DESIGN OF A DELAY-LOCKED LOOP WITH A DAC-CONTROLLED ANALOG DELAY LINE

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Electrical Engineering

in the

College of Graduate Studies

University of Idaho

by

Tyler J. Gomm

March 2001

Major Professor: R. Jacob Baker, Ph.D.

AUTHORIZATION TO SUBMIT

THESIS

This thesis of Tyler J. Gomm, submitted for the degree of Master of Science with a major in Electrical Engineering and titled "Design of a Delay-Locked Loop with a DAC-Controlled Analog Delay Line," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____ Date_____
R. Jacob Baker

Committee Members _____ Date_____
Harry W. Li

_____ Date_____
Larry Stauffer

Department Administrator _____ Date_____
Joseph J. Feeley

Discipline's College Dean _____ Date_____
David E. Thompson

Final Approval and Acceptance by the College of Graduate Studies

_____ Date_____
Charles R. Hatch

ABSTRACT

High-speed synchronous interface circuits require that the controlling clock signals be accurately aligned. A dynamic de-skew circuit can be used to ensure good clock alignment across variations in process, voltage, and temperature variations (PVT). The delay-locked loop (DLL) is such a circuit, using a first-order closed-loop architecture that dynamically aligns its output clock signal with a reference clock signal.

Two basic types of DLL architectures are currently used: analog and digital. The analog DLL uses a continuously variable delay line to remove the skew between the output clock and the reference clock. A digital delay line uses digital elements, making the design more simple and portable, but with quantized steps in the delay time. Typical DLL architectures are explained, as well as the factors that introduce error in the accuracy of synchronized output.

The design of a DLL that uses a differential, analog delay line is presented. A segmented, current-steering digital-to-analog converter (DAC) is designed. This DAC controls the delay line in increments of the DAC resolution. A synchronous up/down counter is used to control the DAC inputs. An arbiter is developed for phase detection. The DLL initialization control circuitry is explained, and final performance characteristics across PVT corners are presented.

This DLL is designed and simulated using Micron Technology, Inc.'s 2.5V, 0.15 micron DRAM process. The lock range is 100MHz to 200MHz, with varying jitter performance. Suggestions are made for improvement of the jitter performance.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. R. Jacob Baker for his insight throughout the course of this work, and for his ability to teach. He has ensured that I have a solid understanding of the design principles and concepts presented, and has had patience with my never-ending questions. I would also like to thank him for his guidance and for setting an example to follow in my professional career.

I extend thanks to my current employer, Micron Technology, Inc., for the opportunities of learning and expertise gained in delay-locked loop circuits, as I have spent the past year working exclusively on DLL-related issues.

Thanks to all of the coordinators and administrators in the University of Idaho's Engineering Outreach program for their assistance throughout the past several years.

Most of all, I would like to thank my wife for her incredible love and support and my daughters for their understanding during this time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF GRAPHS

LIST OF FIGURES

CHAPTER 1 – INTRODUCTION

1.1 MOTIVATION

As the clock frequency of synchronous VLSI circuits increases, there arises a greater need to correctly align system clocks. Emphasis must be placed on suppressing clock skew and jitter. As the clock period is reduced, if jitter and skew remain the same, the total clock phase error is increased. This can affect many aspects of a synchronous system, including setup and hold times, data access times, and accuracy of internal control signals.

To eliminate clock skew, a simple, fixed-delay circuit might be used, but with variations in process, voltage, and temperature (PVT), the delay time would vary. Also, if the clock period were to change, the delay time would need to be modified. Therefore, a dynamic, variable delay circuit is needed to eliminate system clock skew across PVT and varying clock frequencies.

A Delay-Locked Loop (DLL) is such a circuit. Fig. 1 shows a DLL being used to ensure proper synchronization between a synchronous memory device and a memory controller. In this simple example, the DLL within the memory device is used to ensure that there is no skew between the control clock generated by the controller and the data coming out of the memory. This is especially important when a fast control clock is used; if skew remains in the output data while using a

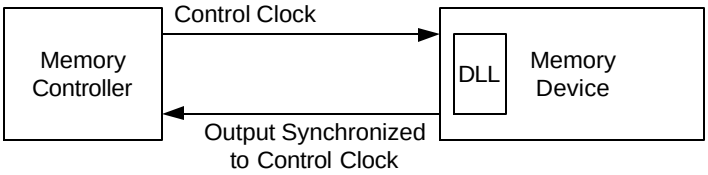


Figure 1 DLL usage example.

fast clock rate, the memory controller might have a difficult time distinguishing from one bit to the next being clocked from the memory.

As synchronous memory moves to other standards, such as double-data rate (DDR) devices, where data is clocked out on both the rising *and* falling edges of the control clock, the problem of internal clock skew is compounded. A DLL can solve this problem by ensuring proper synchronization across PVT as well as variations in the control clock frequency [1], [2].

There are limitations to the accuracy with which the DLL control clock may be synchronized with the external reference clock. Some of these limitations are due to the design of the DLL, while others are results of the process and system noise. This thesis presents some of the common designs for clock synchronization circuits as well as the possible drawbacks to these designs. The design and simulation of a DLL that uses a DAC-controlled analog delay element are presented.

1.2 IDEAL DLL OPERATION

The basic Delay-Locked Loop block diagram and timing are shown in Fig. 2. Note that the DLL has many similarities to a Phase-Locked Loop (PLL). One major difference is that rather than a Voltage-Controlled Oscillator (VCO), a voltage-controlled delay-line is used. If the output of the delay were fed back to the input (forming an oscillator), the DLL could become a PLL. The other significant difference is that the DLL has a first-order response, where the PLL has a second-order response [2], [3], [4], [5]. This means that system stability is usually not an issue for DLL designs, making them easier to realize in silicon. In addition, because there is no VCO in the DLL, and therefore, no accumulation

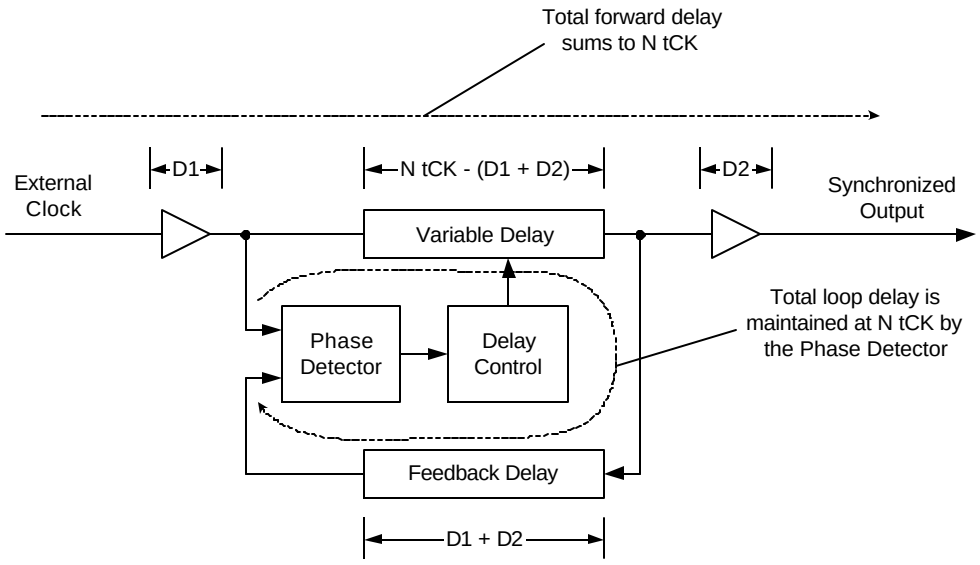


Figure 2 Basic DLL block diagram.

of phase error due to supply noise, the jitter performance of a DLL can be better than that of a PLL [2]. In other words, the jitter transfer function of a DLL is equal to zero, because the reference clock both generates *and* synchronizes the output clock [3].

It should be noted that a PLL could also be used to accomplish the goal of de-

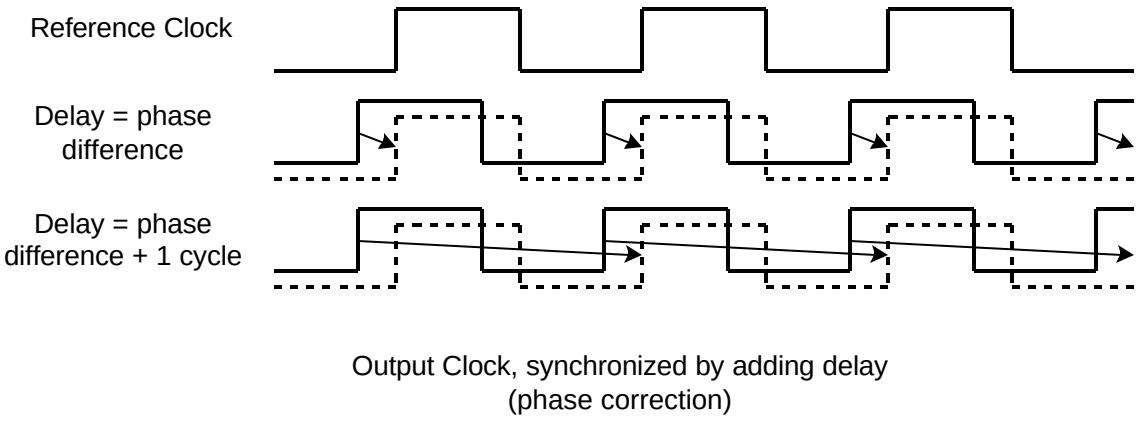


Figure 3 Basic DLL timing.

skewing a control clock and a reference clock signal. However, due to the design constraints and the noisy environments in which these circuits must function, DLL's are more often used for simple de-skew applications [3].

As shown in Fig. 3, the DLL attempts to remove the skew, or phase difference, from the output control clock by adding additional delay such that the phase of the two clocks are equal. Fig. 3 shows that the delay added may be just the minimum phase difference, or may be any multiple of the clock period plus the phase difference.

Referring back to Fig. 2, the phase detector in the DLL will maintain the total delay around the loop such that it is equal to $N \cdot tCK$, where *tCK* is the clock period. In a practical implementation, the external reference clock must propagate through an input buffer, plus any additional control logic that is necessary. In Fig. 2, this is all lumped together as *D1*. Likewise, the synchronized control signal exiting the variable delay line will have a finite propagation path out of the system. This is lumped together as *D2*. It must be recognized that both of these delays are functions of PVT. Therefore, in order for the DLL to properly cancel out the PVT variations of these finite delays, it must be capable of removing *D1* and *D2* from the total forward delay. This is accomplished by modeling the two delays as a fixed delay in the feedback path. The delay model should track *D1* and *D2* across PVT as closely as possible. Notice that because the phase detector maintains the loop delay at $N \cdot tCK$, the variable delay is set at $N \cdot tCK - (D1 + D2)$. The total forward-path delay is now

$$D1 + N \cdot tCK - (D1 + D2) + D2 \quad = \quad N \cdot tCK \tag{1}$$

1.3 NON-IDEAL CONSIDERATIONS

Eq. (1) shows the most ideal DLL operation. The output is no longer a function of PVT; the delays *D1* and *D2* have been eliminated and the phase detector will maintain the loop delay as process, temperature, and voltage vary. Of course, there are limitations to how well this occurs. When non-ideal system components are used, Eq. (1) becomes

$$D1 + N \cdot tCK' - (D1' + D2') + D2 \quad = \quad N \cdot tCK' + (D1 - D1') + (D2 - D2') \tag{2}$$

The 'prime' notation is used in Eq. (2) to denote the fact that the feedback delay model will not perfectly match the input and output delays, and there will be limitations on how well the phase detector and delay line can maintain exactly $N \cdot tCK$ around the loop. This loop delay error might be due to granularity in the delay line, errors in the phase detector, or delay modulation due to power supply variations. Therefore, the total DLL error will reflect the accuracy of the loop delay plus any errors in modeling the input and output delays.

A significant limitation of the DLL is the ability to lock across many frequencies [5]. The variable delay line must be designed so that it can allow the loop delay to reach at least $1 \cdot tCK$ when the longest clock period is used. However, in order to maintain consistent operation, the DLL should be designed so that it does not lock on different harmonics of the clock period. In other words, *N* must be the same value every time for a given clock period. Any imperfections in the delay line will compound as more delay is used; therefore it is desirable to keep *N* to a minimum. To avoid the problem of locking on varying harmonics, the following inequality should be applied:

$$\frac{1}{2} \cdot tCK < t_{DELAY(MIN)} < tCK < t_{DELAY(MAX)} < \frac{3}{2} \cdot tCK \tag{3}$$

Eq. (3) shows that $t_{DELAY(MIN)}$ (the minimum possible delay through the variable delay line) should be greater than 1/2 of the operating period, while $t_{DELAY(MAX)}$ (the maximum possible delay through the variable delay line) should be less than 3/2 of the operating period [6]. Since t_{DELAY} might vary as much as 2:1 over PVT in a typical CMOS process, the conditions imposed by Eq. (3) can be maintained only over a narrow range of operating frequencies. In practice, however, many DLL's can ignore this limitation, and allow the system to lock on harmonics of *tCK*.

Before discussing some of the various DLL implementations, another limitation should be pointed out. If the feedback model error discussed above is negligible (although this is not usually practical), the maximum phase error is

$$e_{f(MAX)} = \frac{2p \cdot t_d}{tCK} \tag{4}$$

In Eq. (4), t_d is the minimum increment in which the variable delay line may be changed. This equation shows that as the input frequency increases, the steady-state phase error will also increase [5]. The minimum delay increment directly determines the jitter performance of the DLL. This error becomes especially significant in digital DLL's, where the minimum delay increment can be relatively large.

CHAPTER 2 – CURRENT DLL DESIGNS

2.1 DIGITAL DLL

There are essentially two types of DLL designs: analog and digital. The design choice is determined by several factors, including design complexity, layout size, system noise levels, necessity of process portability, and required accuracy.

A digital delay-locked loop uses digital devices to implement the variable delay-line (Fig. 4). This means that the minimum change in the delay is some quantized step. A digital delay cell makes up the minimum delay, and many of these delay cells are used to create the delay line. This minimum delay, or delay per stage, will be limited by the CMOS

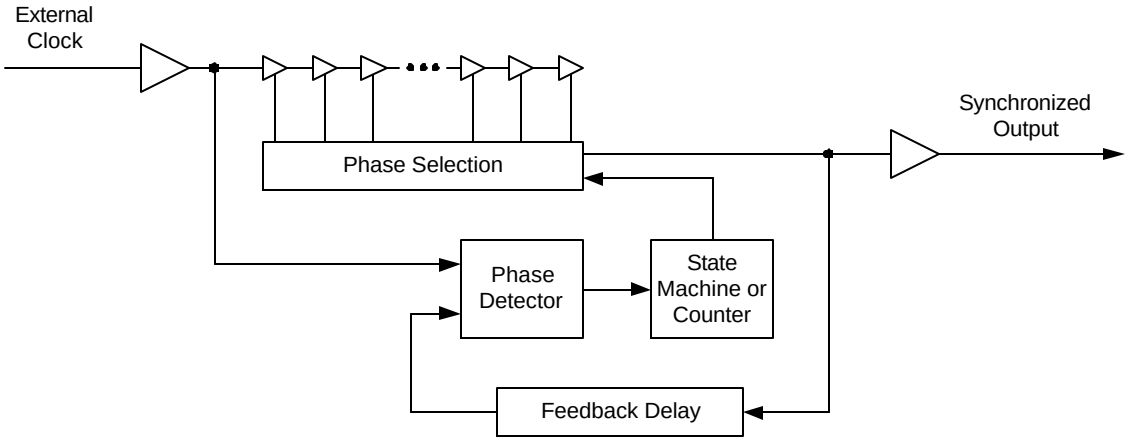


Figure 4 Digital DLL block diagram.

process in which the device is fabricated. However, due to the reliable nature of digital circuit performance across varying processes, a digital design may still be desirable [7].

As shown in Fig. 4, the variable delay is created by sending the reference clock through many digital gates (these might be inverters, NAND gates, or NOR gates), and a phase selector is used to choose which stage's output will be used for the locked signal.

There are many flavors of digital DLL's. The delay line may be implemented using pairs of inverting CMOS gates (to avoid an unwanted phase inversion), with the pairs ranging from NAND-invert to NOR-NAND combinations. This choice is influenced by the required duty cycle, need for symmetry in t_{pHL} and t_{pLH}, as well as the method used to inject or extract the clock into or out of the delay line [9]. In order to reduce the delay per stage, the digital delay line might also be implemented using single cells of inverting CMOS gates, but using a twisted-pair scheme to avoid the unwanted phase inversion [7][10].

Other DLL design variations arise from the method used to control the delay line. Fig. 5 shows a register-controlled DLL (RDLL) where a shift register is used to shift a bit that marks the delay-line entry point from left to right [8] [11].

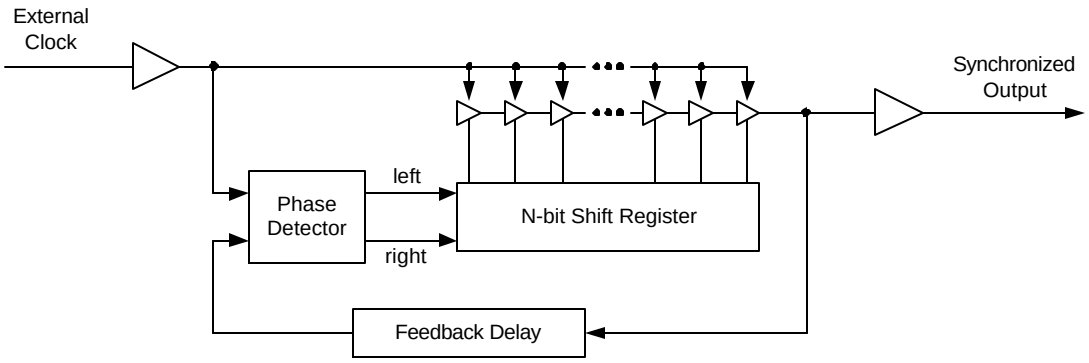


Figure 5 Register-controlled DLL.

In the RDLL, each delay element is equal, and the shift register contains a bit that controls where the reference clock enters the delay line. This can result in a straightforward implementation, as the shift register may be integrated directly with the delay line [11].

Another digital delay line worth mentioning is one in which the delay elements are binary-weighted, allowing an up/down counter to control the amount of total delay (Fig. 6). This architecture is referred to as a counter-controlled DLL (CDLL) [8].

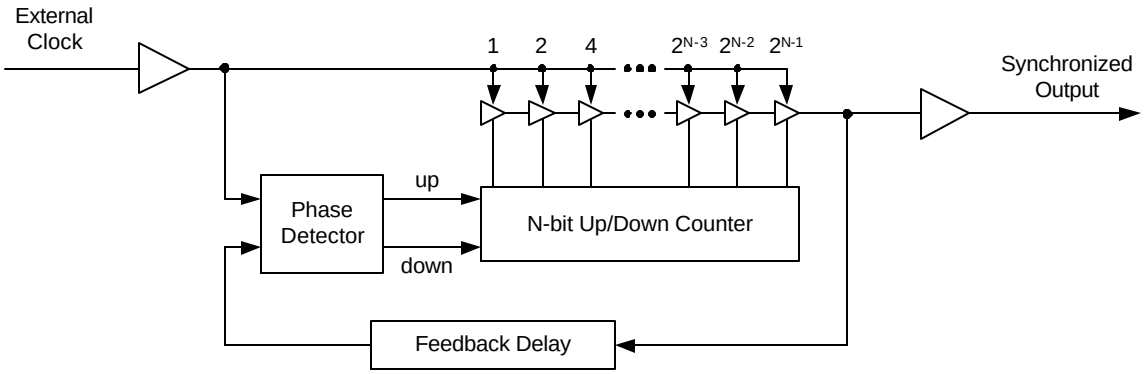


Figure 6 Counter-controlled DLL.

Examining the previous figures, it is apparent that while the method of delay line control will vary, the delay line remains digital. This results in a quantization of the output phase. Shown in Fig. 7, as the input frequency varies, the DLL output can only track the input clock edge to within the minimum resolution of the delay line, t_d. This results in a possible phase error of $2p \cdot t_d \cdot f_{ref}$. In other words, the most ideal digital DLL will be limited by the minimum delay per stage. Again, this assumes that a perfect phase detector is used, and is not generating additional phase error.

The phase detector used in a digital DLL needs only to be sensitive to variations of phase greater than the granularity of the delay line. If it is more sensitive, the system might

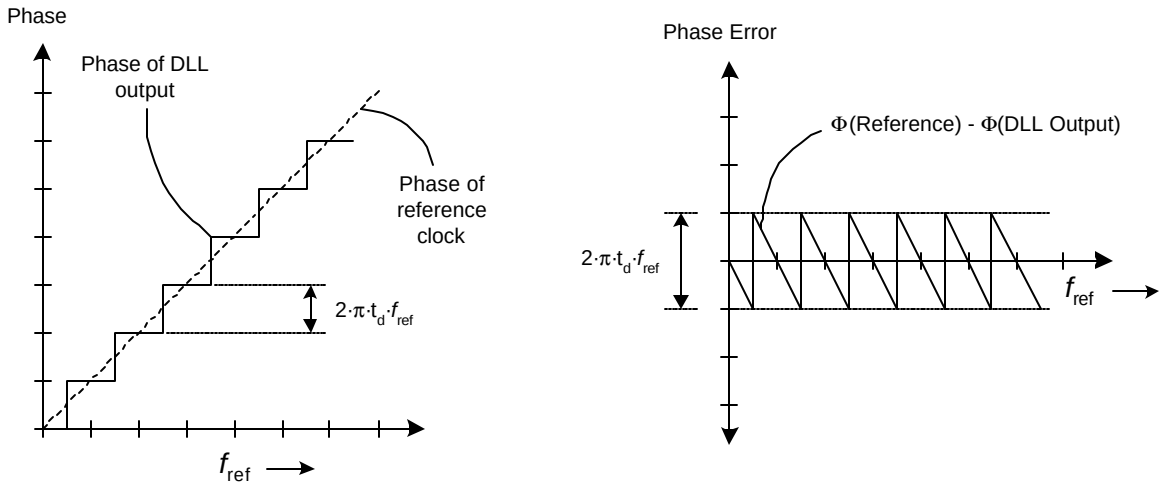
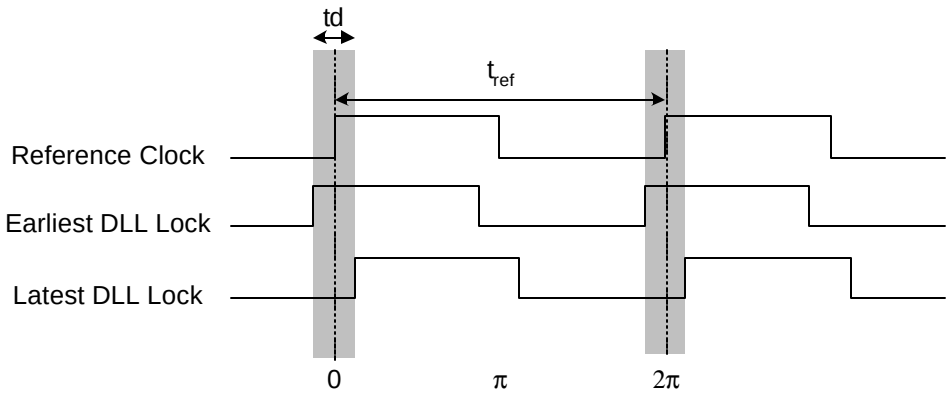


Figure 7 Phase error in a digital DLL.

oscillate about the desired lock point. If not sensitive enough, additional phase error might be added to the total error, as mentioned in the previous paragraph. Fig. 8 illustrates the phase detector timing requirements.



