# WIDE RANGE, LOW JITTER DELAY-LOCKED LOOP USING A GRADUATED DIGITAL DELAY LINE AND PHASE INTERPOLATOR

by

Eric R. Booth

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Engineering, Electrical Engineering

Boise State University

November, 2006

The thesis presented by Eric R. Booth entitled “Wide Range, Low Jitter Delay-Locked Loop Using a Graduated Digital Delay Line and Phase Interpolator” is hereby approved:

_____

R. Jacob Baker Date
Advisor

_____

Scott F. Smith Date
Committee Member

_____

Sin Ming Loo Date
Committee Member

_____

John R. Pelton Date
Dean, Graduate College

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Jake Baker, for lending his time and expertise to this effort. He has been an outstanding teacher and a good friend during my graduate work at Boise State. I am not alone in saying that I appreciate the lengths that he goes to in order to provide education opportunities to students with full-time jobs.

I would also like to thank Tyler Gomm for being a great mentor to me over the last year as we have worked together designing DLLs. Tyler, along with Jongtae Kwak and Gary Johnson were the catalysts for many of the ideas presented in this thesis. I would also like to thank Micron Technology, Inc. for allowing me to use their resources and process models to carry out my work.

Very special thanks to my wife Callie and my daughter Ava for their patience, love, and support.

ABSTRACT

High-speed synchronous integrated circuits (ICs), such as microprocessors and memories, require clock signals to be tightly aligned for proper operation. Clock synchronization circuits are essential to eliminate clock skew across all process, voltage and temperature (PVT) variations. Digital delay-locked loops (DLLs) are commonly used for clock synchronization in modern ICs because of their superior stability and process portability. However, a drawback to these circuits is that a large number of delay elements are required to accomplish high performance across a wide operating range. A digital DLL suitable for use in a DDR-SDRAM is presented. The DLL has a graduated coarse delay line and a phase interpolating fine delay line, allowing it to cover the operating range and alignment requirements of all currently defined DDR-SDRAM families while reducing the number of delay elements by a factor of two over a traditional digital DLL.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# CHAPTER 1: INTRODUCTION

High-speed synchronous integrated circuits, such as microprocessors and memories, require clock signals to be tightly aligned for proper operation. As clock speeds and transistor counts continue to increase, this becomes more challenging. A properly designed clock synchronization circuit is essential to eliminate clock skew across all process, voltage and temperature (PVT) variations.

Double data rate synchronous dynamic random access memory (DDR-SDRAM) is an application where clock synchronization is important. It is common for many DDR-SDRAM components to be placed in parallel on a system to create a wide bandwidth for data transfer. To ensure that the data bits are aligned, each component uses a clock synchronization circuit to align its outputs with a system clock. Figure 1 shows the application of a clock synchronization circuit in an IC – such as a DDR-SDRAM – where

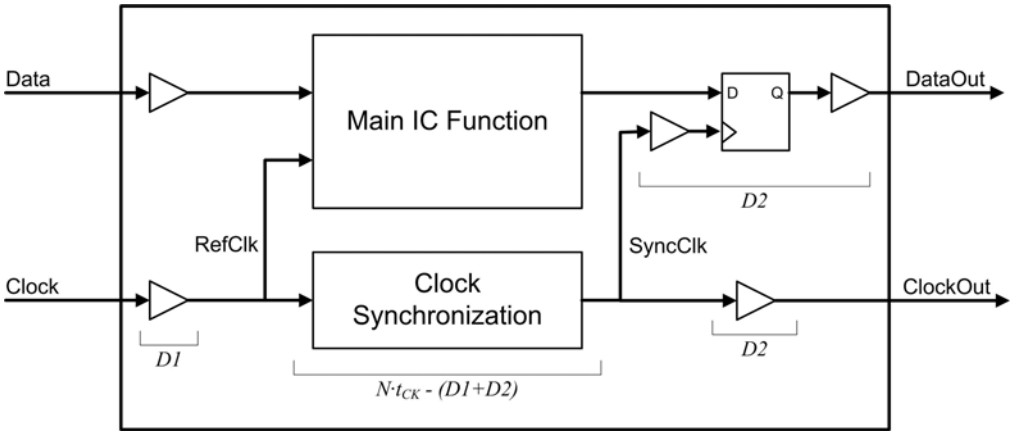



**Figure 1: Application of a clock synchronization circuit**

outputs must be aligned with a system clock. The input to the clock synchronization circuit is the system clock – delayed in time by *D1* through the input buffers – and the output is a clock that is delayed from the input by exactly

$$N \cdot t_{CK} - (D1 + D2)$$

where *D2* is the delay through the clock distribution tree, output latches, and output drivers; $t_{CK}$ is the clock period; and $N$ is an integer multiple of $t_{CK}$ that is greater than *D1+D2*. This creates a total forward path delay of $N \cdot t_{CK}$, guaranteeing that the outputs are aligned with the system clock regardless of the PVT dependent delay through *D1* and *D2* (see figure 2). Clock synchronization circuits can also be designed to accept a data stream as the input instead of a clock signal. This is called clock recovery or clock synthesis [1].

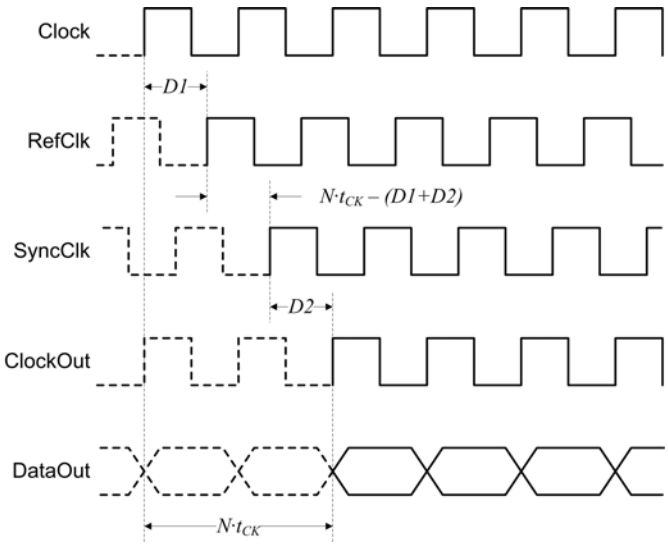



**Figure 2: Clock synchronization waveforms (N=2)**

Both open loop and closed loop clock synchronization topologies are commonly used. In either case, a delay model is used to replicate the IO delay. The devices in the delay model must have the same PVT-dependent delay characteristics as the actual IO devices to accurately track PVT changes.

### 1.1 Open Loop Clock Synchronization Topologies

In the open loop system shown in figure 3, the input clock is sent through a delay model. The clock generation circuit uses the input clock and the delayed clock to create the synchronized output clock. Open loop systems can have very fast initialization times, low standby current, and can be simpler to design than a closed loop system. However, because there is no feedback signal, any deviation from the expected behavior will go undetected and contribute to phase error. The synchronous mirror delay (SMD) and measure controlled delay (MCD) are common examples of open loop clock synchronization circuits [2],[3].

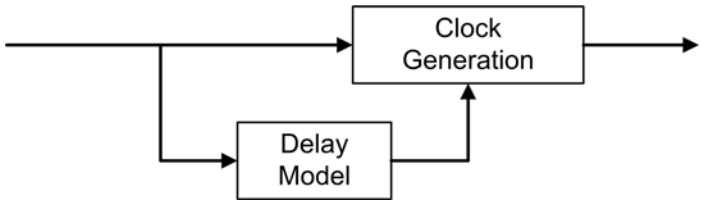



**Figure 3: Open loop clock synchronization model**

Figure 4 shows the operation of a SMD. Once the input clock has propagated through the delay model, it begins to propagate through the forward delay line. On the next rising clock edge, the mirror control circuit transfers the clock signal from the forward delay line to the corresponding point in the reverse delay line, so that each delay line inserts a delay of

$$N \cdot t_{CK} - (D1 + D2)$$

Tracing the delay through the entire path:

$$D1 + (D1 + D2) + 2 \cdot [N \cdot t_{CK} - (D1 + D2)] + D2 =$$

$$2 \cdot N \cdot t_{CK}$$

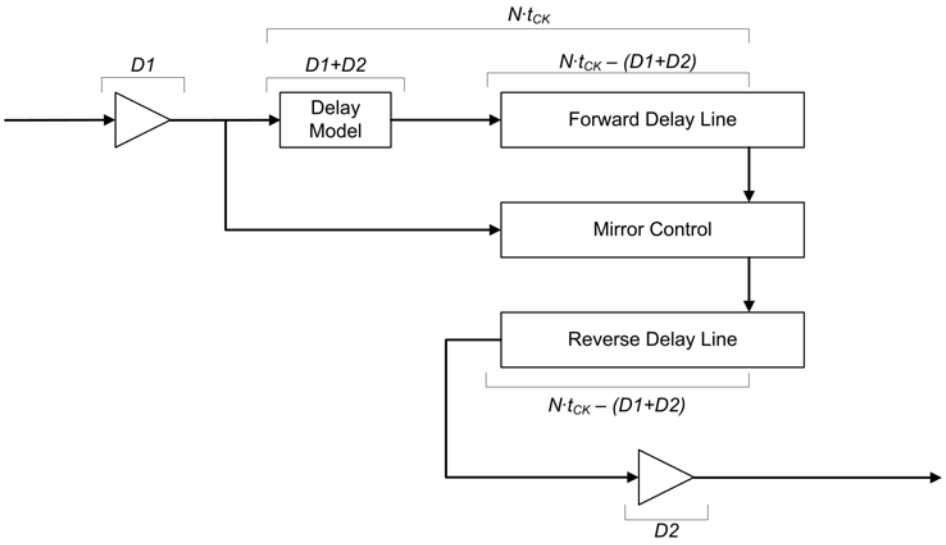



**Figure 4: SMD block diagram**

Figure 5 shows the operation of a MCD. It is similar to the SMD. The main difference is that only one delay line is actually in the forward path of the clock; the measure delay line is only used to determine the delay for the variable delay line. After the input clock has propagated through the delay model it begins to propagate through the measure delay line. On the next rising edge, the measure control circuit measures the depth of the signal in the measure delay line which will be

$$N \cdot t_{CK} - (D1 + D2)$$

The variable delay line is set to the same value so that the forward path delay is

$$D1 + N \cdot t_{CK} - (D1 + D2) + D2 =$$

$$N \cdot t_{CK}$$

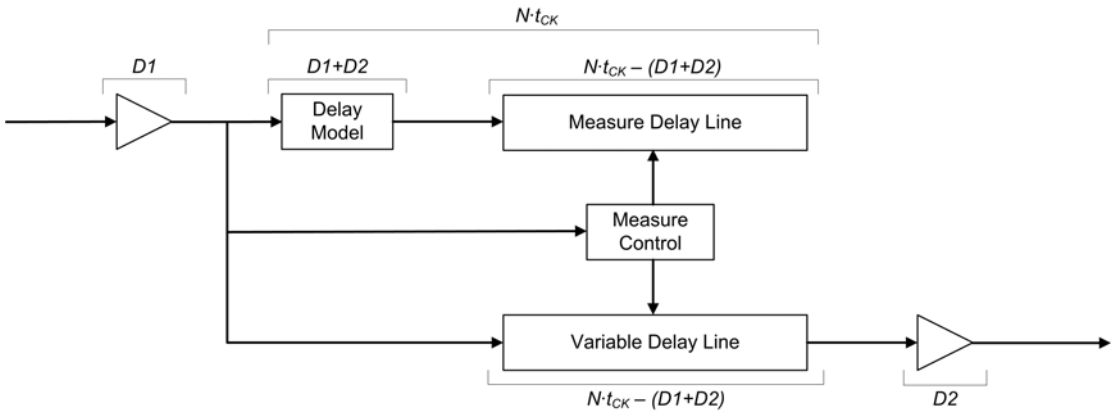



**Figure 5: MCD block diagram**

## 1.2 Closed Loop Clock Synchronization Topologies

In the closed loop configuration shown in figure 6, the output clock is fed back through the delay model and compared with the reference clock. The difference is reported to the clock generation circuit, which continually adjusts as necessary to produce the desired output. Feedback allows the system to self correct any non-ideal behavior of the clock generation circuit, but ensuring that the system remains stable under all conditions presents some design challenges. The input to the clock generation block is drawn as a dotted line because closed loop systems generate an output clock by either adding delay to the input clock or by generating a clock internally. Two closed loop examples are a phase locked loop (PLL) and a delay locked loop (DLL) [4],[5].

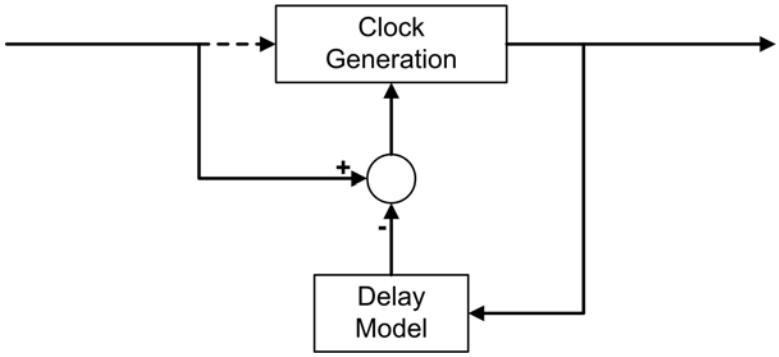



**Figure 6: Closed loop clock synchronization model**

Figure 7 shows the operation of a PLL. It consists of a variable speed oscillator, a phase detector and a loop control circuit. The output of the oscillator is fed back through the delay model and to the phase detector. The control adjusts the oscillator frequency until the phase and frequency of $\Phi_1$ and $\Phi_2$ are equal. Under these conditions, the PLL is said to be locked. The phase at the output of the oscillator can then be described as

$$\Phi_1 - (D1 + D2) \cdot 2\pi f$$

Calculating the phase of the input in reference to $\Phi_1$ yields

$$\Phi_{IN} = \Phi_1 - D1 \cdot 2\pi f$$

Doing the same for the output proves that the input and output phases are aligned:

$$\Phi_{OUT} = \Phi_1 - (D1 + D2) \cdot 2\pi f + D2 \cdot 2\pi f = \Phi_1 - D1 \cdot 2\pi f = \Phi_{IN}$$

Because a PLL generates the clock using an oscillator, it is capable of clock synthesis and can take either a data stream or a reference clock as its input.

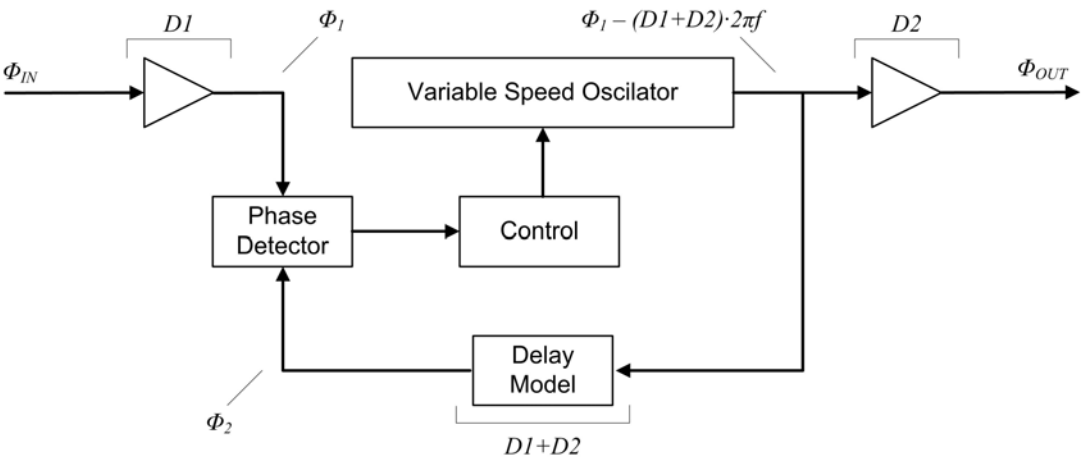



**Figure 7: PLL block diagram**

Figure 8 shows the operation of a DLL. It is similar to a PLL except that a variable delay line is used instead of a variable speed oscillator. The control circuit adjusts the delay of the variable delay line until the inputs to the phase detector are in phase. When the DLL is locked, the delay through the variable delay line is

$$N \cdot t_{CK} - (D1 + D2)$$

and the forward path delay is

$$D1 + N \cdot t_{CK} - (D1 + D2) + D2 =$$

$$N \cdot t_{CK}$$

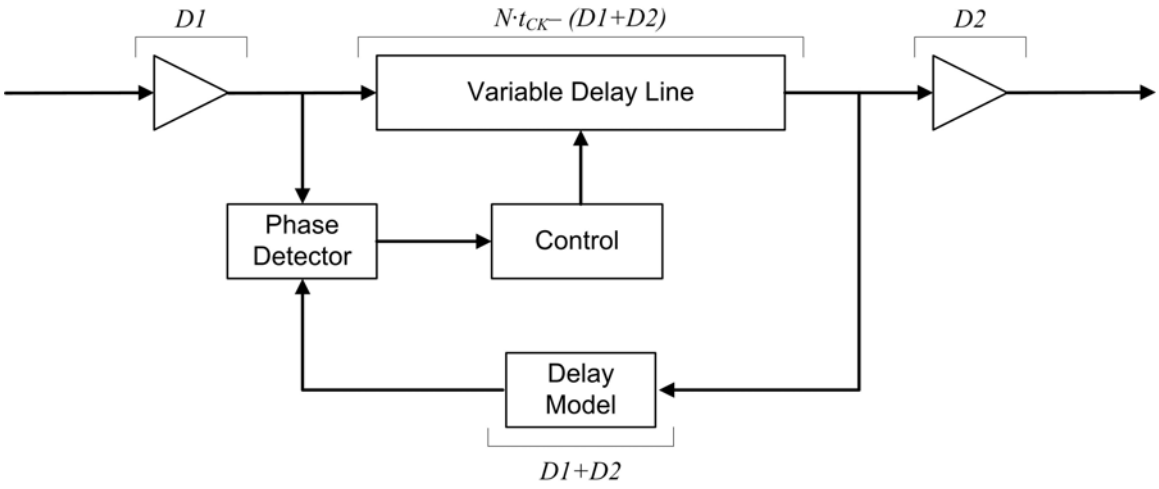



**Figure 8: DLL block diagram**

Without an oscillator, a DLL is incapable of clock recovery like a PLL. However, when a clock of the appropriate frequency is available, a DLL provides several

advantages. It can be designed as a first-order system, and thus is inherently more stable than a PLL which – because of the phase-integrating behavior of an oscillator – has an extra pole in the transfer function making it at least a second-order system [6]. Other advantages of a DLL are simpler design, easier process portability, faster locking time, and the absence of oscillator induced jitter noise [7].

This thesis explores common DLL topologies, emphasizing the effect that a large IO delay model has on design considerations. Chapter 2 discusses current DLL design techniques for both analog and digital topologies. Chapter 3 presents a digital DLL suitable for use in a high speed DRAM. The DLL has a graduated coarse delay line and a phase-interpolating fine delay line, allowing it to cover the operating range and alignment requirements of all currently defined DDR-SDRAM families while reducing the number of delay stages by a factor of two compared to a traditional digital DLL. An auto-measure circuit provides fast initialization, and an N-detection circuit - coupled with a dynamic digital sample and hold filter - allows fast tracking. Conclusions and future work are provided in chapters 4 and 5 respectively. All simulation results, unless otherwise noted, are based on Micron Technology's 80nm DRAM process models with a nominal supply voltage of 1.5V and a temperature of 25°C, using HSPICE and HSIM simulation tools.

# CHAPTER 2: CURRENT DLL DESIGN TECHNIQUES

DLLs are generally classified as either analog or digital based on the type of delay line that is used. An analog DLL uses a continuously variable delay line controlled by an analog input signal, the most common being a voltage-controlled delay line (VCDL). A digital DLL uses a delay line made up of a string of digital devices so that it has a fixed number of discrete delay steps. Usually, a shift register controls the delay setting of the delay line. This is called a register-controlled delay line (RCDL) [8],[9]. Both digital and analog DLLs typically employ phase detectors composed of digital elements, making it more correct to refer to the analog DLLs presented here as mixed-signal. However, for simplicity, they are referred to only as analog.

Because analog DLLs use a continuously variable delay line, they can be designed to exhibit less phase error (or jitter) than digital DLLs, which suffer from quantization error due to the discrete delay steps. Analog DLLs can be designed to use less layout area and consume less power, but they are more process dependent, making them less portable than digital DLLs [10]. Analog DLLs also have a tendency to be more susceptible to digital noise than a digital DLL.

## 2.1 Analog DLL Design

Figure 9 shows a basic block diagram of an analog DLL. The three main components are the VCDL, the phase detector, and the loop filter. These components are discussed in detail.

