DESIGN OF AN INTEGRATED HALF-CYCLE DELAY LINE DUTY CYCLE CORRECTOR DELAY-LOCKED LOOP

by

Eric A. Becker

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical Engineering

Boise State University

April 2008

The thesis presented by Eric A. Becker entitled DESIGN OF AN INTEGRATED HALF-CYCLE DELAY LINE DUTY CYCLE CORRECTOR DELAY-LOCKED LOOP is hereby approved:

R. Jacob Baker Advisor	Date
Jim Browning Committee Member	Date
Scott Smith Committee Member	Date
John R. (Jack) Pelton Dean of the Graduate College	Date

ABSTRACT

High-speed synchronous systems require tightly controlled clock timing allowances for high performance operation. A Delay-Locked Loop (DLL) is a commonly used circuit to de-skew any variations due to process, voltage, or temperature (PVT). While a DLL will effectively align an input reference clock to an outgoing data clock, the DLL will not adjust the reference duty cycle if it is non-ideal. For this purpose a Duty Cycle Corrector (DCC) can be used in tandem with the DLL. Through the combined use of a DLL and a DCC, a high-speed system can be provided with a clock that has been both de-skewed across PVT and has a good duty cycle.

Three DCC designs are compared: the Half-Cycle Delay Line (HCDL), Open Loop (OL), and the Integrated Half-Cycle Delay Line (IHCDL). The HCDL DCC features a stable closed loop duty cycle detection and wide duty cycle range at the cost of larger layout area. The OL DCC features less stable open loop detection with the benefit of minimal forward path delay again at the cost of larger layout area. The IHCDL DCC is a hybrid design that, through the use of only a single delay line for both 0º and 180º phase generation, provides the advantages of the HCDL DCC with a substantial improvement to the required layout area.

The design of a generic DLL and the IHCDL DCC are detailed in this thesis. The performance of the IHCDL is verified through simulation. Across a range of 3 – 10 ns the IHCDL DCC corrects duty cycle to within +/- 5% of 50%. The additional lock time and power consumption of the IHCDL DCC (compared to the DLL only) are evaluated. Finally, the jitter induced by the IHCDL DCC across PVT and voltage supply variations is evaluated. Suggestions are made for the improvement of duty cycle correction.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Jake Baker for his instruction, guidance, and motivation through my time as a graduate student at Boise State University. The skills and knowledge I have gained from him have been an immense help in my development both scholastically and professionally.

I would also like to thank Micron Technology Inc. not only for funding my post-graduation education, but also for providing me with a job where I get to work on DLL's and DCC's on a daily basis.

Thanks also go to Eric Booth, Tyler Gomm, and Brandon Roth for their expert DLL knowledge and tolerance for my interminable questioning. Brandon, especial thanks to you for nurturing me from a fledging college student into a (semi) productive engineer!

Most of all I would like to extend my deepest thanks to my wife, Elena, whose love, patience, and encouragement have made all the difference in all aspects of my life. I would not be the man I am today without your support. Last but certainly not least, thanks to my little man, Thomas, for getting by without me during the long nights of studying.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 – INTRODUCTION

1.1 Motivation

A Double-Data-Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) is an example of an application that uses a Delay-Locked Loop (DLL) to maximize the data valid window [1]. Figure 1.1 shows the timing diagram for a DDR SDRAM including the external clock (XCLK), the output data strobe (DQS), and output data (DQ). The time, $t_{DQV,}$ is the time associated with data valid window. The times, t_{DQSCK} and $t_{AC,}$ are the clock to DQS and DQ skews, respectively.

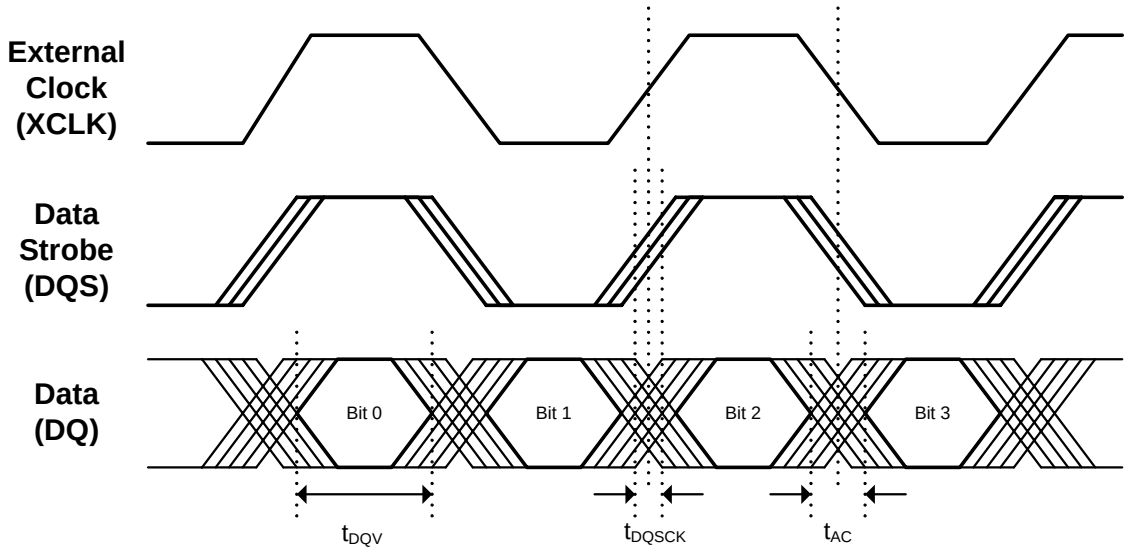

Figure 1.1 Data Timing Chart for DDR DRAM

A DLL is a circuit commonly used in synchronous circuits to align outgoing data with an external clock signal. As circuit speeds increase with shrinking device dimensions, the clock frequencies increase, and the effects of clock skew and jitter on a system becomes an increasingly larger percentage of t_{DQV}. When the data valid window

shrinks, the integrity of the system is detrimentally affected and high performance suffers [2].

A DLL aligns DQS and the DQ's to an external clock provided by the memory controller. Through the dynamic use of a variable delay line (VDL), the DLL effectively accommodates variations in process, voltage, and temperature (PVT) by adding or removing delay between XCLK and DQS. If a fixed amount of delay were used—instead of a DLL—to align the incoming clock and output data, then variations in PVT would significantly increase clock skew (t_{DQSCK}). This increase effectively shrinks the data valid window (t_{DQV}) and makes the system more subject to timing errors.

On a DDR SDRAM application, data is clocked out on both the rising and falling edges of clock. Consequently, the incoming clock duty cycle can also affect the data valid window of the second bit of data (bit 1). The clock high time is proportional to the bit 0 t_{DQV} and similarly the clock low time is proportional to the bit 1 t_{DQV}. A clock signal with an ideal 50% duty cycle has equivalent clock high and clock low periods. A poor duty cycle would be any clock signal with significant difference between the clock high and clock low times. For example, a 25% duty cycle clock would mean that the clock is only high for 25% of the clock period and low for the other 75% of the period. A memory controller that is providing an external clock with a poor duty cycle will automatically be affecting the data window negatively. For example, an input duty cycle of 30% on XCLK will reduce t_{DQV} for the bit 0. Conversely, a 70% input duty cycle would hurt t_{DQV} for bit 1. To ensure that the DLL outputs a clock with a 50% duty cycle regardless of the input duty cycle, a Duty Cycle Corrector (DCC) circuit is commonly used in tandem with a DLL [3], [4], [5].

In high frequency clock systems attaining a perfect duty cycle to all devices can become expensive as costly high-quality components are required to preserve duty cycle throughout the system. It is beneficial if the memory device can handle a non-ideal duty cycle and still output a 50% duty cycle. This capability allows the system to maintain good performance while keeping costs down [2].

In a Very Large Scale Integrated (VLSI) circuit design, it is advantageous to use a digital DLL design for its portability across process nodes. While analog DLL's generally provide better jitter performance and higher phase accuracy, digital DLL designs tend to have faster locking times, lower power dissipation, and less sensitivity to variations in PVT [3], [6]. For these reasons this thesis focuses exclusively on digital DLL and DCC designs.

1.2 DLL Behavior

Figure 1.2 shows a basic DLL block diagram. The DLL itself consists of a variable delay line (VDL), a phase detector, a delay shift control circuit, and replica buffers, which model the input and output buffers. The VDL is comprised of a string of delay elements that can either be increased or decreased in number. The phase detector is the decision circuit that determines whether delay elements should be added or subtracted in the VDL. The delay shift control circuit processes and filters the phase information and sends the proper shift signals to the VDL. The replica buffers model the delay through the input and output buffers. The replica buffers must accurately model the actual buffers for the DLL to precisely synchronize the external clock to the output synchronous clock across variations in PVT.

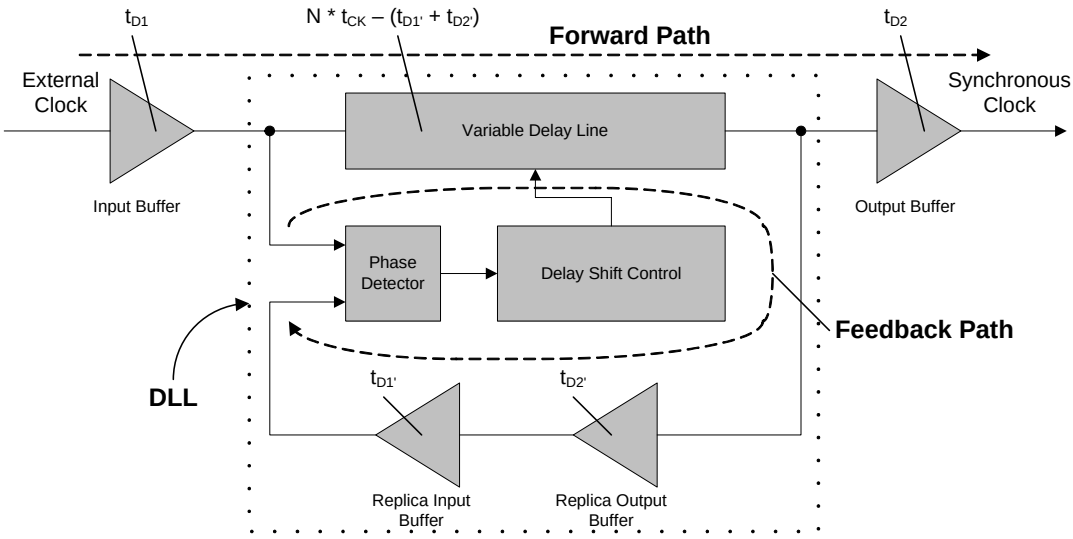

Figure 1.2 DLL Block Diagram

Figure 1.2 shows that the VDL will be adjusted to a value of $N * t_{CK} – (t_{D1'} + t_{D2'})$ where N is an integer number of clock cycles and t_{CK} is clock period. The values of t_{D1} and t_{D2} are the delays associated with the input and output buffers, respectively. Similarly, $t_{D1'}$ and $t_{D2'}$ are the delays associated with the replica (or model) input and output buffers. The delay in the forward path (t_{FP}) and feedback (t_{FBP}) path is shown by equations (1) and (2), respectively.

$$t_{FP} = t_{D1} + [N * t_{CK} - (t_{D1'} + t_{D2'})] + t_{D2} \quad (1)$$

$$t_{FBP} = [N * t_{CK} - (t_{D1'} + t_{D2'})] + t_{D1'} + t_{D2'} = N * t_{CK} \quad (2)$$

Figure 1.3 shows an example of how the DLL can be locked at different N values. Regardless of the value of N, the rising edges of the external clock and synchronous clock have the same phase. Essentially, N is a numerical representation of how many harmonics are present in the DLL feedback loop. When the DLL is locked at a point where N = 1, for example, there is only one cycle delay between the external clock and synchronized clock. When N = 2 there are two cycles delay between the external and synchronized clock and so on for N = 3, 4, 5, etc.

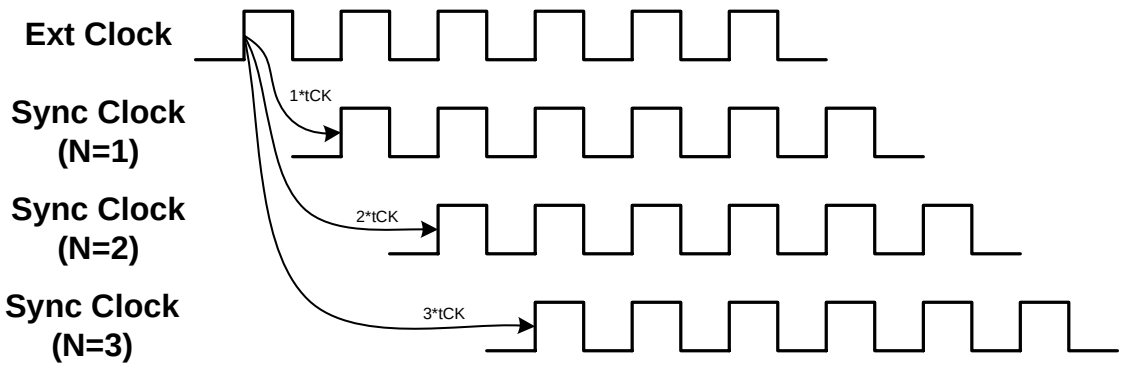

Figure 1.3 DLL with N Lock Points

It is important to note the difference between phase and delay. Delay simply refers to the amount of time it takes for a signal to propagate through a circuit or series of circuits. Generally, this value is t_{Delay}. Phase, ϕ, is an angular representation of t_{Delay} between two clocked signals of the same period (t_{CK}). Phase (in degrees) and delay are related by the following equation [3].

$$\phi = 360^{o} * t_{Delay} * (1 / t_{CK}) \quad (3)$$

When two signals are said to be 180 degrees (180^{o}) out of phase it is equivalent to saying that the later signal is delayed by $t_{CK} / 2$.

When the replica buffers model the actual buffers accurately t_{D1} equals $t_{D1'}$ and t_{D2} equals $t_{D2'}$. According to equation (1) the forward path delay is equal to N * t_{CK} which precisely matches the feedback path delay as shown by equation (2). Ideally, $t_{D1} = t_{D1'}$ and $t_{D2} = t_{D2'}$, but realistically the model buffers will not accurately model the actual buffers across PVT. This mismatch in the modeling of the buffers will result in clock skew between the external and synchronous clock. The best way to avoid this mismatch is accurate modeling and good layout practices.

Assuming that buffer modeling is only introducing negligible clock skew, the DLL will have a maximum phase error (in degrees) of

$$\phi_{error,\,max} = 360^{o} * t_d / t_{CK}, \quad (4)$$

where t_d is the minimum value of the DLL's VDL step and t_{CK} is the clock period. Equation (4) shows that the phase error increases as the clock period decreases [7]. So as clock speeds continue to increase so will the phase error. Figure 1.4 shows a visual representation of the phase error with respect to clock frequency ($1 / t_{CK}$).

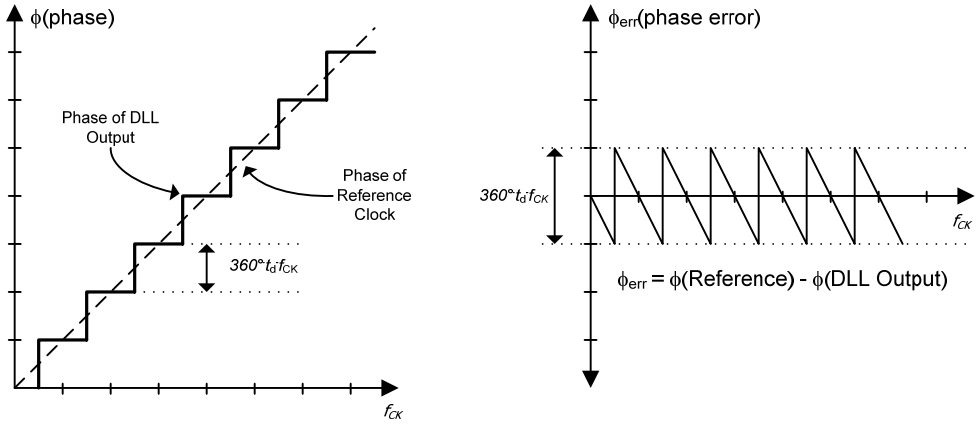

Figure 1.4 Phase Error in a Digital DLL

Because the DLL can only insert or remove a fixed amount of delay, there is a quantization error between the output and reference clock of the DLL. This error is not present in an analog DLL because a voltage controlled oscillator (VCO) controls t_d [7]. In a digital DLL, t_d is a discrete value because it is comprised of logic gates. This error creates the need for phase resolution improvement circuits in digital DLL's. Dual loop DLL's with fine delay elements solve this problem and can effectively reduce the DLL's minimum phase resolution [3].

Figure 1.5 shows a block diagram of a dual loop DLL that uses coarse and fine VDL's. The coarse VDL is used to find the initial coarse lock within a tolerance of the

minimum coarse delay element. Once the initial lock is found the control of the DLL is handed over to the fine delay loop, which has a much finer phase resolution. By using this approach the digital DLL can have a minimum phase step comparable to an analog DLL [3]. An effective method for implementing a fine delay loop is through the use of phase mixing/interpolating [3], [8].

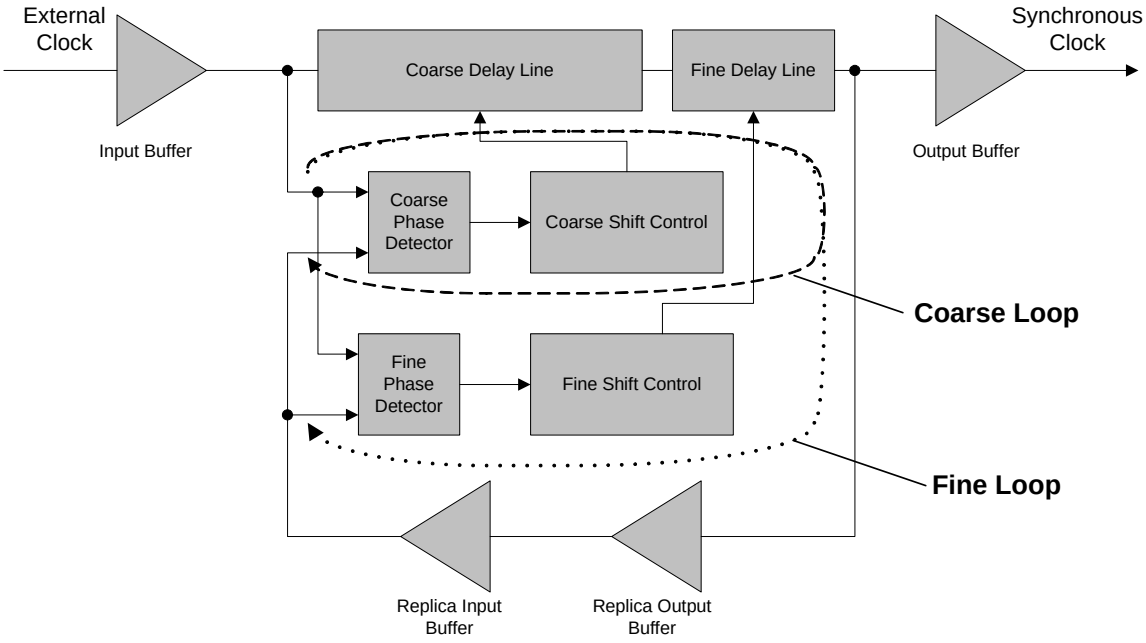

Figure 1.5 Dual Loop DLL

To reduce the noise sensitivity of the DLL, filtering of the phase information is essential to ensure that any shifts are intentional and not induced by voltage supply noise. This filtering can be done in the phase detector itself by dividing the detection rate, but the filter range can be limited in these cases [1]. Averaging filters utilizing a shift register have also been used to keep track of phase information [1]. While filtering does slow the response of the DLL phase tracking, it also helps minimize the output clock jitter by reducing the total amount of shifting.

1.3 DCC Behavior

Ideally, a DCC receives an input clock with an arbitrary duty cycle and outputs a clock signal with the same high and low periods, which by definition is a signal with 50% duty cycle. Figure 1.6 shows a basic block diagram of this black box behavior.

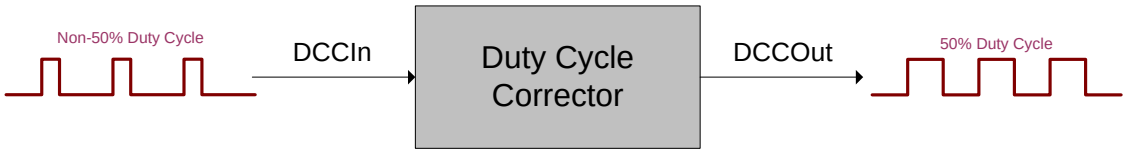

Figure 1.6 DCC Block Diagram

A DCC design can be a stand-alone integrated circuit (IC) in the sense that it is a portable circuit that can be added or removed simply by placing it in the forward path of the DLL's input or output. Stand-alone IC designs have the advantage of convenience during the design stages but have the significant disadvantage of adding another phase loop into the synchronization scheme. Not only will another phase loop potentially extend the lock time of the system, but the DCC phase loop could also make the DLL phase loop unstable [5]. Instability in either the DLL or DCC phase loop will decrease the output phase resolution and reduce the data valid window.

Opposite to a stand-alone design, an integrated DCC design will be more laborious in the design stages; however there are some other considerable benefits to using an integrated design. The major benefit is layout area savings. By not having to simply place a pre-existing DCC block into a layout, an integrated design can share attributes with the DLL to save layout space. Another benefit is the potential to eliminate the dual phase loops common with discrete DCC's. A DLL phase loop will be more stable if the DLL and DCC are both using the same loop for phase detection.

The method of duty cycle correction in a DCC is critical to an effective design and pivots around how the 50% correction is implemented. Determining where the 50% phase point lies can be challenging, but this determination is essential for the accuracy of the DCC. The DCC must perform two functions: define the 50% phase boundary and adjust the duty cycle based on this information.

DCC's can be implemented in either an open or closed loop configuration. A closed loop DCC will have feedback phase information to help determine a lock. This is good for the accuracy of the DCC but can have a destabilizing effect on the DLL, which is also in a closed loop configuration. For example, if the DCC makes an adjustment, it can take the DLL out of lock and force the DLL to reacquire a lock. Small adjustments in either the DLL or DCC can require additional time to settle out which can extend the lock time [5]. Open loop configurations will not affect the DLL phase loop as significantly, but these configurations do not have any feedback and, thus, may have a higher tendency for error accumulation during the duty cycle correction. In either an open or closed loop design, care must be taken to ensure that interaction between the DLL and DCC be kept to a minimum to minimize clock jitter.

A DCC adjusts the falling edge of the input clock signal until it is exactly halfway between the surrounding rising edges of the clock signal. Usually, the falling edge signal (180º Out in Figure 1.7) is separated from the rising edge signal (0º Out), and the two signals are multiplexed using a phase combiner. Figure 1.7 shows a simple block diagram of this operation. Notice that only the rising edges of 0º Out and 180º Out are responsible for creating a transition on the output of the phase combiner, which corresponds to the rising and falling edge of the output waveform, respectively.

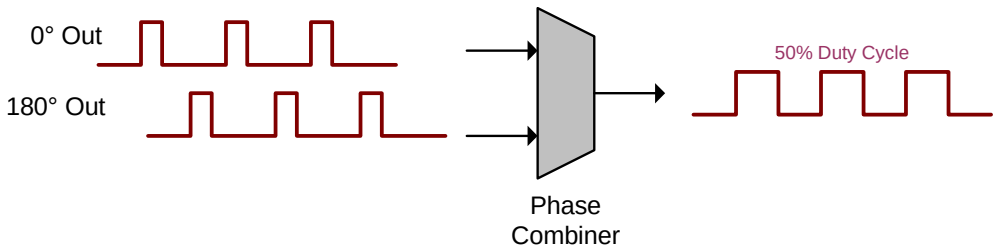

Figure 1.7 Phase Combine Diagram

While a functional DCC will correct any duty cycle problems, there are three main drawbacks to inserting a DCC into a synchronous system. The first is an increase in the forward path delay of the system. Additional forward path delay will increase the system wake up time and increase susceptibility to voltage supply noise-induced jitter [3]. Any DCC will add some amount of forward path delay whether from the phase combiner or DCC devices placed in the forward path. The second drawback for the addition of a DCC is an increase in power consumption. A DCC is an additional circuit that features a full clock frequency toggling VDL along with the associated control logic, all of which consume considerable amounts of power. The final drawback is layout area. As more gates are placed naturally more layout area is consumed.

The placement of the duty cycle corrector is another critical design consideration. While the DCC will be placed in series with the DLL, it can be placed either at the input or the output of the DLL. A number of problems exist if the DCC is placed at the output. The first problem is the assumption that the external clock signal has a duty cycle that is close enough to 50% to allow the clock to pass through all of the internal DLL logic and make it to the DCC [3]. Good design practice will help to ensure that signals with poor duty cycle will pass through the entire forward path. Another problem is that all of the duty cycle correction must be performed in one stage [3]. This requires that the DCC have a wide duty cycle correction range, which can be accommodated by choice in DCC

topology. Placing the DCC at the input of the DLL will provide a 50% clock signal to the DLL, but if the DLL degrades duty cycle then a non-ideal duty cycle will be sent to the output without correction. To avoid this problem, the DCC design considered in this thesis will place the DCC on the output of the DLL to take advantage of duty cycle correction as late as possible in the forward path.

CHAPTER 2 – DLL DESIGN

2.1 Process and Simulation Models

The process used for this DLL design is a 1.5 V, 0.08 μm process (a 0.0575 shrink factor) from Micron Technology, Inc. which has been specifically designed for DRAM production. The typical process parameters, such as oxide thickness, are proprietary and cannot be further disclosed. All simulations were performed using either NANOSIM or HSPICE. Schematic capture was done using Cadence's DFII software. Consistent with most deep-submicron designs, simulations must be relied upon rather than hand calculations for device evaluation.

2.2 Phase Splitter

DLL clock signals are frequently distributed, decoded, and buffered to numerous other timing critical circuits. To minimize clock skew it is essential to propagate these clock signals and their inverse phases in such a fashion that the rising and falling edges arrive at a given circuit simultaneously. A flip-flop is one example of a circuit that can benefit from having two clock phases (inverted and non-inverted) provided to it. When a flip-flop receives the two clock phases at different times a "dead phase" is created that increases the amount of data processing time in the latch [9]. For this reason, a phase splitter circuit was designed to provide identical propagation delays for both an inverting and non-inverting clock phase.