Lock is attained across the phase range:

$$-\frac{td}{t_{ref}}\cdot p \quad \rightarrow \quad \frac{td}{t_{ref}}\cdot p \qquad\qquad \Delta = \frac{td}{t_{ref}}\cdot 2p$$

Figure 8 Phase detector timing for digital DLL.

2.2 ANALOG DLL

To reduce the static phase error incurred by using the quantized timing generated by a digital delay line, an analog delay line may be used [6]. Due to the ability of an analog delay element to vary continuously, the jitter in the output is reduced. Fig. 9 shows the basic analog DLL. The overall topology remains the same; however, a charge-pump/loop filter combination is frequently used to control the analog delay line [6], [8].

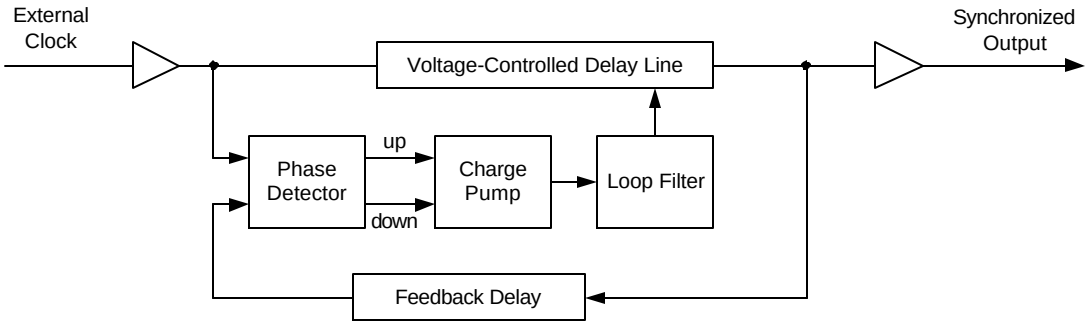


Figure 9 Analog DLL block diagram.

The delay elements used to create the delay line are commonly found in voltage-controlled oscillators (VCO's) typical for PLL design. They may range from a simple, current-starved gate, to complex amplifier-based designs. Again, the selection depends on the environment in which the design will be used, the process tolerances, and the jitter requirements.

Besides the advantage of reduced jitter in the output, an analog design may exhibit a better power-supply rejection ratio (PSRR), can occupy a reduced area in the layout, and consume less current than a digital DLL [7]. However, to achieve these results, the design is usually of higher complexity and requires a process-specific implementation, making it less portable to other processes.

2.3 DUAL-LOOP DLL

Architectures that use two delay lines to enhance the performance of the DLL are also used (Fig. 10). These designs have been proposed to solve the problem of jitter induced by quantization in a digital DLL [6][15]. Either delay line may be digital or analog, depending on the design approach. The intent is to use an easily-implemented coarse delay line to set the loop very near to lock, and then to rely on a fine delay line to make small adjustments to the output phase, reducing the static quantization error shown in Fig. 7.

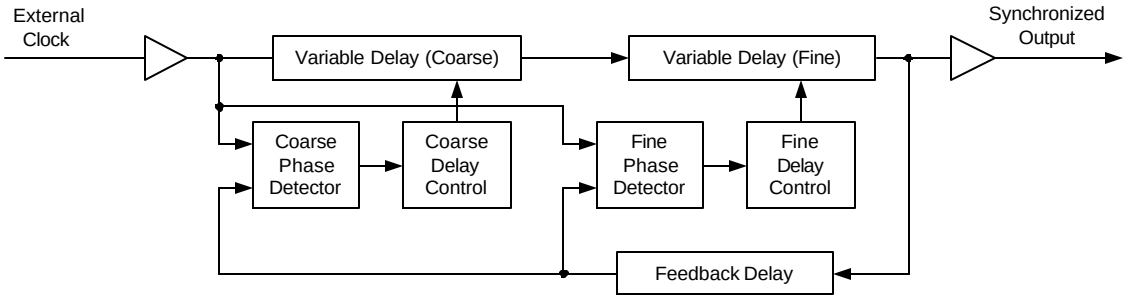


Figure 10 Dual-loop DLL block diagram.

Another method of reducing the quantization error in a two-step process is to use phase mixers, created from simple CMOS gates, to finely tune the DLL output [7], [16].

2.4 SYNCHRONOUS MIRROR DELAY

Although the concept of a synchronous mirror delay (SMD) deviates from the topic of DLL's, it is important to realize that alternate methods of clock synchronization across PVT are possible [12]. The commonality between the synchronization circuits is the ability to create a delay that is equal to $N \cdot tCK - (D1 + D2)$ (referring back to Fig. 2). Because *D1* and *D2* vary across PVT, any circuit that can measure *tCK* minus the input and output

buffer delay (across PVT) can be used to synchronize two clocks. Fig. 11 shows a basic block diagram of the SMD.

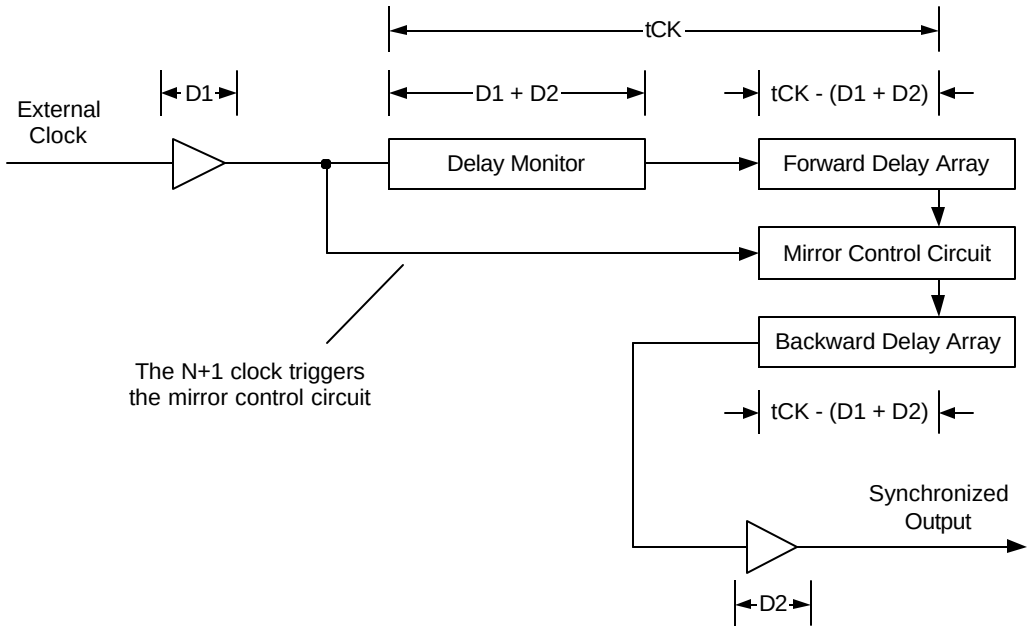


Figure 11 Basic SMD block diagram.

The delay monitor is exactly the same delay used in the feedback path of the DLL; it is used to model the input and output buffer delay. It is important to note that the SMD is not a closed-loop system, therefore it would be incorrect to refer to a 'lock time' or 'lock point'. It is simply a clock *generation* circuit. The Nth clock propagates as far as it can into the forward delay line before the next rising clock edge (N + 1) causes it to move down into the backward delay line. The signal continues on its way out of the system. The total forward path is calculated as follows:

$$D1 + (D1 + D2) + 2 \cdot \left[N \cdot tCK - (D1 + D2) \right] + D2 \quad = \quad 2 \cdot N \cdot tCK \tag{5}$$

Typically, N is equal to 1, allowing a synchronized output to be generated in only two cycles. This is one of the significant advantages of the SMD over the DLL; the SMD clock generation time is much faster than the typical DLL lock time.

The issue of granularity in the delay line still applies, and, as with the DLL, designs have been proposed that use an analog method to time the $N \cdot tCK - (D1 + D2)$ delay in the SMD [13].

Once it is understood that this delay timing is the only element needed to synchronize, other designs are possible. Fig. 12 shows a SMD implementation that uses an up/down counter [14] or even a charge pump to measure the forward and then backward time, instead of a delay line. The output resolution of this design is limited to the period of the clock used to count up and down or the analog resolution of the charge pump. However, it is easy to see that this type of design might be implemented using an extremely small amount of layout area.

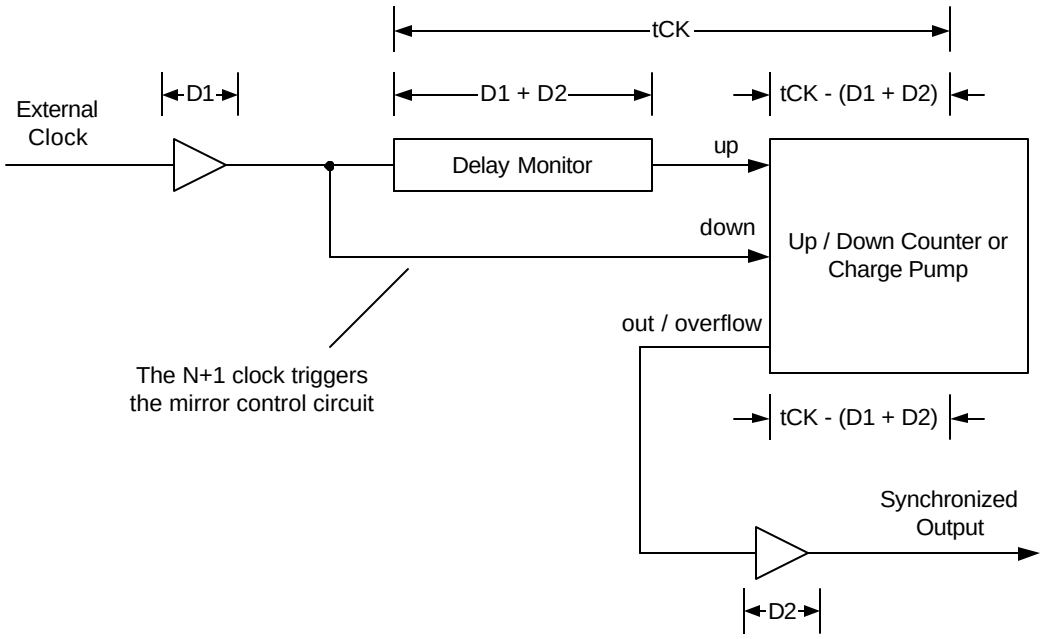


Figure 12 Other possible SMD implementations.

CHAPTER 3 – DLL DESIGN

3.1 PROCESS AND SIMULATION MODELS

The process used for design and simulation of the DLL design proposed by this thesis is Micron Technology, Inc.'s 2.5V, 0.15 micron process (0.124 micron after shrink), specifically tuned for DRAM production. The typical process parameters, such as oxide thickness, *Cox*, etc, are proprietary. The process provides two metal layers and single polysilicon. One important feature of this process is that it is tuned such that the substrate voltage is expected to be negative, with Vbb ≈ -0.65V. This raises the threshold voltage of the N-channel devices. All N-channel transistors used in this design have the bulk node connected to the negative Vbb potential. Because of the increased N-channel threshold voltage, the N-to-P channel drive ratio is about 2:1.

All simulations were performed using HSPICE and/or ADM. Schematic capture was done with Cadence's DFII software. All transistor sizes are given in terms of lambda, or the shrink factor (1 lambda = 0.124 micron).

As with most deep-submicron designs, simulation must be relied upon rather than hand calculations. The deviation from the long-channel models due to short-channel effects reduces the effectiveness of typical hand-calculations.

3.2 PROPOSED DESIGN

The DLL designed for this thesis is intended for a 100MHz to 200MHz reference clock frequency. As discussed in the previous chapter, a significant limitation of the digital DLL is the coarse delay-per-stage, typical of a CMOS gate. In addition, a delay line that

can lock on a wide range of frequencies may occupy a large layout area. Therefore, this design will use an analog delay line with the goal of using very few delay stages and exhibiting continuous-delay behavior. Output jitter (with noise on the power supply) must be in the range of 500ps in order to be a useful design. The power consumption should be just a few milliamps.

Another useful design goal would allow generation of an in-phase output signal and also quadrature signals; 0° (this is the in-phase output), 90°, 180°, and 270° phase (Fig. 13).

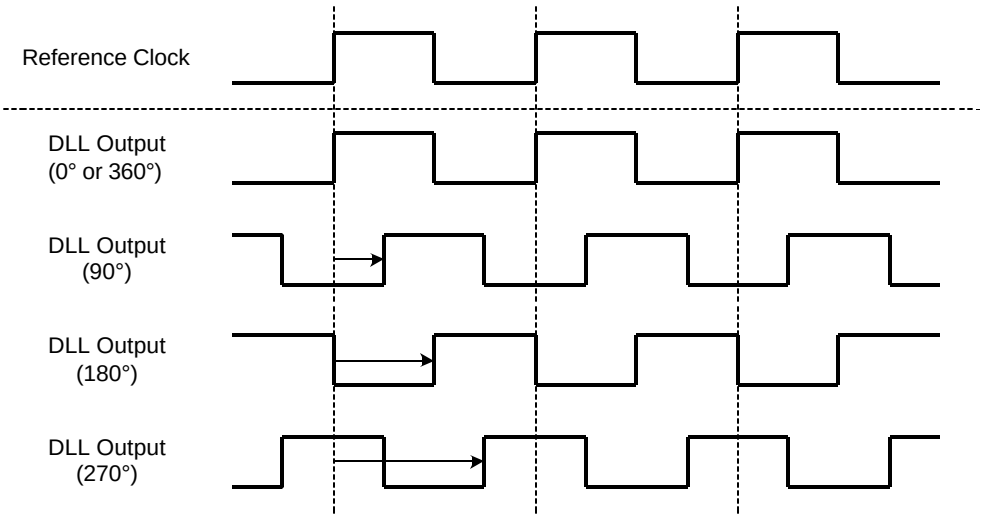


Figure 13 Quadrature signals.

This is a desirable feature if the DLL were to be used in data recovery applications [17], [18]. It could also be very useful for generating control signals for internal system elements, for example, in the creation of internal synchronous DRAM timing signals. The generation of quadrature signals is more easily done when an analog delay line is used, due to its scaling nature. If a multiple of four delay stages is used, the quadrature signals are already generated within the delay line. If a digital delay line were to be used, the 180°

signal could easily be obtained by simply inverting the output, but the 90° and 270° generation would require a more complex approach.

Fig. 14 is a block diagram of the proposed DLL design. Instead of using a charge pump and loop filter to create the control for the delay line, a current-output digital-to-analog converter (DAC) is used. The phase detector generates up/down control signals for a synchronous up/down counter that control the DAC. The concept is to use the features and advantages of an analog delay line, producing quadrature output signals, and to rely on the least significant bit (LSB) of the DAC to be fine enough to set a very small delay change per DAC step in the delay line. This will, in effect, quantize the phase output of the DLL just as in a digital delay line, but if the LSB of the DAC is small enough, output jitter will be less than that of a digital DLL.

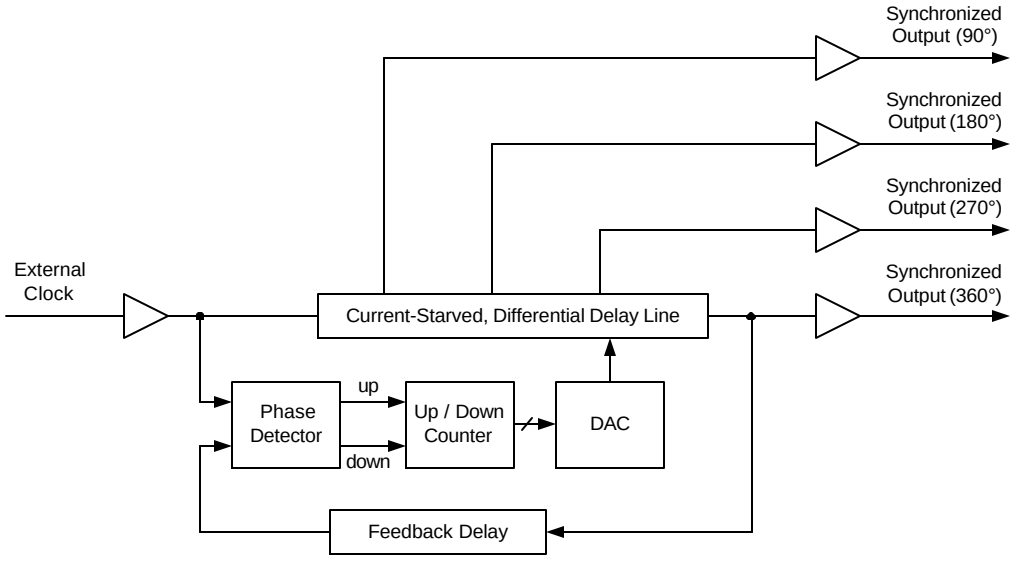


Figure 14 Block diagram of proposed design.

Another advantage to this approach is that by using an analog delay line, the total layout size will be significantly reduced. The DAC will occupy the majority of the layout area.

As previously discussed, a disadvantage of an analog design is the reduced portability of the design from process to process. For the purposes of this design, this is not much of a concern. More important is the ability of the design to maintain proper operation across PVT in the same basic process.

3.3 DELAY LINE DESIGN

The delay line must be capable of supporting the frequency range of interest (100MHz – 200MHz, or 10ns - 5ns cycle time) and have good PSRR. The simplest continuously variable CMOS delay element is the current-starved inverter (Fig. 15). The center transistors form a normal CMOS inverter and the top

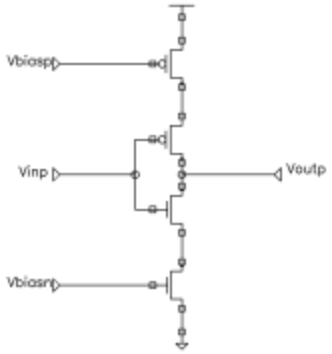


Figure 15 Current-starved inverter.

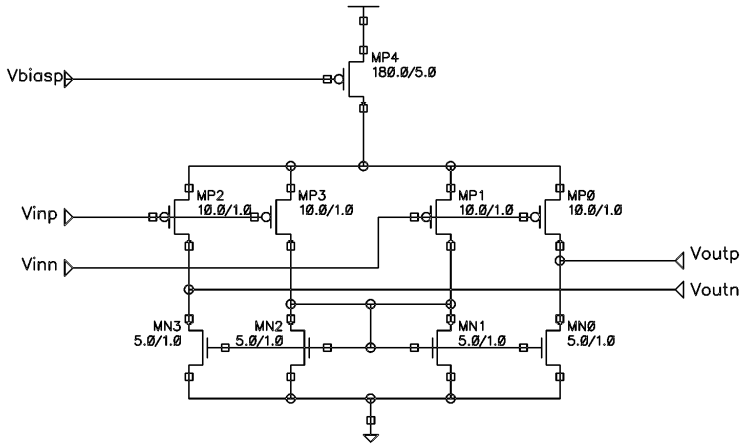


Figure 16 Diff-amp-based delay element.

and bottom transistors serve as variable current supplies. As the current to the inverter is changed, the rate that the output can charge up or down is affected, and therefore the delay through the element is directly controlled. If several of these elements are arranged sequentially, a simple analog delay line is formed. In practice, this type of delay element is not frequently used because of its poor PSRR and susceptibility to noise [19]. In addition, the current sources must be finely tuned to maintain good duty cycle characteristics.

A better design is the diff-amp-based delay element shown in Fig. 16. This circuit has good symmetry and better power-supply rejection [19]. The rise and fall times of this circuit are equal, due to the differential nature of the design. It sources and sinks the same current into the output load, regardless of the switching direction. This is the design chosen for the differential delay line of this thesis.

The range of the delay element stage and the desired frequency lock range of the DLL dictate the number of stages required. Simulations show that the minimum delay per stage is ~400ps. If eight stages are used, the minimum delay for the entire line will be about 3.2ns. This will meet the required minimum delay of 5ns, with margin for PVT variation. Fig. 17 shows the arrangement of the eight differential delay elements. If the

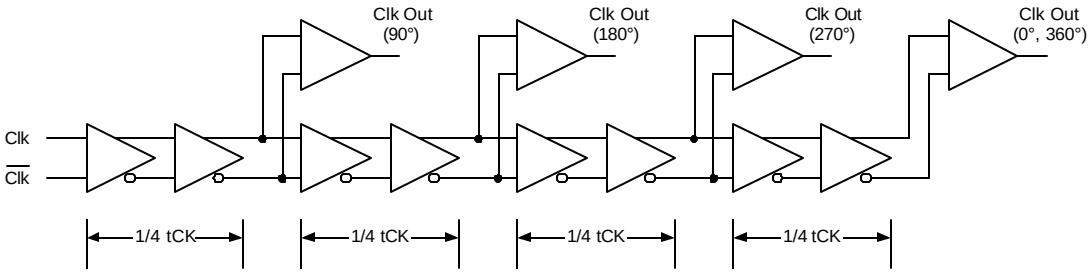


Figure 17 Differential delay line diagram.

delay line is locked such that the entire delay is equal to *tCK*, every two stages will have a

delay of ¼ *tCK*. Tapping the delay line at the ¼ tCK points creates the quadrature DLL outputs. A differential-to-single-ended converter is used to drive the output (Fig. 18).

