A Digital Phase-Locked Loop
For Hard Disk Drive
Read Channel Applications

by

John Leonard Wallberg

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science and Master of Engineering

in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 23rd, 1997

Author _____
Department of Electrical Engineering and Computer Science
May 23rd, 1997

Certified by _____
Hae-Seung Lee
Thesis Supervisor

Accepted by _____
A. C. Smith
Chairman, Department Committee on Graduate Theses

A Digital Phase-Locked Loop
For Hard Disk Drive
Read Channel Applications

by

John Leonard Wallberg

Submitted to the
Department of Electrical Engineering and Computer Science

May 23rd, 1997

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science and Master of Engineering
in Electrical Engineering and Computer Science

ABSTRACT

Phase-locked loops are at the heart of read channels, in both the synthesizing and synchronizing capacities. This thesis presents an all-digital phase-locked loop with a unique acquisition algorithm that employs a frequency estimator. This estimator significantly reduces lock times and provides for good noise rejection.

Thesis Supervisor: Hae-Seung Lee
Title: Professor, M.I.T. Department of Electrical Engineering

Acknowledgments

The first thanks must go to the Good Lord above. Many times in my five years here there has been only one set of footprints in the sand, and the debt I owe God is second only to his love.

To my parents, thank you for you love, support, and prayers throughout my life. It's hard to believe that my time here is finished, but don't get me wrong, I'm not complaining. I hope you realize how blessed I have been to have parents like you.

To all my friends, thanks for your friendship through the years. You were there when I needed someone to talk to, and you also provided the appropriate distractions when they were called for. All work and no play... To my pledge class, JMK, MDS, PLG, JMN, TDR, EDG, VBB, DBN, CGN, DTR, MJN, MML, JCT, KJP, APH, MHJ, Love and Respect. Ru Rah Rega!

This work for this thesis was done at Texas Instruments in Dallas. I would like to thank my supervisor Ben Sheahan, Shawn Fahrenbruch, and the rest of the read channel group for all of your help. Thanks for answering my questions and offering advise on this thesis. I look forward to rejoining you after graduation.

And finally, thank you Professor Lee, my thesis advisor, for your patience during all of this. This hasn't been the easiest of tasks for me to complete, and your understanding has been greatly appreciated.

Contents

1 Introduction **8**

2 Background **10**

2.1 The Hard Disk 11

2.2 Phase-Locked Loops 11

2.3 Timing Loops 13

2.4 Jitter 14

2.5 Data 15

2.6 Zero Phase Restart 16

3 The Algorithm **18**

3.1 Loop Dynamics 19

3.2 The Algorithm 22

3.3 Phase Optimal Frequency Estimation 25

4 PLL Structure **29**

4.1 Voltage Controlled Oscillator 30

4.2 Phase Detector 31

4.3 Coarse Tuning 32

4.4 Fine Tuning 32

5 Simulations **34**

5.1 Chatter 35

5.2 A Sampled System 36

5.3 Jitter on the Reference Clock 38

5.4 Jitter on the VCO 39

5.5 Wander on the VCO 40

5.6 Tones on the Reference Clock 41

5.7 Tones on the VCO 47

6 Conclusions **50**

A Digital Phase-Locked Loop Code **52**

Bibliography **78**

List of Figures

2.1 Hard Disk Overview 11
2.2 Traditional Phase-Locked Loop 12
3.1 Steady-State Error Loop 19
3.2 The root-locus plot of L(s) 21
3.3 Scheme for aligning the VCO and reference clock 23
3.4 Loop response of acquisition algorithm 25
3.5 Loop response of acquisition algorithm with estimator 28
4.1 Fine tuning portion of acquisition algorithm 33
5.1 Jitter vs. LSB 35
5.2 Jitter vs. M Divider 36
5.3 Jitter vs. Reference Clock 37
5.4 Jitter added to the reference clock 38
5.5 Jitter added to the VCO 39
5.6 Wander added to the VCO 41
5.7 Digital phase-locked loop 42
5.8 0.4 MHz Tone on reference clock without estimator 43
5.9 0.4 MHz Tone on reference clock with estimator 43
5.10 0.8 MHz Tone on reference clock without estimator 44
5.11 0.8 MHz Tone on reference clock with estimator 44

5.12 1.6 MHz Tone on reference clock without estimator 45
5.13 1.6 MHz Tone on reference clock with estimator 45
5.14 3.2 MHz Tone on reference clock without estimator 46
5.15 3.2 MHz Tone on reference clock with estimator 46
5.16 0.2 MHz Tone on the VCO without estimator 48
5.17 0.2 MHz Tone on the VCO with estimator 48
5.18 0.4 MHz Tone on the VCO without estimator 49
5.19 0.4 MHz Tone on the VCO with estimator 49
6.1 Locking times with and without the frequency estimator 51

Chapter 1

Introduction

Hard disk drives (HDD) allow users to store data for extended periods of time. As the world around us continually demands faster speeds for life in general, such is the case for the world of hard disk drives. Read channels, which are the physical communication between the analog hard disk and the digital computer, play an essential role in determining the overall speed of data transfer. The demand for faster channels has caused circuit designers to push current process technology to its limits. Every produced chip can expect a production life of only six months before the next generation replaces it.

For many years read channels have been implemented in a BiCMOS technology. This was mainly due to the speed requirements in the channel that could not be met using MOS technology alone. Over the years, due to the wide variety of uses for CMOS, the demand for increases in CMOS technology has catapulted it nearly equal to bipolar technology with respects to speed. Furthermore, where CMOS has really become

attractive is in its ability to change processes with relative ease. This factor, along with the cheaper nature of CMOS processes, has led to a demand for all CMOS designs.

This sums up the motivation for an all-digital phase-locked loop in CMOS technology in the read channel. CMOS is cheaper to process, and a CMOS phase-locked loop that used more traditional analog solutions would not fully utilize the benefits of CMOS, which are its ability to change processes with relative ease.

This thesis presents an acquisition algorithm that a digital phase-locked loop can use in place of a loop filter. This algorithm establishes a quick and efficient methodology that guarantees phase lock. Chapter two provides some basic background information about hard disks and the operation of read channels. Chapter three covers loop dynamics and explains the acquisition algorithm. The heart of this thesis, which has been dubbed phase optimal frequency estimation, is developed in the third section of chapter three. Chapter four covers the physical implementation of the loop for simulation purposes; note that a detailed gate level design is not covered for all parts of the loop. Chapter five covers the loop simulations performed, while chapter six draws conclusions based on the results of chapter five. An appendix of the code used to simulate this digital phase-locked loop is included.

Chapter 2

Background

This section covers some of the basics concepts involved in hard disk drives. The first section discusses the terminology associated with divisions of the hard disk, such as zones, wedges, tracks, and sectors. The next section gives the fundamentals of phase-locked loops. Timing loops internal to the read channel, such as read loop, write loop, and servo, follow in section three. Section four examines jitter, which will be extremely useful in chapter five. A quick look at the nature of data, how it is written to and read back from a disk, is contained in section five. The last section discusses the concept of zero-phase restarts, as well as what it means in this thesis.

2.1 The Hard Disk

The physical hard disk is divided into tracks, sectors, wedges, and zones. See Figure 2.1. Tracks run the circumference of the disk, and the normal density is between 5000 and 7000 tracks per inch (TPI) of the diameter. Sectors are data arcs one track wide that usually contain 512 bytes of user data. Wedges are "pie slices" that contain servo information (radial location) that cannot and must not be overwritten by the user, normally numbering between 50 to 100 per track. Traditionally, all servo information is written at the same frequency, regardless of the radial location on the disk. Zones are an adjacent collection of tracks in which all data is written at the same frequency, approximately 12 per disk.

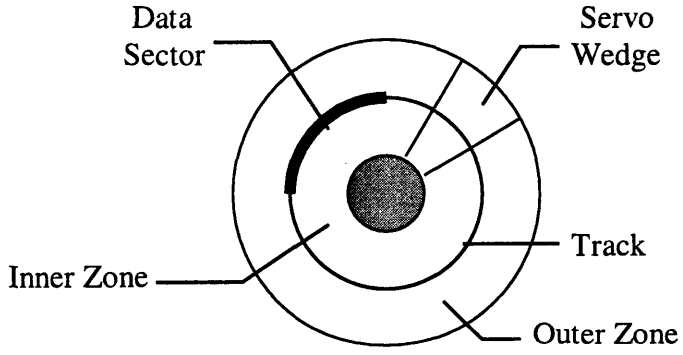

Figure 2.1. Hard Disk Overview

2.2 Phase-Locked Loops

Phase-Locked Loops (PLLs) consist of three main blocks: a phase detector, a loop filter, and a voltage-controlled oscillator (VCO). See Figure 2.2. The phase detector receives

two clocks, the oscillator clock and the reference clock, and outputs a proportional error term. The loop filter's main effect is to influence the loop dynamics. The oscillator is generally a multi-vibrator or ring oscillator, and is controlled in numerous ways, including voltage control, current control, current starving, and capacitive loading.

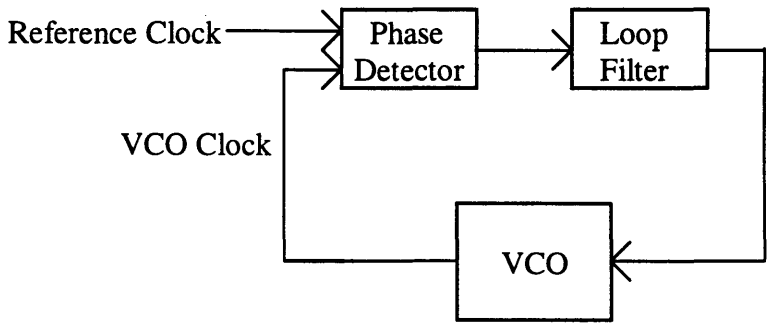

Figure 2.2. Traditional Phase-Locked Loop

PLLs are generally divided into three categories: linear, classical-digital, and all-digital [1]. A Linear PLL uses a four-quadrant multiplier as a phase detector and low-pass filters the result to feed into the VCO. A Classical-Digital PLL uses digital components for the phase detector, and an analog loop filter. An All-Digital PLL consists totally of digital components (logical devices). All-Digital PLLs will be addressed as "digital" PLLs in this paper, with the desire and intent to eliminate the capacitors and resistors of the loop filter, although an internal analog control signal, such as a current or voltage digital-to-analog converter (DAC), may exist.

2.3 Timing Loops

Three main modes exist in Hard Disk Drive read channel operation: read, write, and servo. The read and write mode involve reading and writing user data from and to the disk's data sectors. Servo is a special mode in which information regarding the disk's track number and the head's position is read off the disk and passed to an external controller for correction. Both the read and write modes require timing loops in order to establish an internal clock for the read channel. The servo mode may or may not need a timing loop, depending upon its implementation. In traditional read channel design, these timing loops are primarily implemented by classical-digital PLLs.

The write loop, also known as the synthesizing loop, needs to be able to produce several different frequencies at which data can be written, all in reference to a crystal oscillator of known frequency. Multiple frequencies are required because the disk spins at a constant number of revolutions per second, and in order to maximize capacity, data at the outer diameter (zone) should be written faster than data near the inner diameter (zone). By placing frequency dividers after the VCO (N divider) and reference clock (M divider), ratios (N/M) of the reference clock can be established, therefore creating the multiple frequencies required. It is essential that all frequency and phase transients have settled out before any information is written to the disk. Any transients written on the disk will have to be tracked out by the read loop. This will increase the bit error rate, and if the transients are severe enough, may prevent lock completely.

The read loop, also known as the synchronizing loop, must lock itself to the data stream itself. This loop will already have the approximately correct zone frequency because it was tracking the write loop, but must adjust the phase to acquire phase-lock.

The frequency might also need to be adjusted to account for small changes in frequency from the time of writing to reading due to numerous effects such as temperature, motor speed variation, power supply variation, electronics drift, and crystal drift.

The servo may or may not require a timing loop on chip. A more traditional approach is to have the timing control come from an off-chip head position servo control ASIC; however, this method will allow slight errors due to its asynchronous nature. If such errors cannot be tolerated, then a timing loop must be added. This servo loop would not need to be complex, as the servo information is traditionally written at only one frequency. It must, however, be able to acquire phase-lock quickly, similar to the read loop. Another possibility would be to use the same loop for servo and read. This would require switching frequencies and acquiring phase extremely fast, which could prove to be quite difficult.

2.4 Jitter

One of the toughest demands on this PLL is its jitter requirement. There are two main types of jitter: cycle-to-cycle jitter and absolute jitter. Cycle-to-cycle jitter compares the difference in period between adjacent cycles of a clock and reports that as jitter. Absolute jitter finds the Least-Mean-Squared (LMS) frequency over a determined number of clock samples (8192 for this particular application, which approximates two sectors of data) and compares the actual clock edge with the ideal clock edge according to the LMS frequency. This specification becomes more difficult as the number of clock samples increase. For example, a slow drift in clock frequency may show up as 30ps of

cycle-to-cycle jitter but 1ns of absolute jitter. The jitter requirement for this PLL is +/- 3% absolute jitter, which is typical for HDD industry. Therefore, a 400MHz clock translates into an allowable +/- 75ps of absolute jitter.

One characteristic that has been observed regarding jitter is how and where jitter in the loop is added. Jitter can be added in one of two ways. It can be added such that it affects only the single edge in reference to an ideal clock, which can be thought of as high-frequency jitter, or simply 'jitter.' The second way of adding jitter is allowing it to accumulate. In this case, past history affects the current placement of the edge, which can be thought of as low-frequency jitter, a random walk, or simply 'wander.' This can be expressed as: actual_period(n) = ideal_period + jitter(n) - jitter(n-1) + wander(n), where jitter and wander are both random variables with gaussian distribution.

2.5 Data

An important topic to understand about magnetic media is that a logical "one" is polarized positively, and a logical "zero" is polarized negatively. When reading back what was written, polarization (flux) cannot be directly determined; only changes in polarization (flux) can be read directly off the disk. A step in data (all zeros followed by all ones) will be read back as an impulse, which means that a differentiation is inherent during read back. This impulse will be shaped like a Lorentzian pulse, however, and not an ideal impulse.

If these Lorentzian pulses were written far apart, there would be no need to worry about inter-symbol interference (ISI). The drawback is that user-data density would be extremely low. If these pulses are packed closer together, then past pulses would

interfere with the current samples; therefore, "an equalizer is used to shape the signal into one that has a controlled amount of ISI—hence the name 'partial response.' The partial response is taken into account by a detector, which distinguishes the received sequence of samples to identify the most likely written data sequence—hence the term 'maximum likelihood.'" [2] This is the premise of PRML (Partial Response, Maximum Likelihood) read channels.

When data is written to the disk, the preamble is written first. The preamble is extra data that allows the read loop time to establish lock to the data stream. A synchronizing byte(s) is written after the preamble to establish that user data is coming in the next byte, and it also sets the location of the byte boundary. The user data comes next, which is typically 512 bytes; however, this number is determined by the controller (PC) and is not a limitation of the disk.

2.6 Zero Phase Restart

A zero phase restart is an attempt to start (or line-up) the VCO with minimal phase offset compared with its reference. This term is most commonly used in reference to the read loop, where a read zero phase restart will lock the VCO to the data preamble. If the read loop is guaranteed to phase lock within the required time (the length of the preamble) starting with the correct frequency but arbitrary phase, then the read zero phase restart will not be necessary. But then again, if a read zero phase restart were used, the preamble might be able to be reduced, allowing more room for user data.

