of the paper describes the mixing and integration portion of the demodulator. This part describes efficient methods for extracting magnitude and phase. In particular, the CORDIC method compared to optimized table lookup operations and a phase-locked loop (PLL) based method. The latter two show a significant decrease in latency, with the PLL based method generating the magnitude and phase with virtually no extra latency and a significant savings in resources.

Keywords: Atomic Force Microscopes (AFMs), Mechatronic Systems, Demodulators, Real Time Computation, Phase-Locked Loops (PLLs)

1. INTRODUCTION

Part I, Abramovitch (2011), of this paper describes how to construct and efficient integrator for signal demodulation on an AFM. This part will describe two methods for efficiently computing the magnitude and phase of those integrated signals. The first implements a minimal latency table lookup with automatic scaling of signals. The second method involves using a PLL to align the mixing signal with the average phase of the return signal. In this case, the magnitude and phase calculations are trivialized.

From Part I, Abramovitch (2011), of this paper, we know that the process for estimating the integrals of the I and Q branches of the signal with a digital demodulator are:

- Sample $s(t)$ at rate $f_S = \frac{1}{T_S}$, to get $s(k)$.
- Multiply $s(k)$ by $\sin(2\pi f_0 k T_S)$ and $\cos(2\pi f_0 k T_S)$ where f_0 is the frequency to be demodulated.
- Sum the mixed sample signals over an integer number of oscillation periods to approximate the integrals of Equations 3 and 4 with sums, I_{sum} and Q_{sum}.
- Compute the magnitude and phase of the signal.

2. MAGNITUDE AND PHASE CALCULATIONS

Part I, Abramovitch (2011), gives an accurate estimate of the in phase and quadrature integrals of the demodulation, *i.e.,* if

$$s(t) = C_1 \sin(\omega_0 t + \phi_1) + n(t) \tag{1}$$

and

$$I(t) = s(t)\sin(\omega_0 t) \text{ and } Q(t) = s(t)\cos(\omega_0 t), \tag{2}$$

then

$$I_{sum} \approx \frac{1}{MT_0}\int_0^{MT_0} I(t)dt \approx \frac{C_1}{2}\cos(\phi_1) \text{ and} \tag{3}$$

$$Q_{sum} \approx \frac{1}{MT_0}\int_0^{MT_0} Q(t)dt \approx \frac{C_1}{2}\sin(\phi_1). \tag{4}$$

The classic method of computing magnitude and phase is from a coordinate transformation from rectangular to polar coordinates , *i.e.,*

$$C_1 = 2\sqrt{I_{sum}^2 + Q_{sum}^2} \text{ and } \phi_1 = \text{Arctan}\left(\frac{Q_{sum}}{I_{sum}}\right). \tag{5}$$

The difficulty comes in the resources needed to compute these relationships. For example, a highly efficient algorithm is the so called CORDIC algorithms, Volter (1959); Meher et al. (2009); Vadlaman and Mahmoud (2002). These algorithms compute magnitude and phase by rotating the frame of reference until the frame of reference has a matching magnitude and phase. The CORDIC algorithm is computationally simple, and is at the heart of the trigonometric calculations in the original HP-35 calculator. It is even available now in logic cores for FPGAs, Xilinx (2009). However, they require one computational cycle per bit of accuracy, so a 16 bit accuracy would require an extra computational delay (on top of that done by the integral itself) of 16 clock cycles. In a standard computer, this might be considered fast, but in a DSP or FPGA which typically complete table lookup operations, additions, and multiplies in one or two cycles, this is considered slow. Alternately, some have chosen to offload the magnitude and phase calculation to a DSP chip once the I and Q branch demodulations are done, Proksch et al. (2007).

A more time efficient calculation involves a table lookup. However with two different numbers to look up and the high precision desired in these calculations, the tables can become huge. A few well placed adjustments to and restrictions of the calculation make it possible to use a relatively small table to give reasonably good estimates. The process for doing this involves two steps:

- Scaling the numbers so as to restrict the input range of the table, and then unscaling them after the table has been used.
- Interpolating between points in the lookup table using extra bits from the calculation.

