6

Digital Delay Lock Techniques

Thucydides Xanthopoulos

Cavium Networks

6.1 Introduction

Digital delay locked loops are highly prevalent in integrated systems. They are essentially delay lines under feedback control that can generate derived clocks based on an input reference. Applications include clock distribution, I/O interfaces, clock generation, and frequency multiplication. Digital delay locked loops also have time-to-digital conversion properties and can be used in monitoring and sensing applications.

While DLLs can be designed with digital-only methods, their design involves direct manipulation of clock signals. Therefore, additional techniques are involved as opposed to standard custom digital datapath design. This chapter presents an identification of all essential digital delay locked loop components and addresses relevant design aspects for each part. It concludes with global design issues and an overview of advanced applications.

6.2 What Constitutes a Digital Delay Locked Loop?

The digital delay locked loop (DLL henceforth) is a simple closed loop system that is capable of generating a clock signal that has a precise phase relationship with an input reference clock. Because of feedback, this phase relationship tracks across input frequencies, process, voltage, and temperature. The accuracy of the phase relationship between input and output clocks depends on DLL design parameters, process mismatch characteristics, and on deterministic noise sources such as independent supply noise and forms of coupling.

Digital DLLs can easily be unconditionally stable and are analyzed in the time domain. Because of their all digital nature, they can be ported across process nodes, can be simulated using fast digital simulators, and can be easily monitored and characterized in silicon.

Figure 6.1 shows a simplified DLL block diagram, which identifies the three main system components: The phase detector, the control block, and the delay line.

T. Xanthopoulos (ed.), *Clocking in Modern VLSI Systems*, Integrated Circuits and Systems, 183
DOI 10.1007/978-1-4419-0261-0_6,

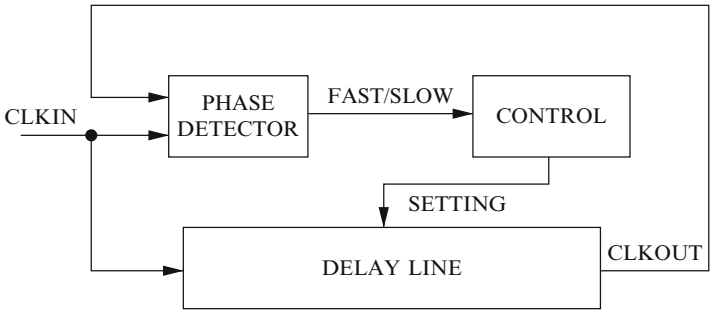


Fig. 6.1. Generalized DLL block diagram

The type of phase detector that will concern us in this chapter has a single bit output that only changes on the positive edge of the reference clock. It is commonly referred to as "bang–bang" in the literature. Such a phase detector can be thought of as a system that has the transfer function depicted in Fig. 6.2. It compares the phase difference between two clocks and outputs a single logic value indicating which clock is ahead in time. It can be thought of as a flip-flop that has the reference clock (CLKIN) as its clock input and the controlled clock (CLKOUT) as the data input. When the flop evaluates to a logic 1, it means the controlled clock is faster than the reference. On the other hand, when the controlled clock is slower than the reference, the flop will evaluate to logic 0. This behavior can be guaranteed as long as the data input is faster than the clock input at least by the flop setup time (T_s in Fig. 6.2) or slower at least by the flop hold time (T_h in Fig. 6.2). If the timing between the two clocks falls within the gray area of Fig. 6.2, the phase detector behavior is not defined and the output can be either a logic 0, a logic 1 or potentially a metastable value. The width of the gray rectangle ($T_s + T_h$) is the primary figure of merit of phase detectors and one of the main DLL design parameters. It is called the phase detector *sampling window* d_{sw} or *dead zone*. It affects the phase locking accuracy of the entire system in addition to other important specifications. Phase detectors will be discussed in Sect. 6.4.

The most design intensive component of the DLL is the Digitally Controlled Delay Line (DCDL). A sample transfer function is shown in Fig. 6.3. A DCDL is a combinational circuit that delays its input by an open loop value that typically has a monotonic relationship with the digital setting input. Such delay value is not precisely defined and is subject to process, voltage, and temperature conditions. A DCDL is primarily characterized by three design parameters: its minimum delay D_{min} (delay value at setting 0), its maximum delay D_{max} (delay value at the maximum setting $N-1$), and its resolution d_r (incremental delay per setting). The dynamic range is defined as $D_{max} - D_{min}$ and is directly related to the capability of the overall DLL to track significant PVT variations or work with an extended range of input clock frequencies. The resolution d_r affects the DLL accuracy along with d_{sw}. DCDL design will be discussed in Sect. 6.5.

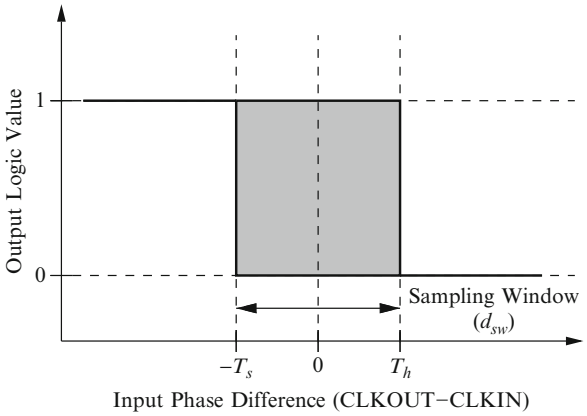


Fig. 6.2. Phase detector transfer function

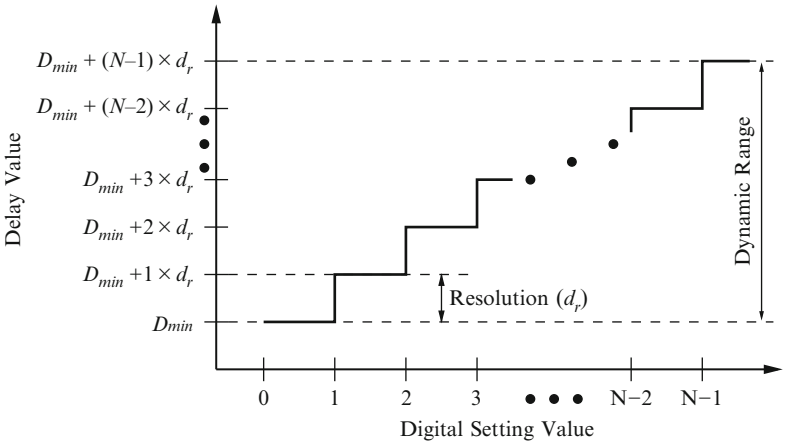


Fig. 6.3. Digitally controlled delay line transfer function

The final building block of Fig. 6.1 is the control module. The control block increases and decreases the DCDL settings based on the output of the phase detector. In its simplest form, it is an up/down counter controlled by the phase detector. In its most general form, it is a finite state machine (FSM) that controls the DCDL settings based on the output of the phase detector and internal state. The inclusion of additional state information can support more complex behavior and extend the DLL capabilities. Control structures will be addressed in Sect. 6.6.

The combination of a phase detector, a delay line, and a control block produces a simple and useful feedback system that can find multiple applications in modern systems-on-a-chip. The DLL of Fig. 6.1 will adjust its delay line until CLKIN and

CLKOUT are matched in phase. At this point, the DLL has *locked*, and the delay through its DCDL is one CLKIN period (or potentially an integral multiple of input clock periods).

6.3 An Overview of DLL Applications

The average modern microprocessor contains multiple digital delay locked loops embedded in various subsystems. Figure 6.4 demonstrates different uses of a basic DLL structure.

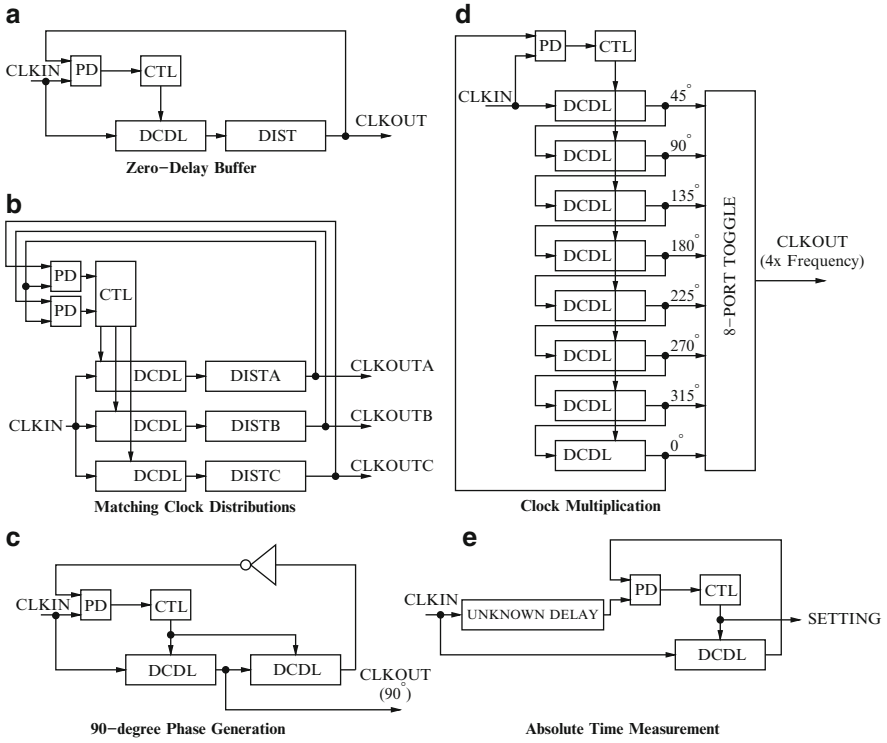


Fig. 6.4. DLL applications

The vast majority of DLL applications are related to clocking. Figure 6.4a demonstrates *zero-delay buffering*. Such a topology is well suited for synchronous I/O interfaces (e.g., PCI/PCI-X). A common clock (CLKIN) is being distributed to multiple bus end points. Each end point is buffering and distributing it to a number of flip-flops. A DLL in the loop ensures that the buffered clock version (CLKOUT) is phase-locked to the master interface clock (CLKIN).

Figure 6.4b shows an application where three separate clock domains are synchronized. Each domain has a separate clock distribution (DISTA, DISTB, DISTC) involving multiple buffer stages. Open loop matching is infeasible in the presence of PVT and random variations. A DLL-controlled delay line at the root of each distribution can guarantee phase matching among all three clocks. The DLL controller is more complex than the one shown in Fig. 6.1, and it uses information from two separate phase detectors. First, CLKOUTA and CLKOUTB are phase locked. As soon as this happens, the controller locks CLKOUTC to CLKOUTA. This scheme is extensible to multiple clocks.

Quadrature clock generation is another application suitable for a DLL. Such a topology is shown in Fig. 6.4c. When this DLL locks, the total delay through both DCDLs and the feedback path inverter is one half period of CLKIN (180°). Therefore, the delay through one DCDL is virtually a quarter period (90°) and CLKOUT is a quadrature clock.

The previous technique can be extended and used for constant factor clock multiplication. In Fig. 6.4d, the delay through all eight DCDLs is one CLKIN period. The delay through a single DCDL is 45°. The outputs of the eight DCDLs are equally spaced phases spanning the entire CLKIN period. The toggle element can be thought of as a toggle flop with eight independent clock ports. Every positive edge of each of the eight phases can toggle the flop, thus, producing a CLKOUT that has a frequency equal to four times that of the input.

The final example (Fig. 6.4e) is a deviation from strictly clocking applications. A DLL can be used for absolute measurements of unknown delays (time-to-digital conversion). First, a 2-point calibration is necessary. The DLL is placed in calibration mode (unknown delay is bypassed) and an input clock of a known period T_0 is fed into the CLKIN input. The DLL locks and the setting is recorded (s_0). The input clock period is changed to T_1, the DLL is allowed to lock and the setting is recorded again (s_1). We now have a system of two equations with two unknowns:

$$D_{\min} + s_0 \cdot d_r = T_0, \tag{6.1}$$

$$D_{\min} + s_1 \cdot d_r = T_1, \tag{6.2}$$

where $D_{\min}$ is the DCDL delay at the minimum delay setting and d_r is the DCDL resolution. We can solve the above system and obtain values for $D_{\min}$ and d_r. The DLL now is placed out of calibration mode, and the unknown delay is multiplexed into the system. The DLL locks and the setting (s_u) is recorded. The absolute delay is, therefore, $D_{\min} + s_u \cdot d_r$.

6.4 Phase Detectors

In a digital delay locked loop, the output of the phase detector is typically processed by a digital circuit such as an up/down counter or in the most general case an FSM controller. The most common phase detector for such an application is a specially-designed flip-flop that has a single bit output indicating leading or lagging feedback

clock as described in Sect. 6.2 and shown in Fig. 6.2. Traditionally, such a structure with a single-bit digital output is called a "bang–bang" phase detector. The main design goals for a flip-flop used as a phase detector are:

1. It must be a fully static design containing cross-coupled nodes exhibiting exponential voltage development with time (Sect. 6.4.1) to minimize time spent in a potential metastable state and prevent system failure.
2. It must have a small sampling window d_{sw}, which is the sum of the underlying flop setup and hold times ($T_s + T_h$) to guarantee good phase matching between feedback clock and reference clock.
3. The setup and hold times must be well-balanced to avoid deterministic bias during phase detection and result in a significant systematic phase error between feedback clock and reference clock. A differential design can be desirable, but it is not a requirement.
4. The open loop gain of the cross coupled gates must be high to guarantee quick exit from a potential metastable condition (Sect. 6.4.1).
5. The capacitance of the cross-coupled nodes must be kept to a minimum given the other constraints in order to guarantee quick exit from metastability (Sect. 6.4.1). This will also help minimize setup requirements but may hurt hold time.
6. Unlike regular flop designs, a short clock-to-q delay is not a critical design requirement, since there is typically a full reference clock cycle available until the phase detector output needs to be setup and processed by the FSM controller. This path is unlikely to be critical. Moreover, unlike a proportional phase detector, balancing clock-to-q delays for a 0-to-1 vs. a 1-to-0 transition is not necessary. There is no phase information encoded in this delay for bang–bang operation.

The edge-triggered fully static flop designs of Chap. 3 such as the master-slave latch (flop) of Fig. 3.4 or the sense-amp flip-flop of Fig. 3.13 can be used as a bang–bang phase detector assuming that the design is tuned to meet the design goals outlined above. One design goal which won't be met is the balancing of the setup and hold times (and tracking across PVT) due to the assymetry in the clock and data path in any regular flip-flop. One way around this problem is illustrated in Fig. 6.5 [1].

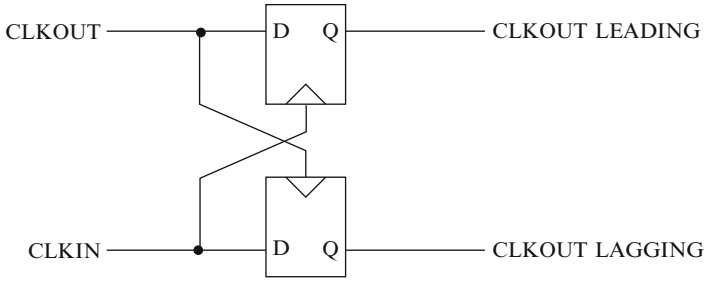


Fig. 6.5. Symmetric phase detector out of asymmetric flops

In a single-flop phase detector, the feedback clock leading decision depends on the setup time and the feedback clock lagging decision depends on the hold time. Asymmetry between these two properties can introduce a systematic phase error. In the coupled flops of Fig.6.5, both decisions depend on the setup time of the flop and assuming identical flops, the systematic phase error is removed. The coupled flops of Fig.6.5 are also a virtual ternary phase detector where a 11 or 00 state can be interpreted as a NOP (no operation) where no adjustment to the delay line takes place.

A flip-flop design that addresses all design goals outlined above is certainly possible. Figure 6.6 shows a non-traditional flop design used as a phase detector in [2]. CLKIN constitutes the clock input and CLKOUT is the data input. This edge-triggered flop is composed of three separate RS latches. Latch (A1,A2) is the master latch, latch (C1,C2) is the slave latch, and latch (B1,B2) is an auxiliary latch whose additional state is necessary for correct operation. Setup and hold timing behavior is entirely set by the master latch. The auxiliary latch is not in the signal path during sampling and the slave latch only affects clock-Q delay which is not relevant for bang–bang phase detector operation. Gates D1 and D2 are only present for load balancing along the CLKIN and CLKOUT paths.

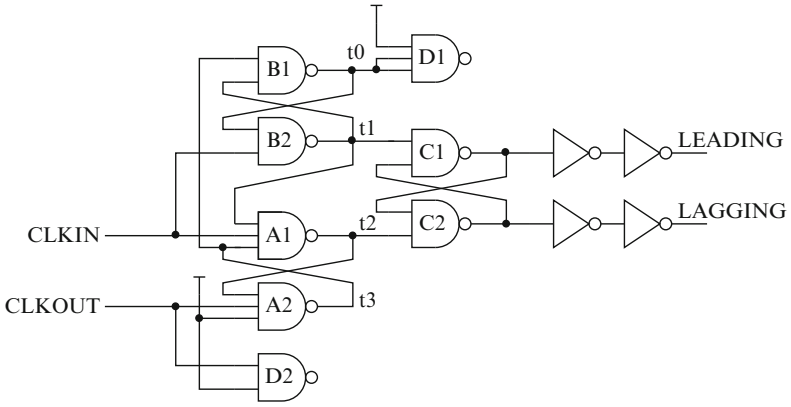


Fig. 6.6. Bang–bang phase detector. Reproduced with permission in a form similar to that in [2], ©1988 IEEE

Table 6.1 shows a truth table for the internal nodes $t0$–$t3$ of Fig.6.6. We observe that when CLKIN = 0, the inputs $t1$ and $t2$ of the slave latch are at logic 1 which means that the phase detector outputs are holding. On the positive edge of CLKIN, the internal nodes will assume a value conditional on the state of CLKOUT around the sampling edge, and the slave latch will be updated accordingly. The full operation of this structure is illustrated in the state sequence Tables 6.2 and 6.3 which show the sequence of internal node and output values for CLKOUT lagging and leading, respectively.

Table 6.1. Truth table for bang–bang phase detector internal nodes

CLKIN	CLKOUT	$t0$	$t1$	$t2$	$t3$
0	0	0	1	1	1
0	1	1	1	1	0
1	0	Hold	Hold	$\overline{t1}$	1
1	1	$\overline{t3}$	$t3$	Hold	Hold

Table 6.2. State sequence for CLKOUT lagging CLKIN

CLKIN	CLKOUT	$t0$	$t1$	$t2$	$t3$	*Leading*	*Lagging*
0	0	0	1	1	1	Hold	Hold
1	0	0	1	0	1	0	1
1	1	0	1	0	1	0	1
0	1	1	1	1	0	0 (Hold)	1 (Hold)
0	0	0	1	1	1	0 (Hold)	1 (Hold)

Table 6.3. State sequence for CLKOUT leading CLKIN

CLKIN	CLKOUT	$t0$	$t1$	$t2$	$t3$	*Leading*	*Lagging*
0	0	0	1	1	1	Hold	Hold
0	1	1	1	1	0	Hold	Hold
1	1	1	0	1	0	1	0
1	0	1	0	1	1	1	0
0	0	0	1	1	1	1 (Hold)	0 (Hold)

The sampling window of this phase detector is set entirely by the master latch (A1,A2). The setup and hold times of interest occur when the CLKIN and CLKOUT positive edges are virtually coincident. The setup time is set by the delay difference of NAND gate A2 on a 1-0 transition on node $t3$ and the delay of NAND gate A1 on a 1-0 transition on node $t2$:

$$T_{\mathrm{s}} = t_{d(\mathrm{A2})} - t_{d(\mathrm{A1})}. \tag{6.3}$$

Similarly, the hold time is:

$$T_{\mathrm{h}} = t_{d(\mathrm{A1})} - t_{d(\mathrm{A2})}. \tag{6.4}$$

Even though the setup and hold time of this phase detector can be minimized by removing all static bias from physical design, the true parameters should be determined statistically through Monte Carlo simulations. A positive setup and hold time must be determined that will guarantee correct decision with arbitrarily high probability in the presence of random process variations.

The phase detector of Fig.6.6 can satisfy all design requirements including being fully static and having nominally small and equal setup and hold times. Moreover, it is a true single phase design which requires no inversion on the clock or data inputs, thus, minimizing setup and hold requirements. The author has been hard pressed to find a better overall design with good portability across process nodes.

6.4.1 Metastability

A bang–bang phase detector will produce the correct lead/lag decision if the controlled clock input falls outside its sampling window (d_{sw}). A DLL in the locked state though will cause frequent controlled clock edges to fall within d_{sw}. A naive approach to this issue would be to assume that in this situation a wrong decision coupled with a small DCDL resolution d_r will not cause a DLL failure but rather a small increase in the DLL phase error which can be accommodated by the application.

A fourteenth century French philosopher Jean Buridan postulated that a donkey located at equal distances between two bales of hay should theoretically starve to death because it will be equally attracted to both [3, 4]. A more precise articulation of this principle in the words of Lamport [4] is as follows:

> *Buridan's Principle:* A discrete decision based upon an input having a continuous range of values cannot be made within a bounded length of time.

The continuous variable in the Buridan example is the position of the donkey along the axis connecting the two bales as a function of time and initial position. The discrete decision is which bale of hay to consume. In the case of the phase detector, the continuous variable is the voltage at the cross-coupled nodes of the flip-flop state element as a function of time and initial voltage. In this case, the initial voltage refers to the voltage established at the cross-coupled nodes right after sampling the flop data input. The discrete decision is whether a particular node will converge to logic 0 or logic 1.

