From: Phase-Locking in High-Performance Systems, IEEE Press, 2003

Delay-Locked Loops - An Overview

Chih-Kong Ken Yang

Abstract — **Phase-locked loops have been used for a wide range of applications from synthesizing a desired phase or frequency to recovering the phase and frequency of an input signal. Delay-locked loops (DLLs) have emerged as a viable alternative to the traditional oscillator-based phase-locked loops. With its first-order loop characteristic, a DLL both is easier to stabilize and has no jitter accumulation. The paper describes design considerations and techniques to achieve high performance in a wide range of applications. Issues such as avoiding false lock, maintaining 50% clock duty cycle, building unlimited phase range for frequency synthesis, and multiplying the reference frequency are discussed.**

I. Introduction

Many applications require accurate placement of the phase of a clock or data signal. Although simply delaying the signal could shift the phase, the phase shift is not robust to variations in processing, voltage, or temperature. For more precise control, designers incorporate the phase shift into a feedback loop that locks the output phase with an input reference signal that indicates the desired phase shift. In essence, the loop is identical to a phase-locked loop (PLL) except that phase is the only state variable and that a variable-delay line replaces the oscillator. Such a loop is commonly referred to as a delay-line phase-locked loop or delay-locked loop (DLL). As with a PLL, the goals are (1) accurate phase position or low static-phase offset, and (2) low phase noise or jitter.

Because a DLL does not contain an element of variable frequency, it historically has fewer applications than PLLs. Bazes in [1] demonstrated an example of precisely delaying a signal in generating the timing of the row and column access strobe signals for a DRAM. Another common application uses a DLL to generate a buffered clock that has the same phase as a weakly-driven input clock. Johnson in [2] synchronizes the timing of the buffered clock of a floating-point unit with the clock of a microprocessor. A similar application recovers the data of a parallel bus by generating a properly positioned sampling clock. Typically, these systems provide a sampling clock with the same sampling rate but with an arbitrary phase as compared to the data (i.e. a "mesochronous" system [4]). A clocked DRAM data bus is an example of such a system. A clock propagates with the data as one of the signals in the bus and therefore has a nominally known phase relationship with the data. However, in order to receive and buffer the clock to sample the data bus, the actual sampling clock is no longer properly aligned with the data. A DLL is commonly used to lock the phase of the buffered clock to that of the input data. The phase locking significantly reduces timing uncertainty in sampling the data, which then enables higher data rates as in [3].

Although aperiodic signals can also be delayed by the delay line in a DLL, the inputs to delay lines are typically clock signals. By using a periodic signal, the delay lines do not need arbitrarily long delays and typically only need to span the period of the clock to generate all possible phases. A data signal can be delayed by sampling the data with the appropriately delayed clock.

The motivation for using DLLs is that the design of the control loop is simplified by having only phase as the state variable. Section II reviews how such a loop is unconditionally stable and has better jitter characteristics. However, a DLL is not without its own limitations. The variable delay line has a finite delay range and finite bandwidth. Section II also discusses these design considerations. Section III describes different implementations of the variable delay line. Within the past ten years, modifications to the basic DLL architecture have enabled clock and data recovery applications in "plesiochronous" systems [4] where the sampling rates for clock and data differ by a few hundred parts-per-million in frequency. Delay lines with effectively infinite delay are also addressed in Section III.

More recently, several researchers such as [5] and [6] have introduced architectures that permit frequency multiplication based on delay lines which further extends their use in clock generation and frequency synthesis. Section IV describes these architectures.

II. DLL Characteristics

The basic loop building blocks are similar to that of a PLL: a phase detector, a filter, and a variable-delay line. Figure 1 illustrates the three main functional blocks. Since phase is the only state variable, a control loop higher than first-order is not needed to compensate a fixed phase error. The resulting transient impulse response is a simple exponential. Although the simple loop characteristics are an advantage that DLLs have over PLLs, the design is complicated by the additional circuitry that is needed to overcome having a limited delay range and not producing its own frequency.

A. First-order Loop

A phase detector compares the phase of the reference input and the delay-line output. The comparison yields a signal proportional to the phase error. The error is low-pass

C.K. Ken Yang is with University of California at Los Angeles, yang@ee.ucla.edu.

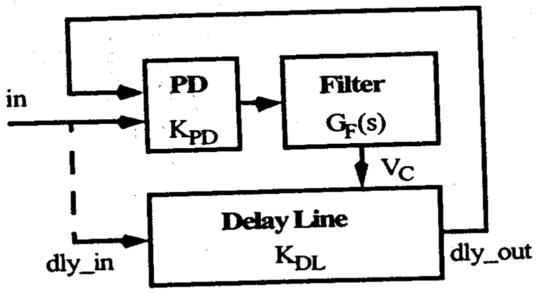


Figure 1: DLL architecture.

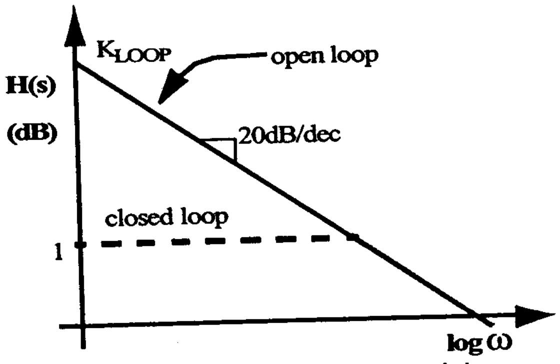


Figure 2: Open- and closed-loop transfer characteristics.

filtered to produce a control voltage or current that adjusts the delay of the delay line. The delay-line input can be either the reference input or a clean clock signal.

The s-domain representation of each loop element is depicted within each block in Fig. 1. The open-loop transfer function can be written as $T(s) = K_{PD}K_{DL}G_F(s)$ where K_{PD} is the phase-detector gain, $G_F(s)$ is the filter transfer function, and K_{DL} is the delay-line gain. If the loop has finite gain at dc, the resulting output signal will exhibit a static phase error as shown in the following equation.

$$H(s)\big|_{s=0} = \left.\frac{1}{1 + 1/(K_{PD}K_{DL}G_F(s))}\right|_{s=0} \qquad (1)$$

To eliminate the static phase error, the filter is often an integrator to store the phase variable. This results in a first-order closed-loop transfer function.

$$H(s) = \frac{1}{1 + (s/K_{PD}K_{DL}G_F)} \qquad (2)$$

The equation assumes that the delay-line input is a clean reference as opposed to the reference input. Higher-order loop filters have not commonly been used but can enable better tracking of a phase ramp (i.e. a frequency difference).