3. CALCULATING MAGNITUDE USING TABLE LOOKUP

It is certainly possible to do a simple table lookup to compute the magnitude, C_1, from the left side of Equation 5. The question becomes how to do this to efficiently make use of memory. Looking at Equation 5 we note:

- Sums of squares of a number are easy to compute using most computer hardware, including DSPs and FPGAs which have built in hardware multiplies.
- Square roots are hard to compute quickly.
- The slope of the $\sqrt{x^2}$ changes very rapidly near $x^2 = 0$, which makes interpolation less accurate.
- Limiting the input range of the $\sqrt{x^2}$ lookup table to the range from 0.5 to 2.0, makes $\sqrt{x^2}$ well behaved.
- Since $I_{sum}^2 + Q_{sum}^2$ is in 2's compliment notation, then it will have one or more leading 0*s*.

We can minimize the entries in the table lookup by shifting to the left until the leading two bits are 01, 10, or 11, and keeping track of the left shifts. A left shift by $2n$ bits effectively multiplies the number by 2^{2n}. This pins the number in the table to be between 0.5 and 2, i.e. if

$$x_{in}^2 2^{2n} = \left(I_{sum}^2 + Q_{sum}^2\right) 2^{2n}, \text{ then} \tag{6}$$

$$0.5 \le x_{in}^2 2^{2n} < 2. \tag{7}$$

Once the square root of the shifted sum of squares is looked up, we shift the result to the right by n bits, since

$$y = \sqrt{\frac{x_{in}^2 2^{2n}}{2^{2n}}} \text{ then } y = \frac{\sqrt{x_{in}^2 2^{2n}}}{2^n}. \tag{8}$$

With a memory that has 2^M locations, improved accuracy can be achieved by splitting the address space and using part of the values for interpolation. For example, with 2^{10} locations, a lookup table of square roots of input values between 0 and 2 has an accuracy of only 4 bits. Using 2^9 locations for lookup points and 2^9 locations for slopes to do linear interpolation (for a total of 2^{10} locations) leads to an accuracy of 6 bits the same number of memory locations. Restricting the input range to being between between 0.5 and 2 for a lookup table with 2^{10} locations gives an accuracy of 9 bits. Using 2^9 locations for lookup points and 2^9 locations for slopes to do linear interpolation (for a total of 2^{10} locations) gives an accuracy of 19 bits as seen in Figure 1.

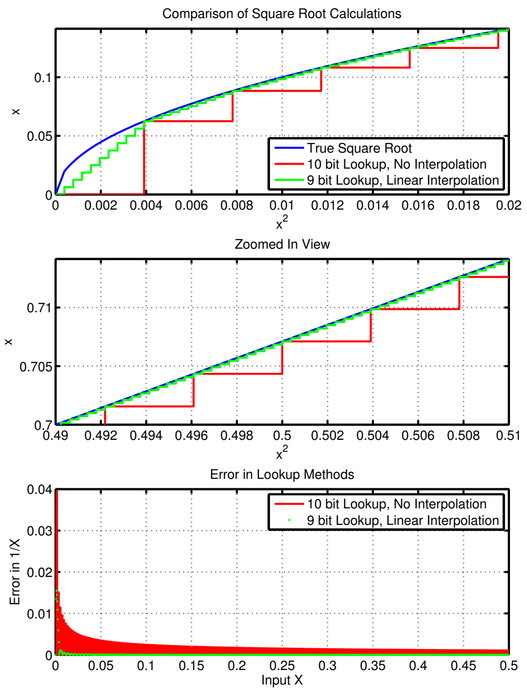

Fig. 1. Comparison of table lookup and errors for $\sqrt{x}$, $0.0 \le x \le 2$. Plots are zoomed in to show more interesting aspects of the data.

