

# SKEW-TOLERANT CIRCUIT DESIGN

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

David Harris

February 1999

(c) Copyright 1999 by David Harris

All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

---

Mark Horowitz, Principal Advisor

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

---

Bruce Wooley

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

---

Abbas El-Gamal

Approved for the University Committee on Graduate Studies:

---

## **Abstract**

As cycle times in high-performance digital systems shrink faster than simple process improvement allows, sequencing overhead consumes an increasing fraction of the clock period. In particular, the overhead of traditional domino pipelines can consume 25% or more of the cycle time in aggressive systems. Fortunately, the designer can hide much of this overhead through better design techniques. The key to skew-tolerant design is avoiding hard edges in which data must setup before a clock edge but will not continue propagating until after the clock edge. Skew-tolerant domino circuits use multiple overlapping clocks to eliminate latches, removing hard edges and hiding the sequencing overhead.

This thesis presents a systematic approach to skew-tolerant circuit design, combining both static and domino circuits. It describes the tradeoffs in clocking schemes and concludes that four phases is a reasonable design choice which can tolerate nearly a quarter cycle of skew and is simple to implement. Four-phase skew-tolerant domino integrates easily with static logic using transparent or pulsed latches and shares a consistent scan methodology providing testability of all cycles of logic. Timing types are defined to help understand and verify legal connectivity between static and domino logic. While clock skew across a large die are an increasing problem, we can exploit locality to budget only the smaller skews seen between pairs of communicating elements. To verify such systems with different skews between different elements, we present an improved timing analysis algorithm. Skew-tolerant circuits will facilitate the design of complex synchronous systems well into the GHz regime.

## Acknowledgments

This thesis is dedicated to my parents, Dan and Sally Harris, who have inspired me to teach and write.

Many people contributed to the work in this thesis. My advisor, Mark Horowitz, gave me a framework to think about circuit design and has been a superb critic. He taught me to never accept that a research solution is “good enough” but rather to look for the best possible answer. Ivan Sutherland has been an terrific mentor, giving me the key insight to I need to break out of writer’s block: “A thesis is a document to which three professors will sign their names saying it is. It is well to remember that it is nothing more lest you take too long to finish.” Bruce Wooley, on my reading committee, gave the thesis a careful reading and had many useful suggestions. My officemate Ron Ho has been a great help, sounding out technical ideas and saving me many times when I had computer crises. I have enjoyed collaborating with other members of the research group and especially thank Jaeha Kim, Dean Liu, Jeff Solomon, Gu Wei, and Evelina Yeung. Zeina Daoud supplied both technical assistance and good conversation. I always learn more when trying to teach, so I would like to thank my students at Stanford, Berkeley, and in various industrial courses. Finally, I would like to thank my high school teachers who taught me to write through extensive practice, especially Mrs. Stephens, Mr. Roseth, and Mr. Phillips.

This work has been supported through funding from NSF, Stanford, and DARPA. I have also been supported, both financially and intellectually, through work with Sun Microsystems, Intel Corporation, and HAL Computer.

# Table of Contents

Chapter 1	Skew-Tolerant Circuit Design.....	1
1.1	Overhead in Flip-Flop Systems.....	1
1.2	Throughput and Latency Trends.....	4
1.2.1	Impact of Overhead on Throughput and Latency.....	5
1.2.2	Historical Trends.....	6
1.2.3	Future Predictions.....	9
1.2.4	Conclusions.....	10
1.3	Skew-Tolerant Static Circuits.....	10
1.4	Domino Circuits.....	12
1.4.1	Domino Gate Operation.....	13
1.4.2	Traditional Domino Clocking.....	18
1.4.3	Skew-Tolerant Domino.....	19
1.5	A Look Ahead.....	22
Chapter 2	Skew-Tolerant Domino Circuits.....	24
2.1	Skew-Tolerant Domino Timing.....	24
2.1.1	General Timing Constraints.....	25
2.1.2	Clock Domains.....	28
2.1.3	50% Duty Cycle.....	30
2.1.4	Single Gate per Phase.....	31
2.1.5	Min-Delay Constraints.....	32
2.1.6	Recommendations and Design Issues.....	34
2.2	Simulation Results.....	36
2.3	Summary.....	38
Chapter 3	Circuit Methodology.....	39
3.1	Static / Domino Interface.....	40
3.1.1	Static to Domino Interface.....	40
3.1.2	Domino to Static Interface.....	41
3.1.3	Timing Types.....	43
3.1.4	Qualified Clocks.....	57
3.1.5	Min-Delay Checks.....	58
3.2	Clocked Element Design.....	61
3.2.1	Latch Design.....	61
3.2.2	Domino Gate Design.....	62
3.2.3	Special Structures.....	63
3.3	Testability.....	65
3.3.1	Static Logic.....	66
3.3.2	Domino Logic.....	67
3.4	Summary.....	70
Chapter 4	Clocking.....	71
4.1	Clock Waveforms.....	72
4.1.1	Physical Clock Definitions.....	72
4.1.2	Clock Skew.....	74
4.1.3	Clock Domains.....	76
4.2	Skew-Tolerant Domino Clock Generation.....	77
4.2.1	Delay Line Clock Generators.....	78
4.2.2	Feedback Clock Generators.....	84
4.2.3	Putting It All Together.....	86
4.3	Summary.....	87

Chapter 5	Timing Analysis .....	89
5.1	Background .....	90
5.2	Timing Analysis without Clock Skew .....	91
5.3	Timing Analysis with Clock Skew .....	94
	5.3.1 Single Skew Formulation .....	95
	5.3.2 Exact Skew Formulation.....	96
	5.3.3 Clock Domain Formulation .....	98
	5.3.4 Example .....	102
5.4	Extension to Flip-Flops and Domino Circuits .....	103
	5.4.1 Flip-Flops .....	103
	5.4.2 Domino Gates .....	104
5.5	Min-Delay .....	105
5.6	A Verification Algorithm .....	107
5.7	Results .....	110
5.8	Summary .....	112
5.9	Appendix: Timing Constraints .....	112
	5.9.1 Skewless Formulation.....	112
	5.9.2 Single Skew Formulation .....	113
	5.9.3 Exact Formulation .....	113
	5.9.4 Clock Domain Formulation .....	115
Chapter 6	Conclusions .....	116
Bibliography		119





# Chapter 1 Skew-Tolerant Circuit Design

Most digital systems today are constructed using static CMOS logic and edge-triggered flip-flops. Although such techniques have been adequate in the past and will remain adequate in the future for low-performance designs, they will become increasingly inefficient for high-performance components as the number of gates per cycle dwindles and clock skew becomes a greater problem. Designers will therefore need to adopt circuit techniques that can tolerate reasonable amounts of clock skew without an impact on the cycle time. Transparent latches offer a simple solution to the clock skew problem in static CMOS logic. Unfortunately, static CMOS logic is inadequate to meet timing objectives of the highest performance systems. Therefore, designers turn to domino circuits which offer greater speed. Unfortunately, traditional domino clocking methodologies [77] lead to circuits which have even greater sensitivity to clock skew and thus can defeat the raw speed advantage of the domino gates. This thesis formalizes and analyzes skew-tolerant domino circuits, a method of controlling domino gates with multiple overlapping clock phases. Skew-tolerant domino circuits eliminate clock skew from the critical path, hiding the overhead and offering significant performance improvement.

In this chapter, we begin by examining conventional systems built from flip-flops. We see how these systems have overhead that eats into the time available for useful computation. We then examine the trends in throughput and latency for high-performance systems and see that, although the overhead has been modest in the past, flip-flop overhead now consumes a large and increasing portion of the cycle. We turn to transparent latches and show that they can tolerate reasonable amounts of clock skew, reducing the overhead. Next, we examine domino circuits and look at traditional clocking techniques. These techniques have overhead even more severe than that paid by flip-flops. However, by using overlapping clocks and eliminating latches, we find that skew-tolerant domino circuits eliminate all of the overhead.

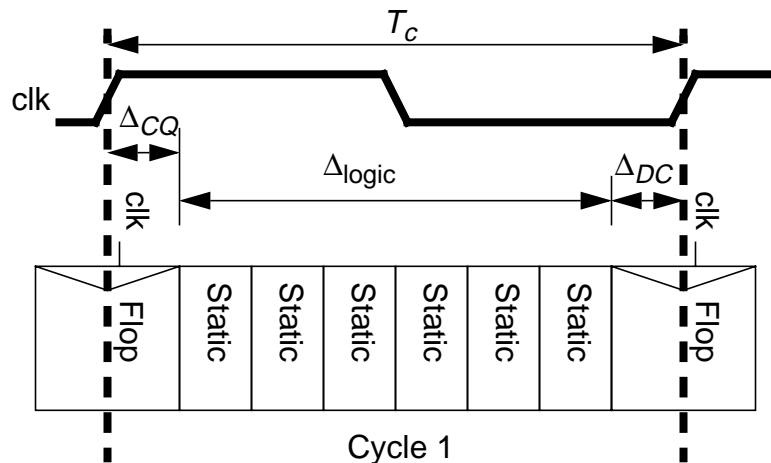
## 1.1 Overhead in Flip-Flop Systems

Most digital systems designed today use positive edge-triggered flip-flops as the basic memory element. A positive edge-triggered flip-flop is often referred to simply as an edge-

triggered flip-flop, a D flip-flop, a master-slave flip-flop, or colloquially, just a flop. It has three terminals: input  $D$ , clock  $\phi$ , and output  $Q$ . When the clock makes a low to high transition, the input  $D$  is copied to the output  $Q$ . The clock-to- $Q$  delay,  $\Delta_{CQ}$ , is the delay from the rising edge of the clock until the output  $Q$  becomes valid. The setup time,  $\Delta_{DC}$ , is how long the data input  $D$  must settle before the clock rises for the correct value to be captured.

Figure 1-1 illustrates part of a static CMOS system using flip-flops. The logic is drawn underneath the clock corresponding to when it operates. Flip-flops straddle the clock edge because input data must setup before the edge and the output becomes valid sometime after the edge. The heavy dashed line at the clock edge represents the cycle boundary. After the flip-flop, data propagates through combinational logic built from static CMOS gates. Finally, the result is captured by a second flip-flop for use in the next cycle.

**Figure 1-1 Static CMOS system with positive edge-triggered flip-flops**



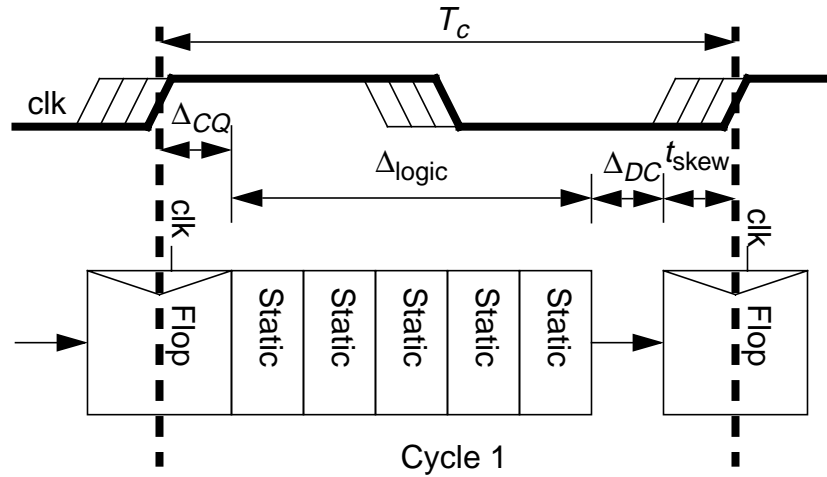
How much time is available for useful work in the combinational logic,  $\Delta_{\text{logic}}$ ? If the cycle time is  $T_c$ , we see that the time available for logic is the cycle time minus the clock-to- $Q$  delay and setup time:

$$\Delta_{\text{logic}} = T_c - \Delta_{CQ} - \Delta_{DC} \quad (1-1)$$

Unfortunately, real systems have imperfect clocks. On account of mismatches in the clock distribution network and other factors which we will examine closely in Chapter 4, the clocks will arrive at different elements at different times. This uncertainty is called clock skew and is represented in Figure 1-2 by a hash of width  $t_{\text{skew}}$  indicating a range of possi-

ble clock transition times. The bold clock lines indicate the latest possible clocks, which define worst-case timing. Data must setup before the earliest the clock might arrive, yet we cannot guarantee data will be valid until the clock-to-Q delay after the latest clock.

**Figure 1-2 Flip-flops including clock skew**



Now we see that the clock skew appears as overhead, reducing the amount of time available for useful work:

$$\Delta_{\text{logic}} = T_c - \Delta_{CQ} - \Delta_{DC} - t_{\text{skew}} \quad (1-2)$$

Flip-flops suffer from yet another form of overhead: imbalanced logic. In an ideal machine, logic would be divided into multiple cycles in such a way that each cycle had exactly the same logic delay. In a real machine, the logic delay is not known at the time cycles are partitioned, so some cycles have more logic and some have less logic. The clock frequency must be long enough for the longest cycles to work correctly, meaning excess time in shorter cycles is wasted. The cost of imbalanced logic is difficult to quantify and can be minimized by careful design, but is nevertheless important.

In summary, systems constructed from flip-flops have overhead of the flip-flop delay ( $\Delta_{DC}$  and  $\Delta_{CQ}$ ), clock skew ( $t_{\text{skew}}$ ), and some amount of imbalanced logic. We will call this total penalty the sequencing overhead<sup>1</sup>. In the next section, we will examine trends in system objectives that show sequencing overhead makes up an increasing portion of each cycle.

1. We also use the term clocking overhead, but such overhead occurs even in asynchronous, unclocked systems, so sequencing overhead is a more accurate name.

## 1.2 Throughput and Latency Trends

Designers judge their circuit performance by two metrics: throughput and latency. Throughput is the rate at which data which can pass through a circuit; it is related to the clock frequency of the system, so we often discuss cycle time instead. Latency is the amount of time for a computation to finish. Simple systems complete the entire computation in one cycle so latency and throughput are inversely related. Pipelined systems break the computation into multiple cycles called pipeline stages. Because each cycle is shorter, new data can be fed to the system more often and the throughput increases. However, because each cycle has some sequencing overhead from flip-flops or other memory elements, the latency of the overall computation gets longer. For many applications, throughput is the most important metric. However, when one computation is dependent on the result of the previous, the latency of the previous computation may limit throughput because the system must wait until the computation is done. In this section, we will review the simple relationships among throughput, latency, computation length, cycle time, and overhead. We will then look at the trends in cycle time and find that the impact of overhead is becoming more severe.

When measuring delays, it is often beneficial to use a process-independent unit of delay so that intuition about delay can be carried from one process to another. For example, if I am told that the Hewlett Packard PA8000 64-bit adder has a delay of 840 ps, I have difficulty guessing how fast an adder of similar architecture would work in my process. However, if I am told that the adder delay is equal to seven times the delay of a fanout-of-4 (FO4) inverter, where a FO4 inverter is an inverter driving four identical copies, I can easily estimate how fast the adder will operate in my process by measuring the delay of a FO4 inverter. Similarly, if I know that microprocessor A runs at 50 MHz in a 1.0 micron process and that microprocessor B runs at 200 MHz in a 0.6 micron process, it is not immediately obvious whether the circuit design of B is more or less aggressive than A. However, if I know that the cycle time of microprocessor A is 40 FO4 inverter delays in its process and that the cycle time of microprocessor B is 25 FO4 inverter delays in its process, I immediately see that B has significantly fewer gate delays per cycle and thus required more careful engineering. The fanout-of-4 inverter is particularly well-suited to expressing

delays because it is easy to determine, many designers have a good idea of the FO4 delay of their process, and because the theory of logical effort [70] predicts that cascaded inverters drive a large load fastest when each inverter has a fanout of about 4.

### 1.2.1 Impact of Overhead on Throughput and Latency

Suppose a computation involves a total combinational logic delay  $X$ . If the computation is pipelined into  $N$  stages, each stage has a logic delay  $\Delta_{\text{logic}} = X/N$ . As we have seen in the previous section, the cycle time is the sum of the logic delay and the sequencing overhead:

$$T_c = \frac{X}{N} + \text{overhead} \quad (1-3)$$

The latency of the computation is the sum of the logic delay and the total overhead of all the cycles:

$$\text{latency} = X + N \cdot \text{overhead} \quad (1-4)$$

Equations 1-3 and 1-4 show how the impact of a fixed overhead increases as a computation is pipelined into more and more stages of shorter length. The overhead becomes a greater portion of the cycle time  $T_c$ , so less of the cycle is used for useful computation. Moreover, the latency of the computation actually increases with the number of pipe stages  $N$  because of the overhead. Because latency matters for some computations, system performance can actually decrease as the number of pipe stages increases.

For example, consider a system built from static CMOS logic and flip-flops. Let the setup and clock-to-Q delays of the flip-flop be 1.5 FO4 inverter delays. Suppose the clock skew is 2 FO4 inverter delays. The sequencing overhead is  $1.5 + 1.5 + 2 = 5$  FO4 delays. The percentage of the cycle consumed by overhead depends on the cycle time, as shown in Table 1-1.:

**Table 1-1 Clocking overhead**

Cycle Time	% overhead
40	13
24	21
16	31
12	42

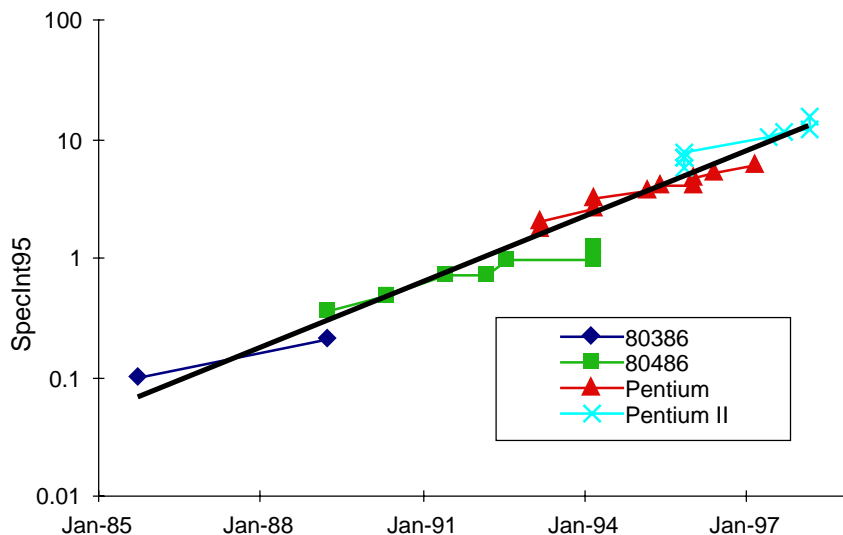
This example shows that although the sequencing overhead was small as a percentage of cycle time when cycles were long, it becomes very severe as cycle times shrink.

The exponential increase in microprocessor performance, doubling about every 18 months [32], has been caused by two factors: better microarchitectures which increase the average number of instructions executed per cycle (IPC), and shorter cycle times. The cycle time improvement is a combination of steadily improving transistor performance and better circuit design using fewer gate delays per cycle. To evaluate the importance of sequencing overhead, we must tease apart these elements of performance increase to identify the trends in gates per cycle. Let us look both at the historical trends of Intel microprocessors and at industry predictions for the future.

### 1.2.2 Historical Trends

Figure 1-3 shows a plot of Intel microprocessor performance from the 16 MHz 80386 introduced in 1985 to the 400 MHz Pentium II processors selling today [36], [50]. The performance has increased at an incredible 53% per year, thus doubling every 19.5 months.

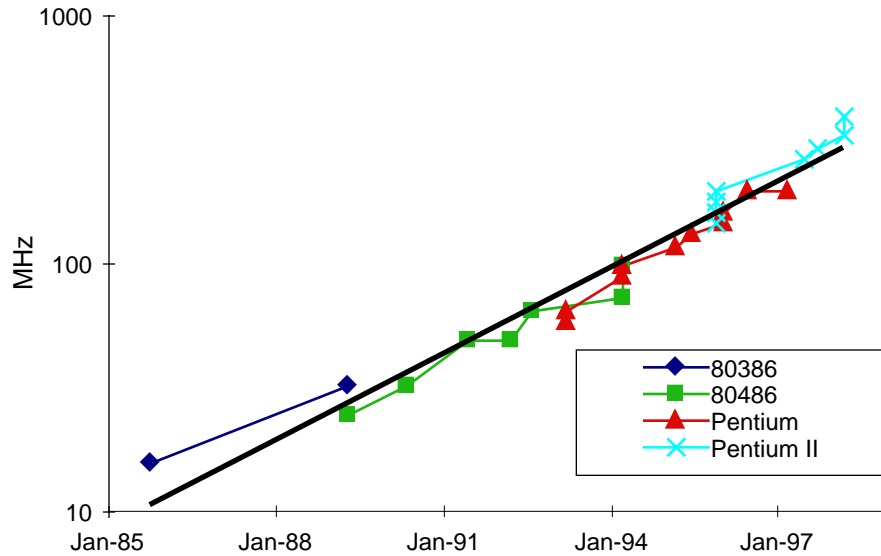
**Figure 1-3 Intel microprocessor performance<sup>1</sup>**



1. Performance is measured in SpecInt95. For processors before the 90 MHz Pentium, SpecInt95 is estimated from published MIPS data with the conversion 1 MIPS = 0.0185 SpecInt95 for these processors.

Figure 1-4 shows a plot of the processor clock frequencies, increasing at a rate of 30% per year. Some of this increase comes from faster transistors and some comes from using fewer gates per cycle.

**Figure 1-4 Intel microprocessor clock frequency**

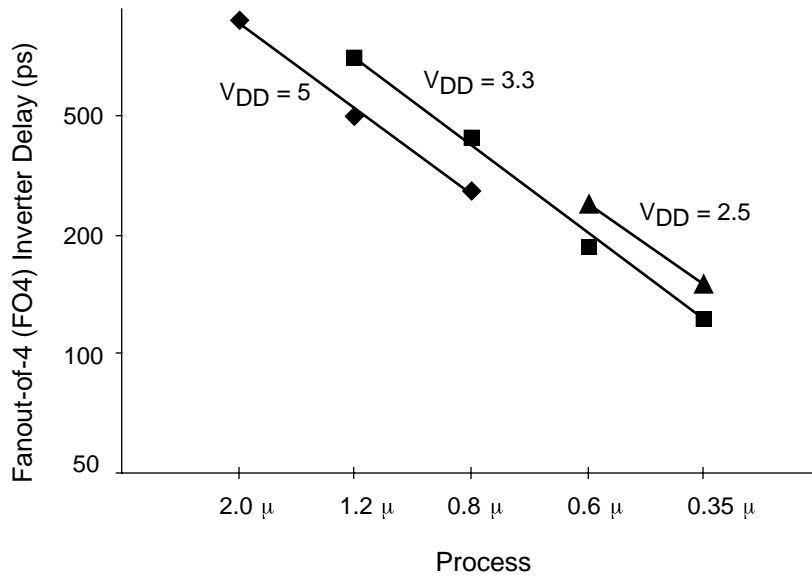


Because we are particularly interested in the number of FO4 inverters per cycle, we need to estimate how FO4 delay improves as transistors shrink. Figure 1-5 shows the FO4 inverter delay of various MOSIS processes over many years. The delays are linearly scaling with the feature size and, averaging across voltages commonly used at each feature size, are well fit by the equation:

$$\text{FO4 delay} = 475 f \tag{1-5}$$

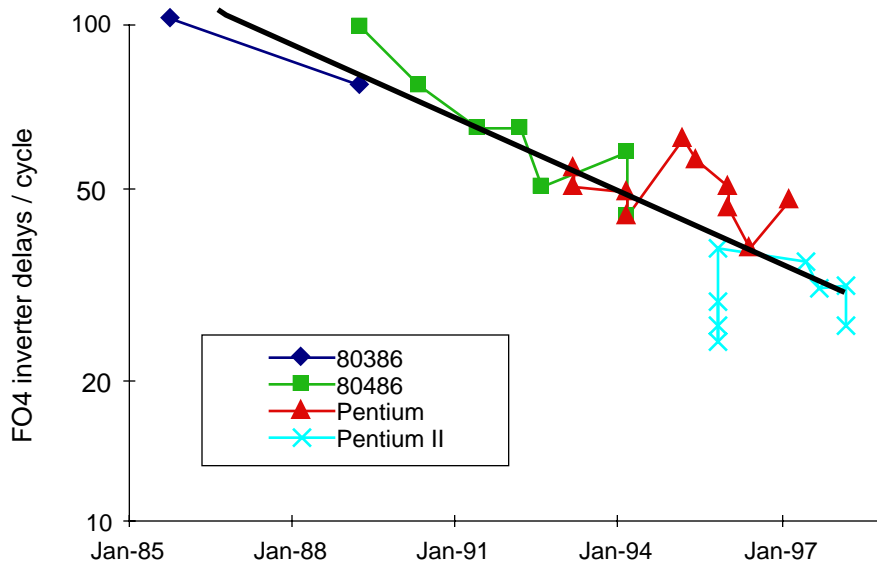
where  $f$  is the minimum drawn channel length measured in microns and delay is measured in picoseconds.

**Figure 1-5 Fanout-of-4 inverter delay trends**



Using this delay model and data about the process used in each part, we can determine the number of FO4 delays in each cycle, shown in Figure 1-6. Notice that for a particular product, the number of gate delays in a cycle is initially high and gradually decreases as engineers tune critical paths in subsequent revisions on the same process and jumps as the chip is compacted to a new process which requires retuning. Overall, the number of FO4 delays per cycle has decreased at 12% per year.

**Figure 1-6 Intel microprocessor cycle times (FO4 delays)**





Putting everything together, we find that the 1.53x annual historical performance increase can be attributed to 1.17x from microarchitectural improvements, 1.17x from process improvements, and 1.12x from fewer gate delays per cycle.

### 1.2.3 Future Predictions

The Semiconductor Industry Association (SIA) issued a roadmap in 1997 [63] predicting the evolution of semiconductor technology over the next 15 years. While such predictions are always fraught with peril, let us look at what the predictions imply for cycle times in the future:

Table 1-2 lists the year of introduction and estimated local clock frequencies predicted by the SIA for high-end microprocessors in various processes. The SIA assumes that future chips may have very fast local clocks serving small regions, but will use slower clocks when communicating across the die. The table also contains the predicted FO4 delays per cycle using a formula that:

$$\text{FO4 delay} = 400f \tag{1-6}$$

which better matches the delay of 0.25 and 0.18 micron processes. The prediction is slightly inaccurate for 1997; the fastest shipping product was the 600 MHz Alpha 21164 shown in parenthesis.

**Table 1-2 SIA Roadmap of clock frequencies**

Process (µm)	Year	Frequency (MHz)	Cycle time (FO4)
0.25	1997	750 (600)	13.3 (16.7)
0.18	1999	1250	11.1
0.15	2001	1500	11.1
0.13	2003	2100	9.2
0.10	2006	3500	7.1
0.07	2009	6000	6.0
0.05	2012	10000	5.0

This roadmap shows a 7% annual reduction in cycle time. The predicted cycle time of only 5 FO4 inverter delays in a 0.05 micron process seems at the very shortest end of feasibility because it is nearly impossible for a loaded clock to swing rail to rail in such a

short time. Nevertheless, it is clear that sequencing overhead of flip-flops will become an unacceptable portion of the cycle time.

#### 1.2.4 Conclusions

In summary, we have seen that sequencing overhead was negligible in the 1980s when cycle times were nearly 100 FO4 delays. As cycle times measured in gate delays continue to shrink, the overhead becomes more important and is now a major and growing obstacle for the design of high performance systems. We have not discussed clock skew in this section, but we will see in Chapter 4 that clock skew, as measured in gate delays, is likely to grow in future processes, making that component of overhead even worse. Clearly, flip-flops are becoming unacceptable and we need to use better design methods which tolerate clock skew without introducing overhead. In the next section, we will see how transparent latches accomplish exactly that.

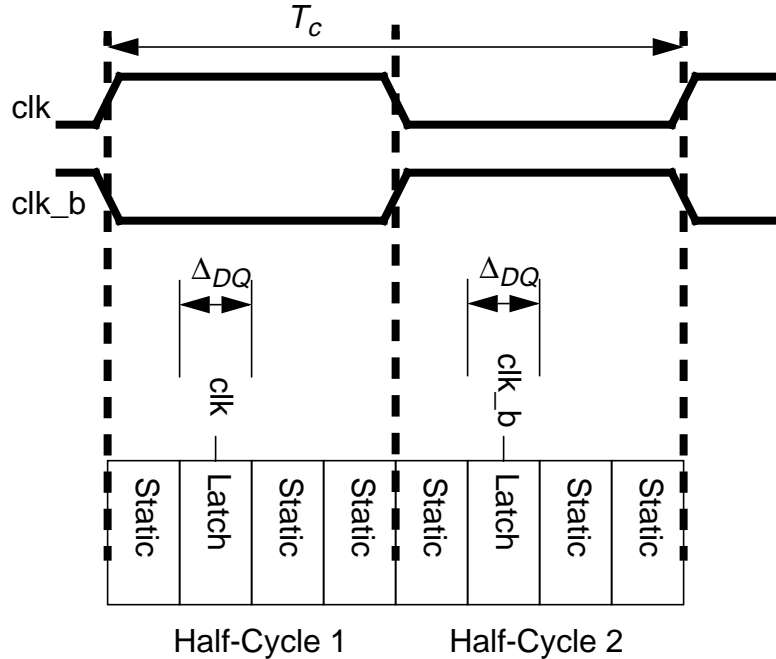
### 1.3 Skew-Tolerant Static Circuits

We can avoid the clock skew penalties of flip-flops by instead building systems from two-phase transparent latches, as has been done since the early days of CMOS [49]. Transparent latches have the same terminals as flip-flops: data input  $D$ , clock  $\phi$ , and data output  $Q$ . When the clock is high, the latch is transparent and the data at the input  $D$  propagates through to the output  $Q$ . When the clock is low, the latch is opaque and the output retains the value it last had when transparent. Transparent latches have three important delays. The clock-to-Q delay,  $\Delta_{CQ}$ , is the time from when the clock rises until data reaches the output. The D-to-Q delay,  $\Delta_{DQ}$ , is the time from when new data arrives at the input while the latch is transparent until the data reaches the output.  $\Delta_{CQ}$  is typically somewhat longer than  $\Delta_{DQ}$ . The setup time,  $\Delta_{DC}$ , is how long the data input  $D$  must settle before the clock falls for the correct value to be captured.

Figure 1-7 illustrates part of a static CMOS system using a pair of transparent latches in each cycle. One latch is controlled by  $\text{clk}$ , while the other is controlled by its complement  $\text{clk\_b}$ . In this example, we show the data arriving at each latch midway through its half-cycle. Therefore, each latch is transparent when its input arrives and incurs only a D-to-Q

delay rather than a clock-to-Q delay. Because data arrives well before the falling edge of the clock, setup times are trivially satisfied.

Figure 1-7 Static CMOS system with transparent latches

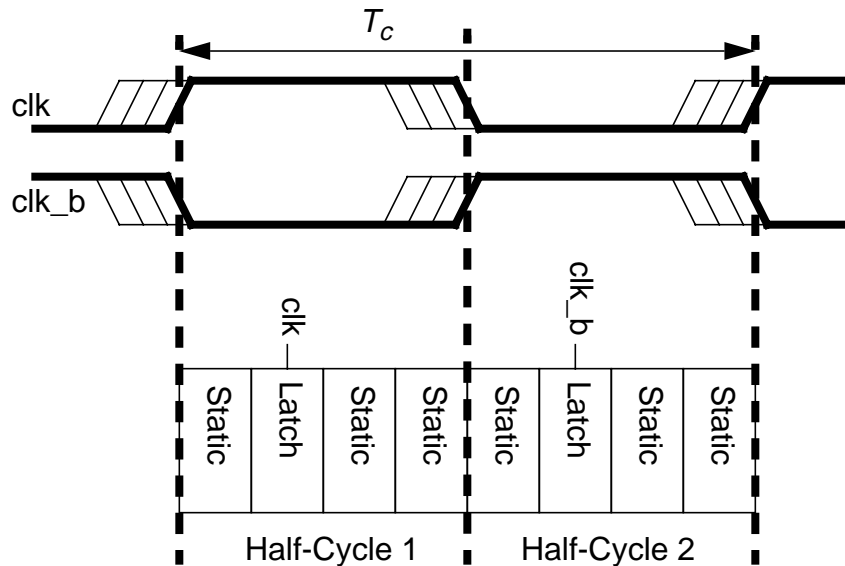


How much time is available for useful work in the combinational logic,  $\Delta_{logic}$ ? If the cycle time is  $T_c$ , we see that:

$$\Delta_{logic} = T_c - 2\Delta_{DQ} \quad (1-7)$$

Flip-flops are built from pairs of back to back latches so that the time available for logic in systems with no skew are about the same for flip-flops and transparent latches. However, transparent latch systems can tolerate clock skew without cycle time penalty, as seen in Figure 1-8. Although the clock waveforms have some uncertainty from skew, the clock is certain to be high when data arrives at the latch so the data can propagate through the latch with no extra overhead. Data still arrives well before the earliest possible skewed clock edge, so setup times are still satisfied.

Figure 1-8 Transparent latches including clock skew



Finally, static latches avoid the problem of imbalanced logic through a phenomenon called *time borrowing*, also known as *cycle stealing* by IBM. We see from Figure 1-8 that each latch can be placed in any of a wide range of locations in its half-cycle and still be transparent when the data arrives. This means that not all half-cycles need to have the same amount of static logic. Some can have more and some can have less, meaning that data arrives at the latch later or earlier without wasting time as long as the latch is transparent at the arrival time. Hence, if the pipeline is not perfectly balanced, a longer cycle may borrow time from a shorter cycle so that the required clock period is the average of the two rather than the longer value.