This is of course the well-known synchronization failure problem [3]. No one can build a phase detector that can guarantee a valid logic output in bounded time if the controlled clock falls within the d_{sw} established by the sampling clock. In this case, the output can be at an undefined logic level (metastable state) for an arbitrarily long time, and eventually this undefined level may be interpreted by two separate logic receivers as different logic values. This can cause catastrophic DLL failure because it can drive the control automaton into a wrong or undefined state. Metastability in static flops has been extensively studied and observed in practice [5].

Although it is not possible to build a phase detector which will never experience metastability for unbounded time, it is possible to design a phase detector that minimizes the probability of system failure. We will study this problem by deriving

an expression for the voltage in the cross coupled gates inside the phase detector as a function of time and initial conditions. Then, we will assign a probability measure to two events: Entering a metastable condition at time $t = 0$ and still being in a metastable condition at time $t = t_d$. Finally, we will derive an expression for the mean-time-between-failures (MTBF) which provides an indication of how often we can expect catastrophic synchronization failures in a DLL.

We develop a first order metastability model based on the analysis of Veendrick [6]. We begin by assuming a linear voltage transfer function for a CMOS inverter (for normalized supply voltage):

$$V_{\mathrm{o}} = -A(V_{\mathrm{i}} - V_{\mathrm{sw}}) + 0.5, \tag{6.5}$$

where V_{i} and V_{o} are the voltages at the input and output of the inverter, respectively, V_{sw} is the switching threshold (defined as the value of V_{i} for which $V_{\mathrm{o}} = 0.5$) and A is a large positive number denoting the inverter gain. There is an underlying assumption that V_{o} is further processed by a nonlinear limiter which clamps V_{o} to 1 for all $V_{\mathrm{o}} > 1$ and also clamps V_{o} to 0 for all $V_{\mathrm{o}} < 0$. All voltages are normalized $(V_{\mathrm{i}}, V_{\mathrm{o}}, V_{\mathrm{sw}})$ and assume values between 0 and 1. Figure 6.7 plots Eq. (6.5) for $A = 10$, $V_{\mathrm{sw}} = 0.5$. Figure 6.8 plots Eq. (6.5) for various A and V_{sw} parameter values.

Cross-coupling two inverters results in three equilibrium positions (Fig. 6.9): Two stable positions (a small voltage perturbation will result in the system canceling it out and remaining in the same state) that signify the two memory states, and an unstable equilibrium position (a small voltage perturbation will result in the system leaving this state and assuming one of the two stable states) signifying the metastable state. The metastable voltage V_{m} that the two system nodes converge is a function of the inverter gain A and switching threshold V_{sw} and can be computed easily by setting $V_{\mathrm{i}} = V_{\mathrm{o}} = V_{\mathrm{m}}$ in Eq. (6.5) (Fig. 6.10):

$$V_{\mathrm{m}} = \frac{2AV_{\mathrm{sw}} + 1}{2(A+1)}. \tag{6.6}$$

We introduce transient behavior in the cross-coupled inverter model by adding an *RC* output stage to the ideal gain elements along the lines of [6] as shown in Fig. 6.11. We omit the limiters and make the assumption that this model is only valid for inverter input voltages that fall in the linear region of the transfer function $V_{\mathrm{sw}} - 0.5/A \leq V_{\mathrm{i}} \leq V_{\mathrm{sw}} + 0.5/A$ (Fig. 6.7).

We assume that the sampling switch is ideal, and that clocking this latch consists of establishing the following initial conditions at $t = 0$ (please note the change of variable names $V_1(t)$ and $V_2(t)$ in Fig. 6.11):

$$V_2(0) = V_0, \tag{6.7}$$

$$V_1(0) = -A(V_0 - V_{\mathrm{sw}}) + 0.5, \tag{6.8}$$

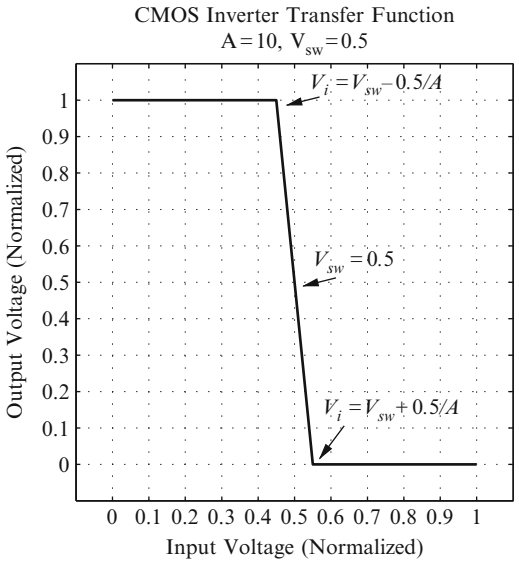


Fig. 6.7. CMOS inverter voltage transfer function

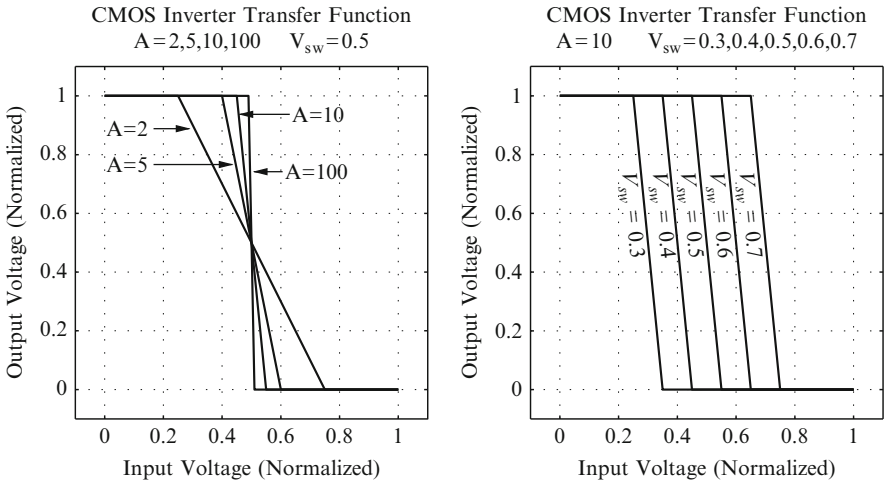


Fig. 6.8. CMOS inverter voltage transfer function parametrization

We want to derive an analytical expression for $V_1(t)$. We begin by enforcing KCL at $V_1(t)$ and $V_2(t)$, respectively:

$$\frac{\mathrm{d}V_1(t)}{\mathrm{d}t} + \frac{1}{RC}V_1(t) + \frac{A}{RC}V_2(t) - \frac{AV_{\mathrm{sw}} + 0.5}{RC} = 0, \tag{6.9}$$

$$\frac{\mathrm{d}V_2(t)}{\mathrm{d}t} + \frac{1}{RC}V_2(t) + \frac{A}{RC}V_1(t) - \frac{AV_{\mathrm{sw}} + 0.5}{RC} = 0. \tag{6.10}$$

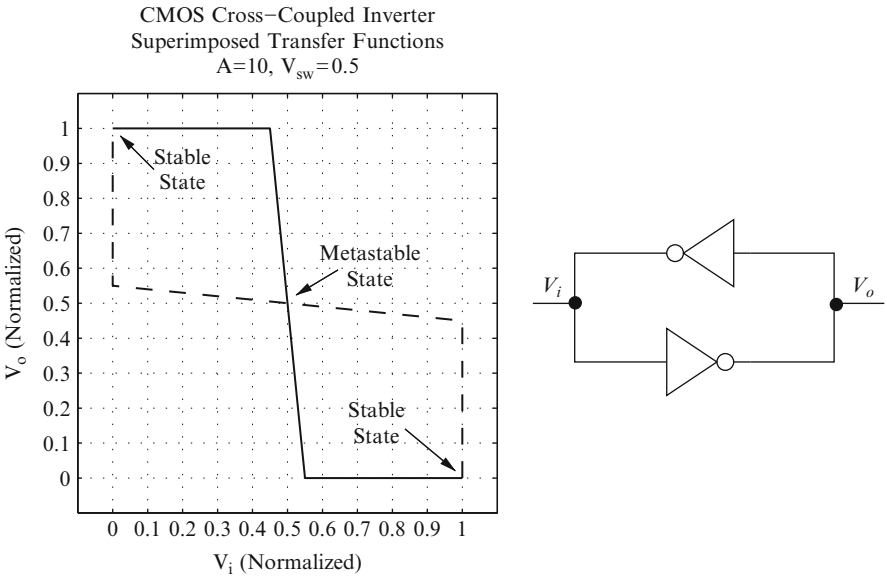


Fig. 6.9. Metastable state in CMOS cross-coupled inverters

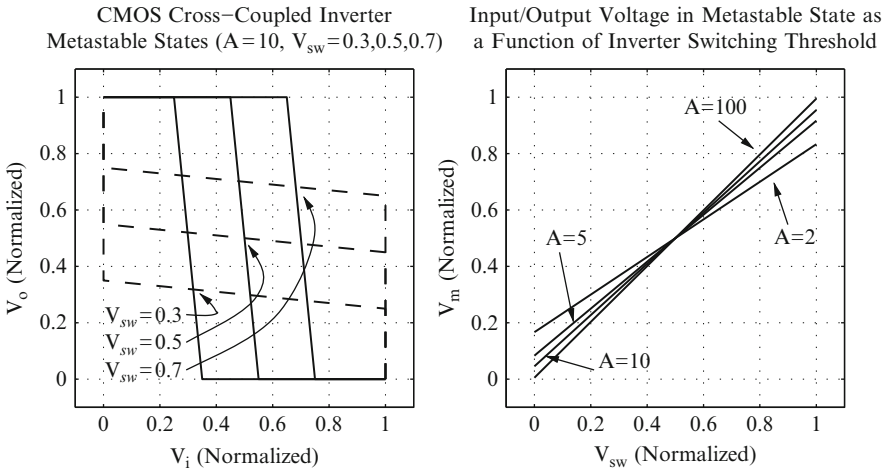


Fig. 6.10. Voltage in metastable state

In order to simplify the analysis, we introduce two new functions:

$$V_{\mathrm{d}}(t) = V_1(t) - V_2(t),$$
$$V_{\mathrm{s}}(t) = V_1(t) + V_2(t).$$

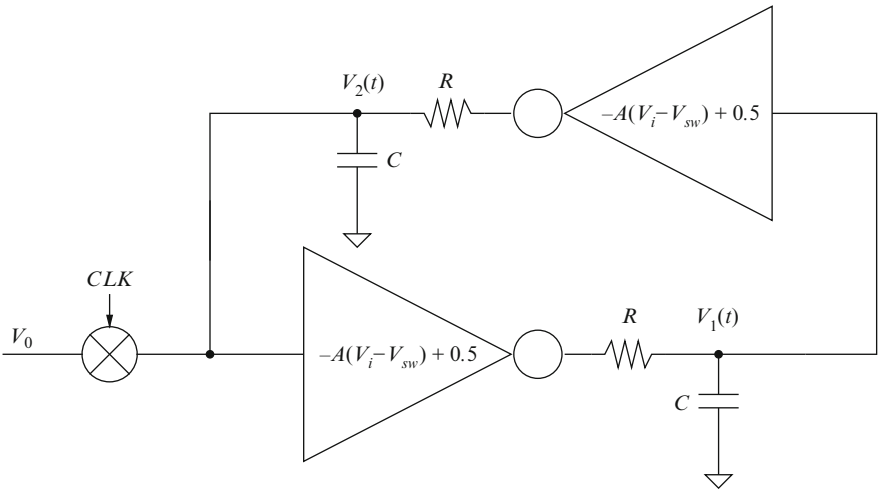


Fig. 6.11. Small signal model of cross-coupled inverters. Reproduced with permission in a form similar to that in [6], ©1980 IEEE

Subtracting (6.10) from (6.9) yields:

$$\frac{\mathrm{d}V_d(t)}{\mathrm{d}t} - \frac{A-1}{RC}V_d(t) = 0. \tag{6.11}$$

Adding (6.10) to (6.9) yields:

$$\frac{\mathrm{d}V_s(t)}{\mathrm{d}t} + \frac{A+1}{RC}V_s(t) - \frac{2AV_{\mathrm{sw}}+1}{RC} = 0. \tag{6.12}$$

Initial conditions for $V_{\mathrm{d}}(t)$ and $V_{\mathrm{s}}(t)$ have been established during sampling:

$$V_{\mathrm{d}}(0) = V_1(0) - V_2(0),$$

$$V_{\mathrm{s}}(0) = V_1(0) + V_2(0).$$

The solution of (6.11) is

$$V_d(t) = [V_1(0) - V_2(0)]\mathrm{e}^{\frac{A-1}{RC}t}. \tag{6.13}$$

The solution of (6.12) is

$$V_s(t) = \left(V_1(0) + V_2(0) - \frac{2AV_{\mathrm{sw}}+1}{A+1}\right)\mathrm{e}^{-\frac{A+1}{RC}t} + \frac{2AV_{\mathrm{sw}}+1}{A+1}. \tag{6.14}$$

Expression (6.14) can be simplified by substituting the expression for the metastable voltage (6.6):

$$V_s(t) = [V_1(0) + V_2(0) - 2V_{\mathrm{m}}]\,\mathrm{e}^{-\frac{A+1}{RC}t} + V_{\mathrm{m}}. \tag{6.15}$$

Reverting back to the original variable $V_1(t)$, we have

$$V_1(t) = \left(\frac{V_1(0) + V_2(0)}{2} - V_{\mathrm{m}}\right)\mathrm{e}^{-\frac{A+1}{RC}t} + \frac{V_1(0) - V_2(0)}{2}\mathrm{e}^{\frac{A-1}{RC}t} + V_{\mathrm{m}}. \tag{6.16}$$

Equation (6.16) can be simplified and become more meaningful if we make the following observation: The decaying exponential quickly vanishes with increasing t and the expression is dominated by the increasing exponential. We can approximate the expression above by only keeping the positive exponential with its coefficient adjusted for the initial condition:

$$\boxed{V_1(t) = (V_1(0) - V_{\mathrm{m}})\mathrm{e}^{\frac{A-1}{RC}t} + V_{\mathrm{m}}.} \tag{6.17}$$

Equation (6.17) is fundamental in the description of cross-coupled circuits and states that a perturbation from the unstable equilibrium position V_{m} results in a time-exponential path toward a stable state. We note that Eq. (6.17) has a consistent solution for $V_1(t = 0)$ and also that for $V_1(0) = V_{\mathrm{m}}$, we have $V_1(t) = V_{\mathrm{m}}$ for all t which is another way of saying that a metastable state may persist indefinitely. We also note that Eq. (6.17) is identical to Eq. (1-18) in [3] although the derivation is very different. Mead and Conway derive the cross-coupled inverter node equation starting from basic circuit relationships in a depletion-loaded NMOS cross-coupled pair.

It is important to understand that Eq. (6.17) is only valid for node voltages in close proximity to the metastable voltage V_{m}. Both initial conditions $V_1(0)$ and $V_2(0)$ must lie in the shaded regions of Fig. 6.12 and be related through the gain curve. Unless this is true, the assumptions made during the derivation of (6.17) (both inverters are in their linear region) are no longer operative. We can still make the assertion that Eq. (6.17) fully describes the metastable state because reaching the voltage limit of the above equation signifies the exit from metastability: At that point, the node voltage has reached a value outside the linear gain limits and any receiver with a similar gain curve will interpret it as a well-defined logic state. Figure 6.13 plots (6.17) for various initial conditions of $V_1(0) - V_{\mathrm{m}}$ and exhibits the exponential nature of the trajectory leading the node voltage out of the metastable state.

Metastability is a stochastic phenomenon and must be studied with probabilistic tools. Equation (6.17) states that there exists an initial condition $V_1(0) = V_{\mathrm{m}}$ which will cause the phase detector to lie in the metastable state indefinitely. At the same time though, if $V_1(t)$ is modeled as a continuous random variable, then the probability of $V_1(t)$ assuming a discrete value V_{m} vanishes. It is not possible to design a perfect phase detector, one that is guaranteed not to lie in the metastable state indefinitely.

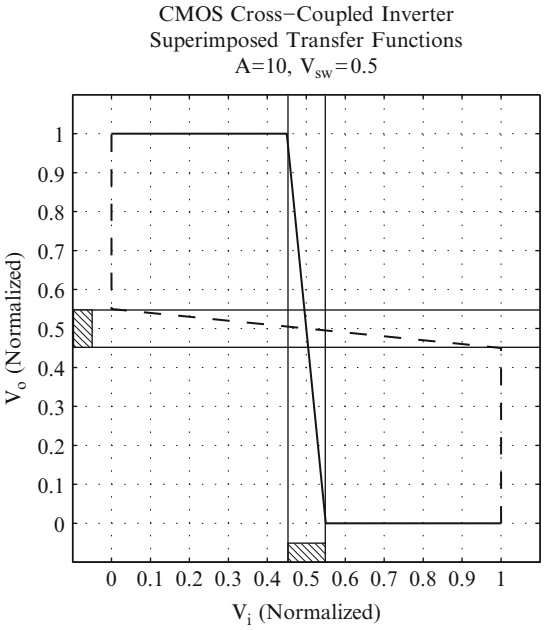


Fig. 6.12. Valid node voltage region for Eq. (6.17)

It is possible though to design a phase detector that has an arbitrarily small conditional probability of being in the metastable state at a given time t_{d} assuming it was in the metastable state at time $t = 0$.

Before proceeding to define the problem in stochastic terms, let us simplify Eq. (6.17) by referring all voltages to V_{m} instead of ground [6]:

$$V(t) = V_1 \mathrm{e}^{\frac{t}{\tau_m}}, \tag{6.18}$$

$$V(t) = V_1(t) - V_{\mathrm{m}}, \tag{6.19}$$

$$V_1 = V_1(0) - V_{\mathrm{m}}, \tag{6.20}$$

$$\tau_m = \frac{RC}{A-1}. \tag{6.21}$$

Furthermore, we need to state two assumptions:

- We consider a phase detector to be metastable when its output voltage $V_1(t)$ is within ΔV of V_{m}:

$$|V(t)| < \Delta V, \tag{6.22}$$

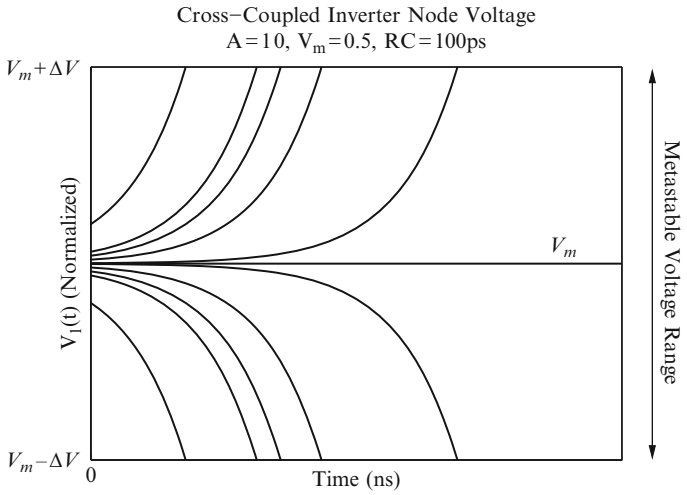


Fig. 6.13. Exponential trajectory towards a stable state

where ΔV is a phase detector/synchronizer parameter and is the maximum voltage deviation from V_{m} which will still guarantee that both cross-coupled nodes are in the linear gain region of the two cross-coupled CMOS gates.

- The initial condition V_1 is a random variable uniformly distributed between $-\Delta V$ and ΔV. By definition then, the system is metastable at $t = 0$.

We wish to quantify the probability that the phase detector is still metastable at $t = t_d$ given that it is metastable at $t = 0$:

$$\Pr(|V(t_d)| < \Delta V) = \Pr(\left|V_1 \mathrm{e}^{\frac{t_d}{\tau_m}}\right| < \Delta V). \tag{6.23}$$

Multiplying both sides of the event inequality in (6.23) with $\mathrm{e}^{-\frac{t_d}{\tau_m}}$ yields:

$$\Pr(|V_1| < \Delta V \mathrm{e}^{-\frac{t_d}{\tau_m}}) = \mathrm{e}^{-\frac{t_d}{\tau_m}}. \tag{6.24}$$

The last equation stems from the fact that V_1 is uniformly distributed between $-\Delta V$ and ΔV. We can arrive at the same result using a different approach [3]: Let us assume that the exit from the metastable state can be modeled as the first arrival of a Poisson process with rate λ. Modeling it as a Poisson arrival implies that the probability of leaving the metastable state within a very small time interval Δt is proportional to the duration of the interval and equal to $\lambda \Delta t$. This makes physical sense since in general, Poisson processes are heavily used to model simple stochastic

phenomena in continuous time. The probability density function of the time until the first Poisson arrival (l_1) is given by the following Eq. ([7]):

$$f_{l_1}(t) = \lambda e^{-\lambda t}. \tag{6.25}$$

The probability that the phase detector is still metastable at time t_d is given by the following expression:

$$\Pr(l_1 > t_d) = \int_{t_d}^{\infty} \lambda e^{-\lambda t} dt = e^{-\lambda t_d}. \tag{6.26}$$

For $\lambda = 1/\tau_m$, expression (6.26) is in agreement with (6.24).