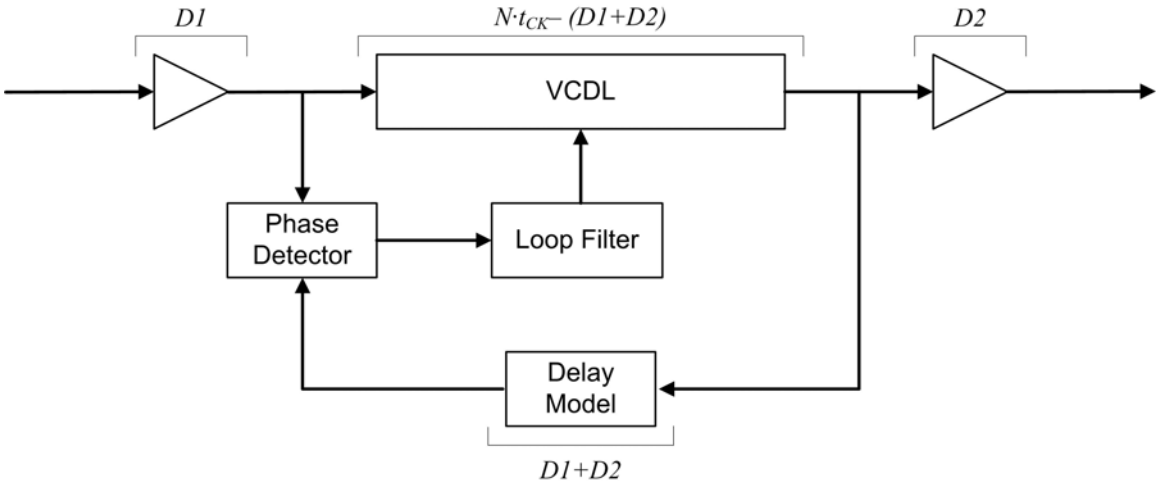
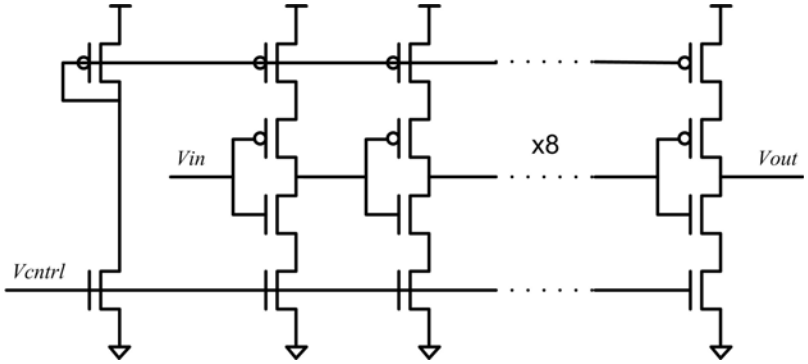



**Figure 9: Analog DLL block diagram**

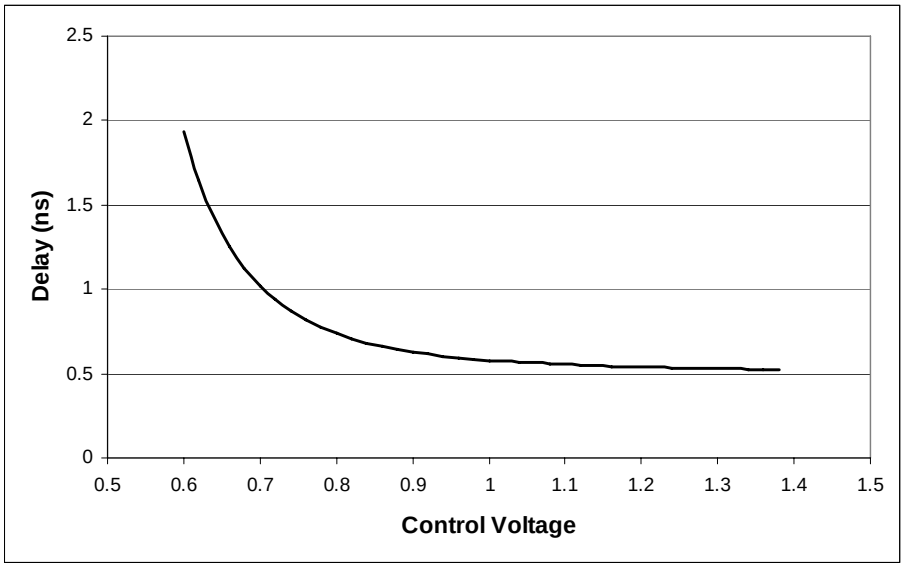
### 2.1.1 Voltage Controlled Delay Line (VCDL)

The VCDL takes two inputs: a control voltage and a clock. The output is a clock of the same frequency as the input, but phase shifted by some amount proportional to the control voltage. Figure 10 shows schematics and transfer function of a VCDL using a current starved inverter (CSI) as a delay element. The maximum current supplied to the inverters is controlled by the bias voltage *Vctrl*. A higher bias voltage increases the cell's current, thereby reducing the delay. The N and P device sizes for this example are 16/1

and 32/1 respectively, chosen to give a nominal delay of 1ns when eight delay elements are used.




**(a)**

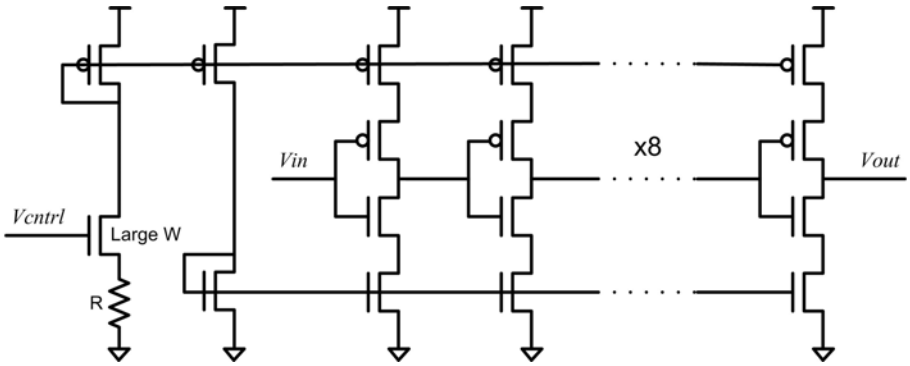



**(b)**

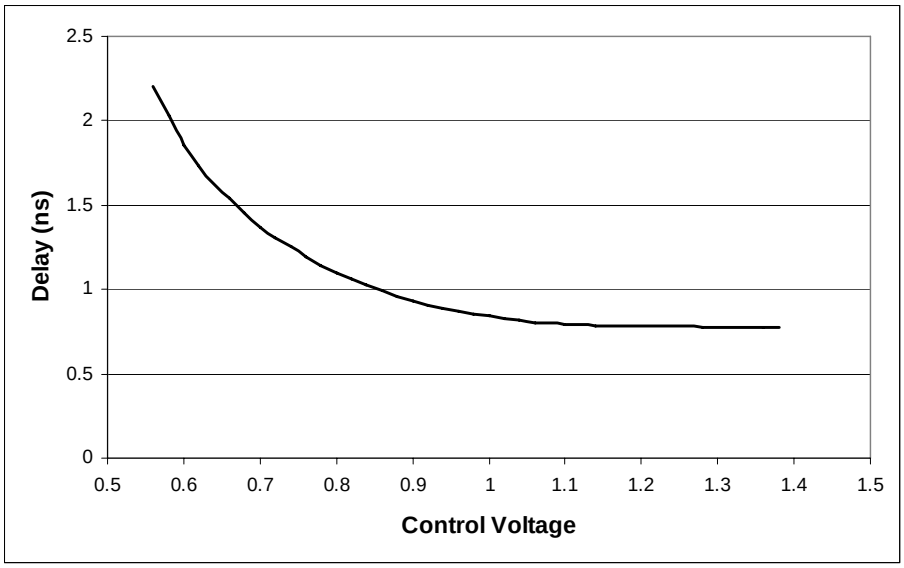
**Figure 10: CSI VCDL (a) Schematic and (b) transfer function**

The transfer function has a $1/X^2$ relationship; the delay is inversely related to the current, which is proportional to the square of the control voltage. These non-linear gain

curves are one of the design challenges of an analog DLL. Figure 11 shows how the bias circuit can be altered so that the current is linear with the input voltage [5]. This eliminates the square term in the transfer function creating a wider semi-linear region in the transfer curves (see figure 11).




**(a)**




**(b)**

**Figure 11: CSI VCDL with linearized current (a) Schematics and (b) transfer function**

Because of high susceptibility to noise and poor power supply rejection, the current starved inverter delay line is not practical for most designs. Figure 12 shows an alternate VCDL. It uses current-mode logic and voltage-controlled resistors [5],[11]. Current mode means that current is constant whether the output is static or switching. The top-most PMOS device sets this current value, which consequently sets the delay. The other two PMOS devices receive differential input to steer the current through one leg or the other. The NMOS pairs on each leg act as voltage-controlled resistors and are used to set the maximum output voltage swing. The bias circuit uses a differential amplifier, with the positive terminal connected to the output of a half-replica delay cell and the negative terminal connected to a reference voltage *Vref*. The output of the op-amp sets the precise voltage for the NMOS bias so that the output swing is always limited to *Vref* regardless of the control voltage *Vctrl*. Keeping the output swing regulated below *Vdd* reduces power supply sensitivity, and the differential signals provide common-mode noise rejection.

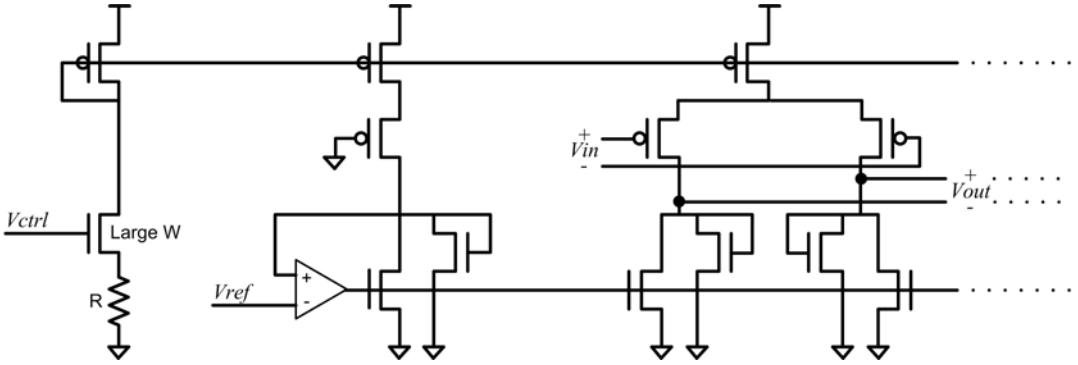



**Figure 12: Current mode logic VCDL schematic**

The improved performance does come with an expense. This delay line requires more layout area for the delay cells, an amplifier in the bias circuit, and level translators at the output. Also, because of the constant current, it will consume more power than a traditional delay line, which only requires current during switching. However, as frequency is increased, there will be a crossover point where the current mode delay line will actually consume less power because of the reduced voltage swing.

### 2.1.2 Analog DLL Phase Detectors

A phase detector generates an output signal that is proportional to the phase difference of two inputs. Analog DLLs typically use phase detectors made from digital components. The XOR, flip-flop, and phase-frequency detector (PFD) are common examples. The output of these circuits is a digital signal that must be converted to an analog voltage before they can control the VCDL.

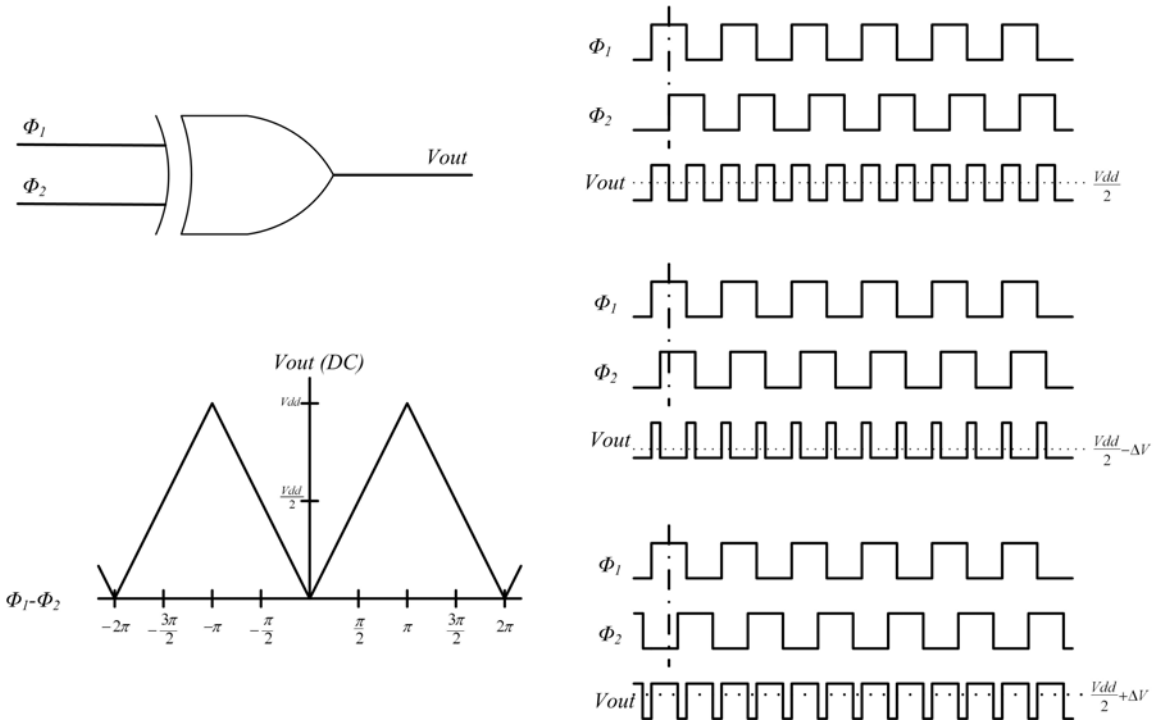



**Figure 13: XOR phase detector**

Figure 13 shows the schematic, waveforms and low-pass gain curves of the XOR PD [12]. When the inputs have a phase difference of 90º the XOR PD produces an output signal with an average value of $\frac{Vdd}{2}$. At that condition, phase error is assumed to be 0. The phase error is defined by $|\Phi_1 - \Phi_2| - 90°$ and the average output voltage will vary from $\frac{Vdd}{2}$ in proportion to that phase error. When the high frequency components are filtered out, the XOR PD has a gain of $\frac{Vdd}{\pi}$.

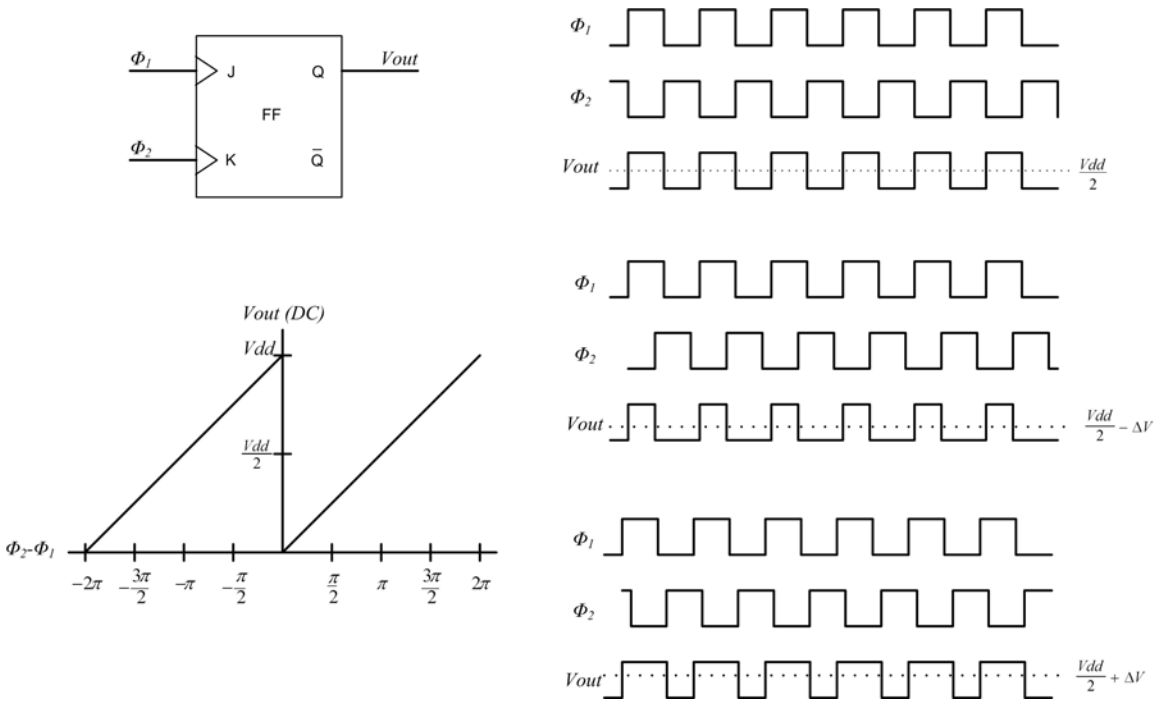



**Figure 14: Flip-flop phase detector**

Figure 14 shows the schematic, waveforms and low-pass gain curves of the flip-flop PD [12]. It is different than a conventional JK flip-flop. The rising edge of J triggers the output Q into the high state, while the rising edge of K triggers it into the low state. When the inputs have a phase difference of 180º the phase error is assumed to be zero. Like the XOR PD, the average of the output is proportional to the phase error of the inputs and is equal to $\frac{Vdd}{2}$ when phase error is zero. The gain is reduced to $\frac{Vdd}{2\pi}$ since only rising edges are used to calculate phase. However, this presents one advantage of the FF PD over the XOR PD; the input duty cycle is not constrained to 50% for correct operation.

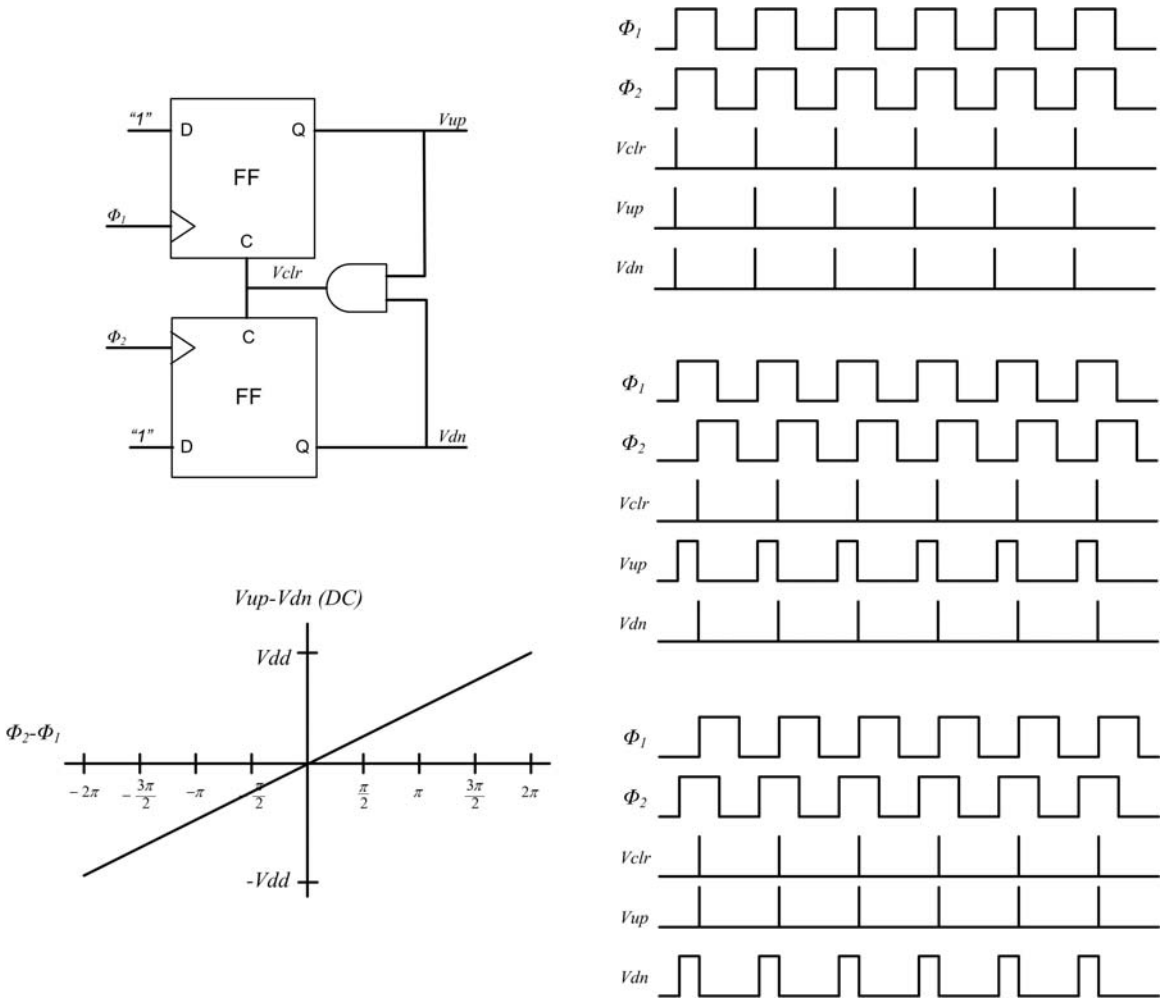



**Figure 15: PFD phase detector**

Figure 15 shows the schematic, waveforms, and low-pass gain curves of the phase-frequency detector (PFD) [12]. It is the most common phase detector used in analog DLLs. It consists of two D flip-flops and an AND gate. It has two outputs: *Vup* and *Vdn*. Because of the AND gate connected to the reset of the flip-flops, both outputs can not be high at the same time. When $\Phi_1$ leads $\Phi_2$, the average value of *Vup* will be proportional to the phase difference and *Vdn* will be zero. When $\Phi_1$ lags $\Phi_2$, the average

value of *Vdn* reports the phase difference and *Vup* will be zero. When the inputs are in phase, both outputs are low. With the output interpreted as *Vup*-*Vdn*, and the high frequency components filtered out, the gain of the PFD is $\frac{Vdd}{2\pi}$. Like the FF phase detector, the PFD phase detector does not require 50% duty cycle.