In summary, systems constructed from transparent latches still have overhead from the latch propagation delay ( $\Delta_{DQ}$ ) but eliminate the overhead from reasonable amounts of clock skew and imbalanced logic. This improvement is especially important as cycle times decrease, justifying a switch to transparent latches for high performance systems.

## 1.4 Domino Circuits

To construct systems with fewer gate delays per cycle, designers may invent more efficient ways to implement particular functions or may use faster gates. The increasing transistor budgets allow parallel structures which are faster; for example, adders progressed from the

compact but slow ripple carry architectures to larger carry look-ahead designs to very fast but complex tree structures. However, there is a limit to the benefits from using more transistors, so designers are increasingly interested in faster circuit families, in particular domino circuits. Domino circuits are constructed from alternating dynamic and static gates. In this section, we will examine how domino gates work and see why they are faster than static gates. Gates do not exist in a vacuum; they must be organized pipeline stages. When domino circuits are pipelined in the same way that two-phase static circuits have traditionally been pipelined, they incur a great deal of sequencing overhead from latch delay, clock skew, and imbalanced logic. By using overlapping clocks and eliminating the latches, we will see that skew-tolerant domino circuits can hide this overhead to achieve dramatic speedups.

### 1.4.1 Domino Gate Operation

To understand the benefits of domino gates, we will begin by analyzing the delay of a gate. Remember that the time to charge a capacitor is:

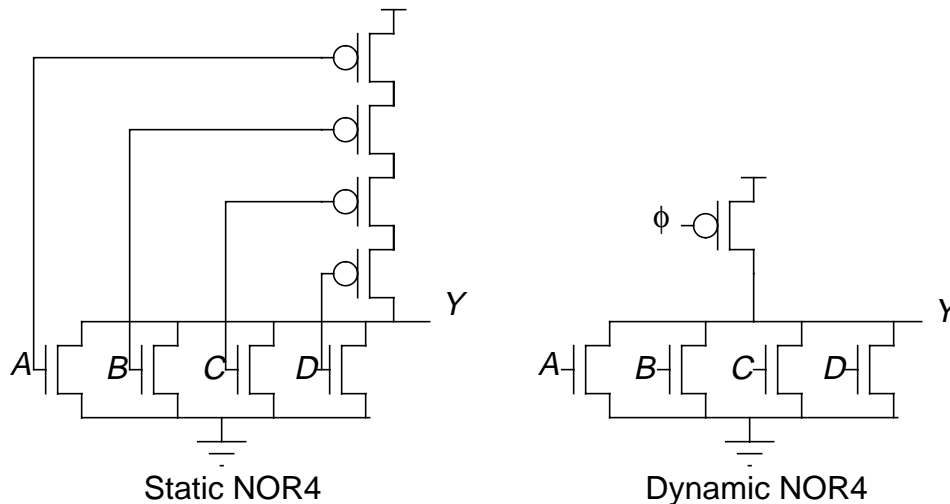
$$\Delta t = \frac{C}{I} \Delta V \quad (1-8)$$

For now we will just consider gates which swing rail to rail, so  $\Delta V$  is  $V_{DD}$ . If a gate drives an identical gate, the load capacitance and input capacitance are equal (neglecting parasitics), so it is reasonable to consider the  $C/I$  ratio of the gate's input capacitance to the current delivered by the gate as a metric of the gate's speed. This ratio is called the *logical effort* [70] of the gate and is normalized to 1 for an static CMOS inverter and is higher for more complex static CMOS gates because series transistors in complex gates must be larger and thus have more input capacitance to deliver the same output current as an inverter.

Static circuits are slow because inputs must drive both NMOS and PMOS transistors. Only one of the two transistors is on, meaning that the capacitance of the other transistor loads the input without increasing the current drive of the gate. Moreover, the PMOS transistor must be particularly large because of its poor carrier mobility and thus adds much capacitance.

A dynamic gate replaces the large slow PMOS transistors of a static CMOS gate with a single clocked PMOS transistor that does not load the input. Figure 1-9 compares static and dynamic NOR gates. The dynamic gates operate in two phases: precharge and evaluation. During the precharge phase, the clock is low, turning on the PMOS device and pulling the output high. During the evaluation phase, the clock is high, turning off the PMOS device. The output of the gate may *evaluate* low through the NMOS transistor stack.

**Figure 1-9 Static and dynamic 4-input NOR gates**

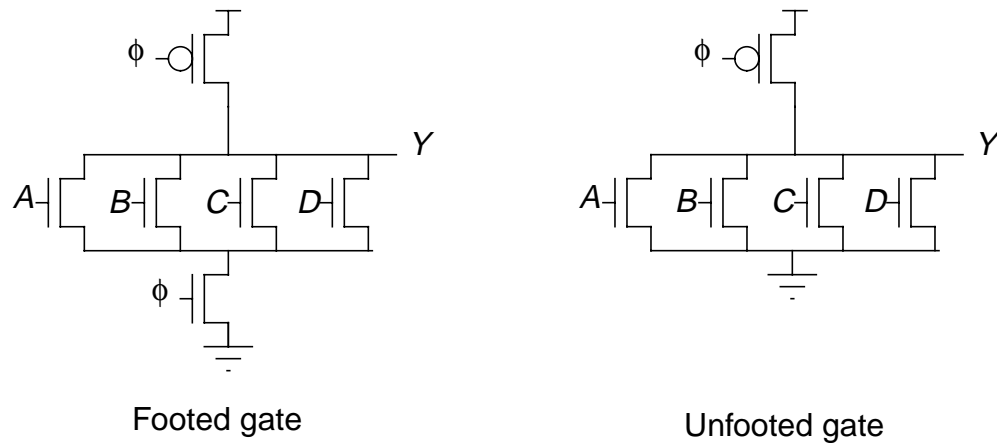


The dynamic gate is faster than the static gate for two reasons. One is the greatly reduced input capacitance. Another is the fact that the dynamic gate output begins switching when the input reaches the transistor threshold voltage,  $V_t$ . This is sooner than the static gate output, which begins switching when the input passes roughly  $V_{DD}/2$ . This improved speed comes at a cost, however: dynamic gates must obey *precharge* and *monotonicity* rules and are more sensitive to noise.

The *precharge* rule says that there must be no active path from the output to ground of a dynamic gate during precharge. If this rule is violated, there will be contention between the PMOS precharge transistor and the NMOS transistors pulling to ground, consuming excess power and leaving the output at an indeterminate value. Sometimes the precharge rule can be satisfied by guaranteeing that some inputs are low. For example, in the 4-input NOR gate, all four inputs must be low during precharge. In a 4-input NAND gate, if any

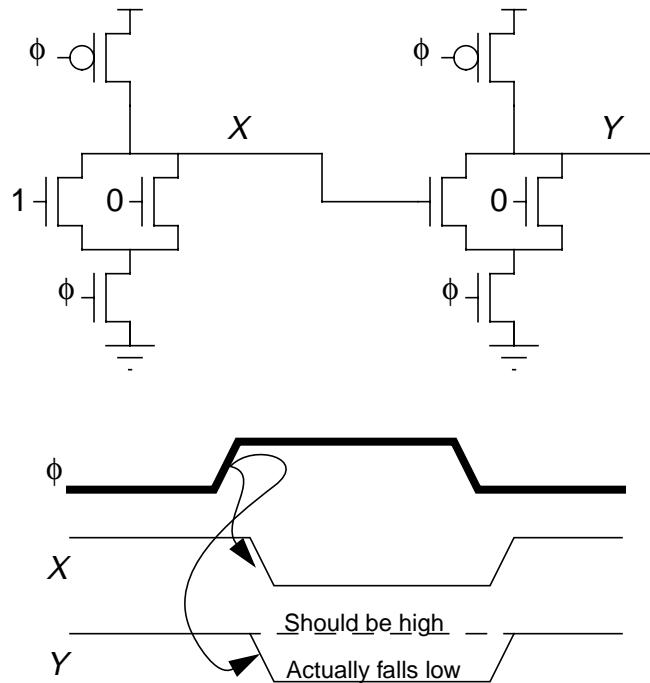
input is low during precharge, there will be no contention. It is commonly not possible to guarantee inputs are low, so often an extra *clocked evaluation transistor* is placed at the bottom of the dynamic pulldown stack, as shown in Figure 1-10. Gates with and without clocked evaluation transistors are sometimes called *footed* and *unfooted* [53]. Unfooted gates are faster but require more complex clocking to prevent both PMOS and NMOS paths from being simultaneously active.

**Figure 1-10 Footed and unfooted 4-input NOR gates**



The *monotonicity* rule states that all inputs to dynamic gates must make only low to high transitions during evaluation. Figure 1-11 shows a circuit which violates the monotonicity rule and obtains incorrect results. The circuit consists of two cascaded dynamic NOR gates. The first computes  $X = \text{NOR}(1, 0) = 0$ . The second computes  $Y = \text{NOR}(X, 0)$  which should be 1. Node X is initially high and falls as the first NOR gate evaluates. Unfortunately, the second NOR gate sees that input X is high when  $\phi$  rises and thus pulls down output Y incorrectly. Because the dynamic NOR gate has no PMOS transistors connected to the input, it cannot pull Y back high then the correct value of X arrives, so the circuit produces an incorrect result. The problem occurred because X violated the monotonicity rule by making a high to low transition while the second gate is in evaluation.

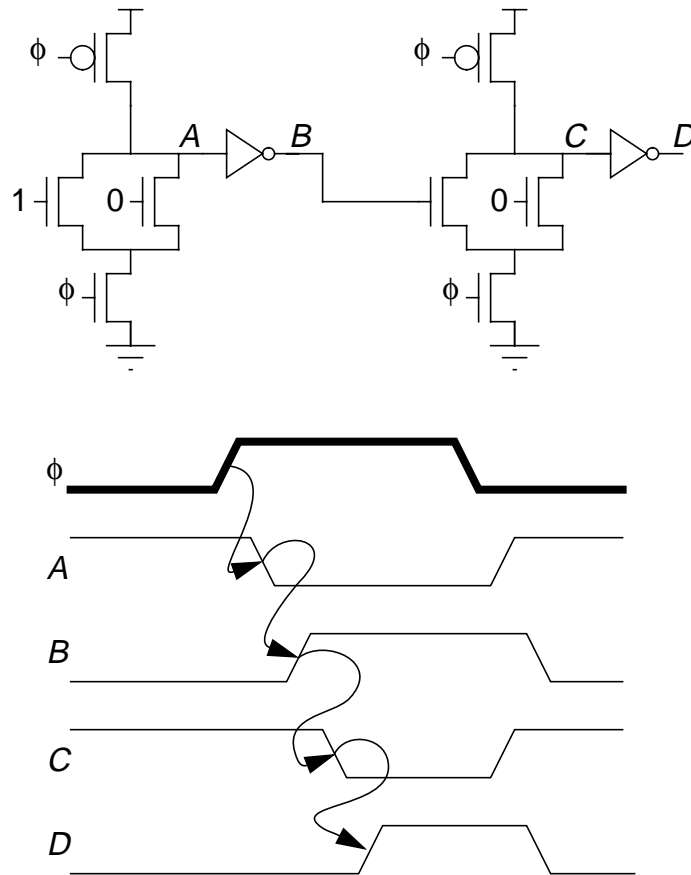
**Figure 1-11 Incorrect operation of cascaded dynamic gates**



It is impossible to cascade dynamic gates directly without violating the monotonicity rule because each dynamic output makes a high to low transition during evaluation while dynamic inputs require low to high transitions during evaluation. An easy way to solve the problem is to insert an inverting static gate between dynamic gates, as shown in Figure 1-12. The dynamic / static gate pair is called a domino gate, which is slightly misleading because it is actually two gates. A cascade of domino gates precharge simultaneously like dominos being set up. During evaluation, the first dynamic gate falls, causing the static gate to rise, the next dynamic gate to fall, and so on like a chain of dominos toppling.



**Figure 1-12 Correct operation with domino gates**



Unfortunately, to satisfy monotonicity we have constructed a pair of OR gates rather than a pair of NOR gates. In Chapter 2 we will return to the monotonicity issue and see how to implement arbitrary functions with domino gates.

Mixing static gates with dynamic gates sacrifices some of the raw speed offered by the dynamic gate. We can regain some of this performance by using *HI-skew*<sup>1</sup> static gates with wider than usual PMOS transistors [70] to speed the critical rising output during evaluation. Moreover, the static gates may perform arbitrary functions rather than being just inverters [74]. All considered, domino logic runs 1.5-2 times faster than static CMOS logic [42] and is therefore attractive enough for high speed designs to justify its extra complexity.

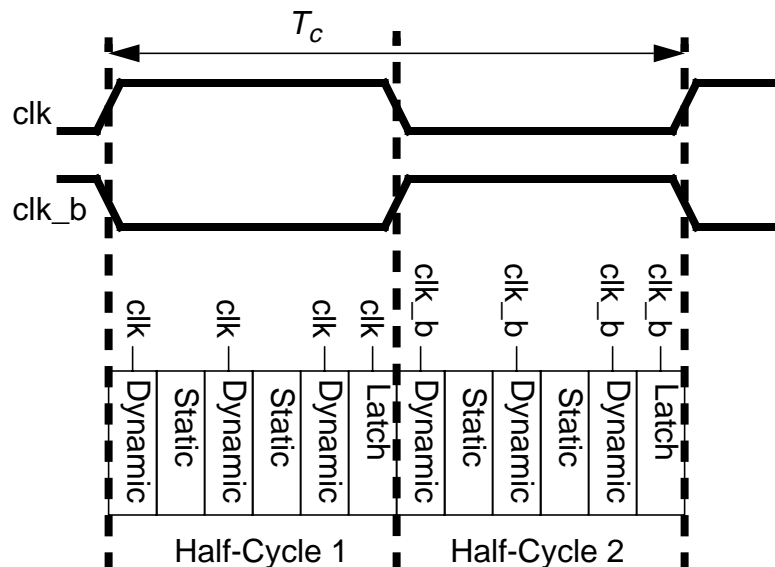
---

1. Don't confuse the word *skew* in "HI-skew" gates with "clock skew."

## 1.4.2 Traditional Domino Clocking

After domino gates evaluate, they must be precharged before they can be used in the next cycle. If all domino gates were to precharge simultaneously, the circuit would waste time during which only precharge, not useful computation, takes place. Therefore, domino logic is conventionally divided into two phases, ping-ponged such that one phase evaluates while the second precharges, then the first phase precharges while the second evaluates. In a traditional domino clocking scheme[77], latches are used between phases to sample and hold the result before it is lost to precharge, as illustrated in Figure 1-13. The scheme appears very similar to the use of static logic and transparent latches discussed in the previous section. Unfortunately, we will see that such a scheme has enormous sequencing overhead.

Figure 1-13 Traditional domino clocking scheme



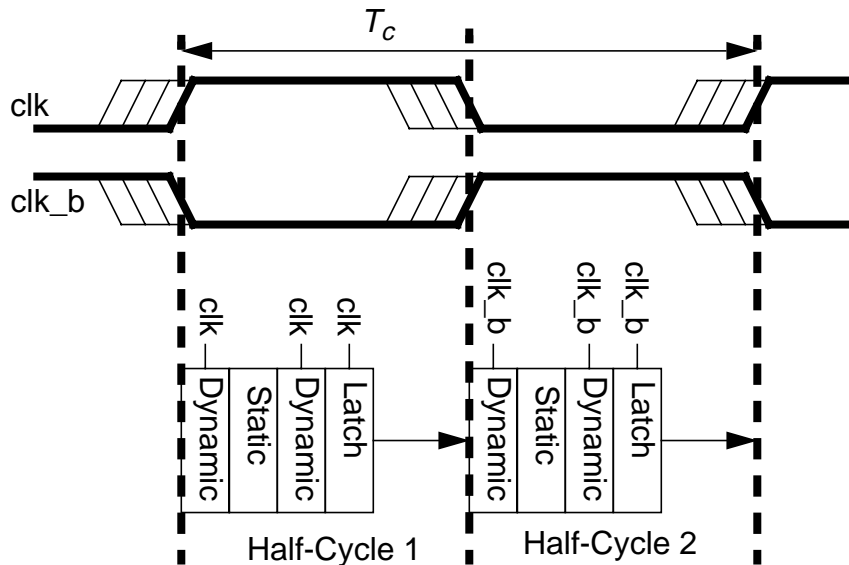
With ideal clocks, the first dynamic gate begins evaluating as soon as the clock rises. Its result ripples through subsequent gates and must arrive at the latch a setup time before the clock falls. The result propagates through the latch, so the overhead of each latch is the maximum of its setup time and  $D$ -to- $Q$  propagation delay. The latter time is generally larger, so the total time available for computation in the cycle is:

$$\Delta_{\text{logic}} = T_c - 2\Delta_{DQ} \quad (1-9)$$

Unfortunately, a real pipeline like that shown in Figure 1-14 experiences clock skew. In the worst case, the dynamic gate and latch may have greatly skewed clocks. Therefore, the dynamic gate may not begin evaluation until the latest skewed clock, while the latch must setup before the earliest skewed clock. Hence, clock skew must be subtracted not just from each cycle, as it was in the case of a flip-flop, but from each half-cycle! Assuming the sum of clock skew and setup time are greater than the latch D-to-Q delay, the time available for useful computation becomes:

$$\Delta_{\text{logic}} = T_c - 2\Delta_{DC} - 2t_{\text{skew}} \quad (1-10)$$

**Figure 1-14 Traditional domino including clock skew**



As with flip-flops, traditional domino pipelines also suffer from imbalanced logic. In summary, traditional domino circuits are slow because they pay overhead for latch delay, clock skew, and imbalanced logic.

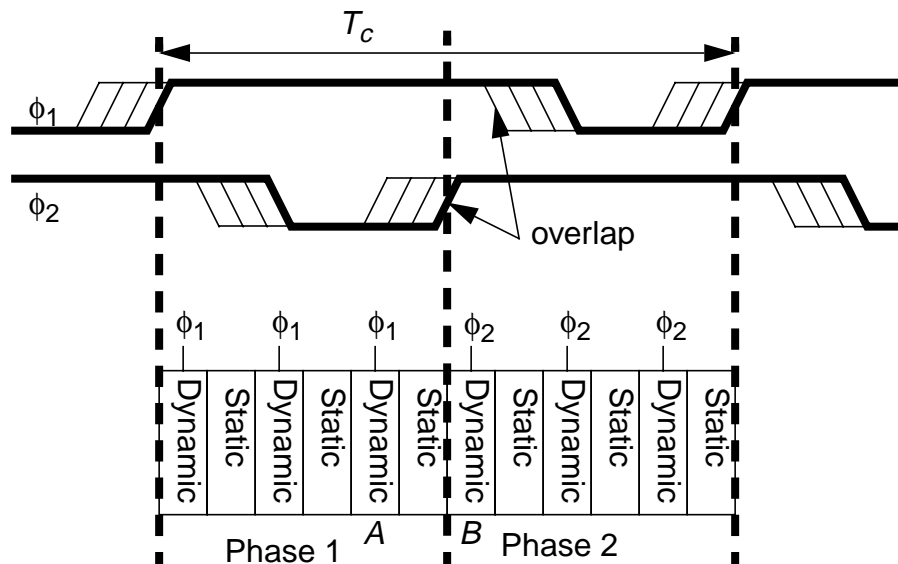
### 1.4.3 Skew-Tolerant Domino

Both flip-flops and traditional domino circuits launch data on one edge and sample it on another. These edges are called *hard edges* or *synchronization points* because the arrival of the clock determines the exact timing of data. Even if data is available early, the hard edges prevent subsequent stages from beginning early. Static CMOS pipelines with transparent latches avoided the hard edges and therefore could tolerate some clock skew and

use time borrowing to compensate for imbalanced logic. Some domino designers have recognized that this fundamental idea of softening the hard clock edges can be applied to domino circuits as well. While a variety of schemes have been invented at most microprocessor companies in the mid 1990's (e.g. [35]), the schemes have generally been held as trade secrets. This section explains how such *skew-tolerant domino* circuits operate. In Chapters 2 and 4 we will return to more subtle choices in the design and clocking of such circuits.

The basic problem with traditional domino circuits is that data must arrive by the end of one half-cycle but will not depart until the beginning of the next half-cycle. Therefore, the circuits must budget skew between the clocks and cannot borrow time. We can overcome this problem by using overlapping clocks, as shown in Figure 1-15:

**Figure 1-15 Two-phase skew-tolerant domino circuits**



This figure shows a skew-tolerant domino clocking scheme with two overlapping clock phases. Instead of using  $\text{clk}$  and its complement, we now use overlapping clocks  $\phi_1$  and  $\phi_2$ . We partition the logic into *phases* instead of half-cycles because in general we will allow more than two overlapping phases. The clocks overlap enough that even under worst case clock skews, the first gate in the second phase has time to evaluate before the last gate in the first phase begins precharge. For example, in Figure 1-15 gate *B* has time to evaluate before gate *A* precharges, even when the clocks are skewed for minimum overlap. As with

static latches, the gates are guaranteed to be ready to operate when the data arrives even if skews cause modest variation in the arrival time of the clock. Therefore we do not need to budget clock skew in the cycle time.

Another advantage of skew-tolerant domino circuits is that latches are not necessary within the domino pipeline. We ordinarily need latches to hold the result of the first phase's computation for use by the second phase when the first phase precharges. In skew-tolerant domino, the overlapping clocks insure that the first gate in the second phase has enough time to evaluate before  $\phi_1$  falls and the first phase begins precharge. When the first phase precharges, the dynamic gates will pull high and therefore the static gates will fall low. This means that the input to the second phase falls low, seemingly violating the monotonicity rule that inputs to dynamic gates must make only low to high transitions while the gates are in evaluation. What are the consequences of violating monotonicity? Gate  $B$  will remain at whatever value it evaluated to based on the results of the first half-cycle when its inputs fall low because both the pulldown transistors and the precharge transistor will be off. This is exactly what we want: gate  $B$  will keep its value even when phase 1 precharges. Hence, there is no need for a latch at the end of phase 1 to remember the result during precharge. The entire cycle is available for useful computation; there is no sequencing overhead from latch delay or clock skew:

$$\Delta_{\text{logic}} = T_c \tag{1-11}$$

Finally, skew-tolerant domino circuits can allow time borrowing if the overlap between clock phases is large compared to the clock skew. The guaranteed overlap is the nominal overlap minus uncertainty due to the clock skew. Gates in either phase 1 or phase 2 may evaluate during the overlap period, allowing time borrowing by letting gates which nominally evaluate during phase 1 run late into the second phase. As usual, it is hard to quantify the benefits of time borrowing, but it is clear the designer has greater flexibility.

In summary, skew-tolerant domino circuits use overlapping clocks to eliminate latches and remove all three sources of overhead that plague traditional domino circuits: clock skew, latch delay, and imbalanced logic. Two-phase skew-tolerant domino circuits conveniently illustrate the benefits of skew-tolerant domino design, but prove to suffer from hold time

problems and offer limited amounts of time borrowing and skew tolerance. Chapter 2 generalizes the idea to multiple overlapping clock phases and derives formulae describing the available time borrowing and skew tolerance as a function of the number of phases and the duty cycle.

## 1.5 A Look Ahead

In this chapter we have examined the sources of sequencing overhead and seen that it is a growing problem in high speed digital circuits, as summarized in Table 1-3. While flip-flops and traditional domino circuits have severe overhead, transparent latches and skew-tolerant domino circuits hide clock skew and allow time borrowing to balance logic between pipeline stages. In the subsequent chapters we will flush out these ideas to present a complete methodology for skew-tolerant circuit design of static and dynamic circuits.

**Table 1-3 Sequencing overhead**

Sequencing method	Sequencing overhead
Flip-flops	$\Delta_{CQ} + \Delta_{DC} + 2t_{\text{skew}} + \text{imbalanced logic}$
Transparent latches	$2\Delta_{DQ}$
Traditional domino	$2\Delta_{DC} + 2t_{\text{skew}} + \text{imbalanced logic}$
Skew-tolerant domino	0

Chapter 2 moves on to domino circuit design. It addresses the question of how to best clock skew-tolerant domino circuits and derives how much skew and time borrowing can be handled as a function of the number of clock phases and their duty cycles. Given these formulae, hold times, and practical clock generation issues, we conclude that four-phase skew-tolerant domino circuits are a good way to build systems.

Chapter 3 puts together static and domino circuits into a coherent skew-tolerant circuit design methodology. It looks at the interface between the two families and shows that the static to domino interface must budget clock skew, motivating the designer to build critical rings entirely in domino for maximum performance. It describes the use of timing types to verify proper connectivity in static circuits, then extends timing types to handle skew-tol-

erant domino. Finally, it addresses issues of testability and shows that scan techniques can serve both latches and skew-tolerant domino in a simple and elegant way.

None of these skew-tolerant circuit techniques would be practical if providing the necessary clocks introduced more skew than the techniques could handle. Chapter 4 addresses clock generation and distribution. Many experienced designers reflexively cringe when they hear schemes involving multiple clocks because it is virtually impossible to route more than one high speed clock around a chip with acceptable skew. Instead, we distribute a single clock across the chip and locally produce the necessary phases with the final stage clock drivers. We analyze the skews from these final drivers and conclude that although the delay variation is nonnegligible, skew-tolerant circuits are on the whole a benefit. In addition to tolerating clock skew, good systems minimize the skew which impacts each path. By considering the components of clock skew and dividing a large die into multiple clock domains, we can budget smaller amounts of skew in most paths than we must budget across the entire die.

By this point, we have developed all the ideas necessary to build fast skew-tolerant circuits. With a little practice, skew-tolerant circuit design is no harder than conventional techniques. However, it is impossible to build multimillion transistor ICs unless we have tools that can analyze and verify our circuits. In particular, we need to be able to check if our circuits can meet timing objectives given the actual skews between clocks in various domains. Chapter 5 addresses this problem of timing analysis, extending previous formulations to handle multiple domains of clock skew.

## Chapter 2 Skew-Tolerant Domino Circuits

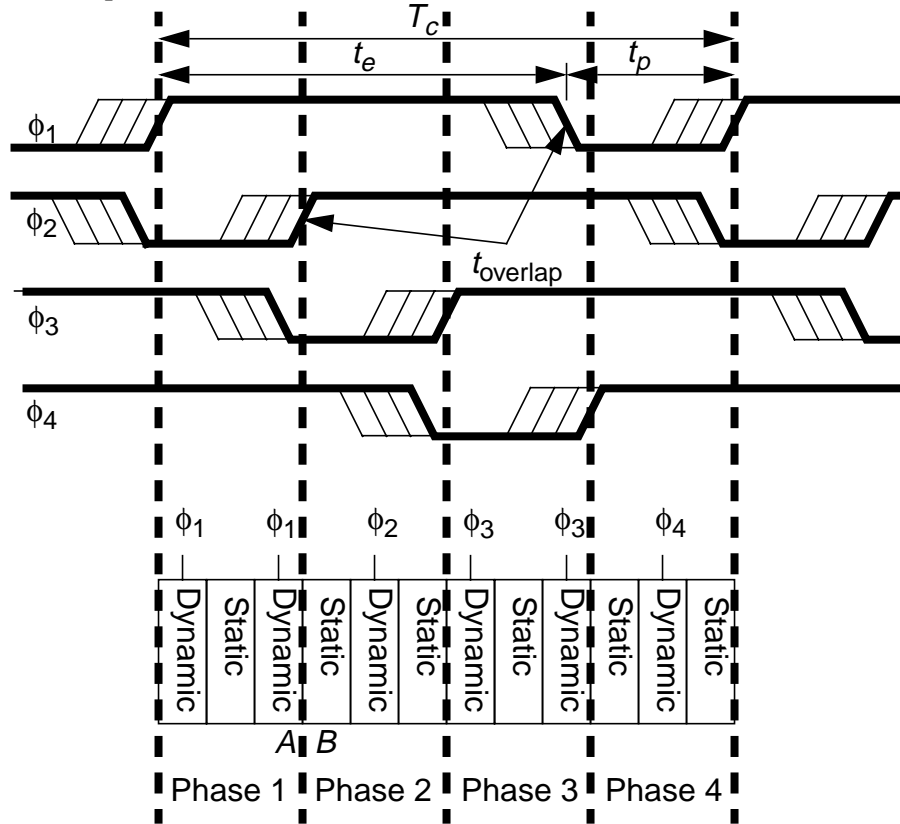
Static circuits built from transparent latches remove the hard edges of flip-flops, allowing the designer to hide clock skew from the critical path and use time borrowing to balance logic across pipeline stages. Traditional domino circuits are penalized by clock skew even more than are flip-flops, but by using overlapping clock phases it is possible to remove latches and thus build domino pipelines with zero sequencing overhead. This chapter derives the timing constraints on domino circuits which employ multiple overlapping clocks to eliminate latches and reduce sequencing overhead. The framework for understanding such systems is given the name *skew-tolerant domino* and is applicable to a variety of implementations, including many proprietary schemes developed by microprocessor designers. Once the general constraints are expressed, we explore a number of special cases which are important in practical designs. By taking advantage of the fact that clock skew tends to be less within a small region of a chip than across the entire die, we can relax some timing requirements to obtain a greater budget for global clock skew and for time borrowing. In fact, the “optimal” clocking waveforms provide more time borrowing than may be necessary and a simplified clocking scheme with 50% duty cycle clocks may be adequate and easier to employ. Another interesting case is when the designer uses either many clock phases or few gates per cycle such that each phase controls exactly one level of logic. In this case, some of the constraints can be relaxed even further. In addition to considering long paths, we also look at short paths and find the minimum delay constraints through each phase. We conclude with an example illustrating the performance benefits of skew-tolerant domino.

### 2.1 Skew-Tolerant Domino Timing

In general, let us consider skew-tolerant domino systems which use  $N$  overlapping clock phases. By symmetry, each phase nominally rises  $T_c/N$  after the previous phase and all phases have the same duty cycle. Each phase is high for an evaluation period  $t_e$  and low for a precharge period  $t_p$ . Waveforms for a four-phase system are illustrated in Figure 2-1.



Figure 2-1 Four-phase skew-tolerant domino circuits



We will assume that logic in a phase nominally begins evaluating at the latest possible rising edge of its clock and continues for  $T_c/N$  until the next phase begins. When two consecutive clock phases overlap, the logic of the first phase may run late into the time nominally dedicated to the second phase. For example, in Figure 2-1, the second  $\phi_1$  domino gate consists of dynamic gate A and static gate B. Although  $\phi_1$  gates should nominally complete during phase 1, this gate runs late and borrows time from phase 2. The maximum amount of time that can be borrowed depends on the guaranteed overlap of the consecutive clock phases. This guaranteed overlap in turn depends on the nominal overlap minus the clock skew between the phases. Therefore, the nominal overlap of clock phases  $t_{\text{overlap}}$  dictates how much time borrowing and skew tolerance can be achieved in a domino system.

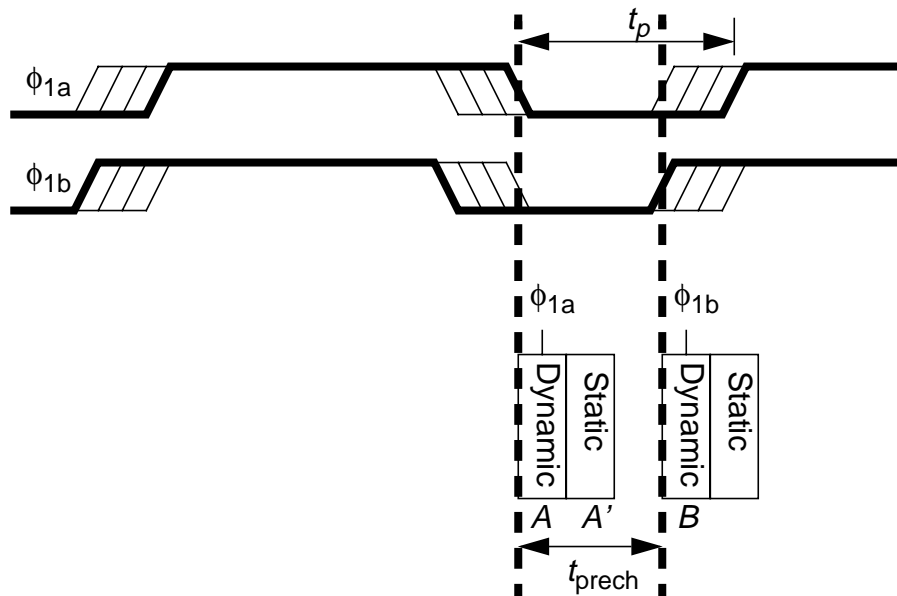
### 2.1.1 General Timing Constraints

We will analyze the general timing constraints of skew-tolerant domino by solving for this precharge period, then examining the use of the resulting overlap. Figure 2-2 illustrates the constraint on precharge time set by two consecutive domino gates in the same phase.  $t_p$  is

set by the requirement that dynamic gate  $A$  must fully precharge, flip the subsequent static gate  $A'$ , and bring the static gate's output below  $V_t$  by some noise margin before domino gate  $B$  reenters evaluation so that the old result from  $A'$  doesn't cause  $B$  to incorrectly evaluate. We call the time required  $t_{prech}$  and enforce a design methodology that all domino gates can precharge in this time. The worst case occurs when  $\phi_{1a}$  is skewed late, yet  $\phi_{1b}$  is skewed early, reducing the effective precharge window width by  $t_{skew1}$ , which is the skew between two gates in the same phase. Therefore, we have a lower bound on  $t_p$  to guarantee proper precharge:

$$t_p = t_{prech} + t_{skew1} \quad (2-1)$$

Figure 2-2 Precharge time constraint



The precharge time  $t_{prech}$  depends on the capacitive loading of each gate, so it is necessary to set an upper bound on the fanout each gate may drive. Moreover, on-chip interconnect between domino gates introduces RC delays which must not exceed the precharge time budget.