Figure 2 shows the open-loop and closed-loop transfer functions. With only a single integrator, the open-loop phase margin is 90°. The loop is unconditionally stable as long as the delay in the loop does not degrade the phase margin excessively. The closed-loop transfer function illustrates that

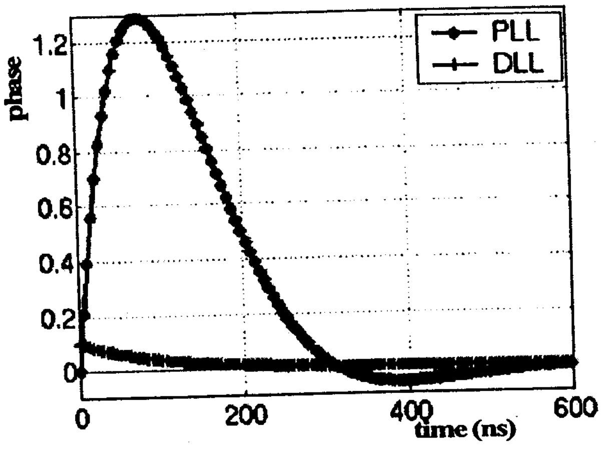


Figure 3: Step response of PLL and DLL (with same loop characteristics).

the tracking of the phase of the input clock changes at different frequencies. Based on the transfer function, the loop bandwidth is $\omega_{bw} = K_{PD}K_{DL}G_F$. For frequencies within the loop bandwidth the phase of the output clock will track that of the reference input and reject noise within the loop. The phase characteristics of the output clock above the bandwidth of the loop depend on the phase behavior of the delay-line input and the noise from the delay line. The noise transfer function from a noise source lumped at the delay-line output is a high-pass response.

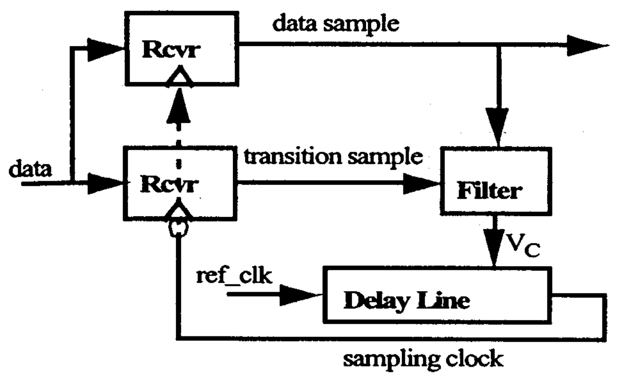
$$H(s) = \frac{(s/K_{PD}K_{DL}G_F)}{1 + (s/K_{PD}K_{DL}G_F)} \qquad (3)$$

In some degenerate cases, the delay-line input is also the reference input. The feedback loop would guarantee a fixed phase relationship between the delay-line output and the reference so any phase variations in the reference would directly appear at the delay-line output in an all-pass response. However, noise due to the delay line is still high-pass filtered.

B. Advantages over a PLL

The loop characteristics are considerably simpler than those of a PLL. A PLL would contain at least two states to store both the frequency and phase information. In order to maintain loop stability, an additional zero is needed. A DLL is less constrained with only a single pole. The loop gain directly determines the desired bandwidth. The only stability consideration is when the loop bandwidth is very near the reference frequency. The periodic sampling nature of the phase detection and the delay in the feedback loop degrade the phase margin. For instance, if the feedback delay is one reference cycle, the loop bandwidth should not exceed 1/4 of the reference frequency.

Figure 3 illustrates the response to a noise step applied to the control voltage for both a PLL and a DLL. A PLL accumulates phase error due to its higher-order loop characteristic. In response to a phase error, the control



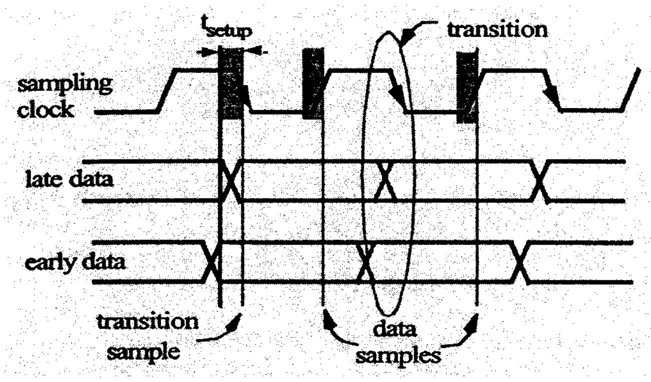


Figure 4: Early-late receiver architecture using the receiver as the phase detector. Timing diagram showing early and late data.

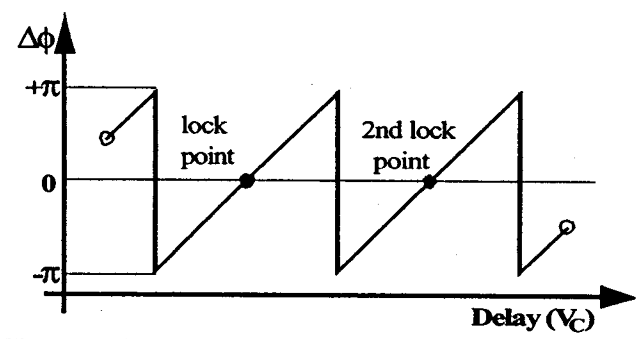


Figure 5: Delay line phase/delay characteristic.

voltage alters the frequency of an oscillator. The output phase is an integration of the frequency change. In response to a noise perturbation, the loop accumulates a phase error before correcting. In contrast, a DLL attenuates the phase error by the time constant of the loop. In the figure, both loops are designed with the same 3-dB bandwidth, the same delay elements, and the PLL is a 2nd-order loop with a damping factor of unity. Clearly, the PLL suffers from larger phase errors due to the phase accumulation.