### 2.1.3 Loop Filter

An analog DLL is controlled by a loop filter circuit. Many different loop filters can be used, and the properties of the filter will have a large impact on the performance of the DLL. Two important performance metrics of a DLL are tracking time and jitter. Tracking time is the time required for the DLL to respond to phase error, and jitter is the amount of phase error on the output when the input is constant. The design of the loop filter must consider the trade-offs between these properties.

Figure 16 shows the control model of a generic analog DLL.

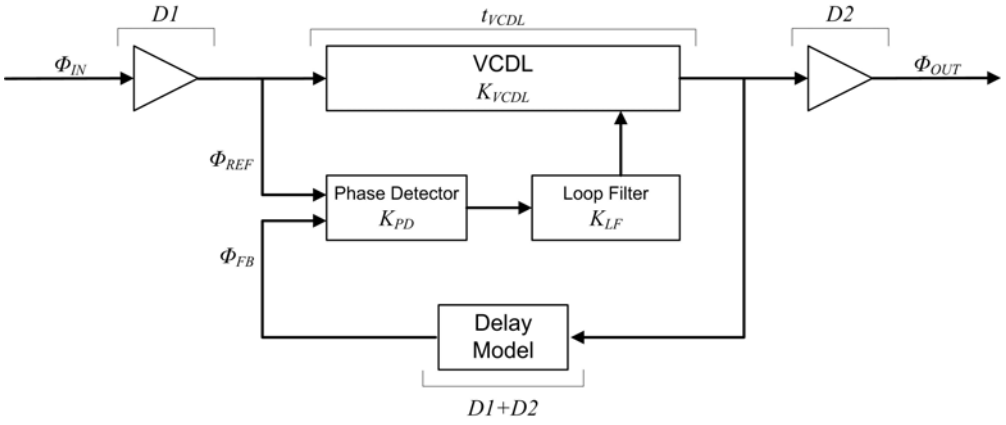



**Figure 16: Analog DLL control model**

The model can be simplified by realizing that the relationship between $\Phi_{IN}$ and $\Phi_{OUT}$ is exactly the same as the relationship between $\Phi_{REF}$ and $\Phi_{FB}$.

$$\Phi_{OUT} = \Phi_{IN} + (t_{D1} + t_{VCDL} + t_{D2}) \cdot 2\pi f$$

$$\Phi_{FB} = \Phi_{REF} + (t_{VCDL} + t_{D1} + t_{D2}) \cdot 2\pi f$$

This is by design, and must be true for the DLL to function properly. The block diagram can now be simplified by replacing $\Phi_{REF}$ and $\Phi_{FB}$ with $\Phi_{IN}$ and $\Phi_{OUT}$ as in figure 17. The true $\Phi_{IN}$ and $\Phi_{OUT}$ will be shifted in time by $D1$.

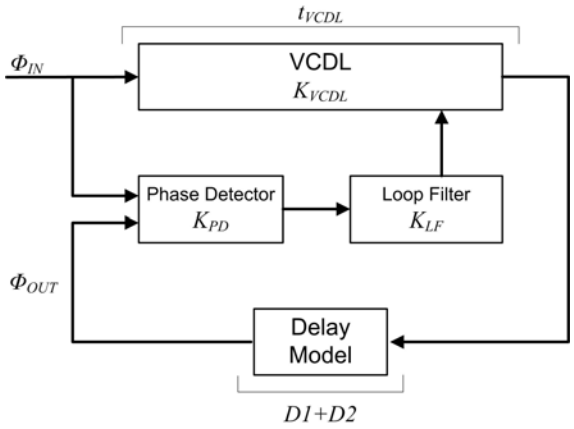



**Figure 17: Analog DLL control model (loop only)**

.

If the DLL is designed such that the DLL bandwidth (the rate at which the DLL will respond to a change in the input phase) is at least one decade below the bandwidth around the loop (the rate at which a signal will propagate around the entire loop) – a requirement for good jitter performance [7], [11] – then the delay model can be ignored

for small signal analysis, and the DLL control model can be further simplified to figure 18.

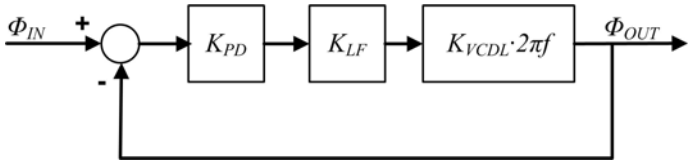



**Figure 18: Simplified analog DLL control model**

Now the transfer function can be easily obtained by

$$\Phi_{OUT} = (\Phi_{IN} - \Phi_{OUT}) \cdot K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f$$

$$\frac{\Phi_{OUT}}{\Phi_{IN}} = \frac{K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f}{1 + K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f} = \frac{1}{1 + \frac{1}{K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f}}$$

The DLL bandwidth for a first order loop filter can be described as $\omega_{DLL}$ where

$$\frac{\Phi_{OUT}}{\Phi_{IN}} = \frac{1}{1 + \frac{s}{\omega_{DLL}}}$$

$$\omega_{DLL} = s \cdot K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f$$

The loop bandwidth $\omega_{LOOP}$ is defined as $2\pi f_{LOOP}$ where $f_{LOOP}$ is the loop frequency and is equal to one over the total delay around the loop:

$$\omega_{LOOP} = \frac{2\pi}{t_{VCDL} + D1 + D2}$$

The VCDL is typically designed so delay is never more than one period, so the maximum loop bandwidth can be written as

$$\omega_{LOOP\max} = \frac{2\pi}{T + D1 + D2}$$

The design rule that DLL bandwidth should be kept at least a decade below the loop bandwidth can be written as

$$\omega_{DLL} \leq \frac{1}{10}\omega_{LOOP\max}$$

$$s \cdot K_{PD} \cdot K_{LF} \cdot K_{VCDL} \cdot 2\pi f \leq \frac{1}{10} \cdot \frac{2\pi}{T + D1 + D2}$$

The performance of the DLL is determined by the relationship between $\omega_{DLL}$ and $\omega_{LOOP}$. The tracking time is proportional to $\frac{1}{\omega_{DLL}}$ and the jitter is proportional to $\frac{\omega_{DLL}}{\omega_{LOOP}}$. Qualitatively, these relationships make sense. The tracking time will be inversely related to the DLL bandwidth (a wider bandwidth equals a faster response). However, making the response too fast will have an effect on jitter. If the DLL bandwidth is greater than the loop bandwidth, the DLL will adjust for errors seen at the phase detector faster than the new phase information can make it around the loop. This will cause the loop filter to oscillate and will be seen as jitter on the output.

### 2.1.4 Charge Pump and Capacitor Loop Filter

A commonly used first order loop filter is a simple capacitor. This loop filter is usually coupled with a PFD phase detector and charge pump in the configuration shown in figure 19.

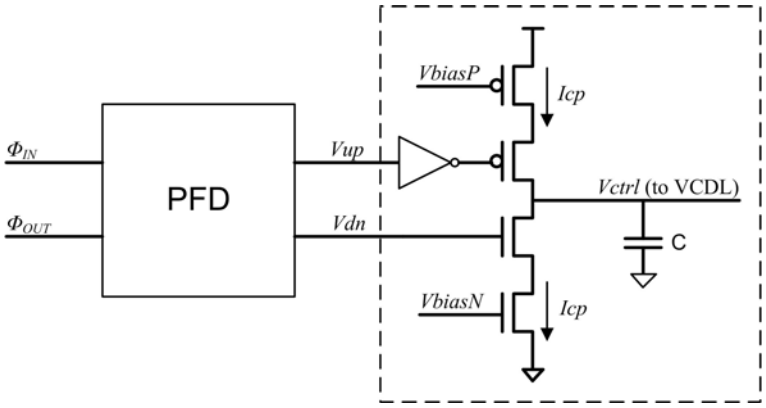



**Figure 19: Charge pump and capacitor loop filter schematics**

The gain of this loop filter is $\frac{I_{CP}}{sC}$. The transfer function of the DLL can now be defined in terms of the loop filter gain $\frac{I_{CP}}{sC}$, the PFD gain $\frac{Vdd}{2\pi}$, and the gain of the VCDL approximated as a constant value $K_{VCDL}$, so that

$$\frac{\Phi_{OUT}}{\Phi_{IN}} = \frac{1}{1 + \frac{2\pi \cdot sC}{Vdd \cdot I_{CP} \cdot K_{VCDL} \cdot 2\pi f}}$$

The DLL bandwidth $\omega_{DLL}$ is

$$\frac{Vdd \cdot I_{CP} \cdot K_{VCDL}}{C \cdot T}$$

The design rule can be used to determine the required charge pump current to capacitance ratio of the filter to minimize jitter.

$$\frac{I_{CP}}{C} \leq \frac{T \cdot 2\pi}{10 \cdot (T + D1 + D2) \cdot Vdd \cdot K_{VCDL}}$$

Using the delay line from figure 11, $K_{VCDL}$ can be approximated at 1.5ns/V. Setting the IO delay *D1+D2* at 5ns, the period *T* at 2ns, and the charge pump current $I_{CP}$ at 50uA, the required value of *C* is approximately 1pF or larger.

$$C \geq \frac{10 \cdot 50uA \cdot 6ns \cdot 1.5V \cdot 1.5\frac{ns}{V}}{1ns \cdot 2\pi} \geq 1.07pF$$

Figure 20 shows simulation results for this DLL using loop filter capacitor values of 1pF, 0.5pF and 0.2pF. The input phase is stepped by 90°. The waveforms show the output of the loop filter as it responds to the phase error and attempts to find the lock point. Using the 1pF filter ($\omega_{DLL} = \frac{1}{10}\omega_{LOOP}$) gives a nice damped response. When the 0.5pF filter is used ($\omega_{DLL} = \frac{1}{5}\omega_{LOOP}$) there is some overshoot, but the DLL still finds the lock point. With a 0.1pF filter ($\omega_{DLL} = \frac{1}{2}\omega_{LOOP}$) the DLL begins to oscillate and never locks.

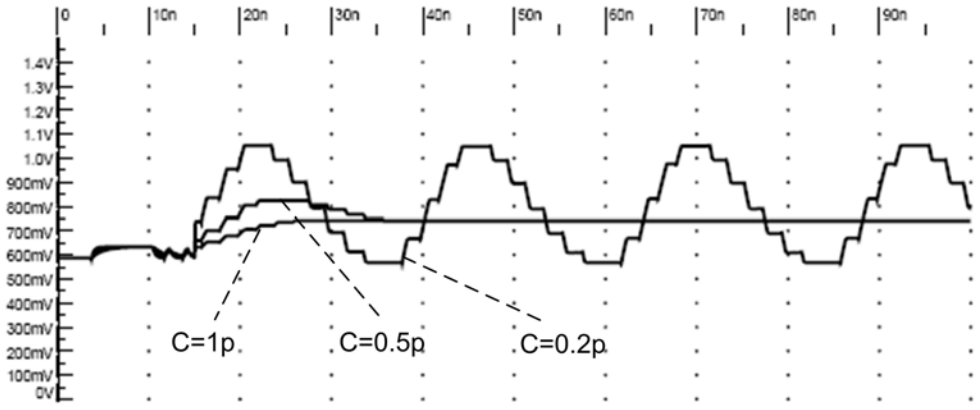



**Figure 20: DLL response using a 1p, 0.5p and 0.2p loop filter**

The stair step shape of the waveforms may look strange, but it is exactly what is expected when using this filter. The PFD turns on the charge pump for a portion of each clock cycle proportional to the amount of phase error. During that interval, the capacitor is charged or discharged at a constant rate. For the rest of the clock cycle, the charge pump is turned off and the capacitor charge remains constant.

### 2.1.5 Other First Order Loop Filters

Another first order loop filter is a simple low-pass RC circuit. It can be applied to the output of the XOR or FF phase detectors to filter out the high frequency components and allow the VCDL to be controlled directly. However, this filter has a limitation. The output of the filter is constrained to $\frac{Vdd}{2}$ when phase error is zero. This is not practical in most designs since the ideal VCDL input will vary widely with PVT and frequency.

An active filter, such as the active proportional + integral filter shown in figure 21, can be used instead of an RC filter. The integrating capacitor between the input and output of the op-amp stores the voltage difference between $\frac{Vdd}{2}$ and the ideal VCDL voltage. The gain of the filter is $-\frac{1+sR_2C}{sR_1C}$.

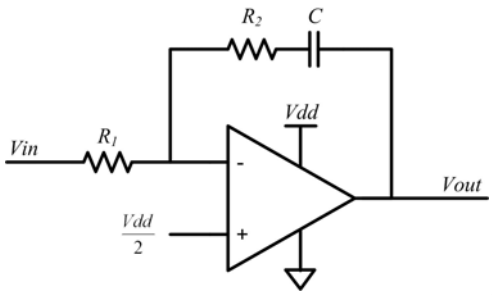



**Figure 21: Active proportional + integral loop filter**

2.1.6 Initialization and Lock Range

The analog DLL requires careful consideration of what happens during initialization, especially when a large IO model is in the feedback path. If the DLL initializes such that

$$t_{VCDL} > (N + \tfrac{1}{2}) \cdot t_{CK} - (D1 + D2)$$

the error at the phase detector will be greater than 180º, forcing the DLL to add delay. The gain curves of the analog VCDL allow practically infinite delay so the DLL will find the next lock point at

$$t_{VCDL} = (N + 1) \cdot t_{CK} - (D1 + D2)$$

The final delay in the delay line will be greater than one clock period, and the VCDL gain will be increased beyond the expected range. This is sometimes referred to as a harmonic lock.

Another, and potentially more dangerous, problem is when the DLL initializes such that

$$t_{VCDL} < (N - \tfrac{1}{2}) \cdot t_{CK} - (D1 + D2)$$

In this case, the phase error will be below -180º. The phase detector will report a positive error, and the DLL will try to remove delay. Since the gain goes to zero when the VCDL approaches minimum delay, it is likely that the VCDL will saturate and never find the lock point. For these reasons, a conventional analog DLL is usually designed so that the VCDL operating range has the following limitations:

$$t_{VCDL\min} > (N - \tfrac{1}{2}) \cdot t_{CK} - (D1 + D2)$$

$$t_{VCDL\max} < (N + \tfrac{1}{2}) \cdot t_{CK} - (D1 + D2)$$

It is possible for the IO delay to vary across PVT by a factor of 3 to 1. If the IO delay is a significant percentage of the clock period, the VCDL must be constrained to a very small range to avoid false lock. As clock speeds have increased, it is common for the IO delay to be greater than the clock period. This makes the traditional analog DLL a poor choice for clock synchronization in many modern ICs.

Numerous designs have been presented to address the range limitations of analog DLLs. A DLL has been designed that uses an auxiliary VCDL to monitor the main VCDL to detect and prevent a harmonic lock condition [13]. Other DLLs have been designed that create multiple tap points from the VCDL, and then employ a phase selection and/or inversion scheme to extend the lock range and prevent false locking [7],[14],[15]. These designs come at the expense of added control complexity, and if phase inversion is applied, the duty cycle of the input clock is no longer maintained.

When IO delay variation across PVT is a concern, a dual loop analog DLL can be used. In this configuration, a primary reference DLL – without a delay model in the feedback loop – locks to the reference clock. This eliminates the range restrictions imposed by the delay model. It also ensures that the reference VCDL contains exactly one clock period so that multiple, equally spaced clocks are available across the entire phase range. A secondary DLL uses a feedback loop with a delay model to select the output tap from the reference VCDL that yields the smallest phase error [16]. The accuracy of a dual loop DLL can be increased by using a phase mixing circuit to interpolate between the reference VCDL tap points [17].

## 2.2 DIGITAL DLL DESIGN

A digital DLL does not suffer from the range restrictions of the traditional analog DLL, and false locking is easier to prevent. Other advantages are the unconditional stability of a zero-order transfer function and easier processes portability. This comes at the expense of larger layout area, more power consumption, and increased jitter – due to quantization of the discrete delay steps.

Figure 22 shows a block diagram of a traditional digital DLL. The three main blocks are the register controlled delay line (RCDL), the phase detector, and a counter or state machine. These components are discussed in detail and compared to their analog counterparts.

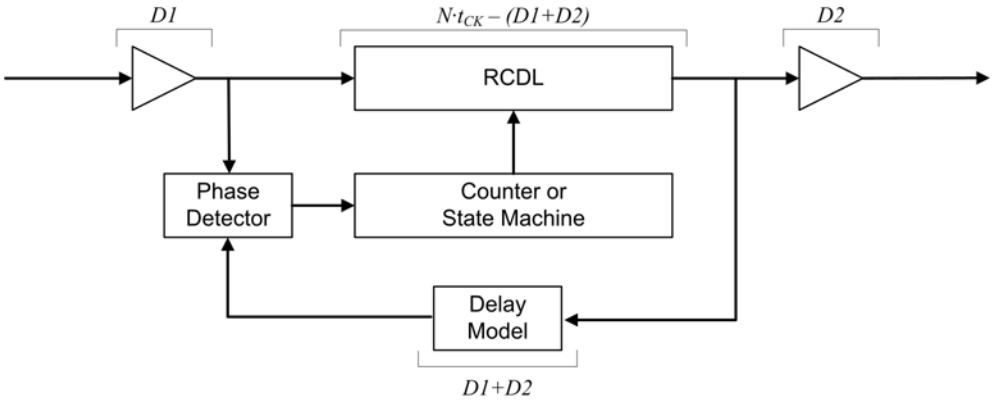



**Figure 22: Digital DLL block diagram**

2.2.1 Register Controlled Delay Line (RCDL)

The RCDL is composed of discrete delay elements and a shift register. The register setting determines how many delay elements are used between the input and output. In other words, the RCDL converts a digital code into a time delay. The transfer function can be expressed simply as delay per stage. This value determines the maximum phase error due to quantization.

Figure 23 shows a block diagram of a simple RCDL using inverter pairs as delay elements. In this topology, a shift register controls a large multiplexer circuit to determine which delay tap is sent to the output; the shift register could be replaced by an alternate control circuit such as an up-down counter or state machine.

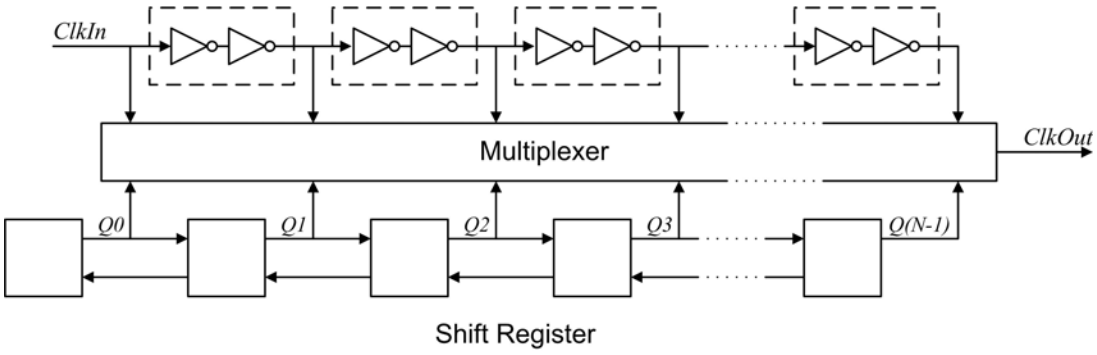



**Figure 23: Inverter-inverter RCDL block diagram**

The delay per stage $t_D$ determines the jitter due to the quantization error. Ideally, the DLL will select the tap point that produces the smallest phase error between the input and output clocks, so the maximum phase error is $\pm \frac{1}{2} t_D \cdot 2\pi f$ and occurs when the ideal output lies directly between two tap points. Some digital DLLs do not have a steady

state. Instead, they oscillate between the two tap points closest to the ideal output. The maximum quantization error is increased to $\pm t_D \cdot 2\pi f$, and occurs when the ideal output is aligned with a tap point (See figure 24).