In this thesis zero phase restart will denote only an approximate attempt to line up the VCO with the reference clock at the output of the M and N dividers, which occurs during the coarse tuning mode of the write loop. This function is necessary due to the digital nature of the loop, which is discussed in further detail in the next chapter.

Chapter 3

The Algorithm

The foundation of this digital PLL is the acquisition algorithm. This algorithm must provide convergence over all ranges of initial conditions over all time. It also must guarantee that the PLL will not lock to a harmonic frequency. Keeping these two issues in mind, the algorithm must provide for fast and efficient frequency and phase locking. Remember that this algorithm has replaced the loop filter, and plays the critical role in determining loop dynamics. If the algorithm is too overdamped, locking can be assured, but the loop response will be slow. Too underdamped of an algorithm may create an unstable loop that will never converge or may oscillate.

This chapter first examines the loop dynamics of phase-locked loops in a traditional analog sense and then discusses the requirements for this digital loop. The next section develops the acquisition algorithm fundamentals. This thesis reaches its pinnacle in the third section with the introduction of an adaptation to the acquisition algorithm that has been dubbed phase optimal frequency estimation.

3.1 Loop Dynamics

Because the phase detector in this digital loop can only measure polarity of phase error and not quantity, zero steady-state phase error at the input of the phase detector is necessary. Most digital PLLs that detect phase error use counters clocked by a high frequency clock. This is not a possible nor acceptable solution for the proposed PLL, so one of the design criterion is to have zero steady-state phase error. This means that e(t) in Figure 3.1 must approach zero as time approaches infinity. The Final Value Theorem [3-1] allows for easy calculation of the steady-state error.

$$\text{Final Value Theorem: } \lim_{s\to 0} s\big(E(s)\big) = \lim_{t\to\infty} e(t) \qquad [3\text{-}1]$$

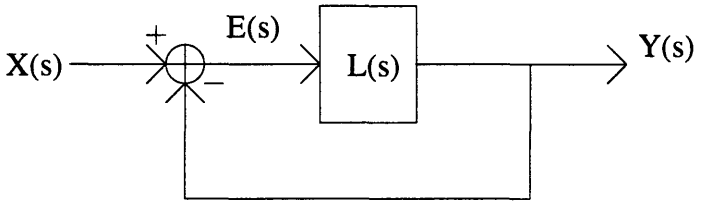

Figure 3.1. Steady-State Error Loop

$$X(s) - L(s)*E(s) = E(s) \;,\; \frac{E(s)}{X(s)} = \frac{1}{1+L(s)} \qquad [3\text{-}2]$$

From equation [3-2], the poles of 1+L(s) and the nature of the input X(s) are the determining factor in the calculation of $\lim_{t\to\infty} e(t)$. The input is a step in frequency; however, this must be translated into terms of phase. Frequency is the derivative of phase, so a step in frequency translates into a ramp of phase. The open loop transfer function L(s) is the total of the phase detector, loop filter, and VCO. The phase

detector's inputs and outputs are all in terms of phase, so the phase detector's contribution to L(s) is simply a constant. The relationship between the controlling voltage and frequency in the VCO is proportional, so in terms of phase, the VCO adds an integrator to L(s). Notice that the proportional terms in the open loop transfer function L(s) are absorbed into the constant term K.

$$\mathrm{L(s)} = \frac{K}{s} \quad , \quad \frac{E(s)}{X(s)} = \frac{s}{s+K} \qquad [3\text{-}3]$$

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0} s\big(E(s)\big) = \lim_{s\to 0} s\left(\frac{1}{s^2} * \frac{s}{s+K}\right) = \frac{1}{K} \neq 0 \qquad [3\text{-}4]$$

From equation [3-3] and [3-4], it is apparent that closing the loop without a loop filter is not satisfactory. Adding an integrator to the loop filter will have the following effect on L(s).

$$\mathrm{L(s)} = \frac{K}{s} * \frac{1}{\mathrm{s}} \quad , \quad \frac{E(s)}{X(s)} = \frac{s^2}{s^2+K} \qquad [3\text{-}5]$$

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0} s\big(E(s)\big) = \lim_{s\to 0} s\left(\frac{1}{s^2} * \frac{s^2}{s^2+K}\right) = \frac{0}{K} = 0 \qquad [3\text{-}6]$$

Having an integrator as a loop filter satisfies the steady-state phase error requirement; however, by examining the root-locus plot of L(s) in Figure 3.2, two integrators alone in a loop are oscillatory by nature. The two poles which are at the origin move up along the jω axis. In traditional analog loops, those in which a charge pump is connected to the loop filter, adding a lead compensation network to the loop filter pulls these poles off of the jω axis into the left hand plane. A lead compensator is used instead of just a single zero in order to prevent instantaneous changes in current from the charge pump causing instantaneous changes in voltage, which leads directly to

the VCO. Since this PLL has eliminated the charge pump and loop filter, a single zero in the left half plane is sufficient to properly pull these poles off of the jω axis. See Figure 3.2. This zero, when combined with the integrator, gives us a proportional plus integral (P+I) compensation scheme. Therefore, the algorithm must mimic a P+I compensator.

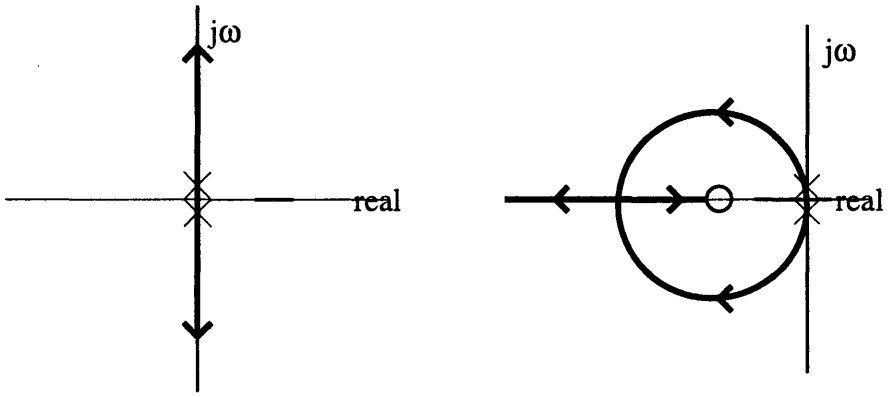

Figure 3.2. The root-locus plot of L(s)

To summarize, for the loop to have zero steady-state error to a step in phase, one integrator must exist in the loop. For the loop to have zero steady-state error to a ramp in phase, two integrators must exist in the loop. Since the oscillator acts like an integrator (converting from frequency to phase), one integrator comes for free, and the other must come from the loop filter. Two integrators alone make a loop oscillatory, so a left hand plane zero must be added to the loop filter. The algorithm must therefore mimic an integrator with a left hand plane zero, or simply a proportional plus integral compensator.

3.2 The Algorithm

This digital PLL uses an acquisition algorithm similar to the one used in "An All-Digital Phase-Locked Loop with 50-Cycle Lock Time Suitable For High-Performance Microprocessors." [3] By starting and stopping the VCO in an intelligent and known manner, a coarse tuning is achieved that leaves the VCO with a close approximation for frequency and phase. After accomplishing a coarse tuning, the acquisition algorithm moves into its fine tuning mode, during which time the VCO is left running continually. The frequency and phase are acquired in this mode, thereby achieving phase lock. The fine tuning mode is also responsible for maintaining phase and frequency lock.

The goal of coarse tuning is to acquire the approximate operating frequency of the VCO. In order to accomplish this, the VCO needs to be able to be started and stopped in a known fashion. The scheme depicted below in Figure 3.3 is one way to do this. After the reset (ZPR) falls low, the VCO and reference clock are enabled on the first rising edge of the reference clock. Both signals leading to the m and n dividers (the reference clock and VCO respectively) are started approximately at the same time. Note that the first period of the reference clock is reduced by one flip flop propagation delay, leaving three possibilities: 1) This is just a coarse setting and the delay isn't too important, 2) Try to equalize the delay by circuit replication, or 3) Implement a trimmed delay line. The coarse tuning state machine waits for the phase detector to make a fast or slow decision about the VCO. After this decision is made, the state machine adds or subtracts from the coarse word accordingly. Then the reset is toggled, and the whole process repeats for a specified number of times.

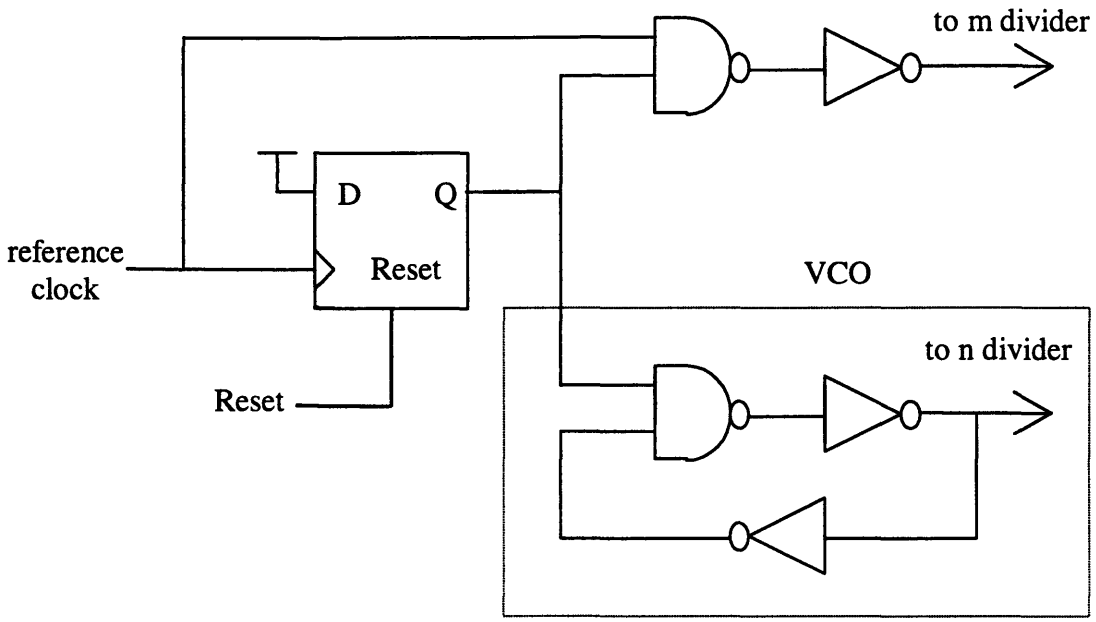

Figure 3.3. Scheme for aligning the VCO and reference clock

In the simulations run with this digital loop, the coarse eight-bit word starts at the midpoint, 128, and has the following either added or subtracted from it in this order: 64, 32, 16, 8, 4, 2, 1, 1. After the last one has been added or subtracted, no more zero phase restarts take place. The last state that the coarse tuning state machine ends up in is called fine tuning.

Fine tuning is divided into two sections: proportional correction and integral correction. Proportional correction occurs continuously, and can be thought of as phase changes. After each phase detector decision, the VCO's phase will either increment or decrement depending on whether it is fast or slow. Integral correction occurs only after the first five phase decisions in the same direction in a row, and every third time

thereafter until the phase decision switches direction. Integral correction changes the fine tuning word, whereas proportional correction does not.

This algorithm is very simple in nature because the proportional and integral correction are both fixed to one step increments. The consequence of a fixed step algorithm is an effect that can be described best as saturation. In a traditional analog loop operating in the linear region, loop characteristics such as overshoot and settling time do not depend upon the input. The output of the phase detector is proportional to the phase error. If the phase detector reaches a rail such that the output remains constant with increasing phase error at the input (saturation), the loop would no longer be linear. This is what happens in this digital loop, except the rail is at zero, so that only polarity of phase error can be determined.

The alternative to a fixed step algorithm is an adaptive algorithm. In an adaptive algorithm, the step size varies depending on previous decisions of the phase detector. The greater the number of like decisions in a row, the greater the step size becomes. For example, if three consecutive one step integral corrections are requested, the algorithm would increment the integral correction step size to two. It is possible for an adaptive algorithm to achieve lock quicker than a fixed step algorithm, but this algorithm can become extremely complex, depending on the number of previous decisions it bases its output on and how many different step sizes it allows.

So why was simplicity chosen over better performance? The answer is twofold, and sends us to the heart and soul of this thesis. The main reason is that a better solution exists, which is developed in the next section. The second reason is for ease in characterizing the loop dynamics of this new algorithm in the fifth chapter. This new

algorithm is actually an adaptation (complementary, not supplementary) of the above described algorithm, and it would be desirable to examine the effects of this adaptation itself, not some complicated underlying algorithm.

3.3 Phase Optimal Frequency Estimation

By examining the nature of an overdamped waveform in Figure 3.4, we find that when the signal is farthest off in frequency (the first overshoot), it is closest in phase. This holds true for any of the peaks in the overdamped response, due to the fact that frequency is the derivative of phase. Therefore, anytime phase error changes directions, the frequency error is at its greatest value. The following example helps illustrate this point.

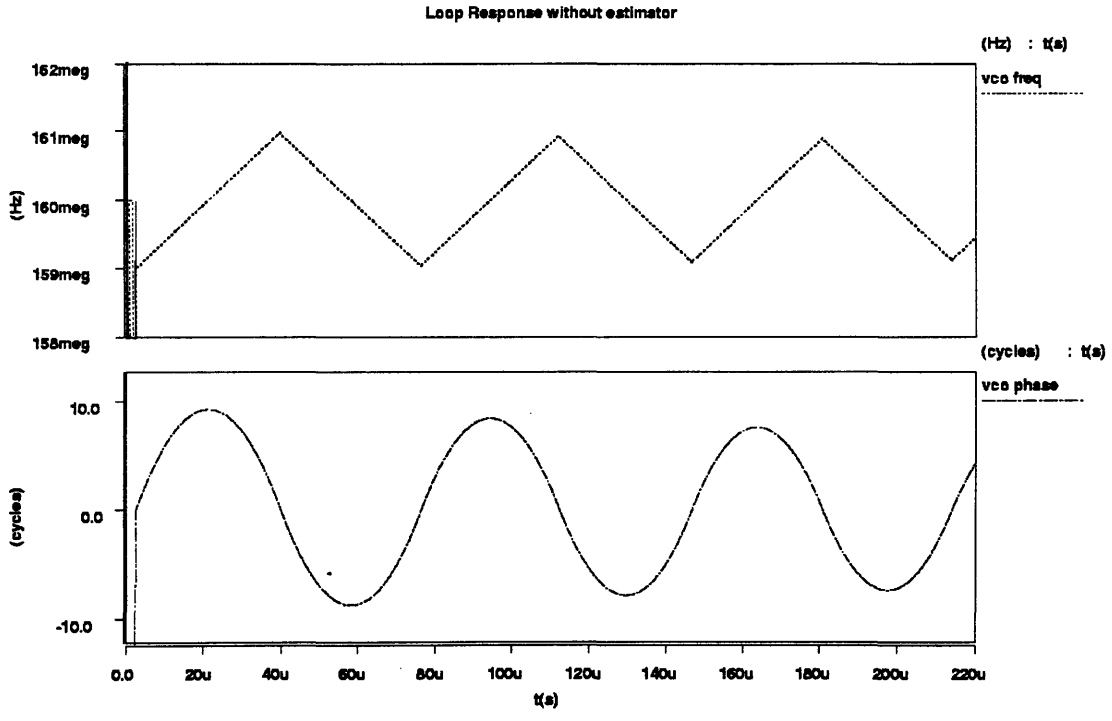

Figure 3.4. Loop response of acquisition algorithm

The coarse tuning will leave the VCO at a frequency ω_o, which is close to the ideal frequency ω_x. If a one step integral frequency correction is used (for simplicity), the two phases are in alignment at time zero (original frequency) and after M cycles (the first overshoot), with M to be determined.

$$\frac{M}{\omega_x} = \sum_{n=0}^{M-1} \frac{1}{\omega_o + n\Delta\omega} = \sum_{n=0}^{M-1} \frac{1}{\omega_o}(1 + n\frac{\Delta\omega}{\omega_o})^{-1} \qquad [3\text{-}7]$$

This equation [3-7] computes the time it takes for two signals that are in-phase but at different frequencies to become in-phase again, given in terms of cycles (M), when one frequency is fixed and the other is changing at a constant rate. It is important to note that the frequency step $\Delta\omega$ is a constant at any specified ω_o, although the value may be different at varying ω_o. Using the approximation [3-8] and the summation equation [3-9], we can come to the equation [3-10].

$$(1+x)^n \approx (1+nx) \text{ if } x \approx 0 \qquad [3\text{-}8]$$

$$0+1+2+3+...+M-2+M-1 = \frac{M(M-1)}{2} \qquad [3\text{-}9]$$

$$\frac{M}{\omega_x} \approx \sum_{n=0}^{M-1} \frac{1}{\omega_o}(1 - n\frac{\Delta\omega}{\omega_o}) = \frac{M}{\omega_o} - \frac{M(M-1)}{2\omega_o}\frac{\Delta\omega}{\omega_o} \qquad [3\text{-}10]$$

$$\frac{\omega_o}{\omega_x} \approx 1 - \frac{M-1}{2}\frac{\Delta\omega}{\omega_o} \qquad [3\text{-}11]$$

$$K_1 = \frac{\omega_o}{\omega_x}, \; K_2 = \frac{\Delta\omega}{\omega_o} \Rightarrow M \approx \frac{2(1-K_1)}{K_2} \qquad [3\text{-}12]$$

$(1\text{-}K_1)$ is the error between the original frequency ω_o and the ideal frequency ω_x. K_2 represents the step size as a percentage of the original frequency ω_o. Therefore the original starting frequency is approximately M/2 steps away from the ideal frequency.