Figure 2.1 shows the schematic of the phase splitter. The phase splitter consists of an enable NAND gate, two inverters (gates 1 and 2) in the inverting path, and three

inverters (gates 3, 4, and 5) in the non-inverting path. Because the enable NAND gate inverts the input, an inverted clock must be provided to the circuit. The goal of the phase splitter is to satisfy the following equation:

$$t_1 + t_2 = t_3 + t_4 + t_5 \quad (5)$$

t_1, t_2, t_3, t_4, and t_5 correspond to the propagation delay of inverters 1, 2, 3, 4, and 5, respectively.

$$t_1 = t_3 + t_5 \quad (6)$$

$$t_2 = t_4 \quad (7)$$

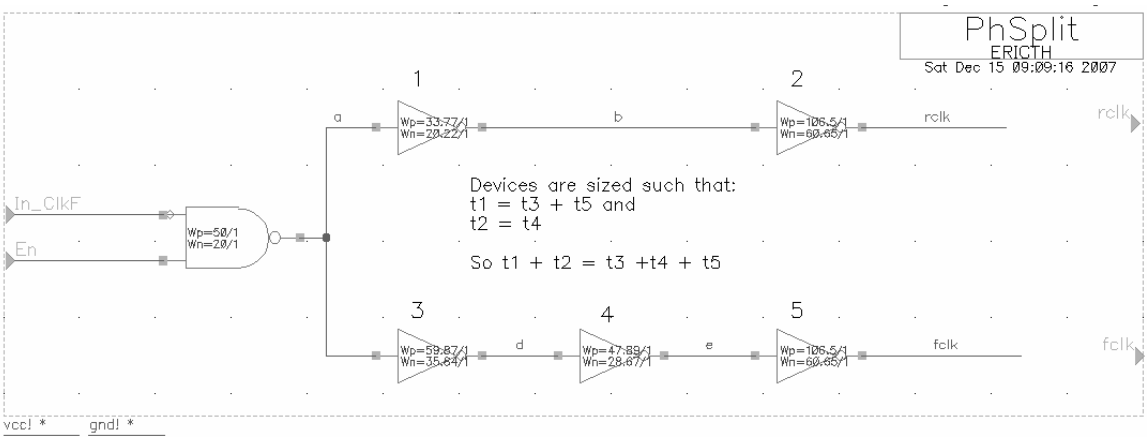

Figure 2.1 Enabled Phase Splitter Schematic

Furthermore, if the inverters are designed such that equations (6) and (7) are true then the delay through both paths will be identical [9].

In order to find device sizes that satisfy equations (5) through (7) a Monte Carlo simulation was run to determine the following parameters: *n* (the base nmos device width), β_{135} (the p-n ratio for inverters 1, 3, and 5), β_{24} (the p-n ratio for inverter 2 and 4), and *f* (the fan-out factor between inverter pairs 1 and 2 and 4 and 5). The resulting simulation output provided correct device sizes to guarantee that the phase splitter will generate identical propagation delays for both phases [9].

Figure 2.2 shows the simulation results when the optimized parameters were used. Notice that there is a 169 ps propagation delay for the rising edge output and a 165 ps delay for the falling edge output. Also note that the crossing point of rclk and fclk is approximately Vdd/2, suggesting that their transition times are equivalent in either direction.

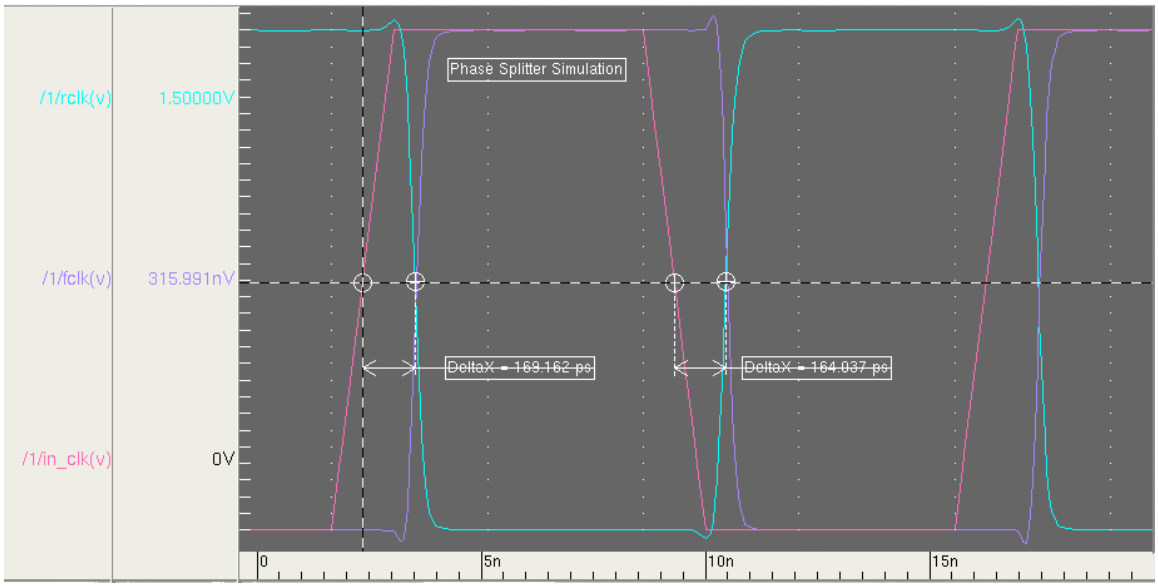

Figure 2.2 Enabled Phase Splitter Simulation

2.3 Delay Element

The VDL portion of the DLL is created with a string of inverting CMOS logic gates. Entry or exit points into this string of gates are called taps (or tap points). These delay line cells are tapped every other gate to ensure that each successive tap is not an inversion of an adjacent tap [3]. In early DLL designs simple inverters were used in the delay line. More recent designs have taken advantage of the NAND gates as the CMOS gate used in the delay line [1].

A NAND gate delay line has a number of beneficial properties. First, during power-up every delay stage can be set to a known digital level. This approach avoids any

ambiguity during the device startup-period. Second, NAND gates have a longer propagation delay than simple inverters, which allows for a longer delay line for an equivalent number of delay cells. Finally, the high-to-low or low-to-high propagation time may be skewed in a NAND gate but, because there are two consecutive NAND gates per delay cell, both edges of the clock signal will be delayed identically [1]. Consequently, skew will not be accumulated with additional delay cells because each delay cell has an identical delay to that of all the other delay cells.

Figure 2.3 shows a diagram of the delay line and shift register initially used on this DLL. Notice that the basic unit delay cell and delay stages are selected. A delay stage consists of a delay cell, a shift register bit, and an entry NAND gate. The input clock to the delay line, ClkIn, is provided to every entry NAND gate, but only the entry tap will allow ClkIn to enter the delay line. The entry point is determined by the shift register via the phase detect and shift control circuitry. Once ClkIn has propagated down the calculated length of the delay line it is output as ClkOut. Ideally, the amount of delay selected in the delay line will be such that the external and output data clocks are in phase [3].

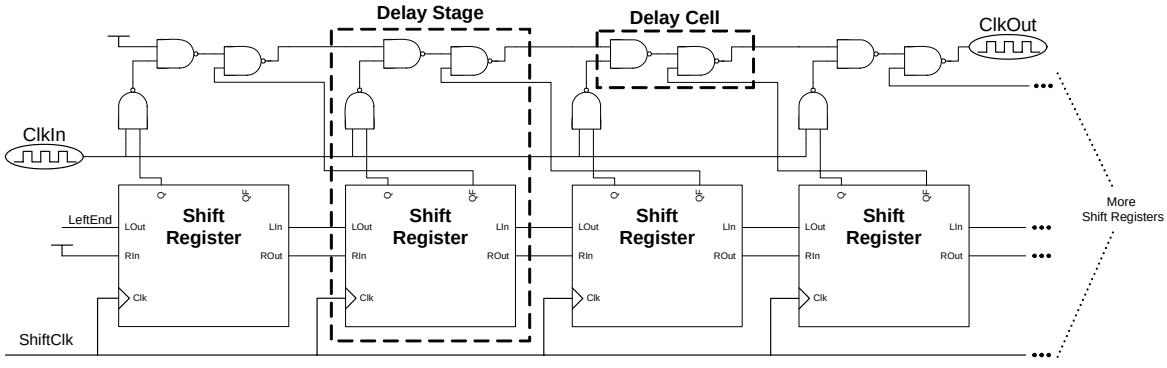

Figure 2.3 Delay Line and Shift Register Diagram

The main purpose of the shift register, as seen in Figure 2.3, is to keep track of the depth in the delay line. The shift register is also used to add or remove delay cells based on phase information that is decoded by the shift control circuitry. A shift register bit is essentially a flip-flop that either stores a '0' or a '1' based on the surrounding bits. The outputs of the shift register, Q and QF, are used to select the entry tap.

Figure 2.4 shows how the value stored in the shift register selects the entry tap point. A '0' stored in a shift register bit will prevent that entry-NAND gate from passing ClkIn and will allow the previous delay cell to pass the clock coming down the delay line. When a '1' is stored, the entry-NAND allows ClkIn to enter the delay line and the previous delay cell is prevented from passing the clock downstream. The entry delay stage is the one that has a '1' stored adjacent to a downstream '0'. The '1' to '0' transition in the shift register sets a unique entry point to the delay line which can be adjusted by the shift control logic to find the proper lock point [1].

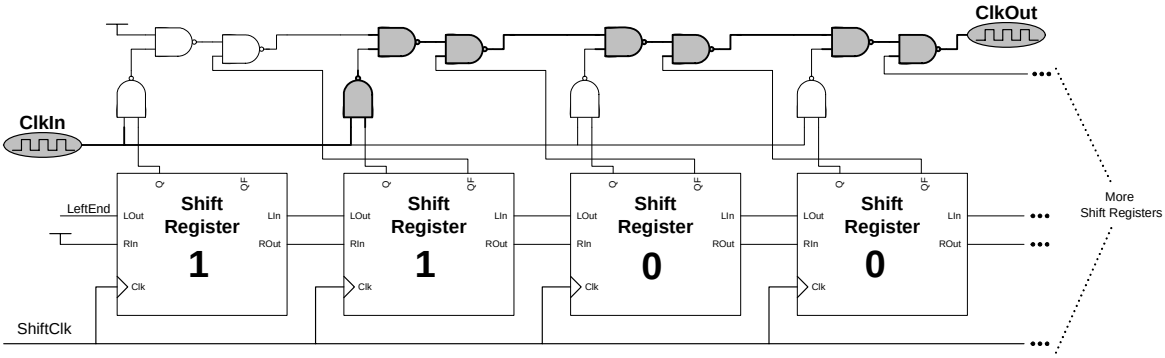

Figure 2.4 Clock Entry Point and Shift Register Value Diagram

Figure 2.5 shows a simplified delay line and shift register diagram. This diagram will be used to describe future topologies, so it is useful to introduce here. The numbers represent the value stored by each shift register bit in the VDL. Each number has a single delay cell associated with it. The vertical line on the right side indicates where the buffer

delay cells start. The buffer delay line marks the minimum depth that the DLL can lock to allow for dynamic variations in voltage and clock period. Again the '1' to '0' transition marks the lock point, which is also indicated by a red 'X'.

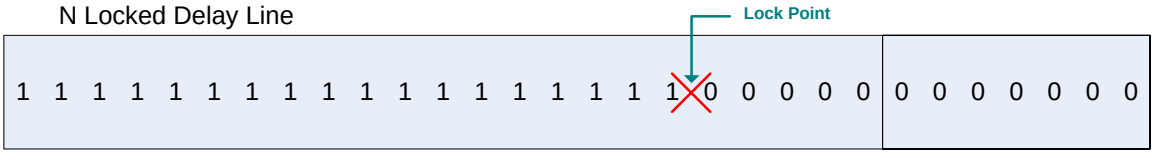

Figure 2.5 Simplified Delay Line and Shift Register Diagram

The schematic of the shift register used in this DLL is shown in Figure 2.6. The shift register is an edge-triggered master-slave flip-flop that is clocked by the coarse shift clock (CSclk). There are three inputs to the shift register: one input to pass information from the upstream bit upon a shift right command (QL); another input to pass information from the downstream bit upon a shift left command (QR); and a third input (MI) to set the shift register to the proper value upon the DLL initialization. The QF from the downstream bit is also sent into the slave latch so that if a '1' is present downstream it will be propagated to all upstream bits.

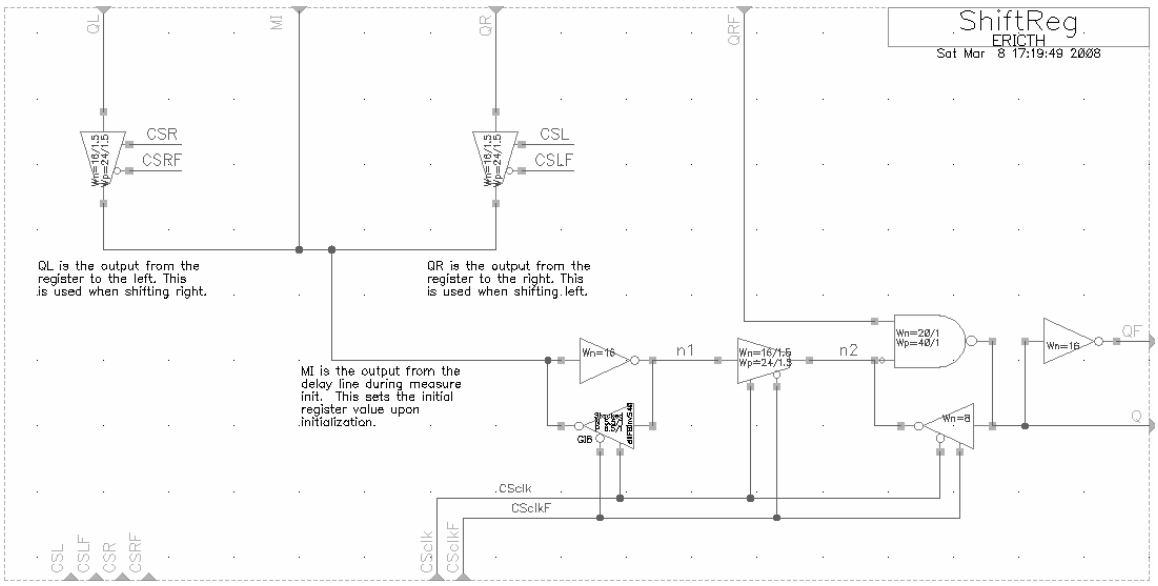

Figure 2.6 Shift Register Schematic

2.4 Phase Detector

The phase detector is the decision circuit for the DLL. The phase detector is used to determine if the reference (external) clock is in phase with the feedback (output) clock [10]. The phase detector in this design can determine whether the feedback is too early, too late, or in phase with the reference clock. Because there is a certain amount of "dead phase," called hysteresis, inserted between the delayed version of the feedback signal, the phase detector is able to determine whether the DLL is in a locked, or phase equal, state.

Figure 2.7 shows the schematic of the coarse phase detector. Notice that the reference and feedback clocks are phase split. The reference clock (RefPD) after being split becomes RefClk, which is used to clock to the comparison flip-flops. The feedback clock (FbPD) after being split becomes DllFbD. DllFbD is then delayed through two coarse delay cells to create DllFbDD. DllFbD and DllFbDD are the inputs to the comparison flip-flops. The output of the comparison flip-flops, after being phase split again, creates the Ph1 and Ph2 signals, which represent the state of the phase during the

rising edge of RefClk. Ph1 and Ph2 are decoded to activate the SL, SR, PhEq, Ph180, and NotPhEq signals, which relate the phase information to the shift control logic. The PDHoldF signal exists to disable the coarse phase detector circuit upon a DLL reset or a coarse phase disable condition.

The feedback clock will have delay added to it in discrete steps equal to the unit delay cell until it is locked with the reference clock. Because the delay line consists of a string of unit delay cells, when locked the feedback clock should fall within range of one unit delay cell. Since the hysteresis is twice this range, the phase detector has the capability to determine when the reference and feedback clocks are in phase [11].

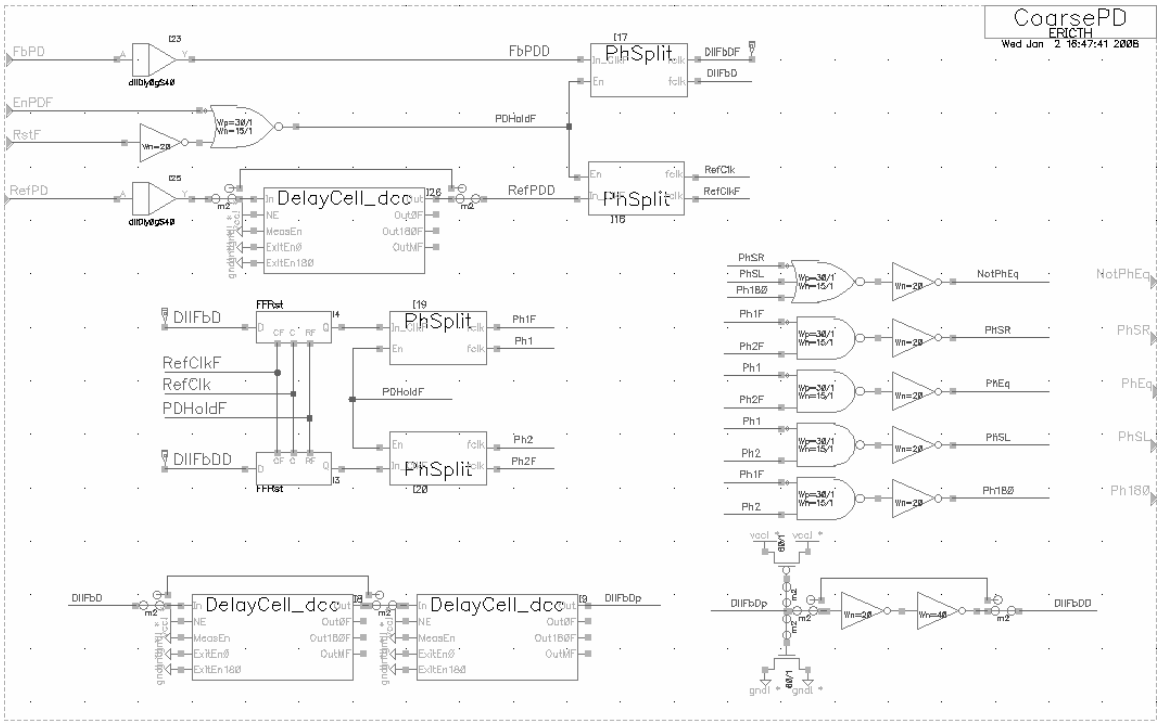

Figure 2.7 Schematic of Coarse Phase Detector

Figure 2.8 shows the phase timing diagram. RefClk clocks the flip-flops whose inputs are DllFbD and DllFbDD. The phase relationship among these three signals determines what command the phase detector outputs to the shift control logic. When

RefClk transitions between the two DllFb signals, then the DLL is locked. The phase detector also determines which direction the DLL should shift if it is not locked. Using Figure 2.8, for example, if RefClk transitions before DllFbD and DllFbDD transition, then delay must be removed from the delay line to make it in phase so a SR (shift right) command is sent to the shift control logic. Conversely, when RefClk comes after both DllFb's transition, then delay must be added to the delay line so a SL (shift left) command is issued. Ph180 is the opposite state of PhEQ as the rising edge of RefClk transitions between the falling edges of the DllFb clocks. In this state the DLL is exactly 180 degrees out of phase.

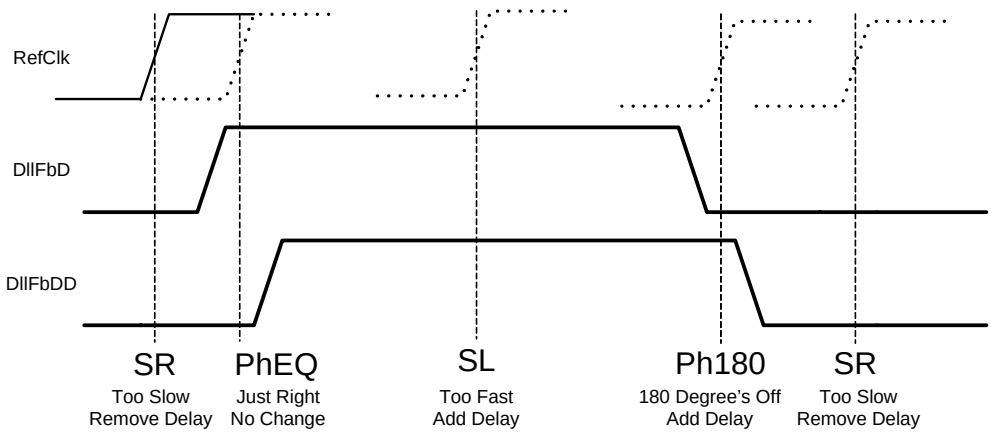

Figure 2.8 Phase Timing

The biggest advantage of the coarse phase detector is the ability to resolve a phase equal state, but the problem is that the phase equal state has a range of two coarse unit delay cells. This range limits how accurately the DLL can find a lock. For example, if the coarse unit delay cell had a delay of 100 ps because of the coarse phase detector hysteresis, the reference and feedback clock could be in a PhEQ state and still be 200 ps out of phase. In other words, the coarse phase detector does not resolve to a very fine

resolution. To improve phase resolution a dual loop DLL is used in this design. The first loop is the coarse loop, which uses the coarse phase detector and the coarse delay line to lock the DLL. Once within the minimum resolution of the coarse loop, the second loop, a fine phase loop, is enabled. The fine phase loop uses a fine phase detector and a fine delay line to improve the minimum phase resolution [3].

For fine phase detection a circuit is needed that can resolve a very tight phase difference. An arbiter can be used to accomplish this [12]. Using a set-reset latch to make the decision, the arbiter can determine whether the reference or feedback clock arrives first. Figure 2.9 shows the schematic of the fine phase detector used in this DLL design and how the arbiter function is adopted as a phase detector.

Similar to the coarse phase detector, RefPD and FbPD are phase split before being sent to the arbiter to ensure that each receives identical delay before the phase detection. After the arbiter has chosen which signal arrived first, the two flip-flops capture the state of the arbiter and hold the value for a full clock cycle so that the phase information can be sent to the shift control logic. In addition, to provide margin for setup time violations, the clock going to the output flip-flops (RefClkD) is delayed by approximately eight gates. The flip-flops are necessary to hold the state of the arbiter because the arbiter makes a decision on every transition of clock. Clocking the flip-flops off of the rising edge of the reference clock makes the fine phase detector send out phase data that was captured only on the rising edge of clock.

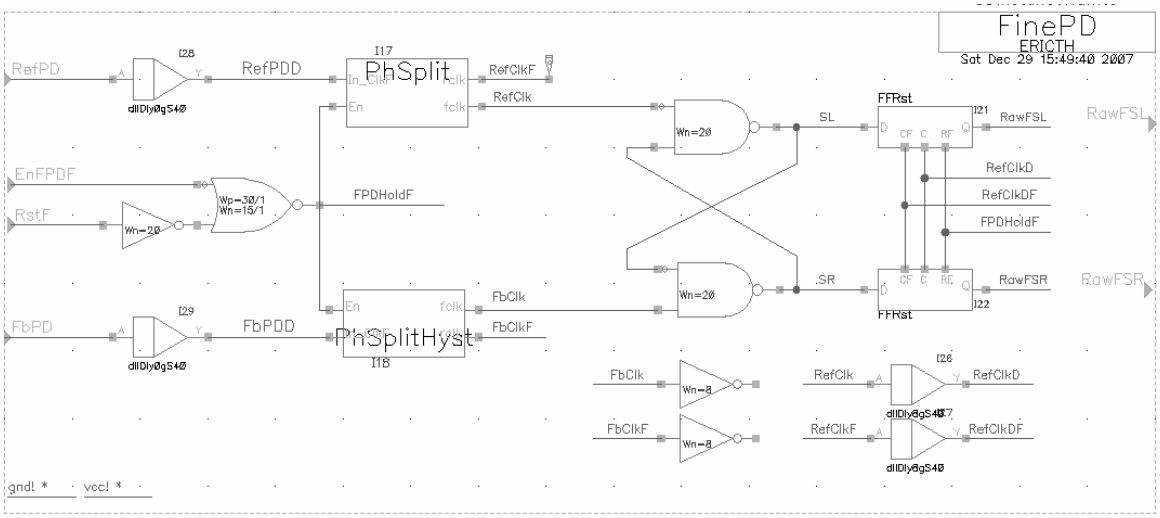

Figure 2.9 Fine Phase Detector (Arbiter)

The advantage of the coarse phase detector is that it can determine when the DLL is locked as well as when it needs to add or remove delay. From a circuit logic perspective, it is useful to have a signal that asserts when the DLL is locked and de-asserts when the DLL is not locked. Even though the fine phase detector can only decide whether the DLL needs to add or remove delay, it can make this decision at a much tighter phase resolution which improves the overall clock jitter. When the DLL is using the fine phase loop, determining the locked state of the DLL is not trivial. Typically, an averaging filter can be used to determine a lock by observing when the DLL has stopped shifting consistently in the same direction. Once in the fine loop, the lock determination becomes a function of the shift control logic.

2.5 Buffer Design

DLL buffers are used to strengthen and improve the voltage swing of the external clock and the outgoing data strobe. The configuration of buffers can vary widely from design to design. For the sake of simplicity, this design uses the same buffer for the input, output, input replica, and output replica models. For further simplicity, a self-

biased wide swing amplifier is used as seen in Figure 2.10. This buffer will not require the use of a voltage reference for operation and will output at full logic levels [12].