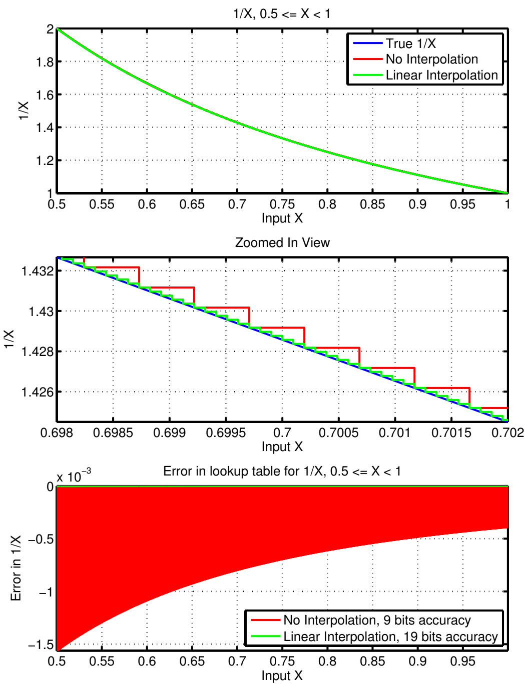

Fig. 2. Comparison of table lookup and errors for $\frac{1}{x}$, $.5 \le x \le 2$.

Making a relatively mild assumption that a single computation can be done in a single clock cycle of the FPGA or DSP, the procedure and the estimated latency are:

1) Square I_{sum} and Q_{sum} (1 clock cycle).
2) Add them together yielding x_{in}^2 (1 clock cycle).

3) Shift left by the maximum even number of leading 0s, $(2n)$ to yield $x_{in,2n}^2$ where $0.5 \leq x_{in,2n}^2 < 2$ (1 clock cycle).
4) Look up the square root in a table (1 clock cycle).
5) (Optional) Look up interploation slope in table (1 clock cycle, in parallel with other lookup).
6) (Optional) Interpolate square root value between lookup points (2 clock cycles, 1 for multiply and 1 for addition).
7) Shift square root value right by half as many bits as the previous left shift (n) (1 clock cycle).

So, this table lookup computes the square root in 7 clock cycles, irrespective of the number of bits in the input. For data widths of greater than 7 bits, this is faster than the CORDIC algorithm.

4. CALCULATING PHASE USING TABLE LOOKUP

We also need to compute the arctangent of $\frac{Q_{sum}}{I_{sum}}$ digitally. Again, the CORDIC may be too slow for our purposes and we want to use a table lookup. We can limit the operation to the first quadrant ($0 \leq \frac{Q_{sum}}{I_{sum}} < \frac{\pi}{2}$) by keeping track of the signs of the input values and then simply working with the absolute values. We then shift the result back into the proper quadrant by some simple math.

The problem is that it is hard to be accurate in a table for the Arctan$\frac{Q_{sum}}{I_{sum}}$ when I_{sum} is close to 0. However, we can easily look up

$$\text{Arccot}\left(\frac{Q_{sum}}{I_{sum}}\right) = \frac{\pi}{2} - \text{Arctan}\left(\frac{I_{sum}}{Q_{sum}}\right), \tag{9}$$

and so we operate in the first half of the first quadrant.

- If $|I_{sum}| = |Q_{sum}|$, then Arctan(1) $= \frac{\pi}{2}$.
- If $|I_{sum}| > |Q_{sum}|$, then lookup Arctan $\left(\frac{Q_{sum}}{I_{sum}}\right)$.
- If $|I_{sum}| < |Q_{sum}|$, then lookup Arctan $\left(\frac{I_{sum}}{Q_{sum}}\right)$ and compute Arccot $\left(\frac{Q_{sum}}{I_{sum}}\right) = \frac{\pi}{2} -$ Arctan $\left(\frac{I_{sum}}{Q_{sum}}\right)$.

Now, we see that before we can look up an Arctan, we need to compute either $\frac{|Q_{sum}|}{|I_{sum}|}$ or $\frac{|I_{sum}|}{|Q_{sum}|}$, depending upon whether $|Q_{sum}|$ or $|I_{sum}|$ is larger. This means looking up $\frac{1}{|Q_{sum}|}$ or $\frac{1}{|I_{sum}|}$.