Another way of stating this is the loop overshoots by approximately the same value of the original error. By this reasoning, the loop appears to be nearly oscillatory, which make sense because the example was pure integral correction. Adding proportional correction reduces the multiple of two in equation [3-12], making the overshoot smaller in nature.

From the above example, it has been shown that the loop overshoots the ideal frequency by nearly equal the original error in an integral correction algorithm. This alone leads to slow locking times for the loop. However, if the ideal frequency is thought of as lying approximately halfway between the original frequency and the frequency of the first overshoot, it seems apparent that a reasonable estimate for the ideal frequency is made by simply taking the midpoint of these two frequencies.

This estimation not only holds for between the original frequency and the first overshoot, but also for the new estimate and the next overshoot as well. The estimation places the VCO at a new original frequency, and then the whole process repeats again indefinitely. This is the basics of the phase optimal frequency estimation scheme. When the phase is optimal (when it is zero, or has just switched polarity), a frequency estimation occurs that places the estimated frequency midway between the original frequency and the overshoot frequency.

An instantaneous frequency change is desired, and realizing an instantaneous frequency change using digital techniques is not that difficult. This digital solution uses two fine tuning words; the first word controls the VCO, while the other word acts as a place holder for the new frequency estimate. Whenever phase error changes direction, the second word replaces the first word, and then the process is repeated.

The first word (VCO controller) changes every time an integral correction adds or subtracts one from the current word (1,2,3,4...). The second word changes every other integral correction, starting with the first (1,3,5,7...). If the second word did not change starting with the first add or subtract, a dead zone may exist where the loop's response would be more sluggish causing extra inherent jitter (chatter).

This resulting adaptation of the acquisition algorithm significantly reduces the time to achieve phase-lock, see Figure 3.5 below. (Compare with Figure 3.4.) A more thorough description of this algorithm's implementation for simulation purposes is discussed in the next chapter.

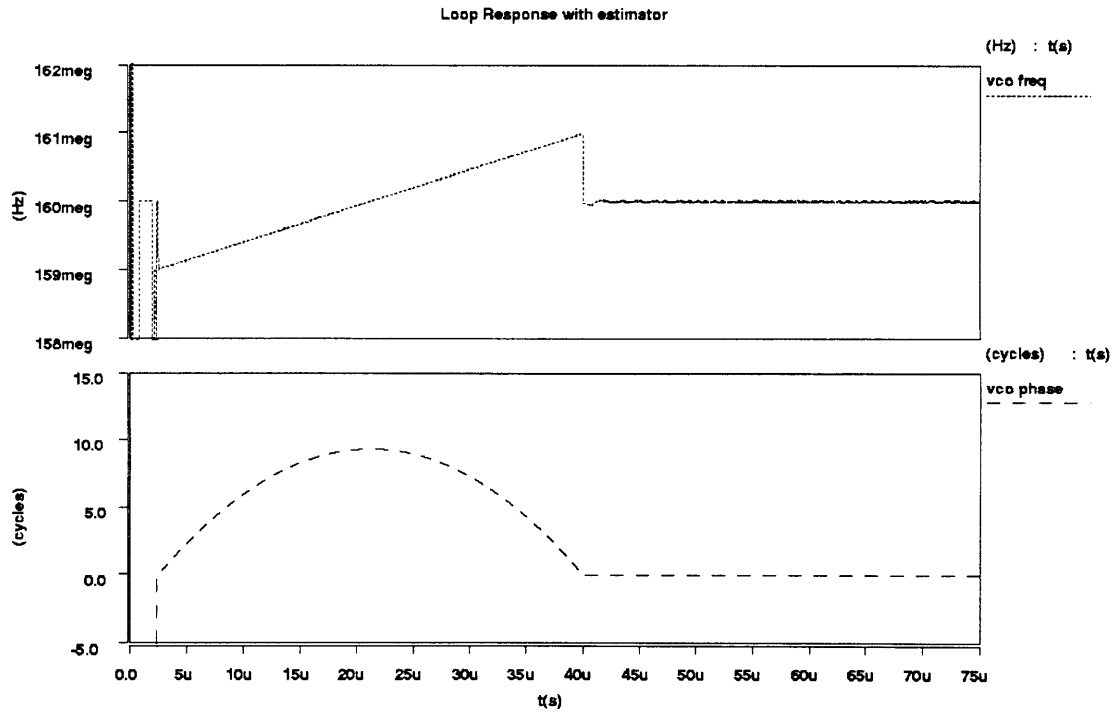

Figure 3.5. Loop response of acquisition algorithm with estimator

Chapter 4

PLL Structure

This chapter covers the actual implementation of the digital phase-locked loop with its different components. The first section examines the VCO and explains some of the specifics about resolution and noise. The next section covers the operation of the phase detector. The last two sections cover the implementation of the acquisition algorithm introduced in the previous chapter, with special attention given to the phase optimal frequency estimator.

4.1 Voltage Controlled Oscillator

Quantized error is a fundamental issue that arises in digital PLLs. Theoretically, analog systems are infinitesimally accurate. However, the accuracy of a digital system depends directly on the least significant bit (LSB) of the control word, which could best be thought of in terms of frequency. This limited resolution creates chatter at the final operating point, appearing as a small oscillation.

In order to meet jitter specifications, chatter must be limited, which means the LSB must be small, but how should it be referenced? If the LSB ($\Delta\omega$) is absolute (fixed for all frequencies), the resolution ($\Delta\omega/\omega_o$) will be greater at higher frequencies than lower frequencies. If the LSB ($\Delta\omega$) is relative (proportional to the operating frequency), then the resolution ($\Delta\omega/\omega_o$) will be constant at all frequencies.

The main argument for an absolute LSB is simplicity. The LSB would be referenced to an absolute value, instead of having a variable reference, which would invariably require more circuitry and could possibly add extra noise.

The main argument for a relative LSB is more consistent loop dynamics, which can best be thought of by referring back to equation [3-12].

$$K_1 = \frac{\omega_o}{\omega_x}\ ,\ K_2 = \frac{\Delta\omega}{\omega_o} \ \Rightarrow\ M \approx \frac{2(1-K_1)}{K_2} \qquad [4\text{-}1]$$

From equation [3-12] we find that if the LSB ($\Delta\omega$) is relative to the original operating frequency (ω_o), K_2 is a constant over all frequencies. This means that the number of steps away from the ideal frequency is only a function of the percentage amount of error, and not the operating frequency of the VCO. The relative LSB imitates a traditional analog loop better than does the absolute LSB, and is therefore used in this digital PLL.

The VCO is implemented as an ideal oscillator. The acquisition algorithm imposes one necessity for the VCO, and that is the ability to be stopped and started in a known fashion, which is most easily accomplished by a ring oscillator. The actual structure is not looked at, and will be left as a black box abstraction.

The VCO is designed to imitate three different noise sources. The first is jitter, which is noise referenced to the ideal clock edge. The second is wander, which is noise that accumulates over time, similar to a random walk. The third is tones on the power supply, which has the effect of modulating the VCO frequency. Section 2.4 covers jitter and wander in more detail.

4.2 Phase Detector

The phase detector consists of three main blocks. The first block eliminates the first edge coming from the dividers after any zero phase restart. These edges happen immediately after a zero phase restart, and are already aligned, so there is no need to make a decision on them. If a decision were to be made on the first edge from the dividers, the order of the two signals would be arbitrary, and the coarse tuning would fail. The second block makes a fast or slow decision, based on whether the M divider (reference clock) or N divider (VCO) was fast or slow. The third block stores the last five decisions into states. The coarse tuning uses only the most recent state, while the fine tuning uses all five.

4.3 Coarse Tuning

The coarse tuning consists logically of a finite state machine and an eight-bit adder. The state machine performs a zero phase restart, waits for the decision from the phase detector, and then based on that decision, sends out an eight-bit word to the adder. The eight-bit adder has its output fed back to one of its inputs through a register, so it acts as an accumulator. After repeating this process several times, the final state of the coarse tuning state machine initiates the fine tuning mode.

4.4 Fine Tuning

The fine tuning consists of control logic and two pointers, one for the VCO and one for the phase optimal frequency estimation. The control logic uses the five states of the phase detector to determine if proportional or integral correction is needed. The most recent state directly governs the proportional control, and other than that is left alone.

It is important to understand how this will look when doing loop simulations. In the ideal case (with no noise), if the ideal frequency is close to an LSB, the fine word will settle at this value. If the ideal frequency is midway between to LSBs, the fine word will oscillate between the two values. Both will have phase correction going on at all times. The proportional correction will have the effect of adding or subtracting an LSB from fine word between decisions. In the case where the ideal frequency is close to an LSB, the loop will oscillate between one LSB above and below the fine word, so it will appear as a two LSB oscillation. The other case will appear as a three LSB oscillation.

Figure 4.1 below gives a clearer understanding of the fine tuning portion of this algorithm. Please note that a zero represents the VCO being slow, and is therefore adding to the fine tuning word, while a one represents the VCO being fast, and is therefore subtracting from the fine tuning word. The most recent state is the left most bit, and the bits are shifted to the right as new decisions come in.

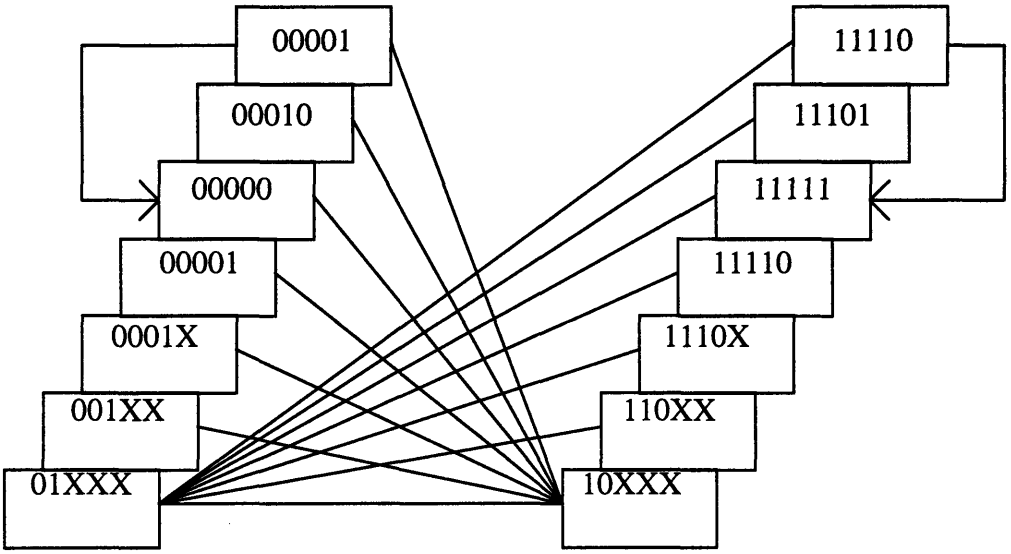

Figure 4.1. Fine tuning portion of acquisition algorithm

Integral correction occurs when all five states are the same. (00000 or 11111). Phase optimal frequency estimation occurs when the two most recent states are different. (01XXX or 10XXX). Once an integral correction is performed, from state 00000 or 11111, the fourth bit is hard wired so that it will toggle, jumping to state 00010 and 11101 respectively, when the next decision is made. This allows integral correction every third consecutive decision, instead of on every decision.

Chapter 5

Simulations

In the ideal world, the proposed adaptation to the acquisition algorithm achieves significantly faster locking times than using the algorithm alone. However, can this adaptation provide better performance in a noisy environment that more accurately models the conditions it must operate under? The following chapter examines this question. Three separate issues of noise shall be addressed: noise inherent in a digital phase-locked loop, noise resulting from circuit operation, and frequency tones on the clocks.

Inherent noise exists due to limited resolution and limited information. Limited resolution leads to chatter at the final operating point. Limited information refers to the time between updates at the phase detector, during which time the PLL is essentially run open loop.

Jitter is divided into three areas of concern: external jitter, loop jitter, and VCO wander. External jitter can be thought of as jitter coming from the reference clock (the crystal oscillator). Loop jitter can be thought of as jitter coming from within the loop,

possibly from a differential to single-ended converter at the VCO. VCO wander comes directly from jitter internal to the VCO. This appears as wander in the loop because the previous edge in an oscillator directly affects the current edge, and therefore has an accumulative effect.

Frequency tones on the clocks are the final area of interest. The tones on the reference clock give a reasonable approximation for the loop's small signal response, demonstrating its ability to track an external reference. The tones on the VCO model the effect of power supply ripples that would modulate the VCO frequency.

Note: Throughout the rest of this chapter, many numbers will appear in graphs and in figures, most reflecting jitter measurements at the VCO. These numbers are meant to be used mainly as a relative comparison, and not as an absolute reference. Also, additive jitter and jitter measurements are given in terms of standard deviations.