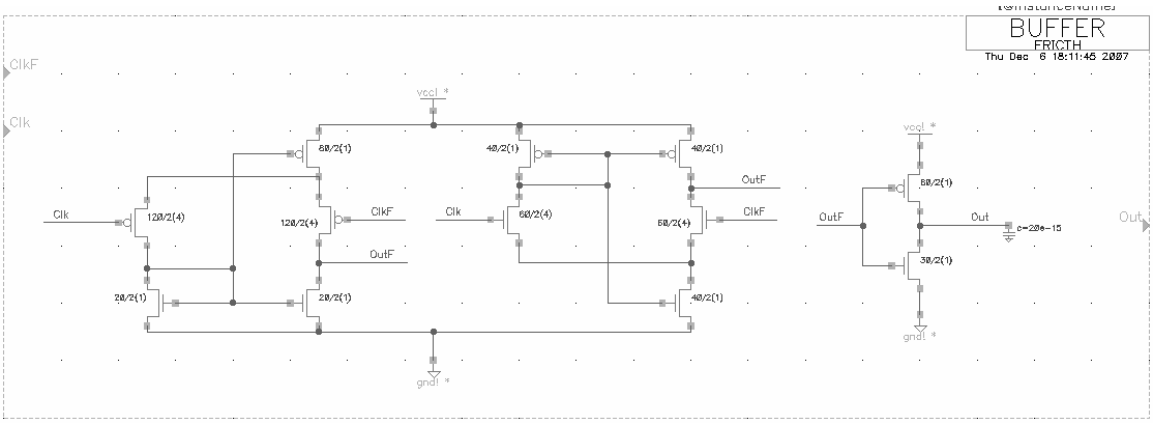

Figure 2.10 Wide Swing Self Biased Operational Amplifier Schematic

Using the same buffer for the replica and the forward path buffer will result in ideal modeling. For this design the input clock and output DQ buffers are the same for the sake of simplicity. As this design is focused on duty cycle correction more realistic modeling was not necessary.

Figure 2.11 shows two simulations performed on this input buffer to show its effectiveness. The top half of Figure 2.11 shows a transient simulation of the self-biased wide-swing buffer accepting a full logic level input clock at a clock period of 3 ns and outputting a full logic level after a propagation delay of 200 ps and 170 ps for the rising and falling transitions, respectively. The bottom half of Figure 2.11 shows the DC behavior of the buffer when the negative terminal (ClkF) is swept across 0 V to 1.8 V for discrete values on the positive terminal (Clk) incrementing in 200 mV steps. There is some distortion for very high or very low values of Clk but, since this buffer will be operated in a differential fashion, this distortion will not be a problem. The simulations show that this buffer will perform adequately for this design.

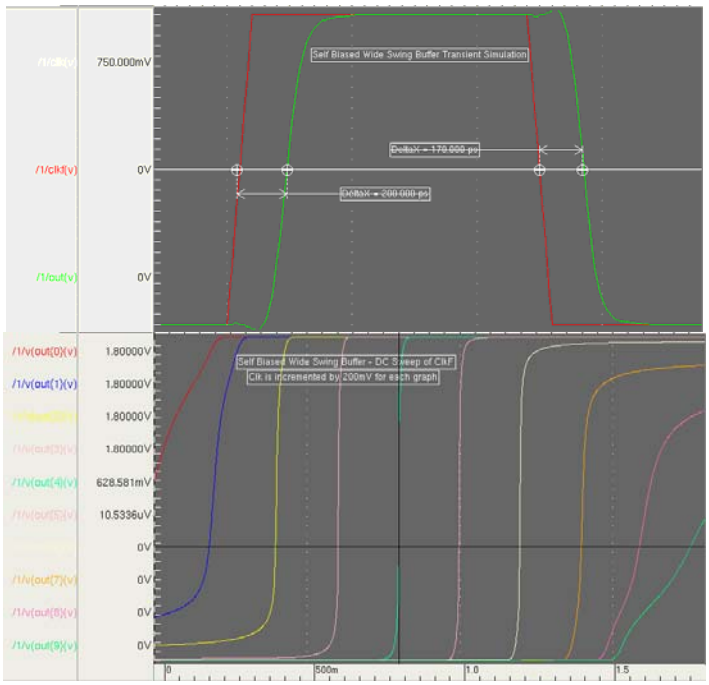

Figure 2.11 Transient (Top) and DC (Bottom) Input Buffer Simulations

2.6 Initialization

The process to determine the proper depth in the delay line such that the external and output clocks are in phase is referred to as the initialization of the DLL. In a register-controlled DLL initialization, delay cells are added consecutively while a phase comparison is done until the reference and feedback clocks are in phase [1]. This process can take upwards of 100 cycles after the resetting of the DLL [10], [13]. Another initialization method, called measure control or just measure initialization, performs a one-time measurement to find the lock point, sets the delay line at the depth, and then resumes register-controlled operation [11]. Measure initialization only takes a few cycles to complete (perhaps 10-20 cycles) and any errors in the measurement are corrected shortly after the register-controlled operation has resumed. Measure initialization is the method employed in this DLL design.

Figure 2.12 shows a block diagram of the measure controlled DLL. A time-to-digital (TDC) and a digital-to-time (DTC) converter are needed to perform the measurement [11]. Upon initialization, the start signal travels down the measure delay line (TDC) until the Stop signal fires and stores this depth in the measure delay line. Since Start and Stop have the same source (In), the difference between these two signals is $t_{CK} - (t_{D1'} + t_{D2'})$. The stored tap value is converted to the VDL (DTC) and the measure initialization is complete. Now the VDL has the same amount of delay as given by Equation (2) with an N = 1.

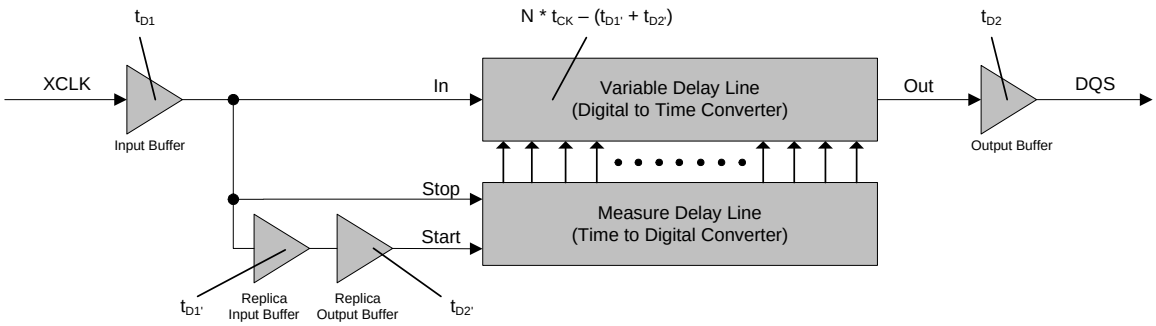

Figure 2.12 Measure Controlled DLL Block Diagram

For certain delay line configurations, it is possible to have the variable and measure delay lines be one and the same. For this to occur, the register-controlled function must be temporarily suspended during the measure initialization. This will be discussed in more detail in Chapter 4, DCC Design.

Figure 2.13 shows the schematic of the measure initialization circuitry. The Start signal from Figure 2.12 is implemented with the MeasDlyClkF signal. Likewise, the Stop signal is the MeasSclkF signal. Because FbPD is the feedback clock, it has propagated through the replica buffers. For this reason, FbPD is used to clock the Start flip-flop. This guarantees that MeasDlyClkF has propagated through the replica buffers.

The Stop flip-flop is initially enabled by MeasSclkF, but is thereafter clocked by RefPD, which is a reference clock that comes directly from the input clock buffer.

In similar fashion to the phase detectors, RefPD and FbPD are used as clocks and are phase split before being used on the Start and Stop flip-flops. Optional delay gates are provided for tuning the Start and Stop timing. Upon a DLL reset, after the phase detection is enabled, a counter is used to delay the start of measure initialization. This is done to allow all of the clocks in the system to settle so that the measurement can be as accurate as possible. The same counter is used to suspend the register-controlled operation and to fire the Start and Stop signals.

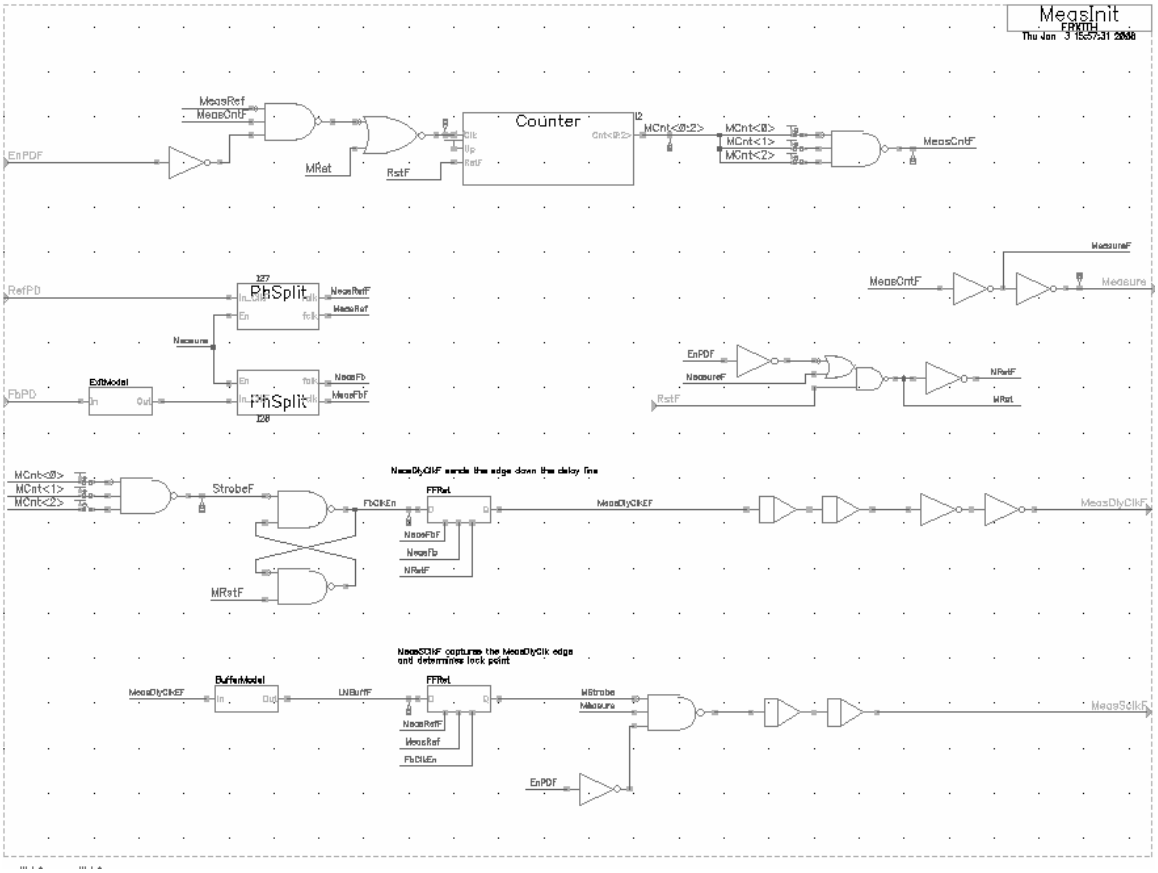

Figure 2.13 Measure Initialization Schematic

2.7 Locking

After initialization, the DLL is not always in phase. Once register-controlled operation has resumed the phase detector will decide if delay needs to be added, subtracted, or if no phase adjustment is necessary (PhEQ). For circuitry that depends on the DLL being in a locked state it is advantageous to ensure that the phase equal state does not just occur for a single cycle [14]. For this purpose a simple filter, consisting of a string of two cascaded flip-flops clocked by a reference clock, verifies that the coarse phase detect circuit outputs PhEQ on two consecutive cycles. This filter can be adjusted to detect one, two, or three consecutive PhEQ commands from the phase detector.

Figure 2.14 shows a flow chart of the locking process in this DLL. Upon a DLL reset, measure initialization is enacted which gets the phase close to the lock point. At this point, the phase detector compares the phase of the reference (RefPD) and feedback (FbPD) clocks. If FbPD comes in too fast relative to RefPD than delay must be added into the delay line. Conversely, if FbPD comes in too late then delay must be removed from the delay line. If RefPD falls within the hysteresis of the FbPD clocks then the phase detector reports a PhEQ. If the phase detector reports two consecutive PhEQ commands then the DLL is considered locked.

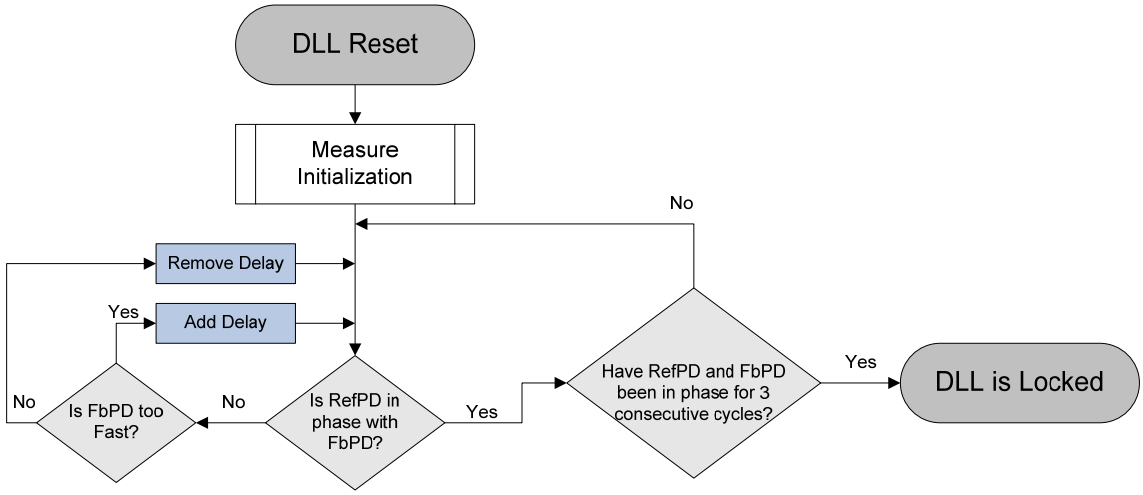

Figure 2.14 DLL Lock Flow Chart

Other functions can be enabled based on the locking of the DLL. In this design, once the DLL is locked, the DCC is enabled and allowed to initialize (this will be discussed in more depth in Chapter 4). Also, once the DLL is locked, the fine phase loop is enabled (and the coarse disabled) to further improve the phase resolution. The locking of the DLL must be gated by the coarse phase detect because the hysteresis of the coarse phase detect allows a phase equal state to be determined. Finding the phase equal state using the fine phase detector is substantially more complicated than just filtering the PhEQ signal as in the coarse phase detect case.

2.8 Filtering

To decrease the DLL's sensitivity to transient noise, it is beneficial to use a digital filter to filter the phase adjustments generated by the phase detector [1], [15], [16]. The filter has no effect on operation during initialization, but afterwards it will slow the response of the DLL to changes in PVT [3]. Also, in some instances when the DLL is close to locking, the DLL can oscillate across the phase equal boundary indefinitely. A

shift filter can be used to filter out this behavior so that no shifts are performed when the phase detect is oscillating. Because a digital filter prevents the DLL from shifting as frequently, power dissipation and clock jitter are both reduced.

The filter used in this design is a basic phase accumulator [17]. Via a string of flip-flops either shift-left (SL) or shift-right (SR) commands from the phase detector are filtered. In order to issue a shift in either direction eight consecutive SL or SR commands must be issued. When in fast shift mode, however, only four consecutive SL or SR commands are needed to issue a shift. The shift filter can be used for either coarse or fine shifts.

It is beneficial to have a faster shift mode because it allows for a faster phase response. The fast shift mode can be useful immediately after the measurement has completed to help find a lock sooner. Once a lock has been determined, the filter can be placed into slow shift mode to slow the phase response, which helps to improve clock jitter. Another useful application comes when the DCC is introduced into this design, which is discussed in Chapter 4.

Figure 2.15 shows the schematic of the shift filter used in this design. There are eight flip-flops connected in a series for both the SL and SR direction. The RefD reference clock is phase split and used to clock the flip-flops in the SL and SR chains. The signals rawSL and rawSR are the inputs to the flip-flop chains and can either be fine or coarse phase signals based on whether the fine loop has been enabled as indicated by EnFPDF. In fast shift mode, rawSL or rawSR must be high for three consecutive cycles. If three consecutive shifts are not issued then a shift is not issued and the filter continues to clock. Slow shift mode operates identically to fast shift mode except that eight

consecutive shifts are needed to issue a shift. The shift filter is only reset when a shift is issued regardless of the direction. This slows the frequency of the DLL's shifting because the minimum number of cycles between shifts is eight for slow shift mode and four for fast shift mode.

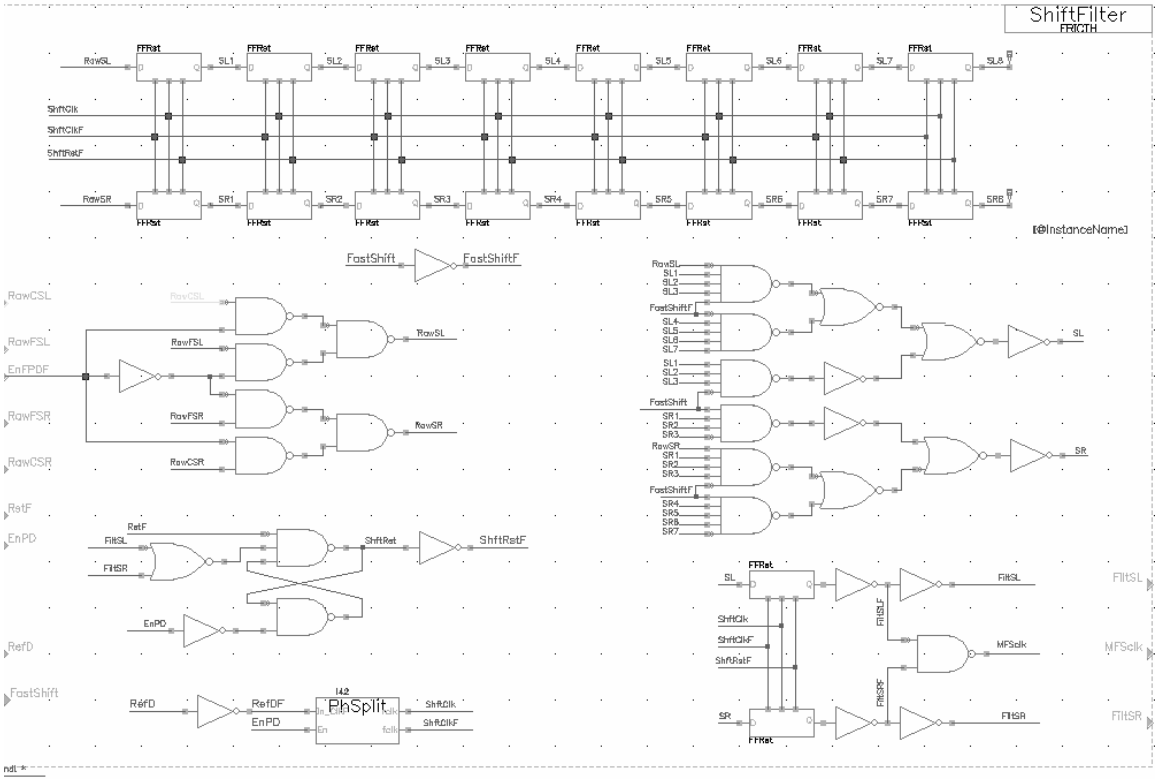

Figure 2.15 Schematic of the Shift Filter

Figure 2.16 shows a simulation of the shift filter for a 5 ns clock period. Notice that a continuous string of rawSL commands are being issued to the shift filter. When FastShift is high the filter is in fast shift mode and there are 20 ns or four cycles between shifts. When FastShift is low the filter is in slow shift mode and there are 40 ns or eight cycles between shifts.

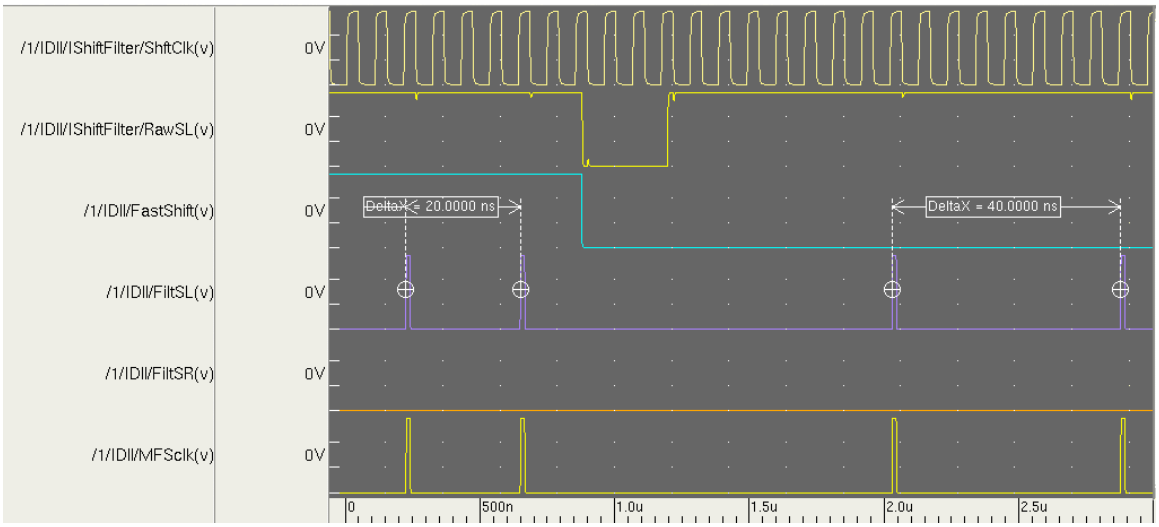

Figure 2.16 Simulation of Shift Filter Showing Slow and Fast Shift Modes

2.9 Fine Delay Line (Dual Loop DLL)

Because the resolution of a digital DLL is limited by its coarse unit delay cell value, it is practical to use a dual loop DLL and a fine delay line to improve the minimum phase resolution [3]. A dual loop DLL block diagram is shown in Figure 1.5. To implement the fine delay line for this DLL a phase mixer or phase interpolator is used. A phase mixer consists of a number of tri-state inverters connected in parallel, which are enabled according to the position in the fine shift register [8].

Figure 2.17 shows the configuration of the phase mixer. Twelve tri-state inverters are connected in parallel, but only six are enabled and actively driving the output inverter at any one time. The phase mixer works by blending two phase signals: ϕ_1 and ϕ_2. If more tri-state inverters are driving ϕ_1 than ϕ_2 then the output will be weighted towards ϕ_2. Conversely, if more tri-state inverters are driving ϕ_1 output will be closer in phase to ϕ_1. If an equal number of inverters drives both ϕ_1 and ϕ_2 then the output should fall exactly halfway between ϕ_1 and ϕ_2.

Figure 2.17(a) shows the default configuration where the output phase is balanced between ϕ_1 and ϕ_2. Figure 2.17(b) shows the configuration where the output is weighted completely towards ϕ_2. This results in the maximum amount of delay from the fine delay line. Figure 2.17(c) shows the configuration where the output is weighted completely towards ϕ_1. This configuration produces the smallest amount of possible delay through the phase mixer.

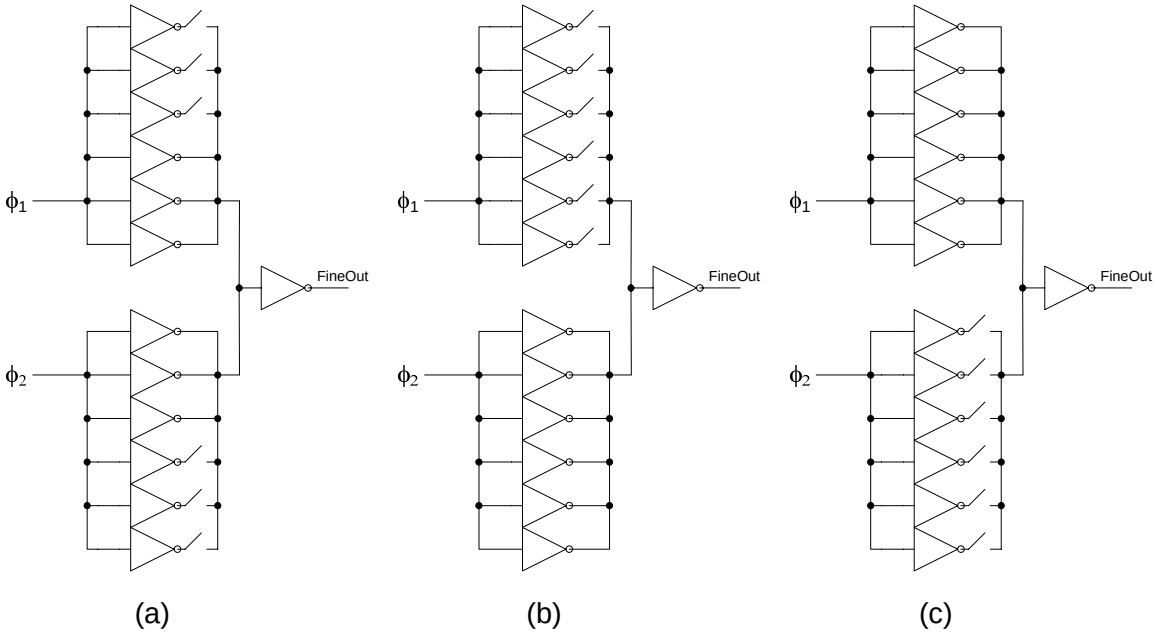

Figure 2.17 Fine Phase Mixer Configurations: (a) Default, (b) Max. Delay, and (c) Min. Delay

Figure 2.18 shows the schematic of the fine phase mixer and fine shift register. The outputs of the shift register bits are the enable signals for the tri-state inverters. The shift register starts with a 000111111000 stored in the 12 shift register bits. A fine shift left (FSL) command will shift all of the 1's in the shift register to the left. Likewise, a fine shift right (FSR) will shift the 1's to the right one register to the right. When the shift register contains an 111111000000 or a 000000111111, the fine delay line is reset and a coarse shift command is sent to the coarse shift control logic.