Equation (6.24) describes a conditional probability where the conditioning event is $|V_1| < \Delta V$ (phase detector is metastable at time $t = 0$). We will now quantify the probability of the conditioning event and determine the overall unconditional probability of phase detector failure. Let us assume that in the DLL locked state, the feedback clock edges are uniformly distributed between $-d_r$ and d_r with respect to the sampling reference clock edge (where d_r is the DCDL resolution). This is a reasonable assumption to make assuming that the digital control module will be designed with a small ± 1 LSB limit cycle and various noise processes will add uncertainty to the feedback clock edges. We further assume a feedback clock slew rate of L in V/s. Figure 6.14 shows graphically that such a distribution of feedback edges will cause a uniform distribution of sampled voltages with a range equal to $2Ld_r$. The probability of the conditioning event is, therefore, $\Delta V/(Ld_r)$, and the overall phase detector unconditional failure probability at t_d is,

$$\boxed{\Pr(\text{Metastable at } t_d) = \frac{\Delta V}{Ld_r} e^{-\frac{t_d}{\tau_m}}.} \tag{6.27}$$

So far, we have made enough assumptions and simplified our model sufficiently to be able to derive a simple formula for ΔV, the sampled voltage deviation from V_m which will place a cross-coupled structure in a metastable state. Without loss of generality, we can simplify the analysis by assuming $V_m = V_{sw} = 0.5$ in Eq. (6.5) (all voltages are normalized with respect to the nominal supply voltage). The range of input voltages which will cause a cross-coupled structure to become metastable should be determined by requiring that the result of applying the inverter transfer function (6.5) to such input voltage should yield a voltage which is still in the linear range of the same transfer function (shown in Fig.6.7). If this is not the case, then a logic gate with similar gain will interpret the cross-coupled output as a discrete logic value. Moreover, the second cross-coupled gate will also interpret such an output as

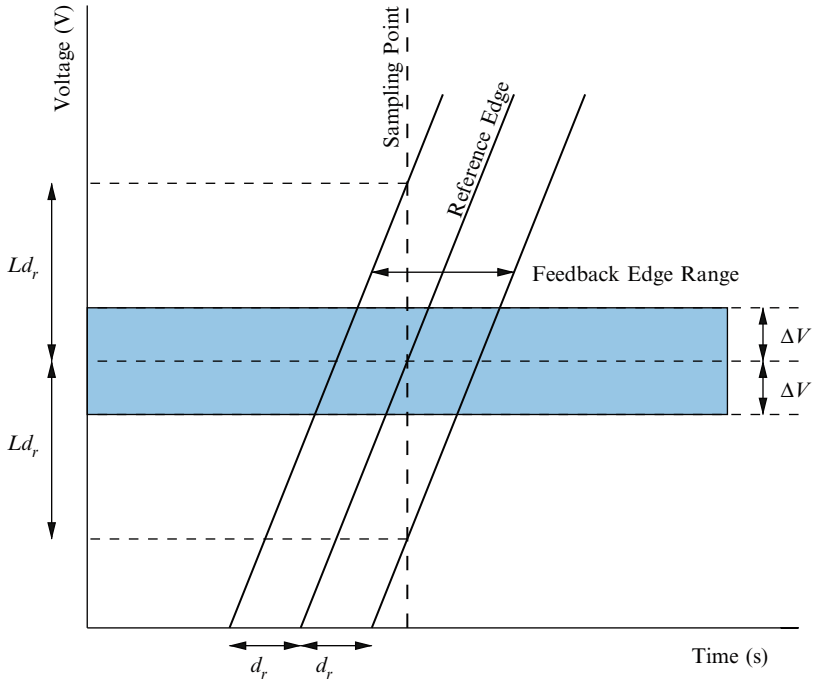


Fig. 6.14. Determining the probability of entering metastability

a discrete logic value, and the latch will converge to a non-metastable state as soon as the sample switch opens up. It is straightforward to calculate that the above condition will be satisfied for:

$$\Delta V = \frac{1}{2A^2}. \tag{6.28}$$

Let us now briefly focus on t_d, the allowed metastability resolution time. For a digital DLL with a bang–bang phase detector, t_d is at least equal to the reference clock period T. It is also common to add additional synchronization stages and effectively extend the resolution time to nT. Updating Eq. (6.27) with our latest observations and also expanding the expression for τ_m we have,

$$\boxed{\Pr(\text{PD Failure}) = \frac{1}{2nA^2 L d_\text{r}} \mathrm{e}^{-\frac{A-1}{r_o C} nT},} \tag{6.29}$$

where

d_r DCDL resolution
L Feedback clock edge rate
r_o Small signal output resistance of gate around V_m.
Should be small signal saturation resistance of pullup device
in parallel with pulldown device

A Small signal gate gain. Should be $g_m r_o$ where g_m is pullup transconductance in saturation in parallel with pulldown transconductance.
C Cross-coupled node capacitance
T Reference clock period
n Number of synchronization stages including the phase detector

An additional scaling factor of $1/n$ has been included to account for the fact that a digital controller typically implements a $1/n$ correction issue rate in order to minimize limit cycle amplitude and guarantee stability (Sect. 6.6). In such a case, the controller only looks at the output of the phase detector once every n cycles and the probability of entering a metastable state must be scaled accordingly. Equation (6.29) quantifies the risk of DLL failure for T seconds of operation. An alternative way of expressing this risk is the mean-time-between-failures (MTBF):

$$\text{MTBF} = T\Pr(\text{PD Failure}). \tag{6.30}$$

The MTBF figure of merit is typically expressed in years and signifies the average duration of error-free operation. In digital system applications, an MTBF in excess of 10–100 years is typically desired.

An Example of Phase Detector Failure Calculation

It is important to realize that Eq. (6.29) is based on a number of assumptions and approximations such as the inverter piecewise linear transfer function of Eq. (6.5). Nevertheless, it captures the right dependencies on design parameters and can be used as a phase detector design guideline. The MTBF value predicted by (6.30) should have substantial margin of a few orders of magnitude to guarantee correct operation even in the presence of modeling inaccuracies that can affect the final result in an exponential nature.

Although it is certainly possible to assign circuit parameters to all contributing factors of Eq. (6.29) [6], it is probably much easier to estimate MTBF using spice small signal analysis around V_m. Figure 6.15 shows the setup of a spice AC analysis for a 45 nm cross-coupled inverter using 45 nm bulk spice predictive models [8–10]. A self-biased inverter is used to compute the metastable voltage (V_{bias} in Fig. 6.15) which is then used as the bias point through an ideal buffer for a cross-coupled pair of identical sizing. An AC source is cascaded in series with the bias point to calculate gain, output resistance, and node capacitance.

For this example, Table 6.4 shows the MTBF calculation under certain assumptions for L, n, T, and d_r. The MTBF calculation is very sensitive to the metastability time constant (τ_m), and the parameters that affect it because of the exponential dependence. As an example, if the node capacitance gets doubled from the value in Table 6.4 due to the necessary addition of a receiving gate, MTBF becomes 4.7624×10^9 years. If it triples due to poor phase detector design, MTBF will become 24.58 years.

Table 6.4. Example MTBF calculation for the circuit of Fig. 6.15

Parameter	Value	Units
Clock period (T)	1×10^{-9}	s
Slew rate (L)	1×10^{10}	V/s
DLL subsampling factor (n)	1	–
DLL resolution (d_r)	50×10^{-12}	s
Gain (A)	4.5456	–
Output resistance (r_o)	9.3079×10^{3}	Ω
Node capacitance (C)	3.3271×10^{-15}	F
Probability of entering metastability: $\frac{1}{2nA^2Ld_\mathrm{r}}$	0.0484	–
Metastability time constant: $\tau_m = \frac{r_o C}{A-1}$	8.7343×10^{-12}	s
Conditional probability of being metastable at nT: $\mathrm{e}^{-\frac{nT}{\tau_m}}$	1.8928×10^{-50}	–
Unconditional synchronization failure probability: $\frac{1}{2nA^2Ld_\mathrm{r}}\mathrm{e}^{-\frac{nT}{\tau_m}}$	9.16×10^{-52}	–
MTBF	3.4615×10^{34}	years

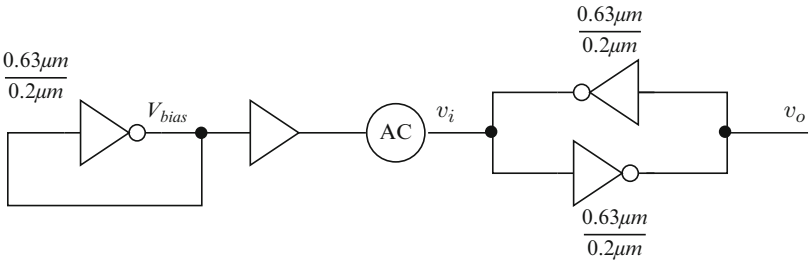


Fig. 6.15. Determining MTBF with a spice simulation

6.5 DCDL Design

There is a very broad range of delay line design options limited only by designer imagination. This section provides a structured overview of various DCDL design approaches. The structure based on DCDL characteristics adopted for this section does not follow a widely accepted classification in the field but is merely done for ease of presentation. Moreover, this section is by no means an all-inclusive exposition of all possible design options. It can be considered a stratified sampling of the design space that presents various alternatives based on the main DCDL characteristics of Sect. 6.2 ($D_\mathrm{min}, D_\mathrm{max}, d_\mathrm{r}$). Additional characteristics (such as input capacitance, linearity, and potential for synchronous vs. asynchronous setting change) will also be identified.

For ease of presentation, we divide DCDLs into two main categories based on d_r: Gate-delay DCDLs (coarse) and Subgate-delay DCDLs (fine).

6.5.1 Gate-Delay DCDLs

Gate-delay DCDLs can be constructed by cascading standard CMOS logic gates to form a delay chain with intermediate outputs routed to a high-fanin multiplexing structure. Alternatively, the chain length can be modulated using low-fanin distributed multiplexing structures as part of the delay cell. They exhibit relatively small D_{min}, arbitrarily high D_{max} but relatively coarse d_r by design.

A very popular coarse delay line is shown in Fig. 6.16 [11, 12]. This particular implementation has four hierarchical delay stages between input A and output Y. It is controlled by a bidirectional shift register with one-hot encoding ($Q[3:0]$). Resolution d_r is $2T_G$, where T_G is the average CMOS gate delay. NAND gates labeled with the letter B form a distributed multiplexer that controls the entry point of the clock (A input) in this cascaded structure. One important observation is that this design presents substantial load to the clock especially for a large number of delay stages since all delay element inputs are shorted. Clock buffering may be necessary which will increase D_{min} to a value larger than $2T_G$. An arbitrarily high number of stages can be cascaded to increase dynamic range provided that input A is sufficiently buffered. D_{max} is $2NT_G$, where N is the number of delay stages.

The folded design of Fig. 6.17 eliminates the heavily loaded A input by modulating the delay line length in a telescopic fashion. This design is also controlled by a bidirectional one-hot shift register. For $Q[3:0] = 0001$ the signal path is Cell $0.A +$ Cell $0.B$. For $Q[3:0] = 0010$ the path is Cell $0.C +$ Cell $1.A +$ Cell $1.B +$ Cell $0.B$. The length modulation of this structure can be compared to the sliding action of a trombone. The wrap-around connection between $IN1$ and $OUT1$ of Delay Cell 3 is not part of the delayed signal path but is necessary to establish the propagating condition for the B NAND gate associated with the delay cell that constitutes the final telescope link (CTL = 1). If the delay cell had non inverting forward and reverse paths, then input $IN1$ of Delay Cell 3 could have been hard-wired to ground. The wrap-around connection makes it toggle between odd and even settings to ensure that the signal will propagate for all setting values. The characteristics of the telescopic line are identical to the previous one with the sole exception that D_{min} can be $2T_G$ since there is no need for input buffering.

A straightforward coarse DCDL implementation is shown in Fig. 6.18. Two cascaded inverters constitute the delay element and a binary multiplexer selects the appropriate output tap. D_{min} is equal to $2T_G + \log_2 N \times T_G$ and d_r is still $2T_G$. This delay line can be controlled by a binary counter ($Q[1:0]$) requiring $\log_2 N$ storage elements as opposed to the N storage elements of the previous two designs. The control structure of this DCDL scales better with N as opposed to the NAND-based ones. D_{min} also increases though with increasing N and for certain applications this may be a concern.

DCDLs with a single T_G resolution are also possible. Figure 6.19 illustrates one based on a differential delay element. The relative sizing of the cross-coupled devices

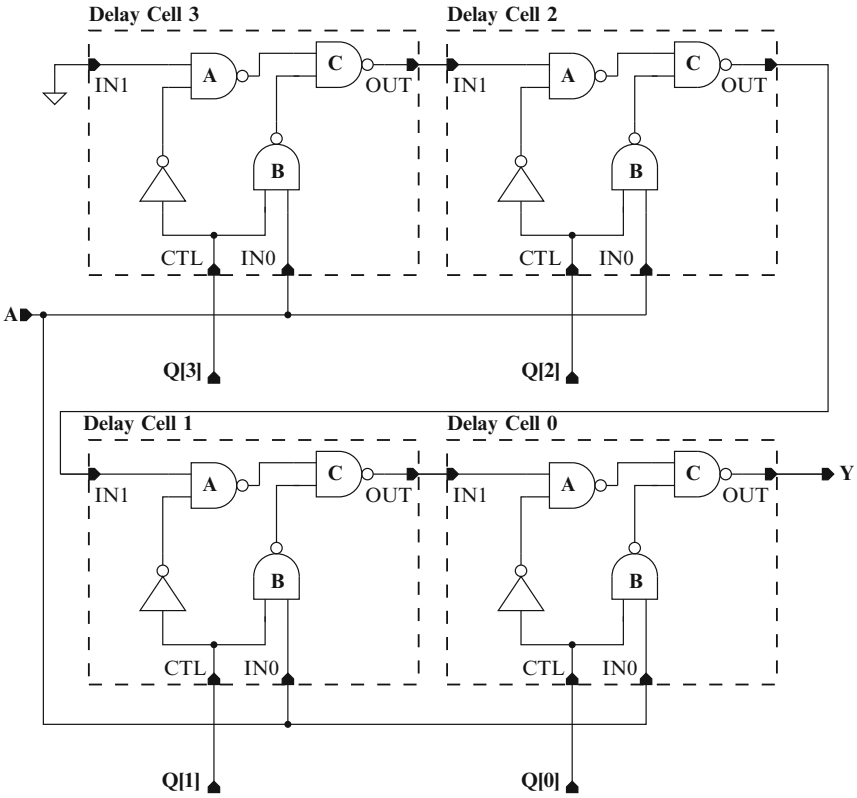


Fig. 6.16. NAND-based registered-controlled 4-stage delay line

vs. the forward devices can be used to fine tune d_r as long as writing the cross-coupled pair can be safely guaranteed across all PVT conditions. Figure 6.20 shows a single-ended single T_G DCDL with a conditionally inverting output stage. The output XNOR gate must be designed with equal delays from both input polarities to the output to ensure linearity.

Design options are certainly not limited to the five examples shown above. Not only different organizations are possible but also different logic families (i.e., dynamic or low swing current model logic) can be used as the base for the delay cell and the multiplexing structure. The characteristics of the examples presented so far are summarized in Table 6.5. Further assumptions are that the single-ended-to-differential converter of Fig. 6.19 and the XNOR of Fig. 6.20 can be implemented with two gates and, therefore, add $2T_G$ to D_{min} and D_{max} of the corresponding DCDLs.

Physical design is very important in precisely controlling DCDL specifications. All delay cells should be identical, and metal capacitance should be thoroughly characterized since it will probably be responsible for a substantial percentage of the cell delay. Post-layout simulation-based characterization across all PVT corners should

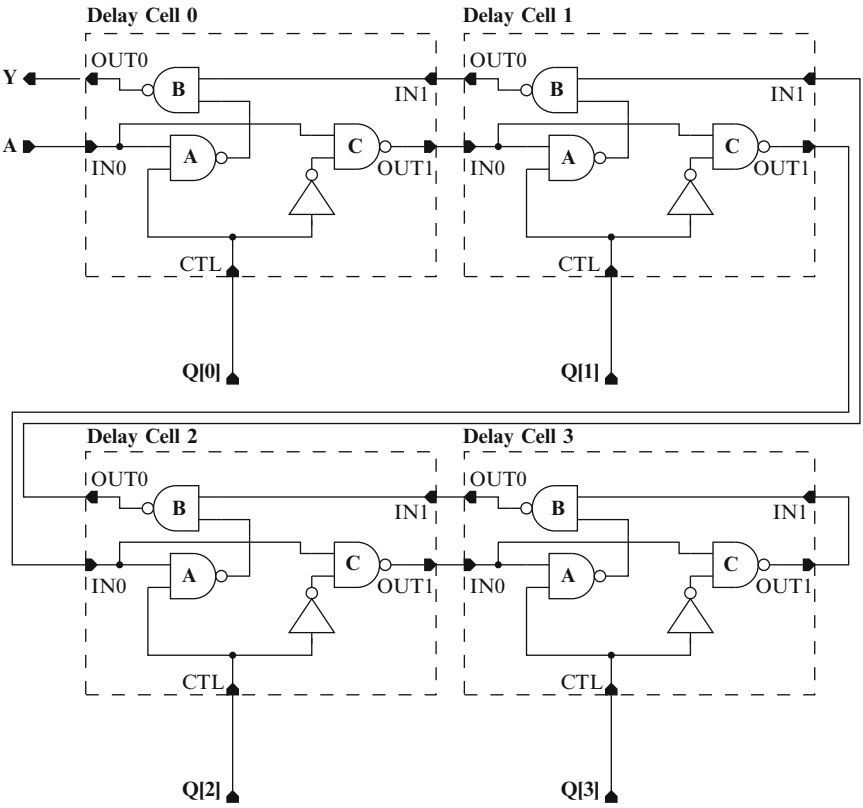


Fig. 6.17. NAND-based telescopic 4-stage delay line

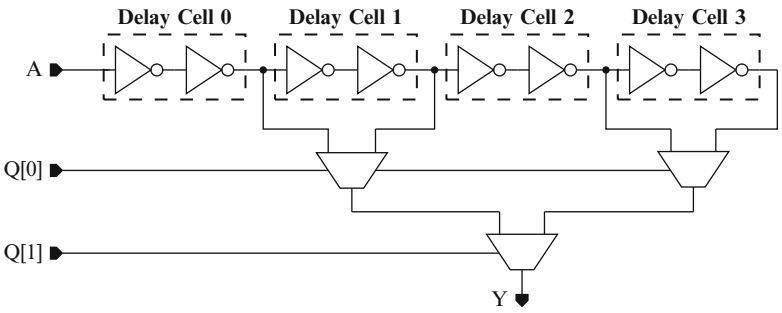


Fig. 6.18. Inverter-based logarithmic 4-stage delay line

be done in order to establish the ranges for $D_{\min}$, $D_{\max}$, and d_r and ensure that the application requirements are met. In most cases, coarse DCDLs will be operating on clock signals, where duty cycle fidelity is important. If there are significant

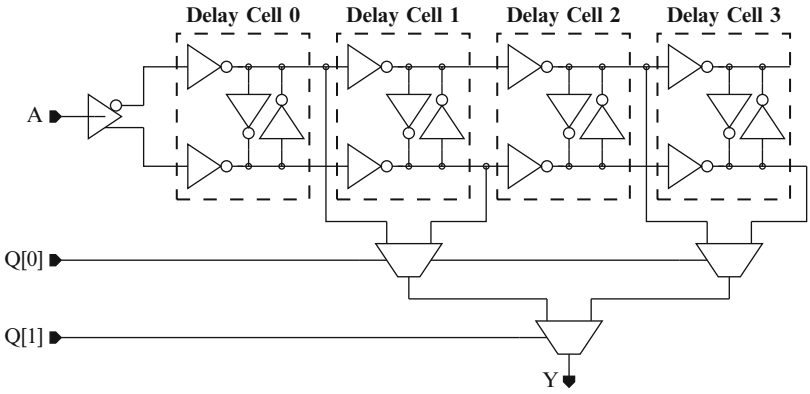


Fig. 6.19. Inverter-based differential 4-stage delay line

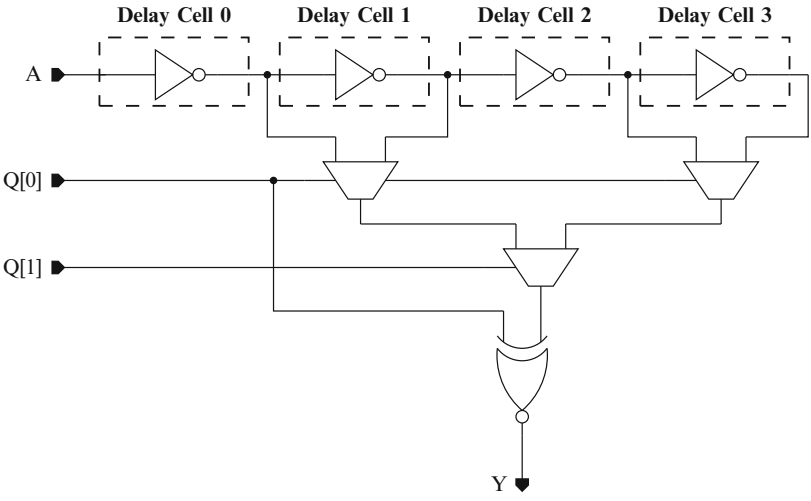


Fig. 6.20. Inverter-based conditional-output 4-stage delay line

Table 6.5. Characteristics of coarse DCDLs

	$D_{\min}$	$D_{\max}$	d_r	Dynamic range	Linearity	CTL flops
NAND (Fig. 6.16)	$> 2T_G$	$> 2NT_G$	$2T_G$	$2(N-1)T_G$	Good	N
Tel. (Fig. 6.17)	$2T_G$	$2NT_G$	$2T_G$	$2(N-1)T_G$	Good	N
Log. (Fig. 6.18)	$T_G(2+\log_2 N)$	$T_G(2N+\log_2 N)$	$2T_G$	$2(N-1)T_G$	Good	$\log_2 N$
Diff. (Fig. 6.19)	$T_G(3+\log_2 N)$	$T_G(2+N+\log_2 N)$	T_G	$(N-1)T_G$	Good	$\log_2 N$
Cond. (Fig. 6.20)	$T_G(3+\log_2 N)$	$T_G(2+N+\log_2 N)$	T_G	$(N-1)T_G$	Good	$\log_2 N$

imbalances between rise vs. fall delay in the delay cell and the multiplexing structure, severe duty cycle distortion may occur at the output signal especially for large delay settings. In extreme cases of high frequency clocks and long DCDLs, the square wave may completely disappear and a DC value may be observed at the output. NMOS vs. PMOS ratios for equal rise and fall times should be used and if possible inverting logic gates should be instantiated in cascaded pairs with similar fanouts to ensure equal treatment of positive vs. negative edges and good duty cycle PVT tracking.