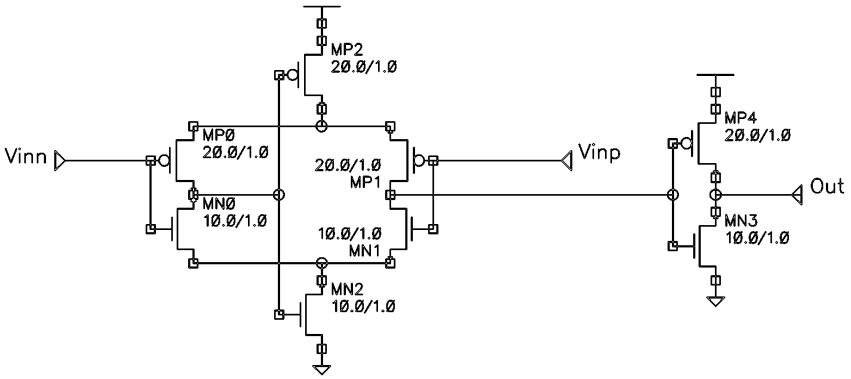


Figure 18 Differential-to-single-ended converter.

SPICE simulation shows the delay line ability to create the proper quadrature signals. Fig. 19 shows the delay line output with a 10ns clock input period and a fixed control current. Note that because the delay line is simulated here as a stand-alone circuit,

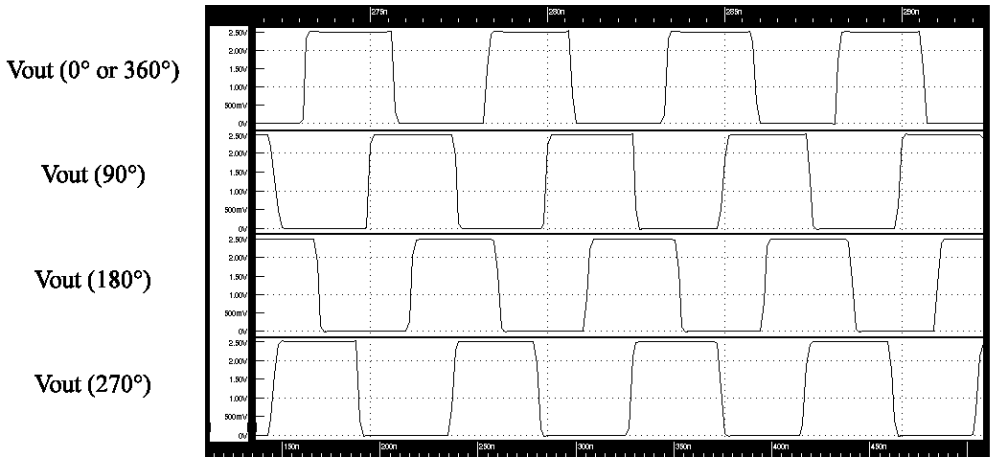
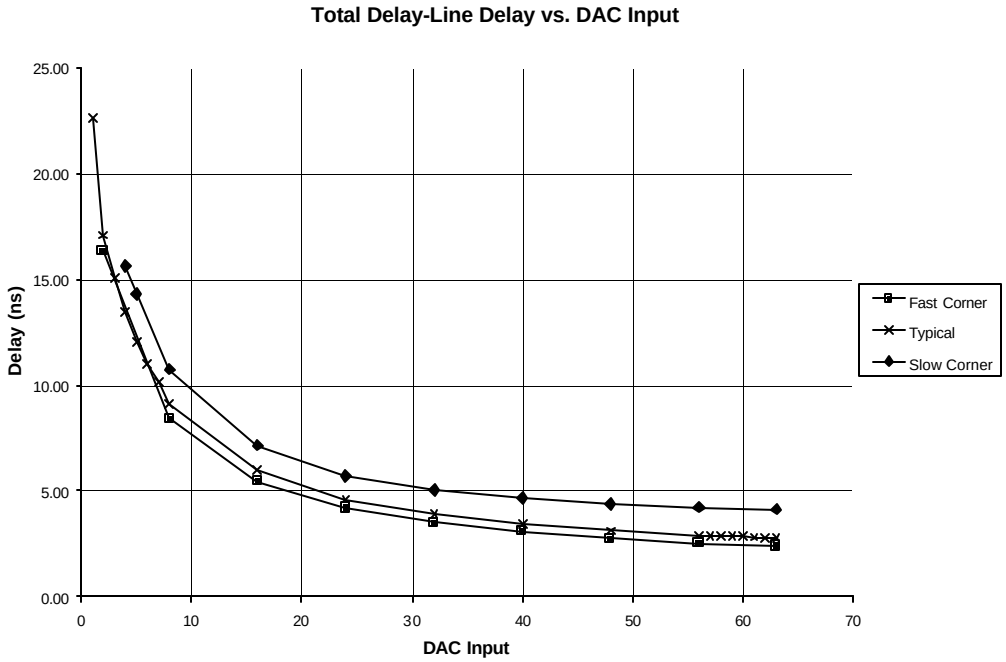


Figure 19 Simulated outputs of delay line.

the total delay through the delay line is not exactly 10ns, and therefore the quadrature shown in Fig. 19 is not perfect. When the delay line is used in a locked DLL, the quadrature will be correct.

One issue should be mentioned about the DLL's ability to properly generate the four quadrature signals. The ideal DLL equations rely on the feedback delay model to cancel out the effect of the input and output buffers. If delay is used in the feedback path, the total delay-line delay will not be an exact multiple of the input clock period. This means that if the delay-line timing is divided into quarters, the quarter delays will not be exactly ¼ *tCK*. Because this design does not have a significant input or output buffer delay time, a feedback delay model will not be used. This allows the circuit to generate better quadrature output signals.

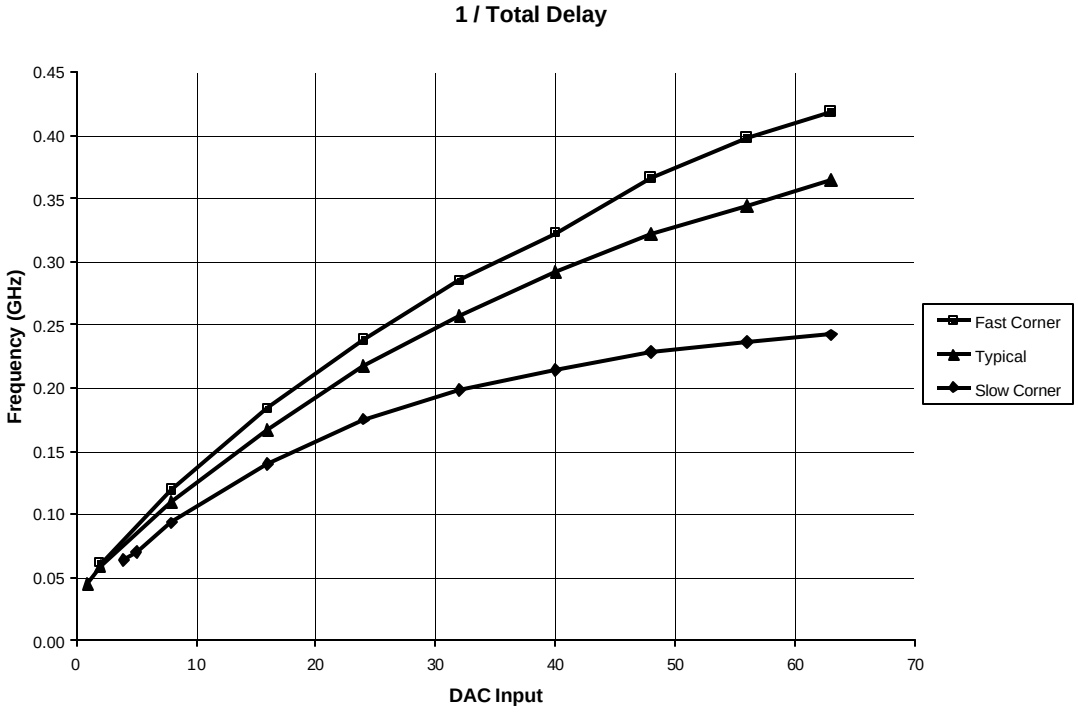


Graph 1 Delay-line performance across PVT.

The simulated characteristics of the differential delay line are shown in Graph 1. The delay line was simulated using an ideal DAC used to generate the control current. Simulations were performed on the fast and slow corners as well as at typical PVT.

Throughout this thesis, the fast corner is simulated using the fastest transistor models for the process, highest voltage (2.8V), and lowest temperature (0° C). The slow corner is run at the slowest transistor models, lowest voltage (2.2V), and highest temperature (80° C). The typical runs are done using typical transistor speeds, nominal voltage (2.5V), and nominal temperature (25° C).

An important observation can be made from the data shown in Graph 1. As the control current decreases, the delay increases in a 1/x fashion. If a linear DAC is used to control the delay line, the delay change per LSB is *not* constant across the delay range. As

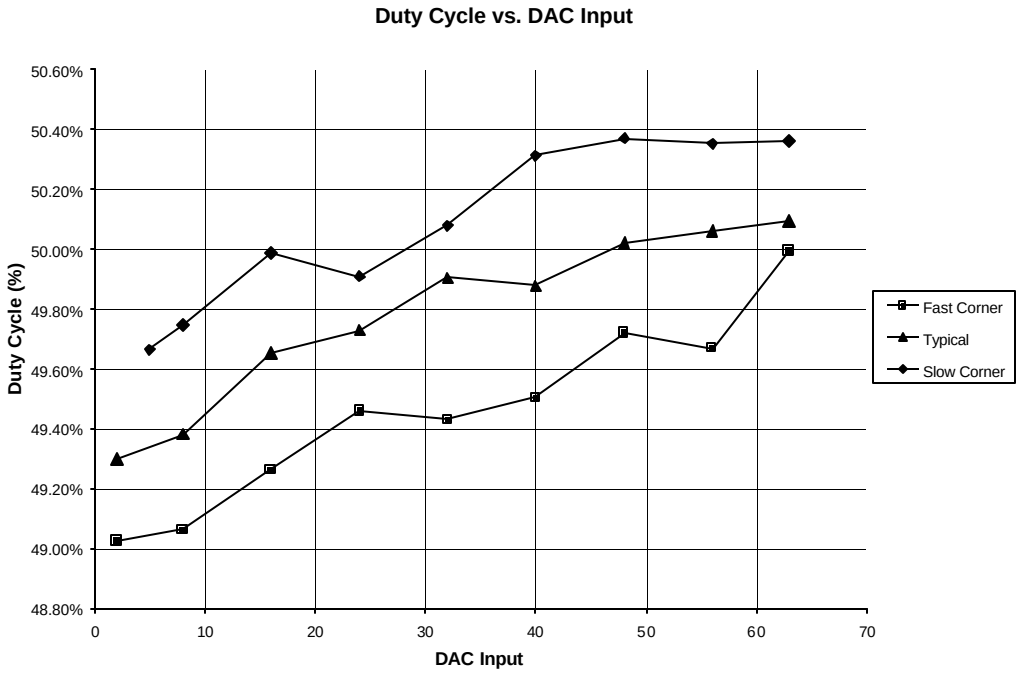


Graph 2 Linear 1/T relationship.

the total delay increases, the minimum delay-line variation also increases. The largest delay change per LSB occurs when the delay elements are fed very little current. The smallest change occurs when the delay elements are only slightly starved. This means that the DLL will have good jitter performance when higher current is used (fastest delay), and poor jitter performance with low current (slowest delay).

For PLL design, this is not a typical concern because the delay line is used as a variable oscillator, and the output of interest is frequency. Graph 2 shows that this delay line, if used as an oscillator, would be much more linear. Of course, this does not help the performance of the DLL.

Of interest is the duty cycle symmetry of the delay line. Graph 3 shows the simulation results of the duty cycle across PVT. As expected, this delay line has good



Graph 3 Delay-line duty-cycle performance.

symmetry due to the diff-amp nature of the delay element. This is important, especially if the control signals generated by the DLL are created using both the rising and falling-edges of the clock.

3.4 CURRENT REFERENCE DESIGN

In order to ensure that the current-output DAC will have good performance across PVT, it is extremely important that a stable current reference be used in the DAC. The ideal current reference has infinite source impedance, and the output does not vary, regardless of the voltage on the output nodes. One significant limitation of current sources, especially in deep submicron designs, is the finite value of r_o and limited output swing [19]. To achieve better analog performance, the transistors that are used to source/sink a linear current are sized so that the length is at least 5 lambda. This increases r_o of the source, making it more linear.

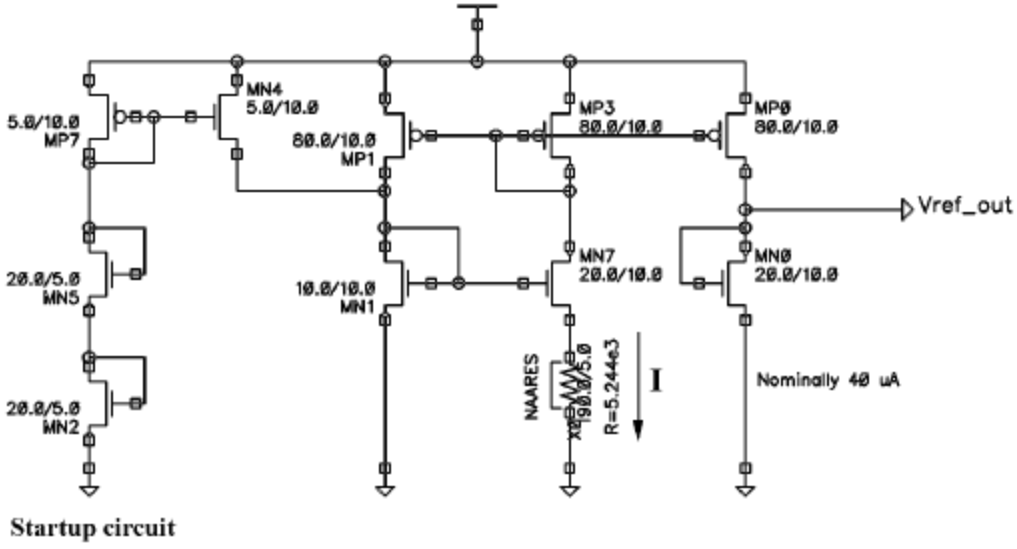


Figure 20 Beta multiplier current reference.

For this design, a beta multiplier reference is used (Fig. 20). The left-hand part of the circuit ensures that at power-up, the self-biased loop converges to the desired operating point, rather than at zero current. This is required for all self-biased circuits [19]. The transistor MN7 is sized so that it is a factor of *K* times larger than MN1. This sizing makes $b_{MN7} = K \cdot b_{MN1}$ which sets $V_{GSMN1} = V_{GSMN7} + IR$. The basic design equation for this circuit is shown as Eq. (6) [19].

$$I = \frac{2}{R^2 b_{MN1}} \cdot \left(1 - \sqrt{\frac{1}{K}}\right)^2 \tag{6}$$

The reference for this design is set such that the current through the rightmost leg is nominally 40uA. The resistor is made from N-type active area. All values and transistor sizes are shown in Fig. 20.

As previously stated, it is desirable to increase r_o of the reference to improve the

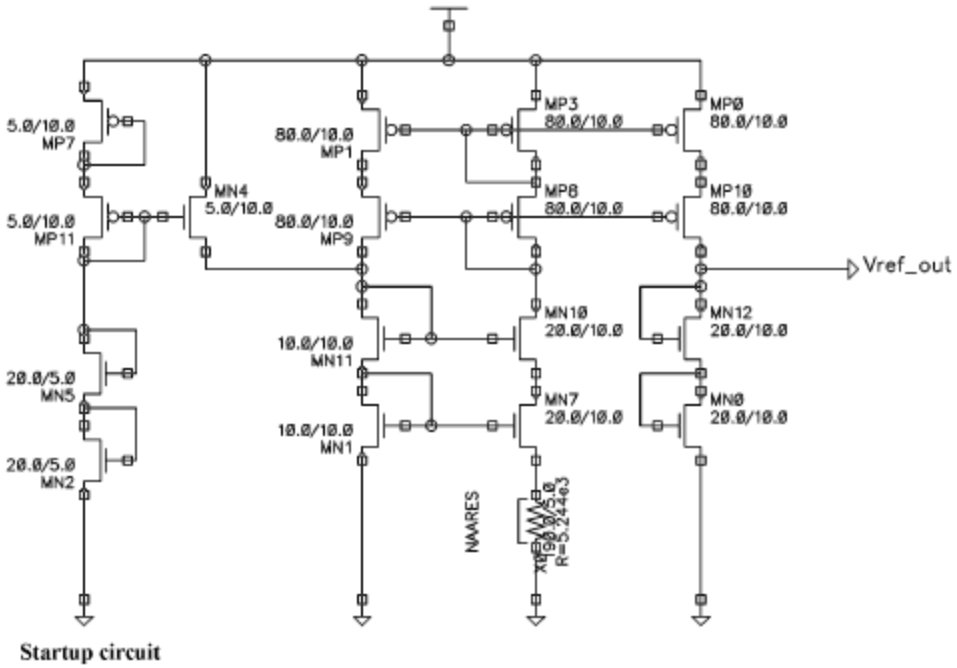


Figure 21 Cascode current reference.

analog performance across variations in voltage. This was attempted by using a cascode version of the beta multiplier shown in Fig. 21. The circuit shows reduced effects of the finite output resistance; however, in a design where Vdd is only 2.5V, the threshold voltage drop across each device makes this reference unusable for this design.

Simulations of the beta multiplier circuit show good performance across PVT (Table 1). To improve the performance across the operation corners, a resistor exhibiting a smaller temperature coefficient would have to be used. However, for the DAC in this design, this reference will suffice.

VDD	Slow Corner	Typical	Fast Corner
2.2 V	37.2 uA	35.3 uA	32.0 uA
2.5 V	44.0 uA	40.0 uA	35.2 uA
2.8 V	49.7 uA	43.6 uA	37.5 uA

Table 1 Current reference performance across PVT.

3.5 DAC DESIGN

The ideal digital-to-analog converter output response is shown in Fig. 22. The

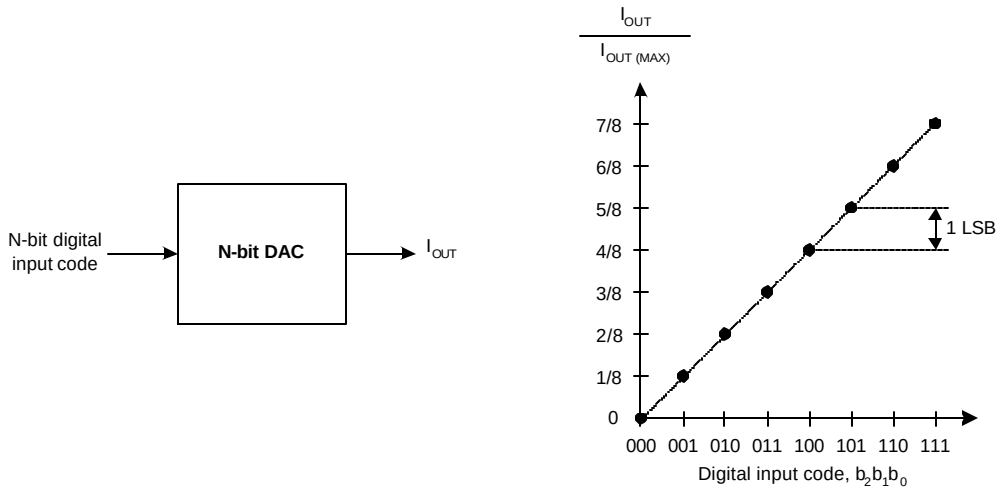


Figure 22 DAC block diagram and ideal transfer curve.

analog output is a current, and is tuned to supply the proper range of control current so that the delay line can move through the correct range of delay. For usage in this DLL, the DAC *must* be monotonic. In other words, for every increase in the input code, the output current increases, and for every decrease in the input word, the output current decreases. As was shown in the simulations of the delay line, the current control versus total delay is already nonlinear, therefore linearity in the DAC is not absolutely necessary, so long as the monotonicity is good. Although not attempted for this design, it would be beneficial to have a DAC design with a nonlinear output response, tuned to the nonlinear response of the delay line. This would maintain consistency in the DLL jitter performance, regardless of the reference clock frequency.

For this design, a six-bit DAC is used. This allows relatively simple implementation, monotonic transfer curves, and good linearity. The DAC will control the delay line in 64 discrete increments.

The most straightforward current-output DAC design uses each bit in the digital input word to switch in a binary-weighted current source (Fig. 23). The minimum current

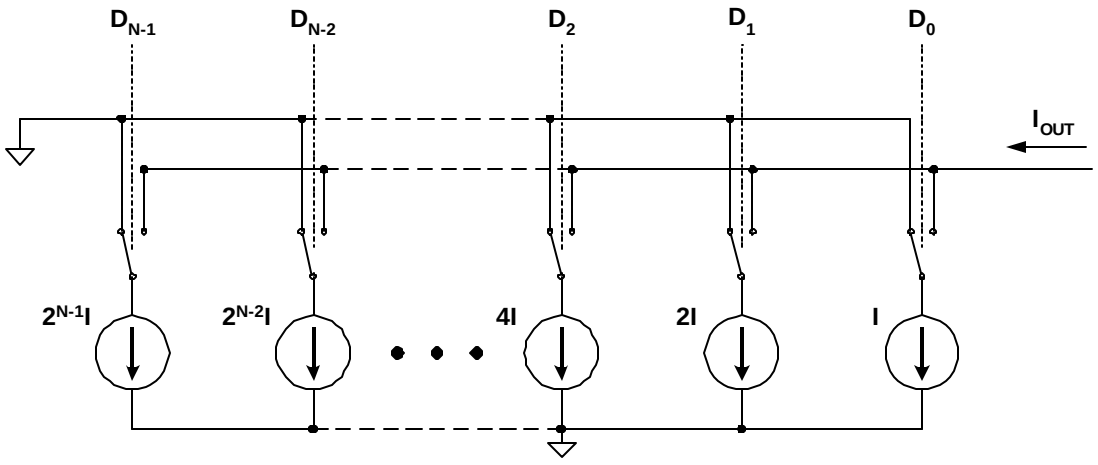


Figure 23 Binary-weighted current steering DAC.