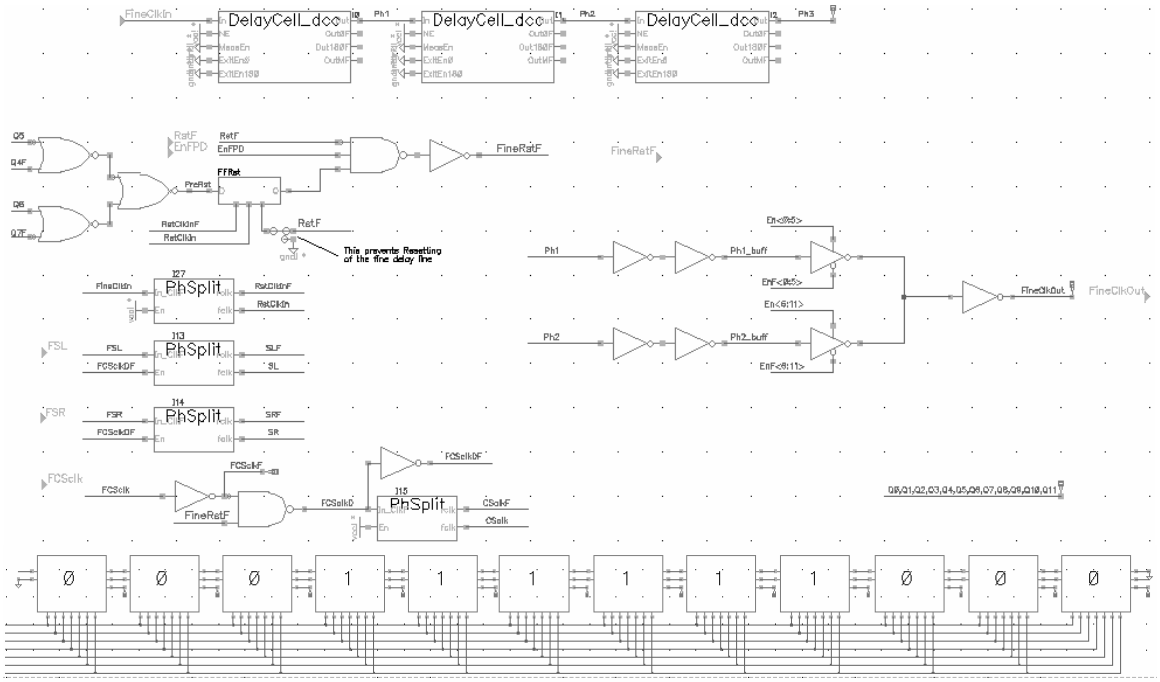

Figure 2.18 Fine Phase Mixer and Shift Register

ϕ_1 and ϕ_2 (Ph1 and Ph2 in the schematic) are generated by sending the input, FineClkIn, through a unit coarse delay cell. For this reason, the difference between the minimum and maximum phase mixer delay is equal to exactly one unit coarse delay cell. Because of this the phase distance from the reset stage to either max ϕ_1 or max ϕ_2 state should be equal to half of a single unit delay cell. This acts to reduce delay mismatches between the coarse and fine shifts [8].

Figure 2.19 shows the simulation result of the fine phase mixer. The difference between the minimum and maximum delay is about 140 ps, which is close to the unit delay cell value of approximately 100 ps. Notice that the phase difference from the reset state to the maximum delay is about 50 ps and the reset to minimum delay is about 90 ps. The maximum fine step, which is the maximum amount that the fine delay line can shift the output phase by in a single delay stage, is about 32 ps. Because this is the largest possible instantaneous phase shift, this is the theoretical resolution limit of the DLL.

Notice also that the rising and falling edges track well with each other which indicates that the fine phase mixer does not introduce any duty cycle skew.

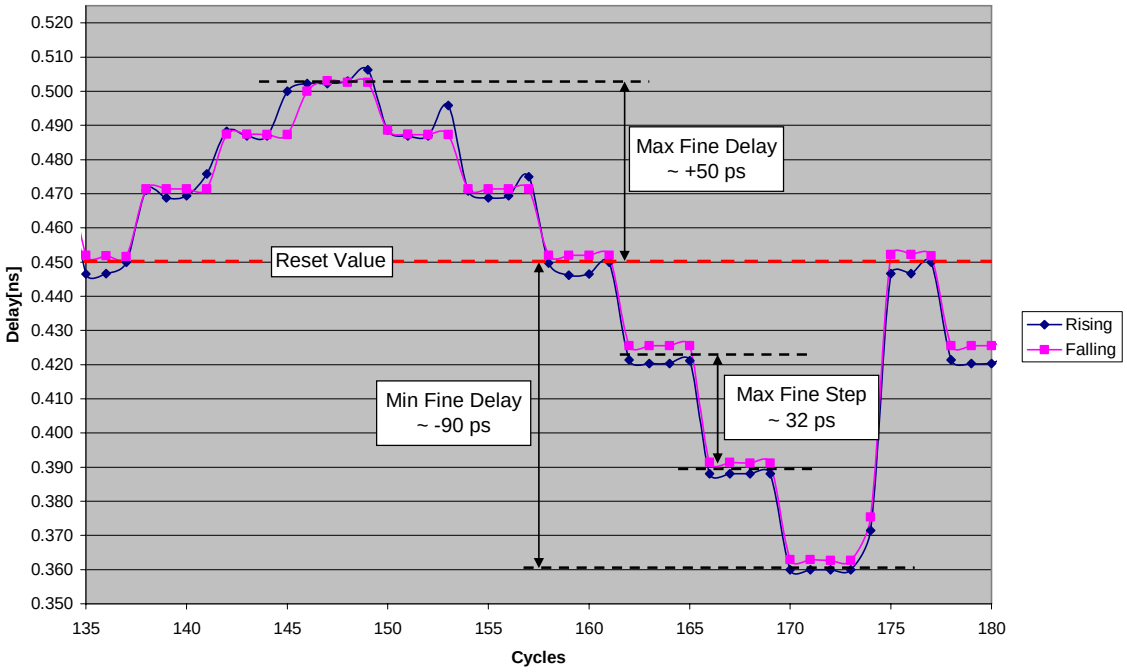

Figure 2.19 Simulation of Fine Phase Mixer

The imbalance between the maximum fine delay and minimum fine delay is a result of non-linear transitioning on the phase-mixed node. When the state of the fine phase mixer is close to the reset state there is more contention on the phase-mixed node. This contention between the ϕ_1 and ϕ_2 nodes causes the transition time to be skewed toward the ϕ_1 phase. Figure 2.20 plots the phase-mixed node vs. time, which shows how this is possible. If the trip point of the output inverter is approximately one quarter of the voltage supply then the phase separation of the phase-mixed node is not balanced and resembles imbalance between the minimum and maximum fine delay stages.

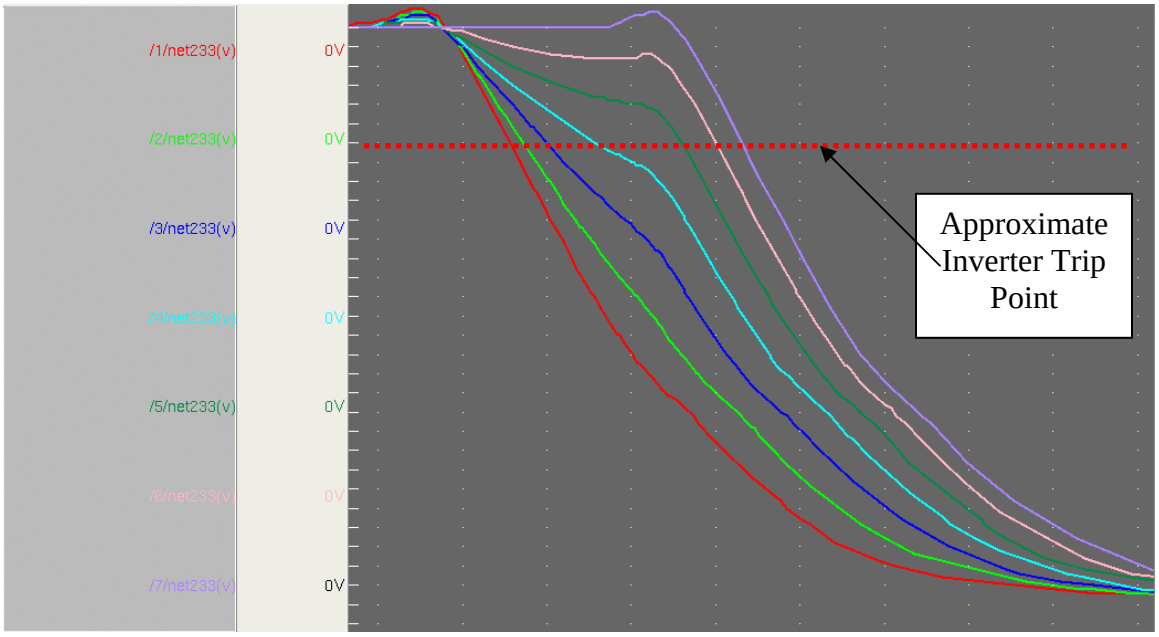

Figure 2.20 Simulation of Phase-Mixed Node for all Phase Mixer Stages

The design of a linear phase mixer is a challenging prospect in and of itself. Because of time constraints further optimization of this circuit was not done for this design. Nonetheless, the fine phase mixer still functions sufficiently as a fine VDL. The biggest advantage of the fine phase mixer is the reduction of the minimum phase resolution from a coarse unit delay cell of approximately 100 ps to a fine delay step which is about 30 ps.

CHAPTER 3 – DCC DESIGN

3.1 Half-Cycle Delay Line DCC

One method for a DCC to determine the 50% duty cycle is through the use of a Half-Cycle Delay Line (HCDL). A HCDL can be created by connecting two identical VDL's in series and adjusting their delays until the total delay through both is equal to the clock period. The delay through one of these VDL's is half of the clock period. This HCDL produces a clock signal whose rising edge is 180 degrees out of phase with the input regardless of the input duty cycle [4], [6], [18].

Figure 3.1 shows a block diagram of a HCDL DCC. A phase detector is used to increase the delay in both HCDL's until the delay through both is equal to the clock period. When this occurs, the output of the first HCDL, 180º Out, is exactly 180 degrees out of phase with the input clock, DccIn. DccIn and the 180º Out are then phase combined to create a single-ended output clock, DccOut, whose duty cycle is 50%.

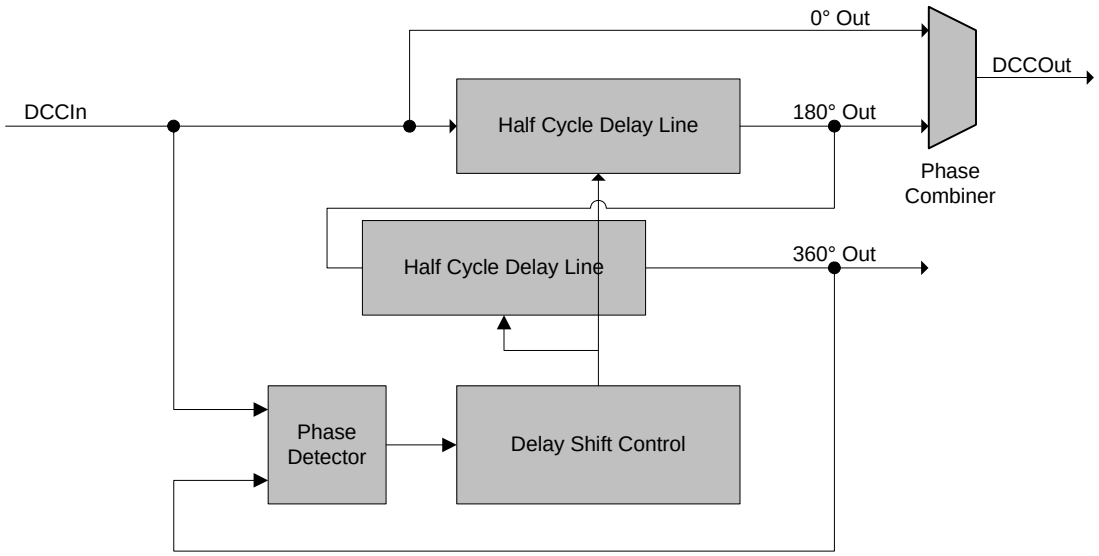

Figure 3.1 HCDL DCC Block Diagram

An example of the HCDL DCC Timing can be seen in Figure 3.2. In this example, the input duty cycle is significantly less than 50%, but after the HCDL delays DccIn by half of the clock period, the rising edge of 180º Out is exactly 180 degrees out of phase with DccIn. The phase combiner uses only the rising edges of DccIn and 180º Out to transition the output. Since the phase combiner does not use the falling edge transitions, the duty cycle of DccIn is inconsequential to the output duty cycle.

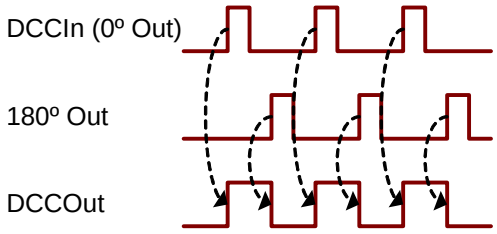

Figure 3.2 HCDL DCC Timing Diagram

There are a number of advantages to the HCDL DCC. The most significant advantage is a wide input duty cycle range. Because the determination of the 180-degree phase is done independently of the input duty cycle, the output duty cycle is limited only by the minimum clock period limitation of the DCC. Another advantage is the closed

loop configuration. There is a closed phase loop between DccIn and the two HCDL's, which results in a stable phase loop. Finally, the HCDL DCC is a stand-alone integrated circuit (IC) design, so it can easily be added or removed to a DLL design.

Unfortunately, while the advantages of the HCDL DCC are appreciable so are the disadvantages. The largest drawback to the HCDL DCC is the layout area requirement. Because the two HCDL's must have enough delay to measure t_{CK}, a significant number of delay cells are needed. In addition to the delay cells, a shift register must also be included, which contains even more transistors than the delay cells. Layout area for control and phase detect logic is also needed. Another disadvantage of the HCDL DCC is its impact to the forward path delay. Because the falling edge must pass through a HCDL it will be delayed half a clock period before arriving at the phase combine circuit. This delay can affect DLL wakeup times. Power consumption is also a considerable disadvantage of this DCC design. Numerous HCDL delay cells have to toggle as well as the DCC control and phase detect logic. All of these operations increase power consumption.

In summary, the HCDL DCC is a viable candidate for a DCC design because it has great input duty cycle range, utilizes a closed phase loop, and is a stand-alone IC. The HCDL DCC is not an ideal candidate for a DCC design because of the layout area requirements, forward path delay impact, and power consumption. Since these drawbacks are too significant for this to be a practical DCC design, other DCC options will be considered.

3.2 Open-Loop DCC

An open loop (OL) DCC utilizes two full-length delay lines for the 0º and 180º phases [19]. The DLL uses the 0º VDL to align the external and output clocks. A duty cycle error correction circuit determines how much duty cycle correction is needed and adjusts the 180º VDL independently of the 0º VDL to correct the duty cycle. A phase combine circuit is used to combine the 0º and 180º phases and output a single-ended clock with a corrected duty cycle.

Figure 3.3 shows the block diagram of the OL DCC. A differential clock buffer provides Ref and RefF to the 0º and 180º VDL's, respectively. Both VDL's are initialized to the same point, and once a lock has been attained, the Duty Error Correction Block is enabled. The Duty Error Correction Block adjusts the 180º VDL based on a duty cycle offset calculated from Ref and RefF. The outputs of the VDL's are phase combined to create a single-ended output clock for the output buffer. The phase combiner could be omitted if the output buffer were also differential.

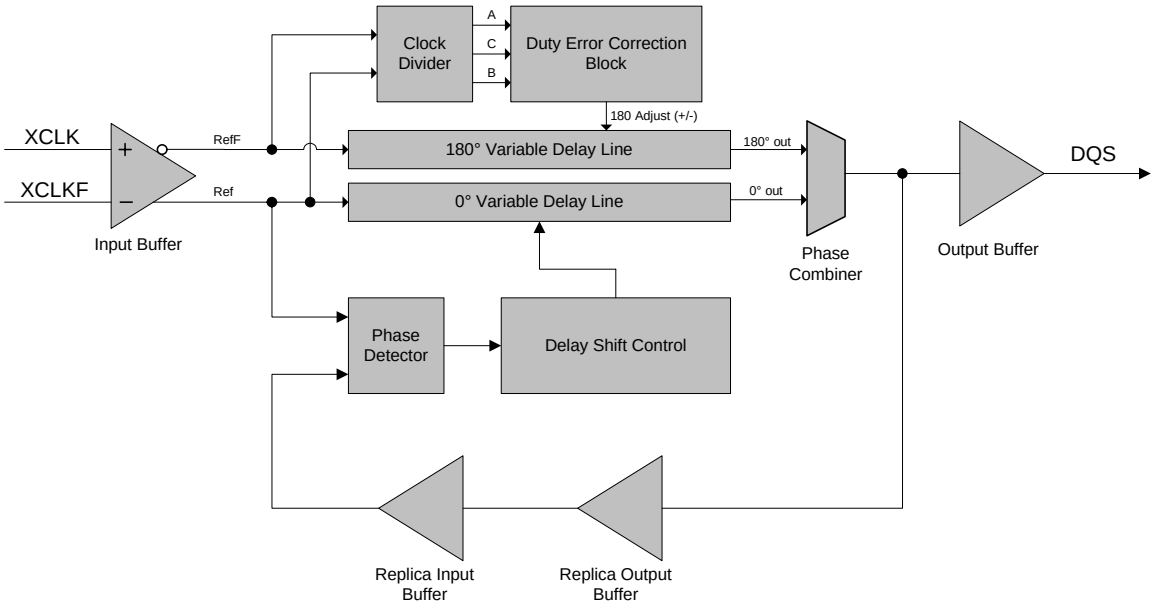

Figure 3.3 Open Loop DCC Block Diagram

Because there is not a closed phase loop in the OL DCC, it is critical for an accurate calculation of the duty cycle. To help accomplish this Ref and RefF are divided to create the A, B, and C signals. A and B are simply divided versions of Ref and RefF. C is an inverted version of A. The clock high time, t_H, is equal to the difference between the rising edges of A and B and the clock low time, t_L, equals the difference between B and C. Figure 3.4 summarizes this timing.

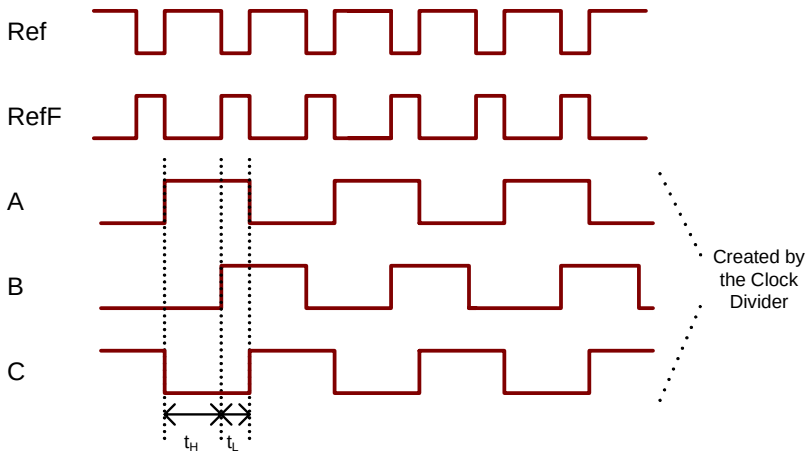

Figure 3.4 Timing Diagram for the Clock Divider

Figure 3.5 shows a block diagram of the clock dividers and the Duty Error Correction Block. The Duty Error Correction Block is essentially two DLL's consisting of a VDL to match t_H or t_L, a phase detector, and some shift control logic that sends information to the Duty Error Calculator. The Duty Error Calculator monitors the difference between t_H and t_L and adjusts the 180º VDL accordingly [20].

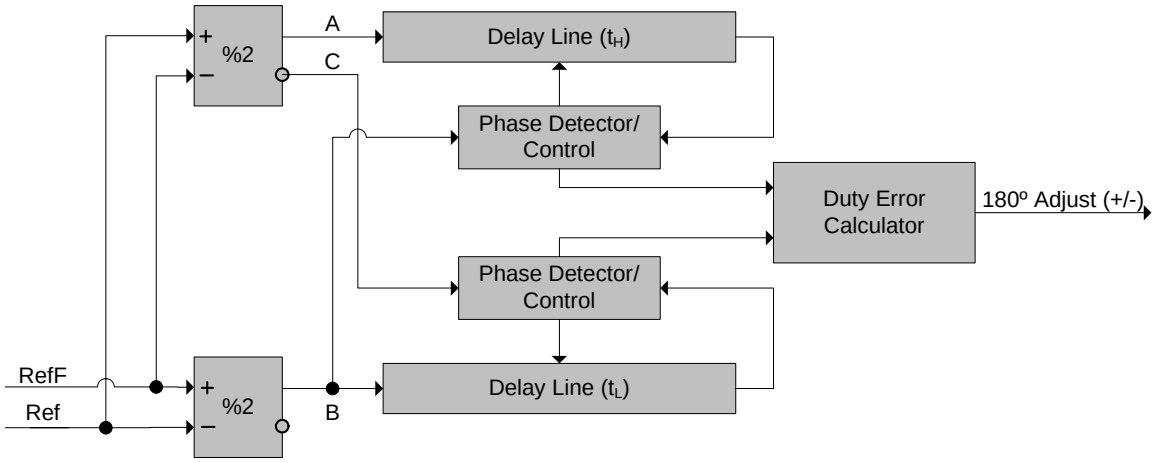

Figure 3.5 Block Diagram of Clock Dividers and Duty Error Correction Block

While not an intuitive way to correct duty cycle offset, the OL DCC does a sufficient job of correcting duty cycle. The 180º VDL allows for a wide duty cycle tuning range. Due to the open loop configuration of the OL DCC it is important to make sure that the Duty Error Calculator computes the duty cycle offset properly. The Duty Error Correction Block is the most critical element to this DCC design.

A significant advantage to this design is the impact on the forward path delay. Since the DCC needs few gates to insert duty cycle correction, there is essentially no impact to the forward path. Another advantage is the integrated nature of the DCC. The DLL function is not affected whatsoever by changes in the DCC. Another advantage of this design is the large duty cycle range allowed by using the full length 180º VDL.

A major drawback of the OL DCC is the fact that it is an open loop system. This means that it is possible for error accumulation to occur. Since there is not a closed loop, the DCC will not be able to determine how far off the duty cycle might be. Another considerable disadvantage is the effort required to design the Duty Error Correction Block. The accuracy of the OL DCC is dependent on the proper implementation of this circuitry. This design could be very time consuming. Finally, power is another drawback to this DCC. The Duty Error Block and Calculator are constantly checking the input for changes in duty cycle. Fortunately, the clock division helps to halve the clock frequency in the DCC so this helps to alleviate the power dissipation.

In summation, the OL DCC is an integrated DCC design that has a wide duty correction range and minimal forward path impact. These benefits come at the cost of a more complex design and higher power dissipation. The most unattractive feature of this DCC design is the fact that the duty cycle correction is an open loop and has the potential for error accumulation. Consequently, another DCC design will be considered.

3.3 Integrated Half-Cycle Delay Line DCC

The Integrated Half-Cycle Delay Line (IHCDL) DCC only uses only a single delay line for both DLL and DCC functions. Through a unique initialization method the HCDL's are implemented in the forward path VDL. This same VDL is also used to align the external and output clocks.

Figure 3.6 shows the block diagram of the IHCDL DCC. The block diagram is similar to the basic DLL block diagram, but does have some significant differences. The biggest difference is the VDL, which now has two tap points: one for the 0º phase and the other for the 180º phase. The 0º Out and 180º Out signals are phase combined before

being sent to the output. The Delay/Shift Control circuitry is also different from the typical DLL as a different shifting method must be employed to implement the duty cycle correction.

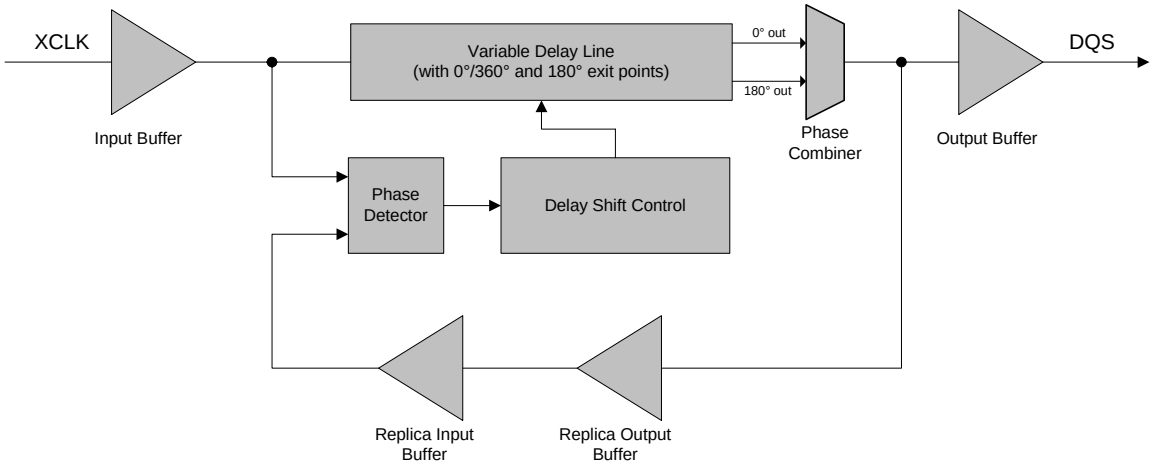

Figure 3.6 Integrated DCC DLL Block Diagram

To find the half cycle time two HCDL's must be generated in the VDL. By taking advantage of the fact that the DLL can lock on multiple harmonics (or N's) the IHCDL DCC can generate two HCDL's in a single VDL. Assuming that there is enough delay available in the VDL and the lock point is found, delay can be added in the VDL until the reference and feedback clocks are in phase again. When the lock point is found again the feedback clock will be delayed by a full cycle from its previous state. This effectively measures the clock period in terms of unit delay cells. Because there are a finite number of unit delay cells that equal the clock period, exactly half this number of delay cells is equal to half of the clock period. Counting the number of delay cells required to lock on the next lock point and dividing this number in half is how the two HCDL's are formed in the IHCDL DCC.

Figure 3.7 uses a shift register diagram to show how the IHCDL DCC finds the next N (or N+1) lock point. The '1' to '0' transition on the N Locked diagram shows the initial lock point. To find the N+1 lock point, delay is added until the clock and reference clocks become in phase again. If the number of delay cells between the N and N+1 lock points is counted then the phase at the tap exactly halfway between the N and N+1 lock points would have a phase that is 180 degrees out of phase with either lock point.