5.1 Chatter

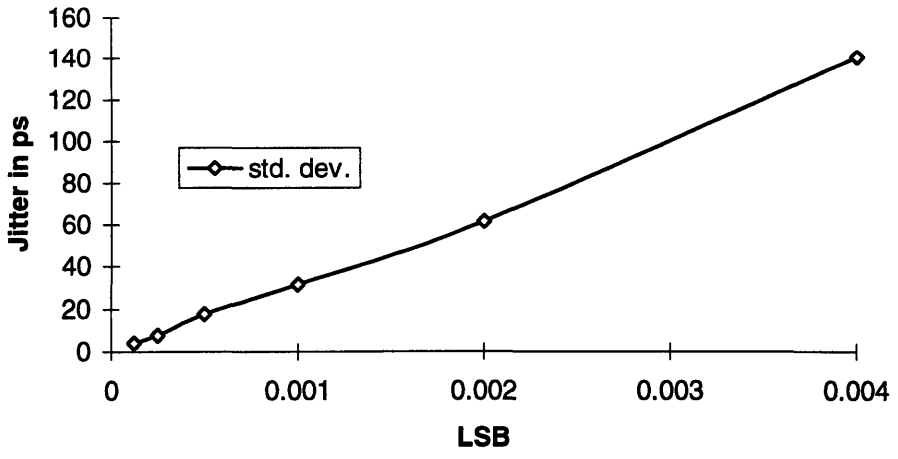

Figure 5.1. Jitter vs. LSB

As mentioned in section 4.1, quantized error is a fundamental issue that arises in digital PLLs. In a noiseless world, the jitter measured on the VCO will decrease proportionally as the resolution increases, as shown in Figure 5.1. The M and N divider and reference clock were kept constant as the LSB resolution varied. This result is verified intuitively by remembering that the fine tuning word is what oscillates at the final operating point.

5.2 A Sampled System

Phase-locked loops are all sampled-data systems by nature, because continuous phase error measurements are not provided. New information appears at the phase detector at the rate of f_{refclk}/M. This fact plays a crucial role in the overall noise performance of any PLL. Figure 5.2 examines the effects of increasing the M divider on VCO jitter.

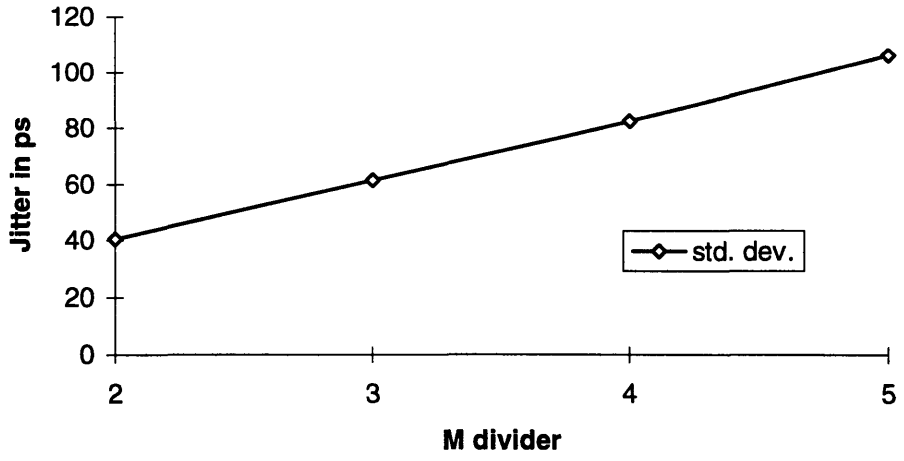

Figure 5.2. Jitter vs. M Divider

Increasing the M divider decreases the ratio f_{refclk}/M thereby reducing effective sampling rate of the phase error. The same relationship holds for changing the reference clock frequency. Increasing the reference frequency increases the effective sampling rate of the phase error, and this reduces the jitter measured at the VCO, shown in Figure 5.3.

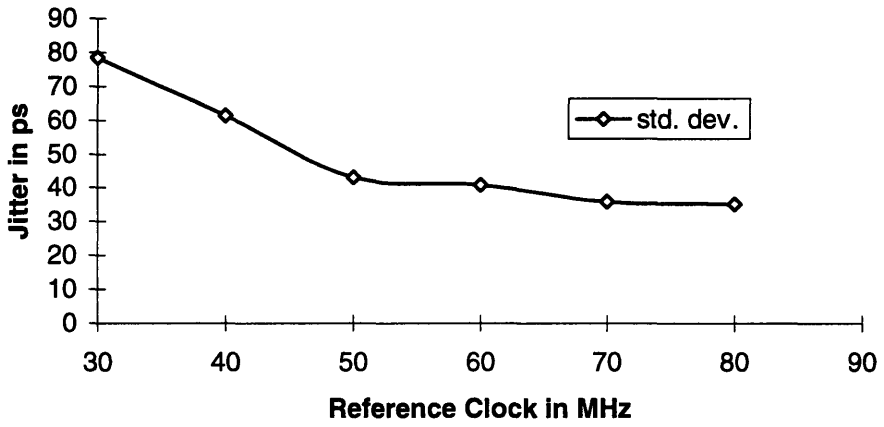

Figure 5.3. Jitter vs. Reference Clock

The important issue to realize is that regardless of the nature of the loop, phase error sampling is an issue that must be dealt with. This is potentially hazardous in a digital loop because both limited information and limited resolution exist simultaneously. M may be required to be a fairly high number in some circumstances to achieve the desired N/M ratio. This means that the VCO will produce N cycles between updates at the phase detector. During these N cycles, the VCO (due to limited resolution) will be off by some amount that is effectively multiplied by N. As N becomes large (~100), the total error can become quite significant.

There are two basic solutions to this scenario. One solution is to limit the M and N dividers to some lower number. This is generally not an option. Such high numbers are used so that the most effective frequency to write user data to the disk can be realized, which is very important in the competitive hard disk drive market. The other solution is to make the resolution fine enough to handle these conditions.

5.3 Jitter on the Reference Clock

Figure 5.4 examines the effect of adding 100ps (picoseconds) of jitter to the reference clock at the VCO, while increasing the M and N dividers proportionally (keeping M/N constant). The three different signals represent the ideal case with no noise, noise added without the frequency estimator on, and noise added with the frequency estimator on. Please note that the LSB resolution for the next three sections is the same, so comparisons between the effects of different types of additive jitter are valid.

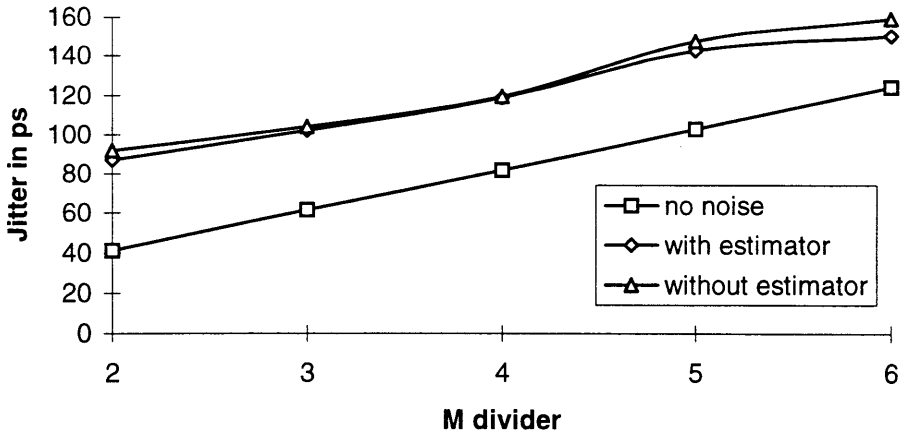

Figure 5.4. Jitter added to the reference clock

The data shows that adding reference clock jitter has nearly the same effect on VCO jitter regardless of whether the estimator is on or not, which is not that surprising considering that the jitter is not accumulative. The fact that the estimator has a slightly better performance can be explained by the fact that once the loop has settled, the estimator acts to retard deviation from the ideal operating point.

In examining VCO jitter caused by reference clock jitter, the LSB resolution plays an important role. The LSB resolution directly affects the loop gain, which in the traditional sense directly affects the loop bandwidth. As the resolution increases, the loop's ability to track higher frequencies decrease, which means that the loop's ability to reject high frequency reference clock noise increases.

5.4 Jitter on the VCO

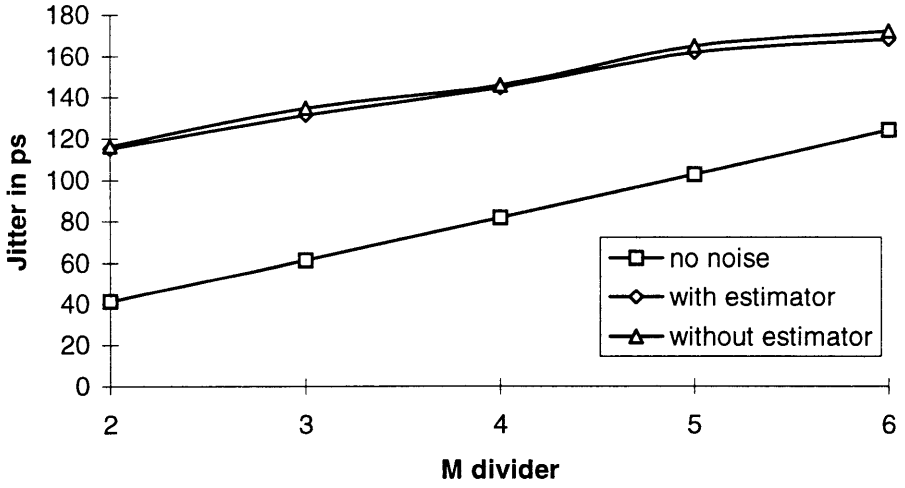

Figure 5.5. Jitter added to the VCO

Figure 5.5 examines the effect of adding 100ps of jitter to the VCO, while increasing the M and N dividers proportionally (keeping M/N constant). The three different signals represent the ideal case with no noise, noise added without the frequency estimator on, and noise added with the frequency estimator on. Notice once again that the estimator only makes the slightest improvement in noise rejection performance, which is not too surprising, remembering the non-accumulative nature of the jitter.

5.5 Wander on the VCO

In section 2.4, the following equation was explained: actual_period(n) = ideal_period(n) + jitter(n) - jitter(n-1) + wander(n). The effects of directly inserting jitter on the VCO were examined in the previous section. Therefore, since the wander term was zero, jitter was referenced to the ideal clock edge every period. However, if the wander term is nonzero, the effects of jitter become accumulative; past noise affects the current clock edge.

To illustrate this point, imagine a 100MHz clock. If this clock has zero-mean, 100ps jitter, after 100 cycles, the expected value of the time elapsed will be 1000 μs, while the standard deviation will be 100ps. However, if this clock has zero-mean, 100ps wander, after 100 cycles, the expected value of the time elapsed will still be 1000 μs, while the standard deviation will now be 1000ps. When adding two random processes, the means and variances add.

Figure 5.5 examines the effect of adding 100ps of wander to the VCO, while increasing the M and N dividers proportionally (keeping M/N constant). The three different signals represent the ideal case with no noise, noise added without the frequency

estimator on, and noise added with the frequency estimator on. Notice that the estimator makes a significant improvement in noise rejection performance. This example better illustrates the retarding effect that the estimator has on the loop. With the accumulative noise, the fine tune word must deviate further from its ideal position to maintain phase lock. When phase reversal occurs, the estimator will bring the fine word back in closer to the ideal operating point.

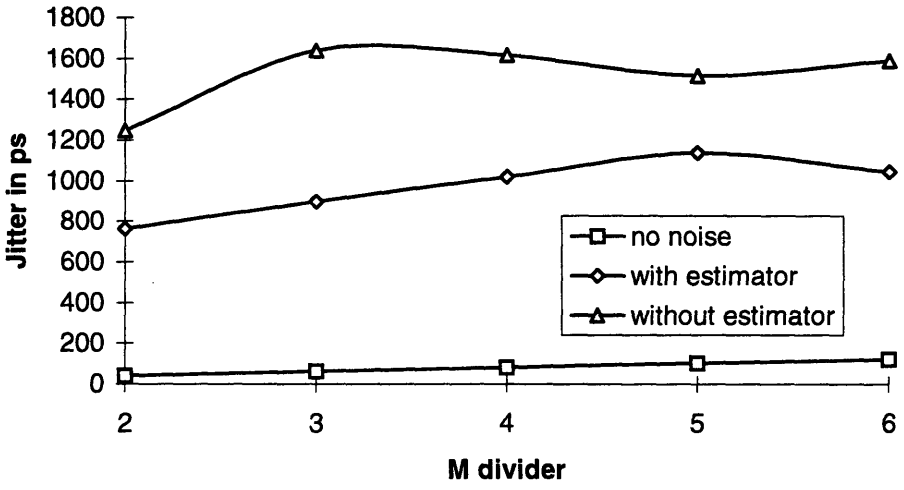

Figure 5.6. Wander added to the VCO

5.6 Tones on the Reference Clock

In the three previous sections, the effects the estimator has on the loop when noise is added have been examined. In each of these instances the ideal operating point for the fine word was constant, and the only reason the fine word changed at all (other than chatter) was to maintain phase lock at the phase detector. In the following two sections the effects of frequency tones will be examined.

Frequency tones affect the fine word in a different way than noise does. When noise enters the system, the fine word must adjust only to maintain phase lock at the phase detector, but the ideal operating location remains the same. When tones are present in the system, the fine word's ideal operating point must change to maintain both frequency and phase lock.

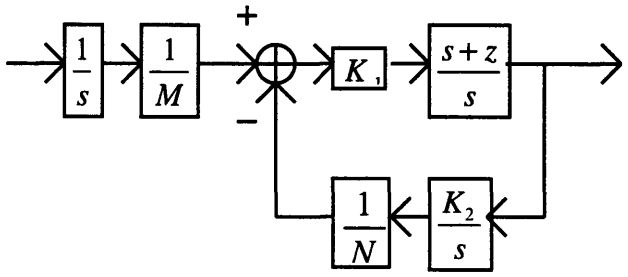

Figure 5.7. Digital phase-locked loop

From examining Figure 5.7, the closed-loop transfer function can be derived.

$$\frac{Y(s)}{X(s)} = \frac{1}{s}\frac{1}{M}\frac{K_1\dfrac{(s+z)}{s}}{1+\dfrac{1}{N}\dfrac{K_2}{s}K_1\dfrac{(s+z)}{s}} = \frac{\dfrac{K_1}{M}(s+z)}{s^2+\dfrac{K_1K_2}{N}(s+z)} \qquad [5.1]$$

From this transfer function we find that at lower frequencies, the response is a constant, and at higher frequencies, the response is a one-pole role off. The overall loop acts as a low-pass filter. From this reasoning, the loop will track low frequency tones on the reference clock, while it will reject high frequency tones. The following set of figures will demonstrate this effect. They show the tracking ability of the loop with and without the estimator at four different frequencies.