(LSB) is set by *I*.

This design was simulated using the current reference of the previous section to

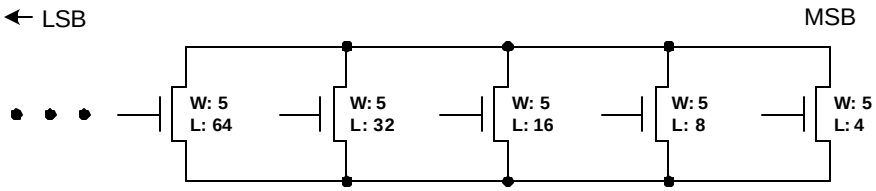


Figure 24 Binary-weighted L current sources.

examine the response. The transistor lengths were used to modify the current sources as shown in Fig. 24. Note that this figure does not show the switches used to gate each source on or off of the output. The simulation results show extremely poor performance (Fig. 25). The output is non-monotonic due to inconsistencies in the current sources. This is caused

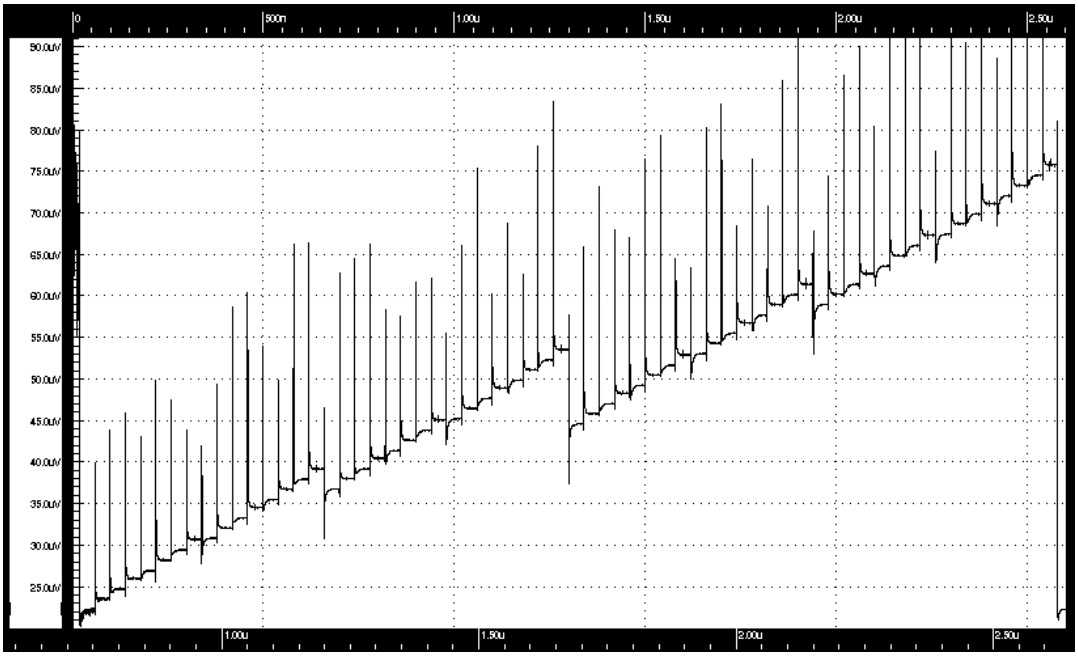


Figure 25 Binary-weighted L DAC transfer curve.

by variations in the percent of lateral diffusion, etc from one source to the next. In addition, the transistor responses are not necessarily linear as the length changes.

In an attempt to improve the monotonicity and reduce the inconsistencies in the current sources, the weighted-L devices may be broken up (Fig. 26). This approach ensures that process parameters such as lateral diffusion affect every device in the same proportions. Other issues arise, however, such as body effect on the stacked transistors.

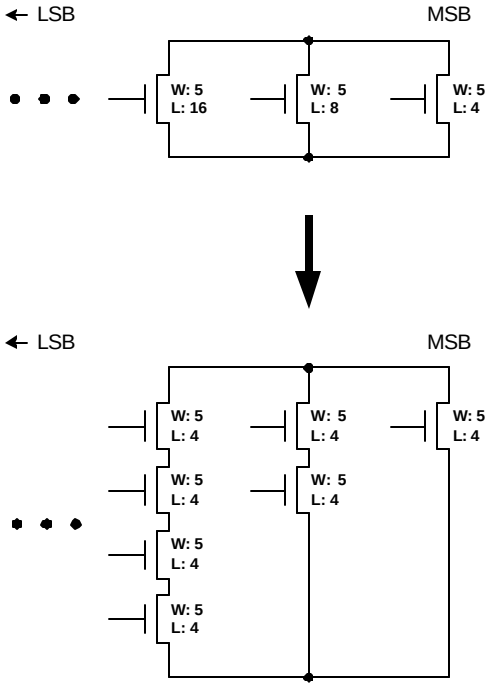


Figure 26 Breaking up the current sources.

Simulation results show that the DAC is now monotonic (continuously

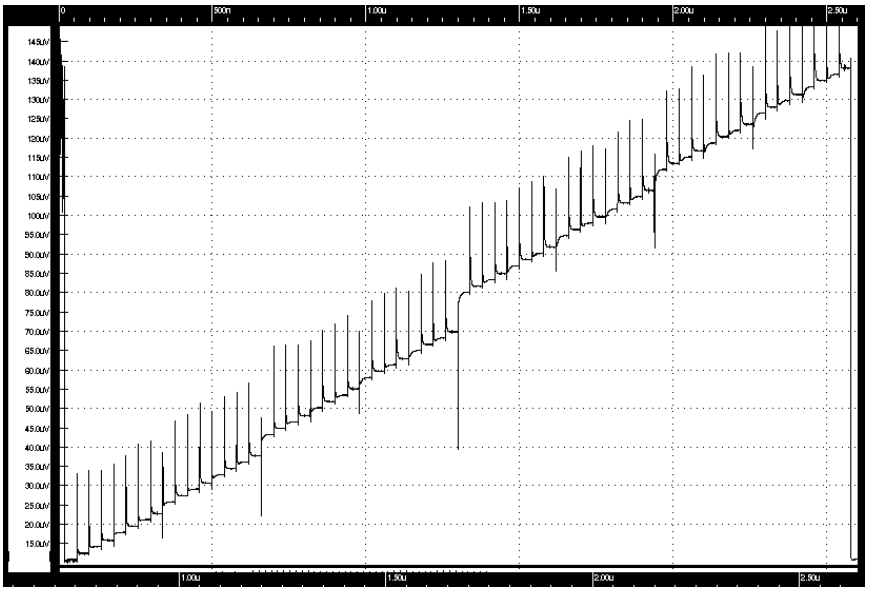


Figure 27 DAC transfer curve with broken-up current sources.

increases), however, the linearity is still very poor (Fig. 27). This is a result of body effect and the nonlinear transistor response of current versus length.

The binary-weighted designs require only *N* current sources, but do not exhibit good linearity in a practical design. The layout also becomes larger when the current sources are broken up as shown in Fig. 26.

In order to achieve good linearity and monotonicity, a non-binary-weighted current-steering approach is used. Fig. 28 shows the basic current steering DAC. Notice that this design requires 2^N current sources. In this case, rather than controlling each current source directly with the bits from the input word, a thermometer code must be generated from the input word [19]. For even a six-bit DAC, it is not trivial to have 2^N, or 64, current sources and to decode the individual controls for each source (thermometer-code controls).

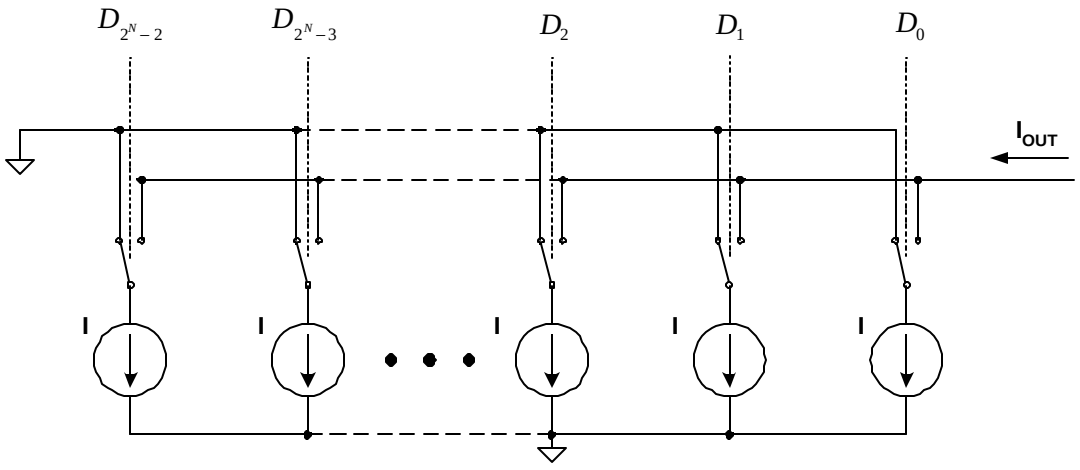


Figure 28 Basic current steering DAC.

As seen in the previous SPICE simulations, these current steering DAC's have the problem of glitches in the output every time the DAC input is changed. This is because all of the sources are in parallel, and it is possible that one source will be on while the next one

switches on [19]. These glitches are faster than the delay line's ability to respond, and are therefore acceptable for this particular design.

The approach used for this design is to use a segmented current steering design. In this scheme, the bits in the lower half of the control word (least significant bits) are used to control three binary-weighted current sources. The bits in the upper half are decoded into thermometer code to control current sources that are all weighted the same. This takes advantage of the linearity in the LSB's of the binary-weighted approach, and relies on the consistency of the MSB's to keep the DAC linear. Fig. 29 shows the block diagram.

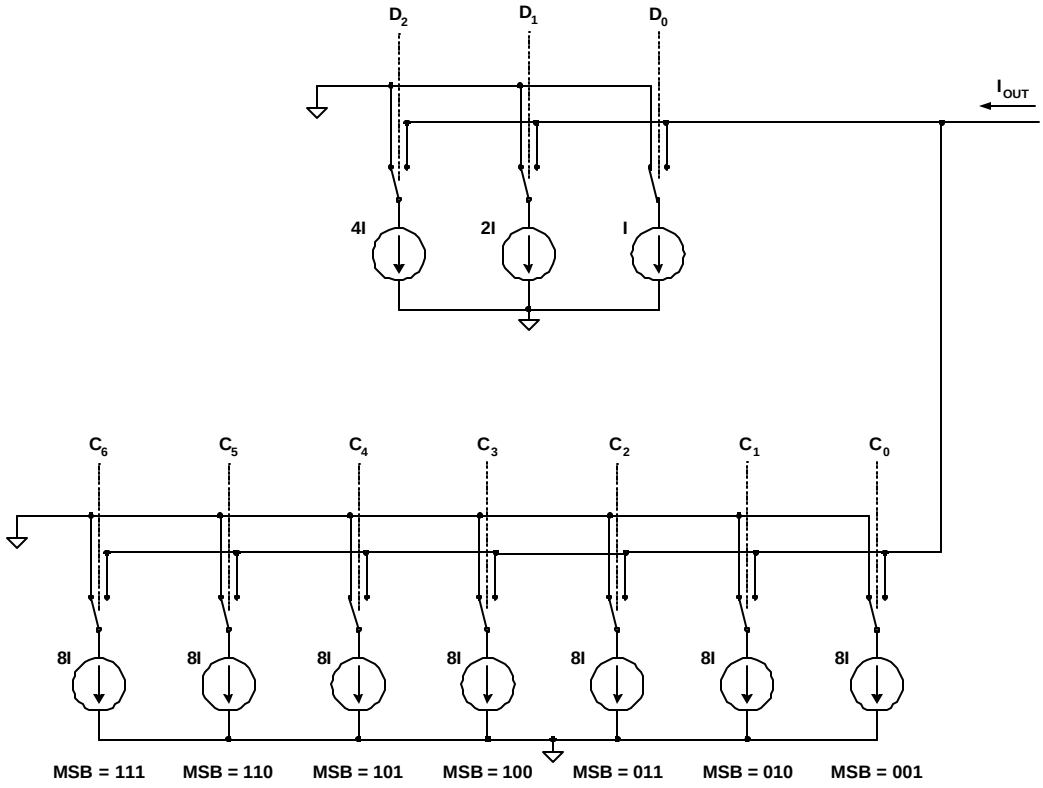


Figure 29 Segmented current steering DAC block diagram.

To set a minimum current, an extra leg is added to the DAC with the control transistor always turned on. Fig. 30 shows the circuit implementation of the segmented current steering design.

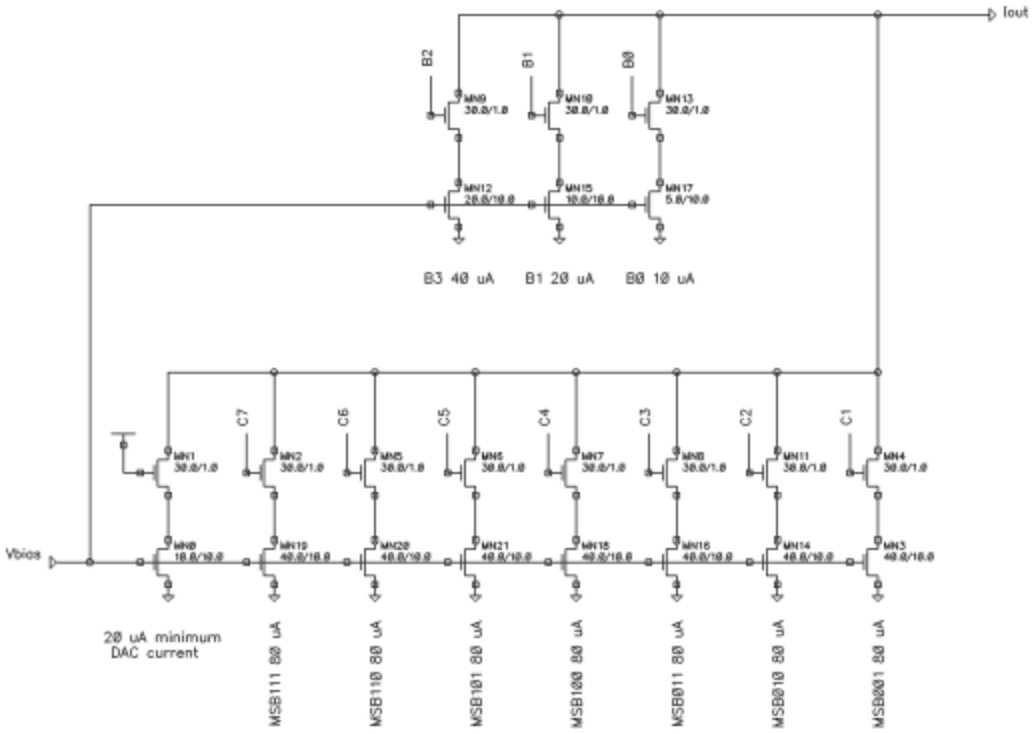


Figure 30 Circuit diagram of segmented current steering DAC.

An observation about the block diagram of Fig. 29 may be made; the circuit can easily be made into a differential design if the current sources were switched from the I_{out} node to a I_{out}* node instead of to ground. A differential design would double the current difference on the output, and would improve the noise sensitivity of the DAC. This was not done for this design.

This DAC cannot directly control the delay line previously discussed. A current mirror is used to provide the necessary current for the DAC and to set the bias voltage for

each analog delay element in the delay line (Fig. 31). The transistor sizing of this current mirror will be discussed later. The additional two legs on the right of the mirror provide

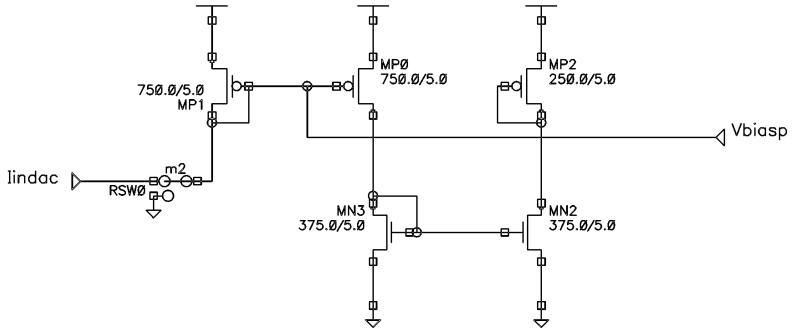
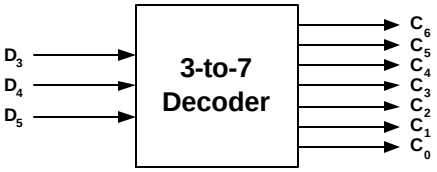


Figure 31 Current mirror for DAC current conversion.

current references for N-channel sources and also a scaled-down reference for P-channel sources. These were used experimentally in this design, but are unused in the final implementation.

The three LSB's of the control word are used to directly switch the three, binary-weighted sources on and off. The three MSB's of the DAC must be decoded to create the thermometer code that controls the seven $(2^3 - 1 = 7)$ equal-weighted current sources. Fig. 32 shows the block diagram, truth table, and logic equations required for the decoder.



D_5	D_4	D_3	C_0	C_1	C_2	C_3	C_4	C_5	C_6
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
1	0	0	1	1	1	1	0	0	0
1	0	1	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

$C_6 = D_5D_4D_3$
$C_5 = D_5D_4$
$C_4 = D_5D_4 + D_5D_3$
$C_3 = D_5$
$C_2 = D_5 + D_4D_3$
$C_1 = D_5 + D_4$
$C_0 = D_5 + D_4 + D_3$

Figure 32 Thermometer-code block diagram and logic equations.

To improve the timing of the decoder so that all seven outputs transition as close together as possible, the various gate sizes were tuned to provide similar delay paths. The current-steering DAC design is already susceptible to glitches on the output, so it is important to ensure that all of the control signals transition simultaneously. This tuning was done by simulating the DAC, looking for significant glitches on the output, tuning the required path, and then reiterating. Fig. 33 shows the final circuit with the gate sizes.

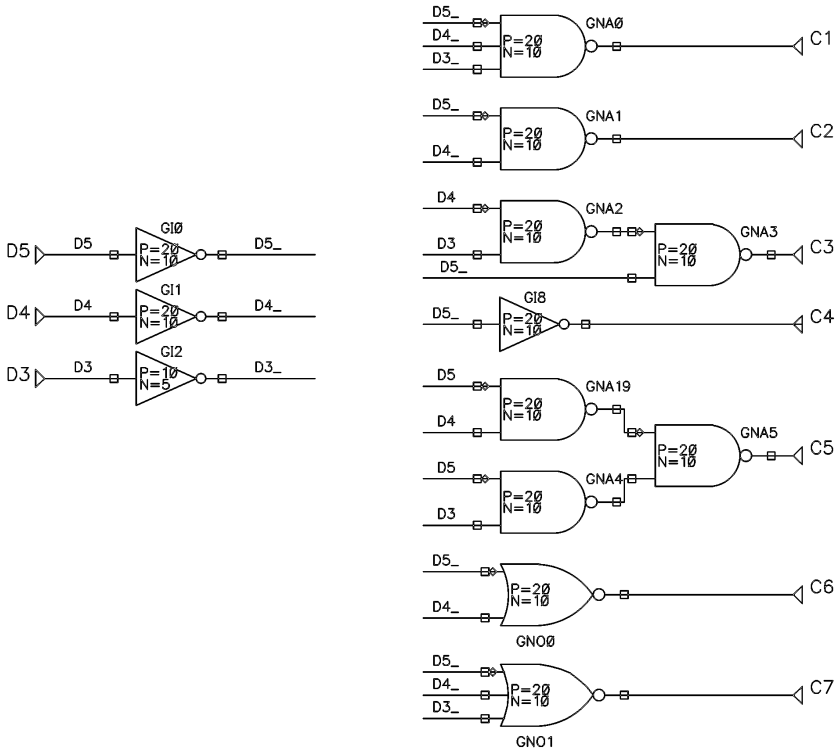


Figure 33 Decoder circuit for segmented current-steering DAC design.

Although the decoder is tuned, the synchronization is not perfect with variations in PVT. For the purposes of this design, the tuning is close enough that the instantaneous glitches produced at the DAC output have a very short duration and can be ignored. If a more perfect synchronization were necessary, all of the control signals, including the

thermometer code and the LSB's, could be latched before being allowed to control the DAC current sources.

3.6 DAC PERFORMANCE

The segmented current-steering DAC has much better linearity and is monotonic. Fig. 34 shows the DAC output at typical process, voltage (2.5V), and temperature (25° C).