A second advantage relates to clock and data recovery applications. An effective way to recover the timing for sampling a data input is to use the data receiver as a phase detector. The architecture, depicted in Fig. 4, uses the 180°-shifted clock to sample the data transitions in addition to sampling the data values [7]. Whenever data changes values, the sampled transition and the data values can be combined to indicate whether the sampling clock edge is earlier or later than the data transition. Phase information is only present with data transitions. The feedback loop locks when the transition sampling clock samples a metastable value. This commonly used design is known as an early-late or bang-bang architecture. The timing diagram in Fig. 4 illustrates examples of the data being early and late. Due to the inherent setup time of the data receiver, the transition sampling clock may not occur at the same time as the data transition. The phase shift compensates for the receiver setup time and maximizes the margin of error for the data sampling.

However, the data receiver is ultimately a binary comparator and the phase detector does not indicate an error that is proportional to the phase difference. Hence, the timing-recovery loop is nonlinear. Although a higher-order PLL using early-late control can be made conditionally stable [8], the resulting phase dithers with a limit cycle translating into jitter. The oscillation depends on the loop parameters and can be considerable for high bandwidth loops. With an early-late DLL, the phase of the clock output also dithers. But because the stability only depends on the delay within the loop, the dithering would only be a few cycles and can be significantly less than the dithering of a PLL.

C. Design Considerations in a DLL

A typical DLL involves several design considerations. First, the delay line usually has a finite delay range. If the desired phase of the output signal is beyond the delay range, the loop will not lock properly. Second, the output of the DLL also depends greatly on the input to the delay line. Since the delay-line input propagates to the DLL output, tracking jitter and the output's duty cycle depend not only on the delay-line design but also on the delay-line input. Third, the basic DLL cannot generate new frequencies different from that of the delay-line input.

A variable-delay line adjusts the delay by varying the RC time constant of a buffer and often has limited adjustment range. Section III will describe several techniques in greater detail. Even though the delay range is limited, DLLs for a periodic clock signal only need the range to exceed 2π in phase across process and systematic variations to cover all possible phases. For systems with a range of operating frequencies, the delay line must span 2π for the lowest input frequency.

An issue known as false-locking occurs when the delay range exceeds 2π. There can be several secondary lock points repeating every 2π. Figure 5 depicts an example of the characteristic of a delay line with two lock points. Since phase detectors must be periodic, if the delay line initializes within π of the second lock point, the phase detector will push the delay line toward lock with a longer than necessary delay. Long delays require large RC time constants for a given variable-delay buffer element. The bandlimiting by the

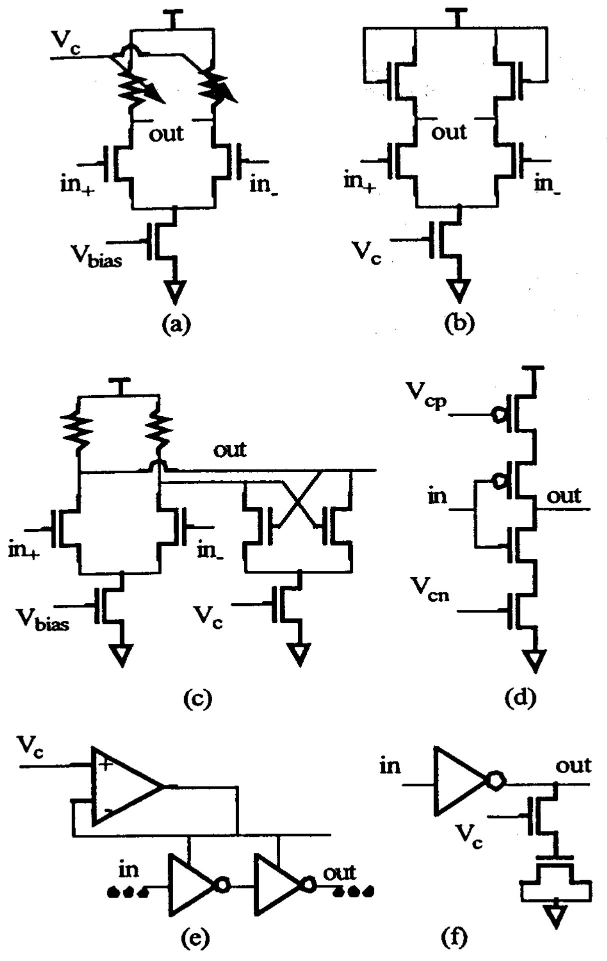


Figure 7: Six different delay elements.

A. Basic Delay Line

A delay line comprises of a chain of variable-delay elements. Each element is controllable by either a voltage or a current. The delay of each element is proportional to its RC time constant and changing the effective resistance or capacitance adjusts the delay.

Figure 7 depicts several examples of buffer elements. For a differential buffer, the load resistance can be an MOS transistor in the triode region [Fig. 7-(a)] where the resistance is proportional to V_{GS}-V_{th}. Varying the gate voltage adjusts the delay of the element. A non-linear device such as a diode can also serve as a load resistance [Fig. 7-(b)]. Since the resistance varies with the current, varying the bias current of the buffer would adjust the delay. Similarly, a negative transconductance that changes with the bias current can be placed in parallel with a fixed load resistance [Fig. 7-(c)]. The varying negative transconductance changes the effective load resistance and hence varies the delay. Because nonlinear elements have resistances that depend on both voltage and current, they can be more sensitive to supply noise.

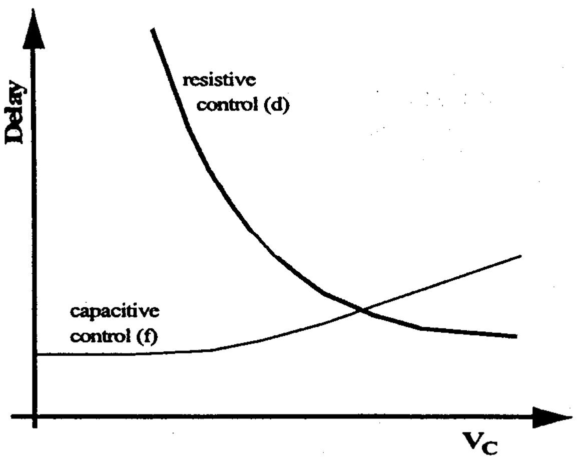


Figure 8: Delay versus voltage for two different delay buffer elements: types (d) and (f) of Fig. 7.