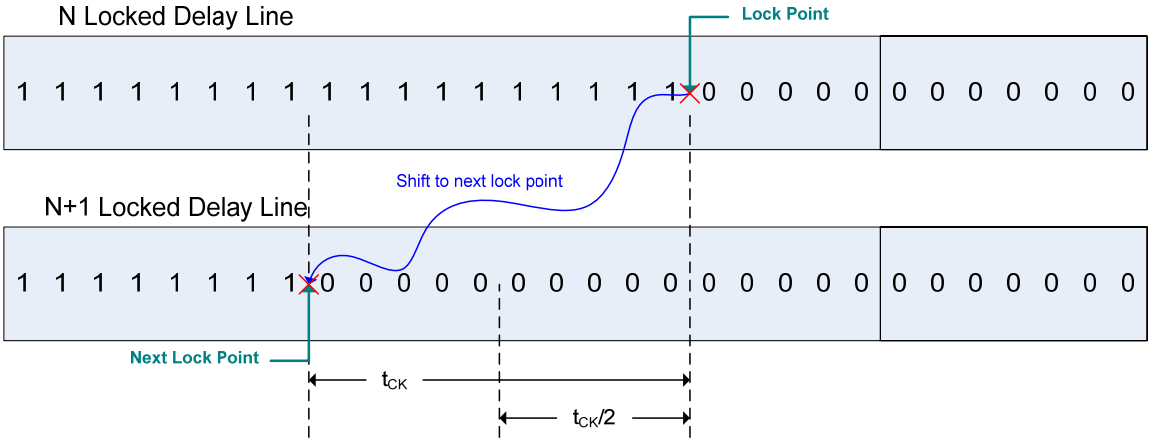

Figure 3.7 How to Find the Next N Delay Line Diagram

For example, let X be the number of unit delay cells required to move the lock point to the next N. If X is an even integer than X/2 is also an integer. Problems arise when X is an odd integer because X/2 is not an integer. As the unit delay cells are discrete and cannot be split this means that, when X is odd, there will be a quantization error of one unit delay cell. This quantization error can result in a substantial error at increased clock speeds. To reduce this quantization error, fine shifting is used to reduce the error to +/- one fine delay cell instead of +/- one coarse unit delay cell.

Because two exit points are needed, the use of a second shift register is required. The 0º shift register sets the exit tap point for the 0º phase. Likewise, the 180º shift register sets the tap point for the 180º phase. Figure 3.8 shows a shift register diagram

with the dual shift registers. Initially, the DLL sets the same lock point in both the 0º and 180º shift register. To find the N+1 lock point the 0º shift register is shifted to the left. For every two shifts in the 0º shift register one shift is issued in the 180º shift register until the reference and feedback clocks (which are still based on the 0º tap) are in phase again.

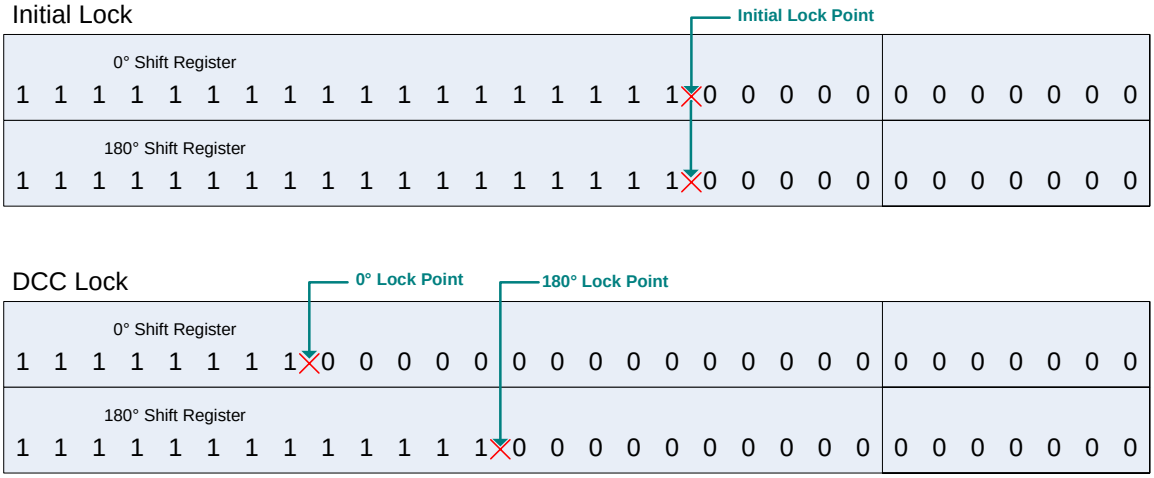

Figure 3.8 Delay Line Diagrams after the DLL and DCC have Locked

Once the 0º and 180º lock points have been established then the 0º and 180º taps are phase combined to generate a single ended clock that has a 50% duty cycle and will make the output clock be in phase with the external clock. Figure 3.9 shows this again with a shift register diagram and a phase combine symbol. It should also be noted that the output timing diagram for the IHCDL DCC will be identical to the HCDL DCC, which is shown in Figure 3.2.

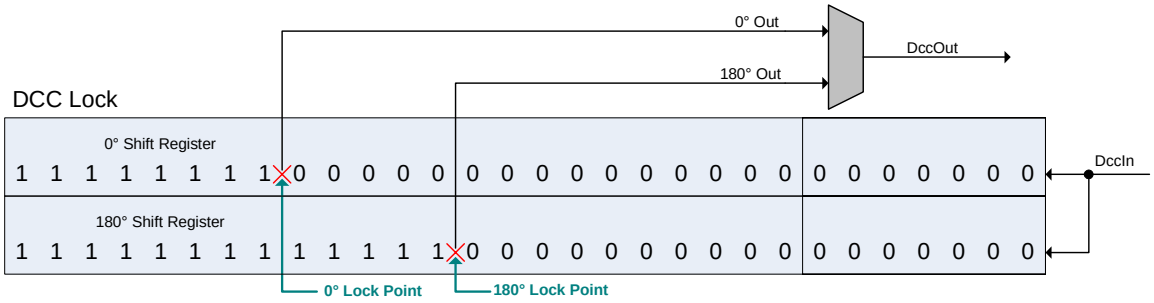

Figure 3.9 Delay Line Diagram Showing How the 0° and 180° Output are Generated

The major advantage to the IHCDL DCC is a reduction in the required layout area. The integrated delay line of the IHCDL DCC consumes less area than either OL or HCDL DCC because they require delay lines for both 0º and 180º phases. Unfortunately, a 180º shift register is required for all three of the DCC designs. The IHCDL DCC uses the same phase detector that the DLL uses which is another advantage over the other two designs, which have DCC specific phase detectors. The shift control logic is also less complicated and smaller than the OL DCC.

Another advantage of the IHCDL DCC is the impact on forward path delay. Neglecting the added depth in the delay line, the IHCDL has approximately the same forward path delay impact as the OL DCC, which is minimal. The duty cycle range also is equivalent to that of the other two wide range duty cycle DCC designs.

The main drawback of the IHCDL DCC is the method in which duty cycle is calculated. The DCC uses a closed phase loop system based on the 0º phase, but the 180º phase is not on a closed loop. The 180º phase is dependent on an accurate shift count by the shift control logic to provide a corrected duty cycle. The design of this DCC focuses on this weakness and uses methods to ensure that the shift count remains accurate.

Power dissipation could be expected to be similar or slightly better than the HCDL DCC, but since a string of delay cells equal to a full clock period are toggling in the delay line, a decent amount of current must be drawn from the power supply. Unfortunately, power dissipation is a necessary tradeoff for duty cycle correction.

Of the three DCC's compared in this chapter the IHCDL DCC has a slight advantage over the OL and HCDL DCC's because of its integrated delay line topology. IHCDL has the same wide duty cycle range as the other DCC designs. IHCDL and the OL DCC both have minimal impact to the forward path delay, which is better than the HCDL DCC. For these advantages, the IHCDL DCC is the chosen DCC to design and characterize for this thesis.

CHAPTER 4 – INTEGRATED HCDL DCC DESIGN

4.1 Exit Point Delay Line

In an Integrated HCDL DCC two delay line exit (tap) points are required for the 0º and 180º phases. An exit point delay topology is used in this DLL and DCC design to accomplish this. In an exit point delay line, the shift register determines the tap in which the clock will exit the delay line as opposed to the entry point delay line where the shift register determines the clock entry point [8].

Figure 4.1 shows a diagram of an exit point delay line and the associated shift register. The unit delay cell has the same delay as the entry point delay line, but the unit delay cell has an exit NAND gate on the output. One input of the exit NAND receives the propagated ClkIn signal while the other input receives an exit enable signal. A delay cell becomes the exit point whenever the current shift register bit has a '0' stored in it and the upstream bit is a '1'. The small red numbers in the diagram indicate the logic levels for the indicated shift register bit values.

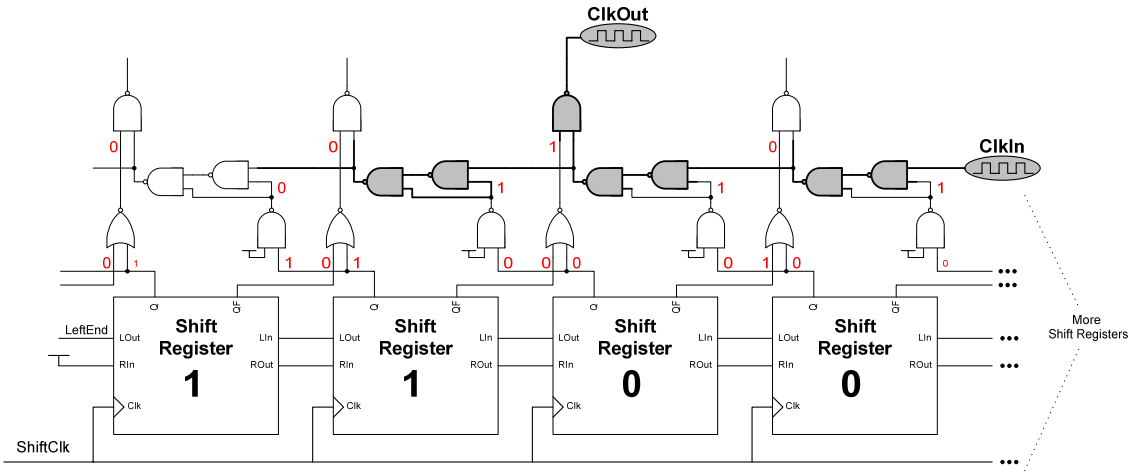

Figure 4.1 Clock Exit Point and Shift Register Value Diagram

For this design the delay line cells were designed to have a 100 ps delay per cell. This was done using a Monte Carlo simulation to determine the sizing of both the delay line NAND gates and the exit NAND gates. A total of 128 stages were used in the delay line due to the configuration of the exit tree, which will be discussed in Section 4.3. This means that the entire delay line should have a length of 12.8 ns.

4.2 Dual Shift Register

A distinct advantage of the exit point delay line is its ability to have more than one exit point. This advantage is utilized in the IHDCL DCC as the 0º and 180º phases are generated at two different tap points separated by a half clock period worth of delay cells. In order to keep track of the two exit points, two shift registers are required. One shift register will keep track of the 0º exit point and the other will keep track of the 180º exit point. Figure 4.2 shows a diagram of the Dual Shift Register used in this DCC design and how the 180º exit tap is determined based on the 180º shift register values.

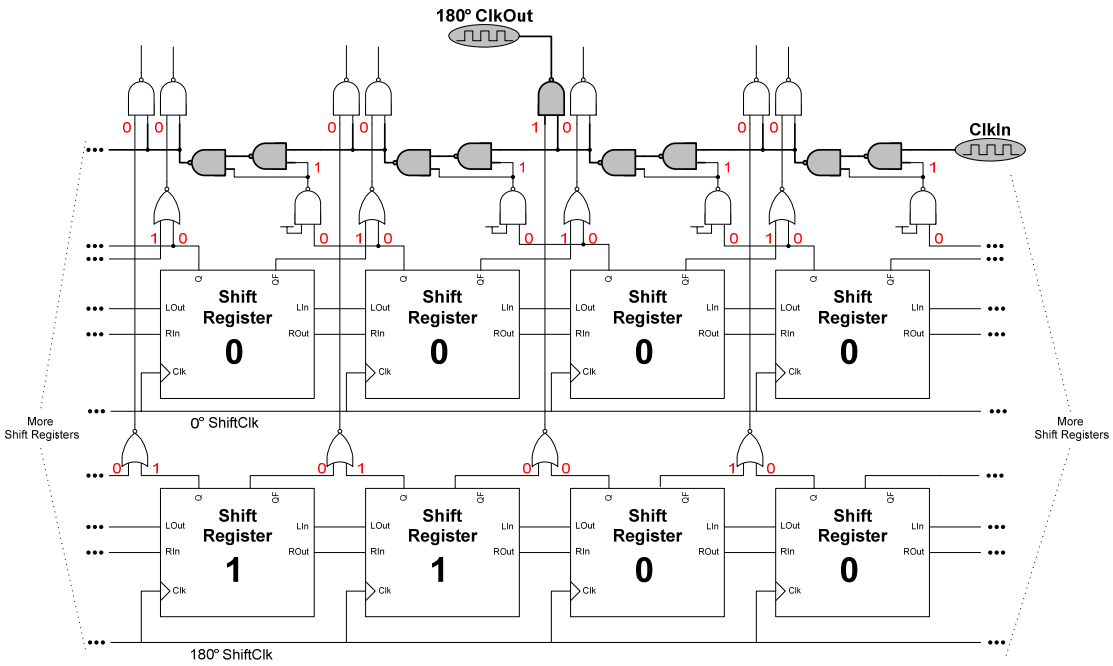

Figure 4.2 Dual Shift Register showing 180° Exit Point

In this configuration, the 0º shift register (the upper shift register) ultimately controls the final exit point of the delay line. Recall that the 180º lock point is the halfway point between the 0º and 360º lock points. The 180º exit point is determined again by the "1" to "0" transition in the 180º shift register. When the 180º upstream bit is a '1' and the downstream bit is a '0,' the 180º exit NAND gate is allowed to pass the clock to the 180º exit tree, which will be discussed in the following section.

Because of the dual exit points, two exit NAND gates are required per unit delay cell for the 0º and 180º exit points, respectively. Because two NAND gates load the second NAND gate in the delay cell, this capacitive gate load must be modeled on the output of the first NAND gate in the delay cell. If this load is not modeled there would be a rising vs. falling edge skew on the output clock which would increase with depth in the delay line. To model this load two more NAND gates must be added to the delay cell.

In total there are six NAND gates in the unit delay cell. Two gates are for the VDL. Another two are placed on the output and are for the 0º and 180º exit points. Two more act to model the exit NAND's, but an additional function is added to one of these gates. During measure initialization, one of the model NAND gates is used during initialization to set the shift register bits. This approach allows measure initialization with only one delay line. The other model NAND gate is simply a dummy gate present only for its capacitive load. Figure 4.3 shows the schematic of the unit delay cell for the IHCDL DCC.

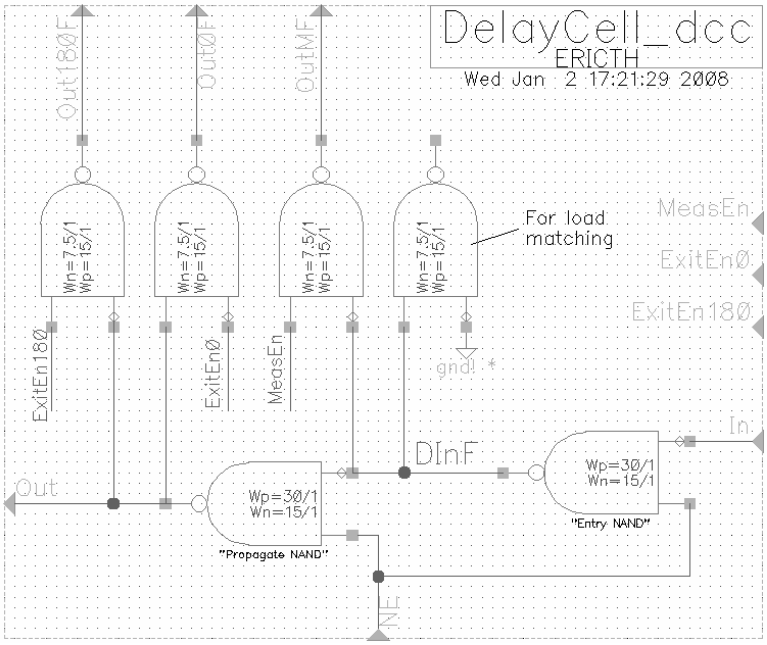

Figure 4.3 Unit Delay Cell Schematic

4.3 Exit Tree

In an exit point delay line the clock can exit the delay line after any delay cell. Because there are a large number of exit points, there are a large number (equal to the number of delay cells) of exit paths for the clock propagating down the delay line. There is a unique exit path for each delay cell, which eventually decodes down to a single path: the output of the delay line. The exit tree effectively decodes all of the exit points down to a single path.

Figure 4.4 shows a diagram of the clock exit tree. The first stage (or entry NAND) of the exit tree is integrated into the unit delay cell. The exit path is enabled by the same enable signal that enables the exit NAND gate. This enable signal progressively enables all of the gates in the path of the exit NAND, which creates the exit path for the clock.

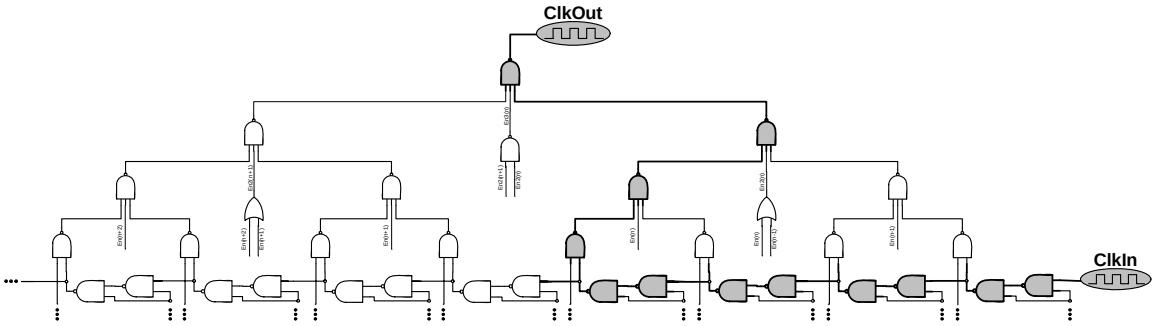

Figure 4.4 Clock Exit Tree Diagram

Since there is an exit point for both the 0° and 180° tap points in the IHCDL DCC then there must also be 0° and 180° exit trees. For simplicity Figure 4.4 only shows a single exit tree, but for this design the two exit trees are interleaved. Since there is a total of 128 stages ($2^{\wedge 7}$ stages) then the exit path must have a total of seven NAND gates. The first exit path NAND is the exit NAND imbedded in each delay cell. The rest of the exit tree contains six tiers of exit NAND gates.

Because the exit path becomes part of the forward path, once the proper delay cell is selected, is it important that the exit tree does not introduce clock skew. A typical two input NAND gate, as seen in Figure 4.5(a), consists of two PMOS transistors in parallel that share an output node with two NMOS transistors connected in series. The two inputs are arbitrarily named A and B. The propagation delay through A and B is different because the propagation path for A has a different loading than does the path for B. For example, in Figure 4.5(a), the source transistor MNA will see the channel resistance of MPB through ground, but MPB will not see this same resistance. MPB will instead have to drive through the channel resistance of MPA to discharge the output. The result of this imbalance is propagation skew between input A and input B, which could mean adjacent delay cells could have different propagation delays through the same exit NAND gate.

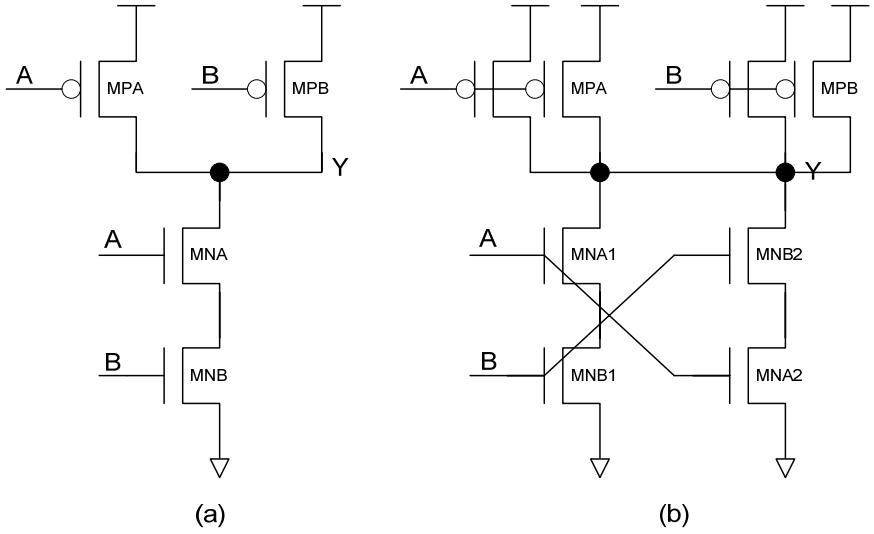

Figure 4.5 Schematic of (a) Typical NAND Gate and a (b) Balanced NAND Gate

To correct this imbalance a balanced NAND gate can be used as shown in Figure 4.5(b). In a balanced NAND the loading for terminal A and B are identical because there are two NMOS branches with the A and B inputs swapped. This guarantees that the propagation delay through A and B will see the same loading and therefore will have the same value.

As can be seen by Figure 4.4 the exit tree NAND gates have three inputs. The A input is used to enable the exit tree NAND and the B and C inputs are used for the exit path. Like the 2-input typical NAND, a 3-input typical NAND gate will also have a propagation delay offset between the A, B, and C terminals. To show the benefit of using an exit tree consisting of balanced NAND gates, a simulation was run using a modeled exit tree. As the exit path requires an exit tree with six tiers to decode 128 stages, a string of six 3-input NAND gates placed in series was used to model the exit tree. Since the enable is always on input A, only inputs B and C will have a time sensitive clock applied to them. For this reason, four strings of exit NAND's were placed: (1) a typical string

using only the B input, (2) a typical string using only the C input, (3) a balanced string using on the B input, and (4) a balanced string using only the C input.

Figure 4.6 shows the results of this simulation. The input clock, ClkInAll, is plotted along with the output of the four exit NAND strings. The blue, light blue, red, and green signals correspond to the typical B input, typical C input, balanced B input, and balanced C input exit NAND string, respectively. Notice that the typical C input exit NAND string has an extra 50 ps propagation delay from the other strings, which are practically identical to each other. This suggests that the propagation time through a typical NAND exit tree for a clock exit path that only passes through the C input will be 50 ps longer than the path that only passes through B input. For this reason, a balanced NAND exit tree is used for this DLL design.

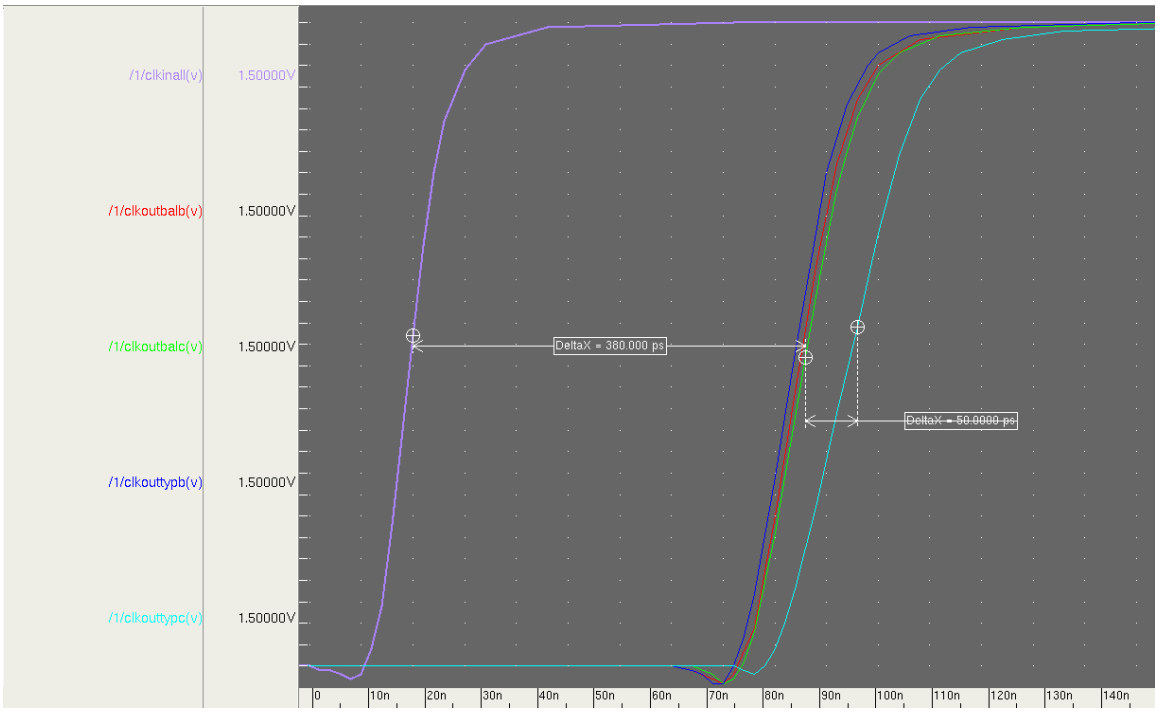

Figure 4.6 Simulation of Balanced vs. Typical NAND Exit Tree

4.4 Phase Combiner

Because the DCC will be providing an independent 180º phase, it is necessary to have a phase combine circuit. The purpose of the phase combine circuit is to mix the rising edge clock from the DLL and the falling edge clock from the DCC and create an output clock that is the combination of both. In the event that the DCC is disabled the phase combine circuit should simply pass the output of the DLL.