Say we want to compute the value $\frac{Y}{X}$ where X is the larger of $|Q_{sum}|$ and $|I_{sum}|$. We have to look up $\frac{1}{X}$ in a table and multiply this by Y. $\frac{1}{X}$ is badly behaved when X is close to 0, but we can shift both numerator and denominator:

$$\frac{Y}{X} = \frac{2^n Y}{2^n X}, \tag{10}$$

so that the leading 0s in X have been eliminated. We have already assumed that $X > Y$ so in a 2's compliment format Y should have at least as many leading 0s as X. This means:

a) The value of $2^n X$ in the lookup table for $\frac{1}{2^n X}$ is always between 1 and 2 so The looked up value is always between $\frac{1}{2}$ and 1.
b) $\left(\frac{1}{X}\right) Y$ is always between 0 and 1.
c) Therefore the looked up and interpolated values should be quite accurate as seen in Figure 2. Without interpolation, the 2^{10} location table achieves 9 bits of accuracy. Splitting the table into two 2^9 location tables where the first part is for lookup and the second part is for linear interpolation results in 19 bits of accuracy.

For simplicity, we can look up values between 0 and 1 and scale them by $\frac{\pi}{4}$ in post processing. So, our procedure is as follows:

1) Compute $|I_{sum}|$, $|Q_{sum}|$, sgn(I_{sum}), and sgn(Q_{sum}) (1 cycle).
2) Determine if $|I_{sum}|$ or $|Q_{sum}|$ is larger. If $|I_{sum}| = |Q_{sum}|$, the Arctan is 1 (1 cycle).
3) Shift both $|I_{sum}|$ and $|Q_{sum}|$ left to eliminate leading zeros in larger value (1 cycle).
4) Look up $\frac{1}{X}$, where X is the larger of $|I_{sum}|$ and $|Q_{sum}|$ (1 cycle).
5) (Optional) Look up interpolation slopes for $\frac{1}{X}$ (1 cycle).
6) (Optional) Interpolate values between lookup points of $\frac{1}{X}$ (2 cycles, 1 for mutliply and 1 for addition).
7) Multiply $\frac{1}{X}$ by Y, where is the smaller of $|I_{sum}|$ and $|Q_{sum}|$ (1 cycle).
8) Look up Arctan$\frac{Y}{X}$ where $0 \leq \frac{Y}{X} < 1$ (1 cycle).
9) (Optional) Look up interpolation slopes for Arctan$\frac{Y}{X}$ (1 cycle).
10) (Optional) Interpolate values between lookup points of Arctan$\frac{Y}{X}$ (2 cycles, 1 for multiply and 1 for addition).
11) Calculate Arccot$\frac{X}{Y} = \frac{\pi}{2} -$ Arctan$\frac{Y}{X}$ if needed (1 cycle).
12) Use sgn(I_{sum}) and sgn(Q_{sum}) to put the Arctan or Arccot in the proper quadrant (1 cycle).

This computation takes 14 cycles, which is more than the original integration from Part I, Abramovitch (2011). However, it seems to be the fastest method of directly computing the phase for any numbers of bits greater than 14, unless one trivializes the magnitude and phase operation. The next section shows how to do just that.

5. USING A PLL TO SIMPLIFY MAGNITUDE AND PHASE CALCULATIONS

The previous sections on computing magnitude and phase point out the issue that even the fastest methods of computation generate a significant amount of serial steps in the process. Assuming that each step can be done in one computation clock cycle (not always the case with FPGAs, and even more rare with DSPs), we still have quite a few cycles. For fast sampling, this can be a large fraction of a sample period. Furthermore, these calculations take significant resources, either in time – if the computation is done on a standard processor or a DSP – or in space if the computation is done in programmable logic – such as with a FPGA.