For push-pull type elements such as inverters, the delay can be changed by changing the rate at which the output capacitance is charged [Fig. 7-(d)]. An adjustable current source limits the peak current of an inverter and varies the delay. An alternative method regulates the supply voltage of the inverters and uses the control voltage to set the supply voltage [Fig. 7-(e)]. The effective switching resistance varies with the supply voltage. Instead of changing the resistance, the effective capacitance can also be made adjustable [Fig. 7-(f)]. A transistor that behaves as an adjustable resistance can be used to decouple an explicit output capacitance. The larger the resistance the less capacitance is seen at the output.

Figure 8 illustrates the delay versus control voltage for a resistively-controlled delay element. For the element of Fig. 7-(d), either V_{GS}-V_{th} or the bias current can be zero and, therefore, a single element's delay can span from the minimum buffer delay to infinite. However, since the time constant is proportional to the delay, a long delay setting would significantly attenuate a high-frequency clock. Delay lines with a wide range for high clock frequencies require a large number of broadband delay elements.

Unlike resistive control, the maximum delay in a capacitively-controlled element [Fig. 7-(f)] is proportional to $R(C_{int}+C_{exp})$ and the minimum delay is proportional to RC_{int} where C_{int} is the intrinsic capacitance of the buffer and the load of the subsequent stage, and C_{exp} is the explicit capacitance added to the circuit. Because of the limited range per buffer, obtaining a wide delay range involves a large number of buffers. The maximum delay of each buffer is chosen to avoid attenuating the signal. In designs where the clock has a large voltage swing, the transistor in series with the explicit capacitance no longer appears as a variable resistor because the device enters saturation and cut-off. For these buffers, the control voltage determines the fraction of current and period of time in which the buffer's current charges the explicit capacitance.

An example of the delay versus control voltage for a capacitively-controlled element is overlaid in Fig. 8. Most filter would significantly attenuate a high-frequency input clock. The attenuation increases the jitter and may even prohibit the input from reaching the output.

Even if the delay line is constrained to span only one lock point but greater than 2π, a second similar issue exists. It is difficult to design a delay line such that the adjustable range is exactly $-\pi$ to $+\pi$ across different operating and processing conditions. If initialized at the minimum or maximum delay, the phase detector may push the loop toward either the maximum or minimum delay limit and "false-lock" to an incorrect phase.

To address false-locking, designers employ several techniques depending on the application. For systems that require a delay line with a known fixed delay, operating condition variations may be small enough such that the delay line only needs a small variable range that is less than $+\pi$ and $-\pi$. For systems that lock to a fixed phase over a wide range of frequencies, one design [9] uses an auxiliary frequency-sensing loop that generates a voltage to coarsely set the delay for the given input frequency. Then DLL only fine tunes the delay for the desired phase. For data recovery applications where the clock phase can be arbitrary with respect to the data, a common design uses a startup circuit for the DLL that initializes the delay line at its minimum delay to avoid any secondary lock points. However, as mentioned earlier, the phase detector may keep the delay line at the minimum delay. A sensing circuit or a state machine detects when the delay line is at its limit and optionally inverts the feedback clock. The phase would flip by 180° and the loop would lock properly. As will be discussed in Section III, a more robust alternative reconfigures the delay line such that the delay only spans 2π and wraps back to 0° when the delay exceeds 360°.

The jitter and duty cycle of the delay-line output clock depend on the input, the coupling of the input to the delay line, and the delay line itself. Often the input is from off-chip and, therefore, it must be carefully received to prevent supply and substrate noise from coupling onto the signal as jitter. In contrast, the high-frequency phase noise of the clock output of an oscillator-based PLL depends primarily on the oscillator design. An improperly received input clock can often result in worse jitter performance in a DLL as compared to a PLL. Similarly, while the duty cycle from an oscillator is only modestly distorted (by the difference between the rising edge and falling edge delays), the duty cycle of the DLL's input clock can be significantly distorted as it propagates to the output. Since duty cycle is a systematic error, a good design corrects duty cycle using an explicit block instead of compounding the difficulty of the delay-line design.

A duty-cycle corrector (DCC) is commonly added to either the DLL input or output. Figure 6 illustrates the basic components of the feedback loop: an input with finite slew rate, a buffer element with adjustable threshold, a comparator, and an integrator. The comparator determines the threshold crossing of the clock waveform. The result is integrated and used to skew the threshold of the buffer stage.

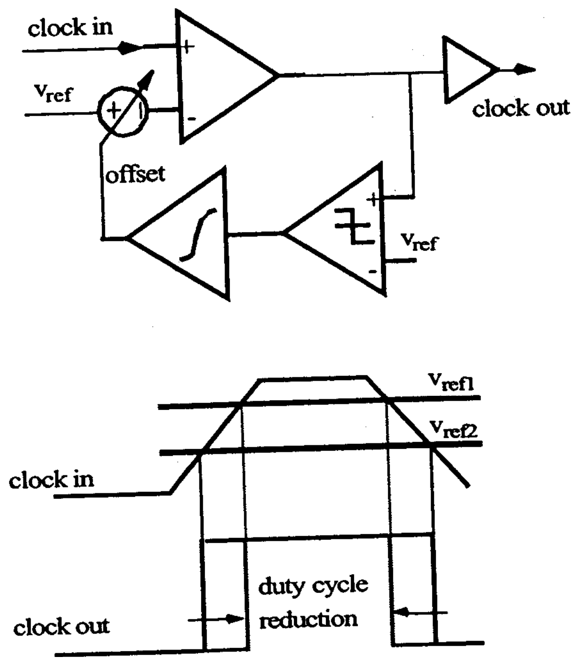


Figure 6: Duty-cycle corrector block diagram. Timing diagram shows change in duty cycle with changing offset.

Because the buffer input has finite slew rate, changing the threshold effectively adjusts the output high and low half-periods. The loop settles when the high and low half-periods are equal. Figure 6 illustrates the reduction in duty cycle as the threshold shifts from V_{ref1} to V_{ref2}. Since random variation of the duty cycle effectively appears as jitter, single-ended implementations such as that shown in the figure can be very sensitive to common-mode noise. For this reason, differential architectures are preferred [3].

For low jitter on the output clock, the loop components must be carefully designed. Many of the loop components are very similar to that of a PLL and are well described in [10]. For a charge-pump based loop filter, since the filter is only first-order, a simple capacitor replaces the RC filter. As in a PLL, noise on the control voltage directly translates into jitter. Designers may use additional filtering to suppress the noise. The loop element that has deviated the most from PLL design and is critical for functionality and performance is the design of the delay line.