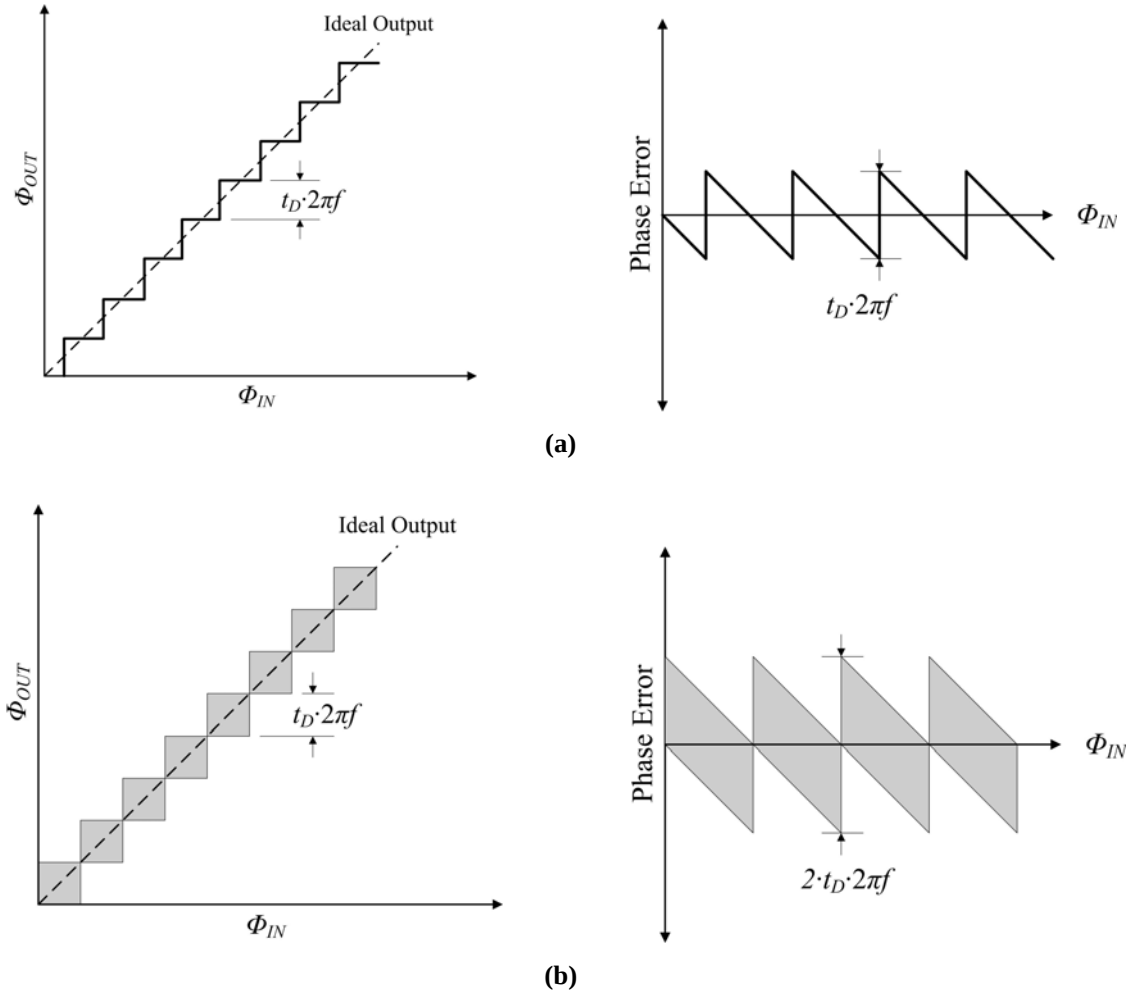



**Figure 24: Quantization error for (a) steady state and (b) oscillating DLL**

The quantization error of a digital DLL can be cut in half by using a single inverter as a delay element. This requires two delay lines which are 180° out of phase

with each other. The output taps alternate between the two delay lines to avoid phase inversion [18]. One drawback to this topology is that mismatch between the two delay lines can increase quantization error.

Inverters have the smallest intrinsic delay of the discrete devices, so they exhibit the smallest quantization error. However, power savings cannot be achieved because there is no mechanism to disable the unused elements, and the large multiplexing circuit required to select a delay tap can cause duty cycle distortion and increase power supply sensitivity.

A NAND-inverter delay element allows power savings by providing an enable path, and eliminates the need for the multiplexing circuit required in the exit point delay line by allowing multiple entry points (figure 25). A drawback to the entry point

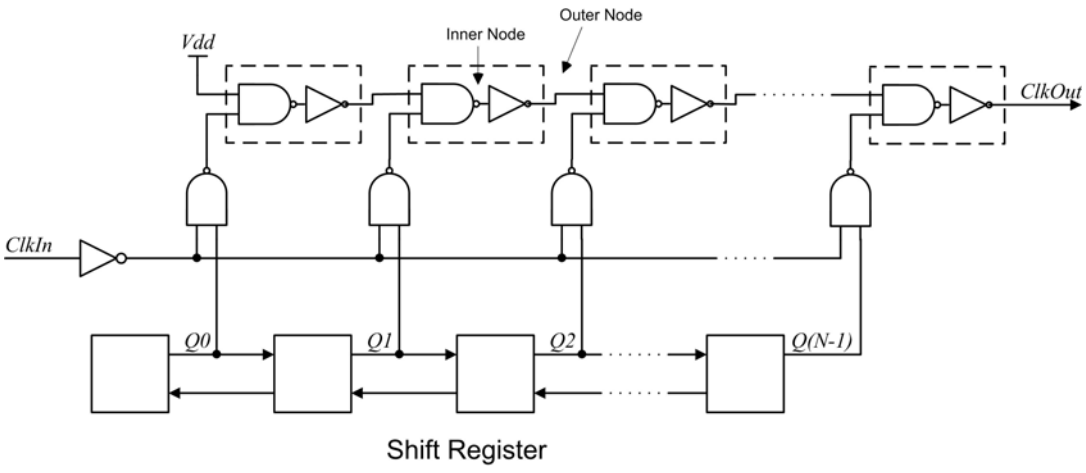



**Figure 25: Entry-point NAND-inverter RCDL block diagram**

topology is that the large drivers required to distribute the clock will cause simultaneous switching noise on the power bus and substrate, which affects the delay line performance. Another problem is duty cycle error caused by using two different logic devices. The duty cycle error can be described in terms of the difference between the propagation delay of a rising edge $t_{Rise}$ and a falling edge $t_{Fall}$.

$$t_{Error} = t_{Rise} - t_{Fall}$$

The propagation delay of a rising edge is calculated by

$$t_{Rise} = N(t_{PHLI} + t_{PLHO})$$

and the falling edge is

$$t_{Fall} = N(t_{PLHI} + t_{PHLO})$$

where $N$ is the number of delay elements, $t_{PHLI}$ and $t_{PHLO}$ are the high to low transition times at the inner and outer nodes respectively, and $t_{PLHI}$ and $t_{PLHO}$ are the low to high transition times of the inner and outer nodes. Using 50% as the switching point, the transition times can be estimated as

$$t_{PHLI} = \ln(\tfrac{1}{50\%}) Rn_{NAND} C_I = 0.7 \cdot Rn_{NAND} C_I$$

$$t_{PHLO} = 0.7 \cdot Rn_{INV} C_O$$

$$t_{PLHI} = 0.7 \cdot Rp_{NAND} C_I$$

$$t_{PLHO} = 0.7 \cdot Rp_{INV} C_O$$

where $Rp_{NAND}$ , $Rn_{NAND}$ ,$Rp_{INV}$ and $Rn_{INV}$ are the effective pull up and pull down switching resistances of the NAND and inverter respectively, and $C_I$ and $C_O$ are the total

capacitance of the inner and outer nodes. Using these relationships, the duty error can be expressed as

$$t_{Error} = 0.7 \cdot N \cdot [C_I (Rn_{NAND} - Rp_{NAND}) - C_O (Rn_{INV} - Rp_{INV})]$$

The equation shows that there are two ways to eliminate duty cycle error in this type of delay line. Either both devices must have an effective P/N drive ratio of 1 (both mismatch terms go to zero), or the capacitive loads must be matched **and** the P/N ratios of both devices must be equal (the mismatch terms cancel each other out). Both of these conditions rely on the P/N ratio to be tightly controlled which is difficult using current fabrication methods.

If a delay line uses delay cells composed of two identical logic gates, the duty cycle error equation simplifies to

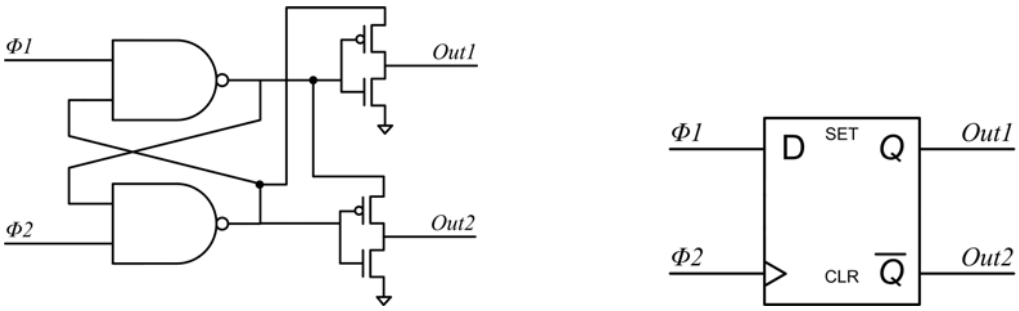
$$t_{Error} = 0.7 \cdot N \cdot (C_I - C_O)(Rn - Rp)$$

This shows that if the capacitive loads are matched at both the inner and outer nodes – a much easier task using current fabrication methods – the duty cycle is not affected by the P/N drive ratios. Using a NAND-NAND delay cell requires some extra layout area but can eliminate P/N ratio induced duty cycle distortion while still allowing the flexibility of an entry point topology or an exit point topology with power savings.

An alternative digital delay line is a cycle controlled delay line [10]. This type of delay line reduces layout area by using a delay line that is only a fraction of the required size. It re-circulates the input through it a fixed number of times, and then uses a secondary delay line to increase resolution. While this design does have potential to greatly reduce layout area, it has complex control and timing requirements.

2.2.2 Digital DLL Phase Detectors

Digital DLLs operate in the z-domain and use phase detectors which output a digital code instead of a voltage. Because the digital code is usually just a binary value indicating whether the DLL should add or remove a unit delay, the phase detector itself does not have a gain, only a direction. A phase detector commonly used in digital DLLs is an arbiter circuit (figure 26a). It consists of a NAND set-reset latch and two inverters. The inverters are powered by the opposing NAND output. This ensures only one output is high and reduces the likelihood of a meta-stable condition. The signal that arrives first will cause the corresponding output to pulse high while holding the opposing output low.




**(a)** **(b)**

**Figure 26: (a) Arbiter and (b) D flip-flop phase detectors**

One drawback of the arbiter is that the output is only valid for half of a clock cycle (while the leading input is high). A D flip flop can be used as a phase detector whose output is always valid (figure 26b). It will be synchronous with *Φ2* and will be

true if *Φ1* arrives early and false if *Φ1* arrives late. The accuracy of this phase detector is dependent on the setup and hold time of the flip-flop.

The arbiter and flip-flop phase detectors – if used alone – will lead to a DLL that oscillates constantly between two tap points. Figure 27 shows how two flip-flop phase detectors can be used together to create hysteresis and provide a steady state condition for a DLL. Not only will this reduce the maximum quantization error, but it also reduces noise and power consumption. A delay is added before the D input of flip-flop 1 so that *Out1* will only go high if *Φ1* leads *Φ2* by at least $½\cdot t_D$. The inputs are reversed to flip flop 2 so that *Out2* will only go high if *Φ2* leads *Φ1* by at least $½\cdot t_D$. This will prevent the DLL from responding to phase differences greater than the quantization error of $\pm½\cdot t_D$ (see figure 27). It is critical that the hysteresis delay match the DLL unit delay. If it is too small, the DLL will oscillate, and if it is too large, the maximum quantization error will be increased.

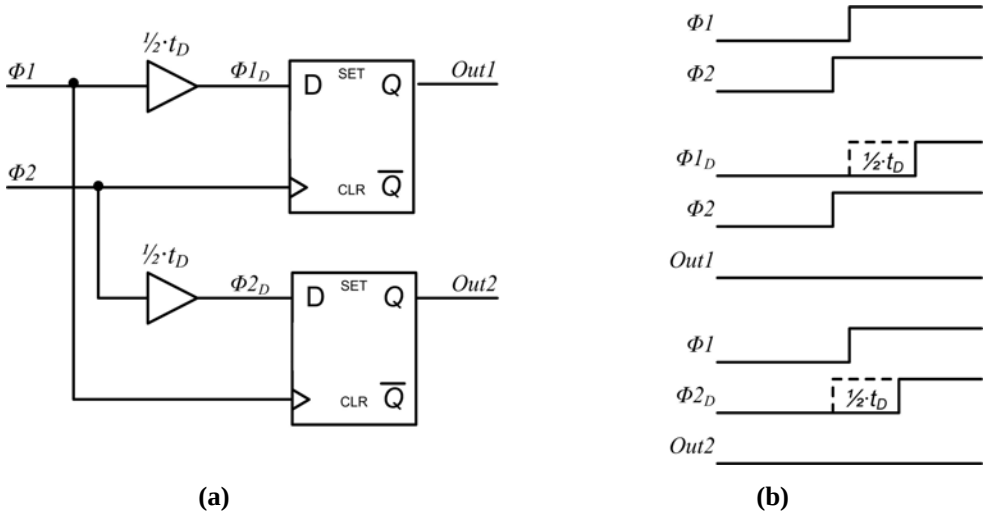



**Figure 27: (a) Schematic and (b) waveforms for a phase detector with hysteresis**

2.2.3 Digital DLL Control

Because a digital DLL operates in the z-domain, and does not have a continuous response like the analog DLL, the design rule – that DLL bandwidth should be one decade below the loop bandwidth – does not directly apply. However a digital DLL must be designed so that there is sufficient time between shifts for new phase information to make it around the loop. This is called the wait time. The worst-case wait time is determined by dividing the maximum loop delay by the minimum clock period:

$$t_{WAIT} = \frac{t_{LOOP\max}}{T_{\min}}$$

A digital DLL usually has some control and buffering devices that add intrinsic delay to the loop. Because the intrinsic delay is inside the loop, the DLL will compensate for it in the delay line, so it does not contribute to phase error. Including the maximum intrinsic delay $t_{DLLmax}$, the wait time can be written as

$$t_{WAIT} = \frac{N \cdot T_{\min} - (D1 + D2 + t_{DLL\max}) + D1_{\max} + D2_{\max} + t_{DLL\max}}{T_{\min}} = \frac{N \cdot T_{\min}}{T_{\min}} = N_{\max}$$

where $N_{\max}$ is the smallest multiple of $T_{\min}$ that is greater than $D1_{\max} + D2_{\max} + t_{DLL\max}$. The DLL must ensure that shift to shift time is at least $N_{\max}$ cycles or jitter will be increased.

There are many ways to implement this control in a digital DLL. The example in Figure 28 shows a generic control scheme where the control block contains a counter and decode that allow the shift register to directly sample the phase detector every $N_{max}$ cycles. An averaging filter can be used in conjunction with the wait counter to filter out random phase noise. This will increase the tracking time of the DLL but make the design more immune to noise events.

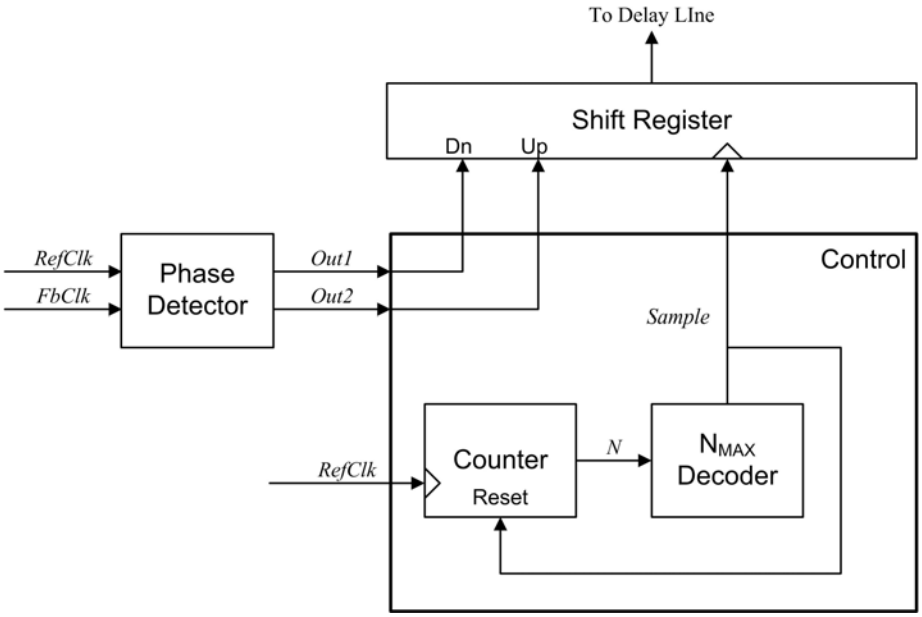



**Figure 28: Digital DLL control**

Table 1 shows the performance metrics of a conventional digital DLL described in terms of the minimum and maximum delay per stage $t_{D\min}$ and $t_{D\max}$, the number of delay stages $X$, and $N_{MAX}$. The jitter is related to the quantization error and is based on the delay per stage. The minimum frequency can be increased by adding more delay stages, and the maximum frequency is limited only by the rise and fall time of the delay elements.

**Table 1: Digital DLL performance metrics**

| | Min | Max | Units |
|---|---|---|---|
| Jitter (steady-state) | 0 | $\pm\frac{1}{2}t_{D\max}$ | seconds |
| Jitter (oscillating) | $\pm\frac{1}{2}t_{D\max}$ | $\pm t_{D\max}$ | seconds |
| Operating Frequency | $\frac{1}{X\cdot t_{D\min}}$ | $\sim\frac{1}{t_{DRise\max}+t_{DFall\max}}$ | hertz |
| Tracking time | $2\pi f^2\cdot\frac{t_{D\min}}{N_{\max}}$ | $2\pi f^2\cdot\frac{t_{D\max}}{N_{\max}}$ | $\frac{\text{radians}}{\text{second}}$ |

2.2.4 Dual Loop Digital DLL

In order to reduce quantization error beyond what is possible with discrete elements, a dual loop topology can be applied to a digital DLL (figure 29). A fine delay line, composed of analog elements, is used to create smaller delay increments. Each fine delay step $t_{DF}$ should be equal to $\frac{t_D}{M}$ where $M$ is the number of fine delay steps. In other words, the total delay of the fine delay line should equal one coarse delay.

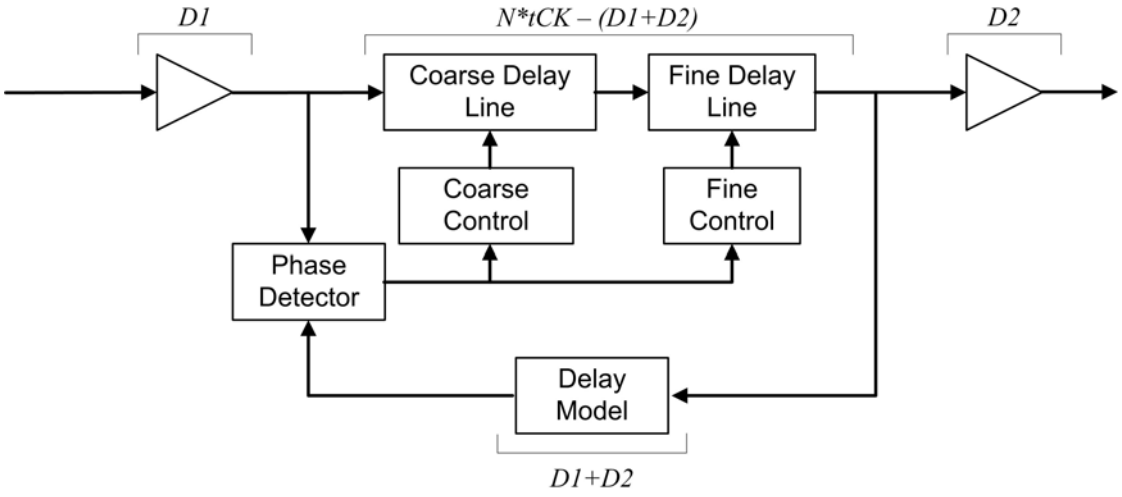



**Figure 29: Digital dual loop DLL block diagram**

Figure 30 shows an example of a fine delay line made up of an array of capacitors that can be independently switched into the clock path. Usually the switches and capacitors are implemented using transistors. The capacitor array should be spread out over two complementary nodes to avoid duty cycle distortion.

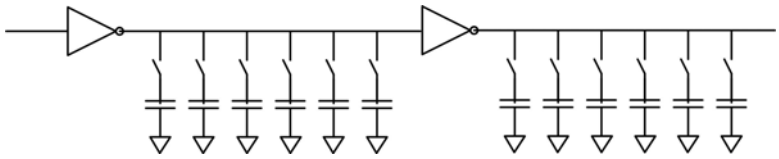


**Figure 30: Capacitor based fine delay line**

A dual delay line topology improves performance, but also introduces two design issues that must be considered. 1) Control discontinuity. When tracking an input whose phase is changing, the DLL adjusts the fine delay line until it reaches a boundary. It then must reset the fine while simultaneously adjusting the coarse. If this does not happen simultaneously, a phase error of up to one coarse delay will be briefly seen on the output. This essentially throws out the performance benefit provided by the fine delay line. 2) Delay discontinuity. This occurs when the maximum fine delay is not equal to one coarse delay, causing a phase error of $t_D - M \cdot t_{DF}$ every time a coarse to fine boundary is crossed. This is much more common since it is difficult to match the delay across PVT. However, the delay discontinuity can usually be kept small.

A phase interpolating circuit can be used to eliminate both the control and delay discontinuity problems of a traditional fine delay line. Just like the analog dual loop DLL, the phase mixer will interpolate between the coarse delay taps. Figure 31 shows two phase mixer topologies, each capable of creating five interpolation points from two inputs. The core principle is to create a temporary contention on a common output node. In the single stage phase mixer, the contention occurs on a single node. A thermal coded input word determines the weighing given to the input phases by selecting which

inverters will be Hi-Z and which will be enabled. In the multi-stage binary phase mixer, the fight occurs on multiple nodes, and the phases are split in a binary fashion.