There is a way to circumvent these extra computations, and this is by noticing that if the mixing signal is in phase with the signal to be demodulated, then the magnitude drops out trivially from the integral. In other words, if the I mixing signal is in phase with the signal to be demodulated, then the Q mixing signal is 90° out of phase meaning that on average, Q_{sum} is very close to 0. This means that the left side of Equation 5 is reduced to

$$C_1 \approx 2\sqrt{I_{sum}^2} = 2I_{sum}. \tag{11}$$

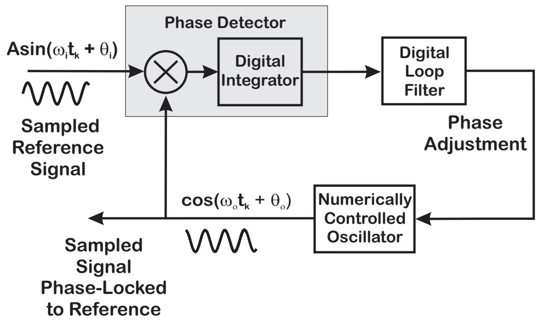

Fig. 3. A digital mixing PLL including post mixing integration. Note that our demodulator mixes the input signal with numerically generated sinusoids and integrates them over an integer number of periods, Abramovitch (2011). The end result is an output that has very little of the 2X frequency component in the classical mixing loop. In other words, the Q branch of our demodulator calculates the phase error between the input signal and the sinusoid.

Furthermore, in place of the right side of Equation 5, we know that the average phase difference between the return signal and our driving signal is equal to the phase difference between our I mixing signal and our driving signal, and this can be read off trivially. The integral of Q_{sum} has an elegant interpretation as the instantaneous phase difference between the return signal and the average phase. The difference between the phase of the mixing NCO and the drive NCO represents the average phase.

However, since the average phase of the return signal is unknown, we need some way to identify it. The classic way is with a phase-locked loop (PLL). A block diagram of a digital PLL is shown in Figure 3. Each PLL has a reference signal and an oscillatory signal which will lock to it, Abramovitch (2002). The oscillatory signal is presumed to be at a frequency close enough to the reference signal so that differences between the frequencies can be considered phase errors. A phase detector extracts the baseband phase difference between the two signals as well as some higher frequency information to be filtered out. The properties of the loop, set by the combination of the phase detector and the loop filter, determine the phase changes that can be followed and which changes will be treated as disturbances to reject.

Looking at the details of Figure 3, the input signal is sampled, just as our return signal from the AFM is sampled. A sampled oscillatory signal is multiplied (mixed) with this sampled signal and the output passes through the loop filter to adjust the phase of the numerically controlled oscillator (NCO), which generates the mixing signal. It turns out that this is very similar to the form that we already have in the demodulator.

Our demodulator mixes the input signal with numerically generated sinusoids and integrates them over an integer number of periods. The end result is an output that has very little of the 2X frequency component in the classical mixing loop. In other words, the Q branch of our demodulator calculates the phase error between the input signal and the sinusoid. All we need to add to the pieces in our demodulator are the ability to adjust the phase of the sine drive (our NCO) and a loop filter to regulate the bandwidth of this loop adjustment.

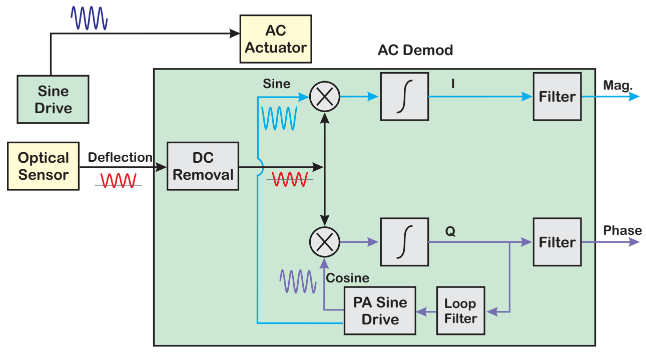

Fig. 4. Coherent demodulation for AFM using a PLL. DC removal and post integration filtering included.