Under certain circumstances, the NAND-based DCDLs of Figs. 6.16 and 6.17 can have a power advantage. Their switching activity is setting dependent and will exhibit reduced switching power at lower settings. A complete power comparison of the DCDLs presented so far is process and application dependent and will provide little if any additional insight.

Synchronous vs. Asynchronous Operation in Coarse DCDLs

DCDL outputs are typically clock signals and as such must always be well behaved and not undergo spurious transitions (glitches) that may cause erroneous circuit operation. In order to demonstrate an important differentiation among DCDLs, we will adopt the ad hoc definition that a DCDL capable of asynchronous operation is a DCDL that can undergo a valid setting change (± 1) at any time without the possibility of a glitch at the output. On the other hand, a DCDL capable of synchronous only operation is a DCDL that requires that valid setting changes be timed synchronously with respect to the input clock in order to guarantee glitch-free output behavior.

DCDLs are essentially combinational (stateless) circuits with clock being simply another input. As a result, spurious transitions at the output are certainly possible unless all inputs are guaranteed to transition at the same time and all inputs have equal delay paths to the output. Neither is true in the DCDL case. Clock is not guaranteed to transition at the same time as the control settings and clock has multiple paths to the DCDL output through the delay element chain. Under certain circumstances though, and using simple analysis, we can convince ourselves that a certain class of DCDLs can have a glitch-free output. Before proceeding with the analysis, we make two important assumptions: We are considering DCDLs where all the control setting inputs have equal delay paths to the output, and we only consider setting changes that increment or decrement the current setting by one position.

A simplified delay line that is consistent with these assumptions is shown in Fig. 6.21. It contains a single delay element of delay d_r and a 2-input multiplexer. For simplicity, we assume that the multiplexer input-to-output and select-to-output delays are both equal to t_m. The timing diagram on the right shows the conditions that can generate a spurious output transition. The select line needs to transition while the clock edge goes through the delay element. Furthermore, d_r needs to be large with respect to t_m for such a glitch to be formed. In reality, if d_r is reasonably close to t_m the glitch will never form because of low pass filtering. The right design approach is to set up the conditions of Fig. 6.21 and simulate across all PVT conditions to ensure that the glitch won't form.

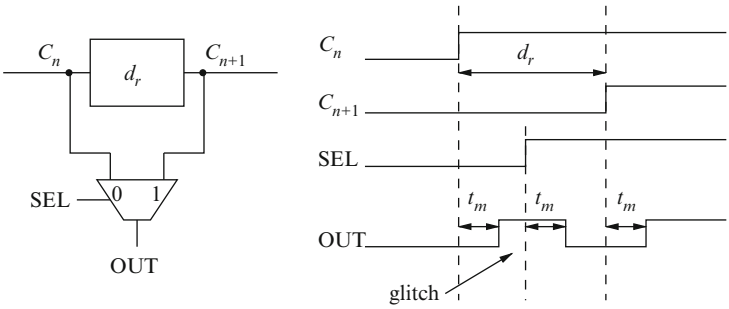


Fig. 6.21. DCDL spurious output transition

Interestingly enough, none of the DCDLs in Figs. 6.16 through 6.20 are glitch-free. Their setting vectors do not have equal delays to the output, and therefore the simple analysis of Fig. 6.21 does not apply. They have a lot more exposure to spurious transitions, and their analysis is rather ad-hoc and can be quite cumbersome. It is safe to create a design restriction stating that *only coarse DCDLs with equal setting-to-output delays should be considered when glitch free operation is necessary*. Distributed and logarithmic multiplexer structures are not allowed.

The simple DCDL of Fig. 6.18 can be a candidate for glitch-free operation if the output logarithmic multiplexer is replaced by a one-hot NAND-based (sum-of-products) multiplexing structure where each control line is at the same logic depth. Another potential solution is to pad the most significant bit with additional delay and the problem transforms to heterogeneous structure delay tracking across PVT and can be rather process-specific.

The coarse DCDLs of Figs. 6.16 through 6.20 are typically used in applications where the clock is not utilized while settings transition. Alternatively, they can be used in a synchronous fashion. The control inputs must be timed in a way that guarantees that when they change, the entire delay chain is at a constant value. This can be accomplished by latching the settings with an appropriate strobe which can either be the input clock or a delayed version of it (perhaps a particular delay chain tap) that meets this timing restriction. Extensive validation is required.

They can also be used asynchronously by adding significant hardware resources and control complexity. Such an example is shown in Fig. 6.22. We have a pair of identical delay lines. Only one is on-line at any single time. When there is a settings change, it is performed on the off-line DCDL. When the change has stabilized and all spurious transitions are gone, the output multiplexer is flipped and the new settings change shows up at the output. At this point, the analysis of Fig. 6.21 applies. In addition to consuming large hardware resources, this structure has a very subtle issue. It is based on the assumption that if we have two instantiations of a delay line on silicon and one has a setting equal to n and the other has a setting equal to $n+1$, then the one with the $n+1$ setting will have a longer delay. This may not be true in deep submicron processes due to large random V_t variation. Monte Carlo analysis can show a substantial probability of such a case. If this happens, then the composite

delay line is non-monotonic. This will affect performance, and under certain circumstances, it may confuse the control automaton and cause catastrophic failure. A better approach is shown in Fig. 6.23 where a single dual output delay line is used. It uses less hardware and does not have potential monotonicity issues. It has the same control complexity though.

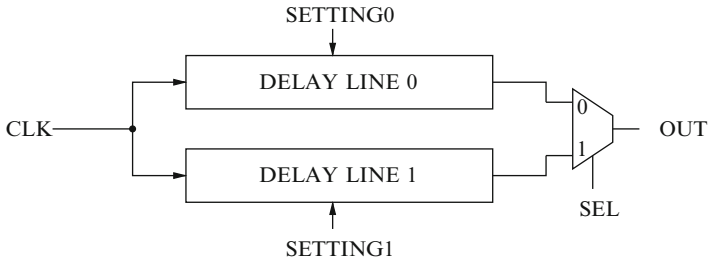


Fig. 6.22. Duplicating DCDLs for glitch suppression

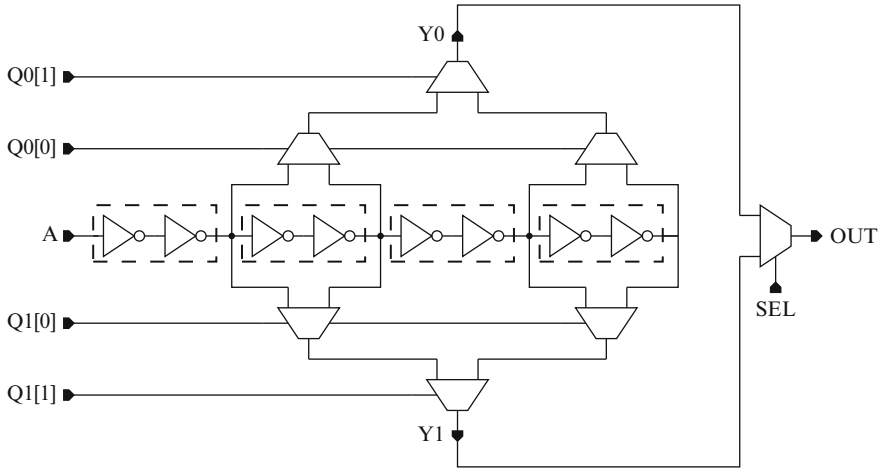


Fig. 6.23. Dual output DCDL for glitch suppression

6.5.2 Subgate-Delay DCDLs

Phase lock accuracy requirements can be tighter than a gate delay. In such applications, a subgate-delay (fine) DCDL must be employed.

A class of fine DCDLs relies on variable RC delays for delay generation. Two examples of fine DCDL stages are shown in Fig. 6.24. The delay stage on the left is based on variable cell driving resistance. Turning on more device branches

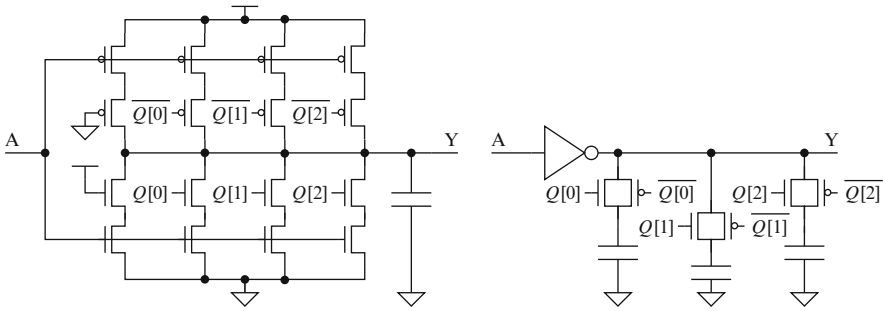


Fig. 6.24. RC-Based fine DCDL stages

through control vector $Q[2:0]$ reduces the overall RC delay through the stage. Transistors in the variable branches are sized for linear control vector encodings. Control bits $Q[2:0]$ are thermometer encoded and have a fixed switching order (i.e., $000 \rightarrow 001 \rightarrow 011 \rightarrow 111$). Logarithmic encoding is not possible due to the non-linearity of parallel resistance addition. Table 6.6 shows a normalized example of branch sizing for linearity. The example should be considered as a starting point only and process specific fine tuning will be required. The number of variable branches is limited by the extreme sizing that will be required to maintain linearity for an increasing number of control bits. The dynamic range can be extended by cascading multiple such stages at the expense of increasing $D_{\min}$.

Table 6.6. Variable resistance DCDL branch sizing (thermometer-encoded control)

	Normalized width	Equivalent resistance	Capacitance	RC delay
Always-on branch (AO)	1	1	4	4
Branch 0 + AO	1/3 + 1	3/4	4	3
Branch 1 + 0 + AO	2/3 + 1/3 + 1	1/2	4	2
Branch 2 + 1 + 0 + AO	2 + 2/3 + 1/3 + 1	1/4	4	1

The stage on the right is based on variable output capacitance. Turning on more capacitive branches increases the RC delay through the cell. Both linear (thermometer) and logarithmic (binary) control encoding are possible due to the linearity of parallel capacitance addition. Output capacitors can be metal-based or device-based depending on accuracy and linearity requirements. Per-stage dynamic range is limited by the maximum edge rate tolerated by the design. The dynamic range can be extended by cascading multiple stages as in the previous case.

An alternative method of constructing fine delay lines is shown in Fig. 6.25. It is based on the delay difference between the top vs. the bottom delay path. Control encoding is thermometer-based. This organization can achieve PVT-independent lin-

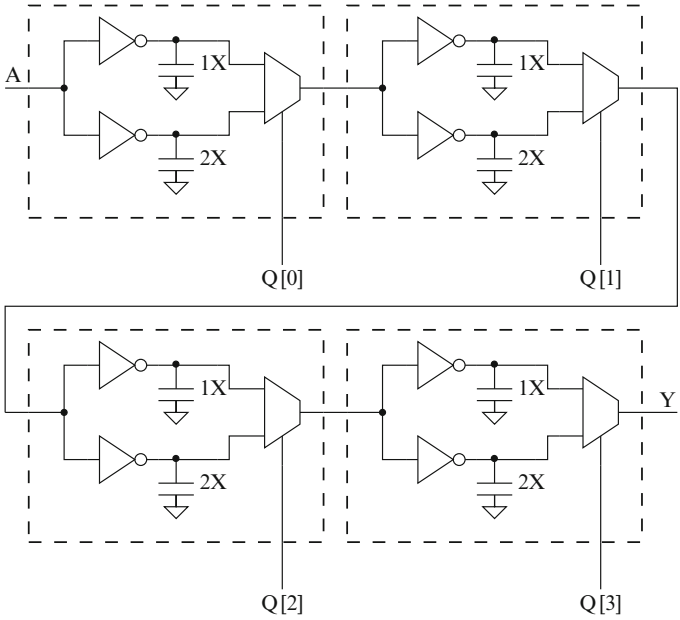


Fig. 6.25. Fine DCDL based on delay differences

earity subject only to random process variation. The main drawback is that both linearity and dynamic range are achieved through large $D_{\min}$ increases which can be a problem for certain applications.

An additional method of constructing subgate delays based on phase interpolation will be discussed in Sect. 6.5.3.

The delay lines of Figs. 6.24 and 6.25 can have resolution on the order of a few picoseconds but a very limited dynamic range. They can be operated asynchronously without the possibility of a glitch forming at the output. The DCDLs of Fig. 6.24 vary resistance and capacitance. Settings changes are not considered combinational logic switching events. On the other hand, the multiplexer-based DCDL of Fig. 6.25 can be analyzed using the method of Fig. 6.21 with a much smaller d_r.

6.5.3 Resolution vs. Dynamic Range in DCDLs

In typical applications, both fine resolution and large dynamic range are highly desirable. High resolution directly affects phase match accuracy. A large dynamic range ensures that the DLL won't be the limiting factor in selecting input clock frequencies or V_{MIN} for the overall design.

From the previous analysis on DCDLs, it would seem that designers must perform a tradeoff between resolution and range. This is rarely true in real applications since both are essential. Instead of using a single delay line that compromises d_r in

favor of dynamic range for a given number of settings, multiple delay lines can be employed with different properties so that the overall design target is met [13]. Typically, a coarse high dynamic range DCDL is cascaded with a fine low dynamic range subgate DCDL to achieve both specifications. This design decision involves additional complexity in the control automaton which must first lock the coarse line to an appropriate setting (which tracks input clock frequency and static process and voltage variations) and then engage the fine delay line which tracks dynamic voltage and temperature variations. Depending on control implementation, the coarse DCDL can readily be of the synchronous type (as described in Sect. 6.5.1) if it is locked only once and settings never change while in mission mode. In this case, the designer must ensure that the fine delay line has enough dynamic range on either side of the lock point to guarantee phase lock maintenance in the presence of voltage and temperature ramps spanning the entire product voltage-temperature range in the worst possible scenario. If this is not possible, the control algorithm must ensure repeated engagement of the coarse DCDL to compensate for large V-T ramps without the possibility of a large phase change affecting system functionality. Such an example is a DDR memory incoming strobe phase lock system which engages the coarse DCDL only when there is no memory transaction present on the bus and, therefore, no one is looking at the data strobe. While memory reads or writes are in progress, only the fine DCDL is engaged which should guarantee DLL-induced deterministic jitter on the order of a few picoseconds and no spurious clock transitions. An additional concern with cascaded DCDLs is increased D_{min} which can be shown to be prohibitive for certain applications since it directly affects the overall minimum supply voltage allowed (V_{MIN}).

There exists a powerful method of achieving both low d_{r} (high resolution) and arbitrarily high dynamic range with low D_{min} and without cascading DCDLs. It is based on the "ping-pong" DCDL arrangement of Figs. 6.22 and 6.23 with a phase interpolator as the output stage instead of the 2-to-1 multiplexer. A phase interpolator receives two input clocks with a phase difference on the order of 1-2 gate delays and a digital setting value. For a setting equal to zero, the interpolator output simply delays the early input phase by a fixed amount. For a setting equal to the maximum possible value, the interpolator output delays the late input phase by the same amount. For any intermediate setting, the interpolator produces an output clock with a phase proportional to the applied setting as measured using the output at setting 0 as a reference.

A sample schematic is shown in Fig. 6.26. The two separate branches are implemented with tri-state inverters controlled by complementary settings. Typically, the output is restored with an output inverter (not shown) which also makes the overall structure non-inverting. For a first order analysis, we will use the methodology and terminology established in [14] for current-mode reduced-swing structures. The time delay between the two interpolated phases is Δt, and it will be used as a parameter throughout the analysis. We make the following assumptions:

1. The overall output resistance of a branch when fully enabled is R and has no voltage dependence.

2. The interpolation setting can be represented by a continuous variable w where $0 \leq w \leq 1$. When $w = 1$, the early phase is fully enabled, and the late phase does not affect the output. When $w = 0$, the early phase is disabled and only the late phase controls the output. For any other value, both the early and the late phase affect the output delay with weights equal to w and $1 - w$, respectively.

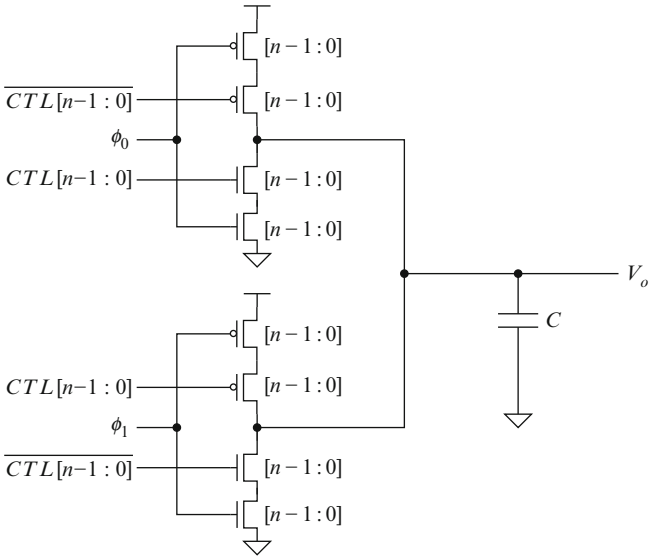


Fig. 6.26. Full swing phase interpolator ($\log_2 n$-bit control)

The interpolator time constant RC (where R is the on-resistance of a fully enabled branch and C is the output capacitance in Fig. 6.26) is a fundamental property which will affect transfer function linearity as will be shown. Without loss of generality, we assume that the interpolator is mixing two falling edges (ϕ_0 and ϕ_1) so that the output waveform is rising. Phase ϕ_0 occurs at time 0 and ϕ_1 occurs at time Δt. For $0 \leq t < \Delta t$, the equivalent circuit is shown in Fig. 6.27a. Writing KCL at V_o yields equation:

$$\frac{dV_o(t)}{dt} + \frac{1}{RC}V_o(t) - \frac{wV_{DD}}{RC} = 0, \tag{6.31}$$

which has the following solution assuming $V_o(0) = 0$:

$$V_o(t) = wV_{DD}(1 - e^{-\frac{t}{RC}}). \tag{6.32}$$

For $t \geq \Delta t$, both branches pull high and resistors R/w and $R/(1-w)$ are connected in parallel. The equivalent circuit is shown in Fig. 6.27b. The governing equation is:

$$\frac{\mathrm{d}V_o(t)}{\mathrm{d}t} + \frac{1}{RC}V_o(t) - \frac{V_{\mathrm{DD}}}{RC} = 0, \tag{6.33}$$

with the following well-known solution (assuming zero initial conditions):

$$V_o(t) = V_{\mathrm{DD}}(1 - \mathrm{e}^{-\frac{t}{RC}}). \tag{6.34}$$

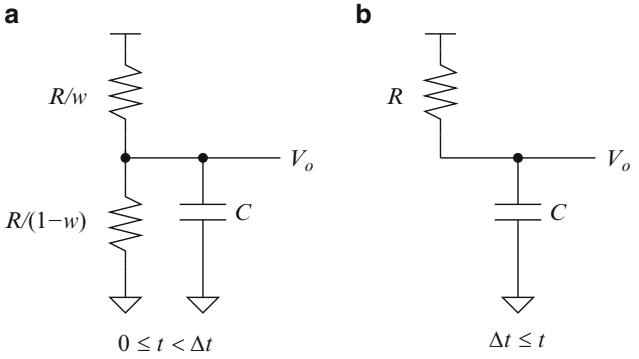


Fig. 6.27. Phase interpolator equivalent circuits

The overall solution is a composite waveform that consists of Eq. (6.32) for $0 \leq t < \Delta t$ and Eq. (6.34) for $t \geq \Delta t$ time-shifted by Δt and adjusted by the initial condition established by Eq. (6.32) at $t = \Delta t$:

$$V_o(t) = wV_{\mathrm{DD}}(1 - \mathrm{e}^{-\frac{t}{RC}})u(t)u(\Delta t - t) + V_{\mathrm{DD}}[1 - (1 - w + w\mathrm{e}^{-\frac{\Delta t}{RC}})\mathrm{e}^{-\frac{t-\Delta t}{RC}}]u(t - \Delta t), \tag{6.35}$$

where $u(t)$ is the unit step function.