The schematic of the phase combine circuit used in this design is shown in Figure 4.7. DllOut0 and DllOut180 are the rising and falling edge outputs, respectively. DllOut0 and DllOut180 are sent through pulse generators that create rising and falling edge pulses. These pulses are provided to a set-reset latch, which will cause the output (DllOut) to rise upon a rising edge pulse and fall upon a falling edge pulse. Because the falling edge pulse has to toggle two NAND gates to reset DllOut, a mux is inserted into the rising edge pulse path. Inserting the additional mux in the set path matches the propagation delays for both the rising and falling edge pulses to DllOut.

When the DCC is disabled (i.e. when DccInitF is high) the pulse generators are bypassed and the DllOut0 and DllOut180 signals are simply passed to DllOut. DllOut0 and DllOut180 must be similarly delayed as if they were going through the pulse generators and phase combiner to prevent any clock skew between the DCC enabled and disabled cases.

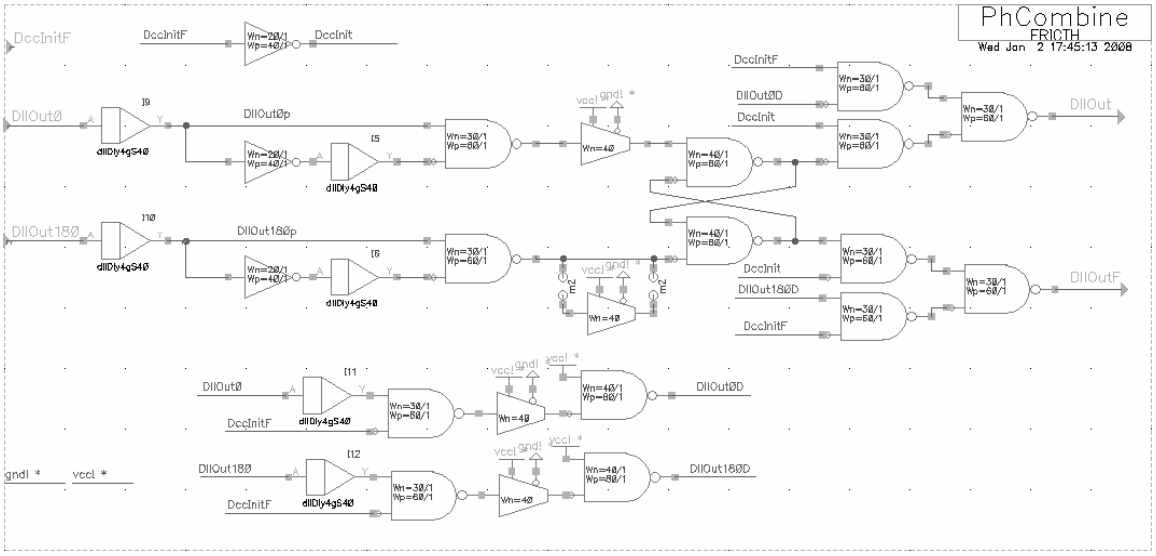

Figure 4.7 Phase Combine Schematic

Figure 4.8 shows a simulation of the phase combine circuit that demonstrates its operation. Because of the pulse generators, the duty cycle of DllOut0 and DllOut180 does not matter: only their rising edges do. There are approximately 500 ps of propagation delay for both DllOut0 and DllOut180. Notice that while DllOut0 and DllOut180 have non-ideal duty cycles, the phase combined outputs, DllOut and DllOutF, can have an ideal duty cycle if the rising edges of DllOut0 and DllOut180 are 180 degrees out of phase.

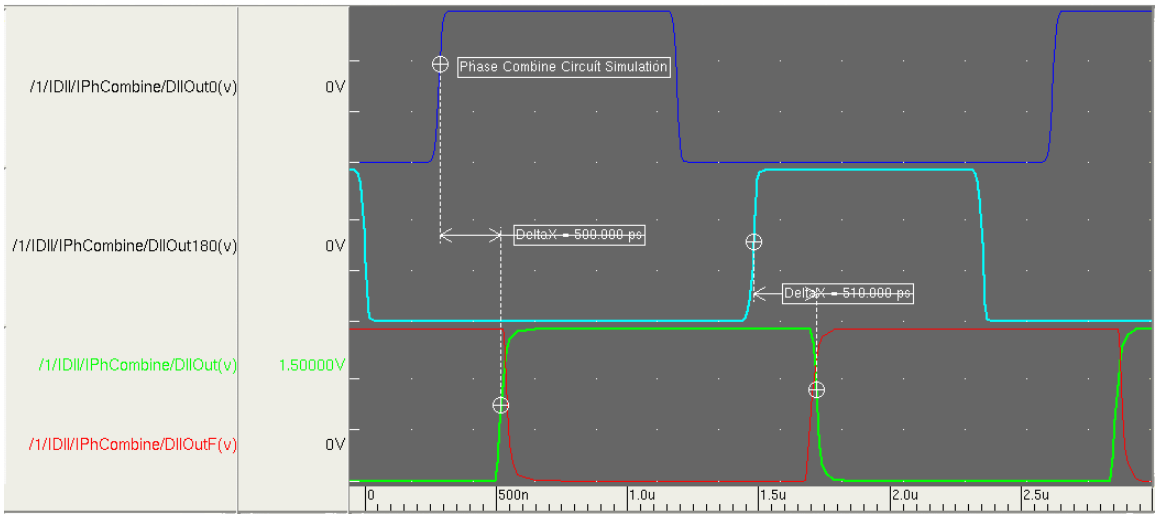

Figure 4.8 Phase Combine Simulation

When the DCC has not been initialized, the phase combine circuit operates in the bypass mode where DllOut0 and DllOut180 are passed to the output. Once the DCC initialization is complete, as indicated by the DccInitF signal, then the phase combine circuit will begin to combine phase. When DccInitF switches there will be a single cycle where the output clock duty cycle is interrupted, but this will only be temporary as the corrected duty cycle will continue to be phase combined from that time forward.

4.5 Shift Divider

In the IHCDL DCC it is critical that the 180º exit point be shifted exactly half as many times as the 0º exit point to ensure that the 0º and 180º phases are 180 degrees out of phase. Any two shifts of the 0º exit point whether shallower or deeper into the delay line must result in a shift in the same direction of the 180º exit point. During the DCC initialization the two exit points are typically shifted deeper into the delay line, but once the lock point is acquired sometimes a shallower lock point is required. For this purpose, a shift divider circuit was designed for this DCC.

Figure 4.9 shows a schematic of the coarse shift divider circuit. Any time a coarse shift clock is issued a pulse on CSclk occurs. This pulse is always passed to the 0º CSclk (CSclk0). For two CSclk pulses in the same phase direction a single CSclk180 pulse is generated. The CSclk pulse always propagates to the CSclk0 path, but only passes alternatively for the CSclk180 path given the phase doesn’t change between CSclk pulses.

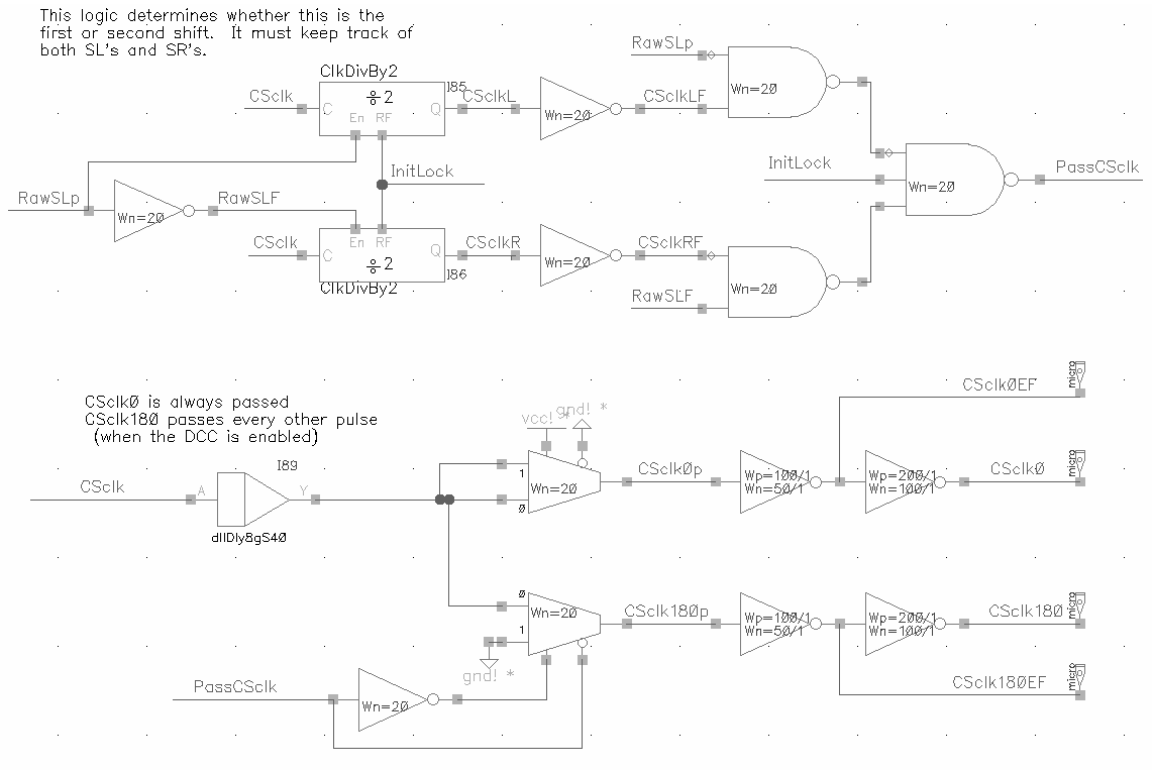

Figure 4.9 Shift Divider Schematic

The toggle (or divide by two) flip-flops (TFF) in the upper portion of the schematic store CSclk pulses for the Shift Left (SL) and Shift Right (SR) directions. Based on the phase (via the RawSL signal), the CSclk pulse is stored, which switches the output of the TFF. When a second CSclk pulse occurs in the same phase direction, the enable signal to the CSclk180 mux, PassCSclk, is changed to allow the CSclk pulse through. Otherwise, the CSclk pulse is blocked because CSclk180 is essentially shorted to ground.

Figure 4.10 shows the timing diagram for the operation of the shift divider. Notice that every other CSclk pulse is passed to CSclk180, but if the phase changes (i.e. RawSL changes state) any CSclk pulses in the opposite phase are still stored. For example, three CSclk pulses are issued when RawSL is high. This results in three CSclk0 pulses, but only one CSclk180 pulse. The third CSclk pulse is stored until the

next CSclk pulse when RawSL is high. When this happens a pulse on CSclk180 is immediately generated. The same argument holds for when RawSL is low.

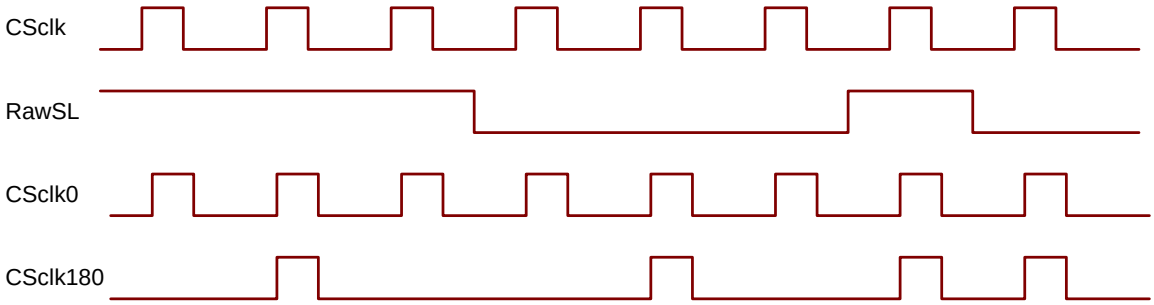

Figure 4.10 Timing Diagram for the Shift Divider

The same shift divider concept must be used for fine shifts or else error can accumulate between the 0º and 180 º phases once the fine delays are enabled. Figure 4.11 shows the schematic of the fine shift control circuit that includes a shift divider for the fine delay line. Now FSL and FSR (fine shift left and shift right decisions from the fine phase detector) determine the shift direction and pulses on FCSclk are counted. The FCSclk pulses are generated from the majority filter after the fine phase loop has been enabled.

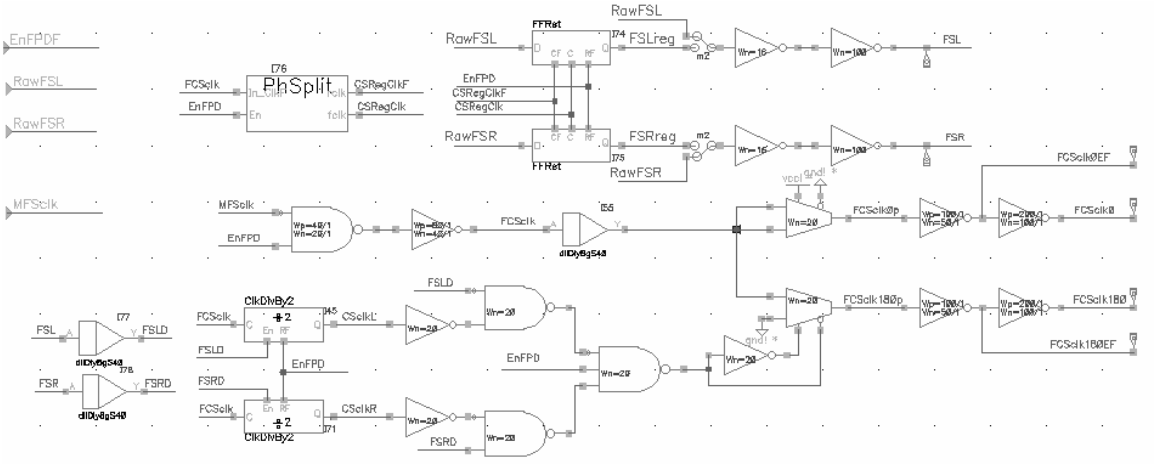

Figure 4.11 Fine Shift Divider

4.6 Initialization and Locking

Figure 4.12 shows a flow chart of the locking of the DLL and DCC. Upon the DLL reset, after a fixed number of cycles the measure initialization occurs, which places the DLL close to its lock point. Register-controlled operation resumes using the coarse phase detect to determine if the feedback clock (FbPD) arrives sooner or later than the reference clock (RefPD). The DLL is considered locked when three consecutive phase equal states are captured within the hysteresis of the coarse phase detector.

When the DCC is enabled the part is put into Force Shift Left mode, which forces the DLL to shift left until the 180 degree phase boundary is crossed. Until the DCC was initialized the 0º and 180º exit points where shifted identically. Because the DCC is enabled, the shift divider is only issuing one shift to the 180º shift register for every two shifts to the 0º shift register. When in force shift left mode a shift is issued every other cycle until the Ph180 phase boundary is crossed. When this happens the majority filter is used to filter the phase information by at a faster rate than at normal operation.

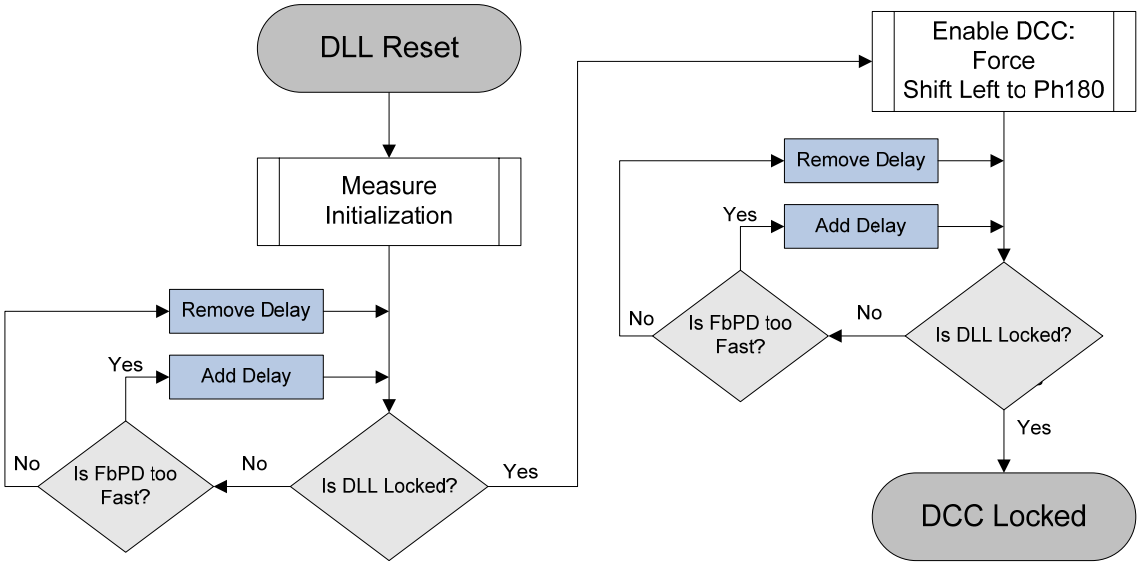

Figure 4.12 DLL and DCC Lock Flow Chart

As discussed in Section 2.8, in fast shift mode the majority filter can issue a shift every four cycles. The majority filter is placed into fast shift mode immediately after force shift left mode so that the DCC will find a lock sooner than if the majority filter were only allowing shifts every eight cycles. Once the phase detector detects three phase equal states between the reference clock and the 360º phase the DCC is consider locked and the fine phase loop is enabled.

When the fine phase loop is enabled, due to the coarse phase detect hysteresis, the DCC should be within one coarse unit delay of the lock point. The DCC is considered locked at this point even though the fine phase detect will continue to shift and eventually lock to a tighter resolution. Because of the fine phase detector's not having a phase equal state and for the sake of simplicity, phase equal detect logic for the fine phase loop was omitted from this design. Not having phase equal detect logic for this design will prevent any phase equal control circuitry from being switch when the DLL is operating in the fine phase loop.

Finally, it should be noted that 0º lock point is actually the 360º lock point. It is possible to phase combine the initial lock point (the true 0° lock point) and the 180º lock point and still create a DccOut signal that has a corrected duty cycle and DLL adjusted phase. The problem with doing this is that the initial lock point must be stored either in another shift register or a latch in one of the current shift registers. This was a design consideration that was intentionally omitted for the sake of simplicity. If the initial lock point were used for the 0º tap the forward path delay would be reduced by a full clock period and the power dissipation would be reduced because the delay cells between the 180º and 360º taps would not toggle.

CHAPTER 5 – INTEGRATED HCDL DCC PERFORMANCE

5.1 Duty Cycle Correction

Figure 5.1 shows how the HCDL DCC adjusts the 0° and 180° phases by plotting the output of coarse VDL. The first 20 cycles are required for the DLL to find a lock via measure initialization. Recall that both the 0° and 180° phases are locked at the same tap. This explains why the phase difference between 0° Out and 180° Out is practically nothing. After 20 cycles, the DCC begins to skew the 0° and 180° phases. The skew is linearly inserted until about the two thirds point where the shift filter slows the shift rate from every other cycle (in force shift left mode) to every four cycles. Once the DCC has finished adjusting the duty cycle, the register-controlled operation of the DLL resumes.

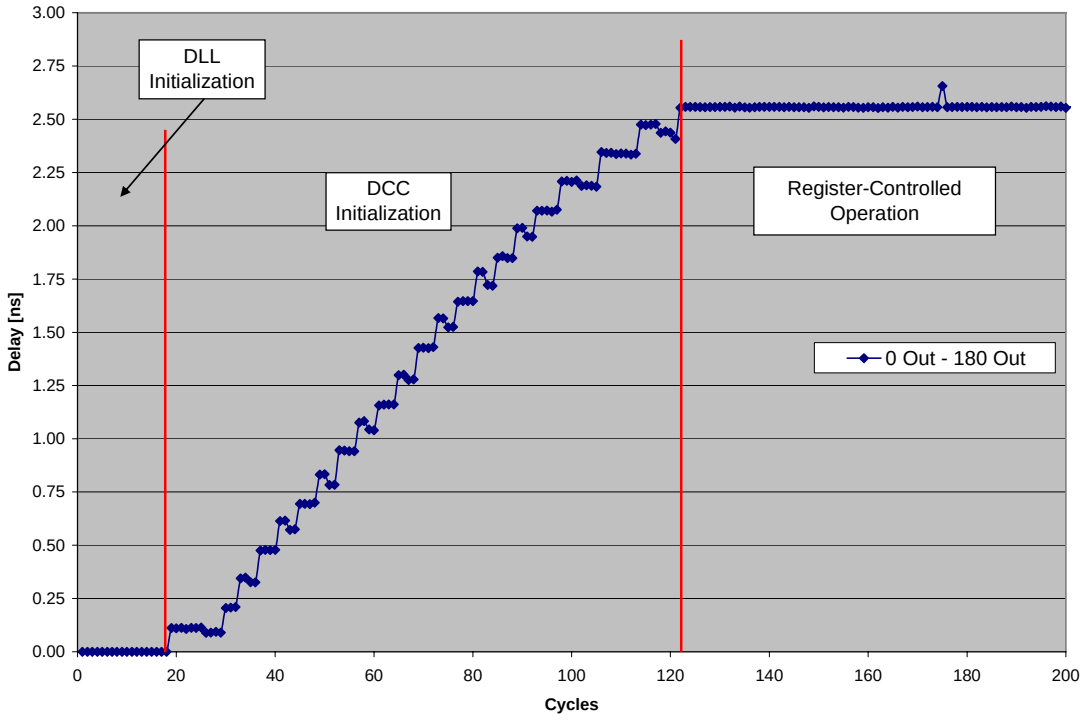

Figure 5.1 Phase Difference Between 0º Out and 180º Out during DCC Initialization (t_{CK} = 5 ns)

The clock period for this simulation is 5 ns, so the ideal phase difference between the 0° and 180° phases is 2.5 ns. Notice that the final phase is about a coarse unit delay from 2.5 ns. Fortunately, the output is not dependent on this resolution as this is only the output of the coarse VDL. The fine VDL will bring the 0° and 180° phases closer to the ideal 50% duty cycle phase. This can be seen in Figure 5.2, which is a magnified plot of Figure 5.1, but with the output of the fine VDL included. Notice that the coarse VDL can get the phase within about 60 ps of the ideal phase whereas the fine VDL improves this margin by almost a factor of two. The fine VDL gets the phase within 25 ps of the ideal phase difference of 2.5ns. For a 5 ns clock period, a 25 ps duty cycle phase offset is equivalent to a duty cycle offset of 0.5%, which is very acceptable if we are targeting +/- 5%.

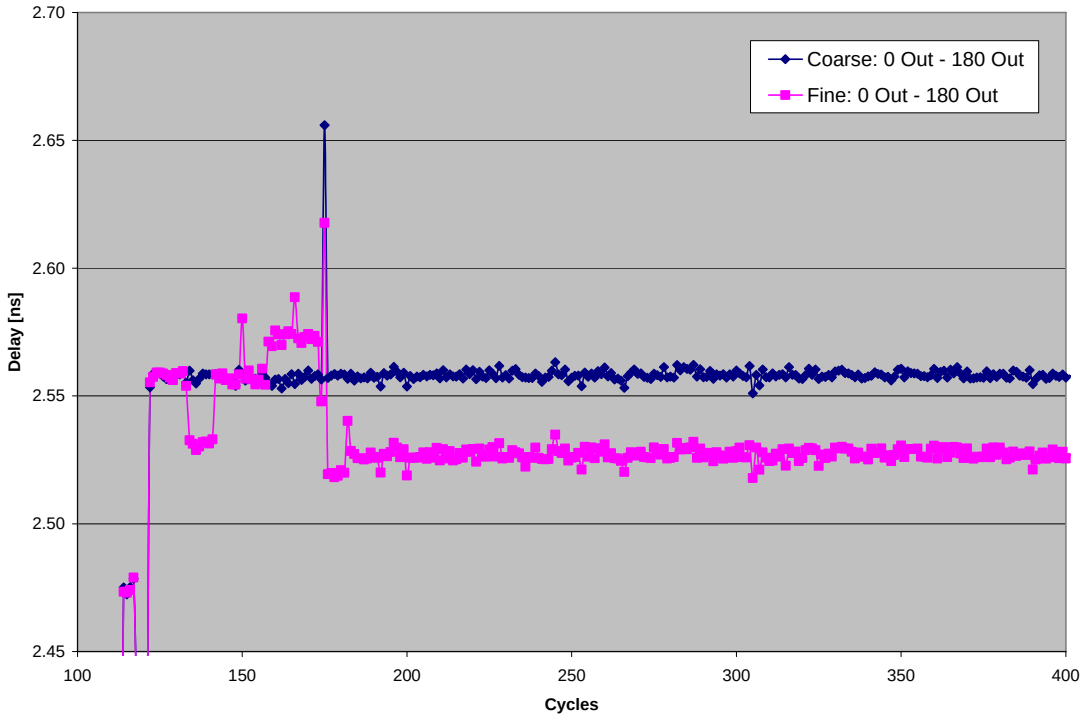

Figure 5.2 Coarse and Fine Phase Difference between 0° Out and 180° Out

While Figures 5.1 and 5.2 show the phase difference at the output of the coarse and fine VDL's, Figure 5.3 plots the output duty cycle (measured from the output data strobe, XDQS). Figure 5.3 also shows the output duty cycle for a range of clock periods from 3 to 10 ns instead of for just a single clock period at 5 ns. Unfortunately, the duty cycle offset is not as tight across this range of clock periods as the 5 ns case.