From Figure 2-1 it is clear that the nominal overlap between consecutive phases  $t_{overlap}$  is the evaluation period minus the delay between phases,  $t_e - T_c/N$ . Substituting  $t_e = T_c - t_p$ , we find:

$$t_{\text{overlap}} = T_c - t_{\text{prech}} - t_{\text{skew1}} - \frac{T_c}{N} \quad (2-2)$$

This overlap has three roles. Some minimum overlap is necessary to ensure that the later phase consumes the results before the first phase precharges. This time is called  $t_{\text{hold}}$ , though it is a slightly different kind of hold time than we have seen with latches. Additional nominal overlap is necessary so that the phases still overlap by  $t_{\text{hold}}$  even when there is clock skew. The remaining overlap after accounting for hold time and clock skew is available for time borrowing. If the clock skew between consecutive phases  $\phi_1$  and  $\phi_2$  is  $t_{\text{skew2}}$ , we therefore find:

$$t_{\text{overlap}} = \frac{N-1}{N}T_c - t_{\text{prech}} - t_{\text{skew1}} = t_{\text{hold}} + t_{\text{skew2}} + t_{\text{borrow}} \quad (2-3)$$

The hold time is generally a small negative number because the first dynamic gate in the later phase evaluates immediately after its rising clock edge while the precharge must ripple through both the last dynamic gate and following static gate of the first phase. Moreover, the gates are generally sized to favor the rising edge at the expense of slowing the precharge. The hold time does depend on the fanouts of each gate, so minimum and maximum fanouts must be specified in a design methodology to ensure a bound on hold time. For simplicity, we will sometimes conservatively approximate  $t_{\text{hold}}$  as zero.

Also, now that we are defining different amounts of clock skew between different pairs of clocks, we can no longer clearly indicate the amount skew with hashed lines in the clock waveforms. Instead, we must think about which clocks are launching and receiving signals and allocate skew accordingly. We will revisit this topic in Chapter 4.

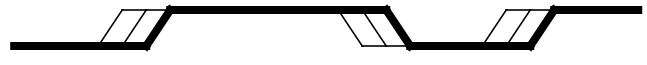




How much skew can a skew-tolerant domino pipeline tolerate? Assuming no time borrowing is used and that all parts of the chip budget the same skew,  $t_{\text{skew-max}}$ , we can solve Equation 2-3 to find:

$$t_{\text{skew-max}} = \frac{\frac{N-1}{N}T_c - t_{\text{hold}} - t_{\text{prech}}}{2} \quad (2-4)$$

For many phases  $N$  and a long cycle  $T_c$ , this approaches  $T_c/2$ , which is the same limit as for transparent latches. Small  $N$  reduces the tolerable skew because phases are more widely separated and thus overlap less. The budget for precharge and hold time further reduces tolerable skew. Notice that if the actual skew between any two dynamic gates exceeds  $t_{\text{skew-max}}$ , the circuit may fail to operate at any frequency, no matter how fast the gates within the pipeline evaluate.

For a numerical example, consider a microprocessor built from skew-tolerant domino circuits with a cycle time  $T_c = 16$  and precharge time  $t_{\text{prech}} = 4$  FO4 inverter delays. How much clock skew can the processor withstand if 2, 3, 4, 6, or 8 clock phases are used? Let us assume the hold time is 0. Figure 2-3 illustrates the clock waveforms, precharge period, and maximum tolerable skew for each number of clock phases. Notice that the precharge period must lengthen with  $N$  to accommodate the larger clock skews while still providing a minimum guaranteed precharge window.

Figure 2-3 Skew-tolerance for various numbers of clock phases

$N$	$\phi_1$ waveform	$t_p$	$t_{\text{skew-max}}$
2		6	2
3		7.33	3.33
4		8	4
6		8.66	4.66
8		9	5

### 2.1.2 Clock Domains

In Figure 2-4 we assumed that clock skew was equally severe everywhere. In real systems, however, we know that the skew between nearby elements,  $t_{\text{skew}}^{\text{local}}$ , may be much less than the skew between arbitrary elements,  $t_{\text{skew}}^{\text{global}}$ . We take advantage of this tighter bound on local skew to increase the tolerable global skew. We therefore partition the chip into multi-

ple regions, called local clock domains, which have at most  $t_{\text{skew}}^{\text{local}}$  between clocks within the domain.

If we require that all connected blocks of logic in a phase are placed within local clock domains, we obtain  $t_{\text{skew1}} = t_{\text{skew}}^{\text{local}}$ . We still allow arbitrary communication across the chip at phase boundaries, so we must budget  $t_{\text{skew2}} = t_{\text{skew}}^{\text{global}}$ . Substituting into Equation 2-3, we can solve for the maximum tolerable global skew assuming no time borrowing:

$$t_{\text{skew-max}}^{\text{global}} = \frac{N-1}{N} T_c - t_{\text{hold}} - t_{\text{prech}} - t_{\text{skew}}^{\text{local}} \quad (2-5)$$

This equation shows that reducing the local skew increases the amount of time available for global skew. In the event that local skew is tightly bounded, a second constraint must be checked on precharge that the last gate in a phase precharges before the first gate in the next phase begins evaluation extremely early because of huge global skew. The analysis of this case is straightforward, but is omitted because typical chips should not experience such large global skews.

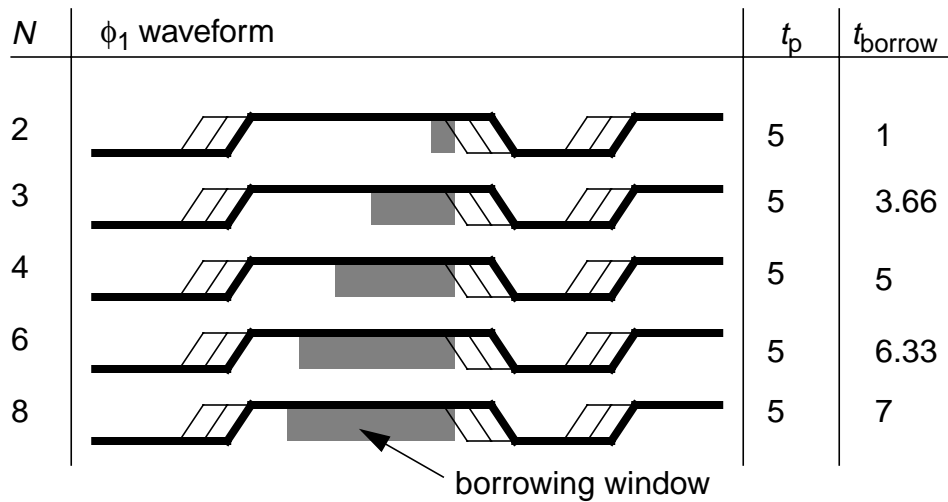
Remember that overlap can be used to provide time borrowing as well as skew tolerance; indeed, these two budgets trade off directly. Again using Equation 2-3, we can calculate the budget for time borrowing assuming fixed budgets of local and global skew:

$$t_{\text{borrow}} = \frac{N-1}{N} T_c - t_{\text{hold}} - t_{\text{prech}} - t_{\text{skew}}^{\text{local}} - t_{\text{skew}}^{\text{global}} \quad (2-6)$$

For another numerical example, again consider a microprocessor with a cycle time  $T_c = 16$  and precharge time of  $t_{\text{prech}} = 4$  FO4 inverter delays. Further assume the global skew budget is 2, the local skew budget is 1, and the hold time is 0 FO4 delays. Figure 2-4 illustrates the clock waveforms, precharge period, and maximum time borrowing for 2, 3, 4, 6, and 8 clock phases. Notice that the best clock waveforms remains the same because the clock skew is fixed, but that the amount of time borrowing available increases with  $N$ . A two-phase system offers a small amount of time borrowing which makes balancing the pipeline somewhat easier. A four-phase design offers more than a full phase of time borrowing, granting the designer tremendous flexibility. More phases offer diminishing returns. In Chapter 4, we will find that generating the four domino clocks is relatively

easy. Therefore, four-phase skew-tolerant domino is a reasonable design choice which we will use in the circuit methodology of Chapter 3.

**Figure 2-4 Time borrowing for various numbers of clock phases**



### 2.1.3 50% Duty Cycle

The previous example shows that skew-tolerant domino systems with four or more phases and reasonable bounds on clock skew may be able to borrow huge amounts of time using clock waveforms with greater than 50% duty cycle. As we will see in Chapter 4, it is possible to generate such waveforms with clock choppers, but it would be simpler to employ standard 50% duty cycle clocks, with  $t_e = t_p = T_c/2$ .

We have seen that the key metric for skew-tolerant domino is the overlap between phases,  $t_e - T_c/N$ . We know that this overlap must exceed the clock skew, hold time, and time borrowing. Substituting  $t_e = T_c/2$ , we find:

$$t_{\text{skew-max}}^{\text{global}} + t_{\text{borrow}} = \frac{N-2}{2N} T_c - t_{\text{hold}} \quad (2-7)$$

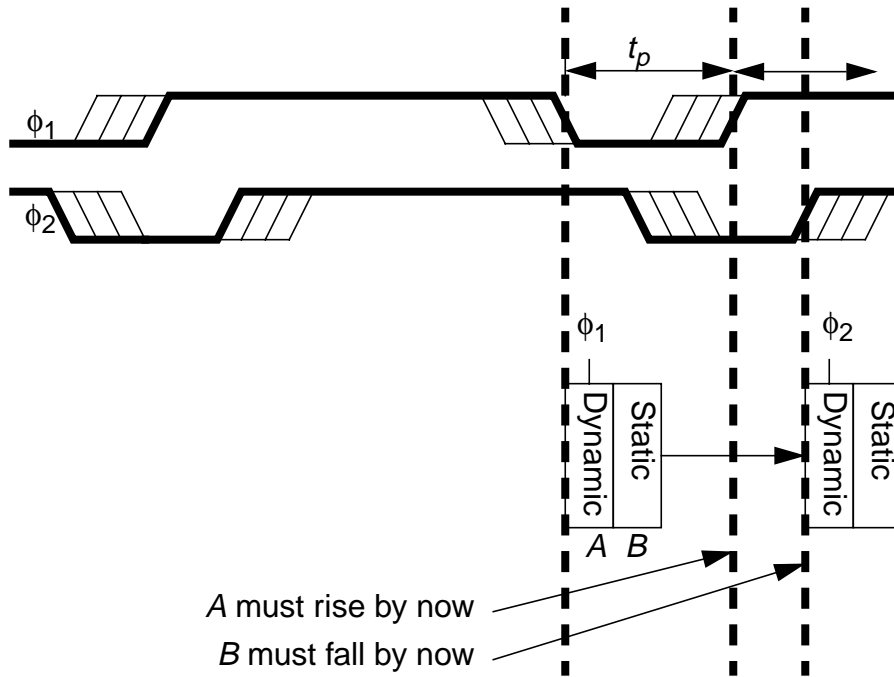
From Equation 2-7, we see that again the overlap approaches half a cycle as  $N$  gets large. Of course we must use more than 2 clock phases to obtain overlapping 50% duty cycle clocks.

Another advantage of the 50% duty cycle waveforms is that a full half-cycle is available for precharge. This may allow more time for slow precharge or may allow the designer to tolerate more skew between precharging gates, eliminating the need to place all communicating gates of a particular phase in the same local clock domain. Of course, in a system with 50% duty cycle clocks, tighter bounds on local skew do not permit greater global skew.

#### **2.1.4 Single Gate per Phase**

In the extreme case of using very many clock phases or very few gates per cycle, we may consider designing with exactly one gate per phase. The precharge constraint of Equation 2-1 requires that a gate must complete precharge before the next gate in the same phase begins evaluation so that old data from the previous cycle does not interfere with operation. Skew between the two gates in the same phase is subtracted from the time available for precharge. Because there is no next gate in the same phase when we use exactly one gate per phase, we can relax the constraint. The new constraints, shown in Figure 2-5, are that the domino gate must complete precharge before the gate in the next phase reenters evaluation, and that the dynamic gate  $A$  in the current phase must precharge adequately before the current phase reenters evaluation. In a system with multiple gates in a phase, both  $A$  and  $B$  must complete precharge by the earliest skewed rising edge of  $\phi_1$ .

Figure 2-5 Relaxed precharge constraint assuming single gate per phase



As a result of these relaxed constraints, a shorter precharge time  $t_p$  may be used, permitting more global skew tolerance or time borrowing. Alternatively, for a fixed duty cycle, more time is available for precharge.

### 2.1.5 Min-Delay Constraints

Just as static circuits have hold time constraints to avoid min-delay failure, domino circuits also have constraints that data must not race through too early.<sup>1</sup> These constraints are identical in form to those of static logic: data departing one clocked element as early as possible must not arrive at the next clocked element until  $\Delta_{CD}$  after the sampling, i.e. falling, edge of the next element.

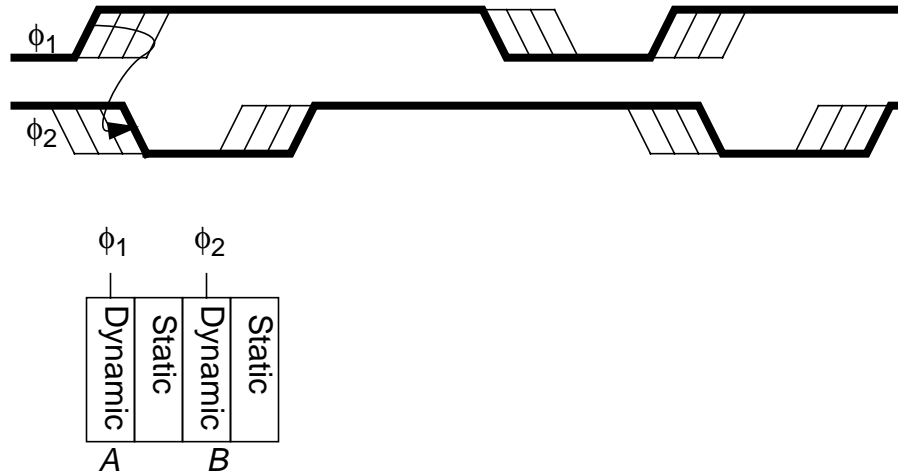
Figure 2-6 illustrates how min-delay failure could occur in skew-tolerant domino circuits by looking at the first two phases of an  $N$ -phase domino pipeline. Gate A evaluates on the rising edge of  $\phi_1$ , which in this figure is skewed early. If the gate evaluates very quickly, data may arrive at gate B before the falling edge of  $\phi_2$  as indicated by the arrow. This can

1. Do not confuse the min-delay hold time  $\Delta_{CD}$  with  $t_{hold}$ , which is the time one phase must overlap another for the first gate of the second phase to evaluate before the last gate of the first phase precharges.



contaminate the previous cycle result of gate B, which should not have received the new data until after the next rising edge of  $\phi_2$ .

**Figure 2-6 Min-delay problem in skew-tolerant domino**



The min-delay constraint determines a minimum, possibly negative, delay  $\delta_{\text{logic}}$  through each phase of domino logic to prevent such racethrough. The data must not arrive at the next phase until a hold time after the falling edge of the previous cycle of the next phase. This falling edge nominally occurs  $T_c/N - t_p$  after the rising edge of the current phase. Moreover, skew must be budgeted because the current phase might begin early relative to the next phase. In summary:

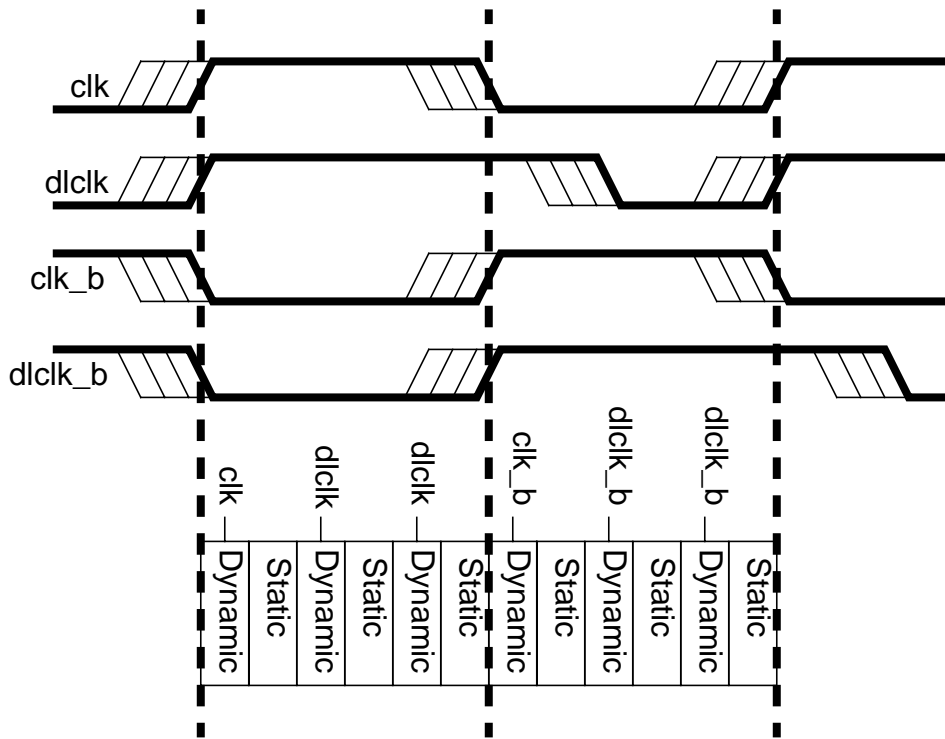
$$\delta_{\text{logic}} \geq \Delta_{CD} + t_{\text{skew}} + \frac{T_c}{N} - t_p \quad (2-8)$$

The hold time  $\Delta_{CD}$  is generally close to zero because data can safely arrive as soon as the domino gate begins precharge.

If the required  $\delta_{\text{logic}}$  is negative, the circuit is guaranteed to be safe from min-delay problems. Applying Equation 2-8 to a few cases common cases, we find two-phase skew-tolerant domino has severe min-delay problems because  $t_p < T_c/2$ , so the minimum logic contamination delay will certainly be positive. Four-phase skew-tolerant domino with 50% duty cycle clocks, however, can withstand a quarter cycle of clock skew before min-delay problems could possibly occur.

To get around the min-delay problems in two-phase skew-tolerant domino, we can introduce additional 50% duty cycle clocks at phase boundaries, as was done in Opportunistic Time-Borrowing Domino [35] and shown in Figure 2-7. dlclk and dlclk\_b play the roles of  $\phi_1$  and  $\phi_2$  in two-phase skew-tolerant domino. 50% duty cycle clocks clk and clk\_b are used at the beginning of each half-cycle to alleviate min-delay problems. However, this approach still requires four phases when the extra clocks are counted and does not provide as much skew tolerance or time borrowing as regular four-phase skew-tolerant domino, so it is not recommended.

Figure 2-7 Two-phase skew-tolerant domino with additional phases for min-delay safety [35]



### 2.1.6 Recommendations and Design Issues

In this section, we have explored how skew-tolerance, time borrowing, and min-delay vary with the duty cycle and number of phases used in skew-tolerant domino. We have found that four-phase skew-tolerant domino with 50% duty cycle clocks is an especially attractive choice for current designs because it provides reasonably good skew tolerance and time borrowing, is safe from min-delay problems that of two-phase skew-tolerant domino, and, as we will see in Chapter 4, uses a modest number of easy-to-generate clocks. Increasing the number of phases beyond four provides diminishing returns and increases

complexity of clock generation. Therefore, we will focus on four-phase systems in the next chapter as we develop a methodology to mix static logic with four-phase skew-tolerant domino. Before moving on, however, it is worth mentioning a number of design issues faced when using skew-tolerant domino circuits.

The designer must properly balance logic among clock phases, just as with transparent latches where logic must be balanced across half-cycles. It is generally necessary to include at least one gate per phase because all of the timing derivations in this chapter have worked under such an assumption. When a large number of phases are employed, it is possible to have no logic in certain phases at the expense of skew tolerance and time borrowing; for example, an eight-phase system with no logic in every other phase is indistinguishable from a four-phase system with logic in every phase. The most common reason phases contain no logic is in paths which are short. These non-critical paths can introduce domino buffers to guarantee at least one gate per phase, or may be implemented with static logic and latches because the speed of domino is unnecessary.

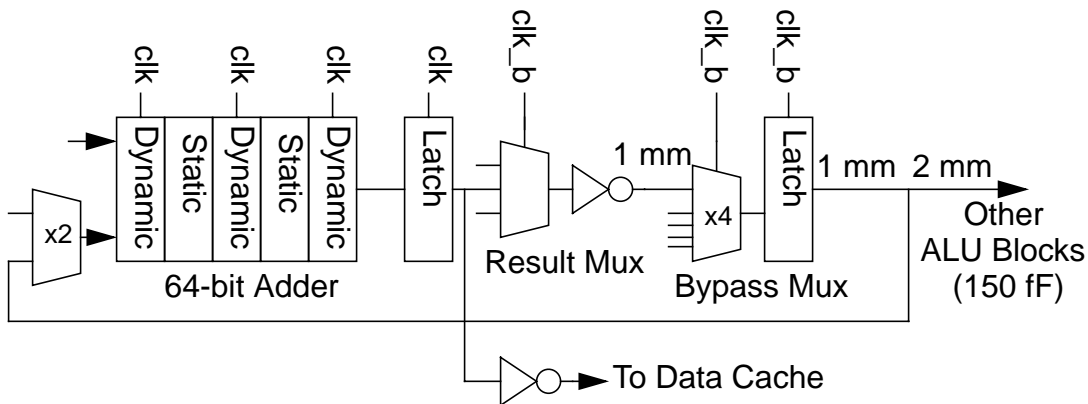
In Section 2.1.2, we showed that if we could build each phase of logic in a local clock domain, we could shorten the precharge period  $t_p$  because less skew must be budgeted between gates in the same phase. Unfortunately, in some cases critical paths and floorplanning constraints may prevent a phase of logic from being entirely within a local domain. If we do not handle such cases specially, we could not take advantage of local skew to increase tolerable global skew. Two solutions are to require that the gates at the clock domain boundary precharge more quickly or to introduce an additional phase at the clock domain crossing delayed by an amount less than  $T_c/N$ .

A final issue encountered while designing skew-tolerant domino is the interface between static and domino logic. Because nonmonotonic static logic must setup before the earliest the domino clock may rise, but the domino gate may not actually begin evaluating until the latest that its clock may rise, we have introduced a hard edge at the interface and must budget clock skew. This encourages designers to build critical paths and loops entirely in domino for maximum performance. The interface will be considered in more detail in Chapter 3.

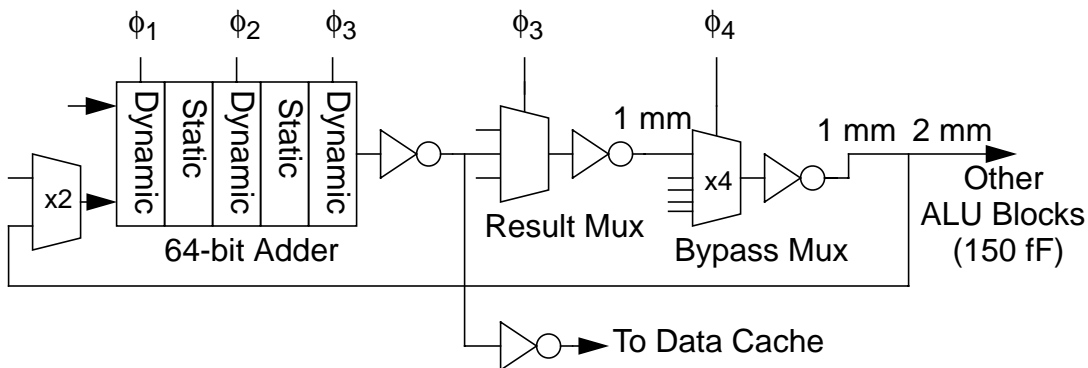
## 2.2 Simulation Results

To evaluate the performance benefits of skew-tolerant domino in the context of high speed microprocessors, we compared two 64-bit adder self-bypass paths, one constructed using traditional domino with latches and the other using four-phase skew-tolerant domino, as shown in Figure 2-8 and Figure 2-9. The paths were simulated in the HP14 0.6  $\mu\text{m}$  3-metal process with an FO4 delay of 138 ps under nominal process parameters at 3.3v, 25  $^{\circ}\text{C}$ . We assume a microarchitecture and floorplan similar to the dual integer ALUs of the DEC Alpha 21164 [4]. The adder involves two levels of carry selection implemented with dual rail domino logic. We assumed contacted diffusion parasitics on each transistor source/drain and worst-case capacitance on long signal lines, but did not model smaller wire parasitics.

**Figure 2-8 Traditional domino adder self-bypass implementation**



**Figure 2-9 Skew-tolerant domino adder self-bypass implementation**



As shown in Figure 2-10, the traditional path has a latency of 13.0 FO4 delays (1.80 ns), but a cycle time of 16.6 FO4 delays because the first half-cycle has more logic than the second. This cycle time bloating is a common problem in ALU design and is often solved in practice either by moving the latch in to the middle of the adder path, a costly choice because the bisection width of the circuit is greater within a carry-select adder so more latches are required, or stretching the first clock phase, an *ad hoc* solution with an effect similar to that systematically achieved with skew-tolerant domino.

**Figure 2-10 Simulated latency and cycle time of adder self-bypass**



The skew-tolerant path improves the latency because latches are replaced with fast inverters. Cycle time equals latency because a modest amount of time borrowing is used to balance the pipeline. The skew-tolerant waveforms are designed with  $t_p = 5.2$  FO4 delays and  $t_e = 6.7$  in order to accommodate  $t_{prech} = 4.2$  and  $t_{skew-local} = 1$ . According to Equation 2-6,  $t_{skew}^{global} + t_{borrow} = 3.7$ . In the actual circuit, we observed a global skew tolerance of 2.5 FO4 delays because some of the overlap was used for intentional time borrowing.

When a skew of 1 FO4 delay is introduced, the traditional latency increases to 15.0 FO4 delays because skew must be budgeted in both half-cycles. The skew-tolerant latency and cycle time are unaffected. Overall, the skew-tolerant design is at least 25% faster than the traditional design, achieving 600 MHz simulated operation.

## 2.3 Summary

Domino gates have become very popular because they are the only proven and widely applicable circuit family which offers significant speedup over static CMOS in commercial designs, providing a 1.5-2x advantage in raw gate delay. However, speed is determined not just by the raw delay of gates, but by the overall performance of the system. For example, traditional domino sacrificed much of the speed of gates to higher sequencing overhead. As cycle times continue to shrink, the sequencing overhead of traditional domino circuits increases and skew-tolerant domino techniques become more important. Skew-tolerant domino uses overlapping clocks to eliminate latches and remove the three sources of sequencing overhead which plague traditional domino: clock skew, latch delay, and unbalanced logic. The overlap between clock phases determines the sum of the skew-tolerance and time borrowing. Systems with better bounds on clock skew can therefore perform more time borrowing to balance logic between pipeline stages. Increasing the number of clock phases increases the overlap, but also increases complexity of local clock generation and distribution. Four-phase skew-tolerant domino, using four 50% duty cycle clocks in quadrature, is a particularly interesting design point because it provides a quarter cycle of overlap while minimizing the complexity of clock generation. The next chapter will further explore the use of skew-tolerant domino in the context of an entire system, describing a methodology compatible with other circuit techniques but which still maintains low sequencing overhead.

# Chapter 3 Circuit Methodology

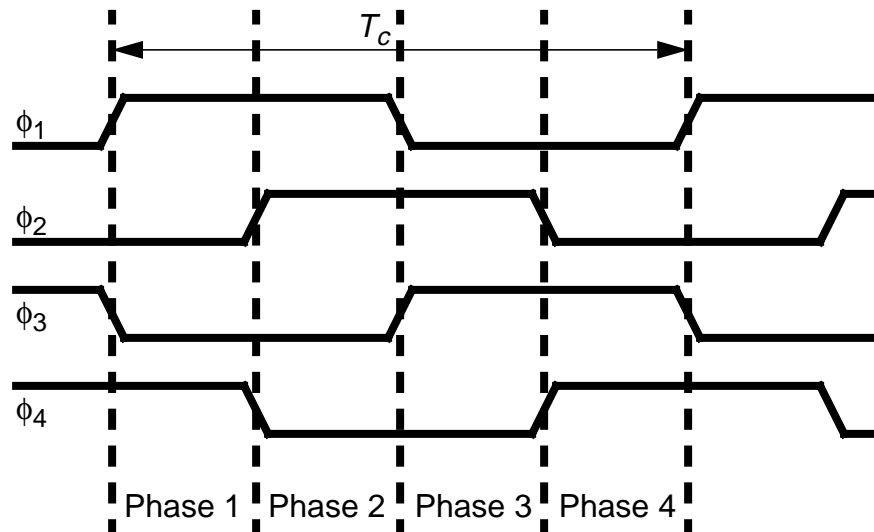
In this chapter, we will develop a skew-tolerant circuit design methodology. Our objective is a coherent approach to combine domino gates, transparent latches, and pulsed latches, while providing simple clocking, easy testability, and robust operation. This guidelines presented are self-consistent and support the design and verification of fast systems, but are not the only reasonable choices.

We will emphasize circuit design in this chapter while deferring discussion of the clock network until the next chapter. Of course circuit design and clocking are intimately related, so this methodology must make assumptions about the clocking. In particular, we assume that we are provided four overlapping clock phases with 50% duty cycles. These clocks will be used for both skew-tolerant domino and static latches.

**Definition 1:** The clocks are named  $\phi_1$ ,  $\phi_2$ ,  $\phi_3$ , and  $\phi_4$ . Their nominal waveforms are shown in Figure 3-1.

The clocks are locally generated from a single global clock  $gclk$ .  $\phi_1$  and  $\phi_3$  are logically true and complementary versions of  $gclk$ .  $\phi_2$  and  $\phi_4$  are versions of  $\phi_1$  and  $\phi_3$  nominally delayed by a quarter cycle. The clocks may be operated at reduced frequency or may even be stopped while low for testability or to save power.

Figure 3-1 Four-phase clock waveforms



The methodology primarily supports four-phase skew-tolerant domino, pulsed latches, and  $\phi_1$  and  $\phi_3$  transparent latches. Other latch phases are occasionally used when interfacing static and domino logic. It is recommended but not required to choose either transparent latches or pulsed latches as the primary static latch to simplify design.

### **3.1 Static / Domino Interface**

In the previous chapters, we have analyzed systems built from static CMOS latches and logic and systems built from skew-tolerant domino. In this section, we discuss how to interface static logic into domino paths and domino results back into static logic. We focus on static logic using transparent latches and pulsed latches because flip-flops are not tolerant of skew. We develop a set of “timing types” which determine when signals are valid and allow checking that circuits are correctly connected. The guidelines emphasize performance at the cost of checking min-delay conditions.

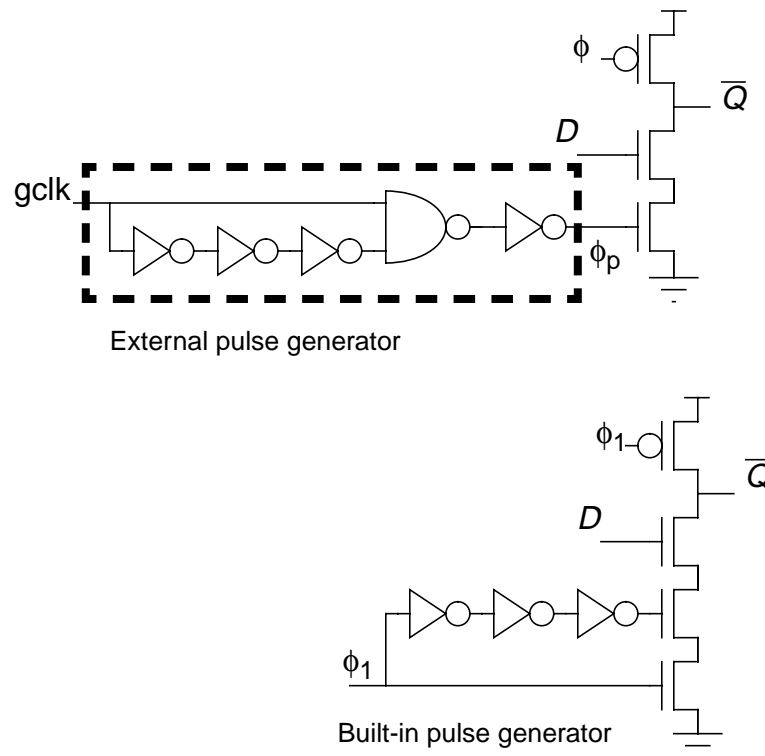
#### **3.1.1 Static to Domino Interface**

When nonmonotonic static signals are inputs to domino gates, they must be latched so that they will not change while the domino gate is in evaluation. The interface also imposes a hard edge because the data must setup at the domino input before the earliest the domino gate might begin evaluation, but may not propagate through the domino gate until the latest the gate could begin evaluation. Therefore, clock skew must be budgeted at the static to domino interface. This skew budget can be minimized by keeping the path in a local clock domain; Section 5.4 computes how much skew must be budgeted.

The latching technique at the interface depends whether transparent or pulsed latches are used. In systems using transparent latches, static logic from one half-cycle can directly interface to dynamic logic at the start of the next half-cycle after the transparent latch. The static outputs will not change while the domino is in evaluation. In systems with pulsed latches, however, the pulsed latch output may change while  $\phi_1$  domino gates are evaluating. Therefore, a modified pulsed latch must be used at the interface to produce monotonic outputs. This is called a “pulsed domino latch” and is shown in Figure 3-2.



**Figure 3-2 Pulsed domino latches with external and built-in pulsed generators**



The pulsed domino latch essentially consists of a domino gate with a pulsed evaluation clock. The pulse may either be generated externally or produced by two series evaluation transistors as shown in the figure. The former scheme yields a faster latch because fewer series transistors are necessary, but requires longer pulses.