A block diagram that shows the PLL as a part of the overall demodulator is shown in Figure 4. We now see that in the magnitude and phase computation blocks have been eliminated by this method. From a delay point of view, this means that the delay through the demodulator is governed only by the integration portion. Furthermore, the adjustment of the mixer's phase by the PLL is done as a sort of background process, outside the time critical flow. Finally, the resources occupied by this method, whether they be CPU time or space on a FPGA are far smaller than the previous method.

Finally, it is often the case with AFMs that the equations are written so that the driving sinusoid is considered a cosine rather than a sine. This doesn't change the behavior, but does change the style of the mathematical analysis.

6. PRE AND POST INTEGRATION FILTERING

If the integrals of Equations 3 and 4 were done in continuous time with infinite precision, then there would be need to filter DC components or harmonics of the input frequency. However, the sampling of the data means that the sinusoids are approximated by a stair step function, and the integration is approximated as described in Part I, Abramovitch (2011), of this paper. This means the rejection of DC offsets and higher harmonics is not complete. Several simple fixes, including DC removal and post integration filtering, improve the behavior of the integration .

The DC removal can be accomplished using a high pass filter, and there are several different methods available for implementing this. For a discrete time ideal high pass filter, we would have

$$H(z) = \frac{\alpha(z-1)}{z-\alpha} = 1 - \frac{(1-\alpha)z}{z-\alpha} = 1 - L(z). \tag{12}$$

In other words, a high pass filter can be implemented by subtracting a low pass value from the original signal. While these are theoretically equivalent, there are some latency advantages to the second method in that the low pass value can be computed in the background (it changes slowly) and only the subtraction is in the "latency" path.

Post integration filtering helps remove artifacts of the integral approximation from the computed magnitude and phase. In particular, harmonics of the original drive frequency often show up, although at a greatly reduced level. A notch filter, set at the frequency of the chosen harmonic can be used. Alternately, an FIR filter similar to the demodulator integrator is used. The main difference is that in the FIR, there is that there is no mixing of the input signals. The output of the demodulator is fed into another integrator, which integrates over an integer number of periods of the original wave. The effect is to null all higher harmonics of the signal, but the FIR lengthens the delay associated with the demodulator by at least half the period of the original oscillation (depending upon the length of the filter and the period of oscillation). A notch filter implemented as a digital biquad only removes a single harmonic at a time, but the added latency is fixed and minimal (typically a few clock periods). In combination with the DC removal, the notching of the $2f_0$ harmonic closely approximates the performance of the FIR.

7. EXAMPLES

The demodulator architecture, implemented in FPGA hardware, was simulated using ModelSim 6.6b Mentor Graphics (2011), and signals of interest to this paper were saved to an ASCII file. The ASCII data was normalized using Matlab so that the number format was back in a more convenient form. Also, small gain differences between the I and Q phase outputs before and after filtering have been normalized out. The last 20% of the data was used to compute the steady state averages (μ) and standard deviations (σ) of these signals. In these simulations, the units of Q map to radians, since Q is used as the phase error.

Figure 5 shows an 88 kHz oscillation frequency, sampled at 1 MHz. The effect of the PLL based demodulator is that the mixing signals are shifted so that the in-phase (I) signal aligns with itself with the average phase of the input signal. The quadrature signal (Q) is aligns itself so that it is 90° out of phase with the average of the input signal. Thus, the magnitude calculation is obtained trivially from the in phase integral. This simulation has no offset in the level of the deflection (input) signal, but a +30° phase offset. Figure 7 show a similar simulation but with an offset of 0.1 and a −179° phase offset. Note how the PLL lock signal corresponds to the mixing signals being close enough to ideal in-phase and quadrature such that I and Q equal the magnitude and phase.

Figures 6 and 8 show the effects of using post integration filtering on the demodulated signals. In particular, Figure 6 which has particularly small phase error throughout, is scaled such that one can really see the improvement of the filtered signals.