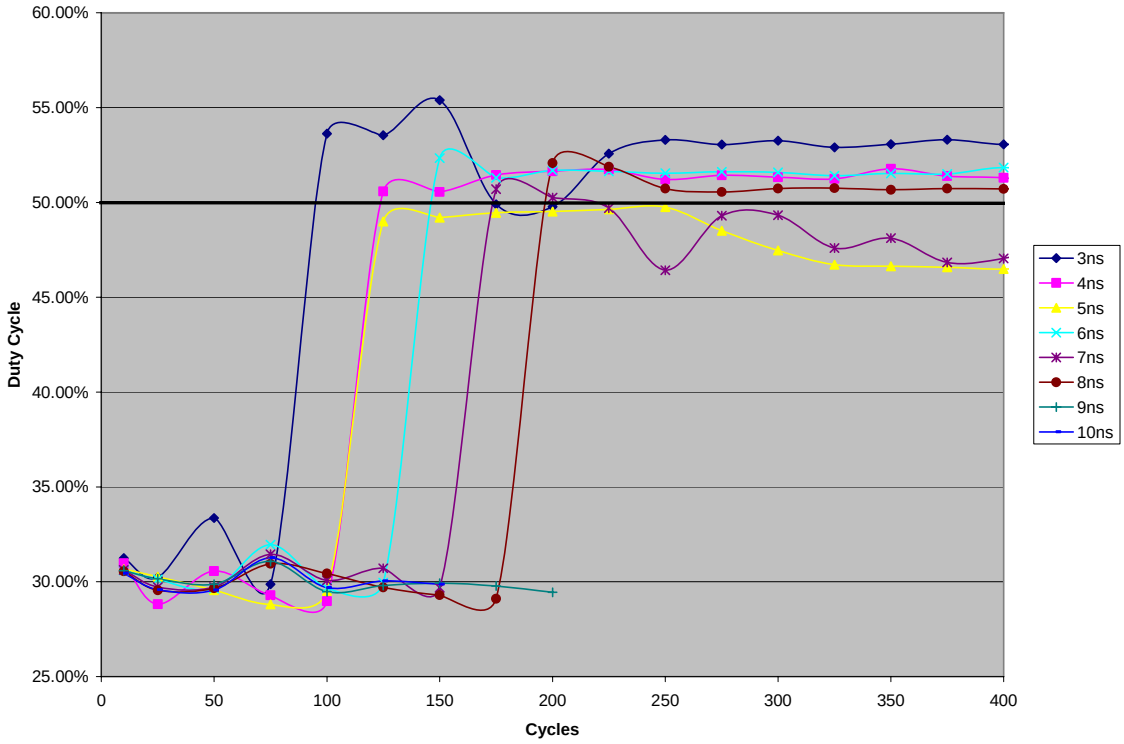

Figure 5.3 Output Duty Cycle vs. Clock Cycles for t_{CK} from 3 ns to 10 ns

As can be seen in Figure 5.3, as t_{CK} gets larger the number of cycles it takes before the duty cycle becomes corrected is pushed out. This push out is because the phase combine circuit waits until the DCC is initialized before switching from a non-corrected to a corrected duty cycle. Notice that there is also a wider spread of final duty cycle variation between the t_{CK}'s, but that all speeds fall within +/- 5% of 50% duty cycle. The clock periods of 9 and 10 ns do not have corrected duty cycle values because

the DCC runs out of delay line as the DLL is locked too deeply for an additional full t_{CK} worth of delay line to be added while finding the next 0º lock point. This data would indicate that the maximum clock period is between 8 and 9 ns (for 1.5V, 85C, and TT process models).

There is also a certain amount of phase drift for some of the t_{CK} plots in Figure 5.3. The 7ns plot, for example, drifts from 50% down to 46% back up to 49% and so on. This is a result of the fine shifting jittering around the lock point. Because the phase of the 0 º and 180 º phases are independent and both can be off by +/- 1 fine delay, there is a total phase offset of +/- 2 fine delay elements. This offset results in the drift of the duty cycle across simulation time.

5.2 Lock Time

Because the DCC shifts the 0º phase a full clock period deeper into the delay line, the initialization time is extended when the DCC is enabled. As t_{CK} is increased so is the time that it takes for the DLL to reacquire a lock on the 0º phase. The DLL is considered locked when the fine phase loop is enabled because the DLL is out phase by no more than one coarse delay unit at the point the fine phase loop is enabled. Table 5.1 shows the lock time in cycles versus the clock period for the IHCDL DCC.

Notice that for each 1 ns increase in t_{CK} there is approximately a 20 cycle increase to the lock time. There is slight lock time variability between clock speeds because of the hysteresis of the coarse phase detector and the requirement of three consecutive PhEQ commands to determine a lock. Again 9 and 10 ns clock periods cannot find a lock point because there are not enough delay cells for the DCC to insert a full t_{CK} worth of delay cells in the delay line.

tCK [ns]	Cycles to Lock
3	83
4	103
5	125
6	142
7	167
8	182
9	No Lock
10	No Lock

Table 5.1 Clock Period vs. Lock Time

The DDR2 JEDEC specifications require that a DLL find a lock within 200 cycles of the DLL reset for clock periods up to 8 ns. The addition of the IHCDL DCC function to this DLL still allows it to pass this specification for clock periods less than or equal to 8 ns. This makes this DCC design a truly viable candidate for use on a DDR2 DRAM.

5.3 Duty Cycle Range

Possibly the most critical characteristic of a DCC is how much duty cycle can it actually correct. For characterization purposes it is useful to examine the duty cycle transfer curve, which is simply of a plot of the input versus output duty cycle. Figure 5.4 plots the duty cycle transfer curve of the IHCDL DCC for a clock period of 5 ns. The blue line curve plots the duty cycle transfer curve when the DCC is enabled and the pink line plots the transfer curve for the DCC disabled case.

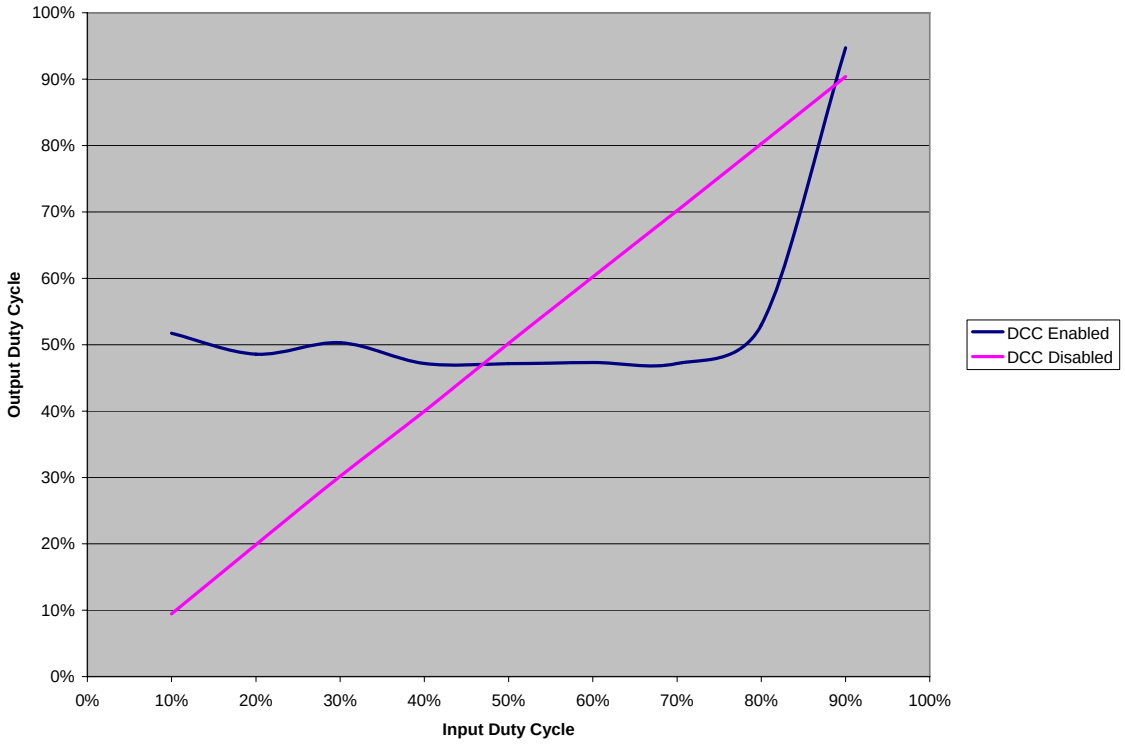

Figure 5.4 Input vs. Output Duty Cycle with DCC Enabled and Disabled (t_{CK} = 5 ns)

Notice that for the DCC enabled case from an input duty cycle of 10% to 80% the output duty cycle is relatively flat and lies between 55% and 45% duty cycle. Over this input duty cycle range the duty cycle transfer curve is nearly flat. A flat duty cycle transfer curve suggests that the output duty cycle is independent of the input duty. This independence is indeed the case for this DCC as it is not dependent upon the falling edge of the input clock for any calculation.

Conversely, the transfer curve for the DCC disabled case is linear instead of being flat. This dependence means that when the DCC is disabled the output duty cycle is a direct function of the output duty cycle. In this case the input to output duty cycle is nearly a one to one relationship so the transfer curve is practically the y = x line.

The DDR2 JEDEC specifies that the input clock duty cycle can vary from 45% to 55%. Figure 5.5 shows a magnified version of Figure 5.4 with these specification limits

plotted in red for both the input and output duty cycle. Notice that the input duty cycle range for the DCC enabled case is wider than 30% to 70% (a range greater than 40%) while the input duty cycle range for the DCC disabled case is equal to 45% to 55% (a range of 10%). From Figure 5.4 the input duty cycle range for the DCC enabled case is approximately 10% to 80% or 70%.

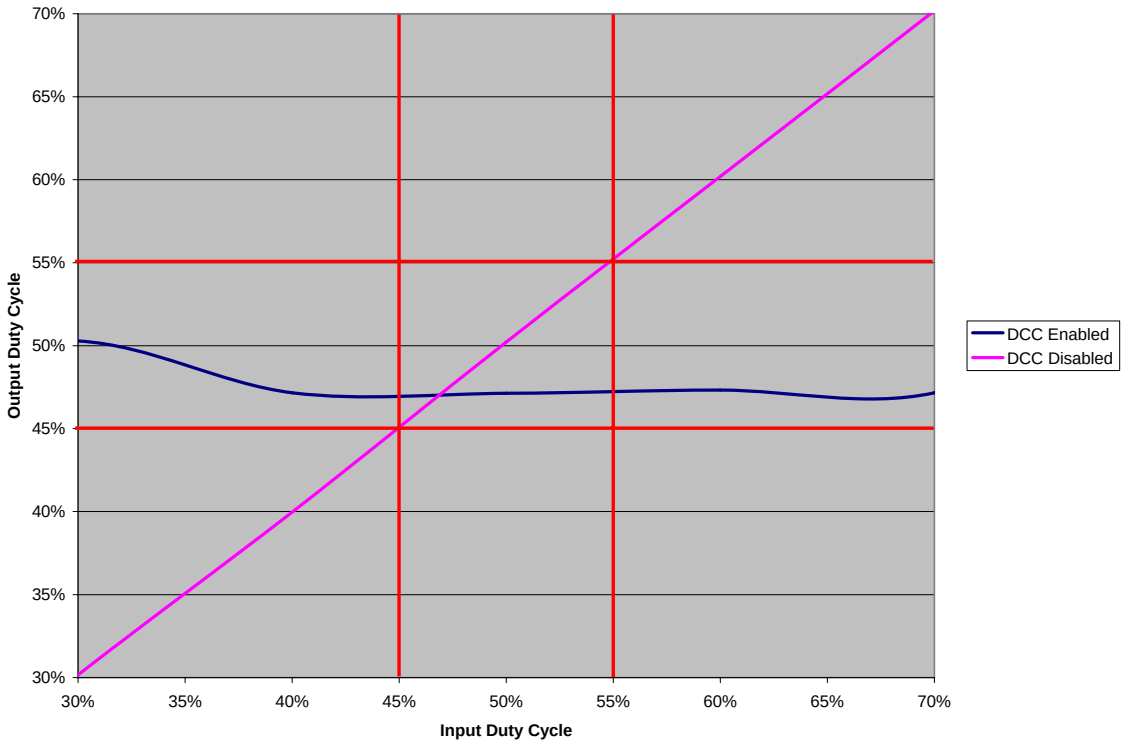

Figure 5.5 Input vs. Output Duty Cycle Plot with Spec Limits

The previous two figures demonstrate that the IHCDL DCC provides a significant improvement to the input duty cycle range. This improvement is because the IHCDL DCC determines where the falling edge of the output clock should be independent of where the input falling edge occurs. The DCC only needs a clock signal with two consistent rising edges spaced apart by the clock period to create the 0º and 180º phases and (after phase combination) an output clock with a corrected duty cycle. This falling edge independence is the most attractive benefit for using a DCC as it allows a

customer's application to supply a non-ideal duty to the part and still receive an output duty cycle close to 50%.

5.4 Duty Cycle Jitter

Table 5.2 shows the duty cycle jitter versus PVT for the IHCDL DCC at 5 ns clock period. For process, three models were used: fast n-channel and fast p-channel (FF), typical n-channel and typical p-channel (TT), and slow n-channel and slow p-channel. The three PVT points were taken such that the fastest (FF, 1.7 V, 0C), slowest (SS, 1.3 V, 130C), and most typical (TT, 1.5 V, 85C) conditions were compared. The results show that the fast corner has the best duty cycle jitter variation with 3.59%. The slow corner has the worst duty cycle variation with 9.54%. Finally, the typical corner has a duty cycle jitter value that is between the fast and slow corners, but nearer to the fast corner value with 4.35%.

PVT			Duty Cycle Jitter
FF	1.7V	0C	3.59%
TT	1.5V	85C	4.35%
SS	1.3V	130C	9.54%

Table 5.2 Duty Cycle Jitter vs. PVT (for t_{CK} = 5 ns)

The slow corner could be out of spec if the IHCDL DCC is not perfectly centered at 50% duty cycle. The DCC will probably not center the duty cycle perfectly so the IHCDL DCC would fail the JEDEC DDR2 specification of +/- 5% duty cycle across PVT. Thus if the IHCDL DCC were to be used in a design, the duty cycle jitter would have to be a critical design focus.

5.5 Power

Because the DCC requires additional logic than the DLL by itself to function, there is an additional current demand for this operation. The result is additional power consumption when the DCC is enabled. Table 5.2 summarizes the current demand for the IHCDL for both the DCC enabled and disabled cases. The average current was measured over 600 cycles including the initialization of DCC. The average power can be found by multiplying by the supply voltage of 1.5 V. At 3 ns, the worst case for power consumption, the DCC only draws an additional 750 μA. A 750 μA additional current draw is an acceptable cost for the benefit duty cycle correction especially given that the DLL itself draws almost five times that much current at the same t_{CK}.

tCK [ns]	Dcc Enabled Average Current [mA]	Dcc Disabled Average Current [mA]	Delta [mA]
3	5.116	4.360	0.756
4	3.796	3.161	0.635
5	3.312	2.610	0.702
6	2.971	2.359	0.612
7	2.734	2.098	0.636
8	2.540	1.922	0.618
9	2.238	1.794	0.444
10	2.014	1.694	0.320

Table 5.3 Clock Period vs. Average Current (Vdd = 1.5V)

5.6 Response to Voltage Supply Variation

Figure 5.6 shows the output duty cycle response to an instantaneous voltage change for two conditions. The first (blue line) is a voltage bump at 750 cycles from 1.5 V up to 1.7 V. The second (pink line) represents the voltage bump from 1.5 V down to 1.3 V. Notice that in both cases after the shifting has stabilized the value is no longer equal to the 50% point of 2.5 ns. This deviation from the nominal correction can be explained by referencing Figure 3.8 again. Because of the positive voltage bump (blue

line), the 0° lock point must be shifted deeper in the delay line to keep the reference and feedback clocks in phase. Naturally, the 180° lock point, due to the shift divider, will move half the distance deeper into the delay line as the 0° lock point. The problem is that because the supply voltage has changed the value of the initial lock point would have also changed. Maintaining the precise difference between the initial lock point and N + 1 lock point is essential for accurate calculation of the 180° tap point. When the voltage supply changes dramatically (like in this example), the full t_{CK} relationship between the initial lock point and the N + 1 lock point is lost. Consequently, there is a duty cycle offset as a result of a voltage bump.

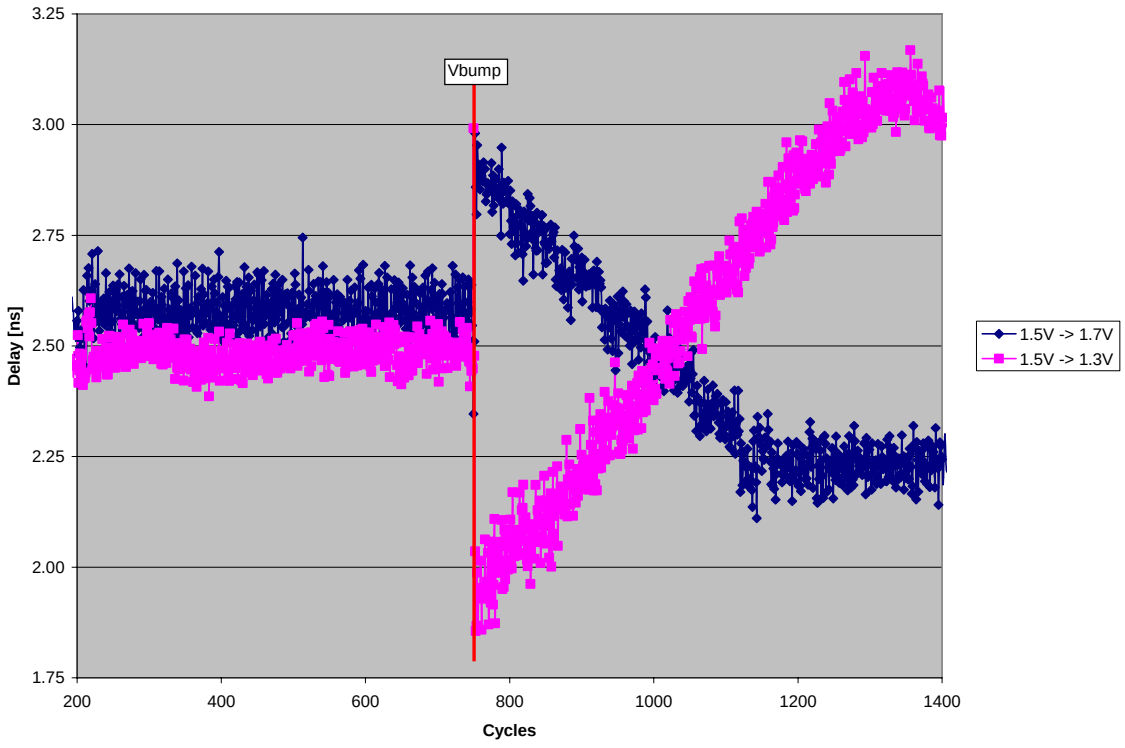

Figure 5.6 Output Duty Cycle Response to an Instantaneous Change in Voltage

It is a characteristic of this design that the relationship between the N (initial) lock point and the N + 1 lock point must be maintained for proper duty cycle correction. This relationship is the detriment of using a single delay line for the duty cycle correction. The IHCDL DCC requires a relative stable voltage supply to provide an accurate corrected duty cycle. This DCC, however, can recover from a voltage bump if the original voltage is reacquired. Figure 5.7 shows the output duty cycle movement for a voltage bump from 1.5 V up to 1.7 V and back down to 1.5 V. While there will be a considerable amount of duty cycle jitter as a result of the voltage supply variation, eventually the DCC will recover the duty cycle to its proper value.

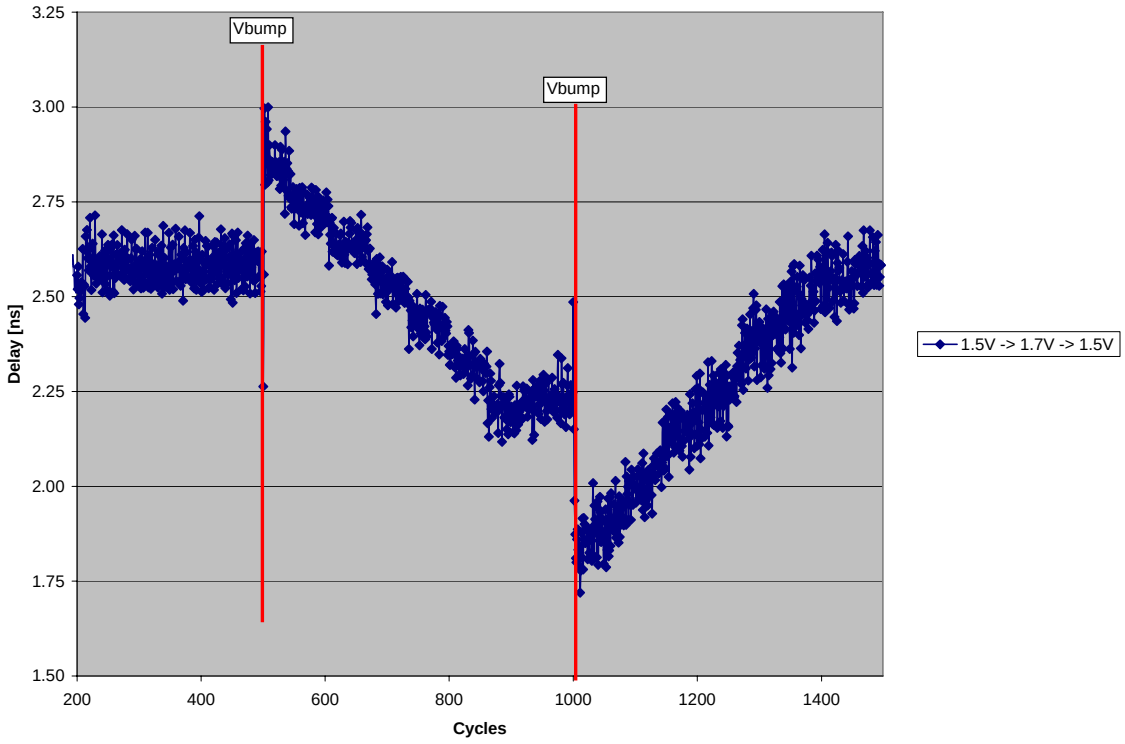

Figure 5.7 Output Duty Cycle Response to a Voltage Change from 1.5 V to 1.7 V to 1.5 V

While Figures 5.6 and 5.7 show the output duty cycle response to an instantaneous voltage supply change, Figure 5.8 shows the duty cycle response to a 100 mV peak sinusoidal variation on the voltage supply. The responses for 1 MHz and 500

kHz are plotted on Figure 5.8. Both frequencies show that the output duty cycle is directly dependent upon the voltage supply variation, which is the same result of the instantaneous voltage supply change.

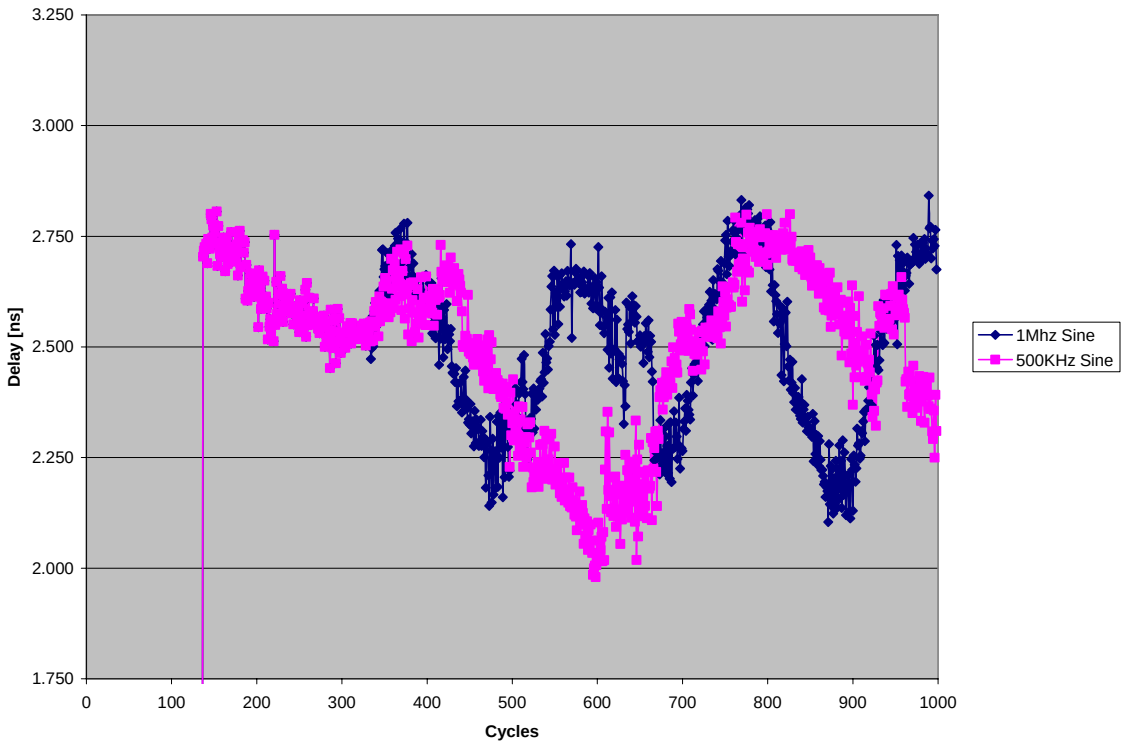

Figure 5.8 Output Duty Cycle for +/-100 mV Sinusoidal Voltage Supply

CHAPTER 6 – CONCLUSIONS

6.1 Conclusions

The IHCDL DCC design presented in this thesis has a wide duty cycle range, good duty cycle correction across a range of typical clock periods, minimal impact to DLL lock time, and reasonable power consumption. Because of the IHCDL DCC's use of only a single delay line to generate the 0° and 180° phases, this DCC offers a significant layout area improvement over the other DCC designs considered. Furthermore, a closed loop duty cycle detection method is employed that reduces the chance for the DCC to accumulate error.

This design has problems with duty cycle jitter across PVT and due to power supply variations. PVT jitter issues are primarily related to the method that the fine delay line was implemented how it interacts with the coarse delay line. Power supply induced duty cycle jitter is a consequence of using the same delay line for the 0° and 180° phases and the jitter unavoidable with this DCC topology.

In future designs, the DCC presented in this thesis could be substantially improved by two methods. First, the implementation of a wider range linear phase mixer would decrease the minimum fine delay step and, when matched properly to the unit coarse delay cell, would reduce the jitter that occurs when the end of the fine VDL has been reached and a coarse shift is needed. Second, the design of a more robust shift divider would more effectively handle the transition between fine to coarse shifting and would prevent any duty cycle offset. Undoubtedly, the design of the shift divider is the most critical design consideration for a successful implementation of the IHCDL DCC.

CHAPTER 7 – REFERENCES

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

[2] 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, 1Gbit, Double-Data-Rate Synchronous DRAM," *IEEE J. Solid-State Circuits*, vol. 34, pp. 1589-1599, Nov. 1999.

[3] 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.

[4] Y.-M. Wang, and J.-S.Wang, "An All-Digital 50% Duty-Cycle Corrector," *International Symp. on Circuits and Systems*, 2004.