Figure 6.28 plots Eq. (6.35) for various Δt using the interpolator weight w as a parameter. It is obvious from the $V_{\mathrm{DD}}/2$ crossing point that linearity with respect to w is a strong function of Δt. The larger Δt is with respect to the interpolator time constant RC, the larger the deviation from linearity with respect to interpolation weight w. This is more clearly shown in Fig. 6.29 that shows interpolator delay transfer function with respect to interpolation weight w using Δt (delay between two input phases) as a parameter. The top graph shows absolute inteprolator delay as a function of w adjusted so that the delay for $w = 1$ (early phase only affecting the output) is zero. The bottom graph shows the same data but normalized to the same $0-1$ range. Linearity is quite good for $\Delta t \ll RC$ but deteriorates quickly for increased spacing between input phases ϕ_0 and ϕ_1. The interpolator time constant RC is a fundamental property that has a strong effect on its linearity. Larger time constants are

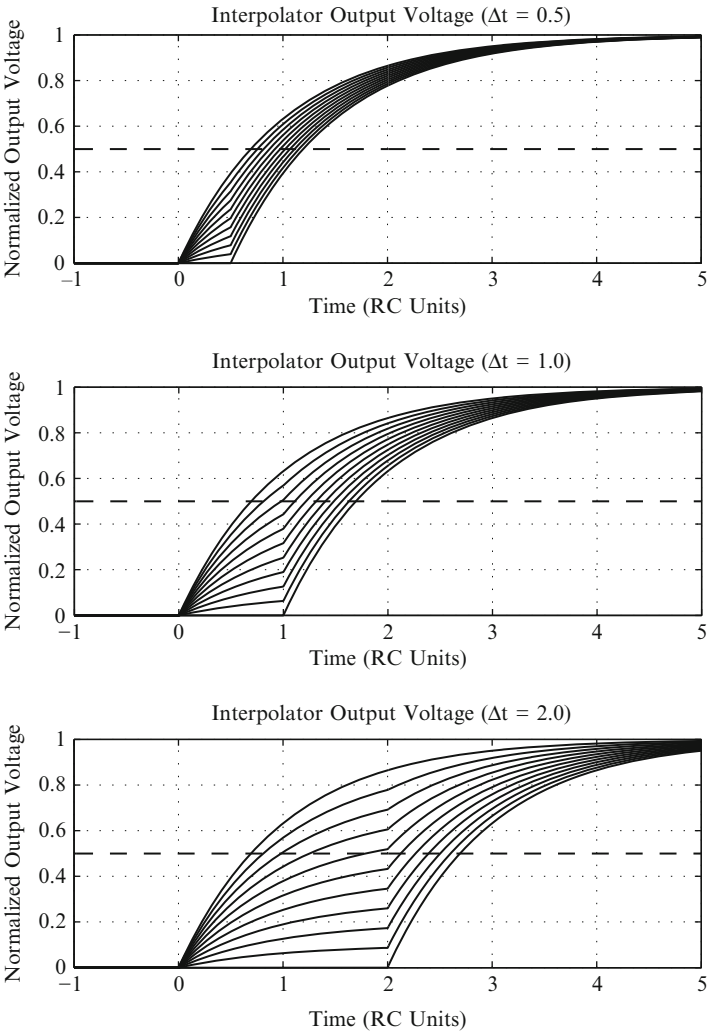


Fig. 6.28. Phase interpolator normalized voltage output for $\Delta t = 0.5, 1, 2$ (w is decreasing from 1 to 0 in steps of 0.1 from left to right in all three plots)

desired if linearity is important. An effect similar to an increased interpolator *RC* can be achieved by slowing the edge rates of input phases ϕ_0 and ϕ_1.

An interpolator based DCDL can achieve both high resolution and high dynamic range at the expense of increased control complexity. Such a DCDL can easily be designed to be free from spurious output transitions. When a coarse setting changes that affects one input of the interpolator, the interpolator weight should be such that only the other non-changing input affects the output.

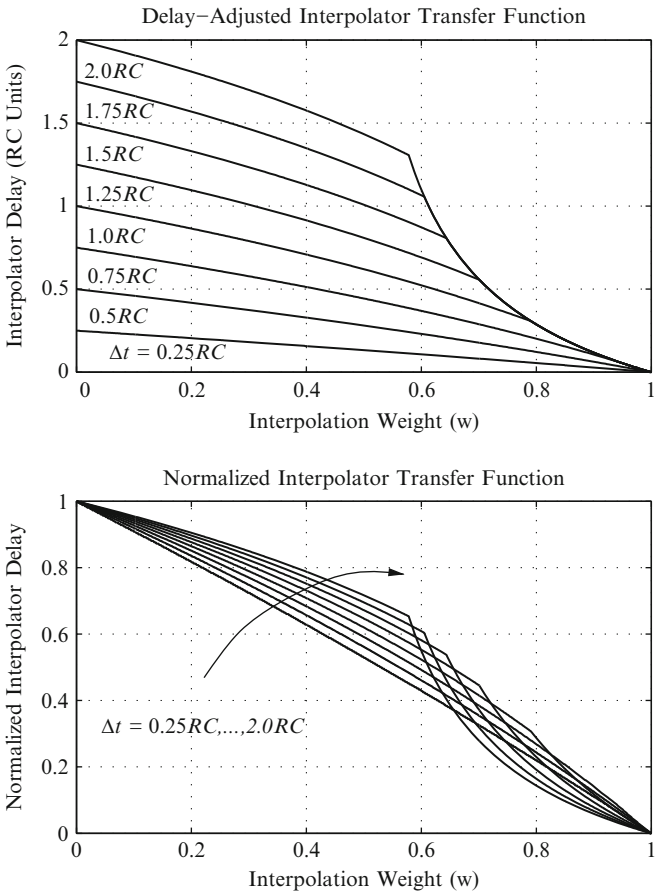


Fig. 6.29. Phase interpolator transfer function for varying Δt. Top graph reproduced in a form similar to that in [14], ©1998 Stefanos Sidiropoulos

6.6 Control

There is one major classification that needs to be made regarding digital DLL control: first order and higher order. A first order digital DLL contains a single DCDL state element (counter or shift register) that maintains the delay line setting and gets updated through some mechanism based on the phase difference between reference and controlled clock. In contrast, a second order DLL will have a second counter which maintains some representation of the *frequency* or rate of phase change between the two clocks. The DCDL is then adjusted by combining both such state variables. All DLL examples in this chapter assume that the reference clock at the phase detector and the source clock of the DCDL are the same. Therefore, there is no possibility of a frequency difference between the two inputs of the phase detector. An application

where a second order control loop would make sense is clock-data recovery, which is briefly described in Sect. 6.9.4. The rest of this section is only concerned with first order control and identical frequencies of reference and feedback clocks.

A typical DLL control block contains a state structure that holds the DCDL settings and a mechanism to update the settings based on input from the phase detector. The state element can be a binary up/down counter if the DCDL has binary encoding. Different DCDL setting encodings can be easily handled with the addition of combinational decoding logic past the main counter. The designer should always keep in mind that post-flop combinational logic can generate spurious transitions and create more conditions for glitches at the output of the DCDL. For one-hot DCDL configurations, it is more common to use a bidirectional shift register which requires no decoding and will produce glitch free control outputs.

The simplest control arrangement is shown in Fig. 6.30a. It involves a single up/down counter (or bidirectional shift register) under the direct control of the phase detector. When the phase detector evaluates to a 1 (controlled clock faster with respect to the reference), the counter is incremented and the DCDL has a larger (slower) setting. When the PD evaluates to a 0, the counter is decremented and the DCDL speeds up. A more general control arrangement is shown in Fig. 6.30b. In this case, a Finite State Machine (FSM) is inserted between the phase detector output and the counter increment/decrement. The FSM will make setting change decisions based not only on the phase detector output but also based on internal state. In this fashion, more complex control algorithms can be implemented which can have a strong impact on DLL capabilities such as sensitivity to initial conditions, dynamic range, and stability.

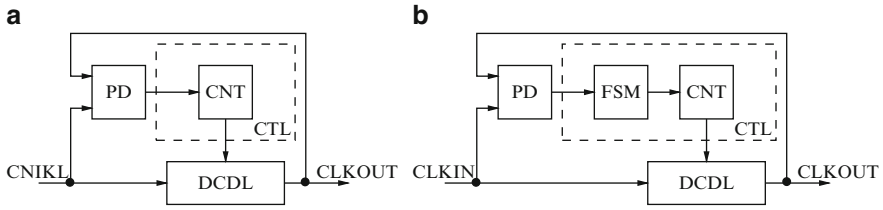


Fig. 6.30. DLL control options

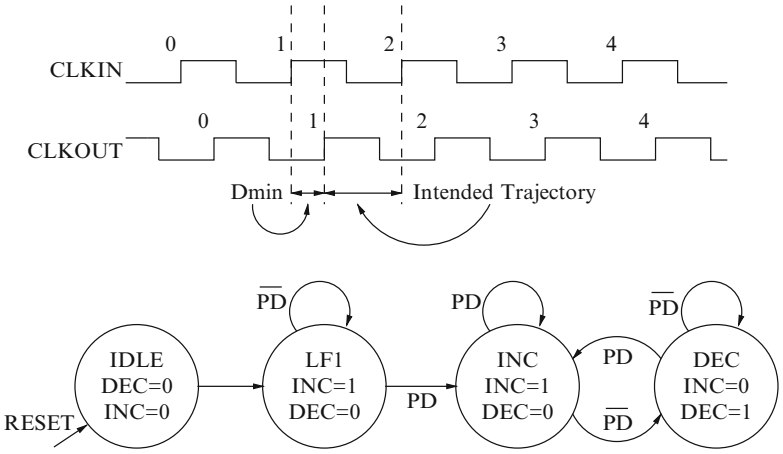
6.6.1 Sensitivity to Initial Phase

Even though a DLL may contain a DCDL with a large dynamic range, a simplistic control design may result in an inability to achieve phase lock. An example is shown in Fig. 6.31. Let us assume that the DLL configuration is the one in Fig. 6.30a with $D_{\min} < 0.5T$ where T is the CLKIN period. The waveforms right after system reset (zero DCDL setting) are shown on the top part of Fig. 6.31. The stateless control design of Fig. 6.30a will attempt to push the DCDL toward the left and match edge 1 of CLKIN with edge 1 of CLKOUT. If the settings counter wraps around, a very large

delay will be added to the DCDL and the DLL will either lock to an undesirable setting (not the minimum delay setting that will achieve phase lock) or will degenerate to an oscillation between $D_{\min}$ and $D_{\max}$ without ever achieving lock. If the settings counter is designed not to wrap-around but saturate at the minimum setting, the DCDL will stay fixed at $D_{\min}$ and again phase lock will never be achieved.

If on the other hand the FSM of Fig. 6.31 (*bottom*) is inserted between the phase detector and the settings counter, lock can be easily achieved. The FSM will keep on incrementing the DCDL settings until a 1-to-0 transition is observed at the phase detector independent of the initial state of the phase detector output. Such a transition indicates a lock condition and the state machine will be bouncing back and forth between states INC and DEC. Edge 1 of CLKOUT will be aligned to edge 2 of CLKIN.

A word of caution is absolutely essential when a state machine similar to the one depicted in Fig. 6.31 (*bottom*) is used. This particular FSM is designed to disregard a 0-to-1 transition from the phase detector and essentially lock the system when a 1-to-0 transition is observed. In reality, a 1-to-0 transition can easily be observed right after a 0-to-1 when the controlled clock (CLKOUT) phase traverses a negative edge of the reference clock (CLKIN). When the positive edge of CLKIN is aligned to the negative edge of CLKOUT the phase detector can undergo multiple transitions due to internal factors such as a large dead zone where its output is undefined or external factors such as supply noise and reference clock jitter. If state DEC is entered (Fig. 6.31) while we have positive to negative edge alignment, the system will never



RESET: State Machine Reset
PD: Phase Detector Output (1 = CLKOUT leads, 0 = CLKOUT lags)
INC: Increment DCDL Settings Counter (Active High)
DEC: Decrement DCDL Settings Counter (Active High)

Fig. 6.31. DLL FSM example for initial condition flexibility

achieve lock but will likely be pushed back to its reset state. State DEC must only be entered when a positive CLKOUT edge has just started to lag a positive CLKIN edge. Otherwise, the phase detector decisions won't be interpreted correctly and the DCDL will be moved to the wrong direction.

There are multiple ways that this issue can be addressed. Digital low pass filtering of the phase detector output can be employed to remove spurious transitions. This method can reduce the probability of a false lock, but it is hard to show that it can totally eliminate it. A more robust method is to insert an intermediate state between LF1 and INC in which the controller will reside for n clock cycles. In this state, the DCDL setting is incremented n times independent of the phase detector output. In this way, we can guarantee that when we have negative edge traversal, the DCDL will be incremented enough to push CLKOUT out of the phase detector sampling window or dead zone (d_{sw}) given worst case supply noise and jitter conditions. The number n should be large enough to guarantee $n \cdot d_{\mathrm{r}} \gg d_{\mathrm{sw}}$ and at the same time small enough to guarantee $n \cdot d_{\mathrm{r}} \ll 0.5T$ (where T is the reference clock period) so that the next positive CLKIN edge is not traversed and the phase detector output does not change interpretation. This should be a rather loose constraint for most applications.

6.6.2 Dynamic Range Increase

It is possible to increase the dynamic range of a DLL with the addition of little extra hardware and incremental complexity in the control automaton. This method assumes that the DCDL output can be conditionally inverted (i.e., XOR output stage) and that the settings counter or bidirectional shift register can provide a carry out (CO) output to the FSM indicating that its maximum value has been reached. The concept is very simple. If the DLL reaches the maximum value of the settings counter without achieving lock (the phase detector output has not undergone a 1-to-0 transition), then the DCDL output is inverted, the counter is cleared and the state machine goes back to the initial state and attempts to lock the inverted DCDL. The inversion has the effect of adding 180° of phase and virtually doubles the DLL dynamic range. This is shown conceptually in Fig. 6.32 (*top*). An example FSM for this configuration is shown at the bottom of the figure.

6.6.3 Stability and Bandwidth

At the beginning of this chapter, a claim was made that a digital DLL can be unconditionally stable. In this section, we qualify this claim. The stability of feedback systems is typically analyzed in the frequency domain. A frequency domain model of a digital DLL is non-trivial and not particularly insightful due to the nonlinearity of a bang–bang phase detector and the sampled nature of the system. Instead, we choose to revisit the analysis of a linear (analog) delay locked loop, establish stability conditions and apply the intuition developed to the non-linear digital counterpart. We follow the methodology and notation of Maneatis in [15]. A very similar approach is also presented by Yang in [16].

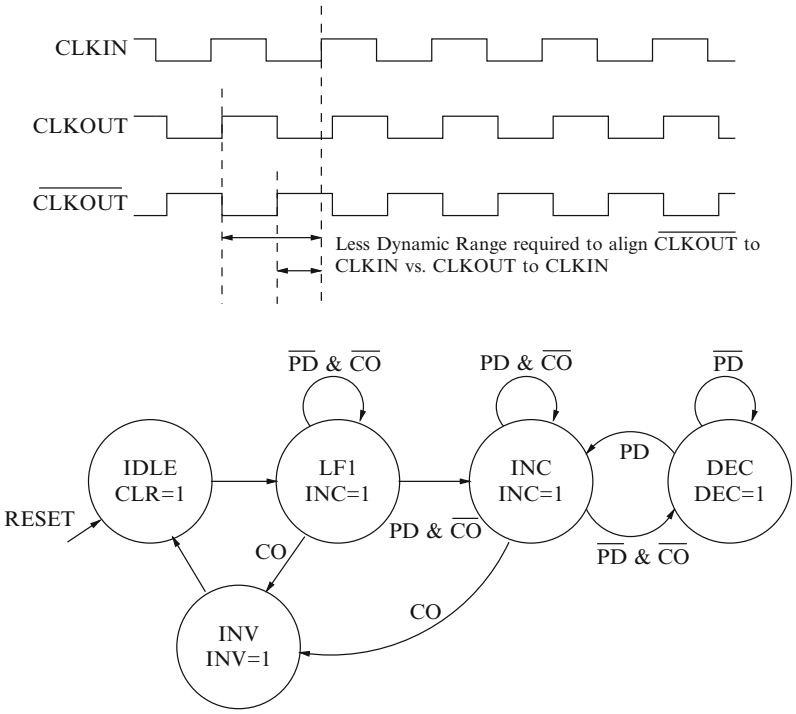


Fig. 6.32. Doubling DLL dynamic range using conditional inversion

Figure 6.33a shows a simple charge-pump based analog DLL. The phase detector is linear rather than bang–bang (Sect. 6.4). Its output is a pulse whose width is proportional to the difference between the CLKIN and CLKOUT phases. The proportional pulse controls a charge pump which integrates a constant current on capacitor *C*. The amount and sign of the integrated current (charge) depend on the amount and sign of the phase difference between the clocks. The capacitor voltage controls an analog delay line which in turn will adjust until CLKOUT matches CLKIN. There are many similarities between this analog DLL and the digital counterpart of Fig. 6.1. The settings counter/accumulator inside the control block performs the same action as the capacitor and essentially integrates the phase detector output. The main difference is that the digital DLL is a non-linear system because the correction in response to the phase error is not proportional but instead has a constant slew rate.

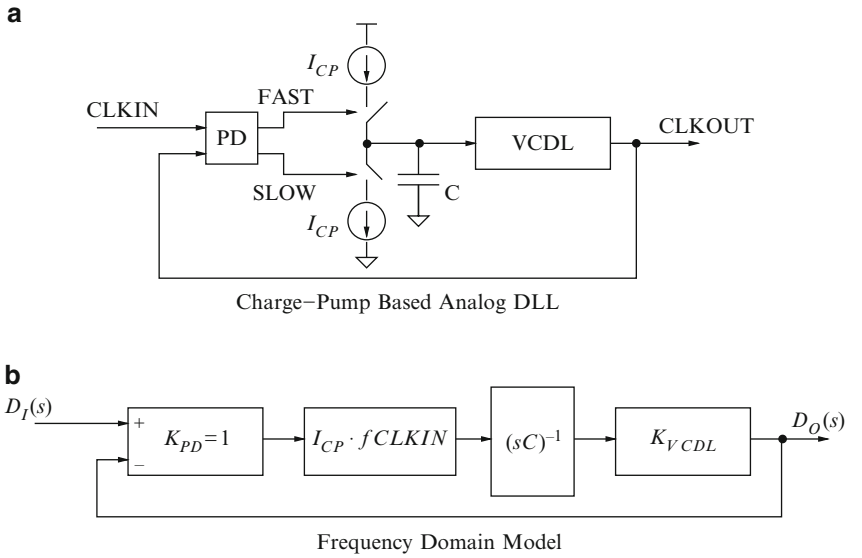


Fig. 6.33. Analog DLL frequency domain model

An additional important difference is that a digital DLL may have more delay around the loop due to inherent clock delays in the FSM controller and settings counter. This can be very important for stability and will be addressed in more detail later on.

A frequency domain model of the analog DLL is shown in Fig. 6.33b. The model terminal variables are $d_{\mathrm{I}}(t)$ (input delay), which is defined as the delay of input CLKIN with respect to an arbitrary point. The output delay of CLKOUT $d_{\mathrm{O}}(t)$ is also defined with respect to the same arbitrary point. The corresponding Laplace transforms are $D_{\mathrm{I}}(s)$ and $D_{\mathrm{O}}(s)$. The frequency domain representation of the phase detector is a simple subtractor with a constant gain of 1 since it generates a pulse with a width equal to the delay difference between CLKIN and CLKOUT. The units at the output of the phase detector are seconds (delay). The charge pump model is also a constant gain since it multiplies the proportional error pulse with a constant current after normalizing it to the CLKIN period ($I_{\mathrm{CP}} \cdot f_{\mathrm{CLKIN}}$, where f_{CLKIN} is the CLKIN frequency). The units at the output of the charge pump are Amperes (current). The charge pump capacitor C acts as an integrator with a transfer function $(sC)^{-1}$. The units at the integrator output are Volts. Finally, the Voltage-Controlled Delay Line (VCDL) is also modeled with a constant gain (K_{VCDL} s/V). The output variable $D_{\mathrm{O}}(s)$ has units of seconds (delay). Writing the loop equation yields:

$$D_{\mathrm{O}}(s) = [D_{\mathrm{I}}(s) - D_{\mathrm{O}}(s)] \cdot I_{\mathrm{CP}} \cdot f_{\mathrm{CLKIN}} \cdot \frac{K_{\mathrm{VCDL}}}{sC}. \tag{6.36}$$

Further manipulation yields the following transfer function [15]:

$$\frac{D_{\mathrm{O}}(s)}{D_{\mathrm{I}}(s)} = \frac{1}{1 + \frac{s}{\omega_N}}, \tag{6.37}$$

where

$$\omega_N = \frac{I_{\mathrm{CP}} \cdot K_{\mathrm{VCDL}} \cdot f_{\mathrm{CLKIN}}}{C}. \tag{6.38}$$

The transfer function of Eq. (6.37) is of first order with a single pole at ω_N (loop bandwidth) and is unconditionally stable with a phase margin of 90°. This is true as long as the delay around the loop is virtually instantaneous and the phase margin is not reduced considerably. Even an analog DLL though is a sampled system and there is loop delay due to the sampled nature of the phase detector: A phase detector measurement won't be taken until the next positive clock edge, thus, reducing the phase margin at unity gain. Such margin reduction though is negligible if $\omega_N \ll 2\pi f_{\mathrm{CLKIN}}$. For all practical purposes, the loop delay can be considered virtually instantaneous if such delay is much lower than the DLL response time. Maneatis [15] sets the unconditional stability criterion at $\omega_N = 2\pi f_{\mathrm{CLKIN}}/10$.

Multiple authors [17, 18] have pointed out that the simple frequency domain model of Fig. 6.33 does not apply for the class of delay locked loops where the same reference clock is driving both the phase detector and the DCDL. Figure 6.33 models the delay line as a 2-terminal block (control input and clock output) and does not include a feed-forward path from the input reference clock to the output through the delay line. As a result, such a model cannot accurately model jitter transfer from input to output clock. The stability analysis changes due to the introduction of a zero in the first order model. However, unconditional or conditional stability can still be shown with root locus methods depending on model choice [18].

The main difficulty in applying a similar analysis to the digital DLL lies in the nonlinear transfer function of the phase detector (Fig. 6.2). The linear frequency domain analysis is not applicable and the steady state response of such a DLL will be oscillatory (limit cycle). Prior art [19, 20] has linearized the phase detector response around the dead zone, which is a valid approach for high CLKIN frequencies, high DCDL resolutions, and multiple averaged measurements since the expected phase error will be rather small and comparable to d_{sw}. For lower frequency applications though, the phase detector will mostly operate in its nonlinear regime.