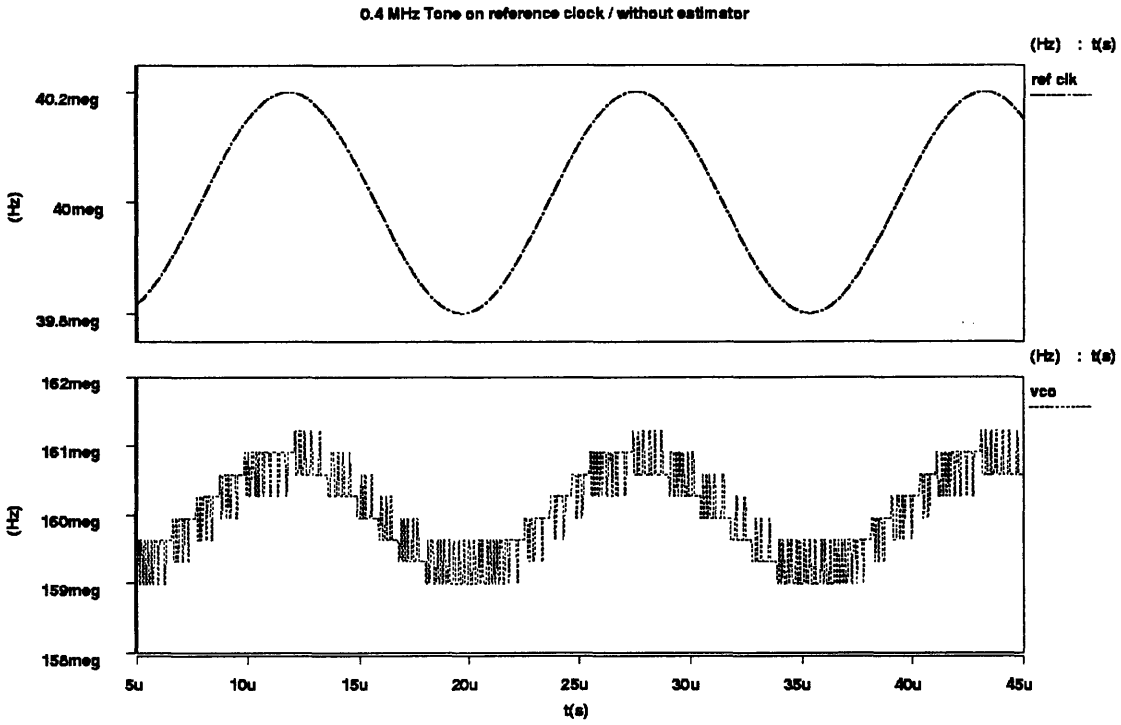

Figure 5.8. 0.4 MHz Tone on reference clock without estimator

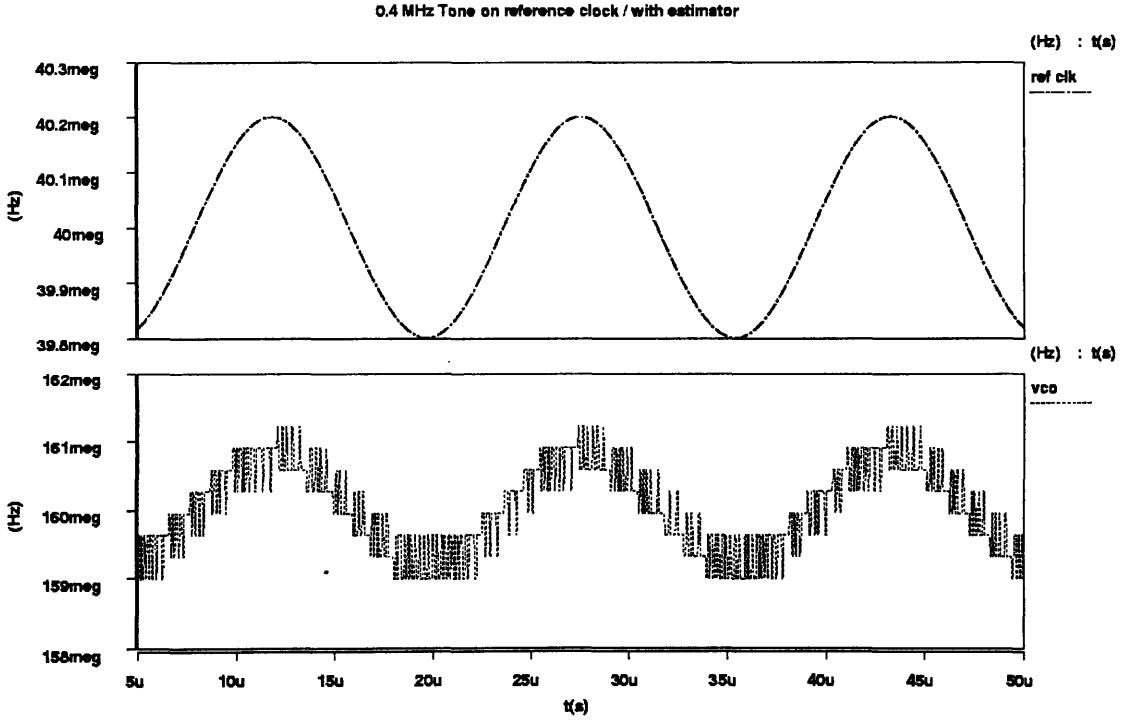

Figure 5.9. 0.4 MHz Tone on reference clock with estimator

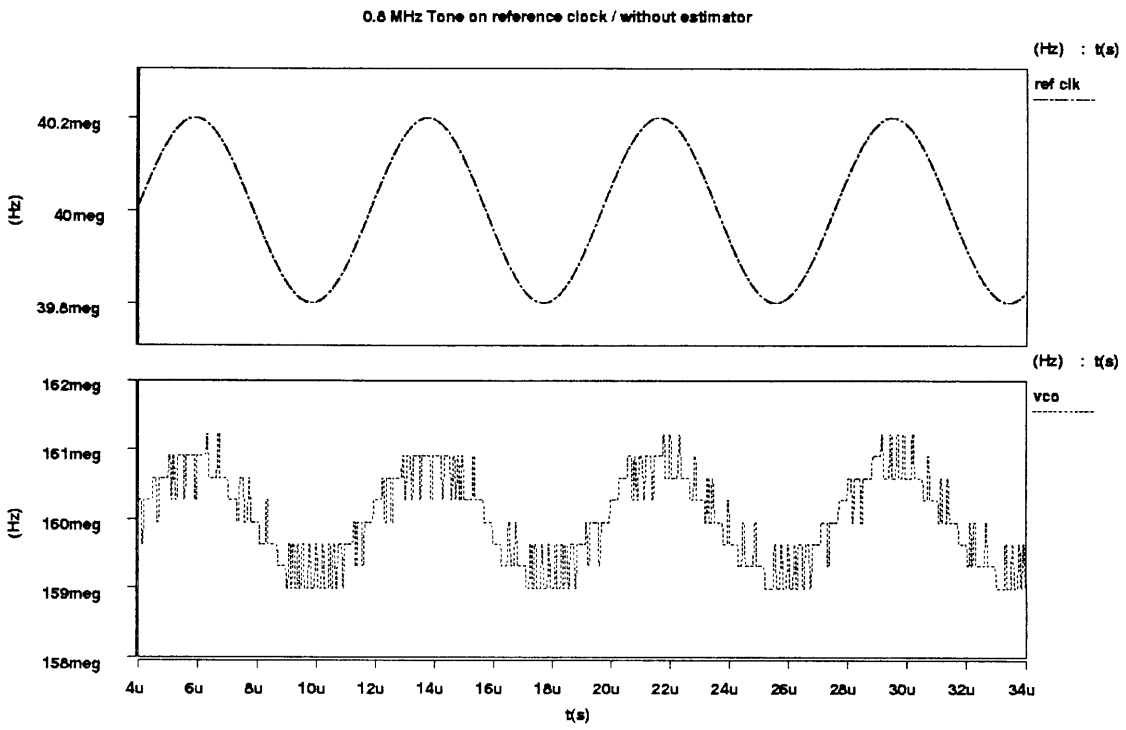

Figure 5.10. 0.8 MHz Tone on reference clock without estimator

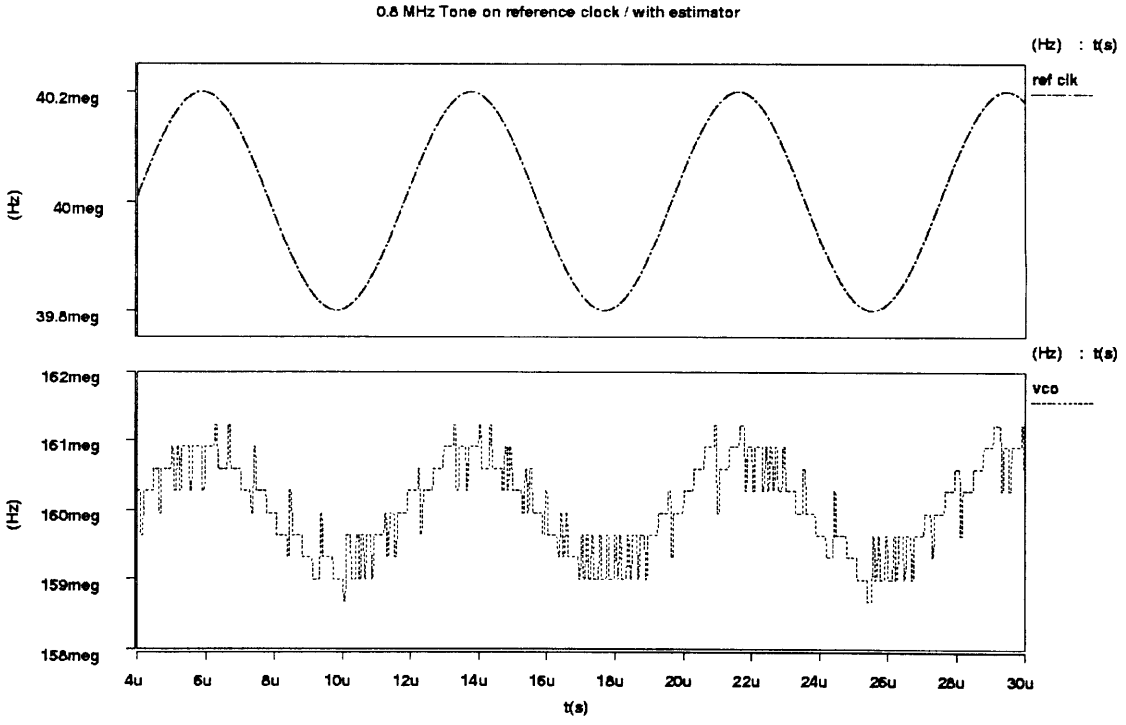

Figure 5.11. 0.8 MHz Tone on reference clock with estimator

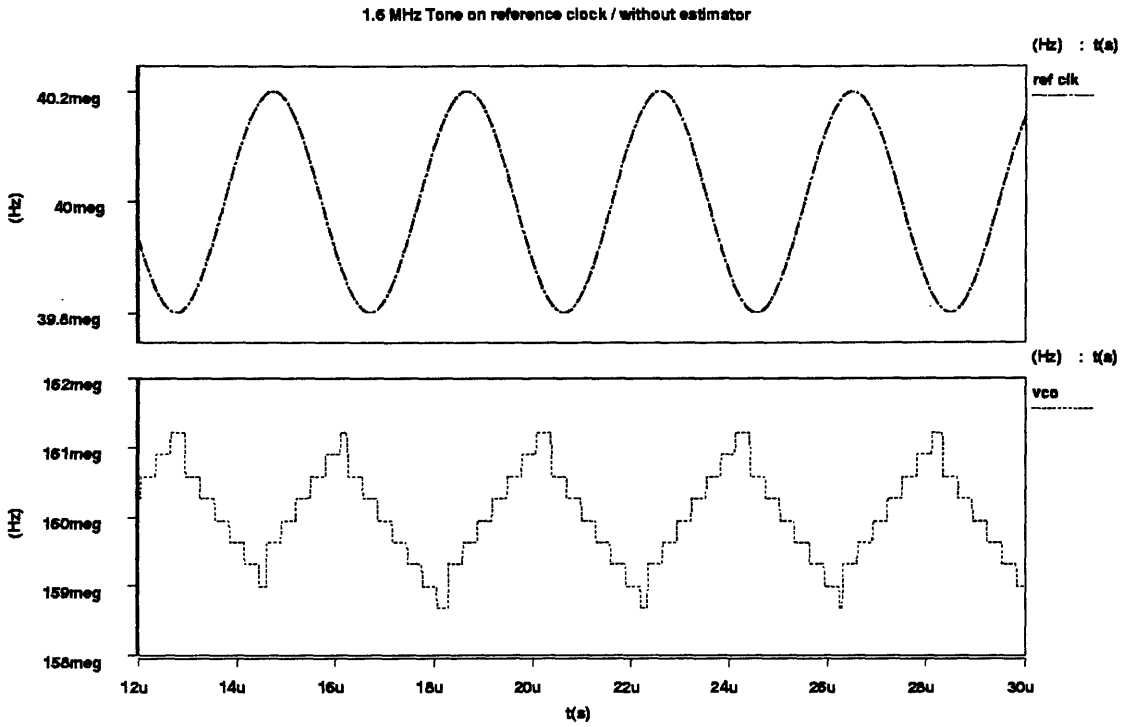

Figure 5.12. 1.6 MHz Tone on the reference clock without estimator

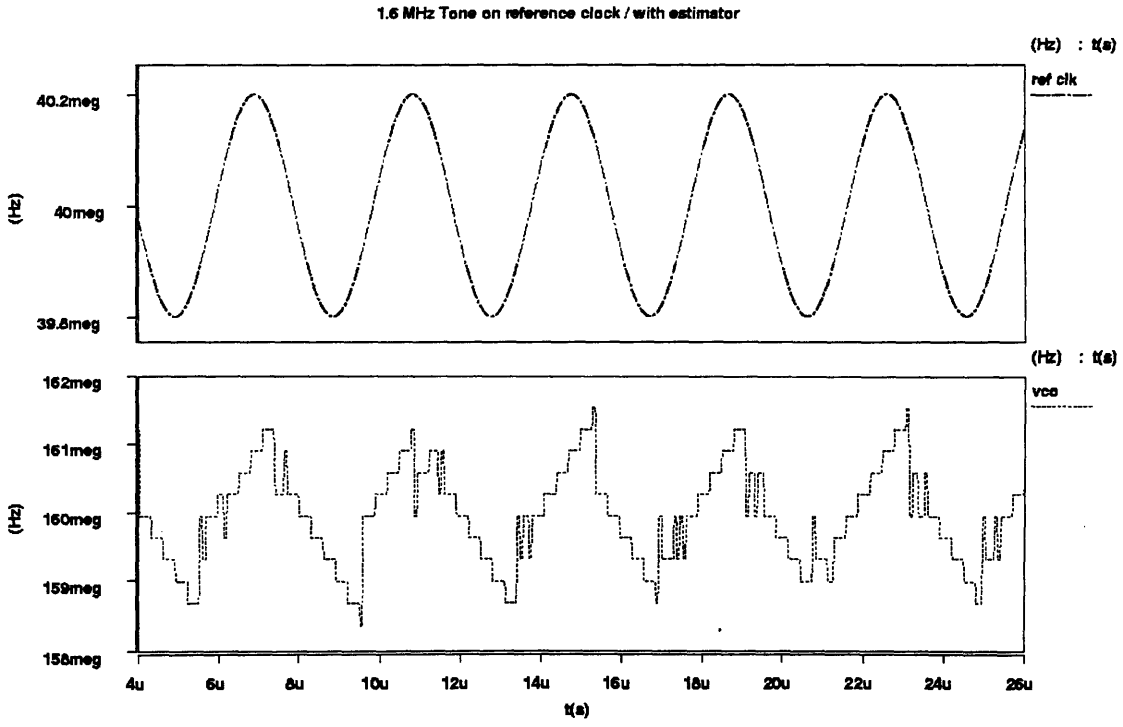

Figure 5.13. 1.6 MHz Tone on the reference clock with estimator

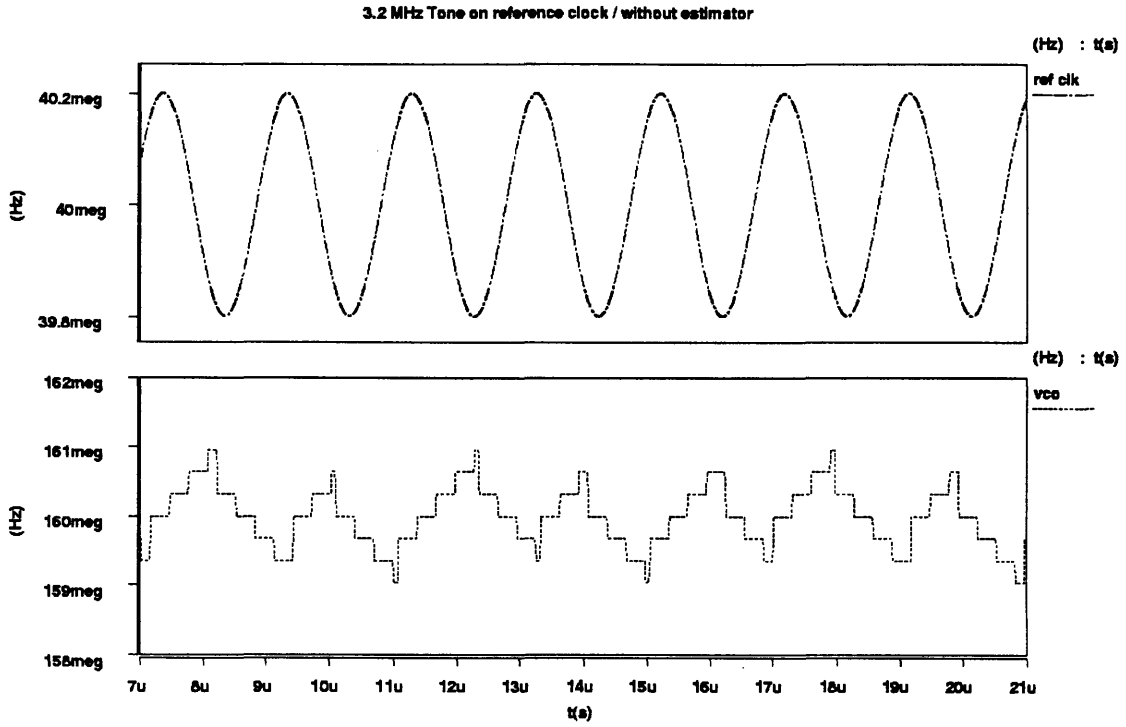

Figure 5.14. 3.2 MHz Tone on reference clock without estimator

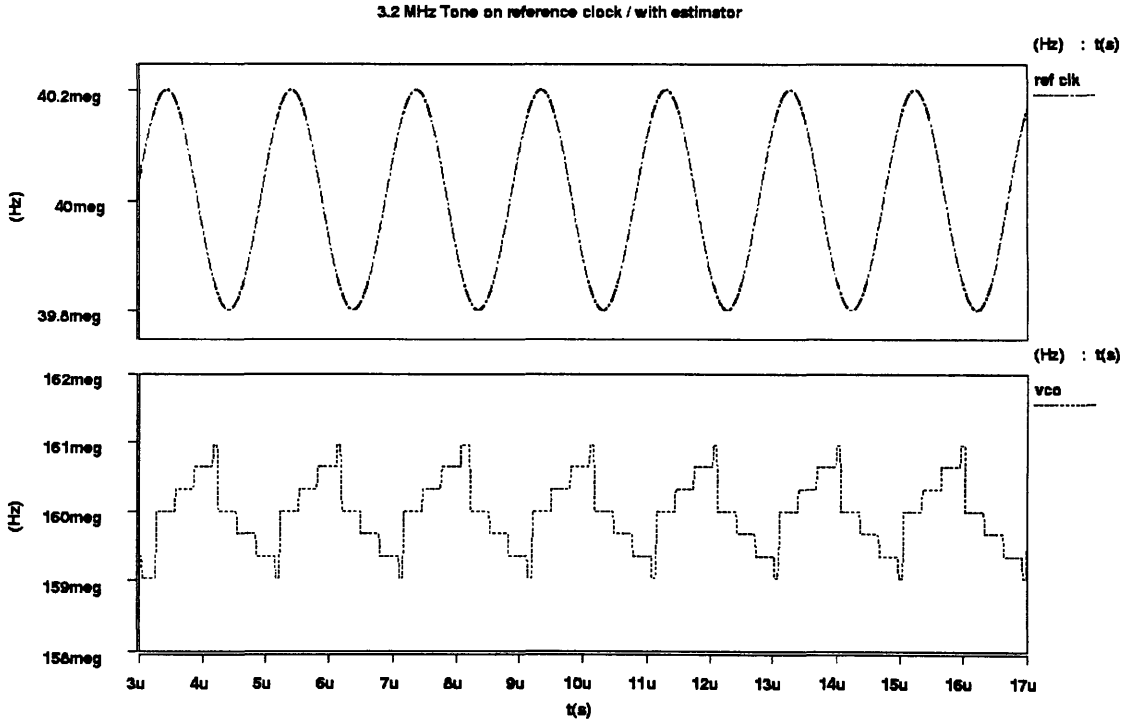

Figure 5.15. 3.2 MHz Tone on reference clock with estimator

The preceding figures show that the loop performs equally as well with or without the estimator. When frequency tones appear on the reference clock, the VCO is supposed to track them at lower frequencies, and reject them at higher frequencies, which it does. The VCO frequency changes directly as the fine tuning word changes.

5.7 Tones on the VCO

When frequency tones appear directly on the VCO, a different scenario happens. In this case the VCO should remain constant while the fine tuning word changes. An example will help demonstrate this idea. Imagine the power supply having a ripple on it. When the power supply increases, the frequency will increase even though the control word remains the same. In fact, the fine control word must decrease in order to offset this increase in frequency.