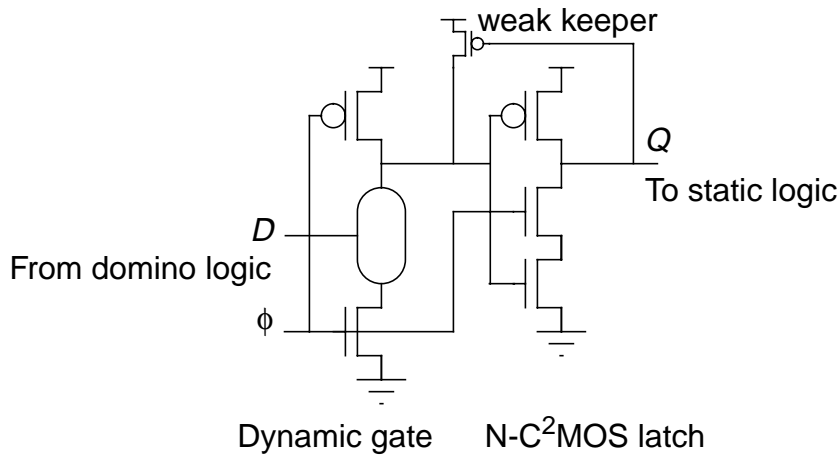
The output of a static pulsed latch may be connected through static logic to  $\phi_2$  or  $\phi_3$  domino gates, so long as the static result settles before the domino enters evaluation. Master-slave flip-flops can be interfaced the same way, but do not directly interlace to  $\phi_1$  domino gates because the output is not monotonic during  $\phi_1$ .

### 3.1.2 Domino to Static Interface

While a signal propagates through a skew-tolerant domino path, latches are unnecessary. However, before a domino output is sent into a block of static logic, it must be latched so that the result is not lost when the domino gate precharges. We will use a special latch at this interface which takes advantage of the monotonic nature of the domino outputs to improve performance.

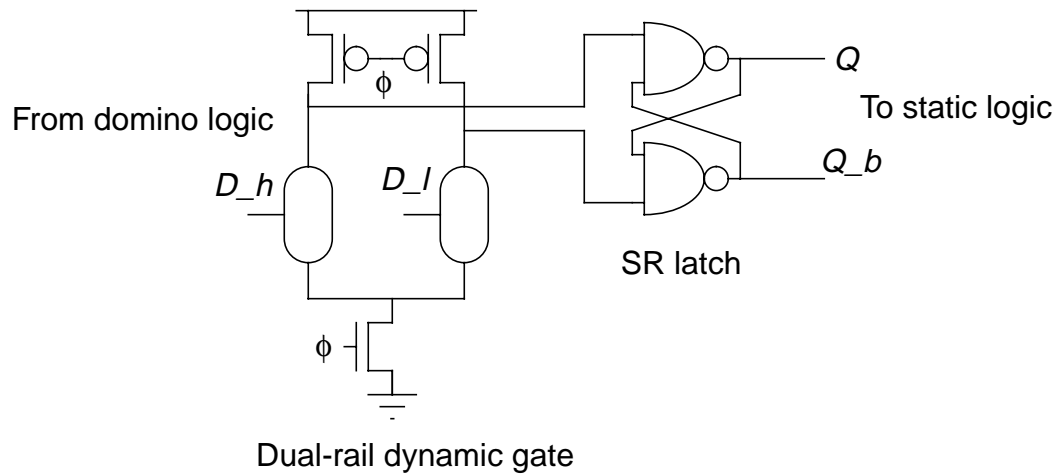
Figure 3-3 shows the interface from domino to static logic. The dynamic gate drives a special latch using a single clocked NMOS transistor. This latch is called an N-C<sup>2</sup>MOS stage by Yuan and Svensson [84]; we will sometimes abbreviate it as an N-latch. When the dynamic gate evaluates, its falling transition propagate very quickly through the single PMOS transistor in the N-C<sup>2</sup>MOS latch. When the dynamic gate precharges and its output rises, the latch turns off, holding the value until the next time the clock rises. A weak keeper improves noise immunity when the clock is high. It is important to minimize the skew between the dynamic gate and N-C<sup>2</sup>MOS latch so that precharge cannot ripple through the latch. This is easy to do by locating the two cells adjacent to one another sharing the same clock wire. In Section 3.1.3.2, we will avoid this race entirely by using a latch clock which falls before the dynamic gate begins precharge. The only overhead at the interface from dynamic to static logic is the latch propagation delay.

**Figure 3-3 Domino to static interface**



The output  $Q$  of the circuit will always fall when the clock rises, then may rise depending on the input  $D$ . This results in glitches propagating through the static logic when  $D$  is held at 1 for multiple cycles; the glitches lead to excess power dissipation. When dual-rail domino signals are available, an SR latch can be used at the domino to static interface, as shown in Figure 3-4. The SR latch avoids glitches when the domino inputs do not change, but is slower because results must propagate through two NAND gates.

**Figure 3-4 Glitch-free, but slower domino to static interface**



A regular transparent latch also can be used at the domino to static interface, but is slower than the N-C<sup>2</sup>MOS latch and has the same glitch problems.

### 3.1.3 Timing Types

The rules for connecting domino and static gates are complex enough that it is worthwhile systematically defining and checking the legal connectivity. To do this, we can generalize the two-phase clocking discipline rules of Noice [54] to handle four-phase skew-tolerant domino. Each signal name is assigned a suffix describing its timing. Proper connections can be verified by examining the suffixes. We first review the classical definition of timing types in the context of two-phase non-overlapping clocks. Most systems use 50% duty cycle clocks, so we describe how timing types apply to such systems at the expense of checking min-delay. We then generalize timing types to four-phase skew-tolerant domino, including systems which mix domino, transparent latches, and pulsed latches. Timing types also include information about monotonicity and polarity to describe domino and dual-rail domino logic.

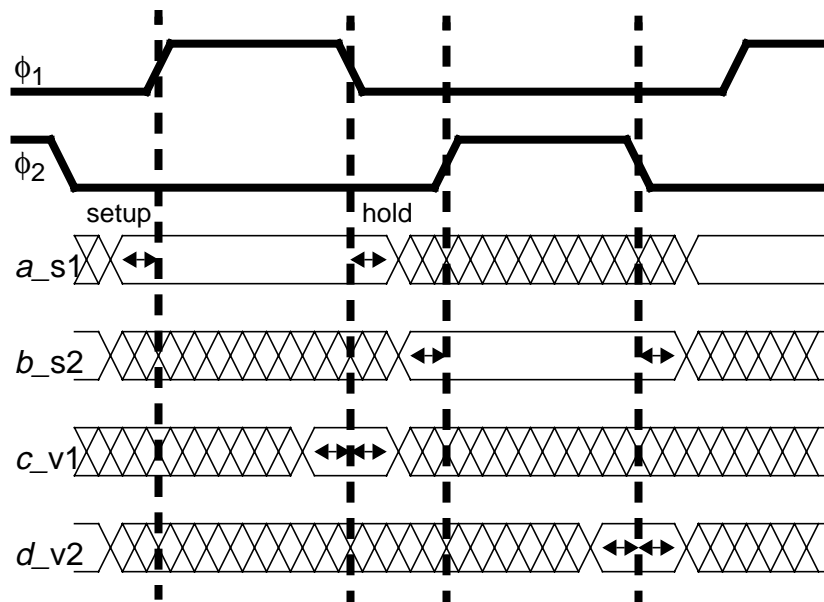
#### 3.1.3.1 Two-Phase Non-Overlapping Clocks

Systems constructed from two-phase non-overlapping clocks  $\phi_1$  and  $\phi_2$  have the pleasant property that as long as simple topological rules are obeyed, the system will have no setup or hold time problems if run slowly enough with sufficient non-overlap, regardless of

clock skew [26]. They are particularly popular in student projects because no timing analysis is necessary. Timing types are used to specify the topological rules required for correct operation and allow automatic checking of legal connectivity. In later sections, we will extend timing types to work with practical systems which do not have non-overlapping clocks. The extension comes at the expense of checking min-delay violations.

Each signal is given a suffix indicating the timing type and phase. The suffixes are `_s1`, `_s2`, `_v1`, `_v2`, `_q1`, and `_q2`. `_s` indicates that a signal is stable during a particular phase, i.e., that the signal settles before the rising edge of the phase and does not change until after the falling edge. `_v` means that the signal is valid for sampling during a phase; it is stable for some setup and hold time around the falling edge of the phase. `_q` indicates a qualified clock, a glitch-free clock signal which may only rise on certain cycles. These timing types denote which clock edge controls the stability regions of the signals, i.e. when the circuit is operated at slow speed, after which edge does the signal settle. Figure 3-5 shows examples of stable and valid signals. Stable is a stronger condition than valid; any stable signal can be used where a valid signal is required.

**Figure 3-5 Examples of stable and valid signals**

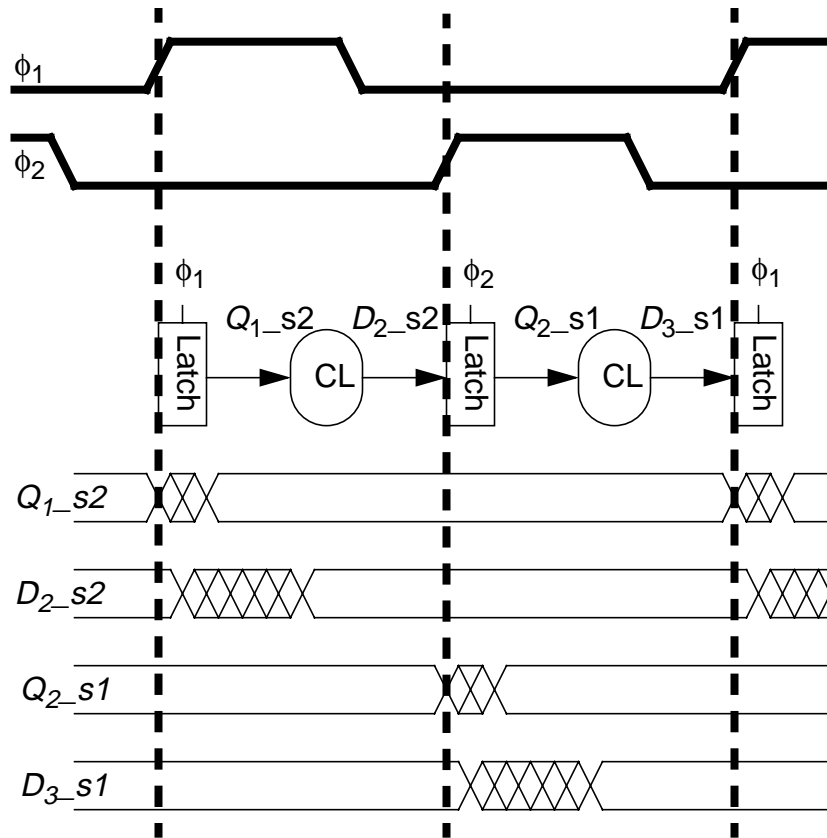


From the definitions above, we see that clocks are neither valid nor stable. They establish the time and sequence references for data signals and are never latched by other clocks. However, it is sometimes useful to provide a clock that only pulses on certain cycles. `_q`

indicates that the signal is such a qualified clock, a clock gated by some control so that it may not rise during certain cycles. Clock qualification is discussed further in Section 3.1.4. Qualified clocks are interchangeable with normal clocks for the purpose of analysis.

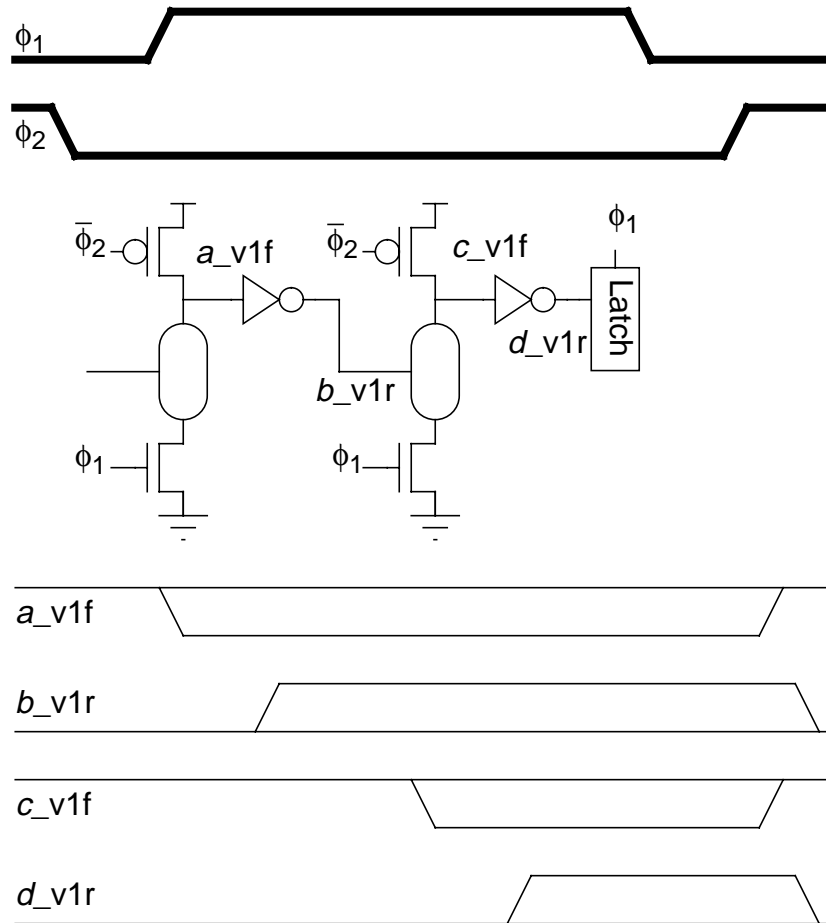
The inputs to latches must be valid a setup and hold time around the sampling edge of the latch clock. For the purpose of verifying correct operation with timing types, it is helpful to imagine operating the system at low frequency so all latch inputs arrive before the rising edge of the clock and no time borrowing is necessary. Thus, a latch output will settle sometime after the rising edge of the latch clock and will not change again until the following rising edge of the latch clock; hence it is stable throughout the other phase. If the system operates correctly at low speed, one can then increase the clock frequency, borrowing time until setup times no longer are met. In summary, a  $\phi_1$  latch requires `_v1` or `_s1` inputs and produces a `_s2` output. A  $\phi_2$  latch requires `_v2` or `_s2` inputs and produces a `_s1` output. Combinational logic does not change timing types because the system can be operated slowly enough that data is still valid or stable in the specified phase. Figure 3-6 illustrates a general two-phase system.

Figure 3-6 General two-phase system



Valid signals are produced by domino gates, as shown in Figure 3-7. The outputs settle sometime after the rising edge of the clock and do not precharge until the rising edge of the next clock, so they are valid for sampling around the falling edge of the clock. Using different precharge and evaluation clocks avoids any races between precharge and sampling. We also tag domino signals as monotonically rising (r) or falling (f). Domino inputs must be either stable or valid and monotonically rising during the phase the gate evaluates. The output of the dynamic gate is monotonically falling and the output of the inverting static gate is monotonically rising. In such a textbook domino clocking scheme, the non-overlap also appears as sequencing overhead.

**Figure 3-7 Domino gates produce valid outputs**



As long as systems using two-phase non-overlapping clocks have inputs to domino and latches using the proper timing types summarized in Figure 3-1, the systems will always function at some speed. Setup time violations caused by long paths or excessive skew are solved by increasing the clock period. Hold time violations caused by short paths or excessive skew are solved by increasing the non-overlap between phases.

**Table 3-1 Two-phase clocked element timing rules**

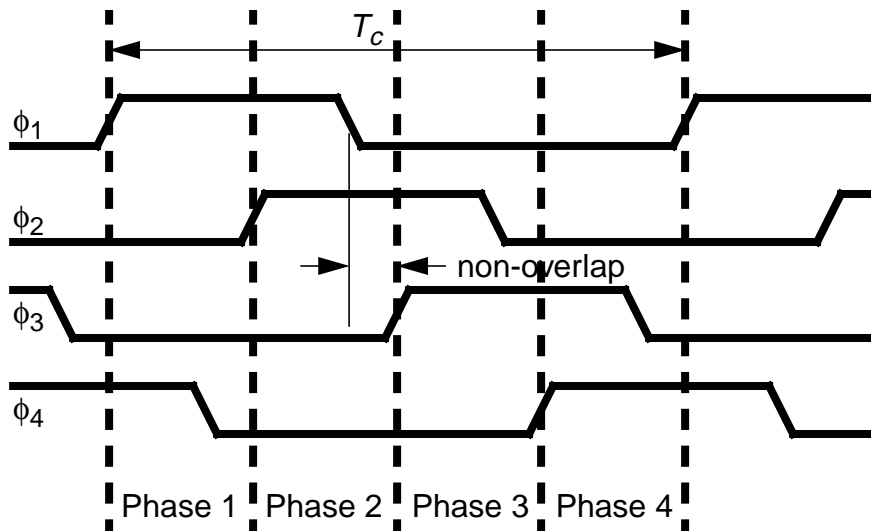
Element Type	Clock	Input	Output
Dynamic	$\phi_1, \_q1$	$\_s1, \_v1r$	$\_v1f$
	$\phi_2, \_q2$	$\_s2, \_v2r$	$\_v2f$
Transparent Latch	$\phi_1, \_q1$	$\_s1, \_v1$	$\_s2$
	$\phi_2, \_q2$	$\_s2, \_v2$	$\_s1$

Most two-phase systems use 50% duty cycle clocks rather than non-overlapping clocks. Timing types are still useful to describe legal connectivity, but clock skew can lead to hold time failures which cannot be fixed by slowing the clock. Therefore, such systems must be checked for min-delay. In essence, the definitions of  $\_v$  and  $\_s$  must change to reflect the fact that the user can no longer control how long a signal will remain constant after the falling edge of a sampling clock. Also, since the two clocks are now complementary, domino gates use the same clock for evaluation and precharge. This leads to another hold time race as domino gates precharge at the same time the latch samples. Timing types are still useful to indicate legal inputs to dynamic gates and transparent latches, but no longer guarantee immunity to min-delay problems.

### 3.1.3.2 Four-Phase Skew-Tolerant Domino

We can generalize the idea of timing types to four-phase skew-tolerant domino. Again, we will construct timing rules assuming that duty cycles can be adjusted to eliminate min-delay problems. Specifically, to avoid min-delay problems, each phase overlaps the next, but non-adjacent phases must not overlap, as shown in Figure 3-8. For example,  $\phi_1$  and  $\phi_3$  are non-overlapping. In Section 3.1.5 we will consider the min-delay races that must be checked when the non-adjacent phases may overlap. We also use timing types to describe the interface of four-phase skew-tolerant domino with transparent latches, pulsed latches, and N-C<sup>2</sup>MOS latches.

Figure 3-8 Ideal non-overlapping clock waveforms





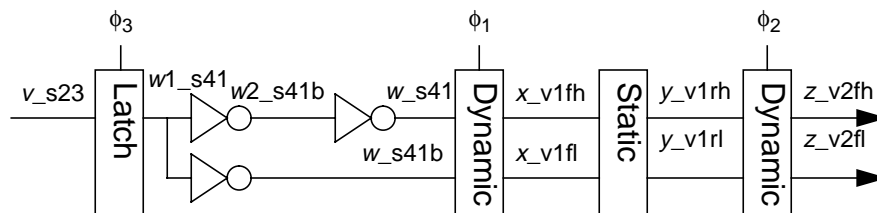
**Guideline 1:** Each signal name must have a suffix which describes the timing, phase, monotonicity, and polarity.

The timing is s, v, or q and the phase is 1, 2, 3, 4, 12, 23, 34, or 41. This is similar to two-phase timing types, but extends the definitions to describe signals which are stable through more than one phase. The monotonicity may be r for monotonically rising, f for monotonically falling, or omitted if the signal is not monotonic during the phase. These suffixes are primarily applicable to domino circuits and skewed gates. Polarity may be any one of (blank), b, h, or l. b indicated a complementary signal. h and l are used for dual-rail domino signals; when h is asserted, the result is a 1, while when l is asserted, the result is a 0. When neither is asserted, the result is not yet known, and when both are asserted, your circuit is in trouble. The signal is asserted when it is 1 for monotonically rising signals (r) and 0 for monotonically falling signals (f). Therefore, dual-rail dynamic gates produce fh and fl signals, while the subsequent dual-rail inverting static gates produce rh and rl signals. The suffix is written in the form:

signalname\_TP[M][Pol]

where T is the timing type, P is the phase, M is the monotonicity (if applicable) and Pol is the polarity (if applicable). A simple path following these conventions is shown in Figure 3-9.

**Figure 3-9 Path illustrating timing types and static to domino interface**

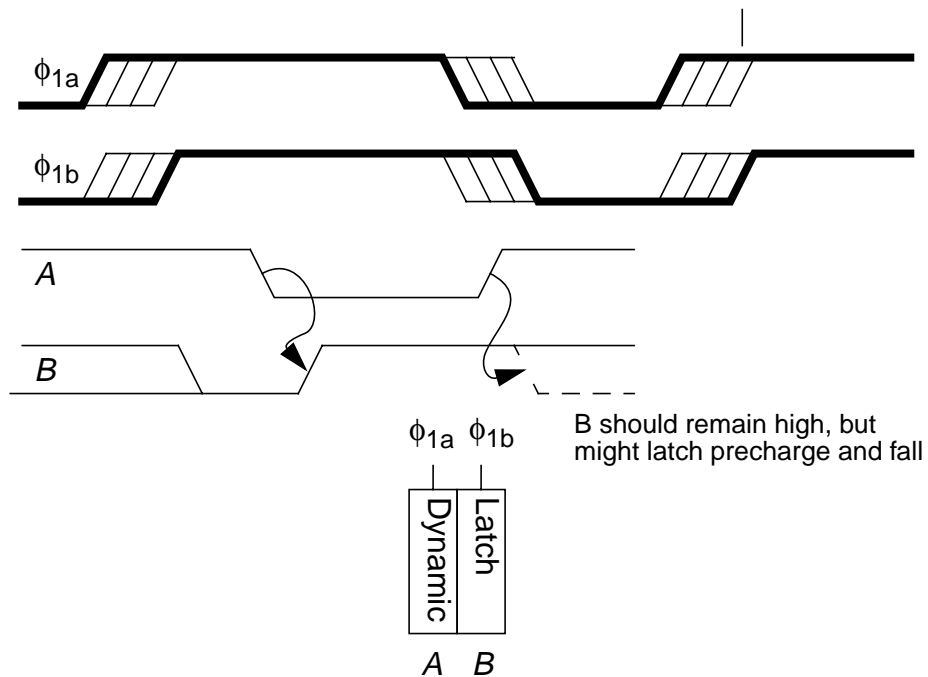


In addition to checking for correct timing and phase, we use timing types to verify monotonicity for domino gates.

Note that, unlike Figure 3-7, we now use the same clock for precharge and evaluation of dynamic gates. Therefore, the definition of a valid signal changes; a valid signal settles

before the falling edge of the clock and does not change until shortly after the falling edge of the clock. This is exactly the same timing rule as a qualified clock, so  $\_v$  and  $\_q$  signals are now in principle interchangeable. Nevertheless, we are much more concerned about controlling skew on clocks, so we reserve the  $\_q$  timing type for clock signals and continue to use  $\_v$  for dynamic outputs with the understanding that the length of the validity period is not as great as it was in a classical 2-phase system. In particular, a  $\_v1$  signal is not a safe input to a  $\phi_1$  static latch because the dynamic gate may precharge at the same time the latch samples. For example, Figure 3-10 illustrates how latch  $B$ 's output might incorrectly fall when dynamic gate  $A$  precharges if there is skew between the clocks of the two elements.

**Figure 3-10** Potential race at interface of  $\_v1$  signal to  $\phi_1$  static latch



**Definition 2:** The  $\_v$  inputs to a domino gate must be monotonically rising (r). The output of a domino gate is monotonically falling (f).

This definition formalizes the requirement of monotonicity.  $\_s$  inputs to a domino gate stabilize before the gate begins evaluation, so do not have to be monotonic.

---

**Definition 3:** Inverting static gates which accept exclusively monotonically rising inputs (r) produce monotonically falling outputs (f) and vice versa.

---

---

**Guideline 2:** Static gates should be skewed HI for monotonically falling (f) inputs and LO for monotonically rising (r) inputs.

---

Skewed gates may use different P/N ratios to favor the critical transitions and improve speed. In Section 1.4.1 we saw HI-skew gates with large P/N ratios should follow monotonically falling dynamic outputs. When a path built with static logic is monotonic, alternating HI- and LO-skew gates can be used for speed.

---

**Guideline 3:**  $\_s$  and  $\_v$  signals are the only legal data inputs to gates and latches.  $\_q$  and  $\phi$  are the only legal clock inputs.

---

This is identical to traditional two-phase conventions. Clocks and gates should not mix except at clock qualifiers (see Rule 7).

---

**Guideline 4:** The output timing types of static gates is the intersection of the input phases. If the intersection is empty, the gate is receiving an illegal set of inputs.

---

Remember that a  $\_v$  signal can be sampled for a subset of the time that a  $\_s$  signal of the same phase is stable. For example, a gate receiving  $\_s12$  and  $\_v2$  inputs produces a  $\_v2$  output. A gate receiving  $\_s12$  and  $\_s34$  inputs has no legal output timing type, so the inputs are incompatible.

---

**Guideline 5:** Table 3-2 summarizes timing rules for the most common elements in a system mixing skew-tolerant domino and transparent latches.

---

Clocked elements set the output timing type and require inputs which are valid when they may be sampled. The types depend on the clock phase used by the clocked element. See Table 3-3 for a complete list of timing rules covering more special cases.

**Table 3-2 Simplified clocked element timing guidelines**

Element Type	Clock	Input	Output
Dynamic	$\phi_1, \_q1$	$\_s1, \_v4r$ (first), $\_v1r$ (others)	$\_v1f$
	$\phi_2, \_q2$	$\_s2, \_v1r$ (first), $\_v2r$ (others)	$\_v2f$
	$\phi_3, \_q3$	$\_s3, \_v2r$ (first), $\_v3r$ (others)	$\_v3f$
	$\phi_4, \_q4$	$\_s4, \_v3r$ (first), $\_v4r$ (others)	$\_v4f$
Transparent Latch	$\phi_1, \_q1$	$\_s1$	$\_s23$
	$\phi_3, \_q3$	$\_s3$	$\_s41$
N-C <sup>2</sup> MOS Latch	$\phi_1, \_q1$	$\_v2f$	$\_s3r$
	$\phi_3, \_q3$	$\_v4f$	$\_s1r$

The output of a dynamic gate is valid and monotonically rising during the phase the gate operates, just as we have seen for two-phase systems. The input can come from static or domino logic. Static inputs must be stable  $\_s$  while the gate is in evaluation to avoid glitches. Inputs from domino logic are monotonic rising (see Rule 2) and thus only must be valid  $\_v$ . The key difference between conventional timing types and skew-tolerant timing types is that valid inputs to the first dynamic gate in each phase come from the previous phase, while inputs to later dynamic gates come from the current phase. Technically, different series stacks may receive  $\_v$  inputs from different phases.

N-latches are used at the interface of domino to static logic. Although transparent latches could also be used, they are slower and present more clock load, so are not suggested in this methodology. Notice that N-latches use a clock from one phase earlier than the dynamic gate they are latching to avoid race conditions by which that dynamic gate may precharge before the latch becomes opaque. Because of the single PMOS pullup in the N-latch, dynamic gate output  $A$  evaluating late can borrow time through the N-latch even after the latch clock falls, as shown in Figure 3-11.



must pass through N-C<sup>2</sup>MOS latches before driving static logic. Static signals can directly drive appropriate domino gates.

Figure 3-12 Examples of legal static and domino connections

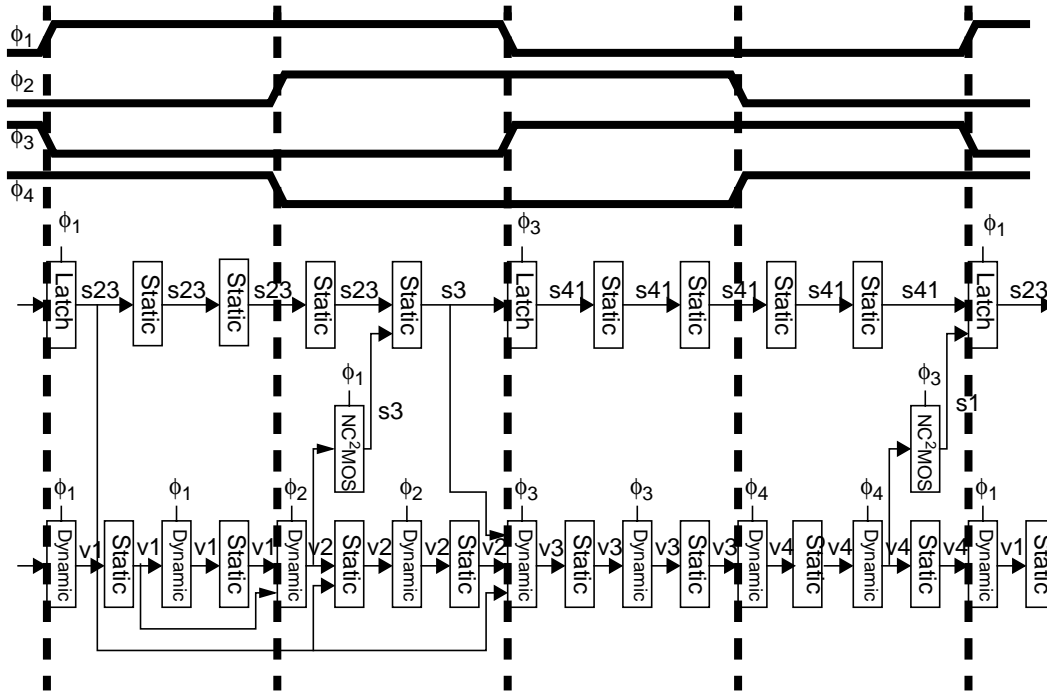
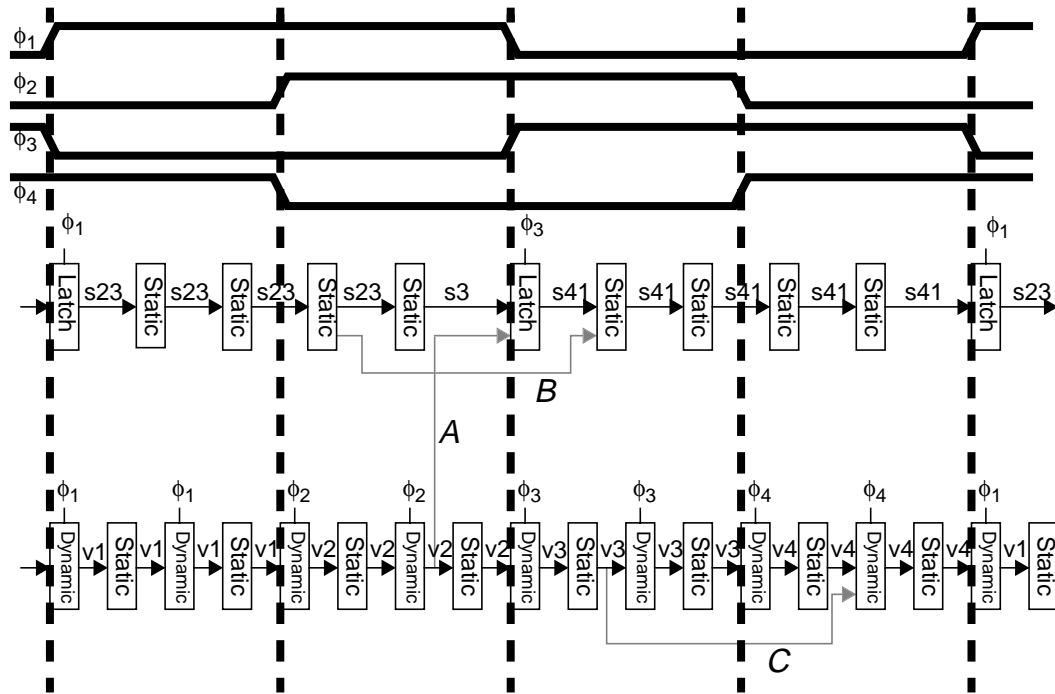


Figure 3-13 illustrates illegal connections between static and domino gates. Connection *A* is bad because the dynamic output will precharge while the  $\phi_3$  latch is transparent, allowing incorrect data to propagate into the static logic. An N-latch should have been used at the interface. Connection *B* is bad because the static path skips over a latch. Connection *C* is bad because the  $\phi_4$  domino gate receives both  $\_v3$  and  $\_v4$  inputs. By the time the  $\_v4$  input arrives, the  $\_v3$  result may have precharged. Each of these illegal connections violates the possible inputs of Table 3-2, so an automatic tool could flag these errors.

**Figure 3-13 Examples of illegal static and domino connections**



Now that we have seen how timing types work, we can expand them to handle pulsed latches and uncommon phases of latches. The complete guidelines are summarized in Table 3-3.