Before proceeding, let us first define what we mean by stating that a non-linear DLL is stable. We know for a fact that a DLL with a non-linear (bang–bang) phase detector will exhibit limit cycle behavior due to the binary nature of the phase detector and its inability to encode a zero phase error. For the purposes of this discussion, let us define DLL stability to mean that the expected limit cycle should have by design the minimum possible amplitude of $\pm d_{\mathrm{r}}$. Frequency domain transformation is not really necessary to assess the stability of such a simple system. The time-domain behavior of the digital DLL (Fig. 6.1) in the locked state can be summarized by the following difference equations:

$$e[n] = \mathrm{sgn}(d_{\mathrm{O}}[n] - d_{\mathrm{I}}[n]), \tag{6.39}$$

$$d_{\mathrm{O}}[n+1] = d_{\mathrm{O}}[n] + e[n]d_{\mathrm{r}}, \tag{6.40}$$

where $e[n]$ is the error computed by the phase detector ($e[n] \in \{-1, 1\}$), $d_I[n]$ is the input delay at discrete time n (nth CLKIN positive edge), $d_O[n]$ is the output delay at discrete time n and d_r is the DCDL resolution. We establish a time-domain stability criterion as follows: *The correction step in Eq. (6.40) at discrete time n should not produce an error ($d_O[n+1] - d_I[n+1]$) at $n+1$ of the same sign and greater magnitude than ($d_O[n] - d_I[n]$) at time n.* This criterion is easily satisfied if the correction term $e[n]d_r$ in (6.40) does not result in crossing over two phase boundaries which will produce a phase error of the same sign ($mod T_{CLKIN}$) and potentially larger magnitude. If $d_r < 0.5T_{CLKIN}$, this should never happen and this constitutes a constraint that is always met for all practical purposes. A DLL with such a large d_r would be highly impractical.

We now focus on the loop delay argument. Eqs. (6.39) and (6.40) do not capture the effect of excessive loop delay and the possibility of reduced "phase margin". A digital DLL has the potential of introducing substantial loop delay through its control mechanism and may turn negative feedback into positive. The reasons for the increased delay can be multiple:

- Additional flip-flop synchronization stages past the phase detector to ensure low probability of metastability.
- Low pass filtering of the phase detector output to remove spurious transitions and potentially emulate "ternary" error detection (fast, slow, NOP).
- Deserialization latency in a CDR (clock-data recovery) loop where phase detection occurs in the divided clock domain on the multiple deserialized bits.
- Pipeline stages in the FSM controller and settings counter.

Let us introduce two more general descriptive DLL design parameters (in addition to d_r, d_{sw}, D_{min} and D_{max}) to help with the analysis of this issue:

N_d indicates the number of CLKIN delay cycles from the input of the phase detector to the setting input of the DCDL. This number is an integer with a minimum value of 1 (delay introduced by the bang–bang phase detector).

N_{bw} is the inverse of the rate at which the FSM controller issues corrections to the DCDL (once every N_{bw}th cycle). One can think as $1/N_{bw}$ as the "bandwidth" of this nonlinear system. The minimum value for N_{bw} is 1 (controller issues corrections on every cycle) and the maximum value can be arbitrarily high. $1/N_{bw}$ is also referred to as the DLL sampling rate because it signifies the normalized frequency of looking at the phase detector output.

We introduce a second stability criterion which states that a digital DLL is stable if $N_d < N_{bw}$. This is the equivalent of stating that a *a digital delay locked loop is stable if it won't issue a DCDL correction before the outcome of the previous correction is fully evaluated by its control mechanism.*

We demonstrate this criterion with a cycle-accurate behavioral model of a DLL with the characteristics of Table 6.7 running at 250 MHz.

The controller is equivalent to the FSM of Fig. 6.31 with the addition of wait states between the states that update the counter to implement $N_{bw} \neq 1$. The modeled

Table 6.7. DLL behavioral model design parameters

$D_{\min}$	1.525 ns	$D_{\max}$	14.275 ns
d_{r}	0.050 ns	d_{sw}	0.0 ns
N_{d}	parameter	N_{bw}	4

DLL has a built-in N_{d} of 3 (phase detector plus 2 stages in the FSM/counter). We, therefore, make N_{bw} equal to 4 to guarantee stability when N_{d} is minimum. Fig. 6.34 shows simulation results (DLL phase error over time) for various N_{d} values. Case $N_{\mathrm{d}} = 3$ satisfies our stability criterion since the phase error changes sign every N_{bw} cycles. All other cases fail the criterion and the phase error is amplified. Loop delay causes the DLL to issue correction steps in the wrong direction. The number of wrong correction steps is $\lfloor N_{\mathrm{d}}/N_{\mathrm{bw}} \rfloor$ and the resulting phase error is bounded by $\pm(\lfloor N_{\mathrm{d}}/N_{\mathrm{bw}} \rfloor + 1)d_{\mathrm{r}}$ as opposed to $\pm d_{\mathrm{r}}$ in the stable case.

A digital DLL can be stabilized given a value for N_{d} by making $N_{\mathrm{bw}} > N_{\mathrm{d}}$. Such an example is shown in Fig. 6.35 where a DLL with $N_{\mathrm{d}} = 15$ is stabilized by increasing N_{bw} from 4 to 16. An expected increase in lock acquisition time is observed due to the reduced rate of issued DCDL corrections. Lock acquisition will be discussed in more detail in Sect. 6.6.4.

Increasing N_{bw} is analogous to reducing loop bandwidth ω_N in the analog domain. Higher N_{bw} increases loop response time and makes it more difficult to track a changing input. Spread spectrum clocking is one important application where tracking a changing input is necessary. In such a case, N_{bw} should be treated as an important design parameter and the designer must ensure that a changing frequency input won't generate a noticeable increase in phase error. A convenient way of describing the tracking ability of a nonlinear digital DLL is its delay slew rate r defined as follows:

$$r = \frac{d_{\mathrm{r}}}{N_{\mathrm{bw}} \cdot T_{\mathrm{CLKIN}}}. \tag{6.41}$$

The slew rate is a unitless quantity that indicates the amount of delay correction that the DLL can produce per unit time. A simple way to determine whether a DLL can track a given spread spectrum (SS) clock is to calculate an equivalent slew rate r_{s} for the spread clock and compare it with the DLL rate r. SS clocks are typically defined with a spread factor A_{s} as a percentage which indicates modulation amplitude and the modulation frequency f_{s} on top of the clock period T_{CLKIN}. The equivalent slew rate of a spread clock indicates the period change per unit time and is given by the following formula:

$$r_{\mathrm{s}} = 4A_{\mathrm{s}}T_{\mathrm{CLKIN}}f_{\mathrm{s}}. \tag{6.42}$$

As an example, a 250 MHz SS clock with a 10% spread factor and a 500 KHz spread frequency will have an r_{s} of 0.8×10^{-3}. A DLL with d_{r} equal to 0.050 ns and a N_{bw} equal to 4 will have an r of 3.125×10^{-3}. Since $r > r_{\mathrm{s}}$, we determine that the DLL can track the SS clock without change to its theoretical minimum phase error of $\pm d_{\mathrm{r}}$. Phase error tracking is demonstrated in Fig. 6.36 using the same behavioral

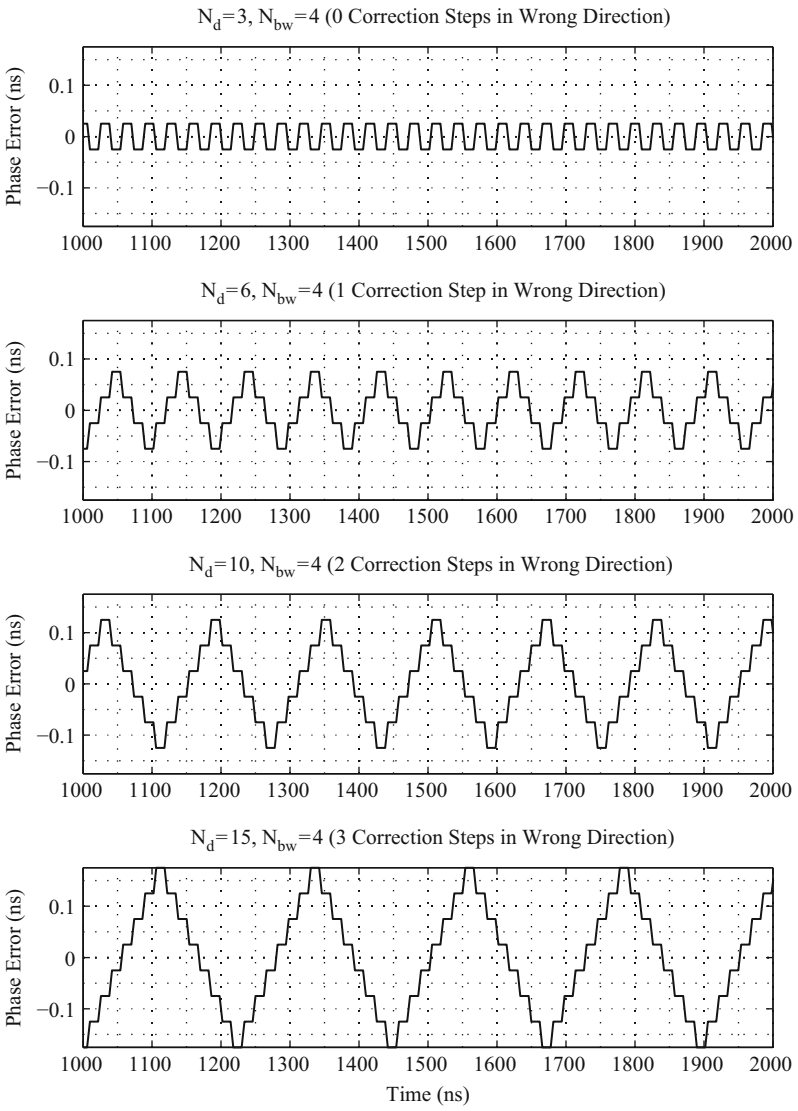


Fig. 6.34. DLL limit cycles as a function of N_d in the locked state

DLL model as past examples in this section. Clock is running at 250 MHz with 10%, 500 KHz spread, N_d is fixed at 3 and N_{bw} is parameterized to demonstrate the slew rate tracking criterion. For $N_{bw} = 4$, $r > r_s$ and phase error is close to the $\pm d_r$ structural value and spread spectrum modulation is tracked. For $N_{bw} = 16$, r is slightly less than r_s and phase error starts to get amplified. Finally, at $N_{bw} = 32$, the DLL has very small bandwidth and is incapable of tracking the SS input. Phase error

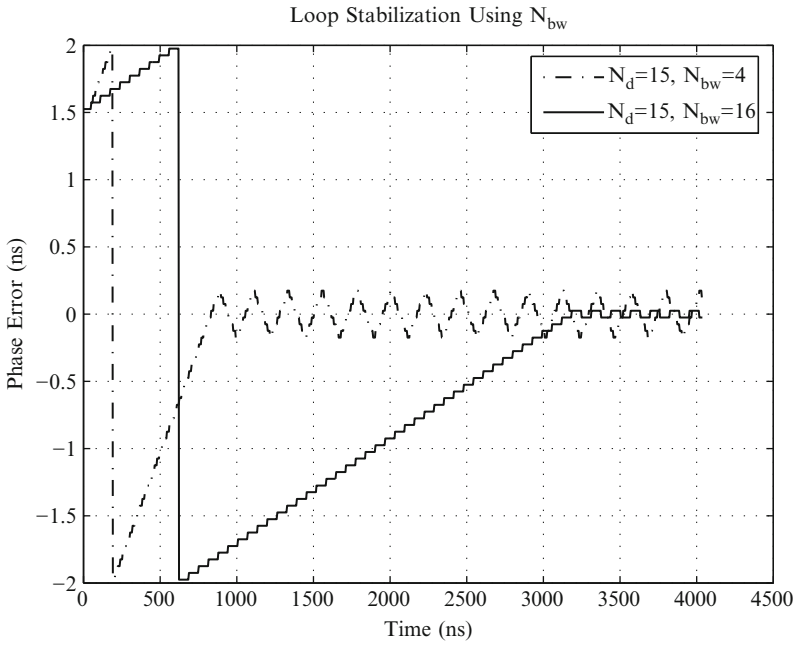


Fig. 6.35. DLL loop stabilization using N_{bw} (DLL is running at $T_{\text{CLKIN}} = 4$ ns)

is very similar to the SS period variation, and there is little difference between this DLL and an open loop delay line matched to the unmodulated input clock period.

It is worth mentioning that this tracking analysis is only relevant if the DCDL is matched to some function of T_{CLKIN}. If the application involves delay-only matching (Fig. 6.4b) then the phase error should not be affected by input clock spreading. All DLL outputs will vary in the same fashion and delay variation will be the same among all outputs resulting in zero additional phase errors.

This section has established two stability criteria and a tracking criterion for digital DLLs. It must be stressed that this analysis applies only to first order digital DLLs as defined in Sect. 6.6.

6.6.4 Lock Acquisition

The lock acquisition procedure is the most open-ended controller aspect from a design perspective. Multiple approaches are possible. A simple controller like the one depicted in Fig. 6.31 will produce lock acquisition profiles similar to the ones shown in Fig. 6.35, where lock is achieved with a constant slew rate equal to $d_{\text{r}}/(T_{\text{CLKIN}}N_{\text{bw}})$. Lock acquisition duration is, therefore, a strong function of N_{bw} selection.

Faster lock acquisition methods are possible such as variable slew rate (higher r during acquisition for lock time reduction and lower r during lock maintenance for stability), binary search [21] and open loop synchronous mirror delay locking

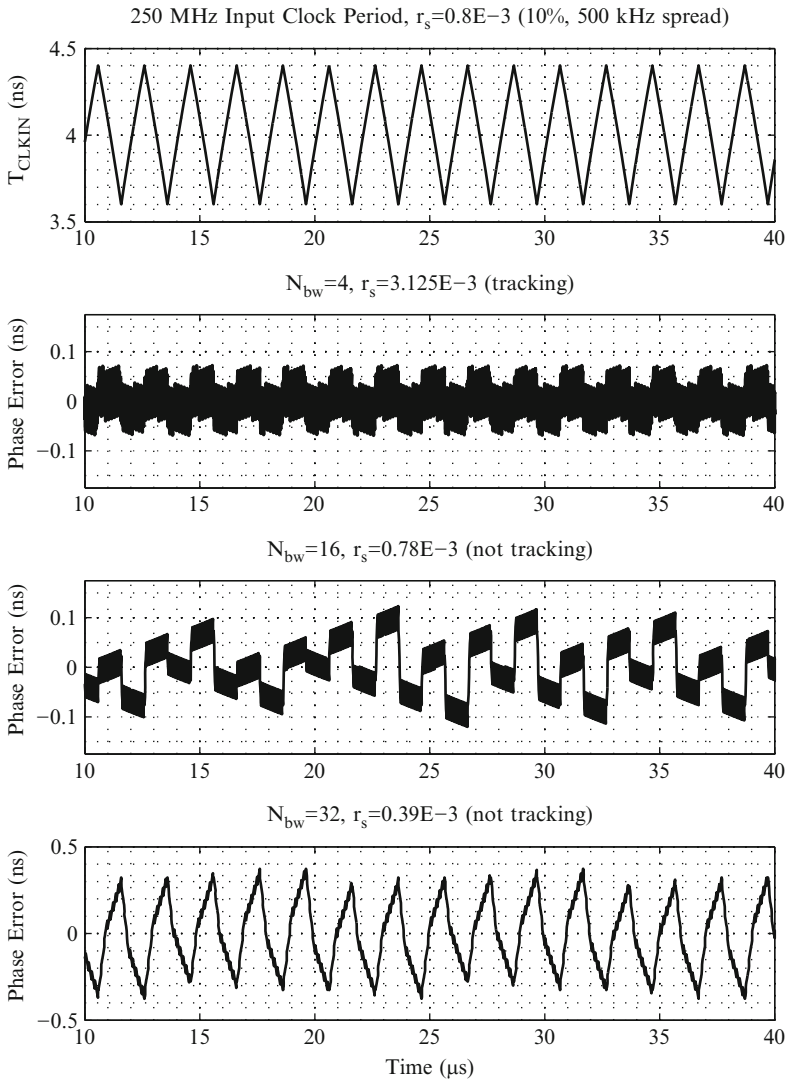


Fig. 6.36. DLL phase error tracking of SS clock as a function of N_{bw}

[22, 23]. The binary search method computes every bit of a logarithmic delay setting from MSB to LSB in a successive approximation fashion in $\log N$ steps where N is the total number of DCDL settings. The synchronous mirror delay method measures the difference between the unknown delay and a full clock period in a single step. This difference is then installed in the DCDL and the system can either revert to regular feedback DLL mode with standard control or remain open loop until the next time a measurement is made. This method is by far the fastest and can achieve delay lock within a few cycles independent of delay size. All of the above lock acquisition

methods are illustrated in Fig. 6.37 using our standard DLL behavioral model ($N_{bw} = 16$) for $T_{CLKIN} = 250$ MHz. The reduction in lock time is rather obvious going from top to bottom.

Fast lock acquisition can be essential for applications that involve clock gating and suspension modes such as memories and low-power/portable systems. Ability to relock quickly as opposed to saving the previous setting before entering the low power state and reinstating it is much more desirable due to voltage/temperature tracking and is mandatory for dynamic voltage/frequency applications.

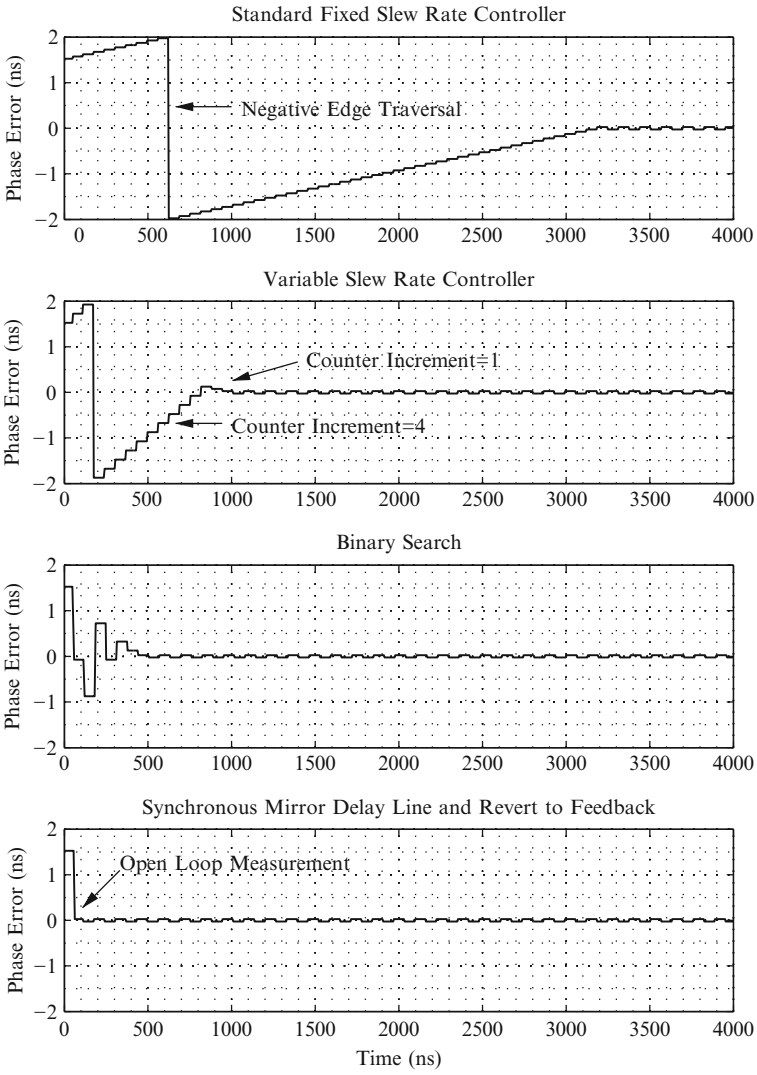


Fig. 6.37. DLL lock acquisition profiles ($T_{CLKIN} = 250$ MHz, $N_{bw} = 16$)

6.7 Putting it All Together

Sections 6.4 through 6.6 have described in considerable detail all DLL system components. In this section, we briefly review important design parameters and outline a basic design flow.

A digital DLL can be generally described by the following design parameters:

d_{sw}	Phase detector sampling window
d_r	Delay resolution
D_{min}	Minimum DCDL delay
D_{max}	Maximum DCDL delay
N_d	Control mechanism delay
N_{bw}	Inverse of DCDL correction issue rate

Phase detector design/selection should always minimize d_{sw}. There is really no tradeoff here. Typically, a designer develops a good phase detector with a small sampling window, and then this can be reused for the vast majority of DLL instances on the chip. Next, an appropriate d_r should be selected based on application requirements and desired DCDL complexity, area and power. D_{min} and D_{max} will be set by the entire range of input clock frequencies that need to be tracked and also by the entire process, voltage, and temperature operating space. Under no circumstances should a DLL prove to be the limiting factor in a chip design in terms of (F_{MAX}, V_{MAX}) or (F_{MIN}, V_{MIN}). Substantial margin should always be provided to allow for small changes in production specifications. The controller should be designed to perform successful lock acquisition for each PVT and for each input clock frequency. N_d should be minimized to the extent possible. In the lock state, the controller should maintain lock with the minimum theoretical phase error of $\pm d_r$ (wherever possible). N_{bw} should be greater than N_d but small enough to allow for input frequency variation tracking if the application demands it.

Other design aspects such as DCDL output glitching, controller robustness around non-locking edge traversal, etc., should be thoroughly investigated and analyzed because they can prove catastrophic.