The top plots of both figures show the I branch, while the lower plots show the Q branch. The unfiltered output of the demodulator is shown in blue. That output, post

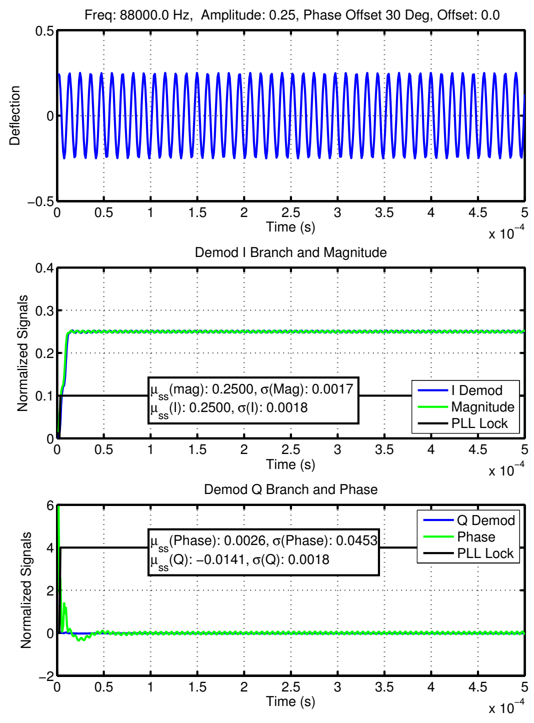

Fig. 5. Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is 0.25. There is no offset in the signal level, but the phase of the signal driving the deflection is 30° ahead of the in-phase (sine) mixing signal at the beginning of the simulation. Note how the I and Q phases converge to the magnitude and instantaneous phase, respectively.

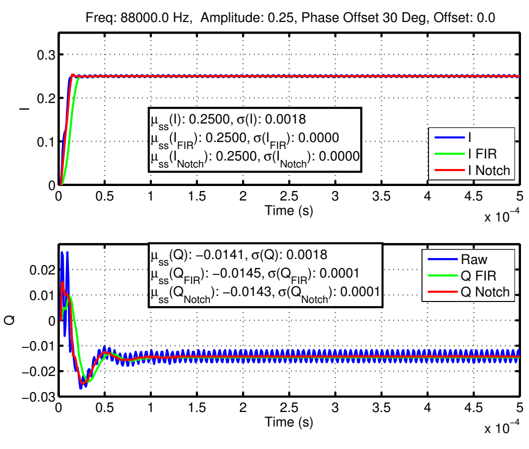

Fig. 6. Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is 0.25. There is no offset in the signal level, but the phase of the signal driving the deflection is 30° ahead of the in-phase (sine) mixing signal at the beginning of the simulation. This plot shows the effect of adding post post integration filtering to the demodulator.

processed with an FIR filter which removes all the harmonics of the oscillation frequency, are shown in green. Looking at the Q outputs which are zoomed in due to the small size of output signals, we can see that the unfiltered demodulator outputs have a component at twice

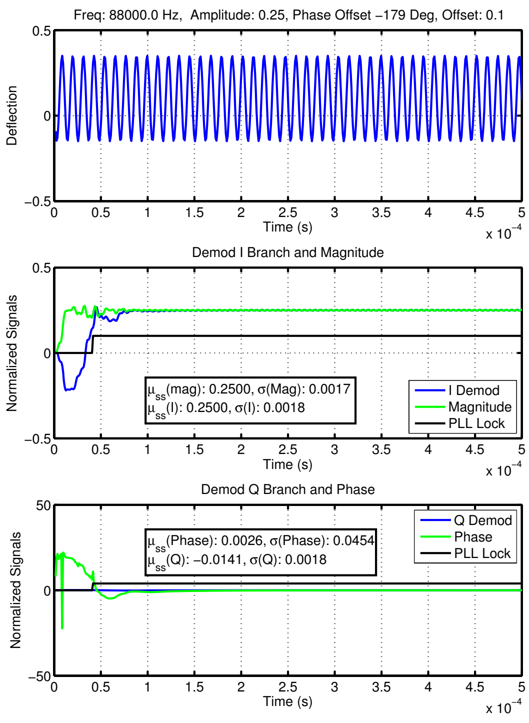

Fig. 7. Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is 0.25. There is a normalized offset of 0.1 in the signal level, and the phase of the signal driving the deflection is 179° behind that of the in-phase (sine) mixing signal at the beginning of the simulation. Again, the I and Q phases converge to the magnitude and instantaneous phase, respectively.