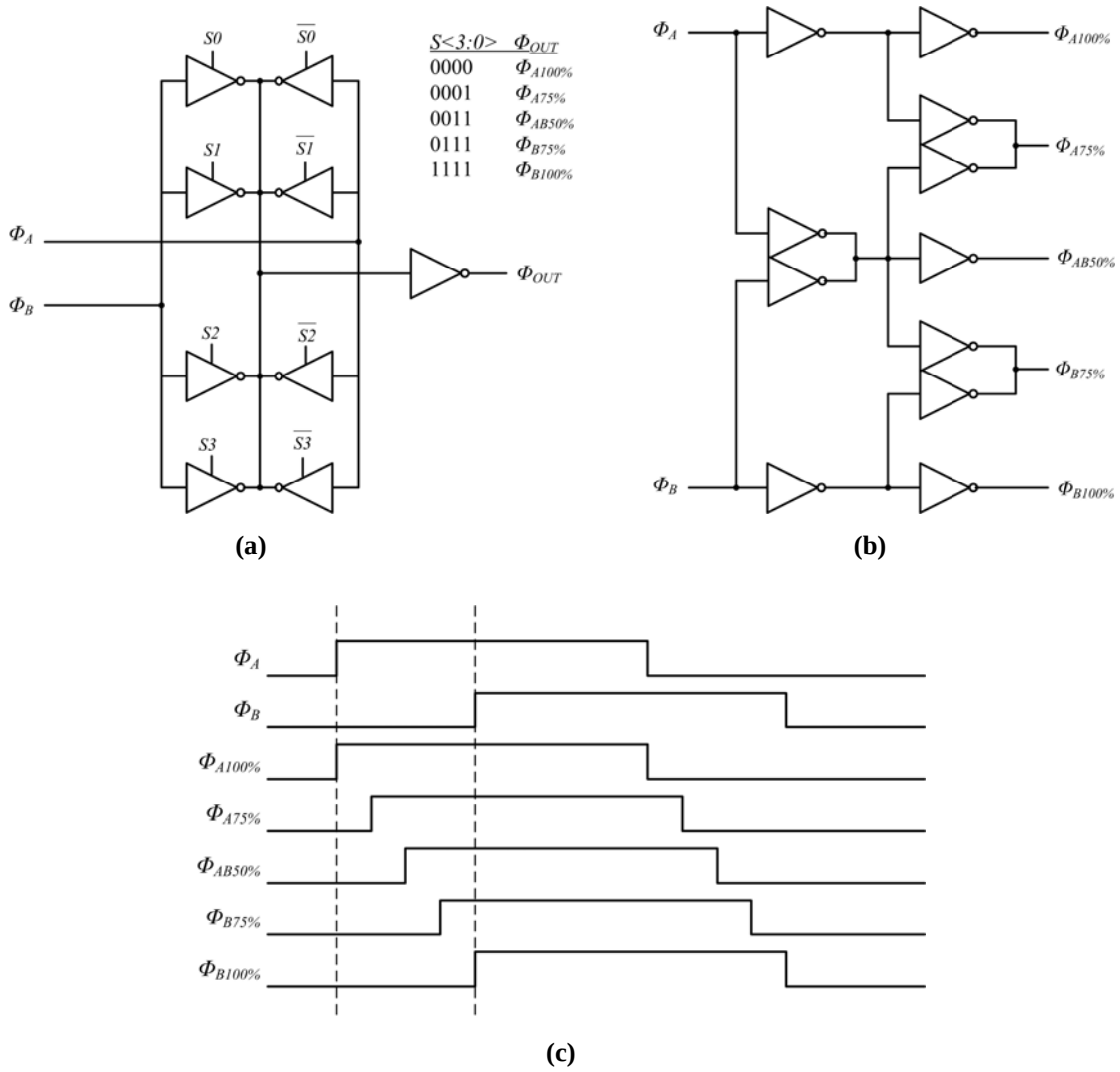



**Figure 31: (a) Single stage multi-weight and (b) multi-stage binary phase mixers with (c) waveforms**

2.2.5 Initialization

Like the analog DLL, a digital DLL is also susceptible to false lock. If the DLL resets with the delay line at minimum delay and the phase detector wants to remove delay, the DLL will never lock. A similar error would occur if the reset condition was for maximum delay and the phase detector wanted to add delay.

There are many ways to initialize a digital DLL to avoid false lock. One technique is to reset to the center of the delay line. This does not require any complex initialization circuitry and ensures that if a lock condition exists, the DLL will eventually find it. However, if there are multiple lock points for a given frequency, the DLL will find the one nearest the center of the delay line. To minimize power supply sensitivity, power consumption, and duty cycle error accumulation, it is best to find the lock point that uses the minimum number of delay elements. One method is to reset to minimum delay and force the DLL to add delay until a positive phase error is detected. This ensures that the 180º boundary has been crossed and allows the DLL to naturally find the lock point of minimum delay. Figure 32 shows a graph of the forward path delay versus clock period when this technique is used. When the DLL is locked, the forward path delay is

$$D1 + [N \cdot t_{CK} - (D1 + D2)] + D2 = N \cdot t_{CK}$$

where $N$ is the smallest integer multiple of $t_{CK}$ that is larger than $D1+D2$. In this example, the IO delay $D1+D2$ = 5ns. The delay below the dotted line represents the IO delay, and the delay above the dotted line represents the delay in the delay line.

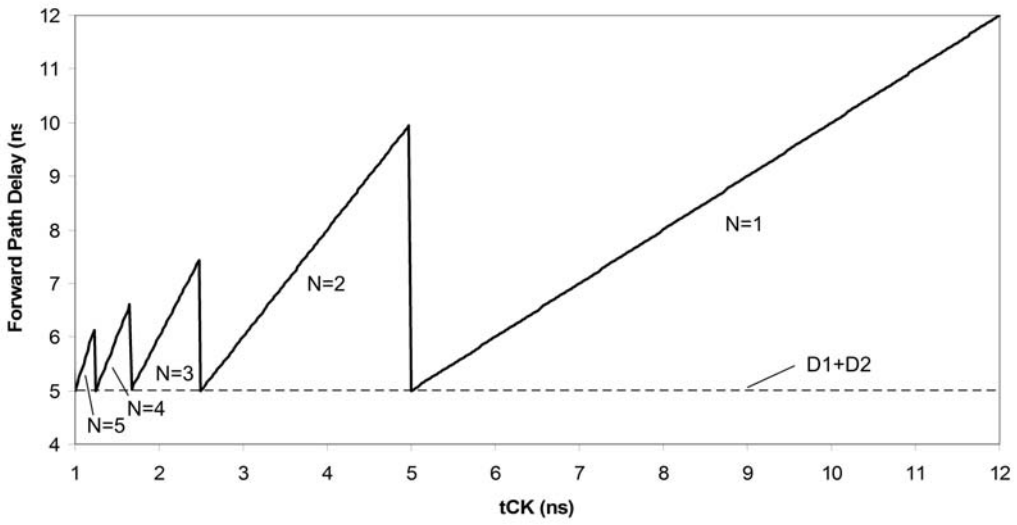



**Figure 32: Digital DLL initialization (N curves)**

The large discontinuity points in these curves do not represent any discontinuity in the performance of the DLL during normal operation. The curves represent only where the DLL will lock when initialized at a given frequency. If the frequency starts to drift after initialization, the DLL will travel up or down the same N curve to track it. For example if the DLL initialized at 4ns $t_{CK}$, and then slowly drifted out to 6ns $t_{CK}$, it would stay on the N=2 curve and slide up to 12ns. An exception to this is if the frequency were to instantaneously change and create a phase error greater than 180°. In that case, the DLL would lose lock on the N that it was tracking and jump to a different N.

Note that when the IO delay is an exact multiple of the clock period, the DLL will initialize with zero delay in the delay line. This is a problem if the temperature, voltage, or frequency shifts, and the DLL needs to remove delay to track. For this reason, the DLL should initialize so that a portion of the delay line is held in reserve. This is called

delay line buffer. The buffer increases power consumption and power supply sensitivity, so it is usually kept as small as possible.

It is interesting to note that the slopes change with N. This is obvious from the equation, but is not intuitive. When N = 2, why must the DLL add 200ps when the period shifts by only 100ps? It is because there are two cycles propagating through the forward path (and the DLL loop). When the period shifts by 100ps, a rising edge at the output is affected by both cycles in the pipeline.

# CHAPTER 3: WIDE RANGE DLL FOR DDR-SDRAM

Double data-rate synchronous dynamic random access memories (DDR-SDRAM) require output data to be aligned with a system clock. To meet the alignment specifications, a clock synchronization circuit is required. A digital DLL is a good choice for several reasons. 1) A clock of the appropriate frequency is provided by a memory controller, so clock synthesis is not required. 2) The alignment specifications of DDR-SDRAM are loose enough that the superior jitter performance of an analog DLL is not necessary. 3) The power savings specifications of DDR-SDRAM require that the clock synchronization circuit retain the lock point even when the clocks are turned off. This is difficult for an analog DLL since the loop filter requires a continuous clock to hold its value. 4) A wide frequency range is required, and a digital DLL can increase low frequency range by simply adding more delay elements. 5) The DDR-SDRAM market is commoditized, making it very competitive, and the exceptional process portability of digital DLLs can increase speed to market.

Currently, three families of DDR-SDRAM have been defined: DDR-I, DDR-II, and DDR-III. A digital DLL is proposed that will meet the requirements of all three. Table 2 lists the performance requirements of the DLL. The frequency range is determined by the minimum frequency of DDR-I (75MHz), and the maximum frequency of DDR-III (1GHz). The alignment requirements allow jitter of approximately ±10% of

the period, but the DLL must be designed to leave margin for non-ideal considerations such as mismatch in the delay model or clock tree across PVT, and power supply or

**Table 2: Performance requirements of DDR-I, DDR-II, and DDR-III**

| | Min | Max | Note |
|---|---|---|---|
| Operating Range | 75MHz | 1GHz | |
| Allowable Jitter | N/A | ±10% of $t_{CK}$ | |
| Lock Time | N/A | 200 cycles | |
| IO Delay | 3ns | 9ns | Estimated values |
| Tracking Time | * | * | See discussion below |

coupled noise that can be induced into the data path. The target for this design was a quantization error no greater than 15% of the allowable jitter window, or 3% of the clock period. The lock time must be less than 200 clock cycles. Tracking time is difficult to define for DDR-SDRAM. After initialization, a change in frequency, temperature, or voltage will create phase error that the DLL must track. Frequency slewing is prohibited, and temperature changes are orders of magnitude slower than a DLLs typical tracking ability, but supply voltage is allowed to vary by up to 10%, and there are no specifications that prevent an instantaneous change in voltage. This is impossible for the DLL to track, but it should be designed to recover as quickly as possible.

Table 3 shows the parameters used for the PVT corners. The maximum voltage for DDR1 and DDR2 are outside of the range for this process node, requiring that the internal voltage be regulated down to at least 1.7V. The minimum voltage of DDR3 is 1.425V, so 1.2V is used to provide at least 200mv of margin at the slow corner.

**Table 3: PVT values used for slow, typical and fast corner simulations**

| | Process | Voltage | Temperature |
|---|---|---|---|
| Slow Corner | N/P Slow | 1.2V | 110ºC |
| Typical Corner | N/P Typical | 1.5V | 25ºC |
| Fast Corner | N/P Fast | 1.7V | -40ºC |

Figure 33 shows the block diagram of the DLL that was developed in this thesis. The target quantization error cannot be achieved using discrete devices only, so a dual loop topology was chosen using a graduated coarse delay line and a phase interpolating fine delay line. A fast initialization circuit allows lock time in less than 200 cycles, and an N detect circuit and dynamic digital sample filter allow fast tracking time. Each component of the DLL is discussed in detail.

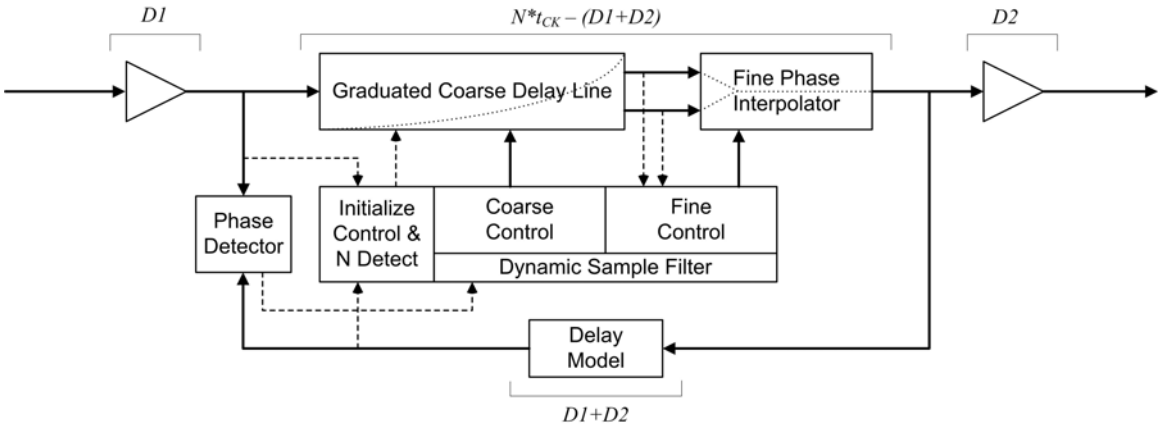



**Figure 33: Block diagram of the proposed DLL**

### 3.1 Graduated Coarse Delay Line

The coarse delay line is made up of NAND-NAND delay cells, a shift register and exit control circuitry. It is called a graduated delay line because the channel lengths of the delay elements – and therefore the delay per stage – increase with delay line depth. This increases the total delay of the delay line without increasing the number of delay elements. The trade-offs associated with a graduated delay line are discussed in detail.

Figure 34 shows a block diagram of the coarse delay line. Each delay cell consists of a shift register cell, a delay element, and exit control logic. Because the fine delay line is a phase interpolator, it requires two inputs. To avoid control and delay discontinuity, only one input should change at a time. To accomplish this, there is one exit point for even taps and one for the odd taps. The exit control enables the delay taps

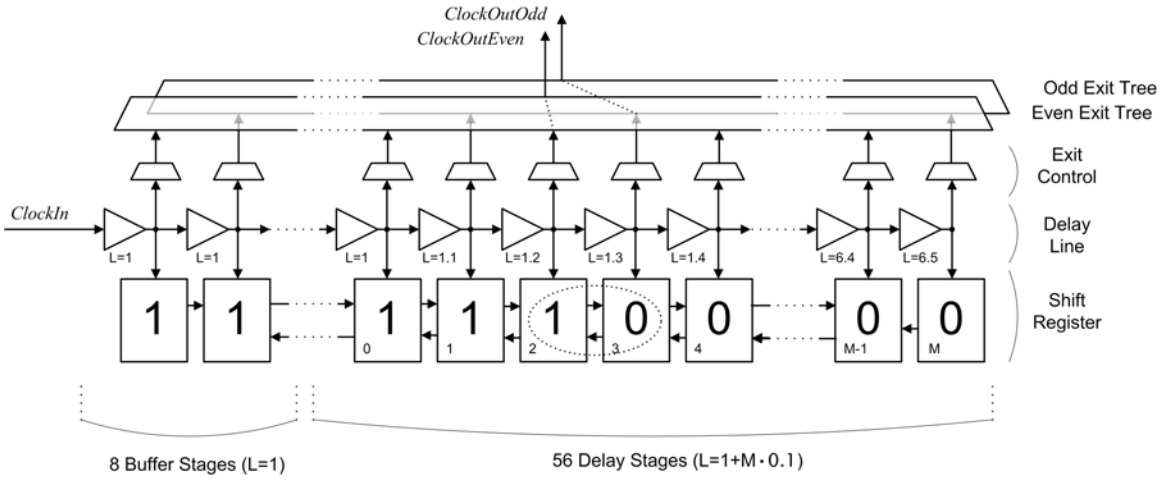



**Figure 34: Block diagram of the graduated coarse delay line**

at the 1→0 boundary. When the delay line shifts, this boundary moves, but only one of the outputs will change. To ensure that this boundary always exists, the left and right-most register cells are hard-tied to 1 and 0 respectively.

Figure 35 shows a detailed diagram of a delay cell. NANDs A and B are the delay elements. The second input to NAND A is the enable control and receives the value of the register on the left. This ensures that the delay cells up to and including the 1→0 boundary are toggling. NAND C enables direct loading of the shift register from the delay cell. This is used for fast initialization. It will be described in detail later, but it essentially turns the delay line into a stopwatch. After resetting the register to all zeros,

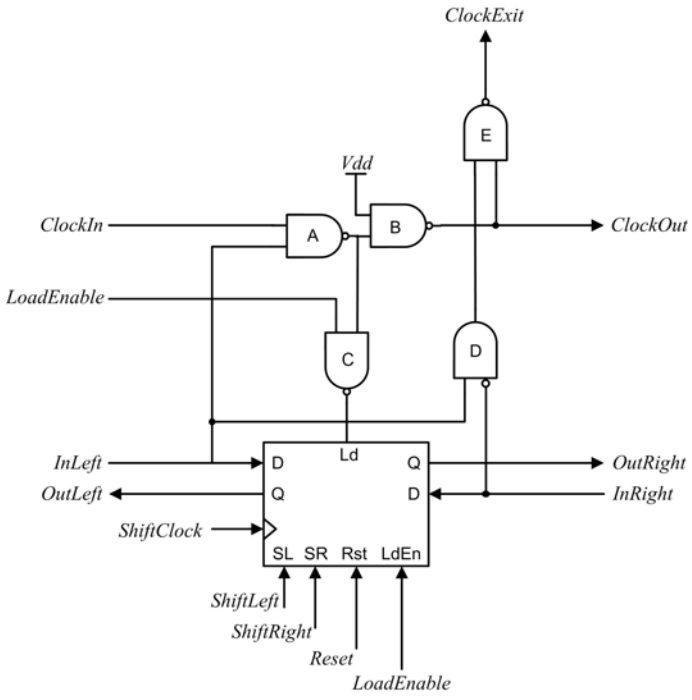



**Figure 35: Graduated coarse delay line delay cell**

the rising edge of *ClockIn* becomes a start signal, and the rising edge of *ShiftClock* becomes a stop signal. The number of ones in the register represents how far the signal propagated through the delay line from start to stop. Logic gates D and E are the exit control. The exit path is enabled only when the register cells on the left and right are 1 and 0 respectively – this is true only if the cell is on a 1→0 boundary. The inputs *ShiftLeft*, *ShiftRight* and *LoadEnable*, enable the register to be loaded on the rising edge of *ShiftClock* from the left, right, or delay cell inputs.

**Table 4: Delay cell device sizes**

| | **Buffer (First 8 Stages)** | | **Delay Stage *M*** | |
|---|---|---|---|---|
| | **W** | **L** | **W** | **L** |
| **NAND A** | 40 | 1 | 40 | $1 + M \cdot 0.1$ |
| **NAND B** | 40 | 1 | 40 | $1 + M \cdot 0.1$ |
| **NAND C** | 40 | 1 | 40 | 1.1 |
| **NAND E** | 40 | 1 | 40 | 1 |

The sizes of NANDs A, B, C, and E are shown in Table 4, where *M* is the number of delay stages beyond the last buffer stage. These sizes are critical to the delay line performance. It was discussed earlier that the capacitive loads must be matched at both the inner and outer nodes of each delay element or a P/N drive mismatch will cause duty cycle error accumulation. Duty cycle performance is critical in a DDR-SDRAM because data is sampled on both the rising and falling edges of the clock. It might seem that if A matches B, and C matches E, the capacitive loads will be the same. However, in a graduated delay line, the outer node will see an extra $W \cdot \Delta L \cdot Cox'$ from the graduation

value of the next stage. To compensate, all four devices must have the same widths, and the length of device C must be greater than device E by the graduation value $\Delta L$ .

Figure 36 shows the duty cycle performance of the delay line with and without load matching. When N and P drives are not matched, duty cycle error begins to accumulate at the point of graduation because of second order effects that cannot be compensated for by load matching (Vt variation, capacitive coupling, etc…). However, load matching keeps the error symmetrical about the typical corner, preventing the error terms from summing as seen in the slow N fast P corner.