III. Delay-Line Architectures

The primary characteristics of a delay line are (1) gain (i.e. change in delay for a given change in voltage), and (2) delay range. For most applications using periodic inputs, the absolute delay is not critical as long as the range spans 2π. Because delay lines are relatively short, they do not contribute significant thermal or 1/f phase noise. However, for large digital systems, low supply/substrate sensitivity is needed to reject the on-chip switching noise.

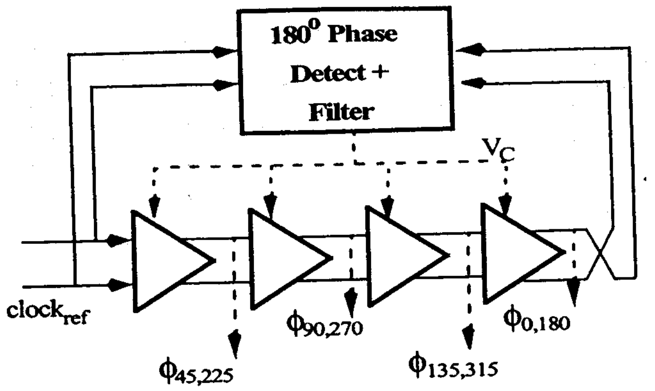


Figure 9: 180°-locked DLL to generate intermediate phases that are a fraction of a cycle.

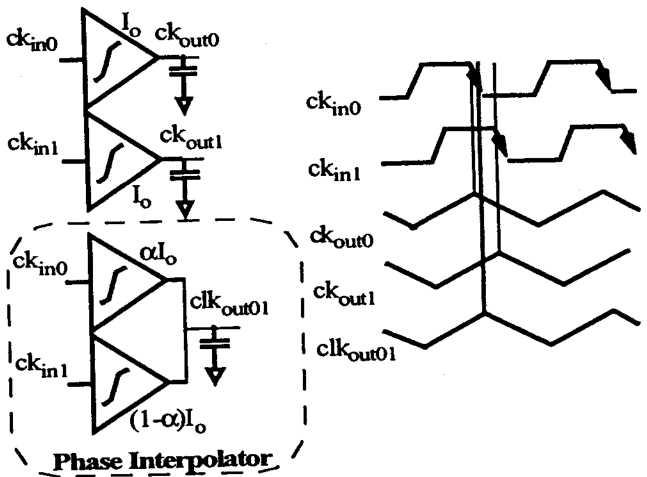


Figure 10: Phase interpolator design by shorting of the output of two integrators/buffers. .

delay elements exhibit some nonlinearity. As a result, the delay-line gain, K_{DL}, is a function of the delay. Because a DLL is unconditionally stable, the loop still functions with the varying loop parameter. However, more linear elements are better for designs that require a constant loop bandwidth. To compensate for the variable K_{DL}, designers add programmability to the loop-filter capacitor.

The control signal for either type of delay elements can be digital. In a digital implementation [11], the current source is binary weighted and switched by a digital word. For capacitively-controlled elements, the capacitance can be binary weighted and switched. A nearly all-digital DLL is then possible by using a simple counter to replace the analog integrating filter.

B. *Phase Interpolation*

Instead of only using the clock phase at the end of a delay line, an earlier clock phase can be tapped from the middle of a delay line. Some applications require the delay line to produce a delay that is a fixed *fraction* of the input-clock period. Figure 9 shows one implementation that uses a DLL to lock the input clock to the output. An 180° phase detector would guarantee the absolute delay of a delay line to be a half-cycle. Tapping from different points on the delay line provides different phases. As shown in Fig. 9, for a 45° phase shift, the clock can be tapped from the first delay stage of a 4-stage differential delay line. If an arbitrary phase is needed, each delay stage can be tapped and multiplexers can select the nearest desired phase. The number of delay elements quantizes the phase step and limits the resolution [12]. Fine phase resolution requires longer delay lines. Yet, the resolution is limited at high clock frequencies because the maximum number of delay elements needed to span 180° is limited.

An arbitrary intermediate phase can be obtained by "interpolating" between two clock phases that are tapped from a delay line. Depending on the weighting, an interpolator produces a clock that has a programmable output phase in between the input clock phases. As long as discrete clock phases that span the entire cycle are available as inputs, any phase for the interpolator's output is possible. Multiplexers are needed to select the phases to interpolate between. For example, with phases tapped from a 4-stage delay line, if the desired output clock phase is 120°, the interpolator inputs would be from the second and third delay elements.

Interpolators essentially perform a weighted average of the input phases. As shown in Fig. 10, ideally, the two input phases drive two integrators which charge a single output. The weighting of the average is by the relative currents of the two integrators. When $\alpha=1$, the output clock phase depends only on ck_{in0}. When $\alpha=0.5$, i.e. the current is split equally between the two integrators, the output phase is additionally delayed by half the phase difference. As illustrated in Fig. 10, the phase of the interpolated output (ck_{out01}) falls between the phases of the non-interpolated outputs (ck_{out0} and ck_{out1}).

With ideal integrators, the interpolation is linear, resulting in a constant K_{DL}. Alternatively, an interpolator can effectively be formed with buffer elements instead of integrators. By weighting the drive strength or current of two buffer elements whose outputs are shorted together, one can adjust the output phase. Because the output is not integrated, the resulting interpolation is slightly nonlinear and depends on (1) the phase difference between the inputs and (2) the slew rate (or time constant) of the input and output signals [13]. Figure 11 depicts the linearity of the interpolation for two different input phase separations, $s=\tau$ and $s=2\tau$ where τ is the buffer's time constant. The larger phase spacing results in greater nonlinearity. Similar to RC delay elements, the interpolation can be digitally controlled. Since the weighting of the interpolation depends on the proportional current, the current sources of the integrators or buffers can be digitally weighted and programmed.