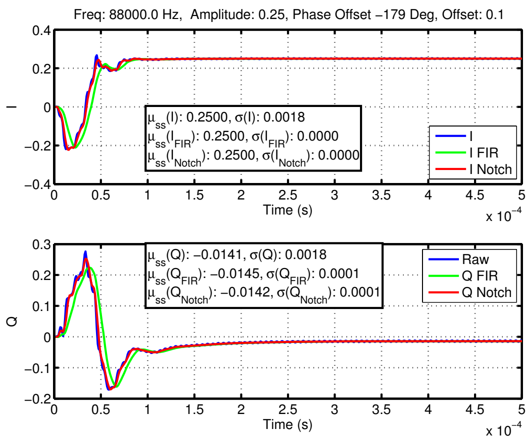

Fig. 8. Output of ModelSim Simulation of FPGA based demodulator. The oscillation frequency is 88 kHz. The normalized deflection amplitude is 0.25. There is a normalized offset of 0.1 in the signal level, and the phase of the signal driving the deflection is 179° behind that of the in-phase (sine) mixing signal at the beginning of the simulation.

the oscillation frequency, f_0. Thus, a simple notch filter, shown in red, can be used to remove this component from the output. The choice between the FIR and the notch depends upon the number of harmonics that one is concerned with versus the additional computational latency that one is willing to accept. However, in this case, the notch at $2f_0$ performs indistinguishably from the FIR, and the latency is clearly less than the FIR, although slightly more than the unfiltered results.

One more signal of interest is the PLL locked indicator. This is applied in the demodulator when the integral of the I branch stays positive and significantly larger than the absolute value of the Q branch integral. Note that once this signal becomes positive, the I branch integral is very close to the input signal magnitude.

8. CONCLUSIONS

This paper has described a very low latency demodulator for use in dynamic mode control of Atomic Force Microscopes (AFMs). Part I, Abramovitch (2011), described the mixing and integration portion of the demodulator. This part described efficient methods for extracting magnitude and phase. Together, they perform operations that extract the magnitude and phase of return signals with high fidelity, but minimal latency. This, in turn, solves one of the more difficult problems in speeding up dynamic mode operation of AFMs.

REFERENCES

Abramovitch, D.Y. (2002). Phase-locked loops: A control centric tutorial. In *Proceedings of the 2002 American Control Conference*. AACC, IEEE, Anchorage, AK.

Abramovitch, D.Y. (2011). Low latency demodulation for atomic force microscopes, Part I: Efficient real-time integration. In *Proceedings of the 2011 American Control Conference*. AACC, IEEE, San Francisco, CA.

Abramovitch, D.Y. (2010). Coherent demodulation with reduced latency adapted for use in scanning probe microscopes. United States Patent 7,843,627, Agilent Technologies, Santa Clara, CA USA.

Meher, P.K., Valls, J., Juang, T.B., Sridharan, K., and Maharatna, K. (2009). 50 years of CORDIC: Algorithms, architectures, and applications. *IEEE Transactions on Circuits and Systems – I:Regular Papers*, 56(9), 1893–1907.

Mentor Graphics (2011). *ModelSim*. Mentor Graphics, www.mentor.com/products/fpga/simulation/modelsim.

Proksch, R., Cleveland, J., Bocek, D., Day, T., Viani, M., and Callahan, C. (2007). Fully digital controller for cantilever-based measurements. United States Patent 7,234,342, Asylum Research Corporation, Santa Barbara, CA USA.

Vadlaman, S. and Mahmoud, W. (2002). Comparison of CORDIC algorithm implementations on FPGA families. In *Proceedings of the Thirty-Fourth Southeastern Symposium on System Theory, 2002.*, 192–196.

Volter, J.E. (1959). The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computation*, 8, 330–334.

Xilinx (2009). *CORDIC v4.0*. Xilinx Corporation, www.xilinx.com.