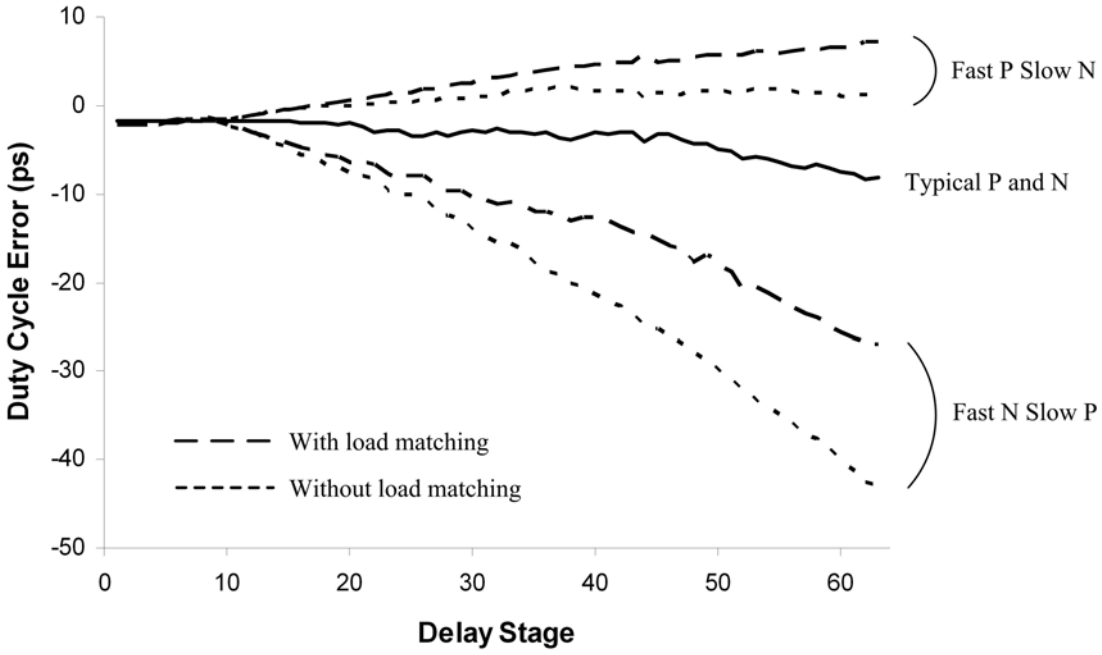
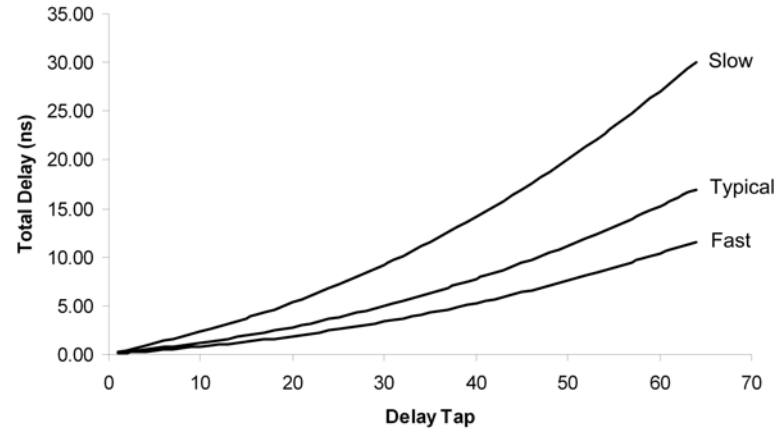
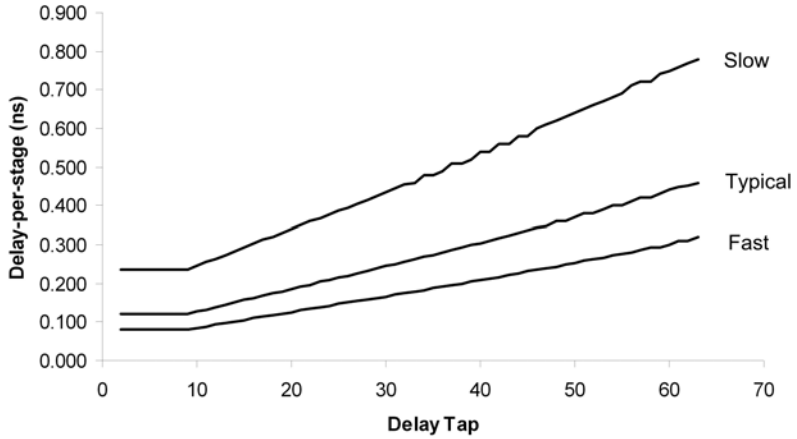



**Figure 36: Duty cycle performance of the graduated delay line**

This is a deep submicron process, so scaling theory says that – because of mobility saturation of the charge carriers – the drain current is not affected by small variations of L [5]. The capacitive load of each delay element will increase linearly with L, so the delay per stage of the graduated delay line increases linearly with delay line depth, and total delay increases exponentially. Figure 37 shows the total delay and delay per stage transfer curves for the delay line based on slow, typical and fast process corner simulations. The delay per stage is constant for the buffer stages, and then begins to increase at a constant rate.



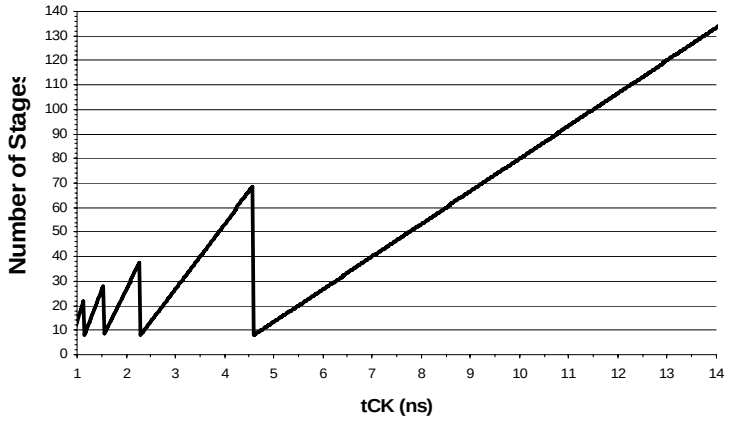

**(a)**



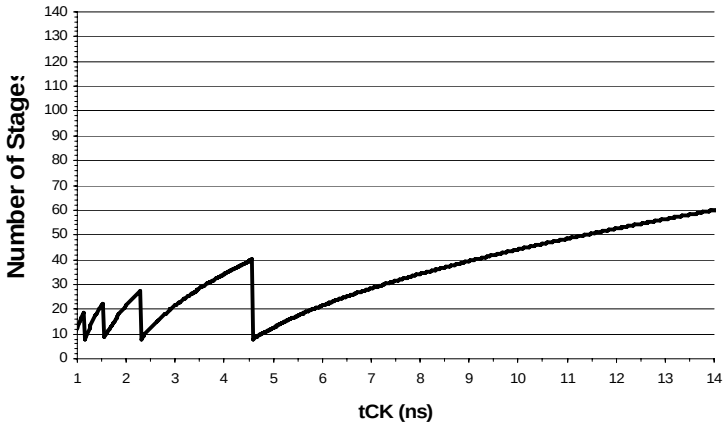

**(b)**

**Figure 37: (a) Total delay and (b) delay per stage versus delay line depth**

The benefit of a graduated delay line can be seen by analyzing the initialization curves discussed earlier. The graphs in figure 38 are similar to the lock point curves shown in figure 32, but the Y axis is number of delay stages instead of total delay. Using




**(a)**




**(b)**

**Figure 38: Lock point curves of a (a) typical and (b) graduated delay line**

the minimum delay per stage of 75ps – taken from the fast corner simulation, and the minimum predicted IO value of 3ns, the maximum number of taps can be determined. The data shows that a conventional delay line would require over 130 delay stages to cover the full operating range. Even though the graduation value is only 3.75ps at the fast corner, the graduated delay line reduces the number of required delay stages by a factor of two. This is a significant savings in layout area because the delay line is usually the largest component of a digital DLL.

Besides the duty-cycle distortion, there is another performance sacrifice involved with using a graduated delay line; the quantization error increases with delay line depth. This is acceptable because at higher frequencies – where delay line depth is shallow – the penalty will not be as severe as at the low frequencies where a larger quantization error can be tolerated. The maximum expected quantization error is now based on delay line depth. The delay line depth – accounting for the buffer stages – is given by

$$[N \cdot t_{CK} - (D1 + D2)] + t_{BUF}$$

The IO delay will vary based on data path design and PVT, so it must be assumed that at any given $t_{CK}$, the maximum delay line depth will be $t_{CK} + t_{BUF}$. However, the buffer stages are not graduated, and therefore do not contribute to an increase in quantization error. The maximum quantization error can then be defined as the delay per stage when the delay line depth is equal to $t_{CK.}$ The following equations define the delay line depth $t_{DT}$ and delay per stage $t_D$, by the number of stages $X$, the delay of the initial delay element $t_{Do}$, and the graduation value $\Delta t_D$.

$$t_D = t_{Do} + X \cdot \Delta t_D$$

$$t_{DT} = X \cdot t_{Do} + \frac{X(X-1)}{2}\Delta t_D$$

Solving for $t_D$ in terms of $t_{DT}$ gives

$$t_D = \sqrt{(t_{Do} - \tfrac{1}{2}\Delta t_D)^2 + 2 \cdot \Delta t_D \cdot t_{DT}}$$

Substituting $t_{CK}$ for $t_{DT}$ gives the maximum quantization error $t_E$ as a function of $t_{CK}$.

$$t_E = \sqrt{(t_{Do} - \tfrac{1}{2}\Delta t_D)^2 + 2 \cdot \Delta t_D \cdot t_{CK}}$$

Figure 39 shows the graph of the error function at slow, typical and fast PVT corners. The slow corner is worse for quantization error, and compared to the target of 3% of the clock period, the worst case $t_{CK}$ is 1ns, exhibiting nearly 10 times the target

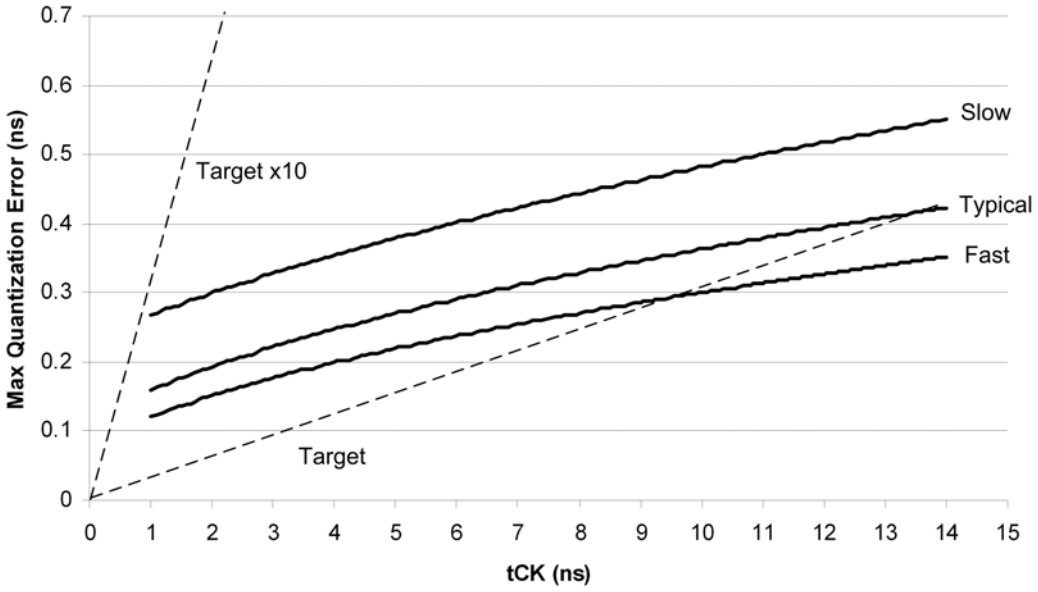



**Figure 39: Quantization error versus $t_{CK}$**

value. However, the coarse delay line alone was not meant to reach that target. The fine delay line will allow the target to be met so long as it has a resolution of at least $\frac{1}{10}$ of a coarse delay.

## 3.2 Phase-Interpolating Fine Delay Line

Figure 40 shows the schematics of the fine delay line. It was implemented using a single stage multi-weighted phase mixer. A phase mixer was chosen for this design because it will scale with the variable delay per stage of the graduated coarse delay line.

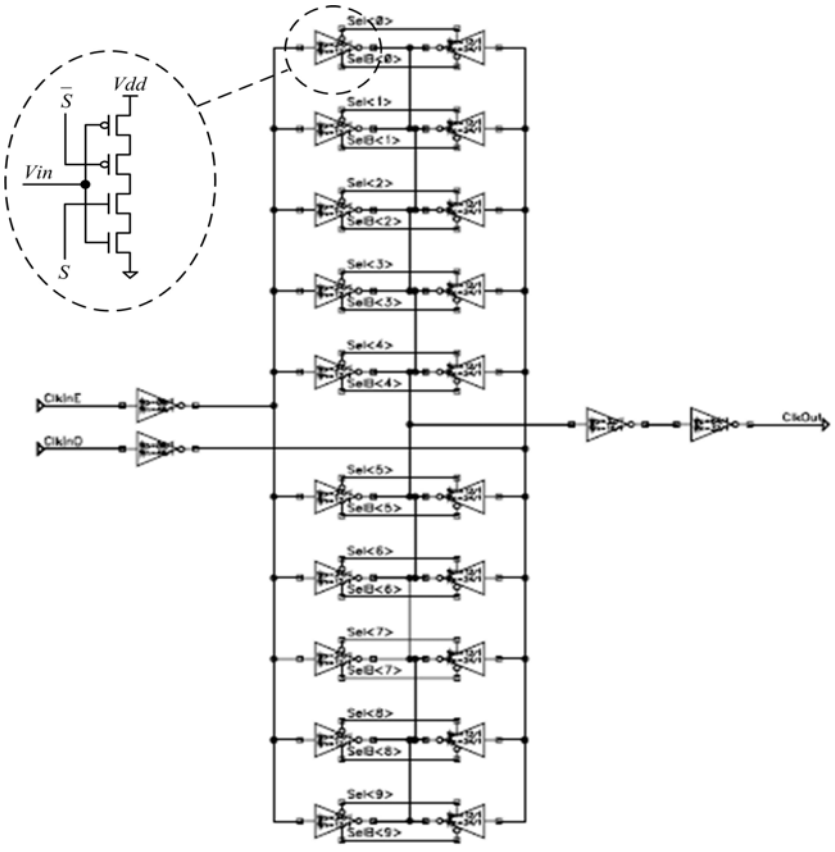



**Figure 40: Phase mixing fine delay line schematics**

A fine delay line with a fixed delay value – such as a capacitor based delay line – would create large delay discontinuity. Ten pairs of tri-state inverters were needed to achieve the required resolution. The N and P devices were sized at 48/1 and 24/1 respectively. The schematics are shown in. A 10-bit shift register (not shown) provides the thermal-coded control word for the tri-state inverters.

Figure 41 shows simulation results at the typical PVT corner with the input phases spaced 200ps apart. Ideally, the output phases should be spaced by 20ps, but they range from 10ps to 40ps. This non-linearity is proportional to the phase difference of the inputs and is one of the weaknesses of this circuit. However, it can be tolerated because the phase difference of the inputs is smaller for high frequencies where the delay line depth is shallow. Other drawbacks include relatively high power consumption due to the deliberate contention created at the common output node and duty cycle sensitivity to N and P drive mismatch.

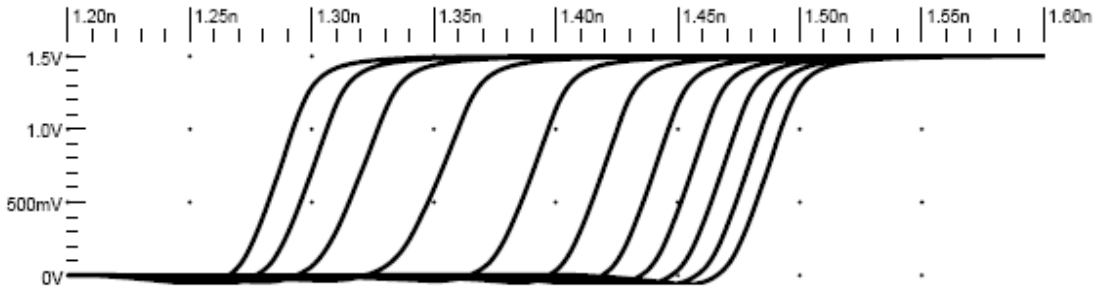



**Figure 41: Phase mixing fine delay line waveforms**

### 3.3 Initialization Control

The DDR-SDRAM specification requires the DLL to lock within 200 cycles of reset. The delay line has 640 effective phase placements (64 coarse * 10 fine), so it would take too long to let the DLL shift to the lock point. A fast initialization circuit is required. The fast initialization uses the stopwatch method mentioned earlier to set the initial value of the coarse delay line near the lock point. The number of cycles in the loop *N* is captured during the stopwatch measurement, which is used by the dynamic filter to set the DLL wait time (the time between shifts). A simplified block diagram and waveforms of the fast initialization scheme are shown in figure 42.

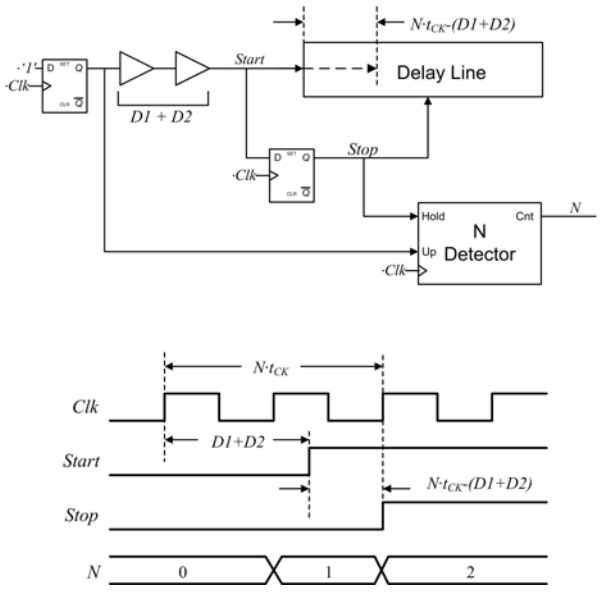



**Figure 42: Initialization and N-detect simplified block diagram and waveforms**

The first rising edge of *Clk* sets the leftmost D flip-flop which enables the N-detector and sends a rising edge into the delay model. The *Start* signal begins to

propagate through the delay model while the N-detector counts the clock cycles. Once the signal has propagated through the delay model, it enables the second D flip-flop and begins propagating through the delay line. On the next rising clock edge, *Stop* captures the delay line depth setting the delay to $N \cdot t_{CK}$ - *(D1+D2)* and disables the counter preserving the value of *N*.

To eliminate the need for extra logic, and to ensure that the measurement is as accurate as possible, the actual DLL loop is used for the measurement. Figure 43 shows a detailed diagram including the intrinsic delays that affect measurement accuracy.

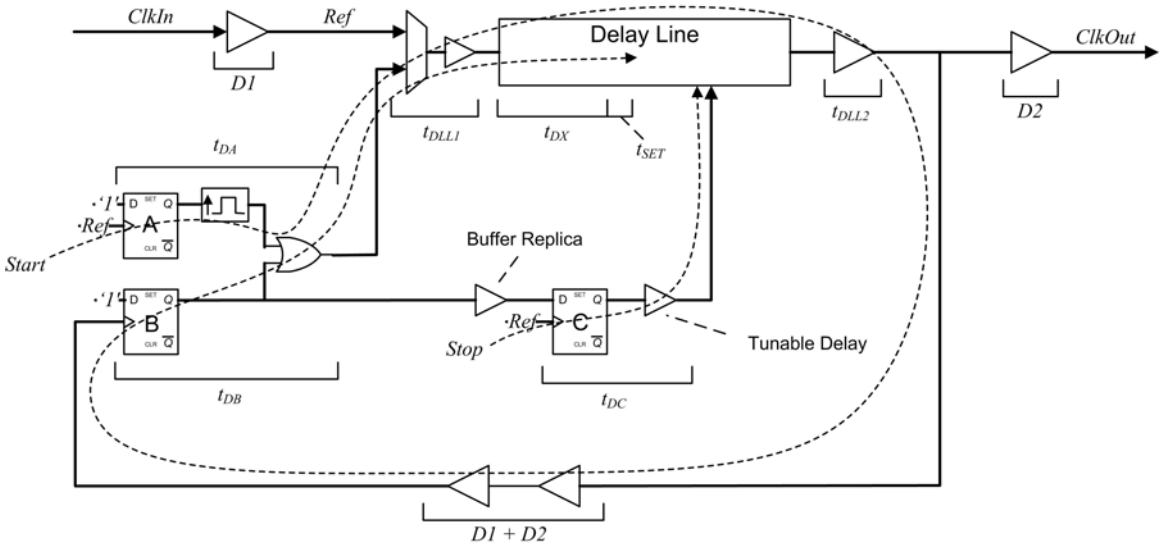



**Figure 43: Initialization circuit detailed block diagram**

Several things happen before the initialization sequence begins: the MUX in front of the delay line selects the lower input instead of *Ref*, the delay line is set to minimum delay, and flip-flops A, B and C are held in a reset state long enough to ensure that the

DLL loop is clear of all old clock cycles. Once the flip-flops are enabled, the start sequence begins when the next rising edge of *Ref* arrives at flip-flop A. This sends a pulse around the loop measuring the DLL intrinsic delay $t_{DLL1}$ and $t_{DLL2,}$ and the IO model *D1+D2*.

When the pulse arrives at flip-flop B, it sends a rising edge into the delay line and through the buffer replica delay to flip-flop C. The buffer replica delay ensures that the edge propagates through the delay line buffer before the stop signal can be asserted. In other words, the buffer replica will increase *N* if the DLL attempts to initialize the delay line in the buffer. On the next rising edge of *Ref*, the stop signal is sent to the delay line through an adjustable delay $t_{DC}$. Figure 44 shows simulation results of the waveforms during initialization.

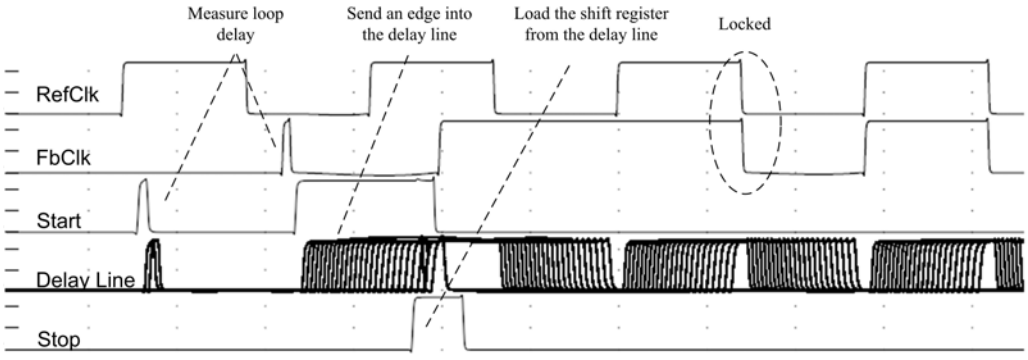



**Figure 44: Waveforms during initialization**

After initialization, the delay $t_{DX}$ in the delay line will be equal to

$$Stop - Start - t_{SET}$$

$$\Rightarrow (N \cdot t_{CK} + t_{DC}) - (t_{DA} + t_{DLL1} + t_{DLL2} + D1 + D2 + t_{DB} + t_{DLL1}) - t_{SET}$$

The desired $t_{DX}$ delay is

$$N \cdot t_{CK} - (t_{DLL1} + t_{DLL2} + D1 + D2)$$

To cancel out the error terms caused by intrinsic delay and the setup time of the shift register, the tunable delay $t_{DC}$ must equal to

$$t_{DA} + t_{DLL1} + t_{DB} + t_{SET}$$

Because the timing of this circuit is so sensitive, it is important to check the operation across all PVT and frequencies. Figure 45 shows the measured phase error immediately after initialization across $t_{CK}$ at the slow, typical and fast PVT corners. If the measurement path is tuned properly, the maximum error should be the quantization error of the coarse delay line. That explains why the error distribution has a wider spread at slow $t_{CK}$ than at the faster $t_{CK}$. The initialization sequence takes about 30 cycles, leaving 170 cycles for the DLL to dial in on the lock point.