In a design for clock and data recovery by [3], quadrature clocks are interpolated to generate an intermediate clock phase within a quadrant. Figure 12 illustrates the mostly analog architecture. An analog control

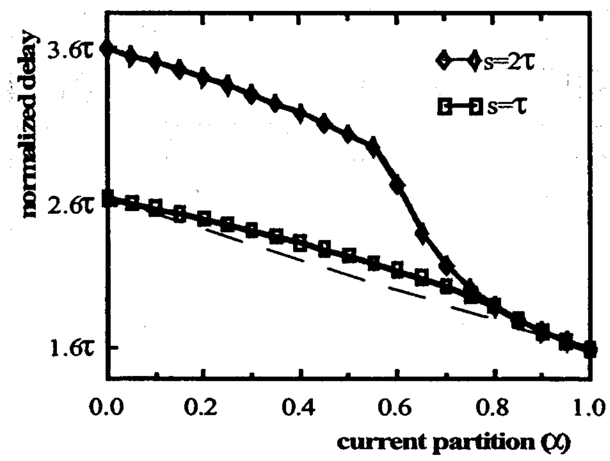


Figure 11: Buffer based phase interpolator linearity.

voltage produced by the phase detector and filter determines the interpolator currents. Comparators indicate when the current is fully steered to one integrator. A finite state machine driven by the comparators selects the appropriate quadrant by switching the interpolator inputs such that all 360° phases are possible. The quadrature input clocks is generated from an external reference clock through the use of a divide-by-two circuit.

Interestingly, because the phase rotates from one quadrant to the next, the architecture effectively has an unlimited delay range. If the input data rate and the reference clock frequency are slightly different, a DLL would continually increase or decrease the delay in order to track the accumulating input phase. A typical DLL with a finite delay range would run out of delay or lose lock. On the other hand, plesiochronous operation is possible with an interpolator-based delay line since the phase smoothly rotates between quadrants.

Interpolating between clocks with large phase spacings such as quadrature clocks results in an output clock with slow slew rate. Such waveforms are more susceptible to noise and result in higher jitter. An enhancement uses more closely-spaced phases that span the cycle. The finer phases spacing is possible using a multi-stage ring oscillator. As shown in Fig. 13, a 4-stage differential oscillator would generate 8 phases 45° apart. To guarantee a correct period for each clock phase, the ring oscillator is locked to the external reference clock using a PLL. The role of the PLL is solely for generating the phases. A purely DLL-based architecture is also possible by replacing by using the DLL in Fig. 9 that locks the delay-line output with a 180° phase shift [13].

The architecture is commonly known as a dual-loop design because the first loop, a PLL or DLL, generates the phases and the second loop, the interpolation-based DLL, recovers the data and phase. Since the first loop is not in the feedback of the second loop (or vice versa), the overall system is stable as long as each loop is individually stable. A dual-loop design is possible with the second loop within the

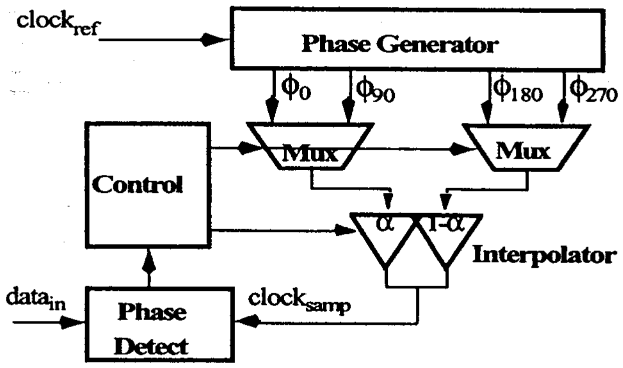


Figure 12: Infinite-range delay line based on phase rotation.

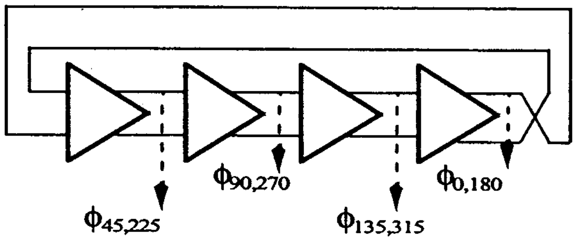


Figure 13: Oscillator with tapped outputs for multiple phases.

feedback of the first loop [15] as long as the stability of the loop is carefully considered.

The data recovery portion of a dual-loop design is conducive to a digital implementation. The binary output of the receiver-replica phase detector can be accumulated using a digital counter. The counter output selects the appropriate phase from the oscillator and controls the digitally programmable interpolators [13],[14]. As long as the quantized phase step is small, the small error only minimally impacts the data recovery.

C. *Oversampled Implementation*

An alternative purely digital approach to clock and data recovery can be implemented by oversampling the data. Figure 14 illustrates an example of a digital architecture. Multiple finely-spaced clock phases oversample the data input. The sampled results are digitally processed to determine both the correct data value and the optimal phase of the data sample. The digital processing can vary in complexity. Simple implementations use the optimal data sample as the received data [18] or take a majority vote from the samples of a single bit [17]. The bit boundaries determine the samples associated with a bit. Transitions that are detected in the samples from the prior or current bits indicate the bit boundaries.

The sampling rate limits the timing error margin. Greater amount of oversampling reduces the data-recovery timing error, but increases the number of clock phases. Low data rate UARTs [16] typically use 8 to 16 times oversampling. For high data rates, generating accurate clock phases separated by sub-100ps is very challenging. More

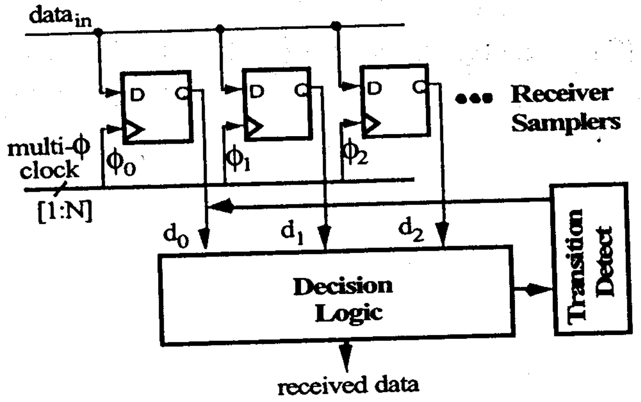


Figure 14: Oversampled data recovery architecture.