[5] C. Yoo, C. Jeong, J. Kih, "Open-loop full-digital duty cycle correction circuit," *ELECTRONICS LETTERS*, vol. 41, no. 11, May 2005.

[6] Y. Jo, S. Kim, and S. Kim, "A Mixed-Structure Delay Locked-Loop With Wide Range and Fast Locking," *International Symp. on Circuits and Systems*, 2006.

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

[8] J. Kwak, C.-K. Kwon, K.-W. Kim, S.-H. Lee, and J.-S. Kih, "A Low Cost High Performance Register-Controlled Digital DLL for 1Gbps x32 DDR2 SDRAM," *Symp. on VLSI Circuits Digest of Technical Papers*, 2003.

[9] M. Shoji, "Elimination of Process-Dependent of Clock Skew in CMOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 875-880, Oct. 1986.

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

[11] 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.

[12] R. J. Baker, *CMOS: Circuit Design, Layout, and Simulation*, IEEE Press, 2008. ISBN 978-0-470-22941-5.

[13] A. Hatakeyama, H. Mochizuki, T. Aikawa, M. Takita, Y. Ishii, H. Tsuboi, S. Fujioka, S. Yamaguchi, M. Koga, Y. Serizawa, K. Nishimura, K. Kawabata, Y. Okajima, M. Kawano, H. Kojima, K. Mizutani, T. Anezaki, M. Hasegawa, and M Taguchi, "A 256-Mb SDRAM Using a Register-Controlled Digital DLL," *IEEE J. Solid-State Circuits*, vol 32, pp 1728-1734, Nov. 1997.

[14] T. Gomm, K. Y. Kim, S. Smith, J. Kwak: Inventors; Micron Technology Inc., assignee. 2006 Mar 3. Method, Circuit, and System for Detecting a Locked State of a Clock Synchronization Circuit. U.S. Patent 20070205817.

[15] J. Miller: inventor; Micron Technology Inc., assignee. 2003 Oct. 17. Low Pass Filters in DLL Circuits. U.S. Patent 6917230.

[16] J. Miller: inventor; Micron Technology Inc., assignee. 2001 Nov. 30. Low Pass Filters in DLL Circuits. U.S. Patent 6664830.

[17] R. J. Baker, *CMOS: Mixed-Signal Design*, IEEE Press, 2002. ISBN 0471227544.

[18] T. Matano, Y. Takai, T. Takahashi, Y. Sakito, I. Fugii, Y. Takaishi, H. Fugisawa, S. Kubouchi, S. Narui, K. Arai, M. Morino, M. Nakamura, S. Miyatake, T. Sekiguchi, and K. Koyama, "A 1-Gb/s/pin 512-Mb DDRII SDRAM Using a Digital DLL and a Slew-Rate-Controlled Output Buffer," *IEEE J. Solid-State Circuits*, vol. 38, pp 762-768, May 2003.

[19] J. Kwak: inventor; Micron Technology Inc., assignee. 2005 Oct. 14. Low Pass Filters in DLL Circuits. U.S. Patent 0070086267.

[20] T. Gomm: inventor; Micron Technology Inc., assignee. 2006 Jan 27. Duty cycle error calculation circuit for a clock generator having a delay locked loop and duty cycle correction circuit. U.S. Patent 20070176659.

CHAPTER 8 – APPENDIX

8.1 Additional Schematics

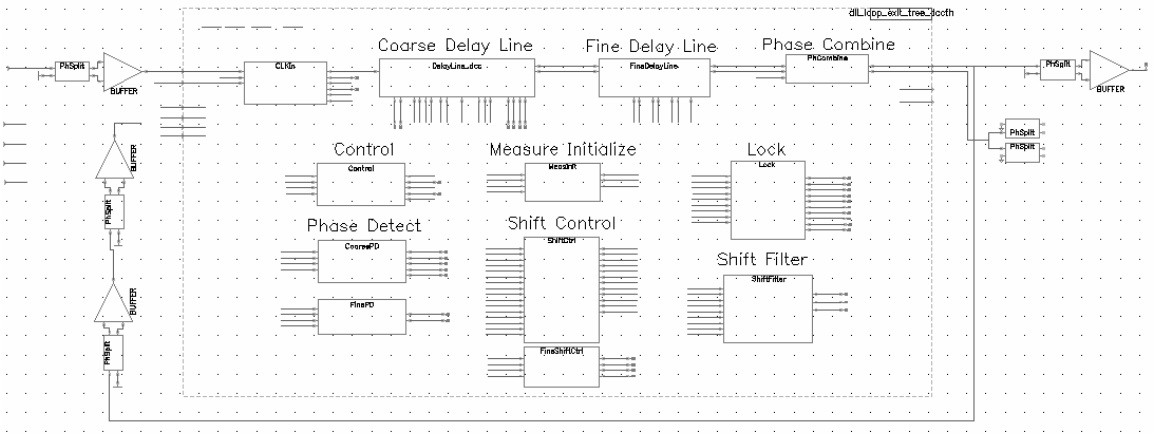

Figure 8.1 Top Level DLL/DCC Schematic

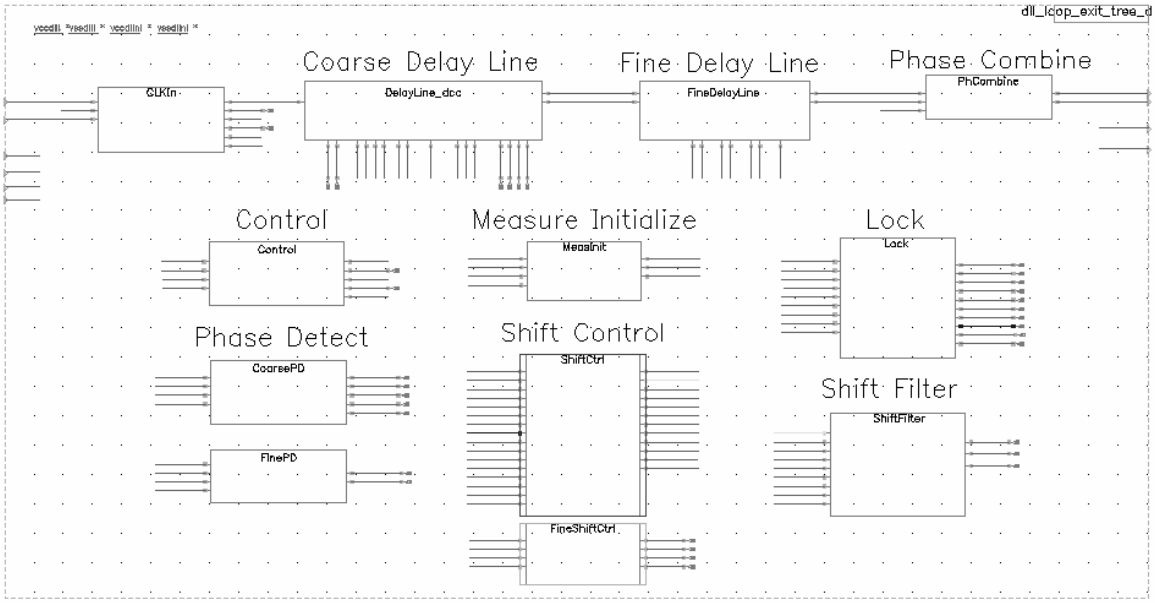

Figure 8.2 Schematic of DLL/DCC (without feedback)

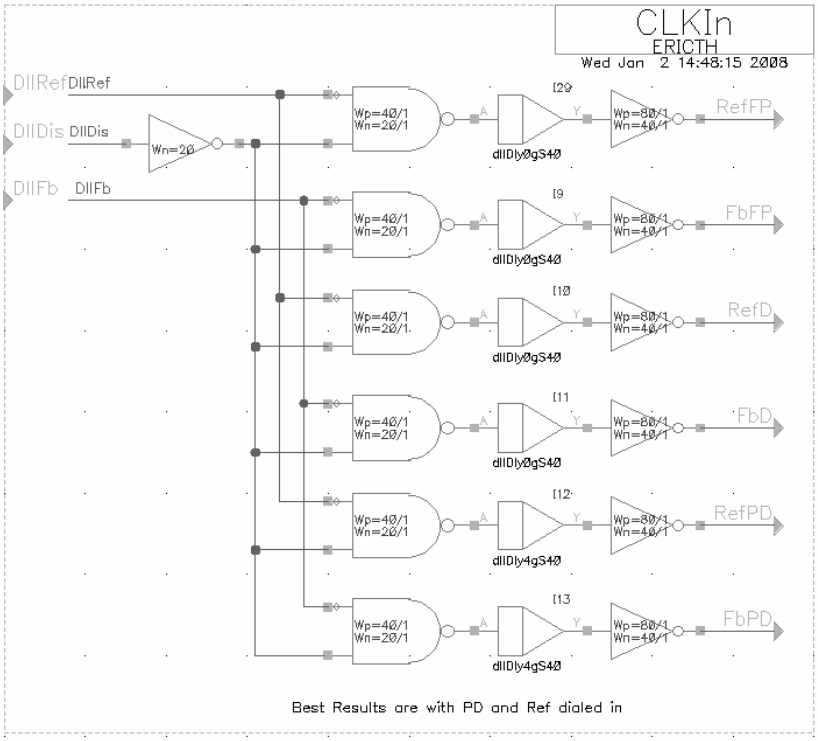

Figure 8.3 ClkIn Schematic

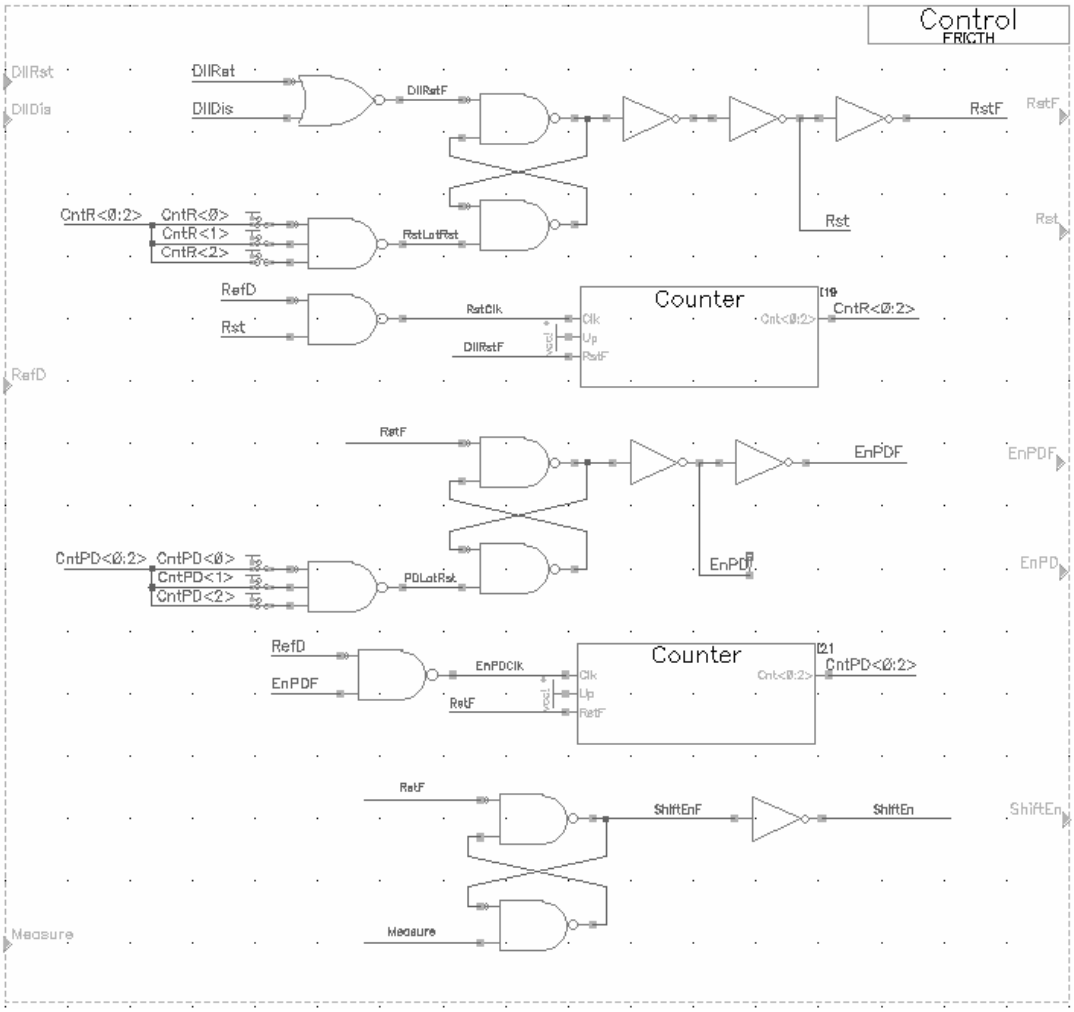

Figure 8.4 Control Schematic

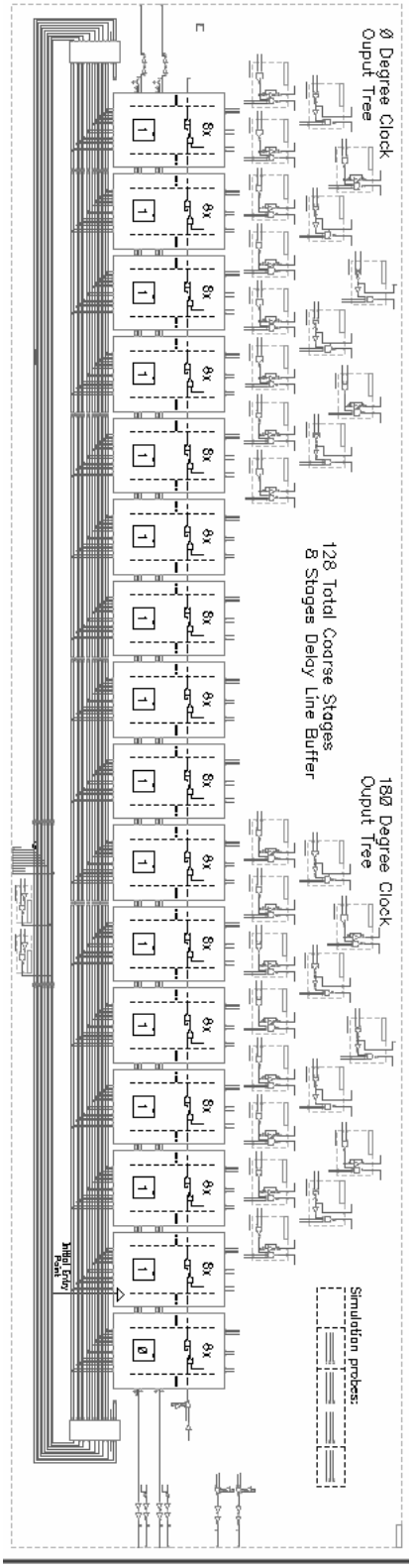

Figure 8.5 Schematic of Delay Line

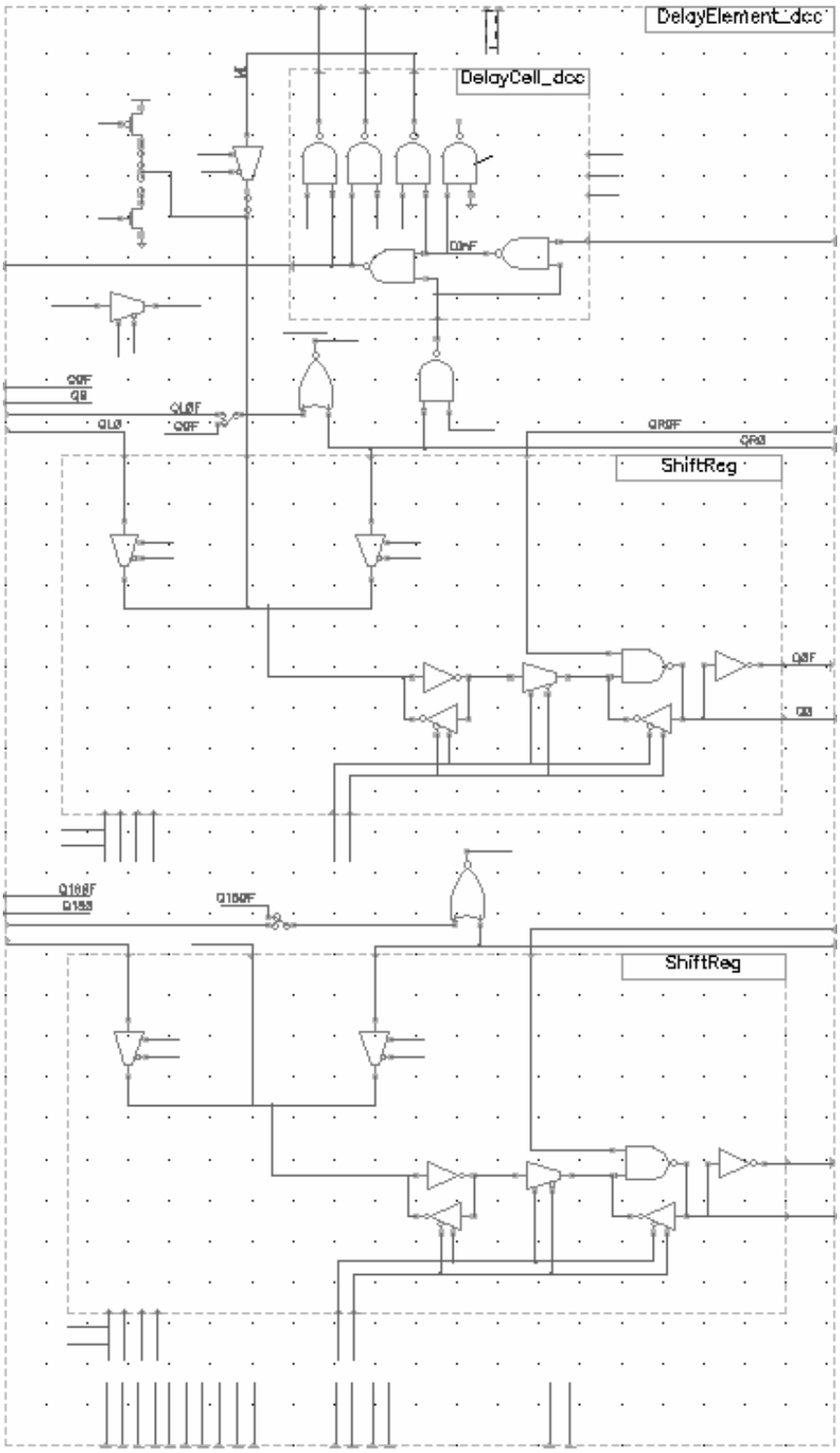

Figure 8.6 DCC Delay Element Schematic

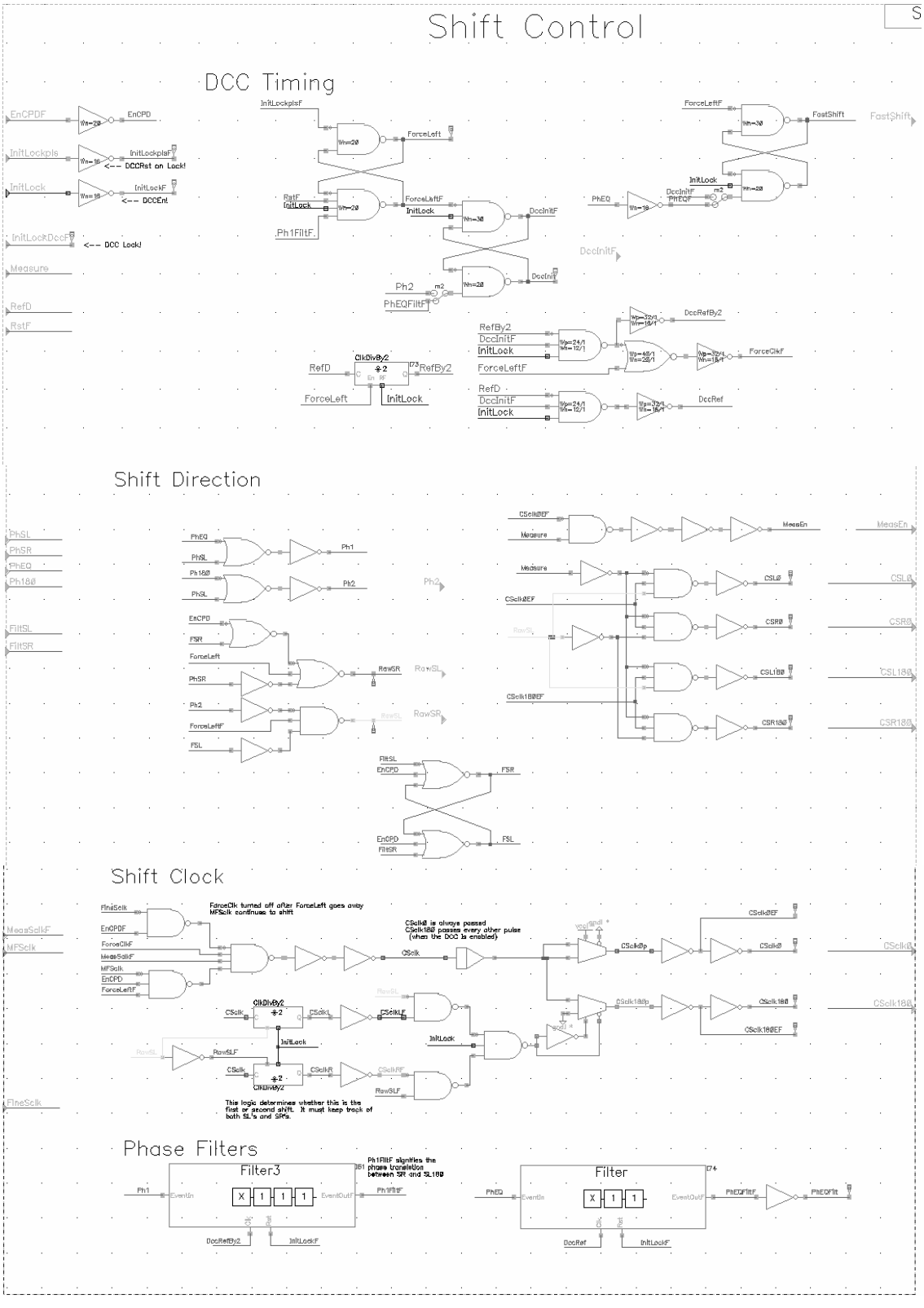

Figure 8.7 Schematic of Shift Control

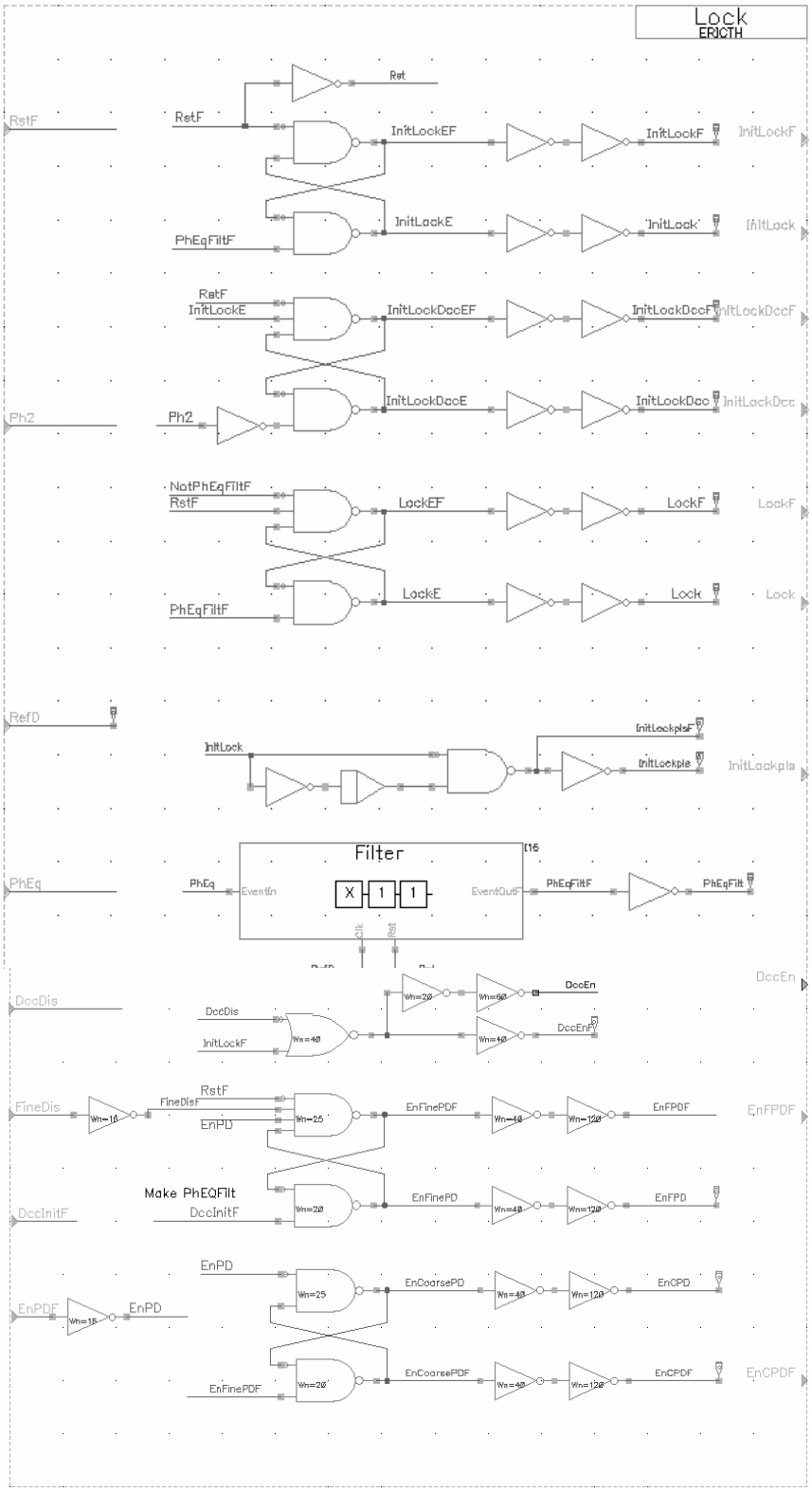

Figure 8.8 Schematic of Lock