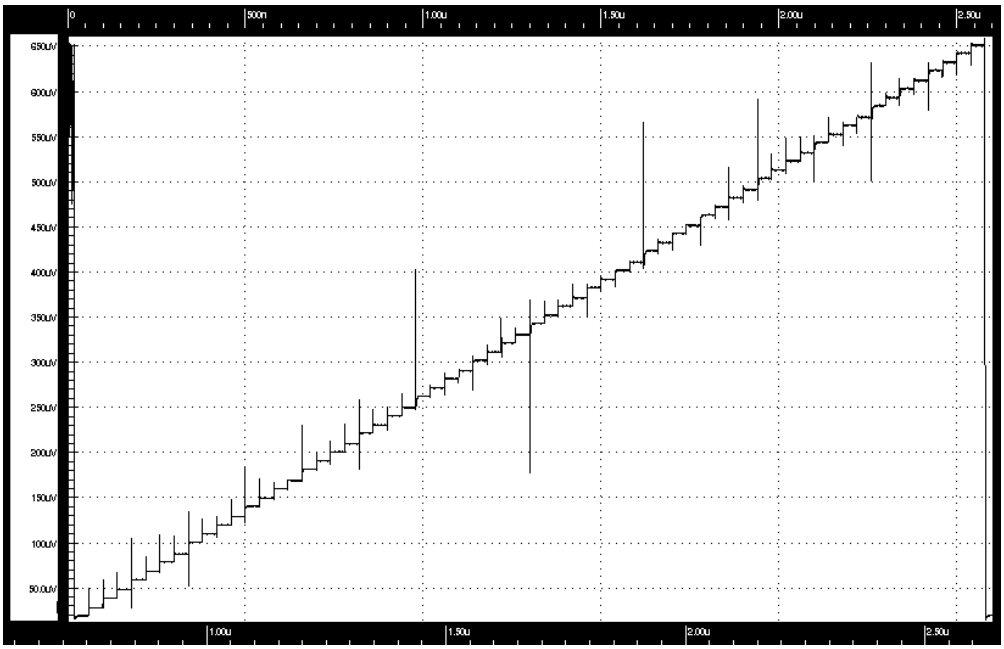


Figure 34 Segmented current-steering DAC transfer curve.

Although the output looks nearly perfect, it is important to quantitatively characterize the DAC linearity. Due to the non-ideal nature of real devices, a DAC will exhibit slight differences between the ideal and actual performance. For data converter characterization, this difference is knows as *differential nonlinearity* (DNL) [19]. It is defined as

$$DNL_n = Actual\ increment\ height\ of\ transition\ n - Ideal\ increment\ height \quad (7)$$

The number *n* refers to the corresponding digital input code transition. When the DNL of a converter is defined, it will be the worst-case DNL across the entire range of operation. For the converter to be termed as accurate to N bits, the DNL must be less than $\pm 1/2$ LSB.

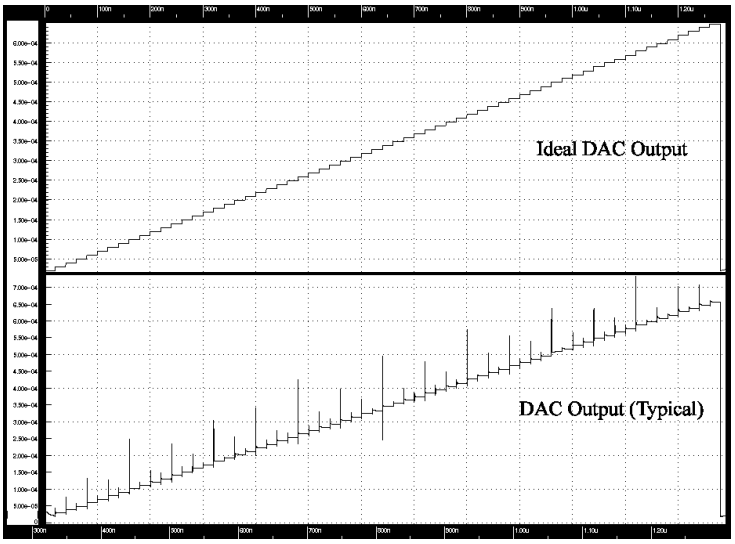


Figure 35 Ideal DAC output vs. typical DAC output.

Fig. 35 shows the ideal DAC output and the actual DAC output (at typical PVT) separately.

Fig. 36 shows that there is a small amount of nonlinearity in the DAC. To calculate the DNL as defined by Eq. (7), the ideal LSB value is subtracted from the actual step height of each output level.

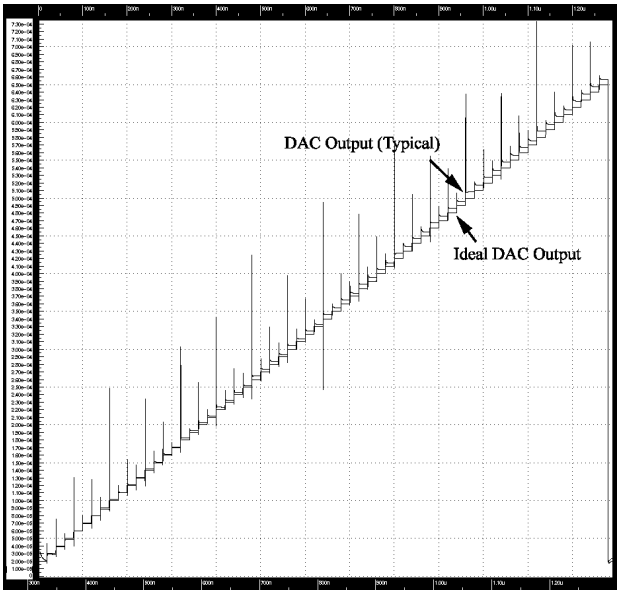


Figure 36 Ideal output overlaid on typical output.

The results are shown in Fig. 37. Notice that the error due to glitches (from the switching input code) has been ignored and only the steady-state difference is used. The magnitude of the DNL is +0.30, -0.11 LSB for this design. This means that the DAC is accurate to 6-bits with regard to linearity.

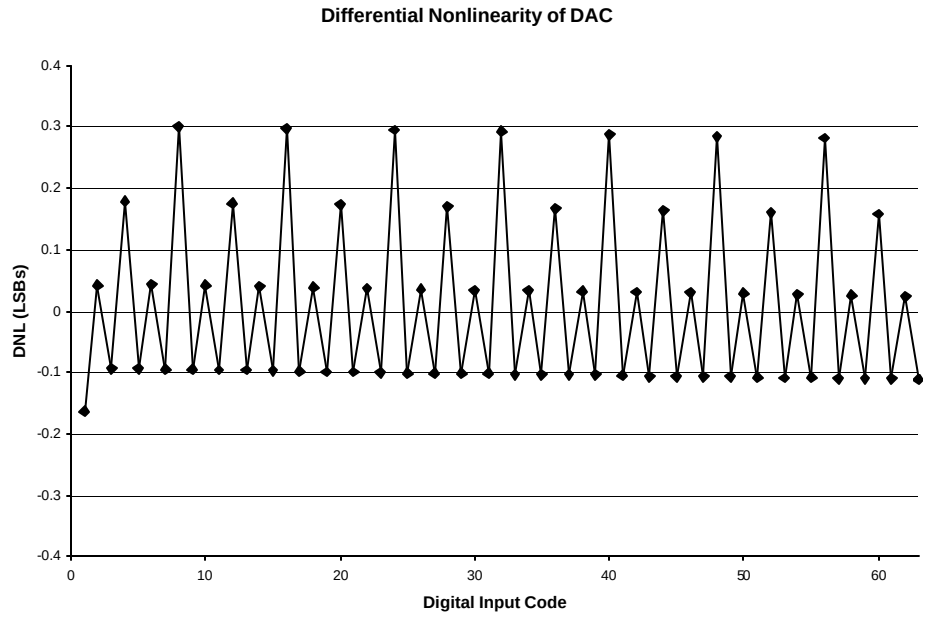


Figure 37 Differential nonlinearity of DAC.

As previously stated, the primary design goal for the DAC is monotonic performance; if the outputs do not move in the same direction as the input code, it will be possible for the DLL to never find a proper lock point. It is easy to see from the DNL shown in Fig. 37 that the DAC is indeed monotonic because the DNL is never less than –1 LSB. In other words, the difference from the actual to the ideal value is never going to cause the output to decrease for an increasing input. The most negative DNL of this design is –0.11 LSB.

Further examination of Fig. 37 shows that the greatest contributor to the DNL of this DAC design is due to the switching of the three LSB's. The three lowest bits were implemented using binary-weighted current sources. The slight error in the sizing of the transistors used as current sources causes the DNL error. Notice that the DNL response is periodic with the eight values created by the three LSB's. Without the error induced by these current sources, the DNL has a slightly negative trend.

As was shown in section 3.3, the total delay line delay has a nonlinear response to a linear control input. Therefore, the slight differential nonlinearity of this DAC is of no consequence for this design. In fact, in order to get a linear response in the total delay time, a non-linear DAC would have to be designed.

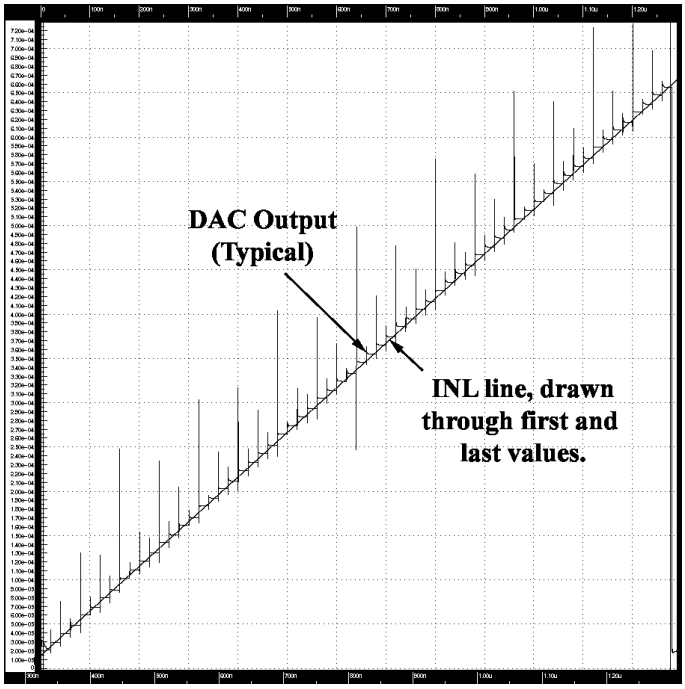


Figure 38 Typical output shown with line used for INL calculation.

Another important DAC characteristic is the *integral nonlinearity* (INL) [19]. This is defined as the difference between the converter output and a straight line drawn between the first and last output values (Fig. 38). This characteristic defines the linearity of the overall transfer curve. The equation used to calculate INL is as follows:

$$INL_n = Output\ value\ for\ input\ code\ n - Output\ value\ of\ reference\ line\ at\ point\ n \quad (8)$$

Fig. 39 shows the integral nonlinearity for this DAC design. Again notice that the glitches have been ignored. The INL is also greater than $\pm 1/2$ LSB, making the converter accurate to only 5 bits. Again, this will not affect the DAC performance as used in the DLL.

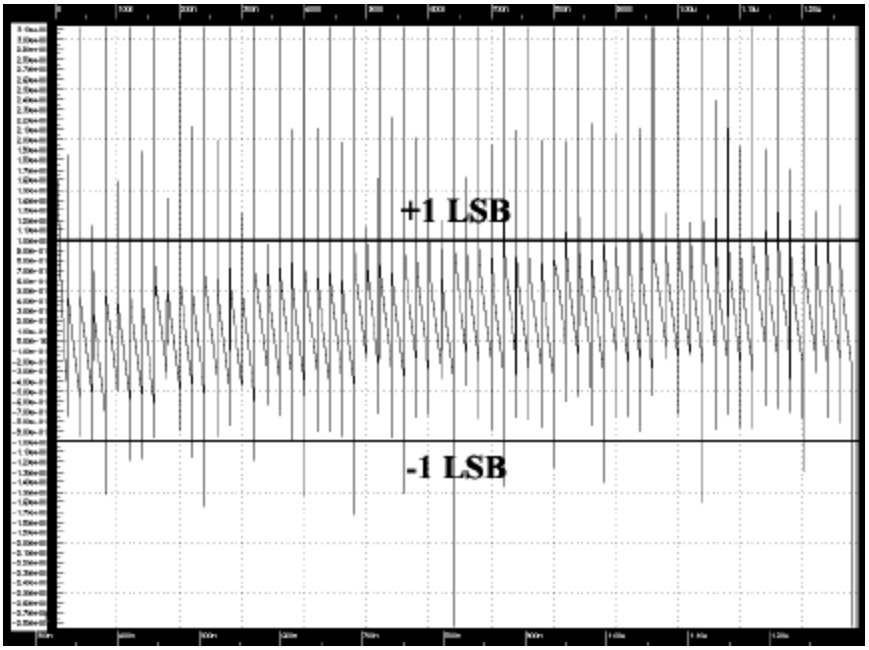


Figure 39 Integral nonlinearity of DAC

When the current mirror of Fig. 31 was presented, the transistor sizing was not explained. The left-most transistor of this reference acts as a supply to all of the current sources in the DAC. As the input word increases, more sources are switched into the

circuit, pulling more current from this source. When the DAC was first being evaluated across PVT corners, poor results were observed at the slow corner (Fig. 40).

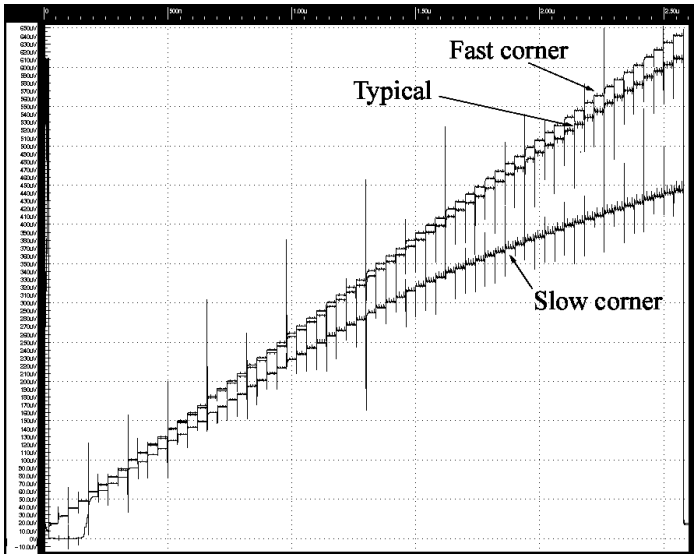


Figure 40 Poor DAC performance at high current.

The source transistor in the mirror was simply not large enough to provide the required current for high-valued DAC input words. This effect was exaggerated at the slow corner, where the transistor currents

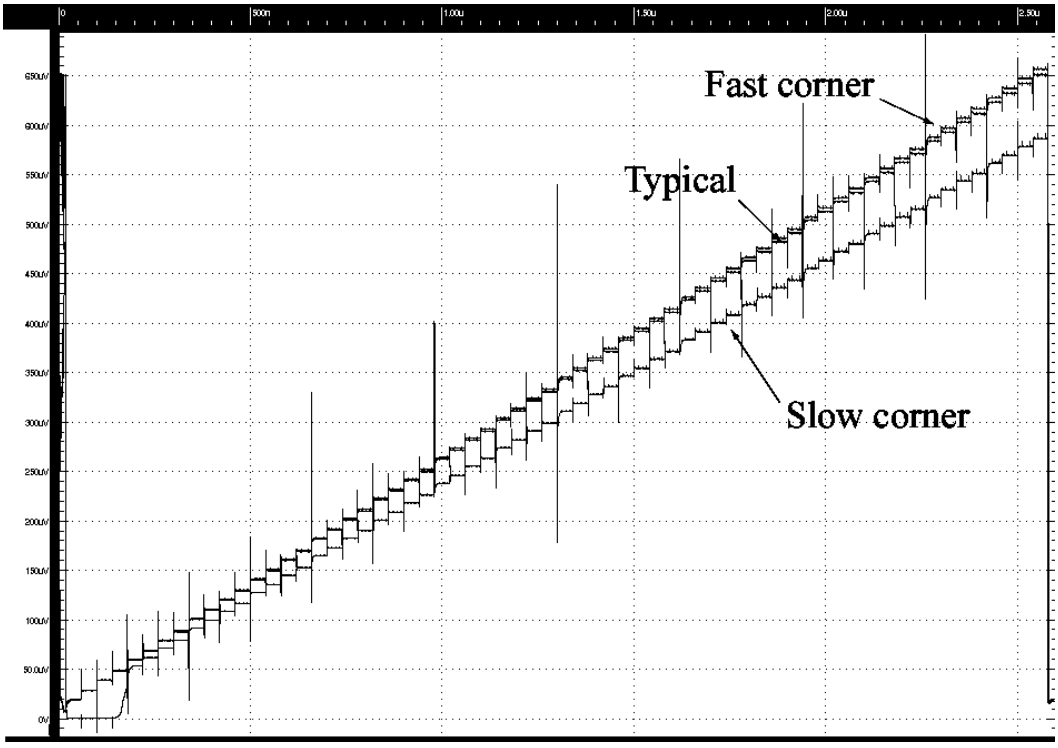
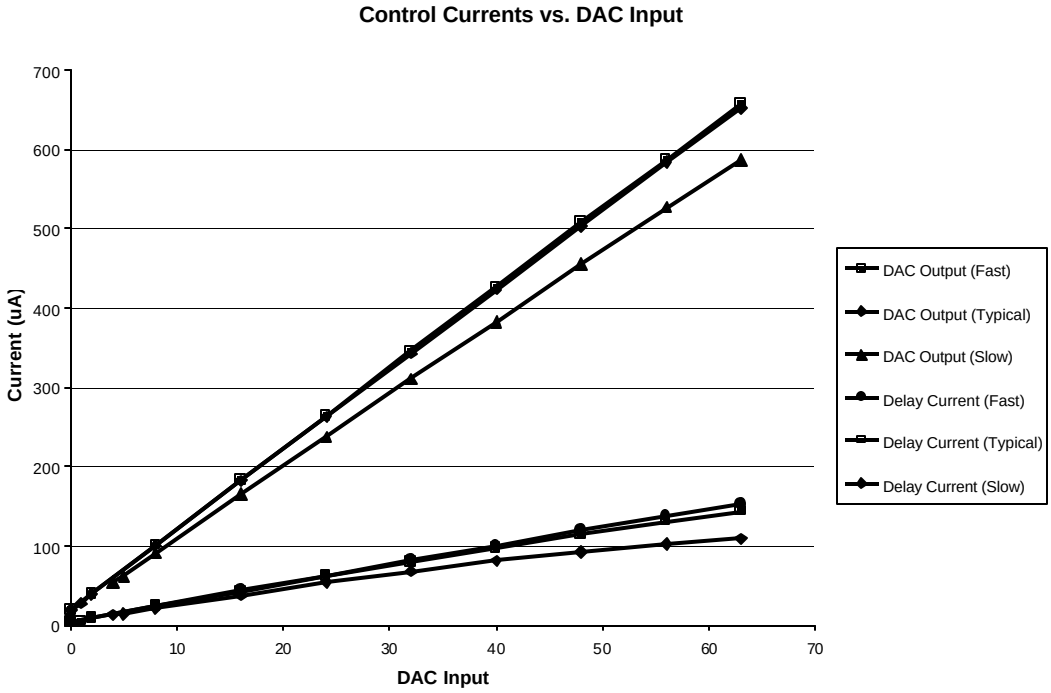


Figure 41 DAC transfer curves across PVT.

are reduced due to high temperature and lower supply voltage. After the source transistor width in the current mirror was sized up to the dimensions shown in Fig. 31, the DAC performance improved significantly. Fig. 41 shows the final DAC performance across the PVT corners.

All of the SPICE output shown to this point does not clearly show the absolute values of the DAC output, nor have the delay line currents been shown. Graph 4 shows the DAC output current as well as the current mirrored into the delay-line elements vs. the DAC input word. This graph also shows the performances across PVT.



Graph 4 Control currents vs. DAC input.

It should be clear at this point just how critical the operation of the current reference used for the DAC has become. Any variation in the reference as the supply voltage,

temperature, and/or process varies will propagate through the DAC and into the delay-line control current.

With good performance across PVT and a monotonic transfer curve, this DAC is well suited for operation in the DLL.

3.7 UP/DOWN COUNTER DESIGN

Referring back to Fig. 14, the DAC is controlled by an Up/Down counter. In effect, this counter is the 'filter' for the DLL. It is essentially the loop filter that is used in PLL design and pure analog DLL design, except that it moves in a linear fashion, preserving the first-order nature of the DLL. Fig. 42 shows the basic counter interface to the DAC.

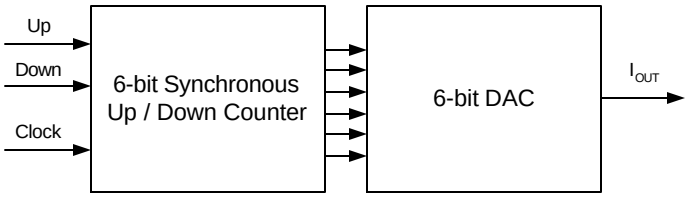


Figure 42 Up/Down counter interface to DAC.

Although the DAC has very fast glitches in the output due to the nature of the design, it is important that the additional noise not be introduced by feeding an unsettled control signal to the DAC. If a simple ripple counter were to be used to count up (Fig. 43),

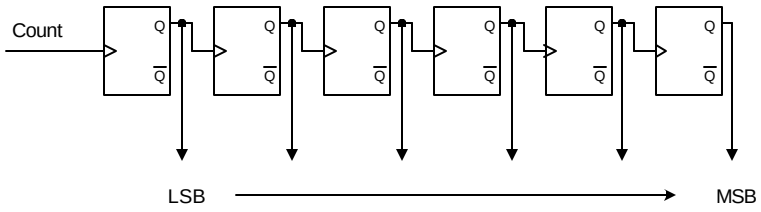


Figure 43 Basic ripple counter.

a finite time must pass before all of the outputs are valid. The output from the LSB stage must propagate all the way through to the output of the MSB stage before the output can be used. If this type of counter were used to directly control the DAC, the glitches would now be of significant width and could no longer be ignored for this design.

To ensure that the counter outputs were completely synchronous, the outputs might simply be passed through a set of latches, which would be clocked after the ripple counter settling time had passed (Fig 44). This is a simple solution that occupies relatively little layout area. However, this counter can only count up, not down.