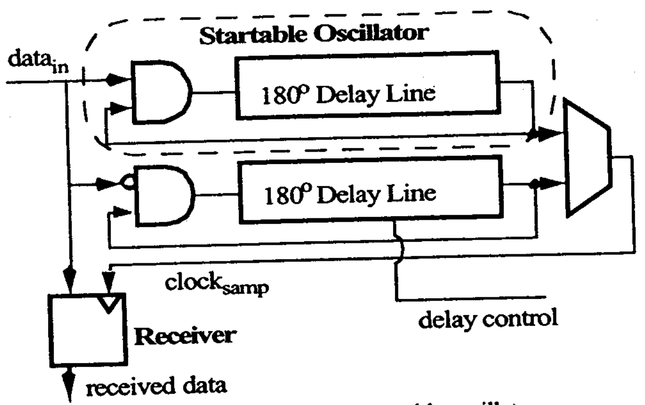


Figure 15: Clock/data recovery using startable oscillator.

aggressive designs with the least amount of clocking overhead and high data rates use a minimum of 3x oversampling [17], [18].

Even though phase spacing scales with the gate delay of a technology, so does the bit time in each generation of applications. For oversampling of the data bits, finely-spaced clock phases are needed. Tapping from a delay line produces phases separated by a buffer delay. For even finer phases, several techniques are commonly used. For example, several interpolators can be used where each interpolator has slightly different weighting to generate intermediate phases with spacing less than a buffer delay [19]. An alternative method uses a chain or array of coupled oscillators [20]. By taking a chain of oscillators and coupling them such that the output and input of the chain are separated by only one gate delay, sub-gate-delay phase spacings result from the outputs of each oscillator. Lastly, if the data can be delayed with a chain of delay buffers along with the clock, the clock at each delay stage can be used to sample the data of the corresponding stage. As long as the data and the clock delay lines have slightly different delays, the sub-sampled outputs are effectively an oversampling of the data. The effective phase spacing depends only on the difference between the data delay and clock delay [21]. The architecture has a drawback in that it requires delaying the data and clock by long delays of several cycles, which can significantly increase jitter.

IV. Clock Multiplication

With a dual-loop architecture, a DLL can produce a frequency plesiochronous to the delay-line input. However, the rate at which the interpolator weight changes limits the frequency difference. Generating a significantly different or multiplied frequency from a low-frequency input reference is not possible with the architecture.

Recently designers have explored several methods of using DLLs for frequency multiplication. One method uses a delay line that is locked to 180°. With the phases that span an entire cycle, the tapped clock edges are combined to form a clock with multiplied frequency. The most direct method uses logical AND-ORs to combine the multiple phased clocks into a single high-frequency clock [23]. Alternatively, the method in [24] converts each phase into a small pulse and ORs the pulses together to form the output clock. In cases where the output capacitance of the logic gates limits the output frequency, one design [6] uses phases to excite a tuned LC tank to combine the clock phases.

Instead of edge combining, the multiplied clock can be the direct output of a delay line. The architecture is similar to a technique for clock and data recovery that uses a startable oscillator [22]. As shown in Fig. 15, the architecture uses data transitions to trigger startable oscillators: high-value data triggers one oscillator and low-value data triggers another. Each startable oscillator comprises of a delay line and an AND gate. The data value enables the AND gate and the triggered oscillator propagates an edge through the delay elements and produces a clock edge delayed by a half-cycle. The edge is used to sample the data. In the absence of input transitions, the delay line is configured an oscillator and generates a sampling edge every cycle. Whenever a new data transition occurs, the oscillator resynchronizes its phase to that of the input. In the implementation by [22], the natural oscillation frequency of the oscillator is determined by an external plesiochronous clock reference. The architecture has not been widely applied to higher data rate designs because the sampling phase is directly derived from the input data without any filtering. The deterministic and random jitter inherent in the data are effectively doubled and can be considerable.

If the input is a low-jitter reference clock, a similar architecture can be used for clock multiplication [5]. As illustrated in Fig. 16, a lower frequency but clean reference clock is one input to a multiplexer that feeds into a delay line. The output of the delay line is fed back to the multiplexer as the second input. When a reference clock edge is available, the multiplexer selects the reference input. Otherwise, the multiplexer configures the delay line as an oscillator with the output frequency controlled by the delay. The multiplexer inputs are selected by a counter circuit that determines the number of cycles to oscillate before accepting the next reference clock edge. A phase detector compares the

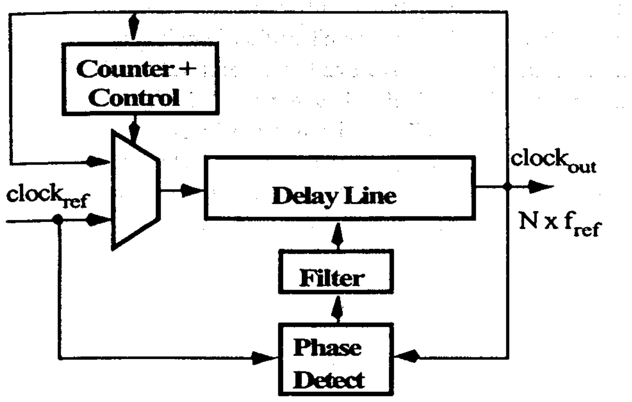


Figure 16: DLL-based clock multiplication.

reference input with the oscillator output and tunes the delay of the delay elements. Once locked, the resulting output clock frequency is a multiple of the input reference frequency. Recent designs [26] extend the frequency range and use an interpolator instead of a multiplexer to blend the delay-line feedback and the low-frequency reference clocks.

Both edge-combining multiplication and delay-line multiplication reduce the phase noise of the output clock because the core DLL does not have an oscillator that accumulates phase error. After N cycles, where N is the divide ratio, a new clean reference clock edge arrives and resets any accumulated phase error to zero. The architecture potentially lowers jitter by eliminating the peaking in the transfer function and allows a high tracking bandwidth. However, matching is critical in these designs. Mismatches in the phase detector or charge pump result in a static phase error that modulates the output frequency at the input reference frequency. Similarly, in the edge combining implementations, if the delay line is mismatched, the output clock would contain significant reference tones. Designers either choose the reference frequency carefully so that the tones do not impact the system performance or employ additional circuitry to compensate for the mismatches.

V. Conclusion

DLLs have been commonly used for generating precise phase delays of a signal and have been increasingly popular in clock generation and data recovery applications. Most importantly, because of the first-order loop characteristics that controls the phase directly, DLLs can be designed with high tracking bandwidths and do not exhibit the phase accumulation of an oscillator-based PLL.