**Guideline 6:** Inputs and outputs of clocked elements should match the timing types defined in Table 3-3.

**Table 3-3 Complete clocked element timing guidelines**

Element Type	Clock	Input	Output
Domino	$\phi_1, \_q1$	$\_s1, \_v4r$ (first), $\_v1r$ (others)	$\_v1f$
	$\phi_2, \_q2$	$\_s2, \_v1r$ (first), $\_v2r$ (others)	$\_v2f$
	$\phi_3, \_q3$	$\_s3, \_v2r$ (first), $\_v3r$ (others)	$\_v3f$
	$\phi_4, \_q4$	$\_s4, \_v3r$ (first), $\_v4r$ (others)	$\_v4f$
Transparent Latch	$\phi_1, \_q1$	$\_s1$	$\_s23$
	$\phi_2, \_q2$ (rare)	$\_s2$	$\_s34$
	$\phi_3, \_q3$	$\_s3$	$\_s41$
	$\phi_4, \_q4$ (rare)	$\_s4$	$\_s12$
Pulsed Latch	$\phi_1, \_q1$	$\_s1, (\_s2, \_s3,) \_s4, \_v4$	$\_s23$
Pulsed Domino Latch	$\phi_1, \_q1$	$\_s1, (\_s2, \_s3,) \_s4, \_v4$	$\_v1r$
N-C <sup>2</sup> MOS Latch	$\phi_1, \_q1$	$\_v2f$	$\_s3r$
	$\phi_2, \_q2$ (rare)	$\_v3f$	$\_s4r$
	$\phi_3, \_q3$	$\_v4f$	$\_s1r$
	$\phi_4, \_q4$ (rare)	$\_v1f$	$\_s2r$

$\phi_2$  and  $\phi_4$  transparent latches are not normally used. They occasionally are useful, however, in short paths to avoid min-delay problem as we shall see in Section 3.1.5.  $\phi_2$  and  $\phi_4$  N-latches are also rare, but may be used to interface from the middle of a half-cycle of domino logic back to static logic. These rare latches have timing analogous to their more common counterparts.

Pulsed latches are controlled by a brief pulse derived from the rising edge of  $\phi_1$ . They accept any signal which will not change on or immediately after this edge, i.e.  $\_s4, \_s1$ , and  $\_v4$ . The output has the same stable time as the output of a  $\phi_1$  transparent latch because it stabilizes after the rising edge of  $\phi_1$  and does not change until after the next rising edge of  $\phi_1$ . Unfortunately, we see that the output of a pulsed latch is  $\_s23$  but neither  $\_s2$  nor  $\_s3$  signals are safe inputs to pulsed latches so it is unsafe for one pulsed latch to drive another. This matches our understanding that pulsed latches cannot be directly cas-



caded without logic between them because of hold time problems. In order to build systems with pulsed latches, we relax the timing rules to permit `_s2` and `_s3` inputs to pulsed latches, then check for min-delay on such inputs. Such checks are discussed further in Section 3.1.5.

Pulsed domino latches have the same input restrictions as pulsed latches, but produce a `_v1r` output suitable for domino logic because their outputs become valid after the rising edge of  $\phi_1$  and remain valid until the falling edge of  $\phi_1$  when the gate precharges.

### 3.1.4 Qualified Clocks

Qualified clocks are used to save power by disabling units or to build combination multiplexer-latches in which only one of several parallel latches is activated each cycle. Qualification must be done properly to avoid glitches.

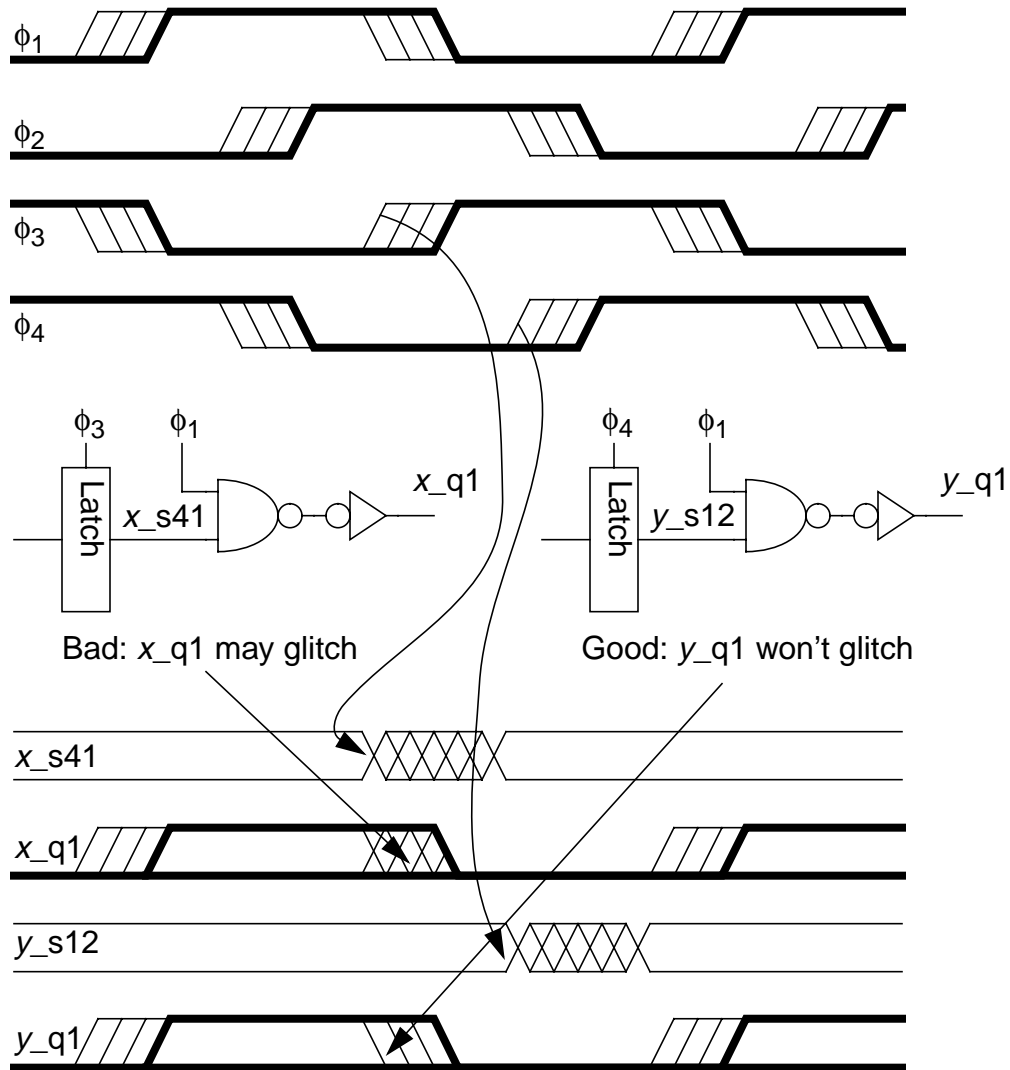
---

**Guideline 7:** Qualified clocks are produced by ANDing  $\phi_i$  with a `_s(i-1)` signal in the clock buffer.

---

To avoid problems with clock skew, it is best to qualify the clock with a signal that will remain stable long after the falling edge of the clock. For example, Figure 3-14 shows two ways to generate a `_q1` clock. The `_s` qualification signal must setup before  $\phi_1$  rises and should not change until after  $\phi_1$  falls. In the left circuit, we AND  $\phi_1$  with a `_s41` signal. If there is clock skew, the `_s41` signal may change before  $\phi_1$  falls, allowing the `_q1` clock to glitch. Glitching clocks are very bad, so the right circuit in which we AND  $\phi_1$  with a `_s12` signal is much preferred. This problem is analogous to min-delay. Like min-delay, it could also be solved by delaying the `_s41` signal so that it would not arrive at the AND gate before the falling edge of  $\phi_1$ . However, clock qualification signals are often critical, so it is unwise to delay them unnecessarily. Like min-delay, it could also be solved by making the skew between the  $\phi_1$  and  $\phi_3$  clocks in the left circuit small.

Figure 3-14 Good and bad methods of qualifying a clock



### 3.1.5 Min-Delay Checks

We have noted that a 2-phase systems usually use complementary clocks rather than non-overlapping clocks and thus lose their strict safety properties, requiring explicit checks for min-delay violations. Similarly, the 4-phase timing types of Section 3.1.3.2 use non-overlapping  $\phi_1$  and  $\phi_3$  to achieve safety, but real systems typically would use 50% duty cycle clocks. In this section, we describe where min-delay risks arise with 50% duty cycle clocks. We also examine the min-delay problems caused by pulsed latches.

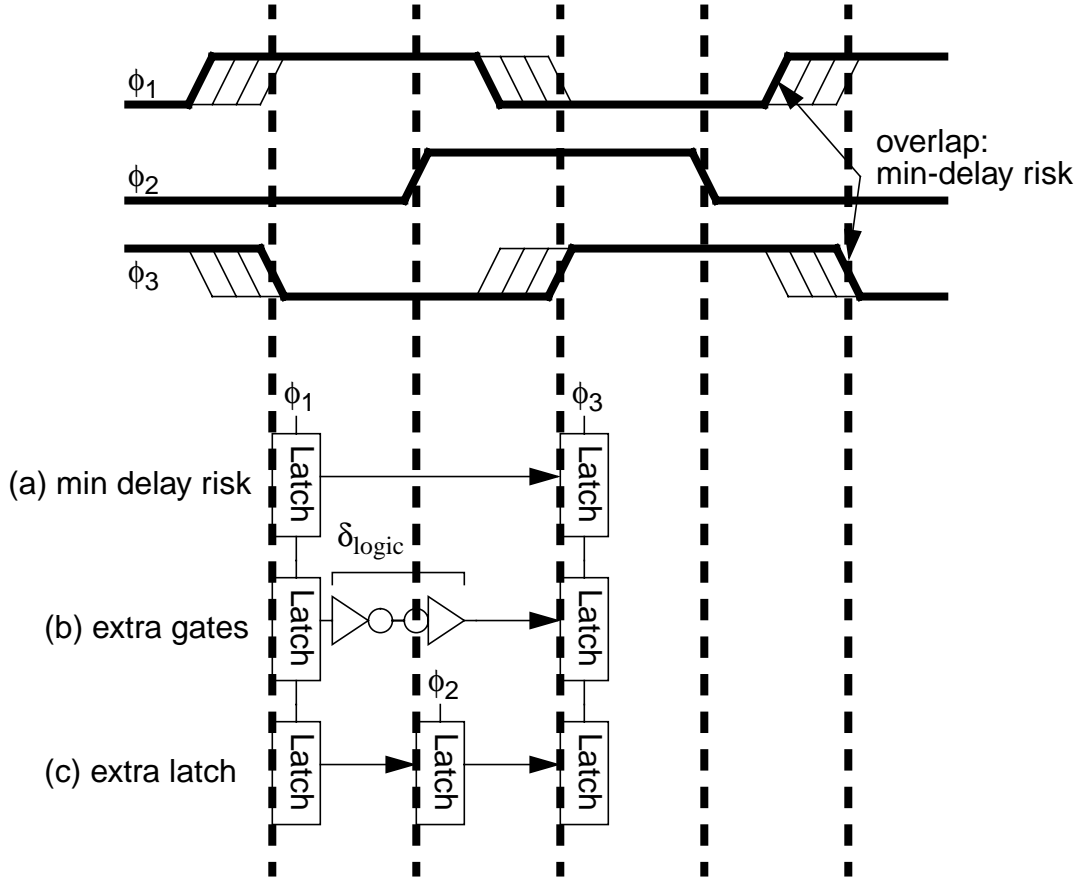
Min-delay is a serious problem because unlike setup time violations, hold time violations cannot be fixed by adjusting the clock frequency. Instead, the designer must conservatively guarantee adequate delay through logic between clocked elements. Min-delay problems should be checked at the interfaces listed in Table 3-4. The top half of the table lists common cases encountered in typical designs. The bottom half of the table lists combinations, which while technically legal according to Table 3-2, would not occur in normal use because  $\phi_2$  and  $\phi_4$  transparent latches and N-latches are seldom used.

**Table 3-4 Interfaces prone to min-delay problems**

Source Element	Source Phase	Destination Element	Destination Phase
Transparent Latch or N-Latch or Pulsed Latch	$\phi_1$	Transparent Latch or Dynamic Gate	$\phi_3$
Transparent Latch or N-Latch	$\phi_3$	Transparent Latch or Dynamic Gate	$\phi_1$
Transparent Latch or N-Latch or Pulsed Latch	$\phi_1$	Pulsed Latch or Pulsed Domino Latch	$\phi_1$
Transparent Latch or N-Latch	$\phi_2$	Transparent Latch or Dynamic Gate	$\phi_4$
Transparent Latch or N-Latch	$\phi_4$	Transparent Latch or Dynamic Gate	$\phi_2$

Min-delay problems can be solved in two ways. One approach is to add explicit delay to the data. For example, a buffer made from two long-channel inverters is a popular delay element. Another is to add a latch between the elements controlled by an intervening phase. Both approaches prevent races by slowing the data until the hold time of the second element is satisfied. Examples of these solutions are shown in Figure 3-15. In path (a) there is no logic between latches. If  $\phi_1$  and  $\phi_3$  are skewed as shown, data may depart the  $\phi_1$  latch when it becomes transparent, then race through the  $\phi_3$  latch before it becomes opaque. Path (b) solves this problem by adding logic delay  $\delta_{logic}$ . Path (c) solves the problem by adding a  $\phi_2$  latch. If the minimum required delay is large, the latch may occupy less area than a string of delay elements

Figure 3-15 Solution to min-delay problem between  $\phi_1$  and  $\phi_3$  transparent latches



Min-delay problems can actually occur at any interface, not just those listed in Table 3-4. For example, if clock skew were greater than a quarter cycle, min-delay problems could occur between  $\phi_1$  and  $\phi_2$  transparent latches. Because it is very difficult to design systems when the clock skew exceeds a quarter cycle, we will avoid such problems by requiring that the clock have less than a quarter cycle of skew between communicating elements.

Depending on the clock generation method, a few other connections may incur races. It is possible to construct clock generators with adjustable delays so that as the frequency reduces, the delay between each phase does not change. However, as we will see in Section 4.2.1, it may be more convenient to produce  $\phi_2$  and  $\phi_4$  by delaying  $\phi_1$  and  $\phi_3$ , respectively, by a fixed amount. Such clock generators introduce the possibility of races which are frequency-independent because the delay between phases is fixed.

One such risky connection is a  $\phi_1$  pulsed latch feeding a  $\phi_2$  domino gate. There is a max-delay condition that data must setup at the input of the domino gate before the gate enters evaluation. Clock skew between  $\phi_1$  and  $\phi_2$  reduces the nominally available quarter cycle. Since the delay from  $\phi_1$  to  $\phi_2$  is constant, if domino input does not set up in time, the circuit will fail at any clock frequency. The same problem occurs at the interface of a  $\phi_1$  transparent latch to  $\phi_2$  domino and of a  $\phi_3$  transparent latch to  $\phi_4$  domino.

Another min-delay problem occurs between  $\phi_2$  transparent latches and  $\phi_1$  pulsed latches or pulsed domino latches. Again, if the delay between phases is independent of frequency, hold time violations cannot be fixed by adjusting the clock frequency.

## 3.2 Clocked Element Design

This section offers guidelines on the design of fast clocked elements. Remember that static CMOS logic uses either transparent latches or pulsed latches. Domino logic uses no latches at all, except at the interface back to static logic where N-latches should be used. We postpone discussion of supporting scan in clocked elements until Section 3.3.

Critical paths should be entirely domino wherever possible because one must budget skew and latch propagation delay when making a transition from static back into domino logic; moreover time borrowing is not possible through the interface. Because most functions are nonmonotonic, this frequently dictates using dual-rail domino. In certain cases, dual-rail domino costs too much area, routing, or power. For high speed systems, going entirely static may be faster than mixing domino and static and paying the interface overhead. If the overhead is acceptable because skew is tightly controlled, try combining as much of the nonmonotonic logic into static gates at the end of the block.

### 3.2.1 Latch Design

---

**Guideline 8:** Generally use only pulsed latches or  $\phi_1$  and  $\phi_3$  transparent latches.

---

We select two phases to be the primary static latch clocks to resemble traditional two-phase design. The  $\phi_2$  and  $\phi_4$  clocks would be confusing if generally used, so they are restricted to use to solve min-delay problems in short paths.

---

**Guideline 9:** Use a N-C<sup>2</sup>MOS latch on the output of domino logic driving static gates as shown in Figure 3-3. Use a full keeper on the output for static operation.

---

Again, the output is a dynamic node and must obey dynamic node noise rules. The N-latch is selected because it is faster and smaller than a tri-state latch and doesn't have the charge sharing problems seen if a domino gate drove a transmission gate latch.

---

**Guideline 10:** The domino gate driving an N-latch should be located adjacent to the latch and should share the same clock wire and  $V_{DD}$  as the latch.

---

The N-latch has very little noise margin for noise on the positive supply. This noise can be minimized by keeping the latch adjacent to the domino output, thereby preventing significant noise coupling or  $V_{DD}$  drop. The latch is also sensitive to clock skew because if it closed too late, it could capture the precharge edge of the domino gate. Therefore, the same clock wire should be used to minimize skew.

### 3.2.2 Domino Gate Design

The guidelines in this section cover keepers, charge sharing noise, and unfooted domino gates.

---

**Guideline 11:** All dynamic gates must include a keeper.

---

The keeper is necessary for static operation on  $\phi_3$  and  $\phi_4$  dynamic gates when the clock is stopped low. It is also necessary on all gates to achieve reasonable noise immunity. Breaking this guideline requires extremely careful attention to dynamic gate input noise margins.

---

**Guideline 12:** The first dynamic gate of phase 3 must include a full keeper.

---

This is necessary to prevent the outputs of the first phase 3 gates from floating when the clock is stopped low and the phase 2 gates precharge. Note that because the first dynamic gate of phase 1 does not include a full keeper, the clock should not be stopped high long enough for the output to be corrupted by subthreshold leakage. Of course, this guideline is

an artifact of the methodology: an alternative methodology which stopped the clock high or allowed clock stopping both high and low would require the full keeper on phase 1. In Section 3.3.2 we will see that the last dynamic gate of phase 4 may also need a full keeper to support scan.

---

**Guideline 13:** The output of a dynamic gate must drive the gate, not source/drain input of the subsequent gate.

---

The result of a dynamic gate is stored on the capacitance of the output node, so this guideline prevents charge-sharing problems. An important implication is that dynamic gates cannot drive transmission gate multiplexer data inputs, although they could drive tri-state based multiplexers.

---

**Guideline 14:** Use footed dynamic gates exclusively.

---

This guideline is in place to avoid excess power consumption which may occur when the pulldown transistors are all on while the gate is still precharging. It may be waived on the first  $\phi_2$  and  $\phi_4$  gates of each cycle so long as the inputs of the gates come from  $\phi_1$  or  $\phi_3$  domino logic which does not produce a rising output until the  $\phi_2$  or  $\phi_4$  gates have entered evaluation. Aggressive designers may waive the guideline on other dynamic gates if power consumption is tolerable.

### 3.2.3 Special Structures

In a real system, skew-tolerant domino circuits must interface to special structures such as memories, register files, and programmable logic arrays (PLAs). Precharged structures like register files are indistinguishable in timing from ordinary domino gates. Indeed, standard 6-transistor register cells can produce dual-rail outputs suitable for immediate consumption by dual-rail domino gates.

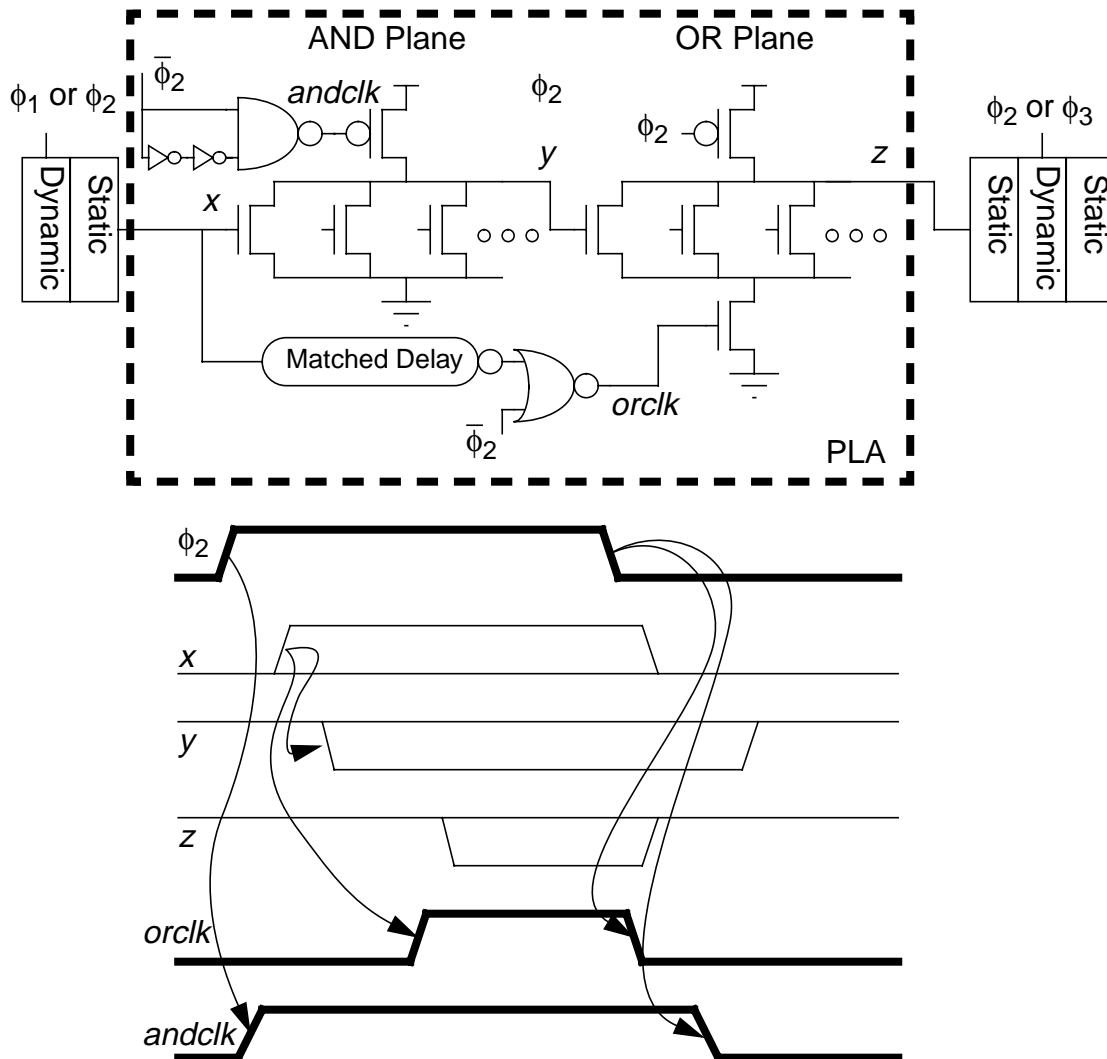
Certain very useful dynamic structures such as wide comparators and dynamic PLAs are inherently non-monotonic and are conventionally built for high performance using self-timed clocks to signal completion. The problem is that these structures are most efficiently implemented with cascaded wide dynamic gates because the delay of a dynamic NOR

structure is only a weak function of the number of inputs. Generally, dynamic gates cannot be directly cascaded. However, if the second dynamic gate waits to evaluate until the first gate has completed evaluation, the inputs to the second gate will be stable and the circuit will compute correctly. The challenge is creating a suitable delay between gates. If the delay is too long, time is wasted. If the delay is too short, the second gate may obtain the wrong result.

A common solution is to locally create a self-timed clock by sensing the completion of a model of the first dynamic gate. For example, Figure 3-16 shows a dynamic NOR-NOR PLA integrated into a skew-tolerant pipeline. The AND plane is illustrated evaluating during  $\phi_2$  and adjacent logic can evaluate in the same or nearby phases. *andclk* is nominally in phase with  $\phi_2$ , but has a delayed falling edge to avoid a precharge race with the OR plane. The latest input *x* to the AND plane is used by a dummy row to produce a self-timed clock *orclk* for the OR plane that rises after AND plane output *y* has settled. Notice how the falling edge of *orclk* is not delayed so that when *y* precharges high the OR plane will not be corrupted. The output *z* of the OR plane is then indistinguishable from any other dynamic output and can be used in subsequent skew-tolerant domino logic.



Figure 3-16 Domino / PLA interface



### 3.3 Testability

As integrated circuits use ever more transistors and overlay the transistors with an increasing number of metal layers, debug and functional test become more difficult. Packaging advances such as flip-chip technology make physical access to circuit nodes harder. Hence, engineers employ design for testability methods, trading area and even some amount of performance to facilitate test. The most important testability technique is scan, in which memory elements are made externally observable and controllable through a scan chain [58]. Scan generally involves modifying flip-flops or latches to add scan signals.

Because scan provides no direct value to most customers, it should impact a design as little as possible. A good scan technique has:

- minimal performance impact
- minimal area increase
- minimal design time increase
- no timing-critical scan signals
- little or no clock gating
- minimal tester time

The area criteria implies that scan should add little extra cell area and also few extra wires. The timing-critical scan signal criteria is important because scan should not introduce critical paths or require analysis and timing verification of the scan logic. Clock gating is costly because it increases clock skew and may increase the setup on already critical clock enable signals such as global stall requests.

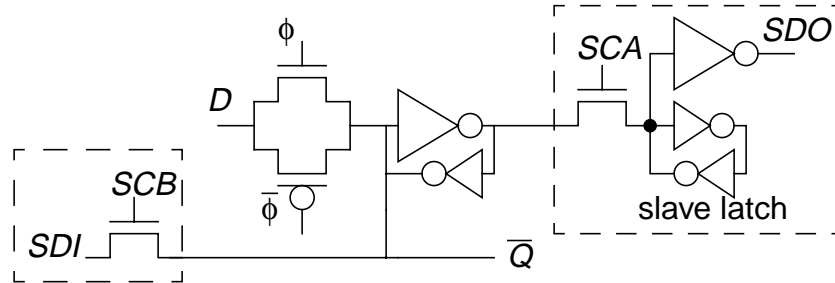
We will assume that scan is performed by stopping the global clock low (i.e.  $\phi_1$  and  $\phi_2$  low and  $\phi_3$  and  $\phi_4$  high), then toggling scan control signals to read out the current contents of memory elements and write in new contents. We will first review scan of transparent and pulsed latches, then extend the method to scan skew-tolerant domino gates in a very similar fashion.

### 3.3.1 Static Logic

Systems built from transparent latches or pulsed latches can be made more testable by adding scan circuitry to every cycle of logic. Figure 3-17 shows a scannable latch. Normal latch operation involves input  $D$ , output  $Q$ , and clock  $\phi$ . When the clock is stopped low, the latch is opaque. The circuits shown in the dashed boxes are added to the basic latch for scan. The contents of latch can be scanned out to  $SDO$  (scan data out) and loaded from  $SDI$  (scan data in) by toggling the scan clocks  $SCA$  and  $SCB$ . While it is possible to use a single scan clock, the two-phase non-overlapping scan clocks shown are more robust and simplify clock routing. The small inverters represent weak feedback devices; they must be ratioed to allow proper writing of the cell. Note that this means the gate driving the data input  $D$  must be strong enough to overpower the feedback inverter. While a tristate feed-

back gate may be used instead, it must still be weak enough to be overpowered by *SDI* during scan.

**Figure 3-17 Scannable latch**



We assume that scan takes place while the clock is stopped low. Therefore, transparent latch systems make the first half-cycle latch scannable and pulsed latch systems make the pulsed latch scannable. The procedure for scan is:

- 1.1 Stop *gclk* low
- 1.2 Toggle *SCA* and *SCB* to march data through the scan chain
- 1.3 Restart *gclk*

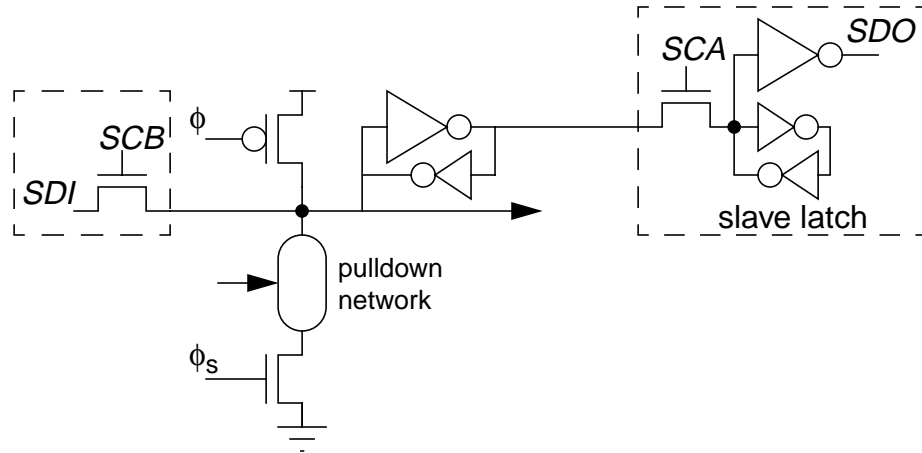
**Guideline 15:** Make all pulsed latches and  $\phi_1$  transparent latches scannable.

### 3.3.2 Domino Logic

Because skew-tolerant domino does not use latches, some other method must be used to observe and control one node in each cycle. Controlling a node requires cutting off the normal driver of the node and activating an access transistor. For example, latches are controlled during scan by being made opaque, then activating the *SCB* access transistor in Figure 3-17. A dynamic gate with a full keeper can be controlled in an analogous way by turning off both the evaluation and precharge transistors and turning on an access transistor, as shown in Figure 3-18. Notice that separate evaluation and precharge signals are necessary to turn off both devices so a gated clock  $\phi_s$  is introduced. A slave latch connected to the full keeper provides observability without loading the critical path, just as it does on a static latch. Note that this is a ratioed circuit and the usual care must be taken that feedback inverters are sufficiently weak in all process corners to be overpowered.

Also, note that only a PMOS keeper is required on the dynamic output node if *SCA* and *SCB* are toggled quickly enough that leakage is not a problem.

**Figure 3-18 Scannable dynamic gate**



Which dynamic gate in a cycle should be scannable? The gate should be chosen so that during scan, the subsequent domino gate is precharging so that glitches will not contaminate later circuits. The gate should also be chosen so that when normal operation resumes, the output will hold the value loaded by scan until it is consumed.

Let us assume that scan is done while the global clock is stopped low, thus with the  $\phi_1$  and  $\phi_2$  domino gates in the first half-cycle precharging and the  $\phi_3$  and  $\phi_4$  gates in the second half-cycle evaluating. Then a convenient choice is to scan the last  $\phi_4$  domino gate in the cycle. This means that the last  $\phi_4$  domino gate must include a full keeper. Scan is done with the following procedure:

- 2.1 Stop gclk low
- 2.2 Stop  $\phi_s$  low
- 2.3 Toggle *SCA* and *SCB* to march data through the scan chain
- 2.4 Restart gclk
- 2.5 Release  $\phi_s$  once scannable gate begins precharge

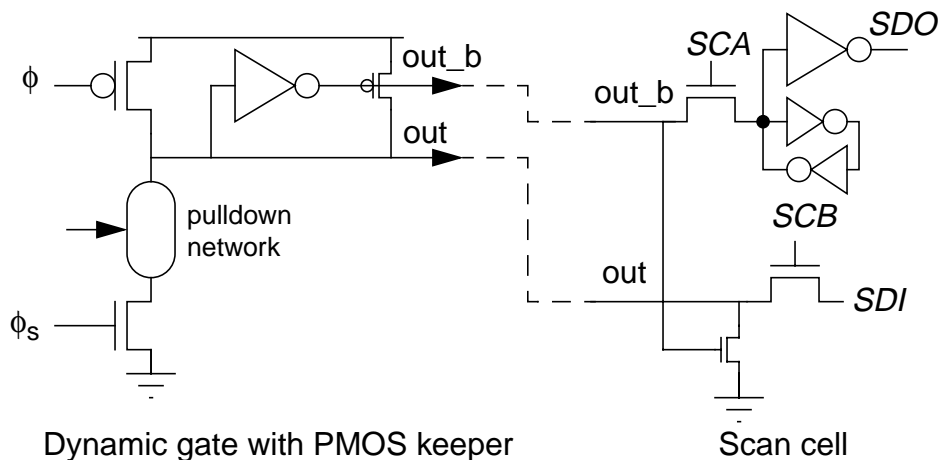
When gclk is stopped, the scannable gate will have evaluated to produce a result. Stopping  $\phi_s$  low will turn off the evaluation transistor to the scannable gate, leaving the output on the dynamic node held only by the full keeper. Toggling *SCA* and *SCB* will advance the

result down the scan chain and load a new value into the dynamic gate. When  $\phi_{clk}$  restarts, it rises, allowing the gates in the first half-cycle to evaluate with the data stored on the scan node. Once the scannable gate begins precharging,  $\phi_s$  can be released because the gate no longer needs to be cut off from its inputs.

Unfortunately, this scheme requires releasing  $\phi_s$  in a fraction of a clock cycle. It would be undesirable to do this with a global control signal because it is difficult to get a global signal to all parts of the chip in a tightly controlled amount of time. It is better to use a small amount of logic in the local clock generator to automatically perform steps (2.2) and (2.5). We will examine such a clock generator supporting four-phase skew-tolerant domino with clock enabling and scan in Section 4.2.3.

A potential difficulty with scanning dynamic gates is that it could double the size of a dynamic cell library if both scannable and normal dynamic gates are provided. A solution to this problem is to provide a special scan cell that “bolts on” to an ordinary dynamic gate. The scan cell adds a full keeper and scan circuitry to the ordinary gate’s PMOS keeper, as shown in Figure 3-19. In ordinary usage, the two clock inputs of the dynamic gate are shorted to  $\phi_4$ , while in a scannable gate  $\phi_4$  and  $\phi_{4s}$  are connected.

**Figure 3-19** Dynamic gate and scan cell



**Guideline 16:** Make the last domino gate of each cycle scannable with bolt-on scan logic.

A cycle may combine static and domino logic. As long as all first half-cycle latches and the last domino gate in each second half-cycle are scanned, the cycle is fully testable.

Static and domino scan nodes are compatible and may be mixed in the same scan chain. Note that pulsed domino latches are treated as first half-cycle domino gates and are not scanned.