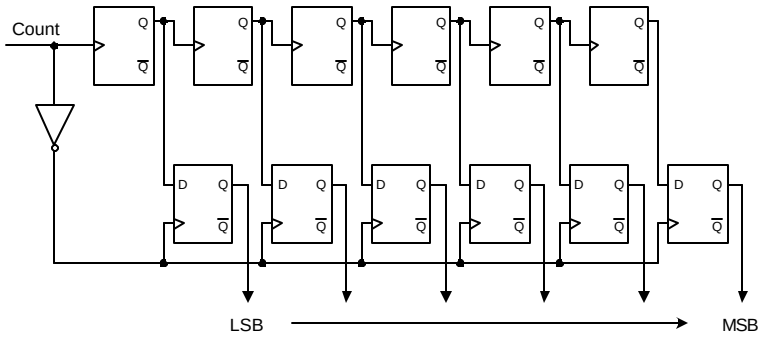


Figure 44 Clocked ripple counter.

Of course, a fully synchronous counter can be designed as a finite state machine (FSM) with all of the 'next' states fully decoded at the inputs of each latch. This design approach guarantees synchronous operation, but the logic required to decode the 'next' states increases exponentially as the number of bits in the counter increase. This might still be a good solution to implement a counter that can count up or down as long as the number of counter bits is relatively small.

For a synchronous counter, consider a design similar to that of Fig. 44, except that rather than a ripple counter used as the input to a register, pipeline logic could ripple

through and determine the next state of the output latches. This concept is shown in the block diagram of Fig. 45. Notice that the output latches are toggle flip-flops and the setup logic simply determines whether or not the bit that it controls will toggle on the next clock cycle.

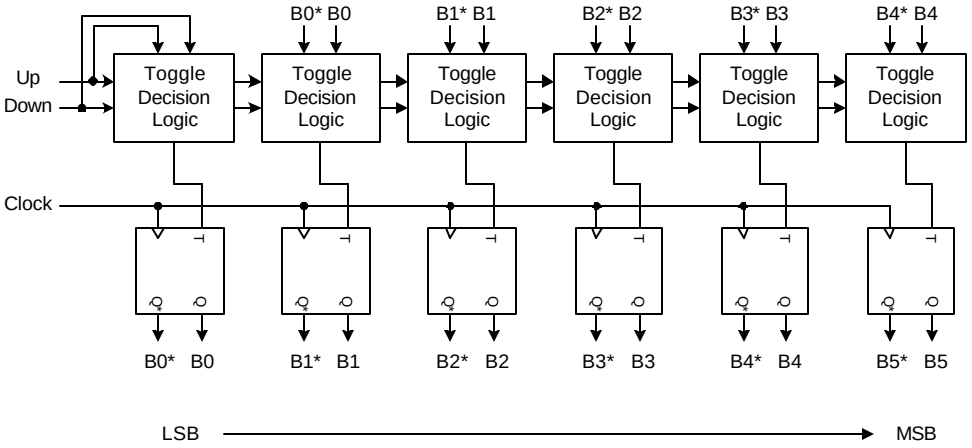


Figure 45 Block diagram of synchronous up/down counter.

This design still requires a finite settling time before the output latches are clocked, but the setup-logic does not increase as bits are added to the counter. If additional bits are required, additional toggle latches and setup-logic are added, and a little more propagation setup time is required.

To develop the toggle decision logic, the truth table of Table 2 must be used. The table is an example of a three-bit counter, showing the state of the up and down control signals, the current bit count, and whether or not a bit will toggle on the next clock cycle (BoutN). The setup logic is implemented as a logic state that will ripple through, depending on the current count and the state of the up and down controls.

This is done by logically ANDing the bits of lower significance in the current count. If all of the lower bits are asserted, it is time to toggle the next bit. The same is done for both counting up and counting down, except that the logic levels are changed.

Down	Up	B2	B1	B0	Bout0	Bout0*	Bout1	Bout1*	Bout2	Bout2*
0	1	0	0	0	1	0	0	1	0	1
0	1	0	0	1	1	0	1	0	0	1
0	1	0	1	0	1	0	0	1	0	1
0	1	0	1	1	1	0	1	0	1	0
0	1	1	0	0	1	0	0	1	0	1
0	1	1	0	1	1	0	1	0	0	1
0	1	1	1	0	1	0	0	1	0	1
0	1	1	1	1	1	0	1	0	1	0
1	0	1	1	1	0	1	1	0	1	0
1	0	1	1	0	0	1	0	1	1	0
1	0	1	0	1	0	1	1	0	1	0
1	0	1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	1	1	0	1	0
1	0	0	1	0	0	1	0	1	1	0
1	0	0	0	1	0	1	1	0	1	0
1	0	0	0	0	0	1	0	1	0	1

Table 2 Counter toggle logic truth table.

If a single AND gate were used to determine whether or not the current bit should toggle, the gate would get very large as more bits were added to the counter ($n-1$ inputs on the AND gate). Also, two of these gates would be required on the input of every toggle flip flop (one for up and one for down). Instead of using a single gate, the AND function can be carried out by propagating through a series of two-input ANDs. This means that for the nth toggle flip-flop, a logical AND of $n-1$ will still be calculated, except that additional propagation time will be required. Fig. 46 shows a single cell that can be cascaded to create the AND function for both the up and down directions.

The input, BAND, is simply the AND function from the lower bit ($n-1$). It is ANDed with the output of the current (nth) bit, BN. The output, if asserted, will cause the

current (nth) flip-flop to toggle. Also, this output is sent to the next counter logic cell ($n+1$). The same is done for the logic used to count down, but the assertion levels are changed.

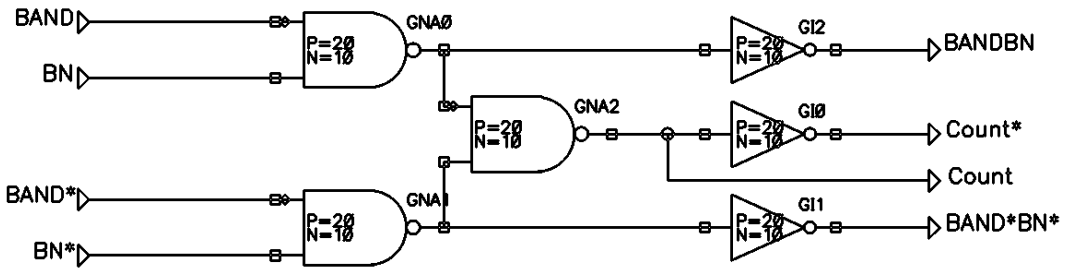


Figure 46 Cascaded AND generator for counter toggle logic.

Fig. 47 shows how the cells are cascaded in front of each flip-flop bit in the counter. This figure shows the entire six-bit synchronous circuit. Each bit is preset to a logic one upon initialization. This is required for DLL initialization, which will be discussed later. Note that an extra toggle logic cell and a regular flip-flop are used (bottom of schematic). This is done to generate a 'carry out', or overflow signal that may used by the DLL control circuitry to detect when the counter (and therefore the DAC) has reached a limit.

The schematics for the toggle flip-flop and the regular flip flop are shown in Fig. 62 and Fig. 63 in the Appendix.

The six-bit counter will have a maximum count frequency limited by the propagation time through the toggle decision logic. Normally, this would be a five-gate delay, but with the overflow detection, a sixth gate delay is added. With a gate-propagation delay of ~100ps per gate, this counter is more than fast enough for the purposes of this design. Fig. 48 shows the SPICE simulation waveforms for the counter.

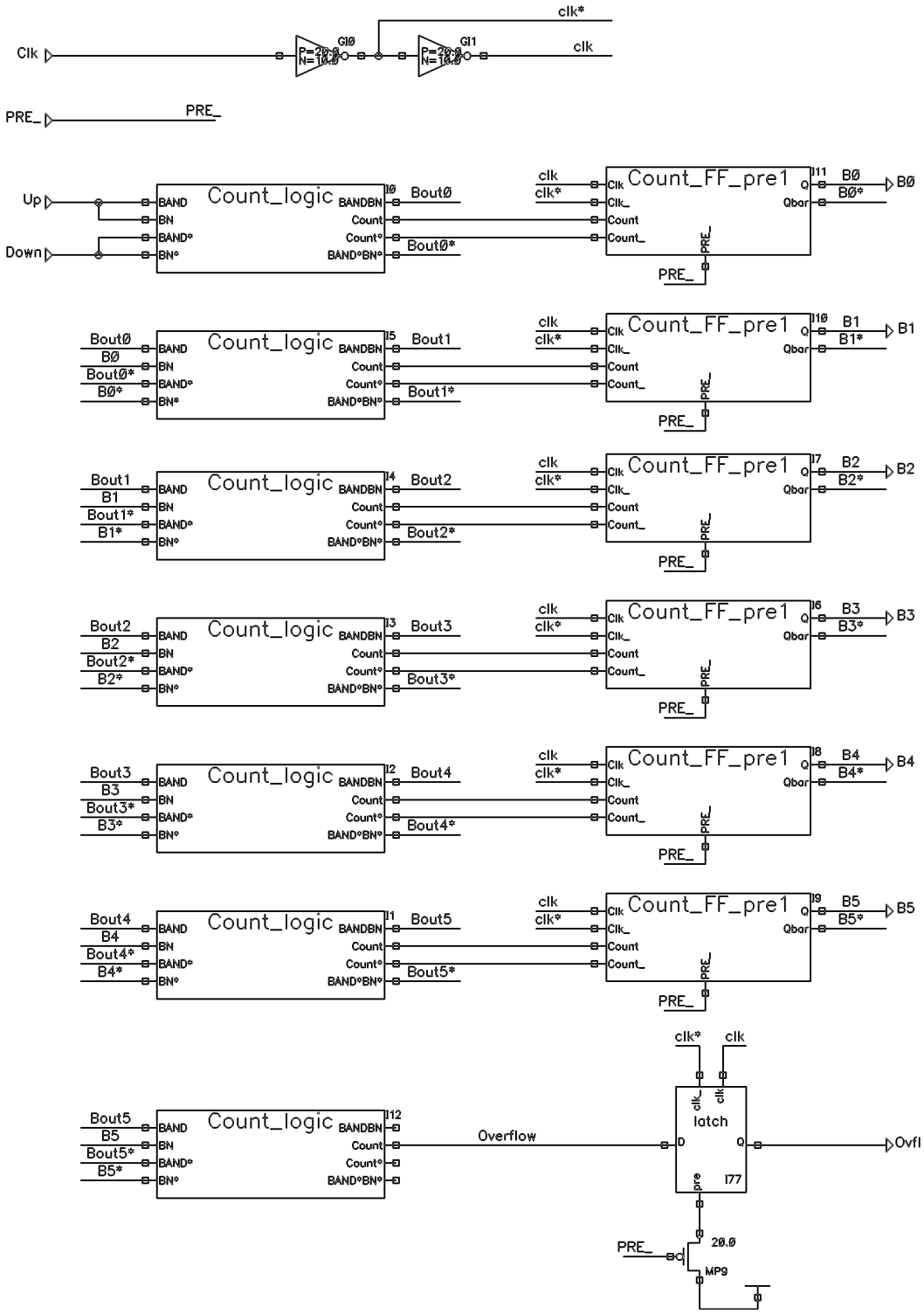


Figure 47 Circuit diagram of synchronous up/down counter.

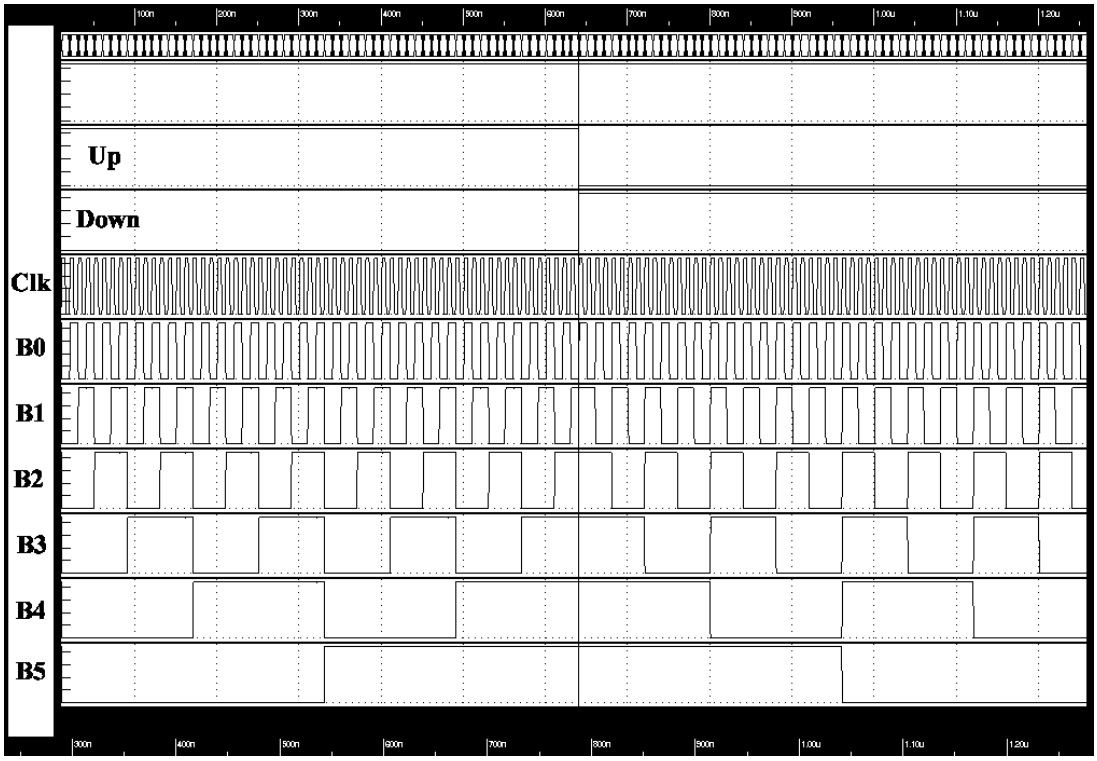


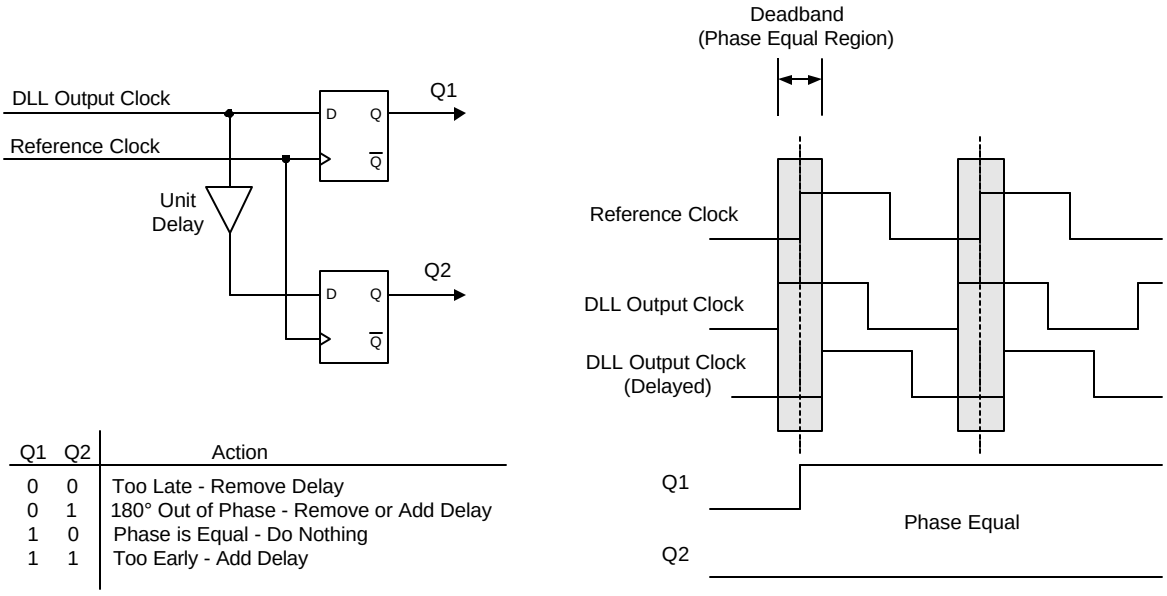
Figure 48 Synchronous counter waveforms.

3.8 PHASE DETECTOR DESIGN

The phase detector for a DLL may be more simple that that used in a PLL, because only phase must be determined, not frequency. However, the detector is critical to the overall performance and the output phase error [20], [21], [22].

As was previously discussed, in order to reduce the jitter and error in the output, it is desirable to use a delay line that can be changed in extremely small increments. Eq. (4) showed that the maximum phase error is a direct function of the smallest increment in which the delay line may be changed. However, this equation, as well as Eq. (1), *assumes* that the phase detector is maintaining the DLL loop delay at exactly $N \cdot tCK$.

Although the smallest increment in which the phase detector can control the loop is set by the delay line, it is important that the phase detector not contribute to the total error. This can be illustrated by examining the simple phase detector of Fig. 49.

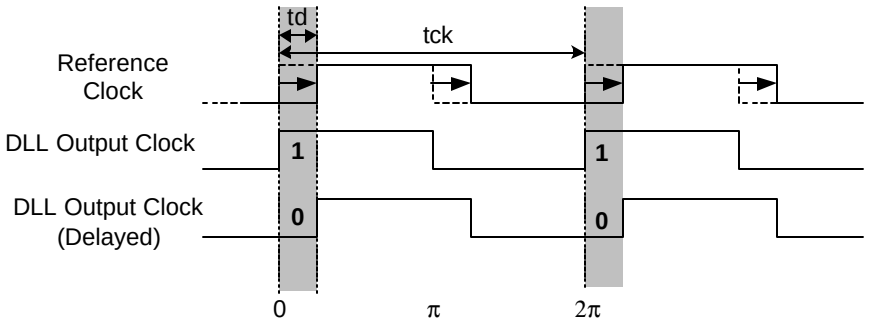


Q1	Q2	Action
0	0	Too Late - Remove Delay
0	1	180° Out of Phase - Remove or Add Delay
1	0	Phase is Equal - Do Nothing
1	1	Too Early - Add Delay

Figure 49 Simple phase detector with deadband.

The unit delay of this detector is set to be identical to the minimum delay increment of the delay line. If set properly, the deadband created by the 'phase equal' region will not contribute to the final DLL phase error. If the delay is set too short, the DLL lock point can oscillate between two lock points, and therefore cause jitter in the output. If the delay is set too long, the deadband behavior will become apparent, and the phase detector will require larger changes in the output phase before taking any action. This will directly create phase error in addition to that calculated in Eq. (4).

The deadband of this phase detector is calculated using waveforms of Fig. 50. An

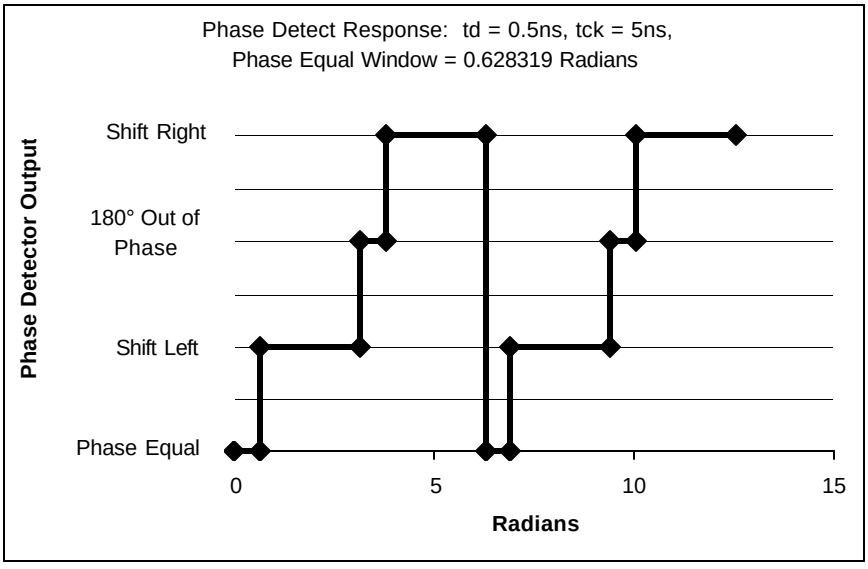


Phase is 'equal' across the phase range:

$$0 \rightarrow \frac{td}{tck} \cdot 2p \qquad \Delta = \frac{td}{tck} \cdot 2p$$

Figure 50 Deadband in the phase equal range.

example of deadband calculation can be seen in Graph 5. For this curve, a unit delay of 0.5ns is used. Because phase is a function of clock period, the frequency of interest must be defined. For Graph 5, a clock period of 5ns is used.



Graph 5 Example phase detector response.

At this point, it should be clear that this simple phase detector could contribute to the total phase error of a DLL system if the deadband is too great. This is not an issue for digital DLL's, where the digital unit delay of the delay line is easily modeled in the detector. However, for an analog continuous delay line, this deadband would not be acceptable, as it would be the limiting factor in the phase performance.

The delay line used in this DLL design is analog. The minimum change in delay is set by the LSB of the controlling DAC. Because the LSB of the DAC can control the delay in finer increments than the delay of a CMOS gate, it is not reasonable to use the simple phase detector previously discussed. The resolution of the detector must be reduced. In this case, a circuit known as an arbiter is a good solution (Fig. 51).

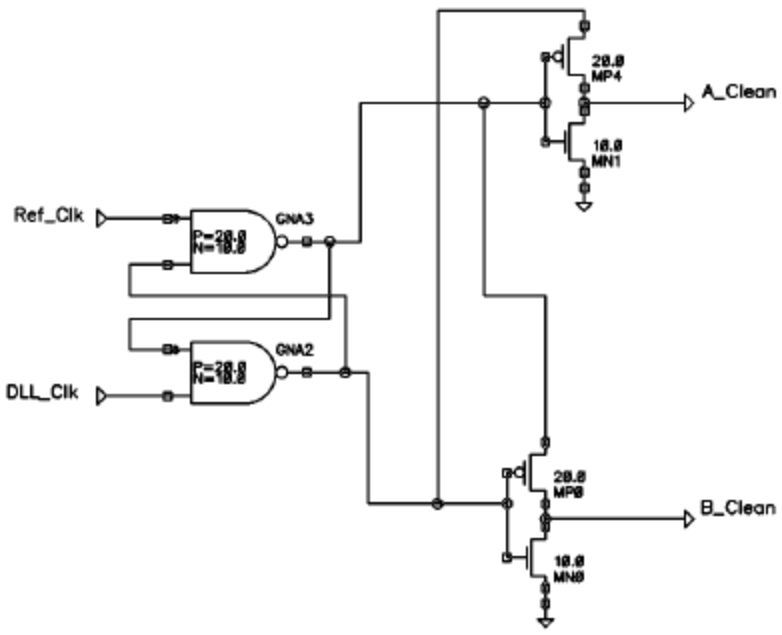


Figure 51 Two-way arbiter.