The problem with this happening in this loop is that the estimator retards deviation from the ideal operating point. However, in this case the ideal operating point is changing. It seems conceivable that the estimator may have the wrong impact, and that it will hamper the fine word from changing in the proper direction, which is not good.

It seems plausible that some critical frequencies exist such that the fine word is just able to keep up with the frequency tone. However, when a phase reversal occurs, the fine word will jump back towards the average fine word, causing extra phase error. The following four figures show the effects of the estimator at two different frequencies. The first set of figures show that the estimator has no appreciable affect. The second set of figures show how the estimator can negatively affect the loop.

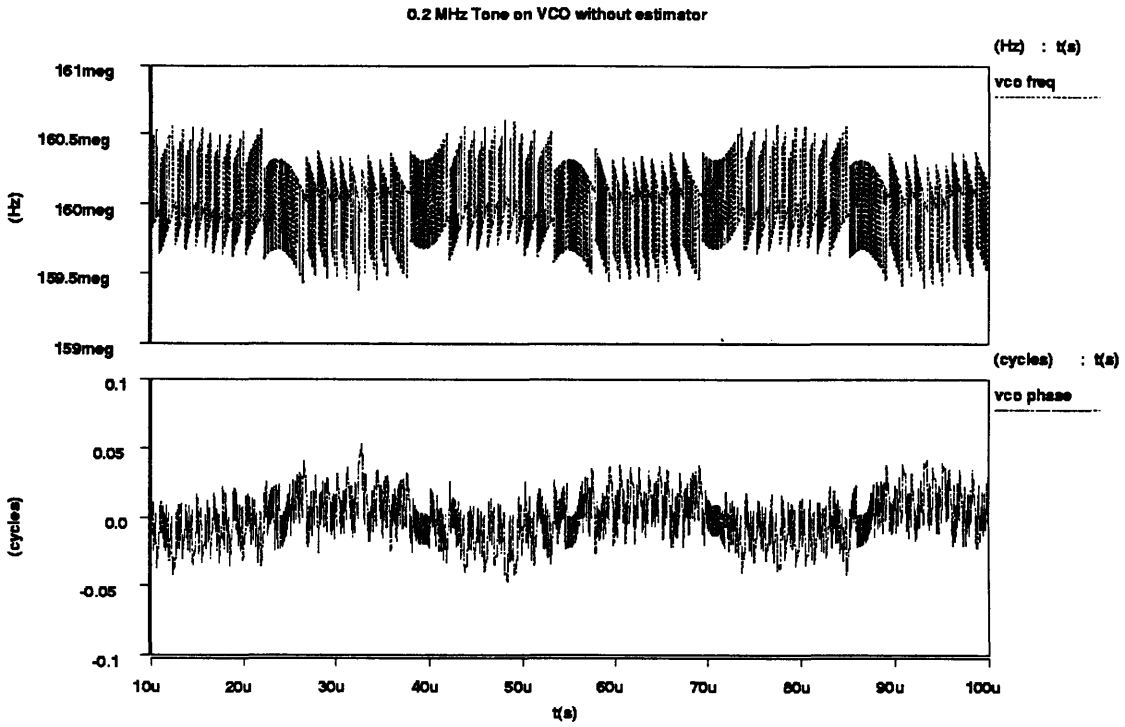

Figure 5.16. 0.2 MHz Tone on the VCO without estimator

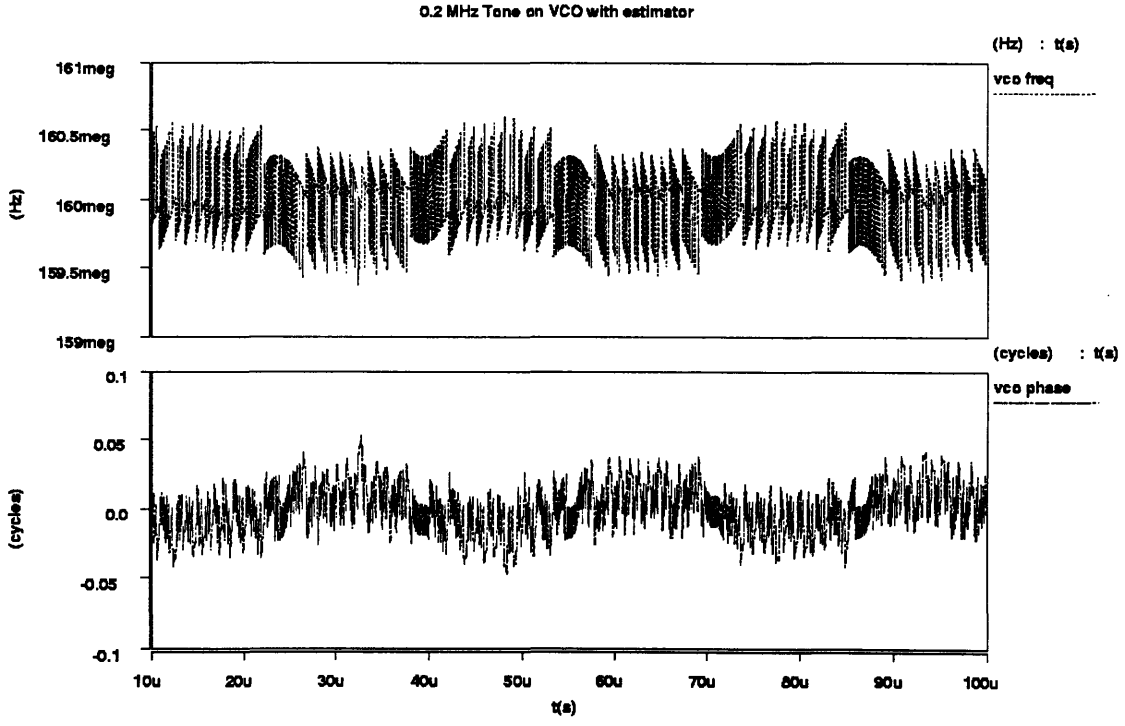

Figure 5.16. 0.2 MHz Tone on the VCO with estimator

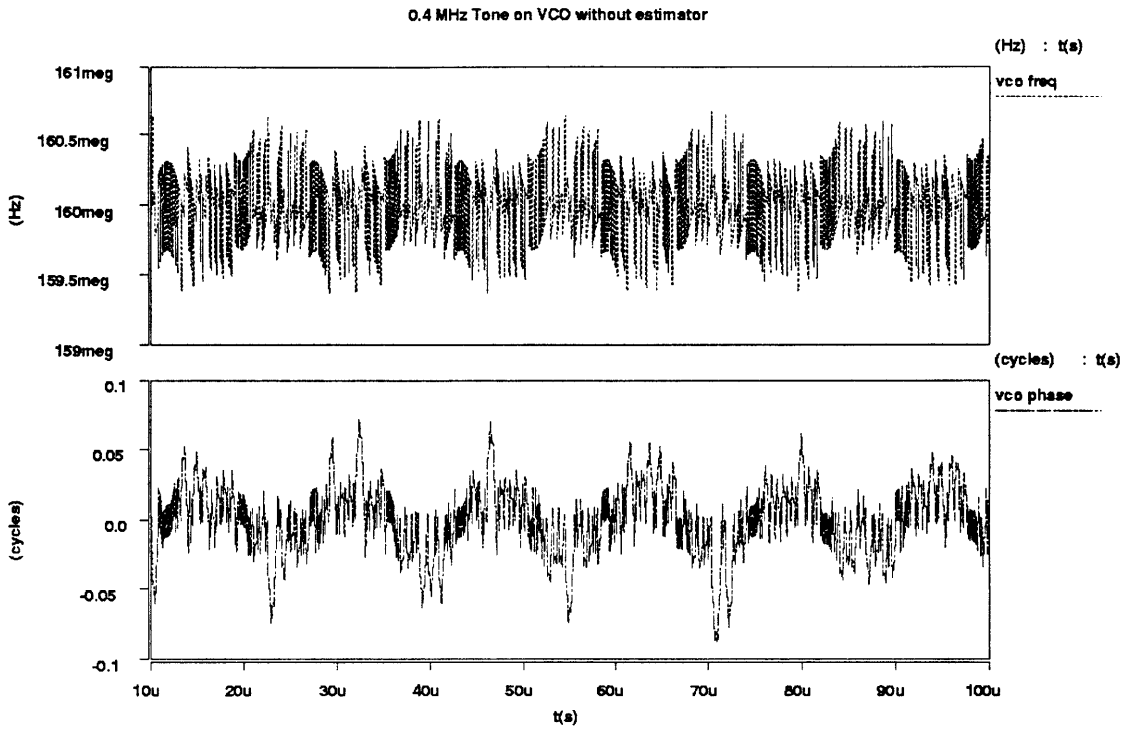

Figure 5.17. 0.4 MHz Tone on the VCO without estimator

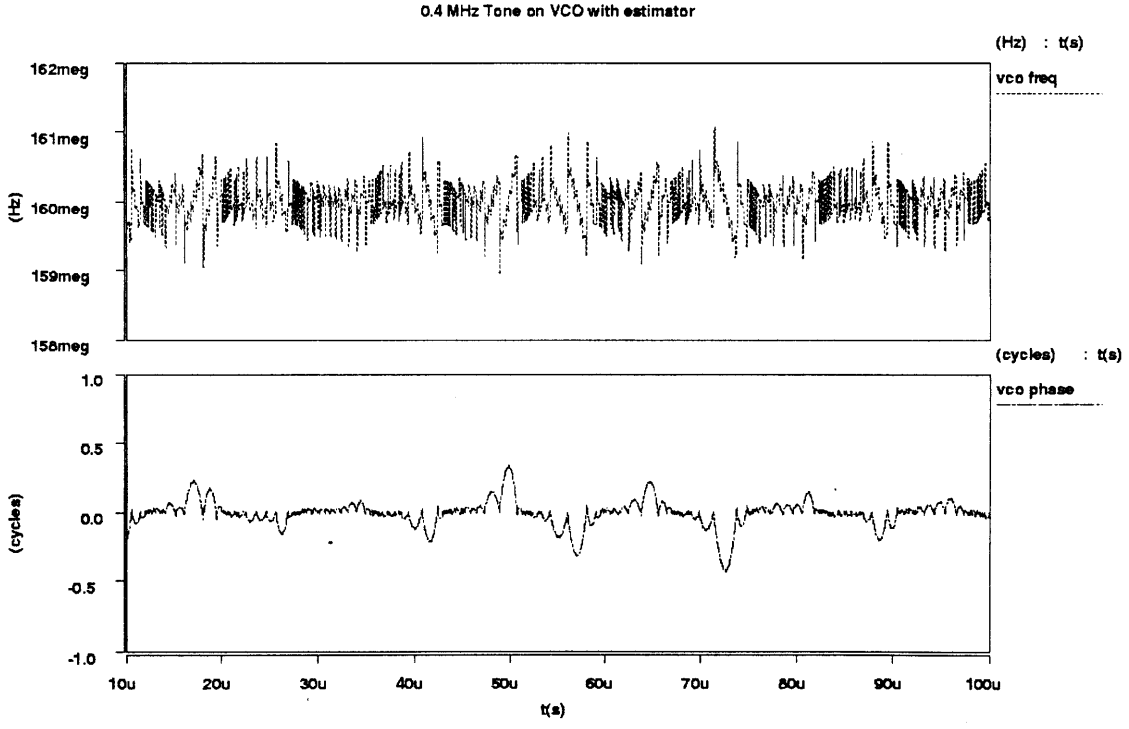

Figure 5.19. 0.4 MHz Tone on the VCO with estimator

Chapter 6

Conclusions

In the previous chapter, the algorithm underwent a complete characterization examining its performance under realistic conditions. The frequency estimator provides significantly faster locking times, as seen in Figure 6.1. The settling time of the loop with the estimator is directly proportional to original error and indirectly proportional to LSB resolution.