6.8 Noise Considerations

Phase noise and jitter transfer have been traditionally analyzed in the frequency domain (Chap. 5.1). In this section, we focus on the time domain because of the difficulties introduced by the non-linear nature of the system. For a comprehensive analog DLL noise analysis in the frequency domain, the reader is referred to [17, 18]. Unlike a PLL which can filter reference clock jitter as described in Chap. 5.1, a digital DLL will add jitter to the reference clock through three primary mechanisms:

The first mechanism is the structural limit cycle which in a well designed DLL will add $\pm d_r$ of phase jitter to the output clock. From a frequency domain perspective, this noise will appear around the f_{CLKIN}/N_{bw} frequency band. The second mechanism is the equivalent of jitter peaking in the analog domain: The digital DLL can

amplify PLL jitter because of delayed response. This stems from the fact that the DLL is not capable of determining whether a disturbance in the reference vs. output clock must be tracked (i.e., spread spectrum or DCDL environmental change) or not tracked and potentially canceled out (reference clock jitter). Since all DLLs are designed to correct phase error, a jitter disturbance in the reference clock can be tracked before it shows up at the output of the delay line in a cycle later, and this can cause jitter amplification in the next cycle. In general though, this jitter amplification will be limited and on the order of $\pm d_{\mathrm{r}}$. Furthermore, it does not accumulate as in the PLL case (Chap. 5.1) and it will add a small cycle time penalty. It is very simple to alleviate both the structural and the jitter amplification components either by using a ternary phase detector with a NOP state [24] or by opening the loop and tracking the phase error with an auxiliary mechanism [25]. The final and most potent mechanism is supply noise induced jitter. Supply noise is particularly dangerous for DLLs because we have clock propagation through a long delay line which typically has significant delay sensitivity to power supply changes. The remainder of this section frames the problem and presents a simple analytical model.

For the analysis in this section, we will adopt the terminology of Sect. 5.2. We are trying to compute the maximum period jitter generated by a fixed delay D subject to known sinusoidal supply noise, as a function of noise amplitude, noise frequency, delay amount D, and clock period T. Before proceeding, we define the following symbols:

$V(t)$	Power supply voltage as a function of time
V_{DD}	Nominal power supply voltage value
A_{n}	Supply voltage noise amplitude (normalized to V_{DD})
f_n	Supply voltage noise frequency
ϕ	Supply voltage noise phase at $t = 0$
D	DLL delay (also referred to as insertion delay)
T	Clock period
t_i	Time of ith clock positive edge at the input of insertion delay D with respect to an arbitrary reference

We define the kth order period jitter introduced by a DLL with a constant insertion delay D as:

$$\Phi'_k = \max_i \left| t_{i+k} - t_i - kT \right| . \tag{6.43}$$

Equation (6.43) is interpreted as the maximum amount that the sum of k consecutive cycles of a given clock of period T can differ from the nominal value. We have introduced an additional subscript k (order) with respect to the jitter definitions of Sect. 5.2 to allow one more parameter in this analysis and have the capability to address multi-cycle paths and long term jitter requirements imposed by certain applications. The subscript k is only relevant for the relative jitter definitions of Sect. 5.2 such as period jitter ($\Phi'[n]$) and cycle-to-cycle jitter ($\Phi''[n]$). The relationship between Eq. (6.43) and the period jitter definition of Sect. 5.2 is

$$\Phi'_1 = \max_n \left| \Phi'[n] \right| . \tag{6.44}$$

In order to derive an analytical expression for Φ'_k which will help us gain some insight regarding supply noise induced jitter, we start by assuming sinusoidal supply noise:

$$V(t) = V_{\mathrm{DD}}[1 + A_n \sin(2\pi f_n t + \phi)]. \tag{6.45}$$

Period jitter is generated by modulating the insertion delay D through supply noise. We make a second assumption, that the actual value $D_m(t)$ of the insertion delay D at time t (where t is the time that an ideal positive edge arrives at the input of the delay D) is given by the following formula:

$$D_m(t) = D \times \left(2 - \frac{\overline{V}(t,D)}{V_{\mathrm{DD}}}\right), \tag{6.46}$$

where $\overline{V}(t,D)$ is simply the forward moving average of $V(t)$ over a period of time equal to insertion delay D:

$$\overline{V}(t,D) = \frac{1}{D} \times \int_t^{t+D} V(\tau)d\tau. \tag{6.47}$$

Equation (6.46) simply states that the DLL insertion delay is linearly dependent on its supply voltage averaged over its flight time (i.e., an $x\%$ increase in average supply voltage results in an $x\%$ decrease in delay). This is an acceptable approximation for values of $\overline{V}(t,D)$ reasonably close to nominal V_{DD}. Obviously, this is not true for much larger or smaller values of $\overline{V}(t,D)$ and the axes crossing points implied by Eq. (6.46) are meaningless. There is an additional approximation in Eq. (6.47) since the moving average is computed over a constant time interval D whereas in reality the time interval should vary due to supply noise modulation. This approximation should introduce little error yet make an analytical approach tractable.

Now that we have explicitly defined $D_m(t)$, we can write an expression for t_i (time of ith positive edge at the output of delay D):

$$t_i = iT + D_m(iT). \tag{6.48}$$

Substituting (6.48) in (6.43) yields

$$\Phi'_k = \max_i |D_m((i+k)T) - D_m(iT)|. \tag{6.49}$$

After substituting (6.46) in (6.49) and with minimal manipulation we have

$$\Phi'_k = \frac{D}{V_{\mathrm{DD}}} \times \max_i \left|\overline{V}(iT,D) - \overline{V}((i+k)T,D)\right|. \tag{6.50}$$

Since we have an expression for supply voltage, we can obtain a closed form expression for the moving average by substituting Eq. (6.45) in (6.47).

$$\overline{V}(t,D) = V_{\mathrm{DD}} + \frac{A_n V_{\mathrm{DD}}}{D} \int_t^{t+D} \sin(2\pi f_n \tau + \phi)d\tau. \tag{6.51}$$

Performing the simple integration yields

$$\overline{V}(t,D) = V_{\mathrm{DD}} + \frac{A_n V_{\mathrm{DD}}}{2\pi f_n D}[\cos(2\pi f_n t + \phi) - \cos(2\pi f_n (t+D) + \phi)]. \tag{6.52}$$

Substituting (6.52) in (6.50) yields

$$\Phi'_k = \frac{A_n}{2\pi f_n} \max_i \left| \begin{array}{l} \cos(2\pi f_n iT + \phi) - \cos(2\pi f_n (iT+D) + \phi) - \\ \cos(2\pi f_n (i+k)T + \phi) + \cos(2\pi f_n ((i+k)T + D) + \phi) \end{array} \right|. \tag{6.53}$$

We now recall the following well-known trigonometric identity:

$$\cos u + \cos v = 2\cos\left(\frac{u+v}{2}\right)\cos\left(\frac{u-v}{2}\right). \tag{6.54}$$

Applying (6.54) on (6.53) yields

$$\Phi'_k = \frac{A_n}{\pi f_n} \max_i |\cos(\pi f_n (2i+k)T + \phi)[\cos(\pi f_n (kT+D)) - \cos(\pi f_n (kT-D))]|. \tag{6.55}$$

Equation (6.55) is expressed in a very convenient form which will enable us to dispense with the maximization operation by inspection: Only the first cosine term $(\cos(\pi f_n (2i+k)T + \phi))$ is a function of i, and its maximum value for every possible i is 1. Therefore, Eq. (6.55) becomes

$$\Phi'_k = \frac{A_n}{\pi f_n} |\cos(\pi f_n (kT+D)) - \cos(\pi f_n (kT-D))|. \tag{6.56}$$

Application of one more trigonometric identity results in the following expression which is more desirable since it separates the effects of variables T and D:

$$\boxed{\Phi'_k = \frac{2A_n}{\pi f_n} |\sin(\pi f_n kT)\sin(\pi f_n D)|.} \tag{6.57}$$

Equation (6.57) constitutes a closed form analytical expression of worst case period jitter as a function of supply noise frequency and amplitude (f_n, A_n), clock period (T), and DLL insertion delay (D). Parameter k denotes whether we are interested in single period jitter $(k = 1)$ or longer term (multi-cycle) jitter $(k > 1)$. In order to gain some insight into this expression and determine whether it makes physical sense, let us focus on the two plots of Fig. 6.38. Both diagrams plot Eq. (6.57) under certain conditions. The top graph shows Φ'_1 as a function of noise frequency f_n for 2 different insertion delays. We make the following observations: First, we note that for all insertion delay values we have zero jitter at $f_n = 0$ (trivial) and at noise frequencies that are integral multiples of the clock frequency (500 MHz and 1 GHz). This is entirely expected because when this occurs, consecutive clock edges will be delay-modulated identically going through any insertion delay D resulting in

zero jitter. Consecutive clock edges will experience the same noise. A second observation is that we have zero jitter at noise frequencies that are integral multiples of the inverse insertion delay (333 MHz and 667 MHz in the top graph for the case where $D = 3$ ns). This is also expected because if the integration interval in the moving average Eq. (6.51) is an integral multiple of the period of the sinusoid that is being integrated, then the integration result is zero. Integrating a sinusoid over an integral multiple of periods always results in zero. This is also clearly observed on the bottom graph of Fig. 6.38 where Φ_1' is plotted as a function of insertion delay for 3 different noise frequencies. All zeroes on the graph are at insertion delays equal to an integral multiple of the noise period. A final observation is that with increasing noise frequency, period jitter tends to decrease due to the $1/f_n$ term in Eq. (6.57). This happens because as f_n increases, the amount of noise averaging in delay line D also increases and the supply moving average will approach the nominal value.

Figure 6.39 shows the jitter dependence on noise frequency and insertion delay simultaneously in 2 dimensions for 1 GHz and 500 MHz clocks respectively. Such contour plots for each relevant clock frequency are very easy to construct using Eq. (6.57) and can provide a broad perspective early in the design phase in order to help determine important implementation aspects such as:

1. Identification of worst case period jitter and particularly undesired noise frequencies.
2. Derivation of detailed specifications of supply voltage frequency and amplitude if external to the IC.
3. Identification of need for internal voltage regulation to minimize A_n if external supply specifications turn out to be too stringent.

Application of Eq. (6.57) is not restricted to delay locked loops but can be applied to any open-loop clock buffer in order to determine supply induced period jitter. DLLs are very convenient because insertion delay D is well defined and tracked with feedback. In an open-loop buffer, D will vary with PVT and can only be estimated to a limited degree of accuracy. Yet, the analysis above reveals an interesting aspect of supply induced jitter: In an ideal world, it can be canceled out by selecting an appropriate clock buffer delay D as Figs. 6.38 and 6.39 indicate. This is not really possible because the supply noise frequency is never known a priori and typically does not consist of a single sinusoid. A supply voltage waveform will typically contain multiple natural frequencies introduced by the package, board, and external decoupling capacitance network. In addition, it will contain particular solutions related to the system clock frequency. Natural frequencies are unknown at IC design time, since they are system dependent. Particular solutions are also hard to estimate especially in programmable ICs. One could imagine the possibility of constructing a feedback system which measures supply noise frequency and then adjusts clock buffer delay accordingly to minimize jitter. Feedback must be close to instantaneous in order to avoid the worst case, and this makes a digital controller very hard to implement. Assuming that supply noise frequency can actually be measured, such a system could potentially track an undesired natural frequency that should not have been there in the first place. Controlling noise amplitude, undesired natural frequencies and clock

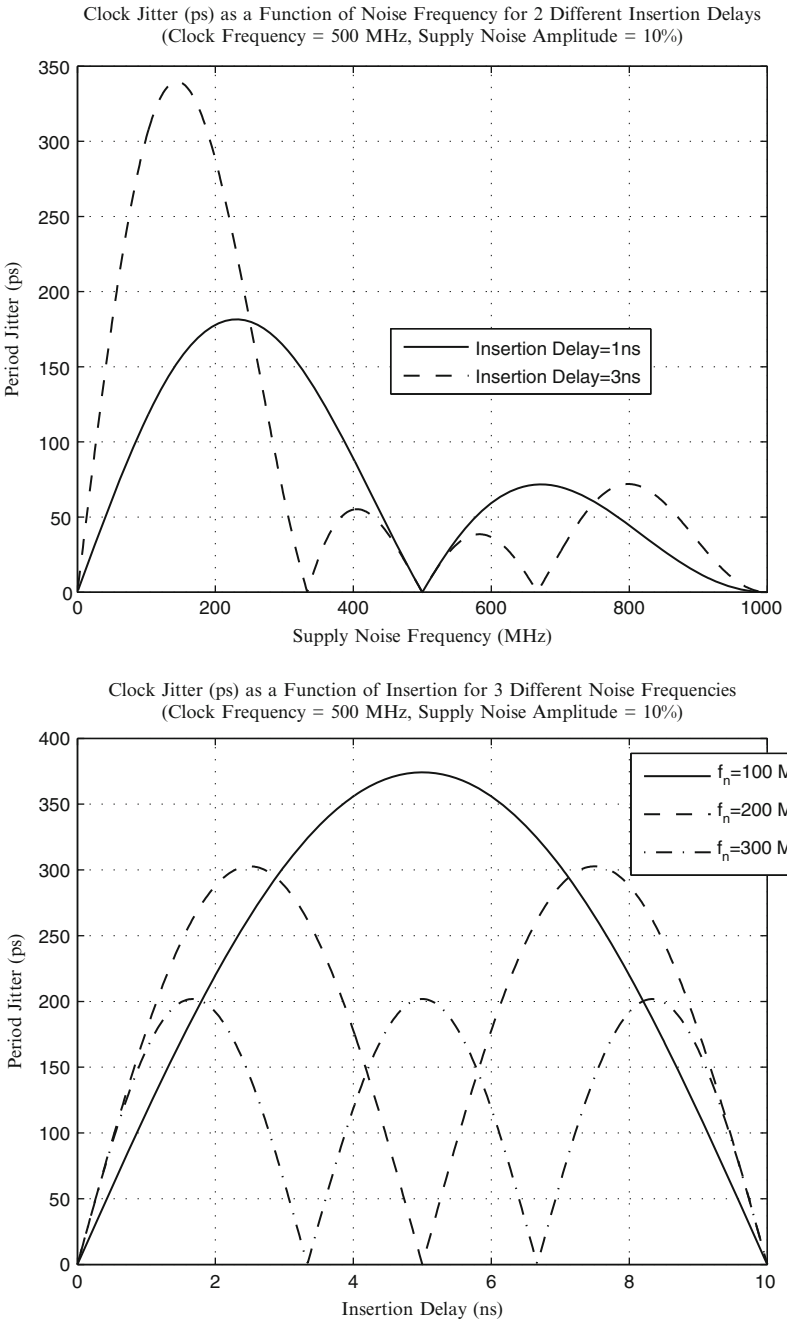


Fig. 6.38. First order period jitter (Φ_1') as a function of supply noise frequency and insertion delay

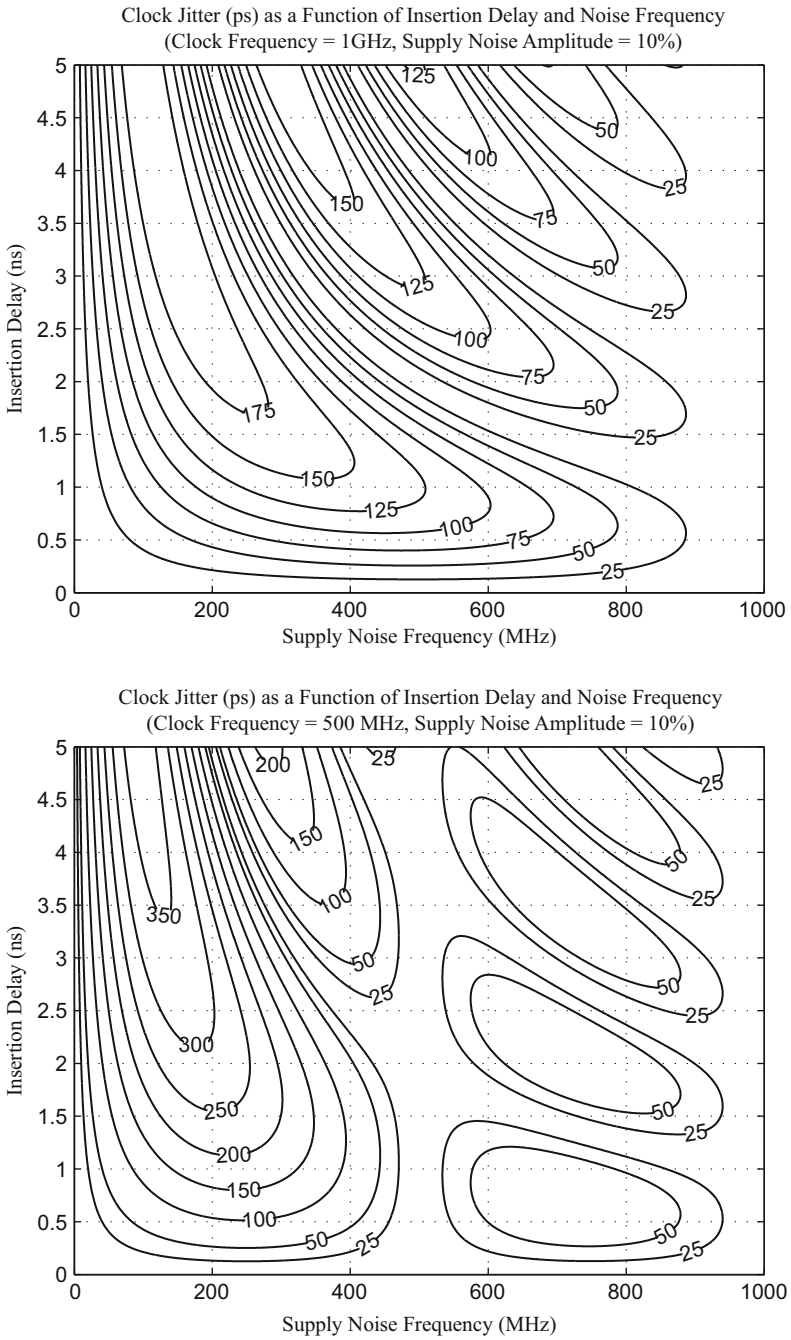


Fig. 6.39. First order period jitter (Φ_1') as a function of insertion delay and supply noise frequency in 2 dimensions

buffer delays are the only effective methods for reducing supply induced jitter that the author is aware of.

6.9 Advanced Applications

In this section, we review a number of non-standard DLL applications and demonstrate the versatility of delay lock techniques.

6.9.1 Duty Cycle Correction

DLLs are heavily used in I/O applications such as DRAM interfaces. In such applications, 50% duty cycle is very important because the interfaces are typically double-data-rate (DDR) and both clock phases are used for signaling. Duty cycle distortion is a source of deterministic jitter and must be reduced.

It is fairly common for DLLs to include a separate stage of duty cycle correction as an output stage [26] or even distributed duty cycle control at various points along the signal path [27]. These are typically analog methods that involve current-steering based structures. An alternative method of duty cycle correction implemented in DRAMs [28, 29] is shown in Fig. 6.40. It uses two independent DLLs. The first one (PD1, DCDL1, CTL1) is the main loop and is responsible for locking the output to the desired CLKIN phase. The second one (PD2, DCDL2, CTL2) has an inverting delay line and locks the positive edge of CLK2 (which has been generated by a negative edge of CLKIN) to the positive edge of CLK1 (which has been generated by a positive edge of CLKIN).

When the second loop locks, CLK1 and CLK2 will have the phase relationship shown in Fig. 6.40. A phase interpolator with equal weights (similar to the one in Fig. 6.26) is used as the final clock output stage. The positive edge of output clock CLKOUT will be generated by the positive edge of CLK1 since both CLK1 and CLK2 have phase locked positive edges. The negative edge of CLKOUT will be generated by the average of the CLK1 and CLK2 negative edges, resulting in duty cycle correction. Figure 6.40 assumes for simplicity that the phase interpolator is a zero-delay, thus non-causal circuit. In reality, there will be delay involved but it has no effect in the overall correction capability of this method. The amount of duty cycle correction possible depends on the *RC* of the interpolator as described in Sect. 6.5.3.

6.9.2 Clock Multiplication

In Sect. 6.3, the basic idea of DLL-based clock multiplication has been presented (Fig. 6.4d). This method of phase-locking a multi-tap DCDL to a reference clock period and then using the intermediate phases to generate a frequency multiple through a clock-multiplying structure such as a multi-port toggle flop has an obvious frequency limitation: it is limited by the minimum delay $D_{\min}$ through each DCDL segment. As an example, the highest output frequency that can be generated by the structure of Fig. 6.4d is subject to the following constraint:

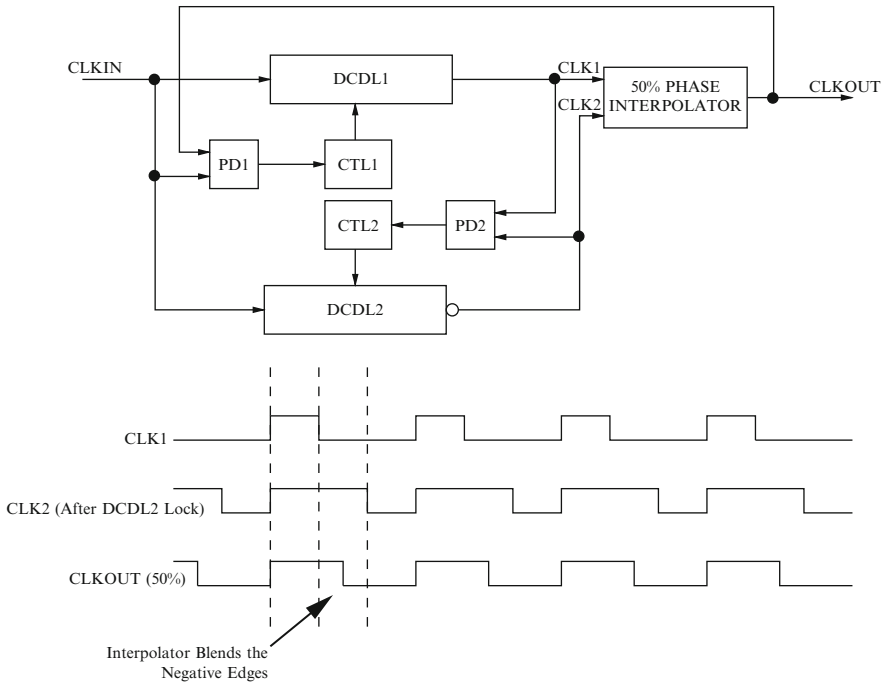


Fig. 6.40. Duty cycle correction in a DLL environment

$$f_{\text{out}} \leq \frac{1}{2D_{\text{min}}}, \tag{6.58}$$

where D_{min} is the minimum possible delay through each of the 8 DCDL segments.