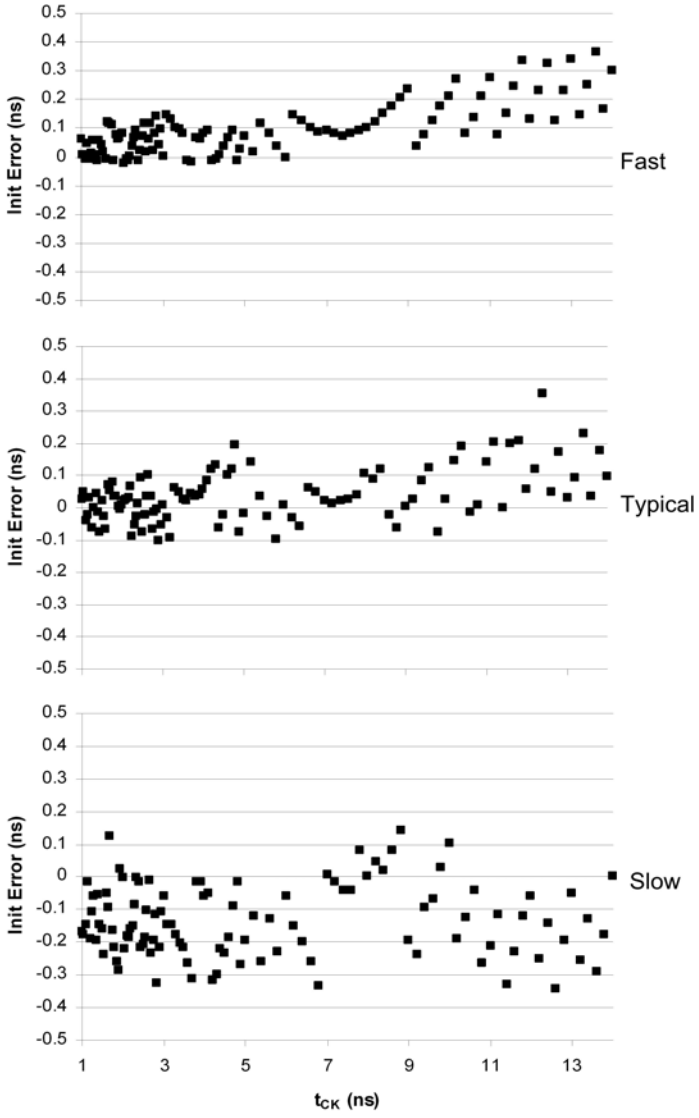



**Figure 45: Phase error after initialization**

### 3.4 N-Detector and Dynamic Sample Filter

The N-detector serves two purposes. First, it holds off the init sequence until the feedback loop can clear out all of the old clock cycles. Second, it detects and stores the N value that is used by the dynamic sample filter. Initially, the N-detector was implemented using a 4-bit binary counter. That topology had a smaller gate count and used less power, but the time required to decode the output and reset the counter was more than 1ns at slow PVT creating a possible race condition. The circuit was redesigned using shift registers. The topology is shown in figure 46.

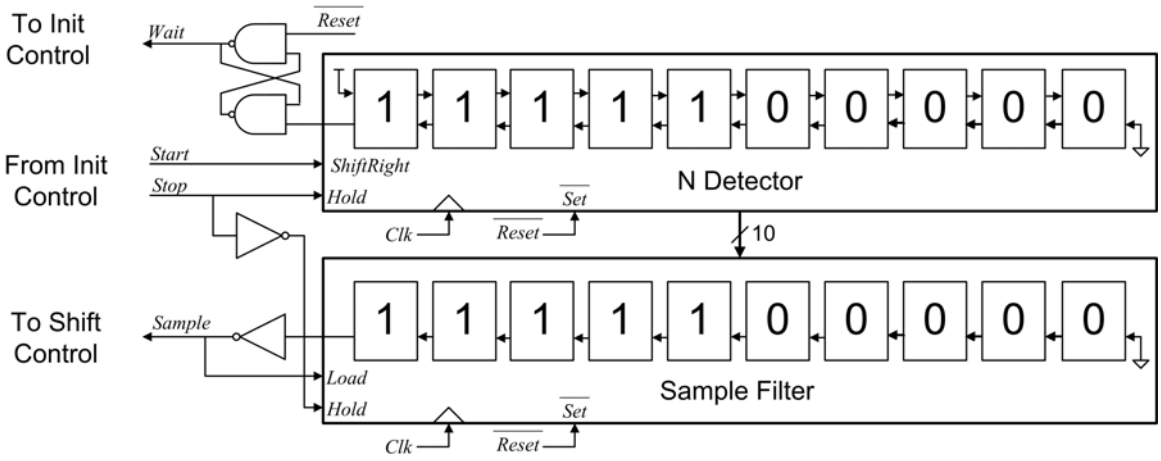



**Figure 46: N-detector and dynamic sample filter block diagram**

Both the N-detector and sample filter consist of a 10 bit shift register, but they differ slightly in their controls. The N-detect register can shift left or right, while the sample filter can shift left or load an external value. The waveforms in figure 47 show the simulated operation of the sample filter and N-detector. Upon reset, both registers

initialize to all ones, and the NAND set-reset latch is set. This asserts *Wait* which holds off the initialization sequence by keeping *Start* and *Stop* low. Because *Start* is low, the N-detector will shift left until a zero is output to the NAND set-reset latch. This clears the latch and allows the initialization sequence to begin. These 10 cycles of hold time are required to ensure that any clocks propagating through the feedback path have cleared out before the initialization sequence begins. Since the N-detector has already pre-loaded itself with zeros, it is ready to begin counting the number of cycles between *Start* and *Stop*. Once *Start* is asserted, it will begin shifting right on every rising clock edge until the *Stop* signal is asserted.

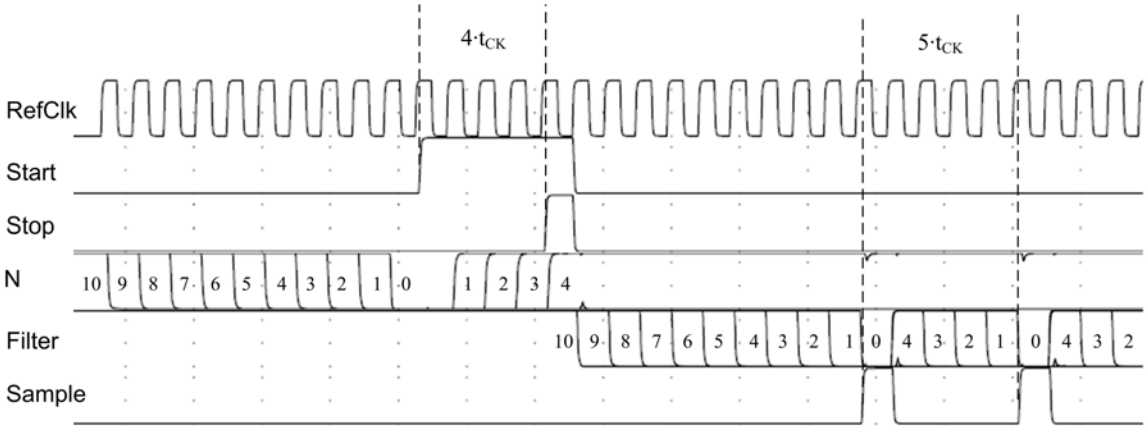



**Figure 47: Waveforms showing the operation of the N-detector and dynamic sample filter**

When *Stop* goes high, it allows the sample filter to begin shifting left. This will take a full 10 cycles, allowing the feedback path to clear out from the initialization sequence and populate with clocks. When a zero is output from the sample filter, it

triggers a sample of the phase detector and asserts the *Load* signal. On the next rising edge, the value from the N-detect register will be loaded directly into the sample filter. This clears the load signal and allows the register to begin shifting left. After N consecutive cycles, it will output another sample and repeat the process so that a sample is triggered every N+1 cycles (N cycles of shifting plus one cycle of sample/load time).

This topology requires very little sequential logic, allowing it to run extremely fast with no race conditions even at the slowest PVT corner, but it is expensive in terms of layout area and power consumption. The graphs in figures 48 through 50 show the lock point and N-detector value versus $t_{CK}$ across all PVT corners. A preliminary design of the N-detector required it to use the same inputs as the flip-flops that controlled the start and stop signal. This created discontinuities at the lock point boundaries if the setup times of the shift register and flop-flops were not perfectly matched. The new topology – using the output of the flip-flops – shows continuity across all PVT and frequency, and is not susceptible to process induced setup and hold mismatch.

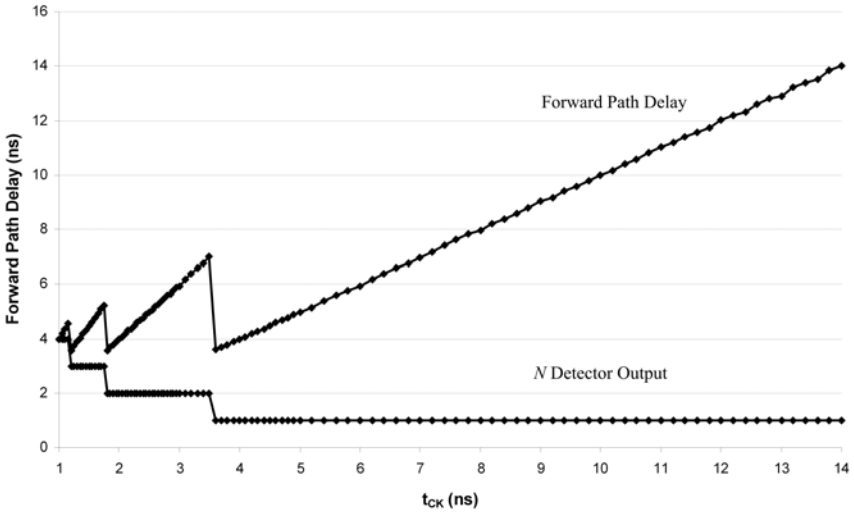



**Figure 48: Lock point curves and N-detector output at fast corner**

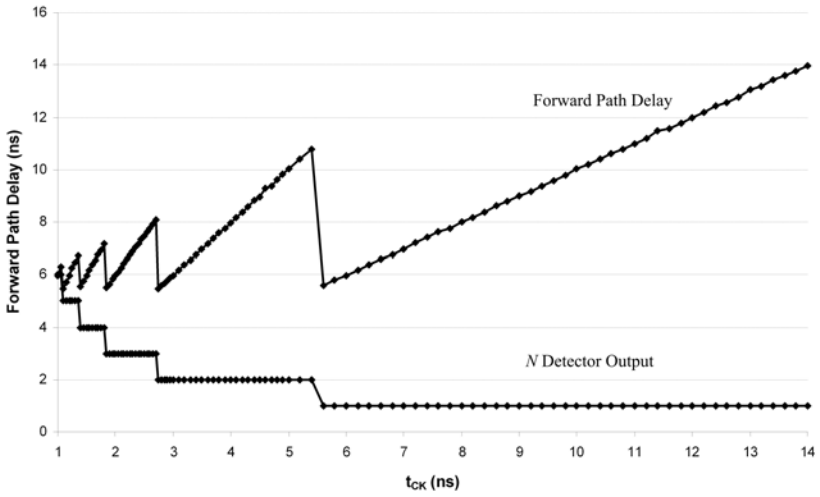



**Figure 49: Lock point curves and N-detector output at typical corner**

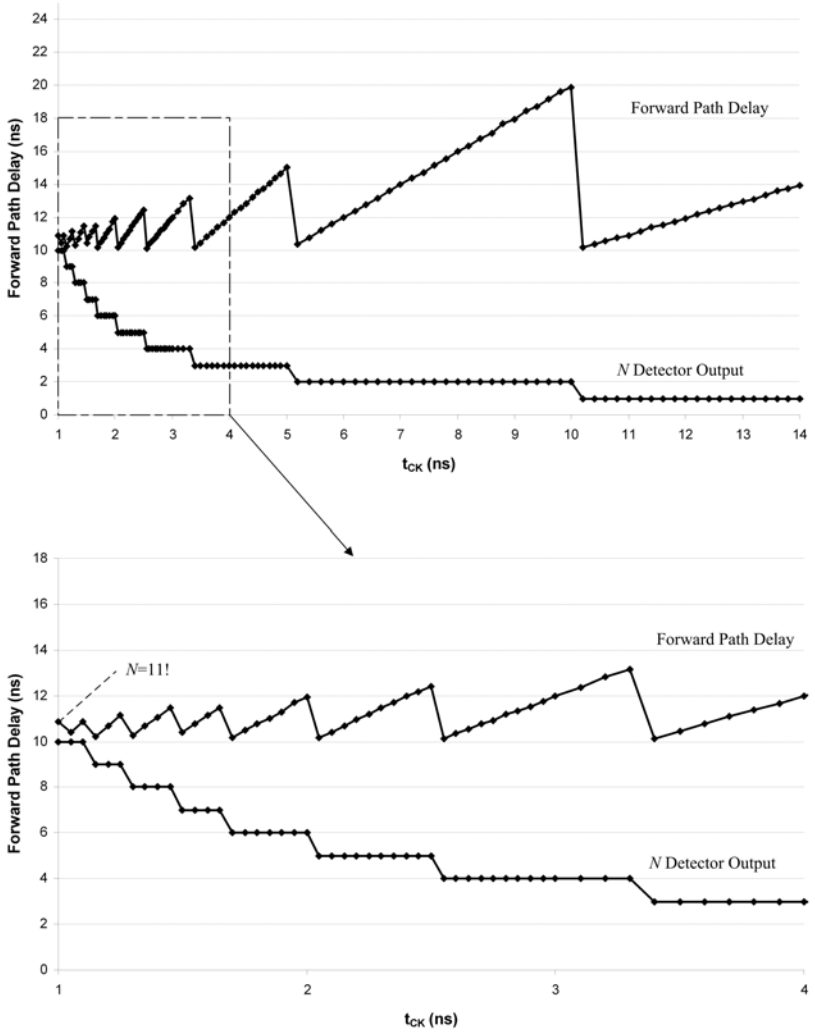



**Figure 50: Lock point curves and N-detector output at slow corner**

Figure 50 shows that at the slow corner, 1ns $t_{CK}$, there are 11 cycles in the forward path, but the N-detector is stuck at the maximum value of 10. This is because the delay through the delay line buffer is 2ns at this corner. That was not taken into account when the maximum value of N was determined. The topology allows N to be increased simply by increasing the size of the shift registers. However, that is not necessary. The buffer

increases the measured loop delay, but it does not contribute to the delay from shift to new information at the phase detector. The fine delay line is after the coarse, and the coarse delay line uses an exit point topology. This means that when a shift occurs, the new phase information does not have to propagate through the buffer.

### 3.5 Phase Detector

The phase detector used in this design is a D flip-flop phase detector. The schematics are shown in figure 51. It is comprised of three NAND set-reset latches, one master and two slaves. The bottom NAND is for load matching, and the two inverters minimize the load and increase the gain of the master set-reset latch.

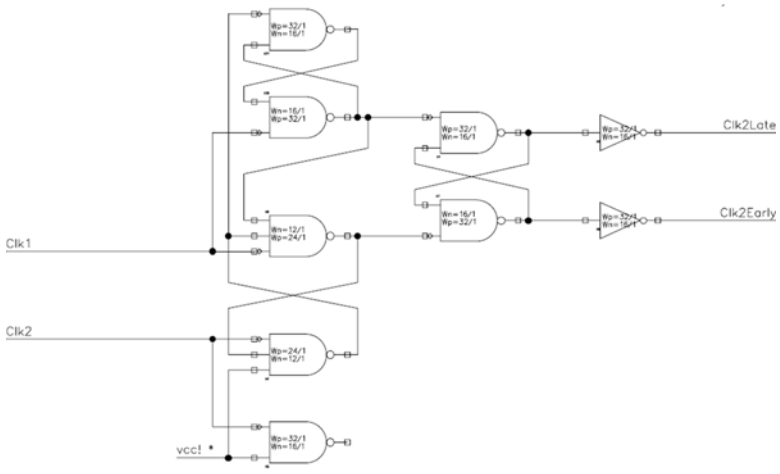



**Figure 51: Schematic of the phase detector**

Hysteresis was not implemented because the nature of the graduated delay line makes it difficult to realize. If the hysteresis delay was set to the minimum delay per

stage, the DLL would oscillate when the DLL is locked deep. If it were set at maximum delay, it would hurt the performance when the delay line was locked shallow. A replica delay line could be used to set the hysteresis delay, but the increase in layout area and power consumption makes this solution impractical.

### 3.6 Shift Control

A shift is performed every time that the phase detector is sampled. The shift control logic determines the direction of the shift and whether it will be applied to the coarse or fine delay line. The schematics are shown in figure 52.

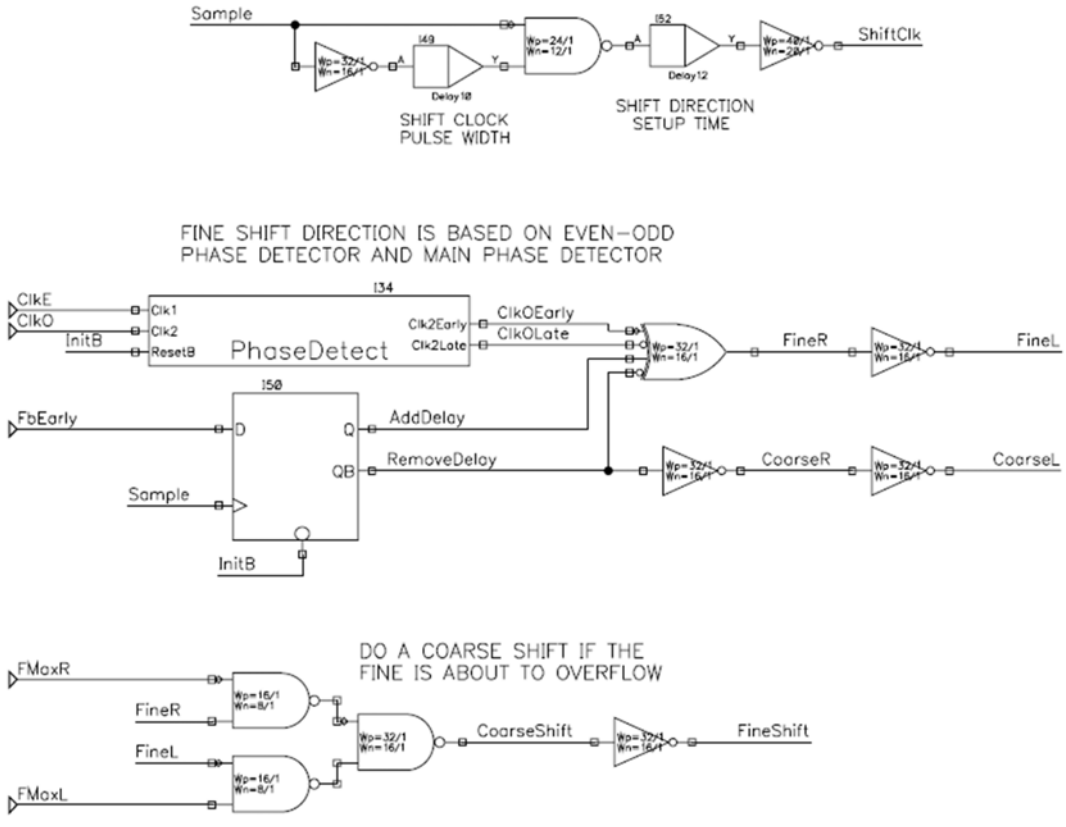



**Figure 52: Schematics of the shift control logic**

When a sample is generated by the filter, the signal *FbEarly* from the main phase detector – indicating whether to add or remove delay – is latched by the D flip-flop. The coarse direction is determined directly from the output of the flip-flop. A shift right will add delay and a shift left will remove it.

The fine direction is not as straightforward. It cannot be determined directly from the main phase detector because the phase mixer does not know which input is early and which is late. This can be determined by decoding the register setting of the coarse delay line, but it is a large register and would require too much logic. Instead, a local phase detector is used to detect which output from the coarse delay line, *ClkE* or *ClkO*, is early. The fine shift direction is determined by an XOR function between the main phase detector and the local phase detector. A shift right is performed to add weight to the phase of *ClkO* if the phase detector wants to remove delay and *ClkO* is early, or if the phase detector wants to add delay and *ClkO* is late. Otherwise a shift left is performed.

The logic on the bottom determines whether a fine or coarse shift is required. The inputs *FMaxR* and *FMaxL* are flags from the fine shift register that are set when the register hits the right or left end. A coarse shift is only done if a fine shift would result in an overflow. The logic on the top creates the shift clock. The pulse generator sets the width of the shift clock pulse, and the delay on the output of the generator creates setup time for the shift direction.