Non-accumulative jitter is rejected almost equally with or without the estimator functioning. Accumulative jitter, or wander, is rejected significantly more in the loop with the estimator than without. The estimator retards deviation from the ideal fine tuning word, which is why it rejects noise better.

Both loops act as a low-pass filter to frequency tones on the reference clock. The major fault of this adaptation is discovered when examining responses to frequency tones on the VCO. At some frequencies the estimator will actually take the fine tuning word further away from the correct location.

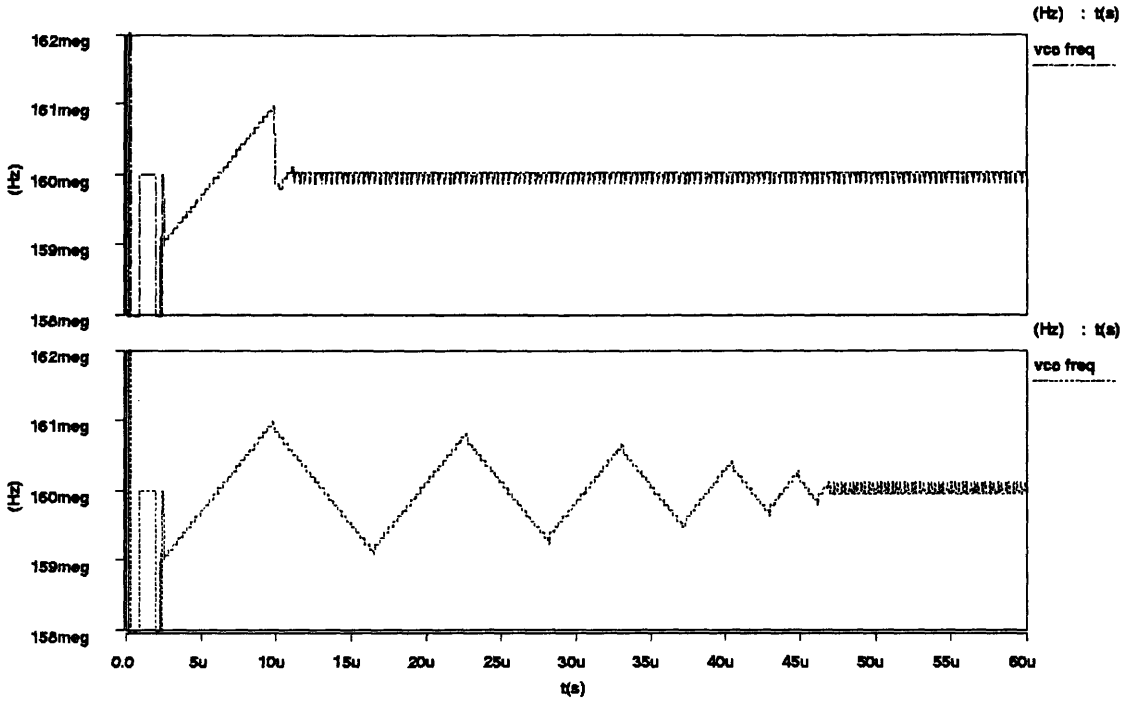

Figure 6.1. Locking times with and without the frequency estimator

Another feature in the digital PLL is the phase detector. Not being able to quantify phase error does have its drawbacks, but it also has its benefits. The loop is desensitized to noise quantity. Regardless of whether the VCO is 10ps or 200ps fast, the decision from the phase detector is the same.

This algorithm could truly prove to be useful in the digital domain, but it may also have uses in the analog realm. Replace the two fine tuning control words with two capacitors. When a phase reversal occurs, instead of taking the midpoint of two digital words, take the average of the two capacitors.

The frequency estimator adaptation to the acquisition algorithm is quick and simple, and it could prove to useful for a variety of applications.

Appendix A

Digital Phase-Locked Loop Code

This appendix includes all of the code written to simulate this digital phase-locked loop, and is included only for the extremely curious. The language is call MAST, and it is meant to run on an application called Saber, which is made by Analogy. The language is fairly readable, and the author is not nearly a good enough programmer to make it difficult to follow. Only the highlights of each program are documented.

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.                *
#*************************************************************************

xpll.x1 nvco coarse fine jit = m=3,n=15,vcojit=0,vcowan=0,\
        refjit=0,fr=40meg,fstep=0.00125
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.  *
#*************************************************************************

template xpll nvco coarse fine jit = m,n,vcojit,vcowan,refjit,fr,fstep,
        vka,vkf,rka,rkf

state logic_4 nvco
state nu coarse,fine,jit
number m=3,n=15,vcojit=0,vcowan=0,refjit=0,fr=40meg,fstep=0.00125
number vka=0,vkf=0,rka=0,rkf=0

{

xdco.i18 output:nvco stopz:stopz fine:fine coarse:coarse = jit=vcojit, \
        wan=vcowan,fmax=355000000,fmin=100000000,fstep=fstep,ka=vka,kf=vkf
xzpr.i17 zrclk:zrclk stopz:stopz reset:reset zpr:zpr \
      refclk:refclk logic_reset:logic_reset logic_zpr:logic_zpr
xcoarse.i14 coarse:coarse clk:refclk reset:reset cf:cf input:input
xreg.i13 q4:logic_reset q3:logic_zpr q2:cf q1:st reset:reset clk:refclk \
      d4:reset_in d3:zpr_in d2:cf_in d1:nstate_in
xlogic.i12 under:under over:over change:change as:q0 st:st x:input \
      reset:reset_in zpr:zpr_in cf:cf_in nstate:nstate_in
xfine.i11 sub2:sub2 add2:add2 under:under over:over fine:fine \
      load:kx zpr:zpr y:y x:x sub:sub add:add
xpd4.i9 zpr:zpr cf:cf ki:ki add:add sub:sub add2:add2 sub2:sub2 \
      x:x y:y kx:kx q0:q0 q1:q1 q2:q2 q3:q3 q4:q4
xpd3.i8 change:change zpr:zpr clk:refclk ki:ki q0:q0 q1:q1 q2:q2 \
      q3:q3 q4:q4 vcofast:vcofast vcoslow:vcoslow
xpd2.i7 vcoslow:vcoslow vcofast:vcofast down:down up:up
xpd1.i6 vco:nvco zpr:zpr up:up down:down mdiv:mdiv ndiv:ndiv
xdiv.i15 reset:zpr out:ndiv in:nvco = x=n
xdiv.i5 reset:zpr out:mdiv in:zrclk = x=m
xrefclk.i4 out:refclk = jit=refjit,fr=fr,ka=rka,kf=rkf
xjitter.ijit clk:nvco jit:jit = m=m,n=n,fr=fr

}
```

```
#*********************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*********************************************************************

# This is a digitally controlled oscillator which takes two inputs:
# coarse and fine, and creates an output signal with a frequency
# based on the algorithm presented below, assuming the input stopz
# is not enabled.  If stopz is enabled, the output is held low.

# The arguments fmax and fmin establish a range for the coarse tuning
# using an eight-bit word (0-255) while fstep establishes what
# percent of the coarse word each bit of the fine word will be worth.
# The fine tuning uses an ten-bit word (0-1023) and is centered at 512.

# The arguments jit and wan setup short term and long term jitter
# characteristics respectively on the output.

template xdco output stopz coarse fine = jit,wan,fmax,fmin,fstep,ka,kf

state logic_4 output, stopz
state nu coarse, fine
number fmin=100000000,fmax=355000000,fstep=0.00125
number jit=0,wan=0,ka=0.01,kf=200000000

{

state nu tmp_fq,ts,t,ex=1,tick,tx
state logic_4 stopzold
state nu jitter, jitter_prev, wander

foreign mast_random    # foreign C function

when (event_on(coarse) | event_on(fine)) {
        tmp_fq=(fmin+(fmax-fmin)*coarse/255)*(1+(fine-512)*fstep)
        ts=1/(16*tmp_fq)
        }

when (dc_init) {
        schedule_event(time,output,l4_0)
        }

when ((time_init) | (event_on(stopz,stopzold))) {
        if ((stopz==l4_1) & (stopzold==l4_0)) {
                schedule_event(time,tick,1)
                ex=1
                }
```

```
        }

when ((stopz==l4_0) & (output~=l4_0)) {
        schedule_event(time,output,l4_0)
        }

when(event_on(tick) & (stopz==l4_1)) {
        tx=ts*(1+ka*sin(time*kf))
        if (jit == 0) t = tx
        else if (ex==8) {
                jitter = mast_random(jit,0,0)
                if(abs(jitter) >= tx/2) jitter = 0
                t = tx + jitter - jitter_prev
                jitter_prev = jitter
                }
        else t = tx
        if (wan == 0) t = t
        else if (ex==8) {
                wander = mast_random(wan,0,0)
                if(abs(wander) >= t/2) wander = 0
                t = t + wander
                }
        schedule_event(time+t,tick,1)
        if (ex==1) {
                if (driven(output)==l4_0) schedule_event(time,output,l4_1)
                else schedule_event(time,output,l4_0)
                ex=2
                }
        else if (ex==2) ex=3
        else if (ex==3) ex=4
        else if (ex==4) ex=5
        else if (ex==5) ex=6
        else if (ex==6) ex=7
        else if (ex==7) ex=8
        else ex=1
        }

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*************************************************************************

# This circuit provide a reference clock with frequency fr at the
# output out.  The short term jitter characteristics are determined
# by the argument jit.

template xrefclk out = jit,fr,ka,kf

state logic_4 out
number jit=0,fr=40000000,ka=0.01,kf=80000000

{

state nu tick,t,jitter,jitter_prev=0
state nu ts = 12.5n

foreign mast_random    # foreign C function

when (dc_init) {
        schedule_event(time,out,l4_0)
        }

when (time_init) {
        if (fr==0) ts=12.5n
        else ts=0.5/fr
        schedule_event(time+ts,tick,1)
        }

when (event_on(tick)) {
        if (fr==0) ts=12.5*(1+ka*sin(time*kf))
        ts=0.5/fr*(1+ka*sin(time*kf))
        if(jit == 0) t = ts
        else {
                jitter = mast_random(jit,0,0)
                if(abs(jitter) >= ts) jitter = 0
                t = ts + jitter - jitter_prev
                jitter_prev = jitter
                }
        if (driven(out)==l4_0) {
                schedule_event(time,out,l4_1)
                schedule_event(time+t,tick,1)
                }
        else {
                schedule_event(time,out,l4_0)
```

```
                schedule_event(time+t,tick,1)
                }
        }

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.   *
#*************************************************************************

# This circuit is the finite state machine which controls the coarse tuning, of which the
# last state enables the fine tuning.

template xlogic change as st under over nstate x cf reset zpr = td

state logic_4 change, as, under, over, cf, zpr, reset
state nu st, nstate, x
number td = 0.5n

{

when ((event_on(change)) | (event_on(as)) | (event_on(st)) | \
        (event_on(under)) | (event_on(over))) {
        if (st==1) {
                schedule_event(time+td,nstate,2)
                schedule_event(time+td,cf,l4_0)
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,reset,l4_0)
                }

        else if (st==2) {
                if (change==l4_1) {
                        if (as==l4_0) schedule_event(time+td,x,64)
                        else schedule_event(time+td,x,-64)
                        schedule_event(time+td,zpr,l4_1)
                        schedule_event(time+td,nstate,3)
                        }
                }

        else if (st==3) {
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,nstate,4)
                }

        else if (st==4) {
                if (change==l4_1) {
                        if (as==l4_0) schedule_event(time+td,x,32)
                        else schedule_event(time+td,x,-32)
                        schedule_event(time+td,zpr,l4_1)
                        schedule_event(time+td,nstate,5)
```

```
                }
        }

else if (st==5) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,6)
        }

else if (st==6) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,16)
                else schedule_event(time+td,x,-16)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,nstate,7)
                }
        }

else if (st==7) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,8)
        }

else if (st==8) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,8)
                else schedule_event(time+td,x,-8)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,nstate,9)
                }
        }

else if (st==9) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,10)
        }

else if (st==10) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,4)
                else schedule_event(time+td,x,-4)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,nstate,11)
                }
```

```
        }

else if (st==11) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,12)
        }

else if (st==12) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,2)
                else schedule_event(time+td,x,-2)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,nstate,13)
                }
        }

else if (st==13) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,14)
        }

else if (st==14) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,1)
                else schedule_event(time+td,x,-1)
                schedule_event(time+td,nstate,15)
                schedule_event(time+td,zpr,l4_1)
                }
        }

else if (st==15) {
        schedule_event(time+td,x,0)
        schedule_event(time+td,zpr,l4_0)
        schedule_event(time+td,nstate,16)
        }

else if (st==16) {
        if (change==l4_1) {
                if (as==l4_0) schedule_event(time+td,x,1)
                else schedule_event(time+td,x,-1)
                schedule_event(time+td,nstate,17)
                schedule_event(time+td,zpr,l4_1)
                }
        }
```

```
        else if (st==17) {
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,nstate,18)
                }

        else if (st==18) {
                if (change==l4_1) {
                        if (as==l4_0) schedule_event(time+td,x,0)
                        else schedule_event(time+td,x,-1)
                        schedule_event(time+td,nstate,19)
                        }
                }

        else if (st==19) {
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,nstate,20)
                schedule_event(time+td,cf,l4_1)
                }

# State 20 is normal operating state during fine tuning mode.

        else if (st==20) {
                if (over==l4_1) schedule_event(time+td,nstate,21)
                else if (under==l4_1) schedule_event(time+td,nstate,23)
                else schedule_event(time+td,nstate,20)
                }

# State 21 is reached only if an overflow in the fine tuning word occurs.

        else if (st==21) {
                schedule_event(time+td,x,2)
                schedule_event(time+td,nstate,22)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,cf,l4_0)
                }

        else if (st==22) {
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,nstate,20)
                schedule_event(time+td,cf,l4_1)
                }
```

```
# State 23 is reached only if an underflow in the fine tuning word occurs.