### **3.4 Summary**

This chapter has described a method for designing systems with transparent and pulsed latches and skew-tolerant domino. It uses a single globally distributed clock from which four local overlapping clock phases are derived. The methodology supports stopping the clock low for power savings and testability and describes a low-overhead scan technique compatible with both domino and static circuits. Timing types are used to verify proper connectivity among the clocked elements.

The methodology hides sequencing overhead everywhere except at the interface between static and domino logic. At the interface of domino to static logic, a latch is necessary to hold the result, adding a latch propagation delay to the critical path. More importantly, at the interface of static to domino logic, clock skew must be budgeted so that inputs settle before the earliest the evaluation clock might rise, yet the domino gate may not begin evaluation until the latest time the clock might rise. This overhead makes it expensive to switch between static and domino logic. Designers who need domino logic to meet cycle time targets should therefore consider implementing their entire path in domino. Because single-rail domino cannot implement nonmonotonic functions, dual-rail domino is usually necessary. Therefore, we should expect to see more critical paths built entirely from dual-rail domino as sequencing overhead becomes a greater portion of the cycle time.

## Chapter 4 Clocking

Clocking is a key challenge for high speed circuit designers. Circuit designers specify a modest number of *logical* clocks which ideally arrive at all points on the chip at the same time. For example, flip-flop-based systems use a single logical clock, while skew-tolerant domino might use four logical clocks. Unfortunately, mismatched clock network paths and processing and environmental variations make it impossible for all clocks to arrive at exactly the same time, so the designer must settle for actually receiving a multitude of skewed *physical* clocks. To achieve low clock skew, it is important to carefully match all of the paths in the clock network and to minimize the delay through the network because random variations lead to skews which are a fraction of the mismatched delays. Previously, we have focused on hiding skew where possible and budgeting where necessary. We must be careful, however, that our skew-tolerant circuit techniques do not complicate the clock network so much that they introduce more skew than they tolerate.

This chapter begins by defining the nominal waveforms of physical clocks. The interarrival time of two clock edges is the delay between the edges. Clock skew is the absolute difference between the nominal and actual interarrival times of a pair of physical clock edges. Clock skew displays both spacial and temporal locality; by considering such locality, we only must budget or hide the actual skew experienced between launching and receiving clocks of a particular path. Skew budgets for min-delay checks must be more conservative than for max-delay because of the dire consequences of hold time violations; fortunately, min-delay races are set by pairs of clocks sharing a common edge in time, so min-delay budgets need not include jitter or duty cycle variation. Since it may be impractical to tabulate the skew between every pair of physical clocks on a chip, we lump clocks into *domains* for simplified, though conservative analysis.

Having defined clock skew, we turn to skew-tolerant domino clock generation schemes for two, four, and more phases. We see that the clock generators introduce additional delays into the clock network and hence increase clock skew. Nevertheless, the extra clock skew is small compared to the skew tolerance, so such generators are acceptable. Four-phase skew-tolerant domino proves to be a reasonable design point combining good skew toler-

ance and simple clock generation, so we present a complete four-phase clock generation network supporting clock enabling and scan.

## 4.1 Clock Waveforms

We have relied upon an intuitive definition of clock skew while discussing skew-tolerant circuit techniques. In this chapter, we will develop a more precise definition of clock skew which takes advantage of the myriad correlations between physical clocks. Physical clocks may have certain systematic timing offsets caused by different numbers of clock buffers, clock gating, etc. We can plan for these systematic offsets by placing more logic in some phases and less in others than we would have if all physical clocks exactly matched the logical clocks; the nominal offsets between physical clocks do not appear in our skew budget. The only term which must be budgeted as skew is the variability, the difference between nominal and actual interarrival times of physical clocks.

### 4.1.1 Physical Clock Definitions

A system has a small number of logical clocks. For example, flip-flops or pulsed latches use a single logical clock, transparent latches use two logical clocks, and skew-tolerant domino uses  $N$ , often four, logical clocks. A logical clock arrives at all parts of the chip at exactly the same time. Of course logical clocks do not exist, but they are a useful fiction to simplify design.

Conceptually, we envision a unique physical clock for each latch, but one can quickly group physical clocks that represent the same logical clock and have very small skew relative to each other into one clock to reduce the number of physical clocks. For example, a single physical clock might serve a bank of 64 latches in a datapath. By defining waveforms for physical clocks rather than logical clocks, we set ourselves up to budget only the skew actually possible between a pair of physical clocks rather than the worst case skew experienced across the chip.

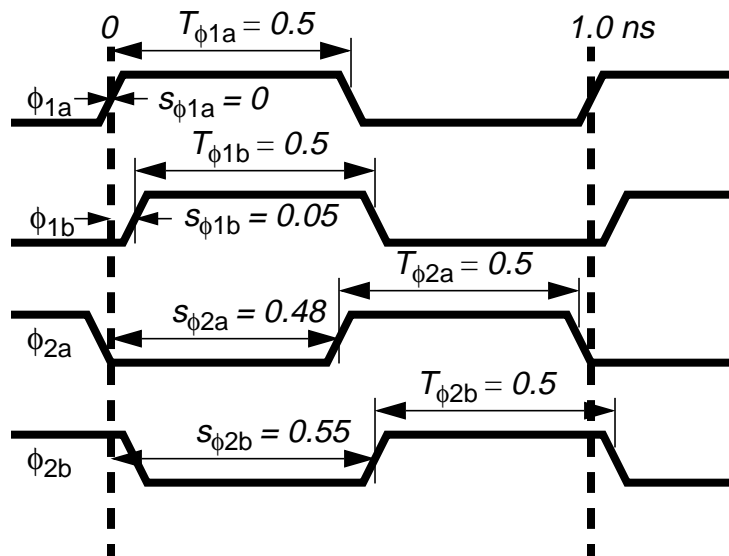
We define the set of physical clocks to be  $C=\{\phi_1, \phi_2, \dots, \phi_k\}$ . We assume that all clocks have the same cycle time  $T_c$ <sup>1</sup>. Variables describing the clock waveforms are defined below



and illustrated in Figure 4-1 for a two-phase system with four 50% duty cycle physical clocks.

- $T_c$ : the clock cycle time, or period
- $T_{\phi_i}$ : the duration for which  $\phi_i$  is high
- $s_{\phi_i}$ : the start time, relative to the beginning of the common clock cycle, of  $\phi_i$  being high
- $S_{\phi_i, \phi_j}$ : a phase shift operator describing the difference in start time from  $\phi_i$  to the next occurrence of  $\phi_j$ .  $S_{\phi_i, \phi_j} \equiv s_{\phi_i} - (s_{\phi_j} + WT_c)$ , where  $W$  is a wraparound variable indicating the number of cycle crossings between the sending and receiving clocks.  $W$  is 0 or 1 except in systems with multi-cycle paths. Note that  $S_{\phi_i, \phi_i} = -T_c$  because it is the shift between consecutive rising edges of clock phase  $\phi_i$ .

**Figure 4-1 Two-phase clock waveforms**



Note that Figure 4-1 labels the clocks  $C = \{\phi_{1a}, \phi_{1b}, \phi_{2a}, \phi_{2b}\}$  rather than  $C = \{\phi_1, \phi_2, \phi_3, \phi_4\}$  to emphasize that the four physical clocks correspond to only two logical clocks. The former labeling will be used for examples, while the later notation is more convenient for expressing constraints in timing analysis in Chapter 5. The phase shifts between these clocks seen at consecutive transparent latches are shown in Table 4-1. Notice that the sys-

1. In extremely fast systems, clocks may operate at very high frequency in local areas, but at lower frequency when communicating between remote units. We presently see this in systems where the CPU operates at high speed but the motherboard operates at a fraction of the frequency. This clocking analysis may be generalized to clocks with different cycle times.

tematic offsets between clocks appear as different phase shifts rather than clock skew. It is possible to design around such systematic offsets, intentionally placing more logic in one half-cycle than another. Indeed, designers sometimes intentionally delay clocks to extend critical cycles of logic in flip-flop based systems where time borrowing is not possible. We save the term skew for uncertainties in the clock arrival times.

**Table 4-1 Phase shift between clocks of Figure 4-1**

$S_{\phi_i\phi_j}$		Receiving clock $\phi_i$			
		$\phi_{1a}$	$\phi_{1b}$	$\phi_{2a}$	$\phi_{2b}$
Launching clock $\phi_j$	$\phi_{1a}$			-0.48	-0.55
	$\phi_{1b}$			-0.43	-0.50
	$\phi_{2a}$	-0.52	-0.57		
	$\phi_{2b}$	-0.45	-0.50		

#### 4.1.2 Clock Skew

If the actual delay between two phases  $\phi_i$  and  $\phi_j$  equalled the nominal delay  $S_{\phi_i\phi_j}$ , the phases would have zero skew. Of course, delays are seldom nominal, so we must define clock skew. There are many sources of clock skew. When a single physical clock serves multiple clocked elements, delay between the clock arrival at the various elements appears as skew. Cross-die variations in processing, temperature, and voltage also lead to skew. Electromagnetic coupling and load capacitance variations [16] lead to further skew in a data-dependent fashion. If all clock paths sped up or slowed down uniformly, the interarrival times would be unaffected and no skew would be introduced. Therefore, we are only concerned with differences between delays in the clock network.

In previous chapters, we have used a single skew budget  $t_{\text{skew}}$  which is the worst case skew across the chip, i.e., largest absolute value of the difference between the nominal and actual interarrival times of a pair of clocks anywhere on the chip. When  $t_{\text{skew}}$  can be held to about 10% of the cycle time, it is simple and not overly limiting to budget this worst case skew everywhere. As skews are increasing relative to cycle time, we would prefer to only budget the actual skew encountered on a path, so we define skews between specific pairs of physical clocks. For example,  $t_{\text{skew}}^{\phi_i\phi_j}$  is the skew between  $\phi_i$  and  $\phi_j$ , the absolute

value of the difference between the nominal and actual interarrival times of these edges measured at any pair of elements receiving these clocks. For a given pair of clocks, certain transitions may have different skew than others. Therefore, we also define skews between particular edges of pairs of physical clocks. For example,  $t_{\text{skew}}^{\phi_i(r), \phi_j(f)}$  is the skew between the rising edge of  $\phi_i$  and the falling edge of  $\phi_j$ .  $t_{\text{skew}}^{\phi_i, \phi_j}$  is the maximum of the skews between any edges of the clocks.

Notice that skew is a positive difference between the actual and nominal interarrival times, rather than being plus or minus from a reference point. When using this information in a design, we assume the worst: for maximum delay (setup time) checks, that the receiving clock is skewed early relative to the launching clock; and for minimum delay (hold time) checks, that the receiving clock is skewed late relative to the launching clock. If skews are asymmetric around the reference point, we may define separate values of skew for min and max delay analysis.

Also, note that the cycle count between edges is important in defining skew. For example, the skew between the rising edge of a clock and the same rising edge a cycle later is called the cycle-to-cycle jitter. The skew between the rising edge and the same rising edge many cycles later may be larger and is called the peak jitter. Generally, we will only consider edges separated by at most one cycle when defining clock skew because including peak jitter is overly pessimistic. This occasionally leads to slightly optimistic results in latch-based paths in which a signal is launched on the rising edge of one latch clock and passes through more than one cycle of transparent latches before being sampled. The jitter between the launching and sampling clocks is greater than cycle-to-cycle jitter in such a case, but the error is unlikely to be significant.

Since clock skew depends on mismatches between nominally equal delays through the clock network, skews budgets tend to be proportional to the absolute delay through the network. Skews between clocks which share a common portion of the clock network are smaller than skews between widely separated clocks because the former clocks experience no environmental or processing mismatches through the common section. However, even two latches sharing a single physical clock experience cycle-to-cycle skew from jitter and duty-cycle variation, which depend on the total delay through the clock network.

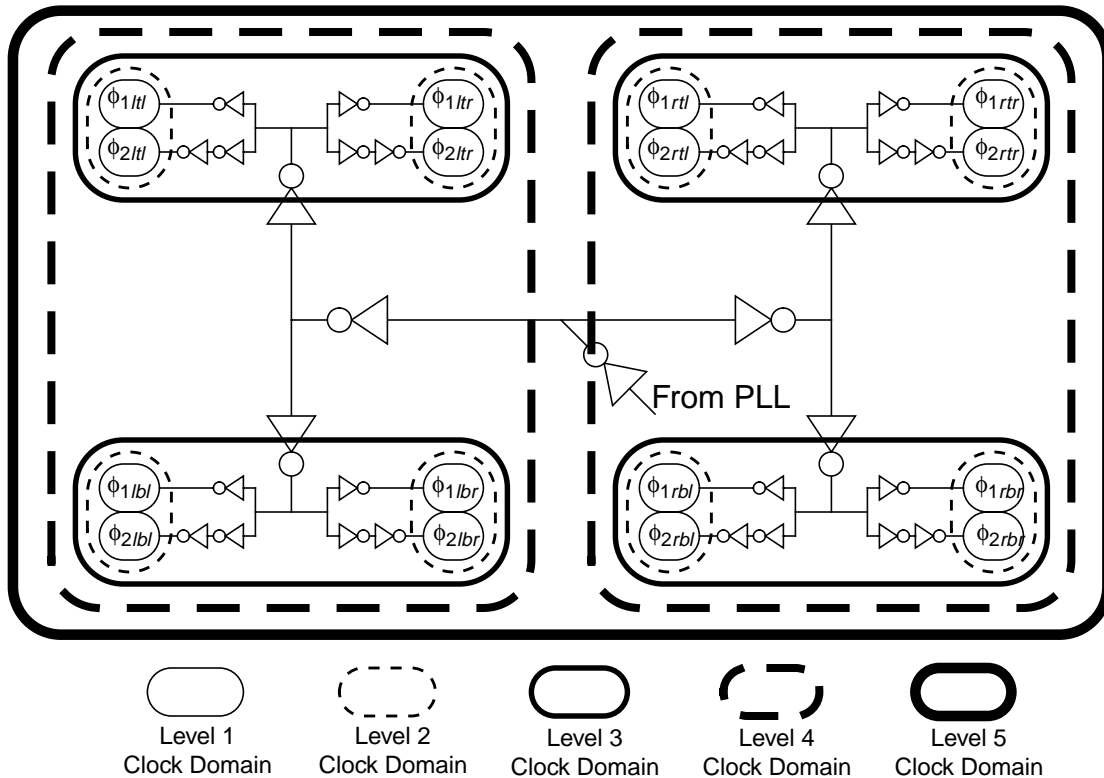
The designer may use different skew budgets for minimum and maximum delay analysis purposes. Circuits with hold time problems will not operate correctly at any clock frequency, so designers must be very conservative. Fortunately, min-delay races occur between clocks in a single cycle, so jitter and duty cycle variation are not part of the skew budget. Circuits with setup time problems operate properly at reduced frequency. Therefore, the designer may budget an expected skew, rather than a worst case skew, for max-delay analysis, just as designers may target TT processing instead of SS processing. This avoids overdesign while achieving acceptable yield at the target frequency. Unfortunately, calculating the expected skew requires extensive statistical knowledge of the components of clock skew and their correlations.

On account of larger chips, greater clock loads, and wire delays which are not scaling as well as gate delays, it is very difficult to hold clock skew across the die constant in terms of gate delays. Indeed, Horowitz predicted that keeping global skews under 200 ps is hard [34]. Moreover, as cycle times measured in gate delays continue to shrink, even if clock skew were held constant in gate delays, it would tend to become a larger fraction of the cycle time. Therefore, it will be very important to take advantage of skew-tolerant circuit techniques and to exploit locality of clock skew when building fast systems.

### **4.1.3 Clock Domains**

While one may conceptually specify an array of clock skews between each pair of physical clocks in a large system, such a table may be huge and mostly redundant. In practice, designers usually lump clocks into a hierarchy of clock domains. For example, we have intuitively discussed local and global clock domains; pairs of clocks in a particular local domain experience local skew which is smaller than the global skew seen by clocks in different domains. We can extend this notion to finer granularity by defining a skew hierarchy with more levels of clock domains, as shown in Figure 4-2 for a system based on an H-tree. In Section 5.9.4 we will formalize the definitions of skew hierarchies and clock domains for the purpose of timing analysis.

Figure 4-2 H-Tree clock distribution network illustrating multiple levels of clock domains



In Figure 4-2, level 1 clock domains contain a single physical clock. Therefore, two elements in the level 1 domain will only seek skew from RC delays along the clock wire and from jitter of the physical clock. Level 2 clock domains contain a clock and its complement and see additional skew caused by differences between the nominal and actual clock generator delays. Remember that systematic delay differences which are predictable at design time can be captured in the physical clock waveforms; only delay differences from process variations or environmental differences between the clock generators appear as skew. Higher levels clock domains see progressively more skew as delay variations in the global clock distribution network appear as skew.

## 4.2 Skew-Tolerant Domino Clock Generation

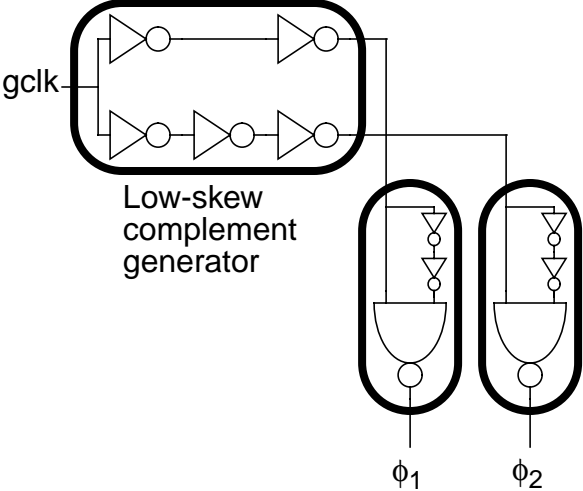
In most high frequency systems, a single clock  $gclk$  is distributed globally using a modified H-tree or grid to minimize skew. Skew tolerant domino can use this same clock distribution scheme with a single global clock. Within each unit or functional block, local clock generators produce the multiple phases required for latches and skew-tolerant domino.

These local generators inevitably increase the delay of the distribution network and hence increase clock skew. This section describes several local clock generation schemes and analyzes the skews introduced by the generators. The simplest schemes involve simple delay lines and are adequate for many applications. Lower skews can be achieved using feedback systems with delays that track with process and environmental changes. We conclude with a full-featured local clock generator supporting transparent latches and four-phase skew-tolerant domino, clock enabling, and scan.

**4.2.1 Delay Line Clock Generators**

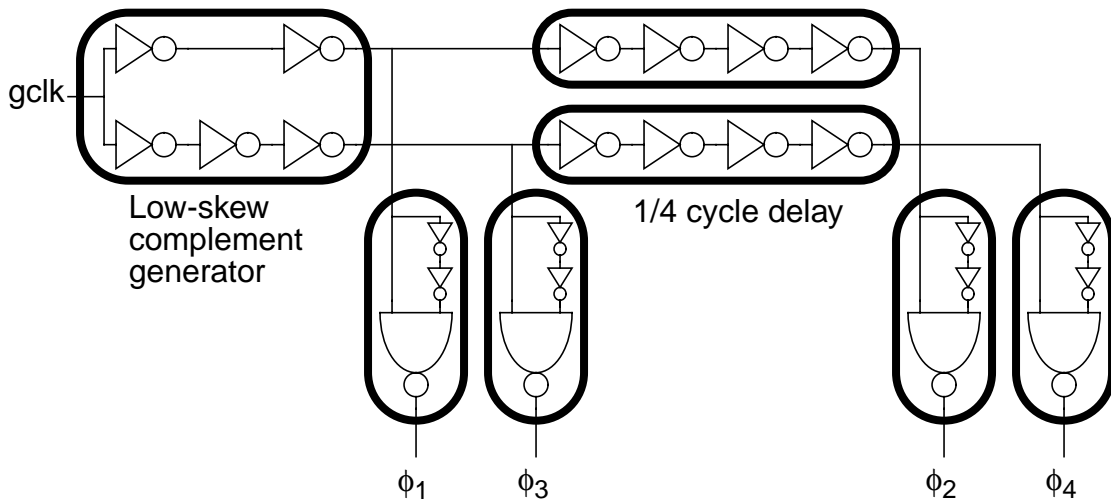
Overlapping clocks for skew-tolerant domino can be easily generated by delaying one or both edges of the global clock with chains of inverters. Figure 4-3 shows a simple two-phase skew-tolerant domino local generator, while Figure 4-4 extends the design to support four phases. The two-phase design uses a low-skew complement generator to produce complementary signals from the global clock. For example, Shoji showed how to match the delay of 2 and 3 inverters to match independently of relative PMOS/NMOS mobilities [68]. The falling clock edges are stretched with clock choppers to produce waveforms with greater than 50% duty cycle. Using a fanout of 3-4 on each inverter delay element sets reasonable delay and minimizes the area of the clock buffer while preserving sharp clock edges.

**Figure 4-3 Two-phase clock generator**



The four-phase design is very similar, but uses an additional chain of inverters to produce a nominal quarter cycle delay. At first it would seem such a clock generator would suffer from terrible clock skews because between best and worst case processing and environment, its delay may vary by a factor of 2! Fortunately, we are concerned not with the absolute delay of the inverter chain, but rather with its tracking relative to critical paths on the chip. In the slow corner, the delay chain will have a greater delay, but the critical paths will also be slower and the operating frequency will be correspondingly reduced. Hence, to first order the delay chain tracks the speed of the logic on the chip; we are now concerned about skew introduced by second order mismatches.

**Figure 4-4 Four-phase clock generator**



#### 4.2.1.1 Local Generator Induced Clock Skew

Since the local generators are not replicas of the circuits they are tracking, and indeed are static gates tracking the speed of dynamic paths, their relative delays may vary over process corners as well as due to local variation in voltage and temperature and local process variations. Simulation finds that when most of the chip is operating under nominal processing and worst case environment but a local clock generator sees a temperature 30 °C lower and supply voltage 300 mV higher, the local generator will run 13% faster than nominal (6% from temperature, 7% from voltage). The relative delay of simple domino gates with respect to fanout-of- four inverters varied up to about 6% across process corners. Finally, process tilt, i.e., fluctuation in  $L_e$ ,  $t_{ox}$ , etc., across the die, may speed the local

clock generator more than nearby logic. Little data is available on process tilt, but if we guess it causes similar 13% variation, we conclude that nearly a third of the total local clock generator delay appears as clock skew.

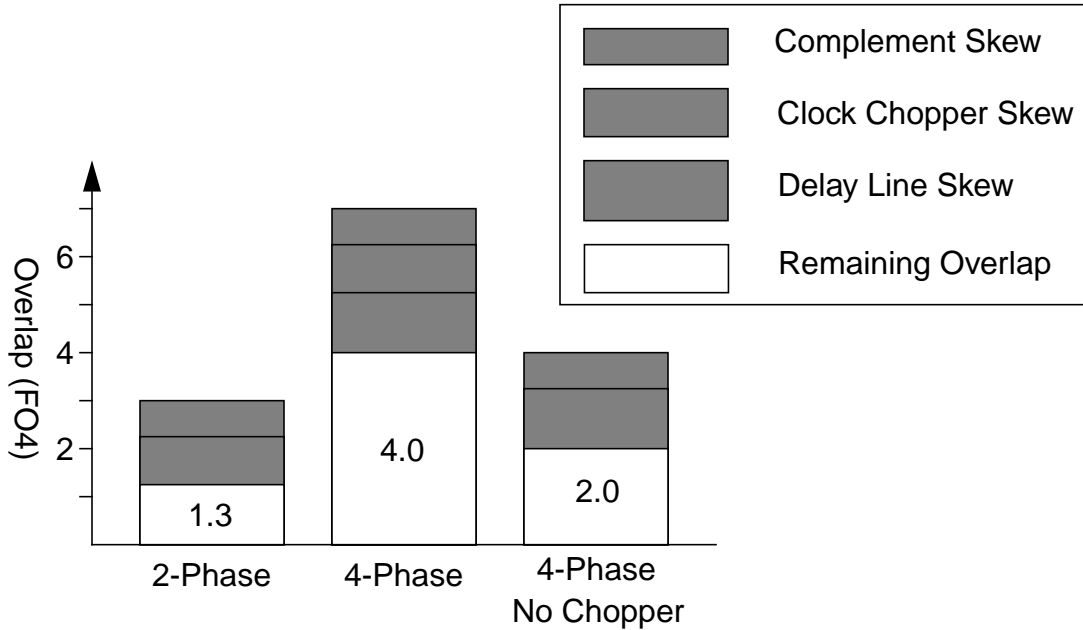
Four-phase clock generators have a quarter cycle more delay than two-phase generators, so are subject to more skew. However, they can also tolerate a quarter cycle more skew than their two-phase counterparts, which is significantly more than the extra skew of the generators. For example, consider two and four-phase systems like those described in Section 2.1.2 with cycle times of 16 FO4 delays and precharge times of 4 FO4 delays. If the local skew is 1 FO4 delay, the nominal overlap between phases is 3 FO4 delays for the two-phase system and 7 FO4 delays for the four-phase system. These overlaps can be used to tolerate clock skew and allow time borrowing. From the overlap we must subtract the skews introduced by the local clock generators. If the complement generator, clock chopper, and quarter cycle delay lines have nominal delays of 2, 3, and 4 FO4 delays, respectively, we must budget 32% of these delays as additional skew. Figure 4-5 compares the remaining overlaps of each system, showing that although the four-phase system pays a larger skew penalty, the remaining overlap is proportionally much greater than that of the two-phase system. The four-phase clock generator can be simplified to use 50% duty cycle clocks as shown in Figure 4-6, eliminating the clock choppers at the expense of smaller overlaps. The four-phase system with 50% duty cycle waveforms still provides more overlap than the two-phase system and avoids the min-delay problems associated with overlapping two-phase clocks. Therefore, it is a reasonable design choice, especially considering the drawbacks of clock choppers which we will shortly note. In Section 4.2.3 we will look at a complete four-phase clock generator including clock gating and scan capability.

The four-phase clock generator with clock choppers appears to offer substantial benefits over the design with no choppers. A closer look reveals several liabilities in the design with choppers. Variations in the clock chopper delay cause duty cycle errors which cut into the precharge time, necessitating a lower smaller overlaps than our first-order analysis predicted. The extended duty cycle also increases the susceptibility to min-delay problems, especially when coupled with the large skews introduced by the clock generator. Finally, the designer may still desire to use 50% duty cycle clocks for transparent latches.

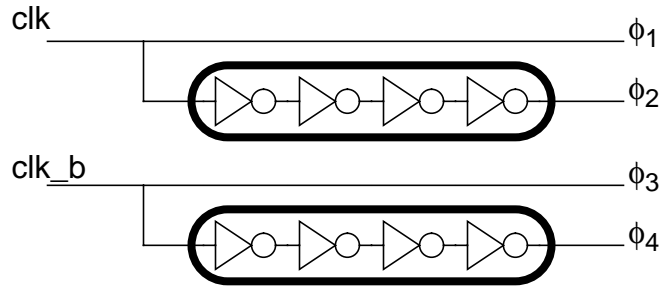


Therefore, the chopperless four-phase scheme is preferred when it offers enough overlap to handle the expected skews and time borrowing requirements.

**Figure 4-5** Overlap between phases for two- and four-phase systems after clock generator skews



**Figure 4-6** Simplified four-phase clock generator



In addition to having adequate overlap for time borrowing and hiding clock skew, domino clocks must have sufficiently long precharge times in all process corners. The local clock generators are subject to duty cycle variation, which might change the amount of time available for precharging. Fortunately, if we design the system to have adequate precharge time in the worst case environment under TT processing, environmental changes will only lead to more precharge time and faster precharge operation. In the SS corner, the clock must be slowed to accommodate precharge, but it is slowed anyway because of the longer critical paths.

#### 4.2.1.2 N-Phase Local Clock Generators

Another popular skew-tolerant domino clocking scheme is to provide one phase for each gate. This offers maximum skew-tolerance and more precharge time, as discussed in Section 2.1.4, at the expense of generating and distributing more clocks and roughly matching gate and clock delays. Figures 4-7 and 4-8 show such clock generation schemes. Figure 4-7 uses both edges of the clock and is the simplest scheme. The exact delay of the buffers is not particularly important so long as the clocks arrive at each gate before the data. Figure 4-8 delays a single clock edge, as used on the IBM guTS experimental GHz processor [53]. To make sure the last phase overlaps the first phase of the next cycle, a pulse stretcher, such as an SR latch, must be used. The stretcher is especially important at low frequency; the first guTS test chip accidentally omitted a stretcher, making the chip only run at a narrow range of frequencies. Another disadvantage of delaying a single edge is that the precharge time of the last phase becomes independent of clock frequency, creating another timing constraint which cannot be fixed by slowing the clock. Finally, the longer delays of the single-edge design lead to greater clock skew. Therefore, the design delaying both edges is simpler and more robust.

Figure 4-7 N-phase clock generator delaying both edges

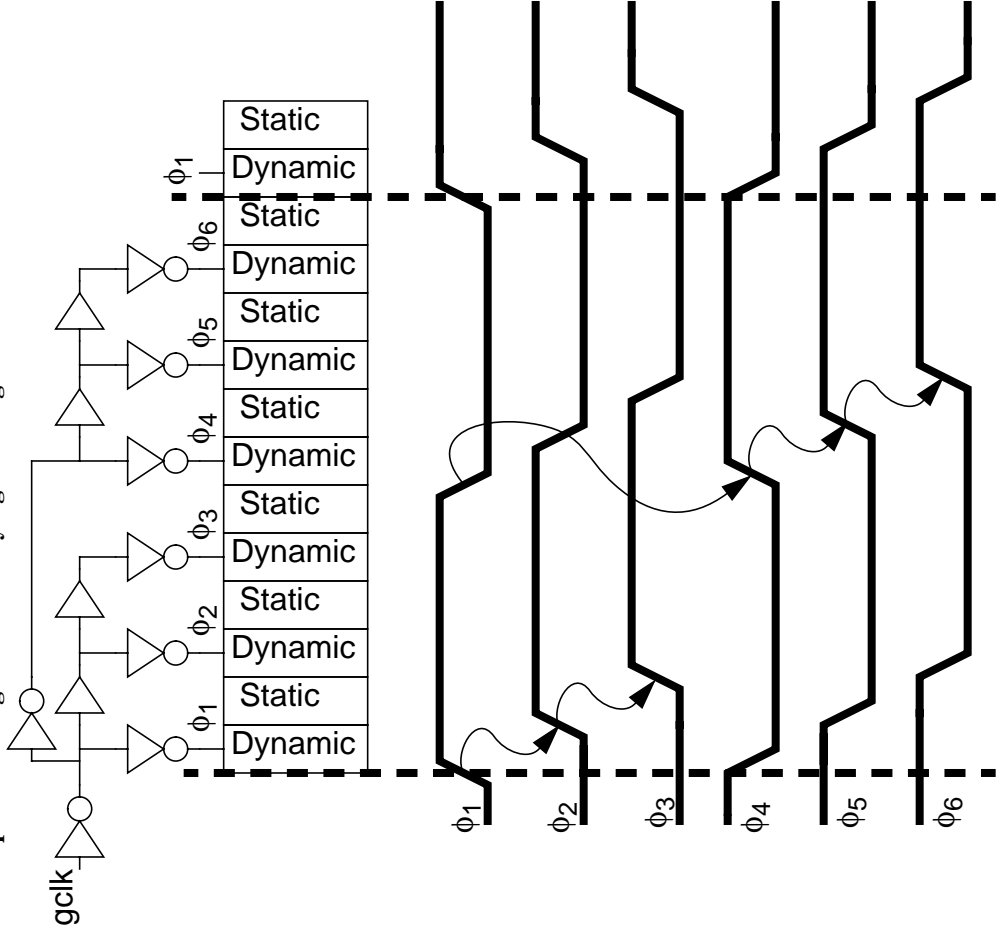
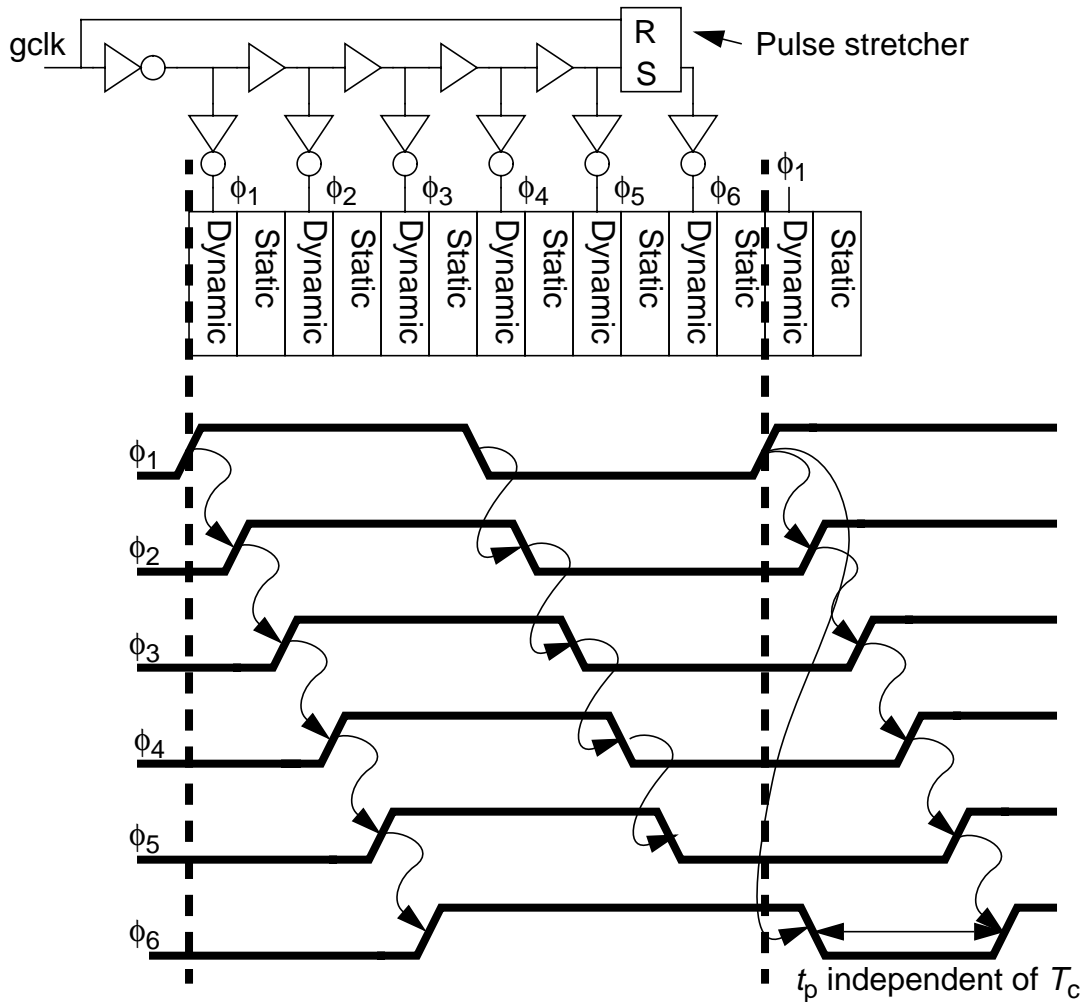


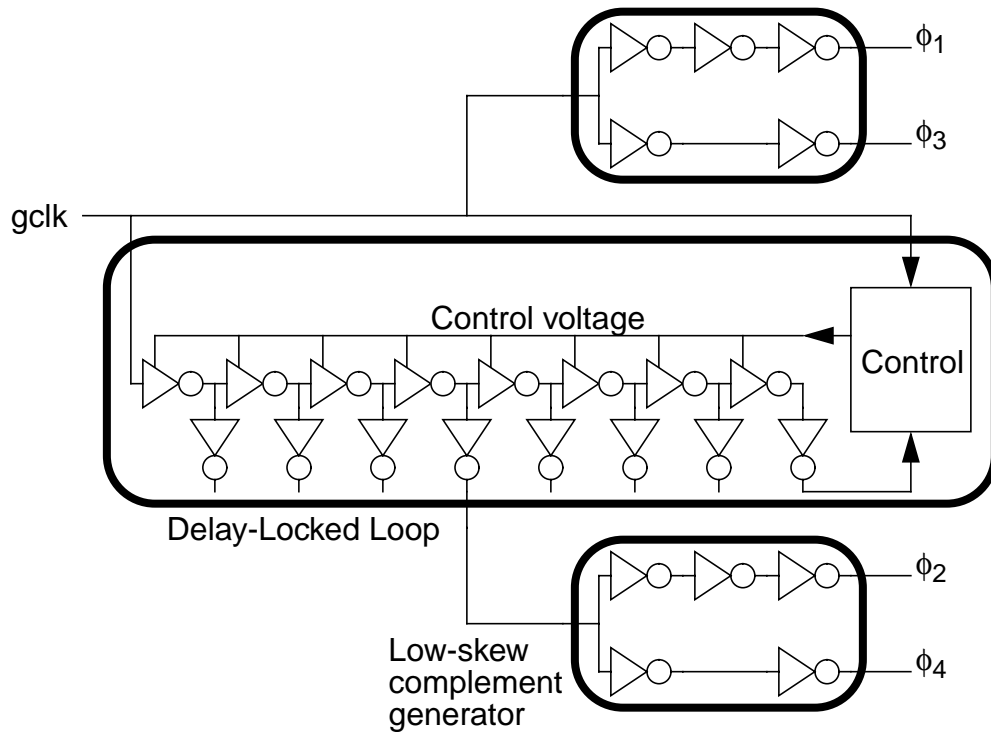
Figure 4-8 N-phase clock generator delaying a single edge



#### 4.2.2 Feedback Clock Generators

To reduce the skew and duty cycle uncertainty of the local clock generators, we may also use local delay-locked loops [12] to produce skew-tolerant domino clocks. Such a system is shown in Figure 4-9. The local loop receives the global clock and delays it by exactly  $1/4$  cycle by adjusting a delay line to have a  $1/2$  cycle delay and tapping off the middle of the delay line. The feedback controller compensates for process and low-frequency environmental variations and even for a modest range of different clock frequencies. The art of DLL design is beyond the scope of this work; the illustration should be considered to be conceptual only.