This circuit simply determines which of the two inputs occurs earliest in time. If Ref_Clk occurs slightly before DLL_Clk then the output A_Clean will go high, and the

output B_Clean will stay low. If DLL_Clk occurs before Ref_Clk, then the output B_Clean will go high while A_Clean stays low. Note that the output inverters get their power supply from the outputs of the Set-Reset (SR) latch. This ensures that A_Clean and B_Clean can not both be high at the same time. When designed properly, this circuit can discriminate between edges with only tens of picoseconds of separation [10].

To create the proper control signals for the up/down counter, a bit of signal conditioning is required. The arbiter cannot directly control the counter. The A_Clean and B_Clean signals do not have good duty cycle and may have glitches present from the times when the inputs return low. The up/down control signals are generated by latching the arbiter outputs as shown in Fig. 52. The SR flip-flop used to create the latch clocks is necessary to remove glitches before triggering the output latches.

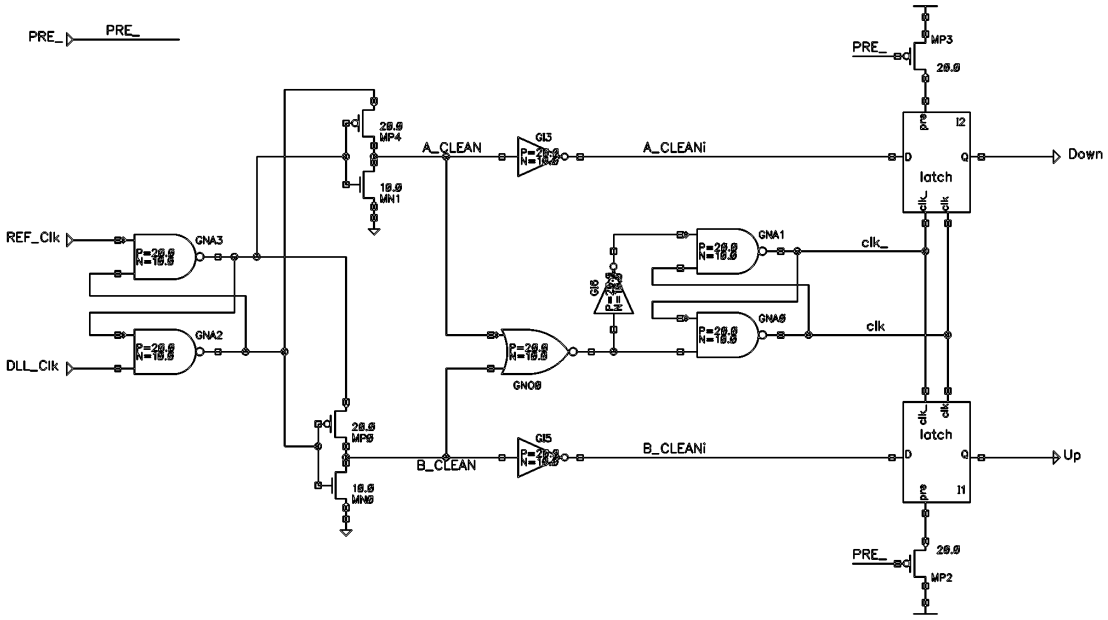


Figure 52 Arbiter-based phase detector circuit.

The operation of this phase detector is apparent from the simulation waveforms shown in Fig. 53. Note how the SR flip-flop used to create the 'clk' signal successfully eliminates the glitches seen on the NOR output. Also note that the detector properly detects the phase difference in the two input signals.

Figure 53 Arbiter waveforms.

It should be noted that this arbiter will never allow the DLL to lock in a steady state. The counter will always count either up or down, causing an oscillation in the output phase. This is similar to the situation where the unit delay is too short in the simple phase detector previously discussed. However, the LSB control of the delay line is expected to be small enough as to be negligible for this design. The ideal design would have an infinitely small delay per unit change in control, and the oscillation caused by the arbiter would not be a consideration. Although not implemented in this design, it would be a simple matter to filter the arbiter outputs and not allow the counter to change unless a sequence of more than

one ‘up’ or ‘down’ signal was detected. Again, this was not considered an issue for this design due to the small delay increment.

3.9 DLL CONTROL CIRCUITRY

Up to this point, the actual direction dictated by the phase detector has been ignored. The waveforms of Fig. 53 show that if DLL_Clk occurs before Ref_Clk, then the counter is to count down. When DLL_Clk occurs after Ref_Clk, the counter is to count up. In addition, the preset condition of the counter has not been discussed.

To determine the count direction, it is important to realize that for this design, as the counter value increases, the delay in the delay line decreases. Therefore, when the arbiter detects that the reference clock signal occurs after the fed-back DLL output, additional delay is required to make the phase equal, and the counter must count down. If the reference clock occurs before the DLL output, then too much delay exists, and delay must be removed. This is accomplished by counting up.

It is important to realize that because the phase detector maintains the delay around the loop at $N \cdot tCK$, any instantaneous changes to the delay line will not be seen at the phase detector inputs for approximately $N \cdot tCK$. This means that if a command to count up (remove delay) is given, it will take *at least* one cycle for the phase detector to see that the delay has been properly removed. If additional commands continue to be given, the phase detector can overcorrect by giving too many count commands. To eliminate this problem, the arbiter output can be sampled every other cycle (assuming that the loop delay is only $1 \cdot tCK$). If the loop delay is greater than one clock period, then the arbiter decision must be sampled even more slowly. Fig. 54 shows the divide-by circuit used to sample the

arbiter output. Instead of directly sampling the arbiter output, the clock frequency used to control the counter is simply reduced. Note that the metal-layer switches shown in Fig. 54 will allow the circuit to be used as a divide-by-two or a divide-by-four, depending on the typical loop delay. This directly reduces jitter in the DLL output.

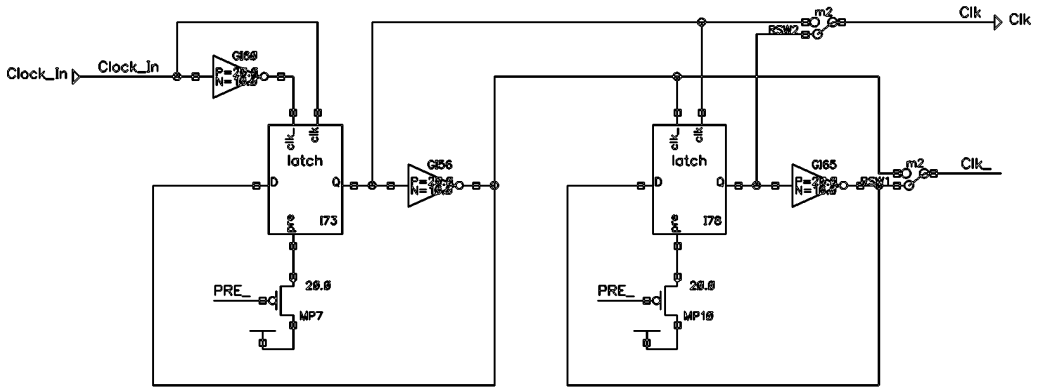
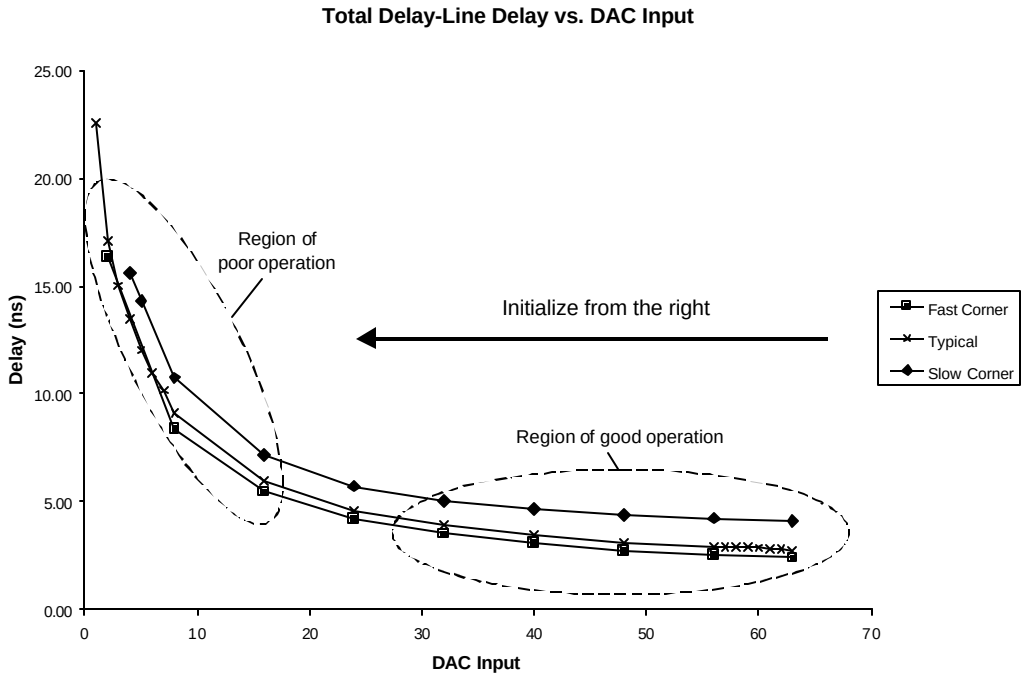


Figure 54 Configurable divide-by circuit.

In addition to the divide-by circuit, the DLL control circuitry must include the ability to properly initialize the system to the optimum lock point. At initialization, the controls must force the counter to move in the proper direction until a lock point is found. Once found, the DLL must be released to naturally track phase differences in the output. The controls must be able to detect when the counter has shifted to an extreme, and either reset the DLL or stop additional counting from occurring.

Due to the nonlinearity in the delay vs. control curve (Graph 6), and the fact that a linear DAC is being used to set the delay, the DLL performance will be poorest when the delay line is operating at a maximum delay. The DAC will always move in LSB steps (due to its linearity), but the 1/x response in the delay time will move in larger increments as the

delay elements become more starved of current. This means that for the DLL to lock in the region of best performance, the controls must ensure that the lock point is searched for from the right-most end of the delay curve. In other words, the delay must initially be set at a minimum (delay line current supply is large), and the counter must be forced to count down until a locked condition is detected (Graph 6). If this were not done, it would be possible for the loop delay to lock to a higher multiple of the reference clock period, in the region of poor operation.



Graph 6 Delay vs. input control, showing direction for proper initialization.

The initialization control circuitry is shown in Fig. 55. The PRE_ input is an asynchronous system reset signal, active low. This signal will initialize the counter to the highest count possible (all ones) and will reset the divide-by circuit and the phase-equal

detector. The Ovfl input is the overflow signal generated by the up/down counter whenever the counter rolls over.

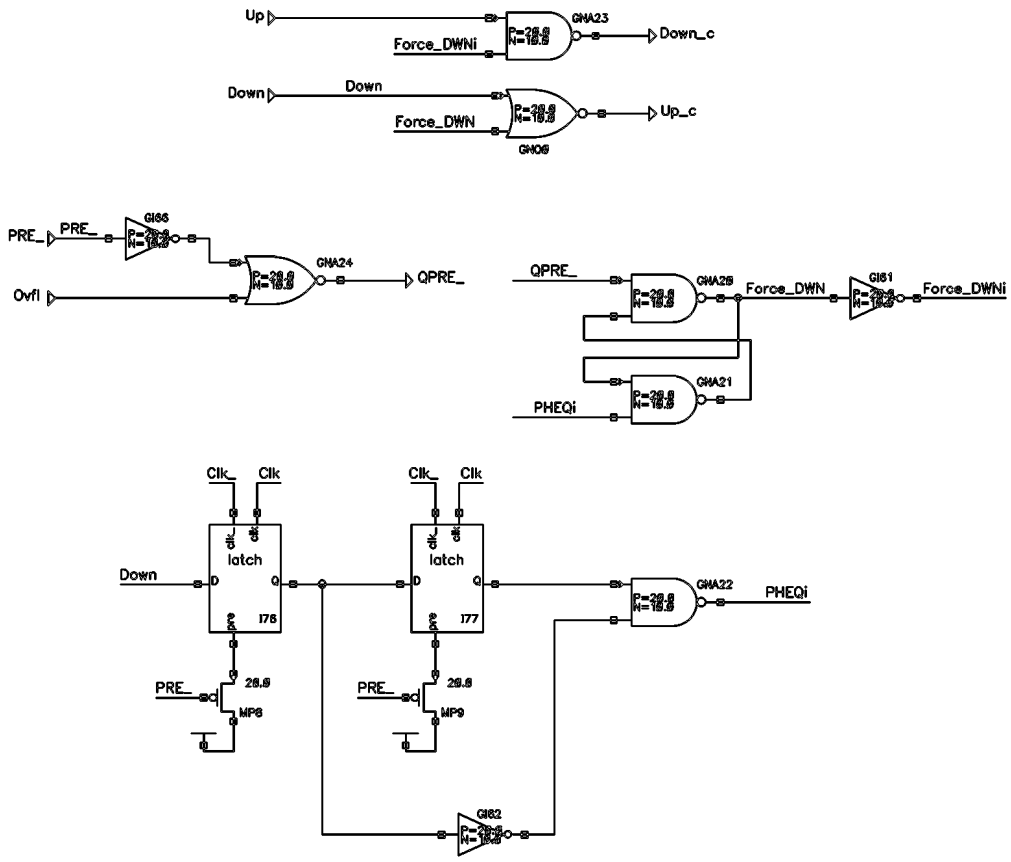


Figure 55 DLL initialization circuitry.

The approach used for this design is to reset the DLL and begin a new search for a lock point if the limits of the delay line delay are reached. This was chosen rather than simply staying ‘stuck’ at an extreme (freezing the count at either end) for two reasons: first, there exists a possibility that the DLL can lock on another harmonic of the reference signal (different multiple of tCK). If this is the case, a ‘reset, search again’ approach lets the

system attempt to find a harmonic. Second, it will be clear when testing the device that the DLL is not finding a lock point by examining the system output signal. If the system were just frozen at the ends, it might appear that the DLL is actually locked instead of frozen.

The Ovfl or the PRE_ signals are allowed to reset the system. The global preset, QPRE_, sets a SR flip-flop, which places the system in a 'force down' condition (Force_DWN). In this mode, the system will force the counter to count down, regardless of the direction the arbiter is trying to move the counter. Because it is understood that the arbiter will oscillate about the lock point once found, the detection of phase equal is accomplished by watching for the arbiter to try to count down in one cycle and then to count up in the next cycle. This indicates that the phase detect has reversed direction. This detection of a lock condition is done by registering the condition of the arbiter's DOWN output into a two-bit pipeline (Fig. 55). When a condition in the pipeline of 'Not DOWN, DOWN' is detected, the PHEQi signal is asserted, which resets the SR flip-flop, taking the system out of the initialization mode.

In reality, this initialization scheme is very simple. The initialization of other DLL designs may be incredibly complex, requiring much more control circuitry that must take care of many special conditions. This complexity is necessary for these designs because it is very important to have consistent DLL operation and lock points from device to device and from run to run. It must be stressed that initialization problems can be the cause of poor DLL operation.

3.10 FULL SYSTEM DESIGN

The entire DLL design is shown in Fig. 54 using a hierarchical schematic. All of the component blocks have been previously discussed. For this design, the differential clock used for the delay line is generated from a singled-ended clock, as shown in this schematic. A pass gate is used to match an inverter delay so that the clocks entering the delay line are exactly 180° out of phase.

The only inputs to the system are the reference clock signal and the asynchronous reset signal, PRE_. The quadrature outputs are buffered after exiting the differential-to-single-ended converters in the delay line block (Fig. 18).

As previously mentioned, the feedback delay normally used in DLL design is not used here. This will allow the loop delay to be contained almost entirely in the delay line, making the simple generation of quadrature outputs possible. Also, for this design, there is no input and output buffer delay, making it unnecessary to model the delay in the feedback path.

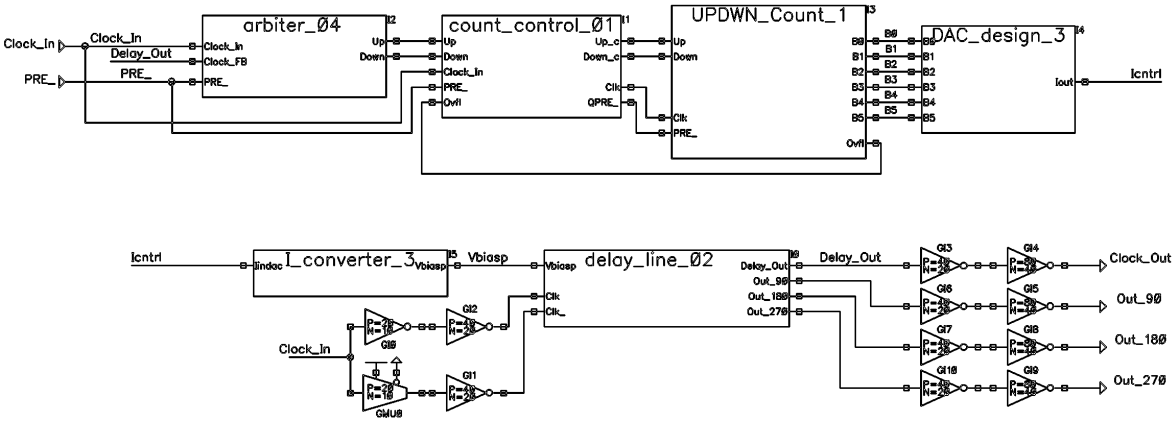


Figure 56 Full DLL design, presented in hierarchical blocks.

CHAPTER 4 – DLL PERFORMANCE

4.1 OPERATION

As discussed in the previous section, the DLL initializes such that the delay is set at the minimum, then searches for a lock point by adding delay in the delay line. This ensures that the lock point of optimum jitter performance is found. Once the lock point is found, the arbiter will cause the system to oscillate by 1 LSB of DAC control current. Fig. 57 shows the typical initialization sequence.

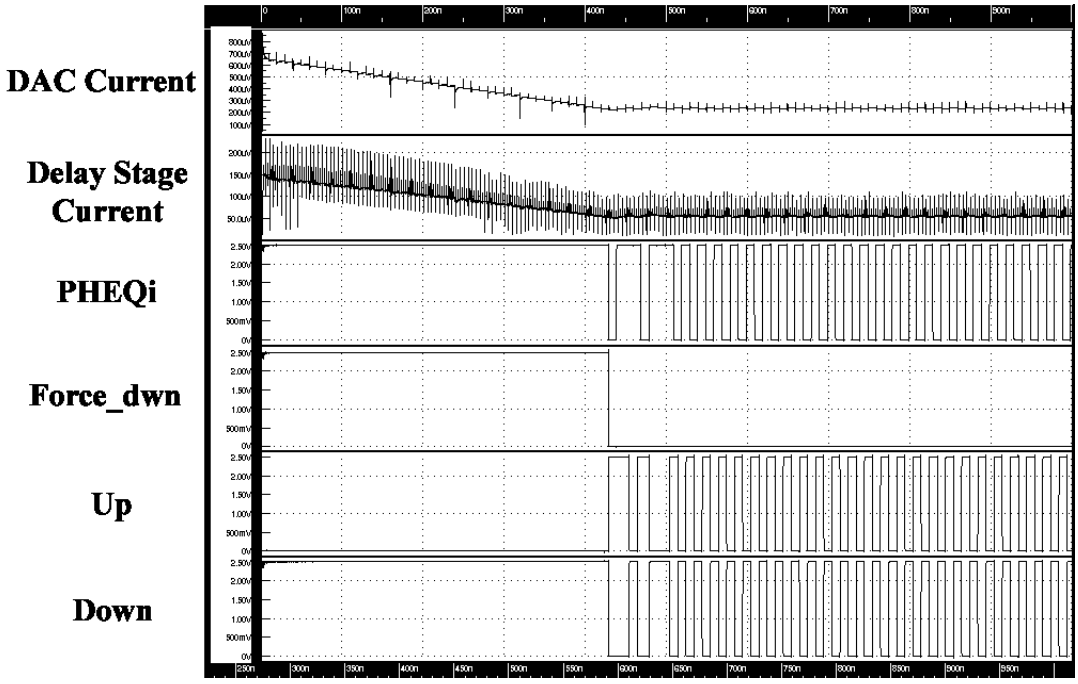


Figure 57 DLL initialization waveforms.

Notice that the total DAC current begins at the maximum current level, then directly approaches the lock point in a linear fashion. This figure clearly illustrates that the DLL is a first-order system; there are no second-order effects, such as overshoot, undershoot, or system oscillation. The current sources for the individual delay stages track the DAC

output current, with a ~5:1 reduction. This is expected, and is due to the reduction across the current mirror. The PHEQi signal is not asserted until the arbiter moves from a count down to a count up condition. At that point, the Force_DWN signal is de-asserted, signaling that a lock point has been found. The DLL is then allowed to track the variations in the input reference clock as Vdd and temperature change. It can be seen in Fig. 57 that once the lock point is acquired, the system begins to oscillate about the lock point as the arbiter moves the counter up and down.