        else if (st==23) {
                schedule_event(time+td,x,-2)
                schedule_event(time+td,nstate,24)
                schedule_event(time+td,zpr,l4_1)
                schedule_event(time+td,cf,l4_0)
                }

        else if (st==24) {
                schedule_event(time+td,x,0)
                schedule_event(time+td,zpr,l4_0)
                schedule_event(time+td,nstate,20)
                schedule_event(time+td,cf,l4_1)
                }
        }

}
```

```
#*********************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*********************************************************************

# This circuit takes the input in and provides a divide by x function,
# which is displayed at the output out. The output rises on the first
# edge of the input, and falls on the second. All edges of the output
# are delayed by the argument td. The rising edge of the output is
# effected by the argument jit, which is short term jitter. If reset
# is high, the output is held low; reset low is normal operation.

template xdiv in out reset = x,td,jit

state logic_4 in, out, reset
number td = 0.5n, x=4, jit=0

{

state nu div, nber = 0
state logic_4 inold, sch_q
state nu t,jitter,jitter_prev=0

foreign mast_random    # foreign C function

when (dc_init) {
        if (x<2) div=2
        else div=x
        }

when (event_on(in,inold) & (reset==l4_0) & (in==l4_1) & \
        (inold==l4_0)) {
        nber = nber + 1
        if (nber==1) {
                if(jit == 0) t = 0
                else {
                        jitter = mast_random(jit,0,0)
                        if(abs(jitter) >= (3*jit)) jitter = 0
                        t = jitter - jitter_prev
                        jitter_prev = jitter
                        }
                schedule_event(time+td+t,out,l4_1)
                }
        if (nber==2) schedule_event(time+td,out,l4_0)
        if (nber>=div) nber = 0
        }
```

```
when (event_on(reset) & (reset == l4_1)) {
        sch_q = l4_0
        nber = 0
        if (sch_q ~= out) schedule_event(time,out,sch_q)
        }

}
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*************************************************************************

# This circuit throws out the first rising edge of the m and n divider.

template xpd1 vco zpr up down mdiv ndiv

{

set_l4_1.i17 set1:hi
inv_l4.x9 out:n1 in:n2 = tphl=2e-9, tplh=2e-9
inv_l4.i7 out:n8 in:zpr = tplh=500e-12
nor2_l4.x6 in2:n6 in1:n4 out:n5 = tphl=500e-12, tplh=500e-12
nor2_l4.x5 in2:zpr in1:n5 out:n4 = tphl=500e-12, tplh=500e-12
nand2_l4.x4 in2:up in1:down out:n3 = tphl=500e-12, tplh=500e-12
nand2_l4.x3 in2:n3 in1:n4 out:n2 = tphl=500e-12, tplh=500e-12
dff_l4.x8 qn:net35 clk:vco d:hi s:hi r:n8 q:n7 = tp=500e-12
dff_l4.x7 qn:net41 clk:vco d:n7 s:hi r:n8 q:n6 = tp=500e-12
dff_l4.x2 qn:net47 clk:ndiv d:hi s:hi r:n1 q:down = tp=500e-12
dff_l4.x1 qn:net53 clk:mdiv d:hi s:hi r:n1 q:up = tp=500e-12

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*************************************************************************

# This circuit make the decision whether the vco is fast or slow.  Note:
# vcoslow = add, vcofast = subtract from control word.  The argument td
# is the delay associated with circuit.

template xpd2 up down vcofast vcoslow = td

state logic_4 up, down, vcofast, vcoslow
number td = 0.5n

{

state logic_4 upold, downold
state nu examine

when (dc_init) {
        examine = 1
        schedule_event(time,vcofast,l4_0)
        schedule_event(time,vcoslow,l4_0)
        }

when (time_init) {
        examine = 1
        schedule_event(time,vcofast,l4_0)
        schedule_event(time,vcoslow,l4_0)
        }

when ((event_on(up,upold)) | (event_on(down,downold))) {

        if (examine==0) {
                if ((down==l4_0) & (up==l4_0)) {
                        schedule_event(time+td,vcoslow,l4_0)
                        schedule_event(time+td,vcofast,l4_0)
                        examine = 1
                        }
                }

        else if (examine==1) {
                if ((down==l4_1) & (downold==l4_0) & (up==l4_1) & \
                        (upold==l4_0)) {
                        examine = 0
                        }
                else if ((up==l4_1) & (upold==l4_0)) {
```

```
                    schedule_event(time+td,vcoslow,l4_1)
                    examine = 0
                    }
                else if ((down==l4_1) & (downold==l4_0)) {
                    schedule_event(time+td,vcofast,l4_1)
                    examine = 0
                    }
                }

        }

}
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.                *
#*************************************************************************

# This circuit saves the previous five decisions of xpd2 as states q0-q4.

template xpd3 change zpr clk ki q0 q1 q2 q3 q4 vcofast vcoslow

{

or2_l4.x13 in2:vcoslow in1:vcofast out:nclk = tphl=500e-12, tplh=500e-12
or2_l4.x11 in2:change in1:zpr out:n3 = tphl=500e-12, tplh=500e-12
inv_l4.x12 out:n7 in:zpr = tphl=500e-12, tplh=500e-12
inv_l4.x9 out:n6 in:n5 = tplh=500e-12
nor2_l4.x7 in2:n4 in1:nclk out:n5 = tphl=500e-12, tplh=500e-12
nor2_l4.x8 in2:n5 in1:n3 out:n4 = tphl=500e-12, tplh=500e-12
set_l4_1.xhi set1:hi
xor2_l4.x6 out:n1 in2:q1 in1:ki = tphl=500e-12, tplh=500e-12
dff_l4.x10 qn:net5 clk:clk d:n6 s:hi r:n7 q:change = tp=500e-12
dff_l4.x5 qn:net4 clk:nclk d:q3 s:hi r:n7 q:q4 = tp=500e-12
dff_l4.x4 qn:net3 clk:nclk d:q2 s:hi r:n7 q:q3 = tp=500e-12
dff_l4.x3 qn:net2 clk:nclk d:n1 s:hi r:n7 q:q2 = tp=500e-12
dff_l4.x2 qn:net1 clk:nclk d:q0 s:hi r:n7 q:q1 = tp=500e-12
dff_l4.x1 qn:net0 clk:nclk d:vcofast s:hi r:n7 q:q0 = tp=500e-12

}
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.                *
#*************************************************************************

# This circuit inputs the five states of xpd3 and controls the fine word if cf is high.

template xpd4 zpr cf ki add sub add2 sub2 x y kx q0 q1 q2 q3 q4

{

set_l4_1.i15 set1:hi
dff_l4.x15 qn:n10 clk:add d:n10 s:hi r:kxz q:add2 = tp=500e-12
dff_l4.x14 qn:n11 clk:sub d:n11 s:hi r:kxz q:sub2 = tp=500e-12
inv_l4.x13 out:n7 in:cf = tphl=500e-12, tplh=500e-12
xor2_l4.x10 out:kx in2:q1 in1:q0 = tphl=500e-12, tplh=500e-12
or2_l4.x9 in2:n6 in1:n5 out:ki = tphl=500e-12, tplh=500e-12
and2_l4.x8 in2:cf in1:n6 out:add = tphl=500e-12, tplh=500e-12
and2_l4.x7 in2:n5 in1:cf out:sub = tphl=500e-12, tplh=500e-12
and2_l4.x6 in2:n4 in1:n3 out:n6 = tphl=500e-12, tplh=500e-12
nor2_l4.i17 in2:zpr in1:kx out:kxz = tphl=500e-12, tplh=500e-12
nor2_l4.x12 in2:q0 in1:n7 out:y = tphl=500e-12, tplh=500e-12
nor2_l4.x5 in2:n2 in1:n1 out:n5 = tphl=500e-12, tplh=500e-12
nor2_l4.x4 in2:q4 in1:q3 out:n4 = tphl=500e-12, tplh=500e-12
nor3_l4.x3 in3:q2 in2:q1 in1:q0 out:n3 = tphl=500e-12, tplh=500e-12
nand2_l4.x11 in2:q0 in1:cf out:x = tphl=500e-12, tplh=500e-12
nand2_l4.x2 in2:q4 in1:q3 out:n2 = tphl=500e-12, tplh=500e-12
nand3_l4.x1 in3:q2 in2:q1 in1:q0 out:n1 = tphl=500e-12, tplh=500e-12

}
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*************************************************************************

# This circuit acts as an accumulator on the rising edge of clk adding
# input to coarse when cf and reset is low.  When reset is high, coarse
# is set to the middle value of eight-bit word (128).  When cf is high
# and reset low, coarse is held constant.  The argument td is the delay
# associated from the rising clock to coarse being changed.

template xcoarse input cf reset clk coarse = td

state logic_4 cf, reset, clk
state nu input, coarse
number td=0.5n

{

state logic_4 clkold

when ((event_on(clk,clkold)) & (clk==l4_1) & (clkold==l4_0) & \
        (reset==l4_0) & (cf==l4_0)) {
        schedule_event(time+td,coarse,coarse + input)
        }

when ((event_on(reset)) & (reset==l4_1)) {
        schedule_event(time+td,coarse,128)
        }

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.   *
#*************************************************************************

# This circuit is four input, four output reset flip flop.  The d1-q1
# pair is a state of no units.  The argument td is the propagation delay
# of this flip flop.  When reset is high q1 is state 1 and the remaining
# outputs are all low.

template xreg d1 d2 d3 d4 clk reset q1 q2 q3 q4 = td

state logic_4 d2, d3, d4, q2, q3, q4, clk, reset
state nu d1, q1
number td = 0.5n

{

state logic_4 clkold
state nu sch_q1
state logic_4 sch_q2, sch_q3, sch_q4

when (event_on(reset) & (reset==l4_1)) {
        sch_q1 = 1
        sch_q2 = l4_0
        sch_q3 = l4_0
        sch_q4 = l4_0
        if (sch_q1 ~= q1) schedule_event(time,q1,sch_q1)
        if (sch_q2 ~= q2) schedule_event(time,q2,sch_q2)
        if (sch_q3 ~= q3) schedule_event(time,q3,sch_q3)
        if (sch_q4 ~= q4) schedule_event(time,q4,sch_q4)
        }

when (event_on(clk,clkold) & (reset==l4_0)) {
        if ((clk==l4_1) & (clkold==l4_0)) {
                sch_q1 = d1
                sch_q2 = d2
                sch_q3 = d3
                sch_q4 = d4
                if (sch_q1 ~= q1) schedule_event(time+td,q1,sch_q1)
                if (sch_q2 ~= q2) schedule_event(time+td,q2,sch_q2)
                if (sch_q3 ~= q3) schedule_event(time+td,q3,sch_q3)
                if (sch_q4 ~= q4) schedule_event(time+td,q4,sch_q4)
                }
        }
}
```

```
#*************************************************************************
#                Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*************************************************************************

# This circuit provides an output which is originally high, and then
# falls after a given time.  This is used to reset the logic at startup.

template xreset reset

state logic_4 reset

{

when (dc_init) {
        schedule_event(time,reset,l4_1)
        }

when (time_init) {
        schedule_event(time,reset,l4_1)
        schedule_event(time+10n,reset,l4_0)
        }

}
```

```
#*****************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.          *
#*****************************************************************

# This circuit is responsible for the almost zero phase restart and
# reset-zpr logic.

template xzpr zrclk stopz reset zpr refclk logic_reset logic_zpr

{

xreset.x7 reset:n2
set_l4_1.x6 set1:hi
inv_l4.x5 out:n1 in:zpr = tphl=500e-12, tplh=500e-12
and2_l4.x4 in2:stopz in1:refclk out:zrclk = tphl=500e-12, tplh=500e-12
dff_l4.x3 qn:n3 clk:refclk d:hi s:hi r:n1 q:stopz = tp=500e-12
or2_l4.x2 in2:logic_zpr in1:reset out:zpr = tphl=500e-12, tplh=500e-12
or2_l4.x1 in2:logic_reset in1:n2 out:reset = tphl=500e-12, tplh=500e-12

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.   *
#*************************************************************************

# This fine tuning circuit keeps track of the current value of fine
# (state n) and the value which is to be jumped to (state m). When a
# zpr occurs, m and n are set to the middle of a ten-bit word (512).
# When a load occurs, n is set to m. If neither of these two occur,
# then the fine tuning may add or subtract 1 from both m (add2 sub2)
# and n (add sub).

# If fine overflows or underflows, over of under will be asserted
# respectfully. The x and y work as the proportional phase correction.
# If both are high, fine increments by 1. If both are low, fine
# decrements by 1. This is not accumulative.

template xfine add sub add2 sub2 x y zpr load fine over under = td, sg

state logic_4 add, sub, add2, sub2, x, y, zpr, load
state nu fine
state logic_4 over, under
number td = 0.5n, sg = 25u

{

state logic_4 old_add, old_sub, old_add2, old_sub2
state nu m, n

when (dc_init) {
        schedule_event(time+td,fine,512)
        schedule_event(time+td,over,l4_0)
        schedule_event(time+td,under,l4_0)
        }

when (event_on(add,old_add) | event_on(sub,old_sub) | event_on(x) | \
        event_on(y) | event_on(load) | event_on(zpr) | \
        event_on(add2,old_add2) | event_on(sub2,old_sub2)) {
        if (zpr==l4_1) {
                n=512
                m=512
                }
        else if (load==l4_1) {
                n=m
                }
        else {
                if ((add==l4_1) & (old_add==l4_0)) {
```

```
                n=n+1
                }
        else if ((sub==l4_1) & (old_sub==l4_0)) {
                n=n-1
                }
        if ((add2==l4_1) & (old_add2==l4_0)) {
                m=m+1
                }
        else if ((sub2==l4_1) & (old_sub2==l4_0)) {
                m=m-1
                }
        if (n<0) {
                n=0
                schedule_event(time+td,under,l4_1)
                }
        else if (n>1023) {
                n=1023
                schedule_event(time+td,over,l4_1)
                }
        }
if ((x==l4_1) & (y==l4_1)) schedule_event(time+td,fine,n+1)
else if ((x==l4_0) & (y==l4_0)) schedule_event(time+td,fine,n-1)
else schedule_event(time+td,fine,n)
}

}
```

```
#*************************************************************************
#               Copyright 1996 John L Wallberg, Texas Instruments, Inc.   *
#*************************************************************************

# This circuit measures jitter on the input clk.  This clock must be a
# constant frequency determined by fr*n/m.  The argument fr is the value
# of the frequency of the refclock, and m and n are the values from the
# frequency dividers.

template xjitter clk jit = m,n,fr

state logic_4 clk
state nu jit
number m=3
number n=15
number fr=40meg

{

state nu p=0,x=0,ideal=0,period
state logic_4 old_clk

when (dc_init) {
        schedule_event(time,jit,0)
        }

when ((event_on(clk,old_clk)) & (clk==l4_1) & (old_clk==l4_0)) {
        if (fr==0) period=25n*m/n
        else period=m/n/fr
        if (p==0) {
                ideal=time
                p=1
                }
        else if (p>=1) {
                ideal=ideal+period
                x=time-ideal
                schedule_event(time,jit,x)
                p=2
                }
        }

}
```

Bibliography

[1] Roland E Best, *Phase-Locked Loops*. New York: McGraw-Hill, Inc. 1993.

[2] Raman Dakshinamurthy and Luke Long, "Good-bye PRML, hello DFE: Adaptive read channel debuts," Data Storage, March 1996, pp. 35-39.

[3] Jim Dunning, et al, "An All-Digital Phase-Locked Loop with 50-Cycle Lock Time Suitable for High-Performance Microprocessors," IEEE Journal of Solid-State Circuits, Vol. 30, No. 4, April 1995, pp. 412-422.