**Figure 4-9 Four-phase clock generator using feedback control**



Unfortunately, the DLL itself introduces skew. In particular, power supply noise in the delay line at frequencies above the controller bandwidth appear as jitter on  $\phi_2$  and  $\phi_4$ . In a system without feedback, power supply variation from  $V+\Delta V$  to  $V-\Delta V$  causes delay variation from  $t+\Delta t$  to  $t-\Delta t$ . In the DLL, a supply step from  $V+\Delta V$  to  $V-\Delta V$  after the system had initially stabilized at  $V+\Delta V$  causes delay variation from  $t$  to  $t-2\Delta t$ . Similarly, a rising step causes delay variation from  $t$  to  $t+2\Delta t$ . Therefore, the DLL has twice the voltage sensitivity of the system without feedback. PLLs are even more sensitive to voltage noise because they accumulate jitter over multiple cycles; therefore, they are not a good choice for local clock generators.

Fortunately, the local high frequency voltage noise causing jitter is a fraction of the total voltage noise. If we assume the high frequency noise in each DLL is half as large as the total voltage noise, the jitter of the DLL will equal the skew introduced by voltage errors on a regular delay line system. Using the numbers from the example in Section 4.2.1.1, this corresponds to 7% of the quarter cycle delay to the line tap. The local clock generator also is subject to variations in the complement generator. If the DLL is designed to achieve negligible static phase offset, the skew improvement of the feedback system over the delay

line system is predicted to be the difference in delay sensitivity, 32%-7%, times the quarter cycle delay, or about 6% of the cycle time. This comes at the expense of building a small DLL in every local clock generator. The DLL may use an improved delay element with reduced supply sensitivity, but the same delay elements may be used in delay lines. The designer must weigh the potential skew improvement of DLL-based clock generators against the area, power, and design complexity they introduce. In today's systems, simple delay lines are probably good enough, but in future systems with even tighter timing margins, DLLs may offer enough advantages to justify their costs.

### 4.2.3 Putting It All Together

So far we have only considered generating periodic clock waveforms. Most systems also require the ability to stop the clock and to scan data in and out of each cycle. We saw in Section 3.3.2 that scan required precise release of the scan enable signal. By building the release circuitry into the clock generator, we avoid the need to route timing-critical global scan signals. In this section we integrate such scan circuitry and clock enabling with four-phase skew-tolerant domino to illustrate a complete local clock generator.

Local clocks are often gated with an enable signal to produce a qualified clock. Qualified clocks can be used to save power by disabling inactive units, to prevent latching new data during a pipeline stall, or to build combined multiplexer-latches. Clock enable signals are often critical because they come from far away or are data-dependent. Therefore, it is useful to minimize the setup time of the clock enable before the clock edge.

Figure 4-10 illustrates a complete local clock generator for a four-phase skew-tolerant domino system. It receives  $gclk$  from global clock distribution network and an enable signal for the local logic block. It generates the four regular clock phases along with a variant of  $\phi_4$  used for scan. Different clock enables can be used for different gates or banks of gates as appropriate. Using a 2-input NAND gate in all local clock generators provides best matching between phases to minimize clock skew; the enable may be tied high on some clocks which never stop. The last domino gate in each cycle uses  $\phi_4$  for precharge and  $\phi_{4s}$  for evaluation. Two-phase static latches use  $\phi_1$  and  $\phi_3$  as  $clk$  and  $clk\_b$ . The clock generator uses delay chains to produce domino phases  $\phi_2$  and  $\phi_4$  delayed by one quarter of

the nominal clock period. Scan is built into static latches and domino gates as described in Section 3.3. Notice that when *SCA* is asserted, a SR latch forces  $\phi_{4s}$  low to disable the dynamic gate being scanned. When  $\phi_4$  falls to begin precharge, the SR latch releases  $\phi_{4s}$  to resume normal operation. Therefore, we avoid distributing any high speed global scan enable signals and can use exactly the same scan procedure as we used with static latches:

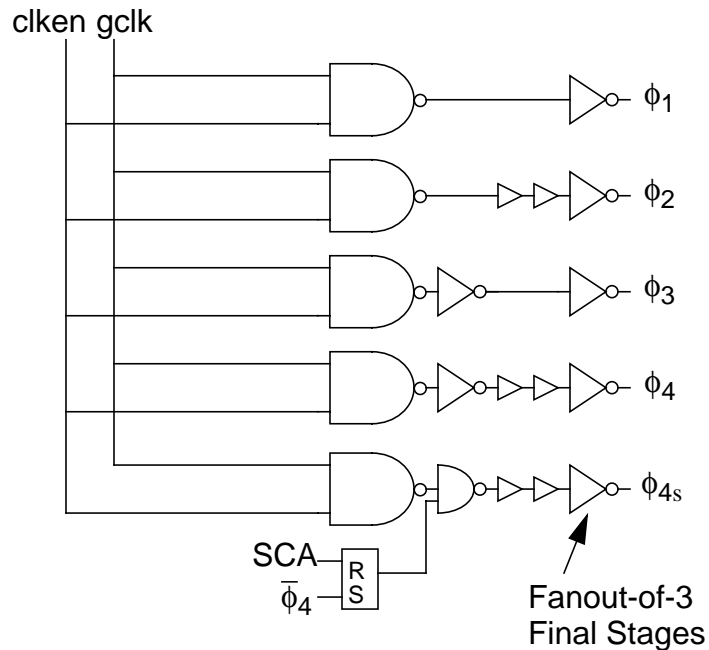
3.1 Stop gclk low

3.2 Toggle *SCA* and *SCB* to march data through the scan chain. The

first pulse of *SCA* will force  $\phi_{4s}$  low.

3.3 Restart gclk. The falling edge of  $\phi_4$  will release  $\phi_{4s}$  to track  $\phi_4$ .

**Figure 4-10 Local four-phase clock generators supporting scan and clock enabling**



### 4.3 Summary

Circuit designers wish to receive a small number of logical clocks simultaneously at all points of the die. They must instead accept a huge number of physical clocks arriving at slightly different times to different receivers. Clock skew is the difference between the nominal and actual interarrival times of two clocks. It depends on numerous sources which are difficult or impossible to accurately model, so it is typically budgeted using conservative engineering estimates. Because clock skew is an increasing problem, it is important to

understand the sources and avoid unnecessary conservatism. Skew budgets therefore may depend on the phases of and distance between the physical clocks, the particular edges of the clocks, and the number of cycles between the edges. Clocks may be grouped into a hierarchy of clock domains according to their relative skews; communication within a domain experiences less skew than communication across domains.

The designer has three tactics to deal with skew: budget, hide, and minimize. Taking advantage of the smaller amounts of skew between nearby elements is a powerful way to minimize skew, but requires improved timing analysis algorithms which are the subject of Chapter 5. Skew-tolerant circuit techniques hide clock skew, but the local clock generators necessary to produce multiple overlapping clock phases for skew-tolerant domino introduce skew of their own. Fortunately, the skews introduced are less than the tolerance provided, so skew-tolerant domino is an overall improvement.



## Chapter 5 Timing Analysis

It is impractical to build complex digital systems without CAD tools to analyze and verify the designs. Therefore, novel circuit techniques are of little use without corresponding CAD tools. Although most standard circuit tools such as simulators, layout-vs.-schematic checkers, ERC and signal integrity verifiers, etc., work equally well for skew-tolerant and non-skew-tolerant circuits, timing analyzers must be enhanced to understand and take advantage of different amounts of clock skew between different clocks.

Timing analysis addresses the question of whether a particular circuit will meet a timing specification. The analysis must check maximum delays to verify that a circuit will meet setup times at the desired frequency, and minimum delays to verify that hold times are satisfied. This chapter describes how to extend a traditional formulation of timing analysis to handle clock skew, including different budgets for skew between different regions of a system.

Our formulation of timing analysis is built on an elegant framework from Sakallah *et. al.* [62] for systems with transparent latches. Although the framework assumes zero clock skew, we can easily support systems with a single clock domain by adding worst case skew to the setup time of each latch. We then develop an exact set of constraints for analyzing systems with different amounts of skew between different elements. This exact analysis leads to an explosion of the number of timing constraints. By introducing a hierarchy of clock domains with tighter bounds on skews within smaller domains, we offer an approximate analysis which is conservative, but less pessimistic than the single skew scenario and with many fewer constraints than the exact analysis. Once we understand how to analyze latches in a system with multiple clock domains, we find analyzing flip-flops is even easier. Domino gates also fit nicely into the framework, sometimes behaving as latches and sometimes as flip-flops. Having solved the problem of max-delay, we show that min-delay is much easier because it does not involve time borrowing. We conclude by presenting algorithms for verifying the timing constraints and showing that, for a large test case, the exact analysis is only slightly more expensive than the skewless analysis.

## 5.1 Background

Early efforts in timing analysis, surveyed in [33], only considered edge-triggered flip-flops. Thus they had to analyze just the combinational logic blocks between registers because the cycle time is set by the longest combinational path between registers. Netlist-level timing analyzers, such as CRYSTAL [55] and TV [39], used switch-level RC models [61] to compute delay through the combinational blocks.

Many circuits use level-sensitive latches instead of flip-flops. Latches complicate the analysis because they allow time borrowing: a signal which reaches the latch input while the latch is transparent does not have to wait for a clock edge, but rather can immediately propagate through the latch and be used in the next phase of logic. Analysis of systems with latches was long considered a difficult problem [55] and various netlist-level timing analyzers applied heuristics for latch timing, but eventually Unger [76] developed a complete set of timing constraints for two-phase clocking with level-sensitive latches. LEAD-OUT [71], by Szymanski, checked timing equations to properly handle multiphase clocking and level-sensitive latches. Champernowne *et al.* [7] developed a set of latch to latch timing rules which allow a hierarchy of clock skews but did not permit time borrowing.

Sakallah, Mudge, and Olukotun [62] provide a very elegant formulation of the timing constraints for latch-based systems. They show that maximum delay constraints can be expressed with a system of inequalities. They then use a linear programming algorithm to minimize the cycle time and to determine an optimal clock schedule. Because the clock schedule is usually fixed and the user is interested in verifying that the circuits can operate at a target frequency, more efficient algorithms can be used to process the constraints, such as the relaxation approach suggested by Szymanski and Shenoy [73]. Moreover, many of the constraints in the formulation may be redundant, so graph-based techniques proposed by Szymanski [72] can determine the relevant constraints. Ishii *et al* [38] offer yet another efficient algorithm for verifying the cycle time of two-phase latched systems. Burks *et al.* [6] express timing analysis in terms of critical paths and support clock skew in limited ways.

## 5.2 Timing Analysis without Clock Skew

We will begin by describing a formulation of timing analysis for latch-based systems from Sakallah *et al.* [62]. The simplicity of the formulation stems from a careful choice of time variables describing data inputs and outputs of the latches. In this section, we consider only D-type latches with data in, data out, and clock terminals. Section 5.4 extends the model to include other clocked elements such as flip-flops and domino gates.

A system contains a set of physical clocks  $C=\{\phi_1, \phi_2, \dots, \phi_k\}$  with a common cycle time  $T_c$ , and a set of latches  $L=\{L_1, L_2, \dots, L_l\}$ . As defined in Section 4.1.1, the clocks have a duration  $T_{\phi_i}$ , start time  $s_{\phi_i}$ , and phase shift operator  $S_{\phi_i\phi_j}$ . For each of the  $l$  latches in the system, we define the following variables and parameters which describe which clock is used to control each latch, when data arrives and departs each latch, and the setup time and propagation delay of each latch:

- $p_i$ : the clock phase used to control latch  $i$
- $A_i$ : the arrival time, relative to the start time of  $p_i$ , of a valid data signal at the input to latch  $i$
- $D_i$ : the departure time, relative to the start time of  $p_i$ , at which the signal available at the data input of latch  $i$  starts to propagate through the latch
- $Q_i$ : the output time, relative to the start time of  $p_i$ , at which the signal at the data output of latch  $i$  starts to propagate through the succeeding stages of combinational logic
- $\Delta_{DCi}$ : the setup time for latch  $i$  required between the data input and the trailing edge of the clock input
- $\Delta_{DQi}$ : the maximum propagation delay of latch  $i$  from the data input to the data output while the clock input is high

Finally, define the propagation delays between pairs of latches:

- $\Delta_{ij}$ : the maximum propagation delay through combinational logic between latch  $i$  and latch  $j$ . If there are no combinational paths from latch  $i$  to latch  $j$ ,  $\Delta_{ij} \equiv -\infty$  effectively eliminates the path from consideration.

Using these definitions we can express constraints on the propagation of signals between latches and the setup of signals before the sampling edges of the latches. Setup time con-

straints require that a signal arrive at a latch some setup time before the sampling clock edge. Thus:

$$\forall i \in L \quad A_i + \Delta_{DCi} \leq T_{p_i} \quad (5-1)$$

The propagation constraints relate the departure, output, and arrival times of latches. Data departs a latch input when the data arrives and the latch is transparent:

$$\forall i \in L \quad D_i = \max(0, A_i) \quad (5-2)$$

The latch output becomes valid some latch propagation delay after data departs the input:

$$\forall i \in L \quad Q_i = D_i + \Delta_{DQi} \quad (5-3)$$

Finally, the arrival time at a latch is the latest of the possible arrival times from data leaving other latches and propagating through combinational logic to the latch of interest. Notice that the phase shift operator  $S$  must be added to translate between relative times of the launching and receiving latch clocks.

$$\forall i, j \in L \quad A_i = \max(Q_j + \Delta_{ji} + S_{p_j p_i}) \quad (5-4)$$

Observe that both  $D_i$  and  $Q_i$  will always be nonnegative quantities because a signal may not begin propagating through a latch until the clock has risen.  $A_i$  is unrestricted in sign because the input data may arrive before or after the latch clock. By assuming that clock pulse widths  $T_i$  are always greater than latch setup times  $\Delta_{DCi}$  and eliminating the  $Q$  and  $A$  variables, we can rewrite these constraints as L1 and L2 exclusively in terms of signal departure times and the clock parameters.

### L1. Setup Constraints:

$$\forall i \in L \quad D_i + \Delta_{DCi} \leq T_{p_i} \quad (5-5)$$

### L2. Propagation Constraints:

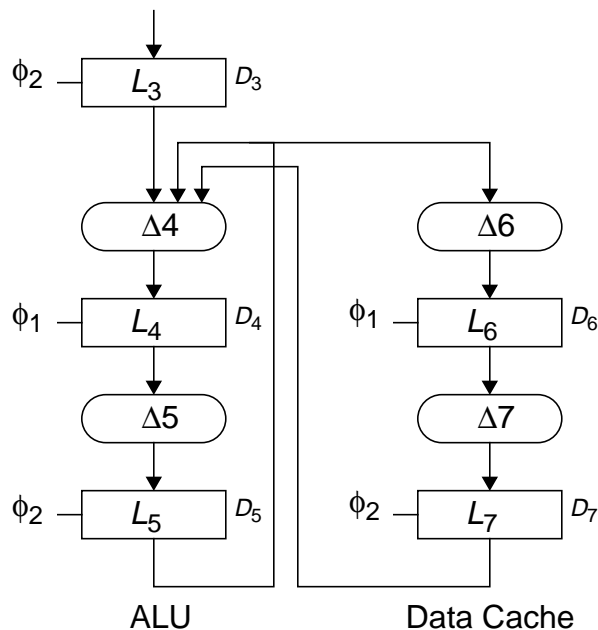
$$\forall i, j \in L \quad D_i = \max(0, \max\langle D_j + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i} \rangle) \quad (5-6)$$

From these constraints, one can either verify if a design will operate at a target frequency or compute the minimum possible frequency at which the design functions. Szymanski

and Shenoy presents a relaxation algorithm for the timing verification problem [73] while Sakallah *et al.* reformulates the constraints as a linear program for cycle time optimization [62]. We will return to these algorithms in Section 5.6.

In the meantime, let us consider an example to get accustomed to the notation. Consider the microprocessor core shown in Figure 5-1. The circuit consists of two clocks  $\{\phi_1, \phi_2\}$  and five latches  $\{L_3, L_4, \dots, L_7\}$  with logic blocks with propagation delays  $\Delta_4$  through  $\Delta_7$ . Latches  $L_4$  and  $L_5$  comprise the ALU, while  $L_6$  and  $L_7$  comprise the data cache. The ALU results may be bypassed back for use on a subsequent ALU operation or may be sent as an input to the data cache. The data cache output may be returned as an input to the ALU. Assume that the latch setup time  $\Delta_{DCi}$  and propagation delay  $\Delta_{DQi}$  are 0, and that the external input to  $L_3$  arrives early enough so that it can depart the latch at  $D_3 = 0$ . The clocks have a target cycle time  $T_c = 10$  units and 50% duty cycle giving phase length  $T_p = T_{\phi_1} = T_{\phi_2} = T_c/2$ . The complete set of setup and propagation constraints are listed in Section 5.9.1. Let us see how logic delays determine latch departure times.

**Figure 5-1** Example circuit for timing analysis



If the logic delays are  $\Delta_4 = 5$ ;  $\Delta_5 = 5$ ;  $\Delta_6 = 5$ ;  $\Delta_7 = 5$ , the latch departure times are:  $D_4 = 0$ ;  $D_5 = 0$ ;  $D_6 = 0$ ;  $D_7 = 0$ . This case illustrates perfectly balanced logic. Each combina-

tional block contains exactly half a cycle of logic. Data arrives at each latch just as it becomes transparent. Therefore, all the departure times are 0.

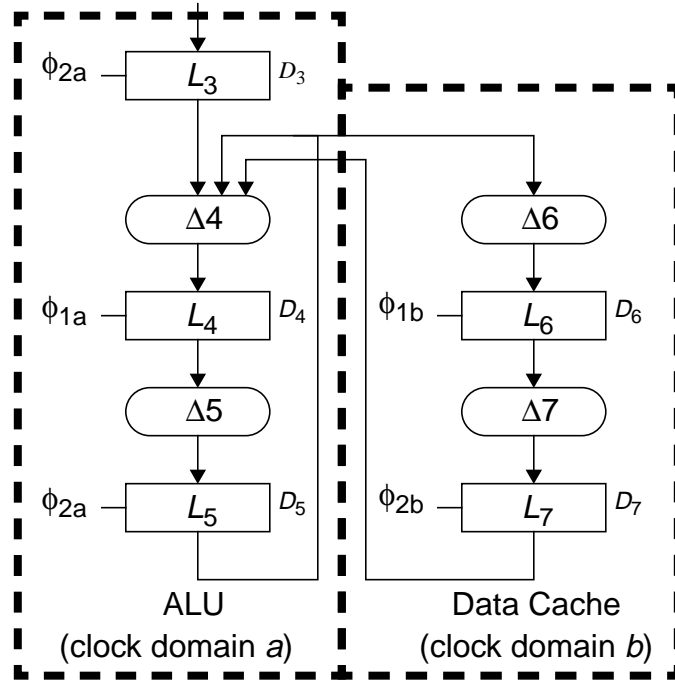
If the logic delays are  $\Delta_4 = 7$ ;  $\Delta_5 = 3$ ;  $\Delta_6 = 5$ ;  $\Delta_7 = 4$ , the departure times are:  $D_4 = 2$ ;  $D_5 = 0$ ;  $D_6 = 0$ ;  $D_7 = 0$ . This case illustrates time borrowing between half-cycles. Combinational block 4 contains more than half a cycle of logic, but it can borrow time from combinational block 5 to complete the entire ALU operation in one cycle. Combinational block 7 finishes early, but cannot depart latch 7 until the latch becomes transparent; this is known as clock blocking. The positive departure time indicates the time borrowing through  $L_4$ .

### 5.3 Timing Analysis with Clock Skew

Recall from Section 4.1.1 that a system has a small number of logical clocks, but possibly a much greater number of skewed physical clocks. Sakallah's formulation, discussed in the previous section, does not account for clock skew; in other words, it assumes that all clocked elements receive their ideal logical clock. Because clock skews are becoming increasingly important, we now examine how to include skew in timing analysis. We first describe a simple modification to the setup constraints which account for a single clock skew budget across the chip. Unfortunately, this is very pessimistic because most clocked elements see much less than worst case skew. Next we develop an exact analysis allowing for different skews between each pair of clocks. This leads to an explosion in the number of timing constraints for large circuits. By making a simple approximation of clock domains, we finally formulate the problem with fewer constraints in a way which is conservative, yet less pessimistic than the single skew approach.

To illustrate systems with clock skew, we use a more elaborate model of the ALU from Figure 5-1. Our new model, shown in Figure 5-2, contains clocks  $C = \{\phi_{1a}, \phi_{1b}, \phi_{2a}, \phi_{2b}\}$  where physical clocks  $\phi_{1a}$  and  $\phi_{1b}$  are nominally identical to logical clock  $\phi_1$ , but are located in different parts of the chip and subject to skew. Only a small  $t_{skew}^{local}$  exists between clocks in the same domain, but the larger  $t_{skew}^{global}$  may occur between clocks in different domains.

Figure 5-2 Example circuit with clock domains



### 5.3.1 Single Skew Formulation

The simplest and most conservative way to accommodate clock skew in timing analysis is to use a single upper bound on clock skew. Suppose that we assume a worst case amount of clock skew,  $t_{skew}^{global}$ , may exist between any two clocked elements on an integrated circuit. Shenoy [65] shows that such skew can be accommodated in the analysis by modifying the setup time constraint. Data must setup before the falling edge of the clock, yet there may be skew between launching and receiving elements such that the data was launched off a late clock edge and is sampled on an early edge. Therefore, we must add clock skew to the effective setup time:

#### L1S. Setup Constraints with Single Skew:

$$\forall i \in L \quad D_i + \Delta_{DCi} + t_{skew}^{global} \leq T_{p_i} \quad (5-7)$$

The propagation constraints are unchanged.

### 5.3.2 Exact Skew Formulation

In a real clock distribution system, clock skews between adjacent elements are typically much less than skews between widely separated elements. We can avoid budgeting global skew in all paths by considering the actual launching and receiving elements and only budgeting the possible skew which exists between the elements.

Unfortunately, the transparency of latches makes this a complex problem. Consider the setup time on a signal arriving at latch  $L_4$  in Figure 5-2. How much skew must be budgeted in the setup time? The answer depends on the skew between the clock which originally launched the signal and the falling edge of  $\phi_{1a}$ , the clock which is receiving the signal. For example, the signal might have been launched from  $L_7$  on the rising edge of  $\phi_{2b}$ , in which case  $t_{\text{skew}}^{\phi_{2b}(r), \phi_{1a}(f)}$  must be budgeted. On the other hand, the signal might have been launched from  $L_5$  on the rising edge of  $\phi_{2a}$ , then propagated through  $L_6$  and  $L_7$  while both latches were transparent. In such a case, only the smaller skew  $t_{\text{skew}}^{\phi_{2a}(r), \phi_{1a}(f)}$  must be budgeted because the launching and receiving clocks are in the same local domain despite the fact that the signal propagated through transparent elements in a different domain. We see that exact timing analysis with varying amounts of skew between elements must track not only the accumulated delay to each element, but also the clock of the launching element.

To track both accumulated delay and launching clock, we can define a vector of arrival and departure times at each latch, with one dimension per physical clock in the system. These times are still nominal, not including skew.

- $A_i^c$ : the arrival time, relative to the beginning of  $p_i$ , of a valid data signal launched by clock  $c$  and now at the input to latch  $i$
- $D_i^c$ : the departure time, relative to the beginning of  $p_i$ , at which the signal launched by clock  $c$  and available at the data input of latch  $i$  starts to propagate through the latch

The setup constraints must budget the skew  $t_{\text{skew}}^{c(r), p_i(f)}$  between the rising edge of the launching clock  $c$  and the falling edge of the clock  $p_i$  controlling the sampling element:

$$\forall i \in L, c \in C \quad D_i^c + \Delta_{DCi} + t_{\text{skew}}^{c(r), p_i(f)} \leq T_{p_i} \quad (5-8)$$



The arrival time at latch  $i$  for a path launched by clock  $c$  depends on the propagation delay and departure times from other latches for signals also launched by clock  $c$ :

$$\forall i, j \in L, c \in C \quad A_i^c = \max(D_j^c + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i}) \quad (5-9)$$

If a latch is transparent when its input arrives, data should depart the latch at the same time it arrives and with respect to the same launching clock. If a latch is opaque when its input arrives, the path from the launching clock will never constrain timing and a new path should be started departing at time 0, launched by the latch's clock. Because of skew between the launching and receiving clocks, the receiving latch may be transparent even if the input arrives at a slightly negative time. To model this effect, we allow departure times with respect to a clock other than that which controls the latch to be negative, equal to the arrival times. Departure times with respect to the latch's own clock are strictly nonnegative. To achieve this, we define an identity operator  $I_{\phi_1, \phi_2}$  on a pair of clocks  $\phi_1$  and  $\phi_2$  which is the minimum departure time for a signal launched by one clock and received by the other: 0 if  $\phi_1 = \phi_2$  and negative infinity if the clocks are different.

These setup and propagation constraints are summarized below. Notice that the number of constraints is proportional to the number of distinct clocks in the system. Also, notice that the constraints are orthogonal; there is no mixing of constraints from different launching clocks.

**L1E. Setup Constraints with Exact Skew Analysis:**

$$\forall i \in L, c \in C \quad D_i^c + \Delta_{DCi} + t_{\text{skew}}^{c(r), p_i(f)} \leq T_{p_i} \quad (5-10)$$

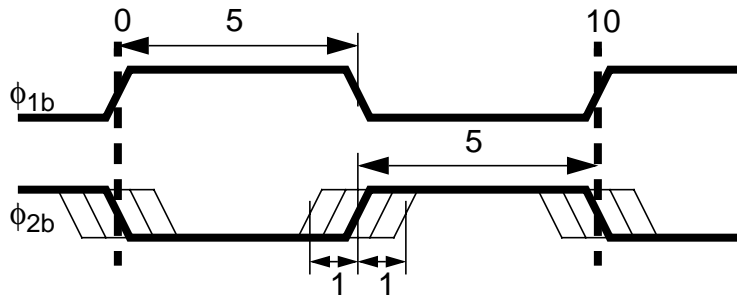
**L2E. Propagation Constraints with Exact Skew Analysis:**

$$\forall i, j \in L, c \in C \quad D_i^c = \max(I_{c, p_i}, \max(D_j^c + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i})) \quad (5-11)$$

A brief example may help explain negative departure times. Consider a path launched from  $L_6$  in Figure 5-2 on the rising edge of  $\phi_{1b}$ :  $D_6^{\phi_{1b}} = 0$ . Let the cycle time  $T_c$  be 10 units, and  $t_{\text{skew}}^{\phi_{1b}, \phi_{2b}}$  be 1. Therefore,  $\phi_{2b}$  may transition up to one unit of time earlier or later than nominal, relative to  $\phi_{1b}$ , as shown in Figure 5-3. Also, suppose the latch propagation delay is 0, so  $A_7^{\phi_{1b}} = \Delta_7 - 5$ . If  $\Delta_7$  is less than 4, the signal arrives at  $L_7$  before the latch

becomes transparent, even under worst case clock skew. If  $\Delta T$  is between 4 and 6 units, corresponding to  $A_7^{\phi_{1b}}$  in the range of -1 to 1, the signal arrives at  $L_7$  when the latch might be transparent, depending on the actual skew between  $\phi_{1b}$  and  $\phi_{2b}$ . If  $\Delta T$  is between 6 and 9 units, the signal arrives at  $L_7$  when the latch is definitely transparent. Because the signal may depart the latch at the same time as it arrives when the latch is transparent, the departure time  $D_7^{\phi_{1b}}$  may physically be as early as -1. We allow the departure time to be arbitrarily negative; if it is more negative than -1, it will always be less critical than the path departing  $L_7$  on the rising edge of  $\phi_{2b}$ . In Section 5.6, we will consider pruning departure times that are negative enough to always be noncritical for the sake of computational efficiency. Departure times must be nonnegative with respect to the clock controlling the latch; for example,  $D_7^{\phi_{2b}} \geq 0$ .

**Figure 5-3 Clock waveforms including local skew**



### 5.3.3 Clock Domain Formulation

The exact timing analysis formulation leads to an explosion in the number of constraints required for a system with many clocks; a system with  $k$  clocks has  $k$  times as many constraints as the single skew formulation. We would like to develop an approximate analysis which gives more accurate results than the single skew formulation, yet has fewer constraints than the exact formulation. To do this, we will formalize the concepts of skew hierarchies and clock domains.

A skew hierarchy is a collection of sets of clocks in the system. The sets are called clock domains. Each clock domain  $d \subset C$  of the hierarchy has an associated number  $h$  which is called the level of the clock domain. A skew hierarchy has  $n$  levels, where level 1 clock domains are the smallest domains and the level  $n$  domain contains all the clocks  $C$  of the system. Define  $H = \{1, \dots, n\}$  to be the set of levels. To be a strict hierarchy, clock domains

must not partially overlap; in other words, for any pair of clock domains, either one is a subset of the other or the domains are disjoint. If one domain contains another, the larger domain has the higher level.  $n = 1$  corresponds assuming worst case skew everywhere.  $n = 2$  is another interesting case, corresponding to a system with local and global skews. We define the following skew hierarchy variables:

- $t_{\text{skew}}^h$ : the upper bound on skew between two clocks in a level  $h$  clock domain. This quantity monotonically increases with  $h$ . The top level domain experiences global skew:  $t_{\text{skew}}^n = t_{\text{skew}}^{\text{global}}$ .
- $h_{ij}$ : the level of the smallest clock domain containing clocks  $i$  and  $j$ , i.e., the minimum  $h$  such that  $t_{\text{skew}}^h \geq t_{\text{skew}}^{i,j}$ .