The initialization sequence was simulated using a divide-by-two in the control circuitry. This reduces the arbiter-induced oscillations by allowing the phase detector to see

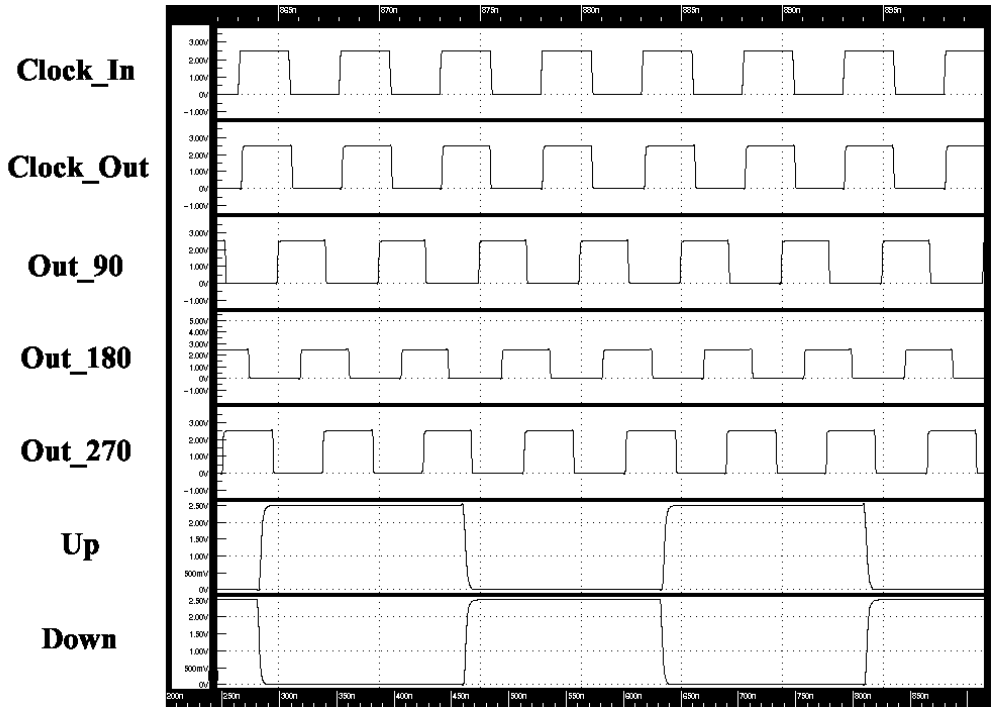


Figure 58 DLL waveforms after lock is achieved.

the results of a count command before issuing an additional command (there is a one-clock period delay around this loop when the system is locked). Fig. 58 shows that the DLL

output clock tracks the reference clock in a locked state. Note that the up and down control signals alternate every two clock cycles. The quadrature waveforms are also shown. Close examination reveals that the quadrature is not perfect, due to the fact that the entire loop delay is not contained in the delay line alone.

4.2 PERFORMANCE CHARACTERISTICS

The previous operational waveforms show that the DLL is functioning correctly, but it is important to quantify the operation. It is understood that the DAC will move into a region of poor performance as the reference clock period increases due to the 1/x relationship between control current and delay. This means that the DLL jitter performance will degrade as the reference frequency decreases.

The original design goal for this DLL is to be capable of locking on frequencies from 100MHz to 200MHz (10ns to 5ns period). Also, an important feature of the DLL is the ability to maintain lock and minimum jitter with variations in the power supply. Of course, the output jitter will also include any jitter on the input reference clock.

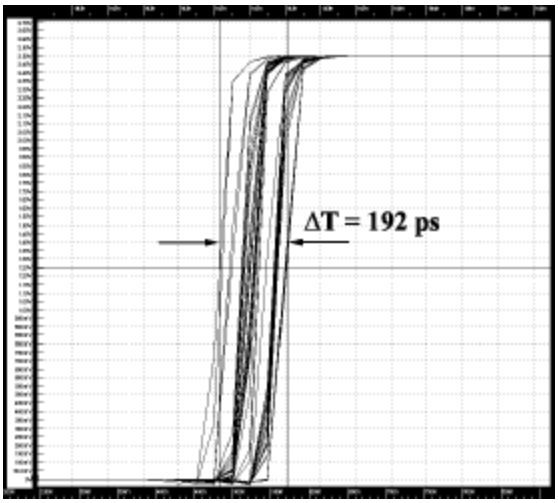


Figure 59 Simulated jitter with input tCK = 5ns, quiet power supply.

System jitter was characterized in simulation by initializing the DLL and allowing for a long period of simulation time after lock is reached. The clock output waveforms are extracted from the point after which lock is declared, and each

clock period is overlaid on top of the previous. This creates a waveform similar to that of an oscilloscope triggering on each period. Fig. 59 shows the simulated jitter using a 5ns input reference clock, nominal transistor models, room temperature, and Vdd = 2.5V. The jitter shown is due to the 1 LSB oscillation created by the arbiter. A peak-to-peak jitter of 192ps is measured.

To determine the DLL's ability to maintain minimal jitter with noise on the power supply, the same experiment is performed with an AC noise superimposed on Vdd. Fig. 60 shows the jitter results using a 10MHz sinusoid. The AC amplitude is ± 150mV (300mV peak-to-peak). System jitter in this case is measured to be 469ps. The increase in jitter is due to the variation in the delay per element as the current supply changes. The phase detector properly detects these variations and reacts accordingly, but it cannot do so instantaneously.

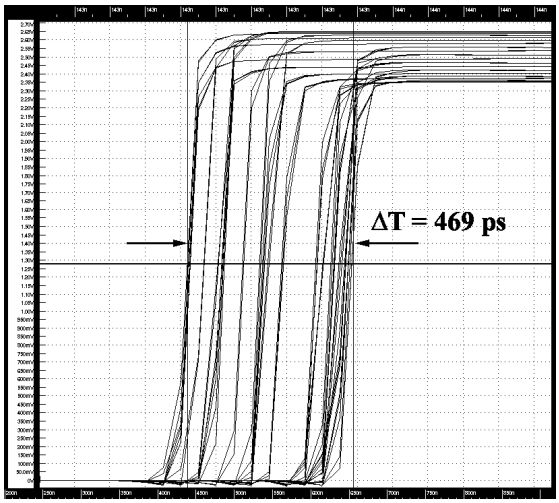
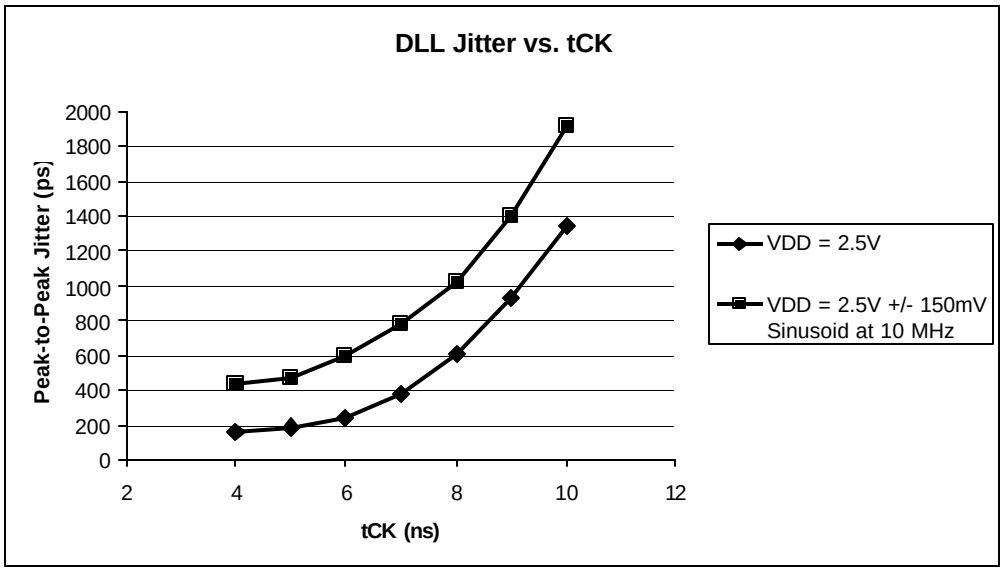


Figure 60 Simulated jitter with input tCK = 5ns, AC noise on power supply is +/-150mV at 10MHz.

tCK (ns)	Jitter (Quiet Supply)	Jitter (AC on Supply)
4	158	432
5	192	469
6	245	588
7	375	783
8	608	1024
9	932	1403
10	1340	1915

Table 3 DLL jitter performance.

These jitter calculations are especially important as the reference clock frequency decreases. Many simulations were run to generate the jitter results shown in Table 3. The AC noise added to VDD is still ± 150mV. This data was collected using nominal process, voltage, and temperature models. The graph of the data shows that the AC noise tends to create a static addition in the system jitter (Graph 7). Notice that the jitter performance



Graph 7 DLL jitter performance vs. tCK.

degrades as expected. The jitter is in an acceptable range of less than 500ps when the reference clock period is 5ns (200MHz). For a reference period of 10ns, the peak-to-peak jitter jumps to a worst-case value of almost 2ns! This would be unacceptable for many DLL usage situations, for example, in a DDR DRAM design, where output data is available for only half the clock period. Once again, this poor performance is due to the non-linear delay vs. current control relationship (see Graph 6). A possible solution to the jitter performance at high values of tCK is discussed in the next chapter.

The DLL's ability to lock on the frequency range of interest was characterized across all corners of PVT (Table 4). The slow corner was run at Vdd = 2.2V, slow transistor models, and temperature = 80° C. Typical simulations were run at Vdd = 2.5V, typical transistor models, and temperature = 25° C. The fast corner was run at Vdd = 2.8 V, fast models, and temperature = 0° C. This DLL design is capable of locking on clock signals with periods of 4ns to 10ns (100MHz to 250 MHz) across PVT. As was shown in Table 3, the jitter becomes larger towards tCK = 10ns, so this DLL will have other practical limitations as to the input frequency. The positive aspect here is that for the fast frequencies, where system clock synchronization becomes more critical, the DLL has the tightest jitter and highest immunity to power supply variation.

PVT Corner	tCK (min)	tCK (max)
Slow	4.0 ns	12 ns
Typical	3.25 ns	11 ns
Fast	2.75 ns	10 ns

Table 4 DLL lock range across PVT.

As with all designs, the power consumption characteristics are important. The current requirements for this circuit are shown in Table 5. Note that because the DAC current is the largest contributor to the power consumption, the demand is greatest at initialization, when the DAC is set to the maximum possible output (fastest delay in the delay line). This level does not change for varying input clock periods. However, after lock is achieved, the current, due to device switching, tracks the input clock frequency.

	Slow Corner		Typical		Fast Corner	
tCK	Initialization	Locked	Initialization	Locked	Initialization	Locked
5 ns	3.2 mA	2.3 mA	3.7 mA	1.8 mA	4.2 mA	1.7 mA
7.5 ns	3.1 mA	1.4 mA	3.7 mA	1.0 mA	4.1 mA	1.2 mA
10 ns	3.1 mA	0.8 mA	3.7 mA	0.7 mA	4.3 mA	0.9 mA

Table 5 Power supply requirements across PVT and tCK.

CHAPTER 5 – CONCLUSIONS

The design presented and characterized in this thesis has good jitter performance for the fastest reference clock frequencies and has good power-supply noise rejection. The most significant limitation of this design is the fact that as the control current for the delay line is reduced, the smallest 'step' in delay increases. This has a direct influence on the system jitter and its ability to maintain lock with minimal phase error. There are several possible solutions that would improve the DLL performance for longer clock periods.

To ensure that the delay line steps remain small, even when the elements are heavily starved, the LSB of the controlling DAC could be reduced by increasing the number of bits in the converter. Even if linearity is not maintained at exactly ± ½ LSB, as long as the converter is monotonic, the DAC would still be appropriate for this application, and the jitter on the output would be reduced.

Instead of increasing the resolution of the DAC, another solution would be to design a nonlinear DAC that maintains the delay vs. current control curves linear. In effect, the DAC would cancel out the non-linear nature of the delay vs. control. With this solution, the jitter performance would be constant, regardless of the input frequency.

Another method would be to change the design into a hybrid DLL, similar to a dual-loop approach, but with only one phase detector. The counter output could be used to switch in a series of larger, coarse delays (analog or digital) as needed, ensuring that the analog differential delay line is always operating in the region of good performance (referring back to Graph 6). This is the region where the delay per unit change of control

current is finest. This architecture (shown in Fig. 61) would actually be a relatively simple addition, requiring a minimal amount of extra layout area.

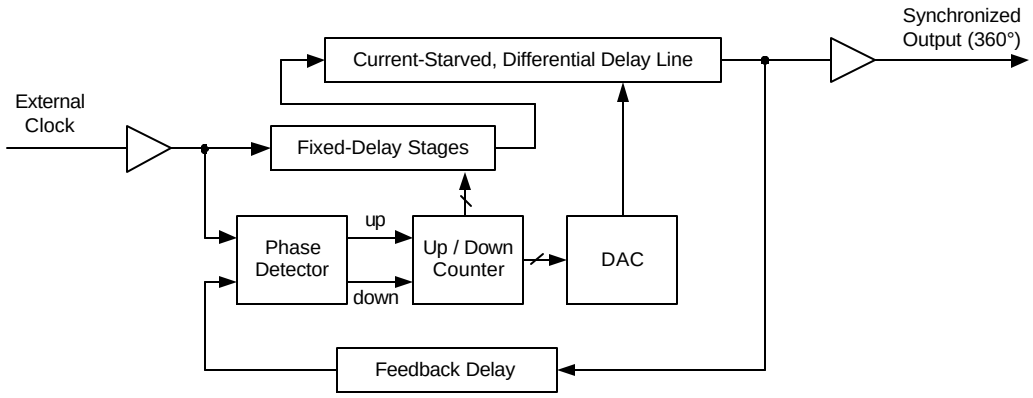


Figure 61 Use of constant delays to increase the useful frequency range.

As a rule of thumb, to achieve the best jitter performance and smallest phase error, the delay line must have as fine of delay steps as possible, and the phase detector must be able to differentiate phase differences to the resolution of the delay line.

Although this DLL could be improved upon in many other ways, the basic components and design approach can remain intact, taking advantage of a delay line that has continuous delay variation (rather than quantized steps), and good power-supply rejection.

BIBLIOGRAPHY

[1] H. Yoon, G.-W. Cha, C. Yoo, N.-J. Kim, K.-Y. Kim, C. H. Lee, K.-N. Lim, K. Lee, J.-Y. Jeon, T. S. Jung, H. Jeong, T.-Y. Chung, K. Kim, and S. I. Cho, "A 2.5-V, 333-Mb/s/pin, 1-Gbit, double-data-rate synchronous DRAM," *IEEE J. Solid-State Circuits*, vol. 34, pp. 1589-1599, Nov. 1999.

[2] T. H. Lee, K. S. Donnelly, J. T. C. Ho, J. Zerbe, M. G. Johnson, and T. Ishikawa, "A 2.5 V CMOS delay-locked loop for an 18 Mbit, 500 Megabyte/s DRAM," *IEEE J. Solid-State Circuits*, vol. 29, pp. 1491-1496, Dec. 1994.

[3] T. H. Lee and J. F. Bulzacchelli, "A 155-MHz clock recovery delay- and phase-locked loop," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1736-1746, Dec. 1992.

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

[5] S. Tanoi, T. Tanabe, K. Takahashi, S. Miyamoto, and M. Uesugi, "A 250-622 MHz deskew and jitter-suppressed clock buffer using two-loop architecture," *IEEE J. Solid-State Circuits*, vol. 31, pp. 487-493, Apr. 1996.

[6] Y. Moon, J. Choi, K. Lee, D.-K. Jeong, and M.-K. Kim, "An all-analog multiphase delay-locked loop using a replica delay line for wide-range operation and low-jitter performance," *IEEE J. Solid-State Circuits*, vol. 35, pp. 377-384, Mar. 2000.

[7] B. W. Garlepp, K. S. Donnelly, J. Kim, P. S. Chau, J. L. Zerbe, C. Huang, C. V. Tran, C. L. Portmann, D. Stark, Y.-F. Chan, T. H. Lee, and M. A. Horowitz, "A portable digital DLL for high-speed CMOS interface circuits," *IEEE J. Solid-State Circuits*, vol. 34, pp. 632-644, May 1999.

[8] G.-K. Dehng, J.-M. Hsu, C.-Y. Yang, and S.-I. Liu, "Clock-deskew buffer using a SAR-controlled delay-locked loop," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1128-1136, Aug. 2000.

[9] Y. Okajima, M. Taguchi, M. Yanagawa, K. Nishimura, and O. Hamada, "Digital delay-locked loop and design technique for high-speed synchronous interface," *IEICE Trans. Electron*, vol. E79-C, pp. 798-807, Jun. 1996.

[10] B. Keeth, R. J. Baker, *DRAM Circuit Design: A Tutorial*, IEEE Press, 2001. ISBN 0-7803-6014-1

[11] F. Lin, J. Miller, A. Schoenfeld, M. Ma, and R. J. Baker, "A register-controlled symmetrical DLL for double-data-rate DRAM," *IEEE J. Solid-State Circuits*, vol. 34, pp. 565-568, Apr. 1999.

[12] T. Saeki, Y. Nakaoka, M. Fujita, A. Tanaka, K. Nagata, K. Sakakibara, T. Matano, Y. Hoshino, K. Miyano, S. Isa, S. Nakazawa, E. Kakehashi, J. M. Drynan, M. Komuro, T. Fukase, H. Iwasaki, M. Takenaka, J. Sekine, M. Igeta, N. Nakanishi, T. Itani, K. Yoshida, H. Yoshino, S. Hashimoto, T. Yoshii, M. Ichinose, T. Imura, M. Uziie, S. Kikuchi, K. Koyama, Y. Fukuzo, and T. Okuda, "A 2.5-ns clock access, 250-MHz, 256-Mb SDRAM with synchronous mirror delay," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1656-1665, Nov. 1996.

[13] D. Shim, D.-Y. Lee, S. Jung, C.-H. Kim, and W. Kim, "An analog synchronous mirror delay for high-speed DRAM application," *IEEE J. Solid-State Circuits*, vol. 34, pp. 484-493, Apr. 1999.

[14] I. Youji, A. Masakazu, and N. Hiromasa, inventors; Hitachi Ltd., assignee. 1999 Sep. 21. Signal Generator with Synchronous Mirror Delay Circuit. U.S. Patent 5,955,905.

[15] S. Sidiropoulos, and M. A. Horowitz, "A semidigital dual delay-locked loop," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1683-1692.

[16] M. Mota and J. Christiansen, "A high-resolution time interpolator based on a delay locked loop and an RC delay line," *IEEE J. Solid-State Circuits*, vol. 34, pp. 1360-1366, Oct. 1999.

[17] M. Rau, T. Oberst, R. Lares, A. Rothermel, R. Schweer, and N. Menoux, "Clock/data recovery PLL using half-frequency clock," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1156-1159, Jul. 1997.

[18] J.-Y. Park and J.-K. Kang, "A 1.0 Gbps CMOS oversampling data recovery circuit with fine delay generation method," *IEICE Trans. Fundamentals*, vol. E83-A, pp. 1100-1105, Jun. 2000.

[19] R. J. Baker, H. W. Li, and D. E. Boyce, *CMOS: Circuit Design, Layout, and Simulation*, IEEE Press, 1998. ISBN 0-7803-3416-7

[20] M. Combes, K. Dioury, and A. Greiner, "A portable clock multiplier generator using digital CMOS standard cells," *IEEE J. Solid-State Circuits*, vol. 31, pp. 958-965, Jul. 1996.

[21] A. Efendovich, Y. Afek, C. Sella, and Z. Bikowsky, "Multifrequency zero-jitter delay-locked loop," *IEEE J. Solid-State Circuits*, vol. 29, pp. 67-70, Jan. 1994.

[22] H. O. Johansson, "A simple precharged CMOS phase frequency detector," *IEEE J. Solid-State Circuits*, vol. 33, pp. 295-299, Feb. 1998.

APPENDIX – ADDITIONAL SCHEMATICS

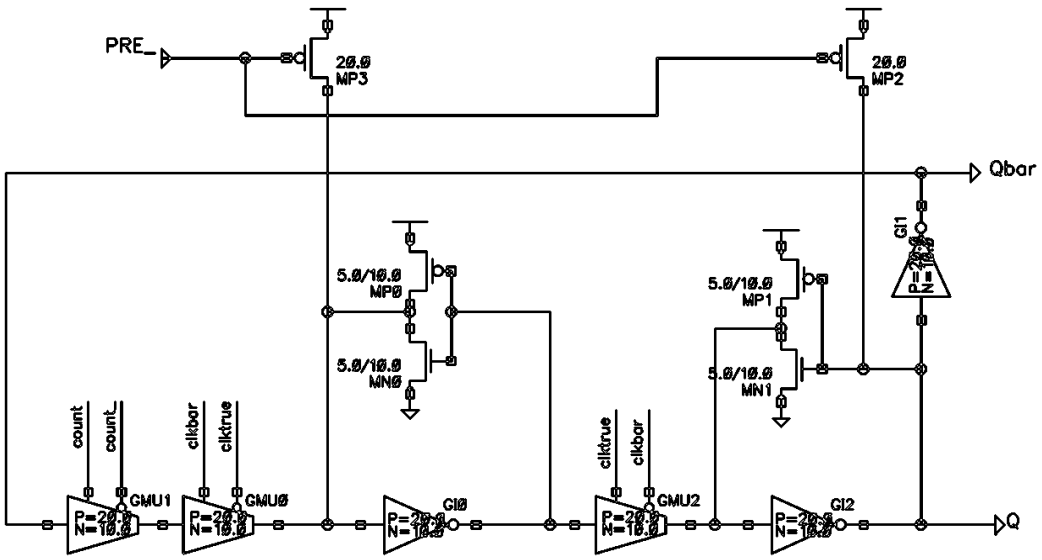


Figure 62 Schematic of toggle flip-flop used in counter.

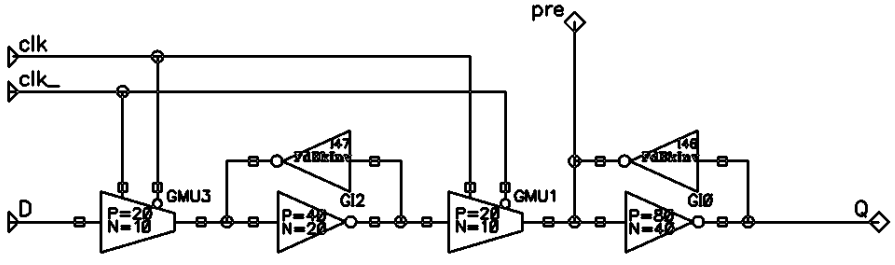


Figure 63 Schematic of regular flip-flop used in counter.