Reference [30] proposes an interesting way to remove this upper bound. The key observation is that the sum of the N DCDLs (where $N/2$ is the intended clock multipler) can be phase-locked to multiple reference clock periods (M) instead of 1 and the resulting phases can be re-sorted before driving the final clock multipler stage. For this method to work, M must not divide N exactly but have a non-zero modulus. Figure 6.41 illustrates this concept. We wish to perform clock multiplication by 4 ($N = 8$), but we want to generate an output clock frequency that is faster than the constraint in (6.58) would allow for. Instead of phase-locking all 8 DCDL segments to one reference clock period T_{ref}, we choose to phase-lock them to $3T_{\text{ref}}$ ($M = 3$) which evidently allows for a much larger D_{min} per DCDL segment. Phases are sorted modulus T_{ref} before being routed to the final clock multiplication stage. The constraint in (6.58) now becomes

$$f_{\text{out}} \leq \frac{3}{2D_{\text{min}}}, \tag{6.59}$$

and the upper bound for the final output frequency has become more loose by a factor of 3. In [30], a 40 GHz output clock is generated by a 4.4445 GHz reference ($M = 2$,

$N = 9$) by employing an LC-based clock multiplier which can produce a full 360° oscillation per stimulating phase.

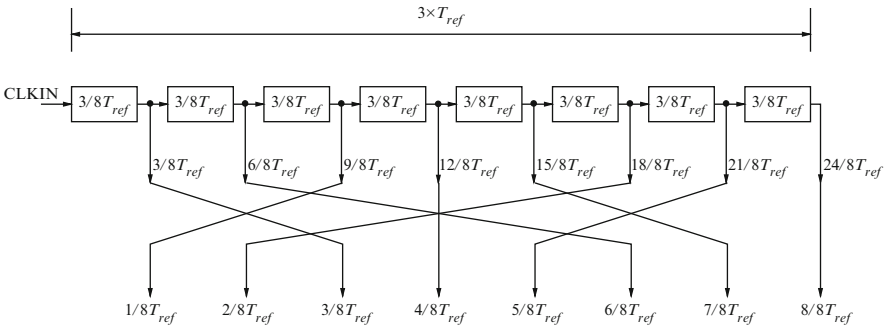


Fig. 6.41. Multiperiod DCDL locking for high output frequency clock multiplication

Clock multiplying DLLs with alternative organizations have been proposed in the past years [31, 32] and are actively pursued as a credible alternative to PLLs for clock generation due to superior jitter performance.

6.9.3 Infinite Dynamic Range

Lock acquisition and phase capture can be limited in standard DLLs with a reduced DCDL dynamic range as explained in Sects. 6.5 and 6.6. One way to address this issue is to extend the delay line dynamic range through additional DCDL stages or conditional inversion (Sect. 6.6.2) coupled with increased complexity in the FSM controller. An alternative method is to employ a dual DLL architecture where the core loop generates phases that span the input clock period range, and the peripheral loop blends a selected pair of consecutive phases to match the reference clock [1, 33]. This architecture is shown in Fig. 6.42. The core DLL phase-locks at 180° and generates 6 phases that are 30° apart. The block labeled "Phase Selection" contains two 3×1 multiplexers that select a pair of phases ϕ and ψ to be interpolated. Before reaching the interpolator, these phases are conditionally inverted inside the "Selective Phase Inversion" block so that the cascade of these three structures can generate all phases at 30° increments that span the entire 0–360° phase interval. Finally, conditionally inverted phases ϕ' and ψ' are interpolated. Although the phase interpolator structure in [33] is CML-based, the analysis is very similar to the full swing interpolator of Sect. 6.5.3. The output of the interpolator Θ is phase-locked to the reference clock ("ref CLK" in Fig. 6.42). The feedback of the peripheral DLL closes with a bang–bang phase detector and an FSM that controls the multiplexers, conditional inversion block and phase interpolator. The infinite dynamic range stems from the fact that the FSM controller can keep on selecting phase pairs to be interpolated around the entire 360° phase interval and eventually the selected pair

will encompass the phase of the reference clock. Interpolation will then guarantee phase lock within the accuracy of the interpolator. In this architecture, the input core DLL clock ("in CLK") and the reference clock ("ref CLK") can be the same clock at 0-phase or identical frequency clocks at any phase relationship. They can even be plesiochronous clocks (of similar but not equal frequency), and this case will be addressed in Sect. 6.9.4.

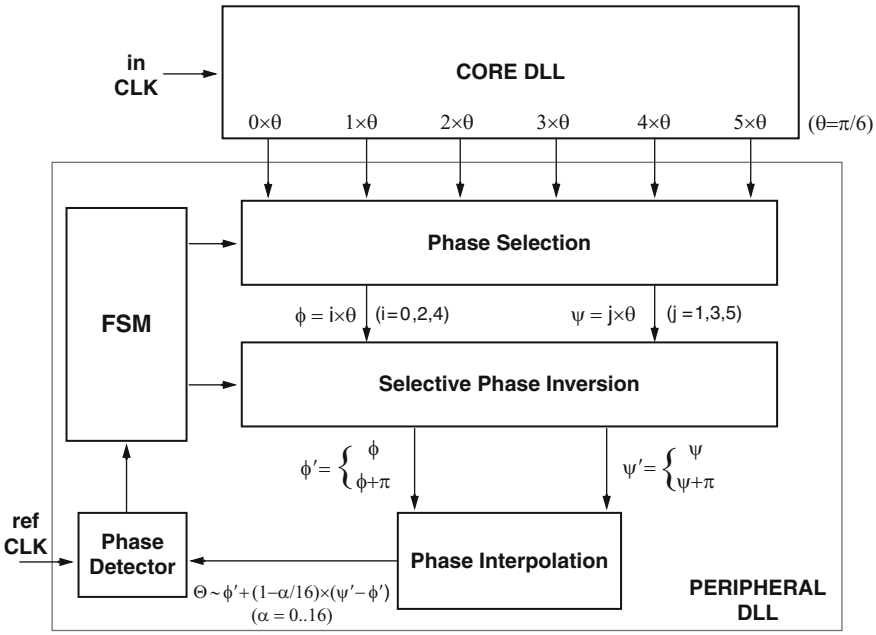


Fig. 6.42. Dual DLL architecture for virtually infinite phase capture range. Reproduced with permission from [33], ©1997 IEEE

DLLs with quadrature phase mixing had been proposed before [26] and they also exhibit infinite dynamic range. The main difference between the dual DLL architecture and quadrature mixing is that the phase interpolation is not limited to 90° clocks. Interpolation can blend phases at smaller intervals (30° in this case) which will have fewer slew rate limitations and better noise performance [33].

6.9.4 Clock-Data Recovery

The full power of the method of Fig. 6.42 cannot be fully appreciated unless we consider the case where "in CLK" and "ref CLK" are plesiochronous and vary slightly in frequency. This architecture can achieve lock in the plesiochronous case because the FSM controller can change the weight of the phase mixing in each cycle in such a way so as to generate a clock that is effectively faster or slower than "in CLK".

When the interpolator reaches the end of its dynamic range, then the phase selection multiplexer chooses a different pair of phases for mixing that are either leading or lagging the previous pair depending on whether "ref CLK" is faster or slower than "in CLK", respectively. This property makes this method ideally suited for clock-data recovery (CDR) applications where a sampling clock must be generated at the receiver and used to sample the incoming data. The generated clock must match the frequency of the received data using multiple free-running locally-generated phases.

This concept is illustrated in Fig. 6.43. This structure forms the basis of a large class of clock-data recovery architectures implemented as part of various high-speed signaling standards such as PCI Express, XAUI, and FBD (Fully-Buffered-DIMM) [34]. The core DLL of Fig. 6.42 has been replaced with a multi-phase PLL that generates a local high frequency clock which can be plesiochronous to the incoming data. In order to collect enough information for clock recovery, a CDR needs some form of data oversampling. In this case, oversampling is accomplished by generating both an in-phase clock (CLKI) and a quadrature clock (CLKQ). High-speed incoming data are sampled with both clocks using two separate slicers (I-SLICER, Q-SLICER). The data are then deserialized typically by a factor of 10 and retimed in a divided clock domain which is usually a divided-down version of CLKQ. The block labeled "EDGE DETECTION/PHASE DETECTION" examines both I and Q deserialized data and determines whether the current sampling clocks CLKI and CLKQ are fast or slow with respect to the incoming data. This decision can be made deterministically due to oversampling. This decision is then forwarded to the CDR FSM which implements a control algorithm and can change the settings of the phase selectors and interpolators to advance or delay the sampling clocks CLKI and CLKQ. The FSM algorithm will determine important capabilities such as phase capture profile, frequency difference between local PLL and incoming data that can be safely absorbed and limit cycle behavior which will cause deterministic jitter in the system. The output of the Q-DESERIALIZER represents parallel data sampled safely with maximum margin in the middle of the eye diagram and is forwarded to the rest of the system for higher layer processing.

All blocks of Fig. 6.43 with the sole exception of the multi-phase PLL constitute essentially a DLL. The role of the delay line is fulfilled by the combination of phase selectors and interpolators that can effectively delay or advance an input clock and the phase detection is carried out by the slicers, deserializers, and decision block. In a typical SERDES application [34], this DLL structure is present on all receiver bits (*lanes* in SERDES terminology) whereas the multi-phase PLL is amortized over a number of receive and transmit lanes. In this application, it is desirable to implement a second order control structure in order to track more effectively the frequency differences between the incoming data and the locally generated clock. Second order control design has not been addressed in this chapter and is beyond our scope.

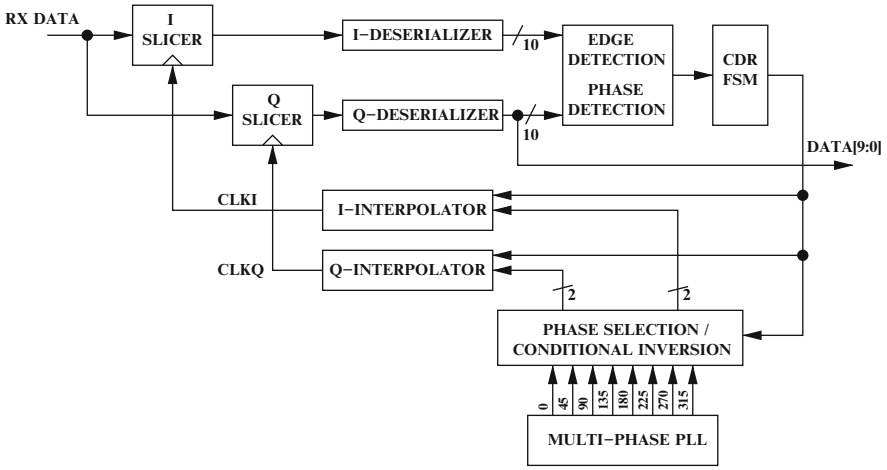


Fig. 6.43. DLL-based clock-data recovery loop

6.9.5 On-Chip Temperature Sensing

An appropriately calibrated and accurately measured delay line can be used as a temperature sensor since temperature affects digital gate delay. It has been shown [35] that when a delay line is trimmed to a fixed delay at a certain temperature in a production environment across multiple chips, it can act as an accurate temperature sensor. Trimming to a fixed delay at production has the effect of suppressing sensitivity to process and voltage. A trimmed delay line will vary only due to temperature across all fused chips and can have an accuracy of $\pm 2^{\circ}$C across the 0–100°C temperature range [35].

Two DLLs constitute this system: A reference DLL is phase locked to a known reference clock which is distributed to the sensor. Tapping at intermediate delay line elements can generate fixed reference delays that are fractions of the original reference clock period. A secondary DLL phase locks the sensing delay line to a fixed delay generated by tapping an intermediate delay output of the reference DLL. This step must be performed at a controlled junction temperature on all chips during manufacturing. The setting in the locked state is then fused and this step performs the trimming (normalization) of the sensing delay line.

During measurement, the temperature can vary and the trimmed sensing line will deviate from its fixed delay value. The secondary DLL then treats it as an unknown delay and measures it using intermediate delay outputs of the reference DLL as the measuring delay line. The digital setting output at the locked state will be an accurate representation of temperature. This dual DLL scheme can provide good accuracy and a high measurement bandwidth (5 ksamples/s) and is appropriate for microprocessor thermal monitoring.

6.10 Conclusion

This chapter has presented an overview of the basic digital DLL structure and its components in addition to a survey of a range of applications. DLLs are an essential component of modern VLSI systems. Increased clocking system complexity, process variation, and wider supply voltage ranges will make their use even more prevalent than it is today. Furthermore, their time-to-digital conversion capabilities make them an attractive candidate for measuring analog on-chip quantities (i.e., voltage drop, temperature, process variation) either for valuable data collection or system normalization to dynamically remove variation.

Acknowledgments

A preliminary version of the power supply noise analytical model in Sect. 6.8 was developed jointly with John Eble when we were colleagues at the Alpha Development Group, Compaq Computer.

References

[1] W. Dally and J. Poulton, *Digital Systems Engineering*. Cambridge University Press, New York, NY, 1998.
[2] M. Johnson and E. Hudson, A variable delay line PLL for CPU-coprocessor synchronization. *IEEE J. Solid-State Circuits*, 23(5), 1218–1223, 1988.
[3] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison Wesley, Reading, MA, 1980.
[4] L. Lamport, Buridan's principle, October 1984. [Online]. Available: http://research. microsoft.com/users/lamport/pubs/buridan.pdf
[5] T. Chaney and C. Molnar, Anomalous behavior of synchronizer and arbiter circuits. *IEEE Trans. Comput.*, C-22(4), 421–422, 1973.
[6] H. Veendrick, The behaviour of flip-flops used as synchronizers and prediction of their failure rate. *IEEE J. Solid-State Circuits*, 15(2), 169–176, 1980.
[7] A. Drake, *Fundamentals of Applied Probability Theory*. McGraw-Hill, New York, 1967.
[8] Nanoscale Integration and Modeling Group, Arizona State University, Predictive technology model. [Online]. Available: http://www.eas.asu.edu/ptm.
[9] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation. In: *Proc. CICC Custom Integrated Circuits Conference*, 2000, pp. 201–204.
[10] W. Zhao and Y. Cao, New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Trans. Electron Devices*, 53(11), 2816–2823, 2006.
[11] 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*, 32(11), 1728–1734, 1997.

[12] F. Lin, J. Miller, A. Schoenfeld, M. Ma, and R. Baker, A register-controlled symmetrical DLL for double-data-rate DRAM. *IEEE J. Solid-State Circuits*, 34(4), 565–568, 1999.

[13] T. Xanthopoulos, D. Bailey, A. Gangwar, M. Gowan, A. Jain, and B. Prewitt, The design and analysis of the clock distribution network for a 1.2 GHz Alpha microprocessor. In: *Digest of Technical Papers IEEE International Solid-State Circuits Conference (ISSCC 2001)*, 2001, pp. 402–403.

[14] S. Sidiropoulos, High performance inter-chip signalling, Ph.D. dissertation, Stanford University, 1998.

[15] J. Maneatis, Design of high-speed CMOS PLLs and DLLs. In: A. Chandrakasan, W. Bowhill, and F. Fox (eds.) *Design of High Performance Microprocessor Circuits*, IEEE Press, New York, 2001, pp. 235–260.

[16] C.-K. K. Yang, Delay-locked loops – an overview. In: B. Razavi, (ed.) *Phase-Locking in High-Peformance Systems*, Wiley-IEEE Press, New York, NY, 2003, pp. 13–22.

[17] M.-J. Lee, W. Dally, T. Greer, H.-T. Ng, R. Farjad-Rad, J. Poulton, and R. Senthinathan, Jitter transfer characteristics of delay-locked loops – theories and design techniques. *IEEE J. Solid-State Circuits*, 38(4), 614–621, 2003.

[18] J. Burnham, G.-Y. Wei, C.-K. K. Yang, and H. Hindi, A comprehensive phase-transfer model for delay-locked loops. In: *Proc. IEEE Custom Integrated Circuits Conference CICC '07*, 2007, pp. 627–630.

[19] J. Lee, K. Kundert, and B. Razavi, Analysis and modeling of bang-bang clock and data recovery circuits. *IEEE J. Solid-State Circuits*, 39(9), 1571–1580, 2004.

[20] J. Sonntag and J. Stonick, A digital clock and data recovery architecture for multi-gigabit/s binary links. *IEEE J. Solid-State Circuits*, 41(8), 1867–1875, 2006.

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

[22] T. Saeki, Y. Nakaoka, M. Fujita, A. Tanaka, K. Nagata, K. Sakakibara, T. Matano, Y. Hoshino, K. Miyano, S. Isa, S. Nakazawa, E. Kakehashi, J. Drynan, M. Komuro, T. Fukase, H. Iwasaki, M. Takenaka, J. Sekine, M. Igeta, N. Nakanishi, T. Itani, I. Yoshida, K. Yoshino, S. Hashimoto, T. Yoshii, M. Ichinose, T. Imura, M. Uziie, S. Kikuchi, K. Koyama, Y. Fukuzo, and T. Okuda, A 2.5-ns clock access, 250-MHz, 256-Mb SDRAM with synchronous mirror delay. *IEEE J. Solid-State Circuits*, 31(11), 1656–1668, 1996.

[23] K. Sung and L.-S. Kim, A high-resolution synchronous mirror delay using successive approximation register. *IEEE J. Solid-State Circuits*, 39(11), 1997–2004, 2004.

[24] A. Efendovich, Y. Afek, C. Sella, and Z. Bikowsky, Multifrequency zero-jitter delay-locked loop. *IEEE J. Solid-State Circuits*, 29(1), 67–70, 1994.

[25] B. Mesgarzadeh, Low-power low-jitter clock generation and distribution, Ph.D. dissertation, Linkoping University, 2008.
[26] T. Lee, K. Donnelly, J. Ho, J. Zerbe, M. Johnson, and T. Ishikawa, A 2.5 V CMOS delay-locked loop for 18 Mbit, 500 megabyte/s DRAM. *IEEE J. Solid-State Circuits*, 29(12), 1491–1496, 1994.
[27] B. Garlepp, K. Donnelly, J. Kim, P. Chau, J. Zerbe, C. Huang, C. Tran, C. Portmann, D. Stark, Y.-F. Chan, T. Lee, and M. Horowitz, A portable digital DLL for high-speed CMOS interface circuits. *IEEE J. Solid-State Circuits*, 34(5), 632–644, 1999.
[28] J.-B. Lee, K.-H. Kim, C. Yoo, S. Lee, O.-G. Na, C.-Y. Lee, H.-Y. Song, J.-S. Lee, Z.-H. Lee, K.-W. Yeom, H.-J. Chung, I.-W. Seo, M.-S. Chae, Y.-H. Choi, and S.-I. Cho, Digitally-controlled DLL and I/O circuits for 500 Mb/s/pin x16 DDR SDRAM. In: *Digest of Technical Papers IEEE International Solid-State Circuits Conference (ISSCC 2001)*, 2001, pp. 68–69, 431.
[29] W.-J. Yun, H. W. Lee, D. Shin, S. D. Kang, J. Y. Yang, H. O. Lee, D. U. Lee, S. Sim, Y. J. Kim, W. J. Choi, K. S. Song, S. H. Shin, H. H. Choi, H. W. Moon, S. W. Kwack, J. W. Lee, Y. K. Choi, N. K. Park, K. W. Kim, Y. J. Choi, J.-H. Ahn, and Y. S. Yang, A 0.1-to-1.5GHz 4.2mW all-digital DLL with dual duty-cycle correction circuit and update gear circuit for DRAM in 66nm CMOS technology. In: *Digest of Technical Papers IEEE International Solid-State Circuits Conference (ISSCC 2008)*, 2008, pp. 282–283, 613.
[30] C.-N. Chuang and S. luan Liu, A 40GHz DLL-based clock generator in 90nm CMOS technology. In: *Digest of Technical Papers IEEE International Solid-State Circuits Conference (ISSCC 2007)*, 2007, pp. 178–179, 595.
[31] R. Farjad-Rad, W. Dally, H.-T. Ng, R. Senthinathan, M.-J. Lee, R. Rathi, and J. Poulton, A low-power multiplying DLL for low-jitter multigigahertz clock generation in highly integrated digital chips. *IEEE J. Solid-State Circuits*, 37(12), 1804–1812, 2002.
[32] B. Helal, M. Straayer, G.-Y. Wei, and M. Perrott, A highly digital MDLL-based clock multiplier that leverages a self-scrambling time-to-digital converter to achieve subpicosecond jitter performance. *IEEE J. Solid-State Circuits*, 43(4), 855–863, 2008.
[33] S. Sidiropoulos and M. Horowitz, A semidigital dual delay-locked loop. *IEEE J. Solid-State Circuits*, 32(11), 1683–1692, 1997.
[34] D. Stauffer et al., *High Speed Serdes Devices and Applications*. Springer Science and Business Media, New York, 2008.
[35] K. Woo, S. Meninger, T. Xanthopoulos, E. Crain, and D. Ham, Dual-DLL-based CMOS all-digital temperature sensor for microprocessor thermal monitoring. In: *Digest of Technical Papers IEEE International Solid-State Circuits Conference (ISSCC 2009)*, 2009, pp. 68–69.