The more simple loop characteristics belie many subtleties in DLL design. The delay-line input clock must have low-jitter and good duty-cycle. Furthermore, it must be carefully received and coupled to the input of the delay line to maintain good jitter performance. This source of jitter counter-balances the jitter accumulation of PLLs and results in less jitter improvement. Additional circuitry is often needed to prevent false-locking. Since a delay line does not restore a clock's duty cycle, the output clock requires correction circuitry. To use DLLs in plesiochronous systems, the delay line must have even more circuitry to achieve an unlimited delay range. In clock multiplication applications, very careful matching in the DLL components is critical to eliminate reference tones. In the many designs that have addressed these subtleties, DLLs have demonstrated low-jitter clock outputs for a variety of clock generation and data recovery applications.

References

[1] Bazes, M., "A Novel Precision MOS Synchronous Delay Line," *IEEE Journal of Solid-State Circuits*, vol sc-20, no 6, Dec. 1985, pp. 1265-71

[2] Johnson, M.G., E.L. Hudson, "A Variable Delay Line PLL for CPU-Coprocessor Synchronization," *IEEE Journal of Solid-State Circuits*, vol 23, no 5, Oct. 1988, pp. 1218-23

[3] Lee, T.H., et. al., "A 2.5V CMOS Delay-Locked Loop for an 18 Mbit, 500Megabytes/s DRAM," *IEEE Journal of Solid-State Circuits*, vol 29, no 12, Dec. 1994, pp. 1491-6

[4] Messerschmitt, D.G., "Synchronization in Digital System Design," *IEEE Journal on Selected Areas in Communications*, Oct. 1990, pp. 1404-1420

[5] Waizman, A., "A Delay Line Loop for Frequency Synthesis of De-Skewed Clock," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 1994, San Francisco, Session 18.5

[6] Chien, G., P.R. Gray, "A 900-MHz Local Oscillator Using a DLL-Based Frequency Multiplier Technique for PCS Applications," *IEEE Journal of Solid-State Circuits*, vol 35, no 12, Dec. 2000, pp. 1996-9

[7] Alexander, J.D., "Clock Recovery from Random Binary Data," *Electronic Letters*, vol 11, Oct. 1975, pp 541-2

[8] D'Andrea, N.A., F. Russo, "A binary quantized digital phase locked loop: a graphical analysis," *IEEE Transactions on Communications*, vol.COM-26, (no.9), Sept. 1978. p.1355-64

[9] Moon, Y., "An All-Analog Multiphase Delay-Locked Loop Using A Replica Delay Line for Wide-Range Operation and Low-Jitter Performance," *IEEE Journal of Solid-State Circuits*, vol 35, no 3, Mar. 2000, pp. 377-84

[10] Razavi, B., "Design of Monolithic Phase-Locked Loops and Clock Recovery Circuits - A Tutorial," *Monolithic Phase-locked Loops and Clock Recovery Circuits*, IEEE Press 1996 New Jersey, pp. 1-28

[11] Dunning, J., 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, Apr. 1995, pp. 412-22

[12] Efendovich, A., et. al., "Multifrequency Zero-Jitter Delay-Locked Loop," *IEEE Journal of Solid-State Circuits*, vol 29, no 1, Jan. 1994, pp. 67-70

[13] Sidiropoulos, S., M.A. Horowitz, "A Semidigital Dual Delay-Locked Loop," *IEEE Journal of Solid-State Circuits*, vol 32, no 11, Nov. 1997, pp. 1683-92

[14] Garlepp, B., et. al., "A Portable Digital DLL for High-Speed CMOS Interface Circuits," *IEEE Journal of Solid-State Circuits*, vol 34, no 5, May 1996, pp. 632-44

[15] Larsson, P., "A 2-1600-MHz CMOS Clock Recovery PLL with Low-Vdd Capability," *IEEE Journal of*

Solid-State Circuits, vol 34, no 12, Dec. 1999, pp. 1951-60

[16] Cordell, R., "A 45-Mbit/s CMOS VLSI Digital Phase Aligner," *IEEE Journal of Solid-State Circuits*, vol 23, no 2, Apr. 1988, pp. 323-28

[17] Lee, K., et. al., "A CMOS Serial Link For Fully Duplexed Data Communication," *IEEE Journal of Solid-State Circuits*, vol 30, no 4, Apr. 1995, pp. 353-64

[18] Yang, C.K., et al., "A 0.5-µm CMOS 4.0-Gb/s Serial Link Transceiver with Data Recovery Using Oversampling," *IEEE Journal of Solid-State Circuits*, vol 33, no 5, May 1998, pp. 713-22

[19] Weinlader, D., et al., "An Eight Channel 36-GS/s CMOS Timing Analyzer," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2000, San Francisco, pp. 170-1

[20] Maneatis, J., M. Horowitz, "Precise Delay Generation Using Coupled Oscillators," *IEEE Journal of Solid-State Circuits*, vol 28, no 12, Dec. 1993, pp. 1273-82

[21] Gray, C., et. al., "A Sampling Technique and Its CMOS Implementation with 1Gb/s Bandwidth and 25ps Resolution", *IEEE Journal of Solid-State Circuits*, vol 29, no 3, Mar. 1994, pp. 340

[22] Ota, Y. et. al., "High-Speed, Burst-Mode, Packet Capable Optical Receiver and Instantaneous Clock Recovery for Optical Bus Operation," *IEEE Journal of Lightwave Technology*, vol 12, no 2, Feb. 1994, pp. 325-330

[23] Foley, D., M.P. Flynn, "CMOS DLL-Based 2-V 3.2ps Jitter 1-GHz Clock Synthesizer and Temperature-Compensated Tunable Oscillaor," *IEEE Journal of Solid-State Circuits*, vol 36, no 3, Mar. 2001, pp. 417-23

[24] Kim, C., I. Hwang, S.M. Kang, "Low-Power Small-Area +/-7.28ps Jitter 1GHz DLL-Based Clock Generator," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 8.3

[25] Farjad-rad, R., et. al., "A 0.2-2GHz 12mW Multiplying DLL for Low-Jitter Clock Synthesis in Highly-Integrated Data Communication Chips," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 4.5

[26] Ye, S., L. Jansson, I. Galton, "A Multiple-Crystal Interface PLL with VCO Realignment to Reduce Phase Noise," *IEEE ISSCC Dig. of Tech. Papers*, Feb. 2002, San Francisco, Session 4.6

[27] Kim, J., et. al., "A Low-Jitter Mixed-Mode DLL for High-Speed DRAM Applications," *IEEE Journal of Solid-State Circuits*, vol 35, no 10, Oct. 2000, pp. 1430-3