# CHAPTER 4: CONCLUSION

This thesis discussed common topologies for clock synchronization. Analog and digital DLL implementations were discussed in detail, paying special attention to the effects of a large delay model in the feedback path. Design considerations were discussed and supported with simulation results. A digital DLL was presented that used a graduated digital delay line to cover the full operating range of all currently defined DDR-SDRAM families.

To measure the performance of the DLL, the circuit was simulated across PVT corners and over the full operating range. The DLL was given exactly 200 cycles to lock. The next 150 cycles of data were captured, and the DLL output was analyzed for jitter. This allowed both lock time and jitter to be measured simultaneously; because if the DLL did not lock in time, it would show up as increased jitter.

It is generally accepted that the jitter transfer function of a DLL is unity. A DLL simply adds delay, so jitter on the input clock is passed directly through to the output. The total jitter is the input jitter plus the jitter induced by the DLL. However, it has been shown that jitter can be amplified in a DLL under certain conditions [19]. This is mainly a concern for applications where multiple DLLs are cascaded. A full treatment of jitter transfer and phase noise suppression is beyond the scope of this paper. The simulations were run without any input jitter, and jitter amplification will be lumped with other non-ideal considerations for which the DLL has been designed to provide margin.

Figure 53 shows simulation results at the typical process corner, 2ns $t_{CK}$. The waveforms show 24.4ps of maximum jitter, and the histogram below shows the jitter grouped into two nearly equal groups spaced approximately 20ps apart. This is consistent with the expected results. The two groups represent two discrete delay steps that the DLL is oscillating between. Assuming that the delay model is tuned properly, the ideal clock output will lie somewhere between the two steps, and the maximum possible error is 24.4ps.

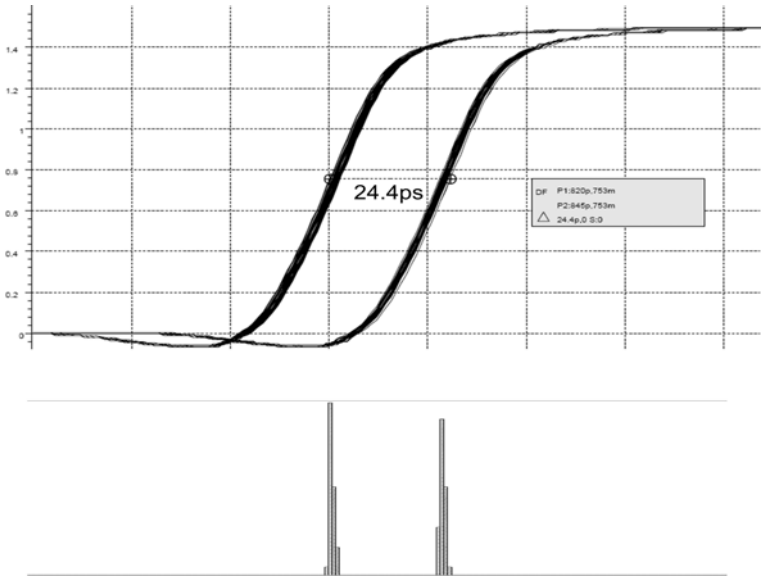



**Figure 53: Jitter waveform and histogram at typical corner, 2ns $t_{CK}$**

Figure 54 shows the simulated jitter values across PVT and frequency. The target of 3% of $t_{CK}$ is achieved except at 1ns. At $t_{CK}$ above 2ns, the DLL provides plenty of margin to the target. However, at 1n, slow corner, the DLL exhibited jitter of 50ns compared to the target of 30ns.

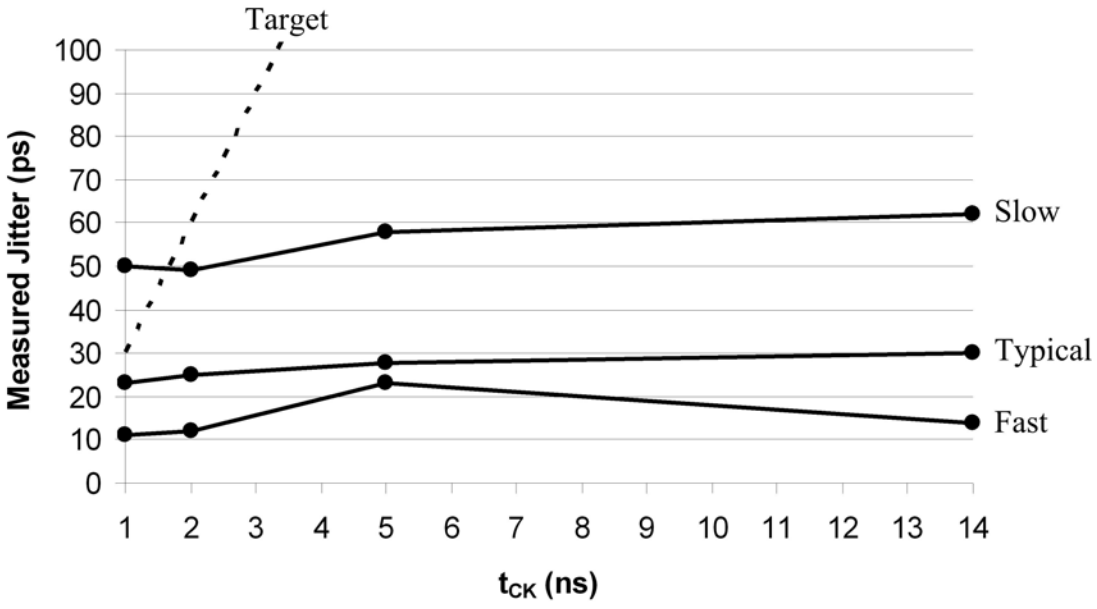



**Figure 54: Maximum jitter versus $t_{CK}$**

Figure 55 shows the jitter waveform and histogram at 1ns $t_{CK}$, slow corner. Three jitter bands are evident instead of two, implying that the DLL is actually oscillating between three delay steps. If these were real measurements taken from a physical circuit on a DRAM, this would not be surprising because of the noisy environment, but in an ideal simulation environment this was not expected. One possibility is that the wait time was too short. This would cause a second shift to be issued before the phase information from the first shift made it around the loop.

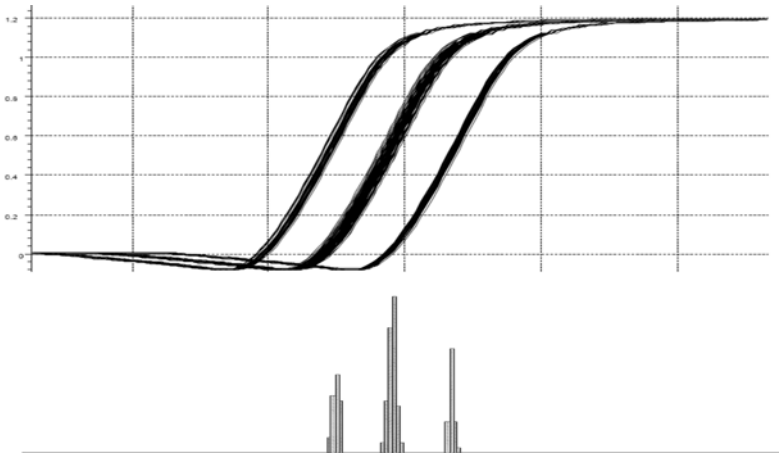


**Figure 55: Jitter waveform and histogram at slow corner, 1ns $t_{CK}$**

Tracing the signal around the loop showed that the new phase information was actually making it around the loop with time to spare, but the phase detector was not changing state. It was not sensitive enough to detect a phase difference of a single unit delay at the slow corner, so it was forcing two consecutive shifts in the same direction.

Figure 56 shows an alternate phase detector with improved accuracy. The core of this phase detector is the arbiter circuit described in the digital DLL section. It is more accurate than the D flip-flop for two reasons. First, there is no setup versus hold issue, because the paths to the output are identical for both inputs. Second, the arbiter core itself has no "memory"; the previous state is cleared as soon as both inputs go low. A flip-flop based phase detector retains its previous state making it more likely to repeat the same decision.

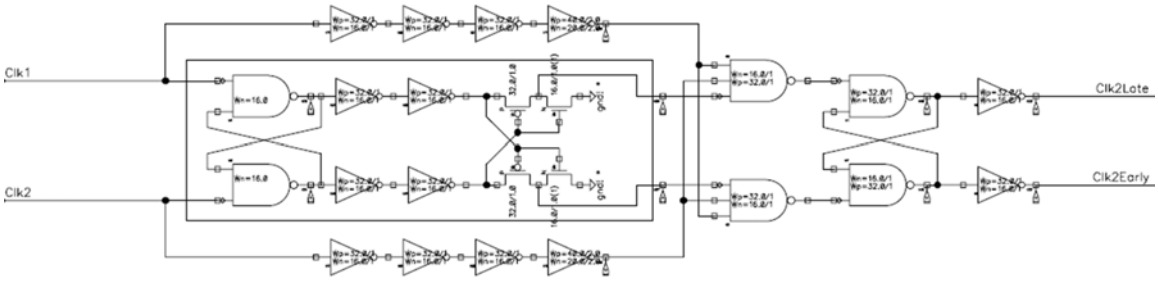



**Figure 56: Arbiter based phase detector schematic**

The extra inverters inside the arbiter core increase the gain of the NAND latch and provide protection from capacitive coupling back into the latch. The extra circuitry outside of the core is used to latch the state of the arbiter so that the output is valid for a full clock cycle. Using the new phase detector, the DLL behaved as expected, and the jitter target of 30ps was achieved at the slow corner 1ns $t_{CK}$ (see figure 57).

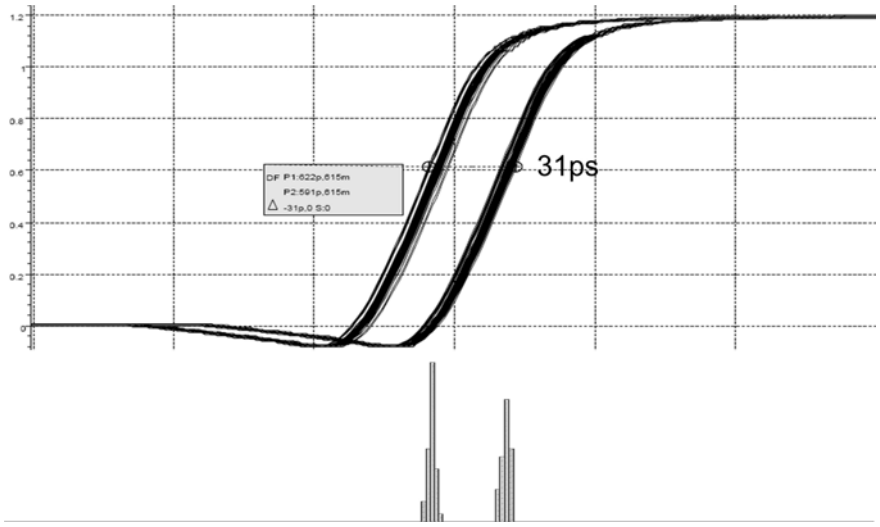



**Figure 57: Jitter waveform and histogram at slow corner, 1ns $t_{CK}$ with new phase detector**

Although the design provides margin for non-ideal operating conditions, it is important to understand how noise will affect the performance of the DLL. Figure 58 shows the performance of the DLL under slow power supply noise. A voltage bump of 25mV over 1µs was applied after the DLL had locked at 2ns $t_{CK}$, typical corner. This is enough noise to cause the IO delay to shift by over 100ps, but since the DLL is tracking, jitter is kept within the target window. The jitter increased by about 15ps when compared to the simulation results presented in figure 53 – the same conditions but without noise. The main reason for the increase in jitter is due to the phase interpolator. The DLL was forced to shift across an entire coarse delay step, so the worst case non-linearity was brought out.

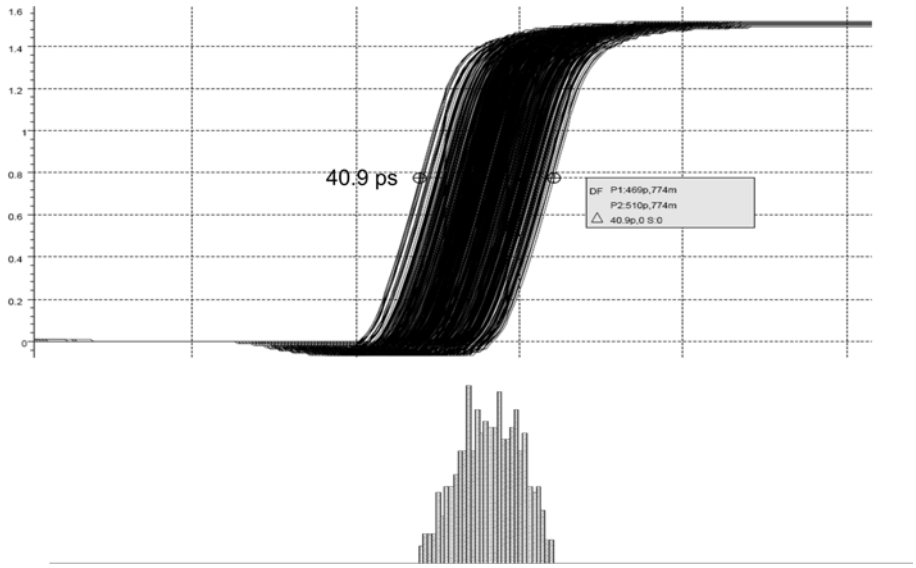



**Figure 58: Jitter waveform and histogram at typical corner, 2ns $t_{CK}$, with slow power supply noise**

Figure 59 shows the DLL performance under fast power supply noise. A 10MHz sine wave with peak to peak amplitude of 50mV was applied after the DLL had locked at 2ns $t_{CK}$, typical corner. The DLL cannot track noise at that frequency, so the jitter is increased significantly. This illustrates the importance of good power supplies for DLL performance. Any supply noise that is outside of the DLL tracking bandwidth will be directly translated to jitter. For this reason, a DLL typically uses power rails that are isolated from other circuitry as much as possible, and large decoupling capacitors are used to filter out AC noise.

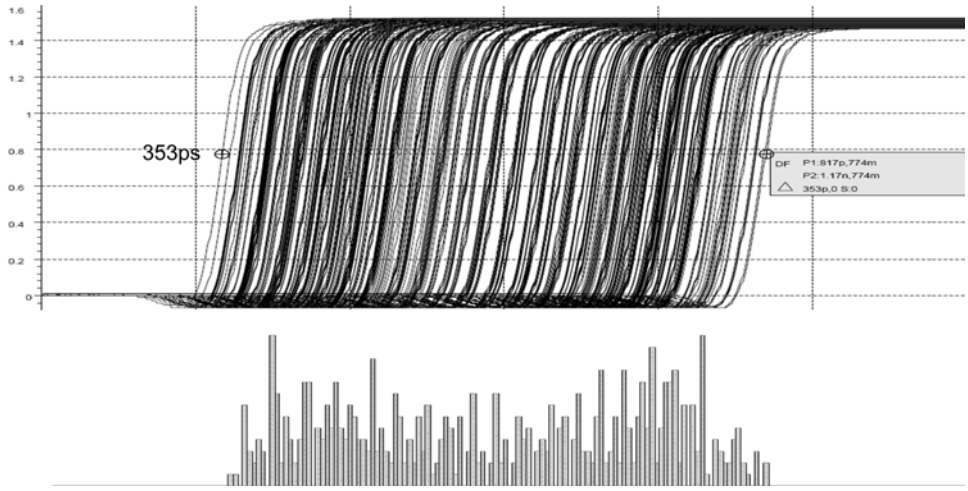



**Figure 59: Jitter waveform and histogram at typical corner, 2ns $t_{CK}$, with fast power supply noise**

# CHAPTER 5: FUTURE WORK

The jitter simulations showed that at high frequency there was much less margin to the jitter target than at lower frequencies. To improve this, the delay line could be divided into three sections: a shallow section with no graduation for maximum performance at high frequencies, a middle section with moderate graduation for balanced performance versus layout area, and a deep section with a large graduation value to take advantage of the jitter margin at the lower frequencies. Because the shallow section is not graduated, a phase detector with hysteresis could be used to create a steady state condition to further improve the ideal jitter performance at high speed.

Although the simulation results show that this design will perform well in an ideal environment across all PVT corners, improvements could be made to make it more immune to the noisy environment of a DRAM circuit. The fast initialization is a one-shot measurement that will be directly affected by instantaneous input clock jitter; power supply noise; or cross-coupled noise in the delay line, initialization circuitry, or any point along the loop. If the noise event were severe enough, the DLL might not lock in time. An initialization scheme that sent several pulses around the loop and averaged the measurement would be more immune to noise.

Noise events will also be an issue when the phase detector is sampled. If the noise event coincided with a sample, the DLL might shift in the wrong direction. This would not be a problem for a single event because the DLL would quickly recover.

However, if the frequency of the noise event was harmonic with the sample frequency, multiple shifts could occur in the wrong direction, severely affecting jitter performance. An averaging filter could be used in conjunction with the dynamic sample filter to provide some noise protection.

Another possible area of improvement is the fine phase mixer. The graduated delay line topology would not be possible without it, but the non-linearity and duty-cycle distortion hurts the overall performance of the DLL. A full analysis of the performance of different phase mixing topologies is enough work for a thesis within itself but would be the next step to improve this design further.

# REFERENCES

[1] T. Hsu, B. Shieh and C. Lee, "An all-digital phase-locked loop (ADPLL)-based clock recovery circuit," *IEEE J. of Solid-State Circuits*, vol 34, pp. 1063–1073, Aug. 1999.

[2] T. Saeki *et al*., "A 2.5-ns clock access, 250-MHz, 256-Mb SDRAM with synchronous mirror delay." *IEEE J. of Solid-State Circuits*, vol 31, pp. 1656-1668, Nov. 1996.

[3] W. Kim, K. Kim, Y. Jeon, and S. Kim, "A Ring & Counter Controlled Delay Line for a Wide Operation Range and Low-Jitter Performance," *Journal of the Korean Physical Society*, vol. 42, pp. 246-250, Feb 2003.

[4] F. M. Gardner, *Phaselock Techniques*, Wiley-Interscience, 2005.

[5] R. J. Baker, *CMOS Circuit Design, Layout and Simulation*, IEEE, 2005.

[6] M. G. Johnson and E. L. Hudson, "A variable delay line PLL for CPU-coprocessor synchronization," *IEEE J. of Solid-State Circuits*, vol 23, pp. 1218-1223, Oct. 1988.

[7] H. Chang, J. Lin, C. Yang and S. Liu, "A wide-range delay-locked loop with a fixed latency of one clock cycle," *IEEE J. of Solid-State Circuits*, vol 37, pp. 1021-1027, Aug. 2002.

[8] F. Lin, J. Miller, A. Schoenfeld, M. Ma and R. J. Baker, "A Register-Controlled Symmetrical DLL for Double-Data-Rate DRAM," *IEEE J. of Solid-State Circuits*, vol 34, pp. 565-568, Apr. 1999.

[9] A. Hatakeyama *et al*., "A 256-Mb SDRAM using a register-controlled digital DLL," *IEEE J. of Solid-State Circuits*, vol 32, pp. 1728-1734, Nov. 1997.

[10] H. Chang and S Liu, "A wide-range and fast-locking all-digital cycle-controlled delay-locked loop," *IEEE J. of Solid-State Circuits*, vol 40, pp. 661-670, Mar. 2005.

[11] J.G. Maneatis, “Low-jitter process-independent DLL and PLL based on self-biased techniques,” *IEEE J. of Solid-State Circuits*, vol 31, pp. 1723-1732, Nov. 1996.

[12] R. E. Best, *Phase-Locked Loops: Design, Simulation, and Applications*, McGraw-Hill, 2003

[13] E. Song, S. Lee, J. Lee, J. Park and S. Chae, “A reset-free anti-harmonic delay-locked loop using a cycle period detector,” *IEEE J. of Solid-State Circuits*, vol 39, pp. 2055-2061, Nov. 2004.

[14] S. J. Kim, S. H. Hong, J. K. Wee, J. H. Cho, P. S. Lee, J. H. Ahn and J. Y. Chung, “A low-jitter wide-range skew-calibrated dual-loop DLL using antifuse circuitry for high-speed DRAM,” *IEEE J. of Solid-State Circuits*, vol 37, pp. 726-734, Jun. 2002.

[15] B. Kim and L. Kim, “A 250-MHz-2-GHz wide-range delay-locked loop,” *IEEE J. of Solid-State Circuits*, vol 40, pp. 1310-1321, Jun 2005.

[16] Y. Jung, S. Lee, D. Shim, W. Kim, C. Kim and S. Cho, “A dual-loop delay-locked loop using multiple voltage-controlled delay lines,” *IEEE J. of Solid-State Circuits*, vol 36, pp. 784-791, May 2001.

[17] S. Sidiropoulos and M. A. Horowitz, “A semidigital dual delay-locked loop,” *IEEE J. of Solid-State Circuits*, vol 32, pp. 1683-1692. Nov. 1997.

[18] B, Garlepp, K. S. Donnely, J. Kim, P.S Chau, J. L. Zerbe, C. Huang, C. V. Tran, C. L. Portman, D. Stark, Y. Chan, T. H. Lee, M. A. Horowitz, “A portable digital DLL for high-speed CMOS interface circuits,” *IEEE J. of Solid-State Circuits*, vol 34, pp. 632-644, May 1999.

[19] M.-J. E. Lee, W. J. Dally, T. Greer, H.-T. Ng, R. Farjad-Rad, J. Poulton, R. Senthinathan, “Jitter transfer characteristics of delay-locked loops - theories and design techniques,” *IEEE J. of Solid-State Circuits*, vol 38, pp. 614-621, Apr 2003.