We can also refer to skew between individual edges of clocks within a clock domain. For example,  $t_{\text{skew}}^{1(r,r)}$  is the skew between rising edges of two clocks within a local clock domain. Because duty cycle variation occurs independently of clock domains, such skew between the same pair of edges is likely to be much smaller than the skew between different edges, such as  $t_{\text{skew}}^{1(r,f)}$ .

Skew hierarchies apply especially well to systems constructed in a hierarchical fashion. For example, Figure 4-2 illustrates an H-tree clock distribution network. It attempts to provide a logical two-phase clock consisting of  $\phi_1$  and  $\phi_2$  to the entire chip with zero skew. Although there are only two phases, the system actually contains 16 physical clocks for the purpose of modeling skew. All of the wire lengths in principle can be perfectly matched, so it is ideally possible to achieve zero systematic clock skew in the global distribution network. Even so, there is some RC delay along the final clock wires. Also, process and environmental variation in the delays of wires and buffers in the distribution network cause random clock skew. The clock skews between various phases depend on the level of their common node in the H-tree. For example,  $\phi_{1tl}$  and  $\phi_{2tl}$  only see a small amount of skew, caused by the final stage buffers and local routing. On the other hand,  $\phi_{1tl}$  and  $\phi_{1rbr}$  on opposite corners of the chip may experience much more skew. The boxes show how the clocks could be collected into a five level skew hierarchy.

The concept of skew hierarchies also applies to other distribution systems. For example, in a grid-based clock system, as used on the DEC Alpha 21164 [24], local skew is defined to be the RC skew between elements in a 500  $\mu\text{m}$  radius, while global skew is defined to be the RC skew between any clocked elements on the die. Global skew is 90 ps, while local skew is only 25ps. Therefore, the chip could be partitioned into distinct 500  $\mu\text{m}$  blocks so that elements communicating within blocks only see local skew, while elements communicating across blocks experience global skew.

The huge vector of timing constraints in the exact analysis are introduced because we track the launching clock of each path so that when the path crosses to another clock domain, then returns to the original domain, only local skew must be budgeted at the latches in the original domain. An alternative is to only track whether a signal is still in the same domain as the launching clock or if it has ever crossed out of the local domain. In the first case, we budget only local clock skew. In the second case, we always budget global clock skew, even if the path returns to the original domain. This is conservative; for example, in Figure 5-2, a path which starts in the ALU, then passes through the data cache while the cache latches are transparent and returns to the ALU would unnecessarily budget global skew upon return to the ALU. However, it greatly reduces the number of constraints, because we must only track whether the path should budget global or local skew, leading to only twice as many constraints as the single skew formulation. In general, we can extend this approach to handle  $n$  levels of hierarchical clock domains.

Again, we define multiple departure times, now referenced to the clock domain level of the signal rather than to the launching clock.

- $A_i^h$ : the arrival time, relative to the beginning of  $p_i$ , of a valid data signal on a path which has crossed clock domains at level  $h$  of the clock domain hierarchy and is now at the input to latch  $i$
- $D_i^h$ : the departure time, relative to the beginning of  $p_i$ , at which the signal which has crossed clock domains at level  $h$  of the clock domain hierarchy and is now available at the data input of latch  $i$  starts to propagate through the latch

When a path crosses clock domains, it is bumped up to budget the greater skew; in other words, the skew level at the receiver is the maximum of the skew level of the launched signal and the actual skew level between the clocks of the departing and receiving latches. As usual, departure times with respect to the latch's own clock are strictly nonnegative while departure times with respect to other clocks may be negative. Because we do not track the actual launching clock, but treat all clocks within a level 1 clock domain the same, we require that departure times from level 1 domains be nonnegative. To achieve this, we define an identity operator  $I_h$  on a level of the skew hierarchy which is the minimum departure time for a departure time at that level of the hierarchy: 0 for departures with respect to level 1, and negative infinity for departures with respect to higher levels.

The setup and propagation constraints are listed below. Notice that the number of constraints is now proportional only to the number of levels of the clock domain hierarchy, not the number of clocks or even the number of domains. For a system with two levels of clock domains, i.e. local and global, this requires only twice as many constraints as the single skew formulation.

**L1D. Setup Constraints with Clock Domain Analysis:**

$$\forall i \in L, h \in H \quad D_i^h + \Delta_{DCi} + t_{\text{skew}}^{h(r,f)} \leq T_{p_i} \quad (5-12)$$

**L2D. Propagation Constraints with Clock Domain Analysis:**

$$\begin{aligned} \forall i, j \in L, h_1 \in H, h_2 = \max(h_1, h_{p_i p_j}) \\ D_i^{h_2} = \max(I_{h_2}, \max(D_j^{h_1} + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i})) \end{aligned} \quad (5-13)$$

Yet another option is to lump clocks into a modest number of local clock domains, then perform an exact analysis on paths which cross clock domains. The number of constraints in such an analysis is proportional to the number of local clock domains, which is smaller than the number of physical clocks required for exact analysis, but larger than the number of levels of clock domains. Paths within a local domain always budget local skew. This hybrid approach avoids unnecessarily budgeting global skew for paths which leave a local domain but return a receiver in the local domain.

### 5.3.4 Example

Let us return to the microprocessor example of Figure 5-2 to illustrate applying timing analysis to systems with four clocks and a two-level skew hierarchy. We will enumerate the timing constraints for each formulation, then solve them to obtain minimum cycle time. This example will illustrate time borrowing, the impact of global and local skews, and the conservative approximations made by the inexact algorithms.

Suppose the nominal clock and latch parameters are identical to those in the example of Section 5.2, but that the system experiences  $t_{\text{skew}}^{\text{local}} = 1$  of skew between clocks in a particular domain and  $t_{\text{skew}}^{\text{global}} = 3$  of skew between clocks in different domains.

The timing constraints are tabulated in Section 5.9 for each formulation and were entered into a linear programming system. Various values of  $\Delta_4$  to  $\Delta_7$  were selected to test the analysis. The values were all selected so that a cycle time of 10 could be achieved in the case of no skew. The examples illustrate well-balanced logic, time borrowing between phases and across cycles, cycles limited by local and global skews, and a case in which the clock domain analysis yields conservative results.

Table 5-1 shows the values of combinational logic delay and cycle times achieved in each example. Bold data indicates conservative results caused by inexact analysis throwing away information. The clock domains results match the exact results in all cases but one, in which a path started in the ALU, passed through the cache while the latches were transparent, and returned to the ALU. Only local skew must be budgeted on return, but the clock domain analysis method conservatively budgeted global skew, leading to a pessimistic cycle time. The single skew formulation is conservative in three cases which used large

amounts of time borrowing where only local skew actually applied but global skew was unnecessarily budgeted.

**Table 5-1 Examples of timing analysis results**

$\Delta 4$	$\Delta 5$	$\Delta 6$	$\Delta 7$	$T_c$ exact	$T_c$ clock domains	$T_c$ single skew	Notes
5	5	5	5	10	10	10	balanced logic
6	3	6	5	10	10	10	time borrowing
0.5	9.5	2.5	5	10.5	10.5	<b>12.5</b>	local skew limit
2	8	5	5	10.67	10.67	<b>11</b>	global skew limit
8	2	5	5	11	11	11	global skew limit
7	2	6	5	10	<b>10.5</b>	<b>10.5</b>	conservative result

## 5.4 Extension to Flip-Flops and Domino Circuits

So far, we have addressed the question of timing analysis for transparent latches. Pulsed latches have identical cycle time constraints as transparent latches and therefore are also handled. We can easily extend the framework to mix latches and edge-triggered flip-flops, which are simpler because they do not allow time borrowing. We can also extend the framework to handle domino circuits, which may have the timing requirements of latches or flip-flops, depending on usage. The main change introduced in this section is to track both arrival and departure times, because inputs to edge-triggered devices must arrive some setup time before the edge and do not depart until after the edge. We present only the exact analysis; the simplified formulation assuming clock domains is very similar.

### 5.4.1 Flip-Flops

For flip-flops, data must arrive before the rising edge of the clock phase, rather than the falling edge. Let  $F = \{F_1, F_2, \dots, F_f\}$  be the set of flip-flops. Data always departs the flop at the rising edge. We must therefore separately track arrival and departure times and introduce a set of departure constraints which relate arrival and departure times and nonnegativity. The setup and departure constraints are written differently for flip-flops and latches:

#### Setup Constraints for flip-flops

$$\forall i \in F, c \in C \quad A_i^c + \Delta_{DCi} + t_{\text{skew}}^{c(r), p_i(r)} \leq 0 \quad (5-14)$$

Note that the sampling edge for a flip-flop is the rising edge, so the skew is between two rising edges, rather than between the rising edge of the launching clock and the falling edge of the sampling clock as is the case for latches.

### Setup Constraints for latches

$$\forall i \in L, c \in C \quad A_i^c + \Delta_{DCi} + t_{\text{skew}}^{c(r), p_i(f)} \leq T_{p_i} \quad (5-15)$$

### Departure constraints for flip-flops

$$\forall i \in F \quad D_i^{p_i} = 0 \quad (5-16)$$

Note that there is no departure constraint from clocks other than the flop's launching clock because flip-flops are not transparent.

### Departure constraints for latches

$$\forall i \in L, c \in C \quad D_i^c = \max(I_{c, p_i}, A_i^c) \quad (5-17)$$

These departure constraints now capture the non-negativity constraint of the latch.

### Propagation Constraints for all elements

$$\forall i \in L \cup F, j \in L \cup F, c \in C \quad A_i^c \geq D_j^c + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i} \quad (5-18)$$

Although this formulation has more variables than the formulations including only latches, it actually involves less computation because the arrival times of latches are just intermediate variables requiring no more computation and because flip-flop analysis is simpler than latch analysis because time borrowing never occurs. Also note that we can use the same setup constraints for flip-flops as for latches if we substitute  $T_i = 0$  for flip-flop clocks.

## 5.4.2 Domino Gates

Domino gates can easily be extended to this framework. When inputs to a domino gate are monotonically rising, they may arrive after the gate has entered evaluation and the domino gate may be modeled exactly as a latch. When the inputs to the domino gate are nonmonotonic, they must arrive before the gate has entered evaluation and the gate may be modeled



as a flip-flop for cycle-time calculations, with the additional caveat that the inputs must not change while the gate is evaluating; i.e. the hold time is quite long. Hold times only appear in min-delay calculations and are discussed in the next section. Additional constraints can be added to ensure precharge finishes in time. The amount of skew budgeted at the interface of nonmonotonic to domino logic depends on the skew between launching and receiving clocks. In the case of a path which starts at a domino gate, passes through some non-monotonic logic, and loops back to the same domino gate, the skew may be only the cycle-to-cycle jitter of the domino clock. In summary, one can determine the monotonicity of inputs from the timing type labels described in Section 3.1.3 and model domino gates as latches or flip-flops, accordingly, with additional constraints to ensure precharge is fast enough.

## 5.5 Min-Delay

Timing analyzers must not only compute long paths, but also short paths. Indeed, short paths are more serious because a chip can operate at reduced clock frequency if paths are longer than predicted, but will not operate at any frequency if min-delay constraints are not met. Such min-delay analysis checks that data launched from one latch or flip-flop will not propagate through logic so quickly as to violate the hold time of the next clocked element. Therefore, min-delay analysis only must check from one element to its successor; this is much easier than cycle time analysis in which a path may borrow time through many transparent latches.

To avoid min-delay failure, data departing one element must pass through enough delay that it does not violate the hold time of the next element. The earliest data could possibly depart an element is at time 0 with respect to the element's local clock; this earliest time is guaranteed to occur if the chip is run at reduced frequency where no time borrowing occurs. We define minimum propagation delays through the clocked element and combinational logic:

- $\Delta_{CDi}$ : the hold time for latch  $i$  required between the trailing edge of the clock input and the time data changes again

- $\delta_{DQi}$ : the minimum propagation delay of latch  $i$  from the data input to the data output while the clock input is high
- $\delta_{ij}$ : the minimum propagation delay through combinational logic between latch  $i$  and latch  $j$ . If there are no combinational paths from latch  $i$  to latch  $j$ ,  $\delta_{ij} \equiv \infty$ .

Equation 5-19 describes this min-delay constraint between adjacent latches and flip-flops. A circuit is safe from race-through if, for every consecutive pair of clocked elements, data from the earlier element cannot arrive at the later element until some hold time after the previous sampling edge of the later element. In the worst case, data departs one element on the rising edge of its clock at time zero and arrives at the next after the minimum propagation delay through the element and combinational logic. Time is adjusted by the phase shift operator to be relative to the receiver's clock. Data must not arrive at the receiver until a hold time after its sampling edge of the previous cycle; clock skew between the launching and receiving clocks effectively increases the hold time. Note that the sampling edge is the falling edge for latches, but the rising edge for flip-flops. As in Section 5.4, we substitute  $T_i = 0$  for edge-triggered flip-flops.

$$\forall i, j \in L \cup F \quad \delta_{DQj} + \delta_{ji} + S_{p_j p_i} \geq T_i + \Delta_{CDi} + t_{\text{skew}}^{p_j(r), p_i(r/f)} - T_c \quad (5-19)$$

Better estimates of the skew between launching and receiving clocks makes guaranteeing min-delay constraints easier. A conservative design may assume a single worst-case skew between all elements on the chip; this leads to excessive minimum propagation delay requirements between elements. By using a skew hierarchy or computing actual skews between each clock, smaller skews can be budgeted between nearby elements.

As discussed in Section 4.1.2, excess skew causes complete failure in the case of hold time violations, but only reduced operating frequency in setup time violations. Therefore, the designer may use a more conservative skew budget for min-delay than max-delay analysis. Fortunately, min-delay races occur between clocks launched from the same global clock edge on the same cycle, so the skew budget does not include cycle-to-cycle jitter or duty cycle variation.

## 5.6 A Verification Algorithm

Szymanski and Shenoy present a relaxation algorithm for verifying that timing constraints are met at a given cycle time assuming no clock skew [73]. We extend the algorithm to handle arbitrary skews between elements and prune unnecessary constraints, as shown in the pseudo-code of Figure 5-4. A key aspect of the algorithm is the introduction of extra variables for each latch,  $D_i^{\max}$  and  $c_i^{\max}$ , which track the latest departure from a latch with respect to any launching clock so that other paths through the latch can be pruned if they cannot be critical.

**Figure 5-4 Pseudocode for timing verification**

```

1   For each latch  $i$ :                                     ; Initialization
2        $D_i^{P_i} = 0$ 
3        $D_i^{\max} = 0$  ;  $c_i^{\max} = p_i$ 
4       Enqueue  $D_i^{P_i}$ 
5   For each flip-flop  $i$ :
6        $D_i^{P_i} = 0$ 
7       Enqueue  $D_i^{P_i}$ 
8   While queue is not empty                               ; Iteration
9       Dequeue  $D_j^c$ 
10      For each latch  $i$  in fanout of  $j$ 
11           $A = D_j^c + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i}$ 
12          If  $(A > D_i^c)$  AND  $\left( A + t_{\text{diff}}^{c_i^{\max}(r), c(r)} > D_i^{\max} \right)$ 
13              If  $(A + \Delta_{DCi} + t_{\text{skew}}^{c(r), p_i(f)} > T_{p_i})$ 
14                  Report setup time violation
15              Else
16                   $D_i^c = A$ 
17                  Enqueue  $D_i^c$ 
18                  If  $(A > D_i^{\max})$ 
19                       $D_i^{\max} = A$  ;  $c_i^{\max} = c$ 
20      For each flip-flop  $i$  in fanout of  $j$ 

```

```

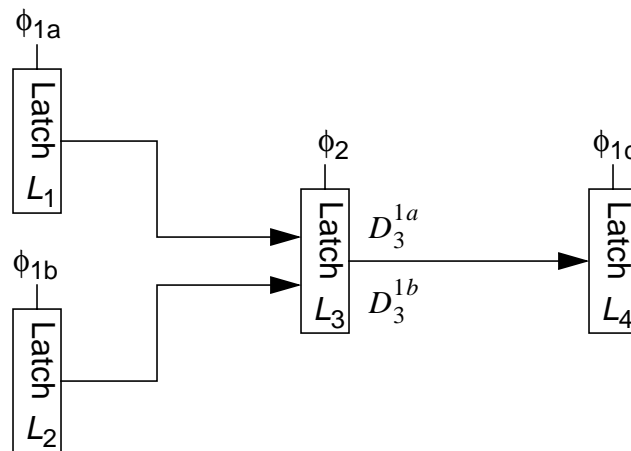
21       $A = D_j^c + \Delta_{DQj} + \Delta_{ji} + S_{p_j p_i}$ 
22      If  $(A + \Delta_{DCi} + t_{skew}^{c(r), p_i(r)} > 0)$ 
23          Report setup time violation

```

Let us first see how this algorithm handles latches, then return to the simpler case of flip-flops. The algorithm initializes the departure times from each latch with respect to its own clock to be zero. It also initializes a variable  $D_i^{\max}$  to track the latest departure time from the latch with respect to any clock and a variable  $c_i^{\max}$  to track the clock that launched that latest departure (Step 3). The algorithm then follows paths from each latch to its successors and computes the arrival time at the successors with respect to the launching clock (Step 11).

A key idea of the algorithm is to prune paths which arrive early enough that they could not possibly be more critical than existing paths. To be potentially more critical and hence avoid pruning, an arrival time must satisfy two conditions (Step 12). One is that the arrival time must be later than all other departure times with respect to the same clock. The other is that the arrival time must potentially be as critical as the latest previously discovered departure time. If there were no clock skew or a single global skew budget everywhere, an arrival would only be more critical than the latest existing departure time if it actually were later:  $A > D_i^{\max}$ . However, we allow different amounts of skew between different clocks. Figure 5-5 shows how this complicates our pruning:

**Figure 5-5 Pruning of paths with different clock skews**



Suppose that  $t_{\text{skew}}^{\phi_{1a}, \phi_{1c}} = 3$ , while  $t_{\text{skew}}^{\phi_{1b}, \phi_{1c}} = 1$ . Suppose that the departure time from  $L_3$   $D_3^{1b}$  on a path launched from  $L_2$  is 2 units and that we find a path arrives at  $L_3$  from  $L_1$  at time 1 unit. Can we prune this path? If the clock skews were the same, we could because the path from  $L_1$  arrives earlier than the path from  $L_2$ . Because the clock skews are different, however, data launched from  $L_1$  must arrive at  $L_4$  earlier than data launched from  $L_2$ . Therefore, the path from  $L_1$  may also be critical even though its departure time is earlier.

Clearly, if one path arrives at a latch more than the worst case global clock skew before another, the early path cannot possibly be critical and may be trimmed. We can prune more aggressively by computing the difference in clock skews between a pair of launching clocks  $\phi_i$  and  $\phi_j$  and any possible receiving clock,  $t_{\text{diff}}^{\phi_i, \phi_j}$ :

$$t_{\text{diff}}^{\phi_i, \phi_j} = \max(t_{\text{skew}}^{\phi_i, \phi_r} - t_{\text{skew}}^{\phi_j, \phi_r}) \quad \forall \phi_r \in C \quad (5-20)$$

From this definition, we can show that  $t_{\text{diff}}^{\phi_i, \phi_j} \leq t_{\text{skew}}^{\phi_i, \phi_j}$ . Moreover, in a system budgeting a single global skew between all clocks,  $t_{\text{diff}}$  is 0 and negative departure times never occur, agreeing with the single skew formulation.

We check this criteria (Step 12), pruning all paths with arrival times before the latest departure by more than the clock skew between the launching clock  $c$  of the path under consideration and the launching clock  $c_i^{\text{max}}$  of the path causing the latest departure time. If the path is not pruned, it is checked for setup time violations and added to the queue so that paths to subsequent latches can be checked. Also, if it is later than the latest previously discovered departure time, it replaces the previous time (Step 19).

Flip-flops are handled in a similar fashion, but are much simpler because no time borrowing takes place. As discussed in Section 5.4.2, domino gates are analyzed either as latches or flip-flops, depending on the monotonicity of the inputs.

The algorithm performs a depth first path search if elements are enqueued at the head of the queue and a breadth first search if elements are enqueued at the tail. Breadth first is likely to be faster because it can prune paths earlier.

The algorithm is very similar to one which assumes no clock skew, but may take longer because it may trace multiple paths through the same latch. This occurs when paths originating at different latches with skew between them all arrive at a common latch at nearly the same time. Fortunately real systems tend to have a relatively small number of critical paths passing through any given latch so the runtime is likely to increase by much less than the number of constraints. Timing analysis with clock domains is similar to analysis with exact skew. The runtime may be somewhat improved because a hierarchy of  $h$  levels of clock domains must trace at most  $h$  paths through any given latch. Of course, the results are more conservative.

So far, we have addressed the question of verifying that a design meets a cycle time goal because this is the primary question asked by designers. It is also straightforward to compute the minimum cycle time of a design using Sakallah's linear programming approach [62]. The constraints as presented are not quite linear because they involve the max function. The max function can be replaced by multiple inequalities, transforming the constraints into linear inequalities while preserving the minimum cycle time. These inequalities can be solved by linear programming techniques. Although conventional linear programming is much slower than the relaxation algorithm for verifying cycle time, new interior point methods [83] may be quite efficient.

## 5.7 Results

To evaluate the costs and benefits of the exact formulation, we analyzed a timing model of MAGIC, the Memory and General Interconnect Controller of the FLASH supercomputer [43], implemented in a 0.6 micron CMOS process. MAGIC includes a 2-way superscalar RISC processing engine and several large data buffers. We extracted a timing model from the Standard Delay Format (SDF) data produced by LSI Logic tools, then trimmed long paths such as those involving reset or scan. After trimming, we found 1819 latches and 10559 flip-flops connected by 593153 combinational paths (Model A). To obtain an entirely latch-based design, we replaced each flip-flop with a pair of latches and divided the path delay between the two latches, obtaining a system with 22937 latches (Model B).

The chip was partitioned into ten units, each a local clock domain. We assumed 500 ps of global skew between domains and 250 ps of local skew within domains.

We applied the timing analysis algorithm of Section 5.6 to the timing model. Table 5-2 shows the minimum cycle times achievable and number of latch departures enqueued in each run, a measure of the analysis cost. Model B is uniformly faster than Model A because latches allows the system to borrow time across cycles, solving some critical paths. The exact analysis shows that the system can run 50-90 ps faster than a single skew analysis conservatively predicts. Each latch departure is enqueued at least once when its departure time is initialized to 0. Paths borrowing time enqueue later departure times. The exact analysis also enqueues more latch departures because potentially critical paths from multiple launching clocks may pass through a single latch. The exact analysis enqueues 143 more than the single skew analysis in Model A and 333 more in Model B. These differences are less than 4% of the total number of departures, indicating that pruning makes the exact analysis only slightly more expensive than the single skew approximation. In all cases, the CPU time for analysis is under a second, much shorter than the time required to read the timing model from disk.

**Table 5-2 Timing Analysis Results**

	Model A	Model B
Single Skew	9.43 ns 3866 departures	8.05 ns 24995 departures
Exact Skew	9.38 ns 4009 departures	7.96 ns 25328 departures

These results indicate that the exact skew formulation works well in practice because only a small fraction of paths require time borrowing, as noted by Szymanski and Shenoy [73], and because an even smaller fraction of paths involve negative departure times. In this particular problem, no critical paths depart a clock domain and return to it, so the clock domain formulation would have found equally good cycle times. However, the cost of the exact skew formulation is low enough that no approximations are necessary.

## 5.8 Summary

In this chapter, we have extended the latch-based timing analysis formulation of Sakallah, Mudge, and Olukotun to handle clock skew, especially different amounts of clock skew between different elements. Allowing a single amount of clock skew everywhere effectively increases the setup time of each latch. An exact analysis allowing different amounts of skew between different elements involves tracking the clock which launched each path so that paths which leave a local skew domain and then return only budget the local skew. This leads to a multiplication of constraints proportional to the number of clocks. By making a conservative approximation that the chip is hierarchically divided into clock domains with bounded skew within a level of the clock domain hierarchy and that once a path leaves a domain, it continues to budget the larger skew even upon return, the number of constraints only increases proportionally to the number of levels of the domain hierarchy. In particular, a two-level hierarchy, consisting of local and global clock domains, provides less pessimistic timing analysis at a modest computational cost. Most practical systems use clocked elements besides just transparent or pulsed latches, so we also incorporate edge-triggered flip-flops and domino gates into the timing analysis formulation by separately tracking arrival and departure times at each clocked element. In addition to verifying cycle time, we check for min-delay violations, effectively increasing the hold time of each element by the potential clock skew between launching and receiving elements. Finally, we presented an relaxation algorithm for verifying the timing constraints. The uncertainty from the clock skew may increase the number of paths that must be searched, but results on the MAGIC chip show that this increase is very modest because most paths do not borrow time.

## 5.9 Appendix: Timing Constraints

To illustrate the formulations described in Section 5.2 and Section 5.3, we present a complete set of the timing constraints of each formulation, applied to the simple microprocessor example from Figure 5-1 and Figure 5-2.

### 5.9.1 Skewless Formulation

The timing constraints with no skew are:



### Setup Constraints

$D_4 \leq T_{\phi_1}$	$D_5 \leq T_{\phi_2}$	$D_6 \leq T_{\phi_1}$	$D_7 \leq T_{\phi_2}$
-----------------------	-----------------------	-----------------------	-----------------------

### Propagation Constraints

$D_4 = \text{max of:}$	$D_5 = \text{max of:}$	$D_6 = \text{max of:}$	$D_7 = \text{max of:}$
0	0	0	0
$D_3 + \Delta 4 - T_p$	$D_4 + \Delta 5 - T_p$	$D_5 + \Delta 6 - T_p$	$D_6 + \Delta 7 - T_p$
$D_5 + \Delta 4 - T_p$			
$D_7 + \Delta 4 - T_p$			

### 5.9.2 Single Skew Formulation

The timing constraints budgeting global skew everywhere are very similar to those with no skew:

### Setup Constraints

$D_4 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1a}}$	$D_5 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2a}}$	$D_6 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1b}}$	$D_7 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2b}}$
--	--	--	--

### Propagation Constraints

$D_4 = \text{max of:}$	$D_5 = \text{max of:}$	$D_6 = \text{max of:}$	$D_7 = \text{max of:}$
0	0	0	0
$D_3 + \Delta 4 - T_p$	$D_4 + \Delta 5 - T_p$	$D_5 + \Delta 6 - T_p$	$D_6 + \Delta 7 - T_p$
$D_5 + \Delta 4 - T_p$			
$D_7 + \Delta 4 - T_p$			

### 5.9.3 Exact Formulation

Because there are four clocks, there are four times as many setup and propagation constraints for the exact analysis:

## Setup Constraints

$D_4^{\phi_{1a}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1a}}$	$D_5^{\phi_{1a}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2a}}$	$D_6^{\phi_{1a}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1b}}$	$D_7^{\phi_{1a}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2b}}$
$D_4^{\phi_{2a}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1a}}$	$D_5^{\phi_{2a}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2a}}$	$D_6^{\phi_{2a}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1b}}$	$D_7^{\phi_{2a}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2b}}$
$D_4^{\phi_{1b}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1a}}$	$D_5^{\phi_{1b}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2a}}$	$D_6^{\phi_{1b}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1b}}$	$D_7^{\phi_{1b}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2b}}$
$D_4^{\phi_{2b}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1a}}$	$D_5^{\phi_{2b}} + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2a}}$	$D_6^{\phi_{2b}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1b}}$	$D_7^{\phi_{2b}} + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2b}}$

## Propagation Constraints

$D_4^{\phi_{1a}} = \text{max of:}$	$D_5^{\phi_{1a}} = \text{max of:}$	$D_6^{\phi_{1a}} = \text{max of:}$	$D_7^{\phi_{1a}} = \text{max of:}$
0			
$D_3^{\phi_{1a}} + \Delta 4 - T_p$	$D_4^{\phi_{1a}} + \Delta 5 - T_p$	$D_5^{\phi_{1a}} + \Delta 6 - T_p$	$D_6^{\phi_{1a}} + \Delta 7 - T_p$
$D_5^{\phi_{1a}} + \Delta 4 - T_p$			
$D_7^{\phi_{1a}} + \Delta 4 - T_p$			

$D_4^{\phi_{2a}} = \text{max of:}$	$D_5^{\phi_{2a}} = \text{max of:}$	$D_6^{\phi_{2a}} = \text{max of:}$	$D_7^{\phi_{2a}} = \text{max of:}$
	0		
$D_3^{\phi_{2a}} + \Delta 4 - T_p$	$D_4^{\phi_{2a}} + \Delta 5 - T_p$	$D_5^{\phi_{2a}} + \Delta 6 - T_p$	$D_6^{\phi_{2a}} + \Delta 7 - T_p$
$D_5^{\phi_{2a}} + \Delta 4 - T_p$			
$D_7^{\phi_{2a}} + \Delta 4 - T_p$			

$D_4^{\phi_{1b}} = \text{max of:}$	$D_5^{\phi_{1b}} = \text{max of:}$	$D_6^{\phi_{1b}} = \text{max of:}$	$D_7^{\phi_{1b}} = \text{max of:}$
		0	
$D_3^{\phi_{1b}} + \Delta 4 - T_p$	$D_4^{\phi_{1b}} + \Delta 5 - T_p$	$D_5^{\phi_{1b}} + \Delta 6 - T_p$	$D_6^{\phi_{1b}} + \Delta 7 - T_p$
$D_5^{\phi_{1b}} + \Delta 4 - T_p$			
$D_7^{\phi_{1b}} + \Delta 4 - T_p$			

$D_4^{\phi_{2b}} = \text{max of:}$	$D_5^{\phi_{2b}} = \text{max of:}$	$D_6^{\phi_{2b}} = \text{max of:}$	$D_7^{\phi_{1a}} = \text{max of:}$
			0
$D_3^{\phi_{2b}} + \Delta 4 - T_p$	$D_4^{\phi_{2b}} + \Delta 5 - T_p$	$D_5^{\phi_{2b}} + \Delta 6 - T_p$	$D_6^{\phi_{1a}} + \Delta 7 - T_p$
$D_5^{\phi_{2b}} + \Delta 4 - T_p$			
$D_7^{\phi_{2b}} + \Delta 4 - T_p$			

### 5.9.4 Clock Domain Formulation

Finally, we write the constraints with the approximation of clock domains. Because there are two levels of the clock domain hierarchy, this requires only twice as many constraints as the single skew formulation. The timing constraints are:

#### Setup Constraints

$D_4^1 + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1a}}$	$D_5^1 + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2a}}$	$D_6^1 + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{1b}}$	$D_7^1 + T_{\text{skew}}^{\text{local}} \leq T_{\phi_{2b}}$
$D_4^2 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1a}}$	$D_5^2 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2a}}$	$D_6^2 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{1b}}$	$D_7^2 + T_{\text{skew}}^{\text{global}} \leq T_{\phi_{2b}}$

#### Propagation Constraints

$D_4^1 = \text{max of:}$	$D_5^1 = \text{max of:}$	$D_6^1 = \text{max of:}$	$D_7^1 = \text{max of:}$
0	0	0	0
$D_3^1 + \Delta 4 - T_p$	$D_4^1 + \Delta 5 - T_p$		$D_6^1 + \Delta 7 - T_p$
$D_5^1 + \Delta 4 - T_p$			

$D_4^2 = \text{max of:}$	$D_5^2 = \text{max of:}$	$D_6^2 = \text{max of:}$	$D_7^2 = \text{max of:}$
$D_7^1 + \Delta 4 - T_p$		$D_5^1 + \Delta 6 - T_p$	
$D_3^2 + \Delta 4 - T_p$	$D_4^2 + \Delta 5 - T_p$	$D_5^2 + \Delta 6 - T_p$	$D_6^2 + \Delta 7 - T_p$
$D_5^2 + \Delta 4 - T_p$			
$D_7^2 + \Delta 4 - T_p$			

# Chapter 6 Conclusions

*Prediction is very difficult, especially if it's about the future.*

--Nils Bohr

Digital system clock frequencies have been exponentially increasing at a remarkable rate and are expected to continue this increase for the foreseeable future. Part of the increase comes from process improvements, but part arises from using fewer gates per cycle and more speedy domino logic. As the number of gate delays per cycle shrinks, fixed amounts of sequencing overhead consume a greater fraction of the clock period. Traditional domino circuits, while offering fast raw gate delays, suffers particularly large overhead from clock skew, latch delay, and the inability to balance logic between cycles through time borrowing. This overhead can waste 25% or more of the cycle time in aggressive systems. Fortunately, the designer can hide such overhead through better design techniques. Skew-tolerant domino circuits use overlapping clocks and eliminate latches to hide the overhead and allow modest amounts of time borrowing. Four-phase skew-tolerant circuits use simple clock generation circuits and can handle about a quarter cycle of clock skew, so they are attractive for current designs.

Adopting skew-tolerant domino has a number of design implications. Since overhead must still be budgeted at each interface from glitchy static logic into domino pipelines, critical paths are best built entirely from domino logic. Without the availability of static logic to perform nonmonotonic functions, dual-rail domino is usually required. This increases the number of wires, the area, and the power consumption of such domino blocks; however, static logic may be unable to attain the target operating frequency at any area or power.

Skew-tolerant domino must be cleanly integrated with the rest of a design for best performance. Static inputs are latched to remain stable while driving domino. Domino outputs use a fast N-latch before driving static logic so precharge does not ripple through the static gates. Signal names may be tagged with suffixes, called timing types, to remind the designer when the information is available for sampling and to permit static checking of legal connectivity among latches and domino gates. Systems cannot be economically built

unless they can also be easily tested during debug and manufacturing. To assist test, skew-tolerant domino can be incorporated onto the same scan chain as static logic.

Unfortunately, global clock skew tends to track as a fraction of clock distribution network delay. Since integrated circuits are getting larger, the ratio of wire to gate delay is rising, and the clock load of chips is increasing, clock distribution delays are not improving as fast as cycle times, leading to greater clock skew as a fraction of cycle time. Systems operating above 1 GHz may not be able to achieve acceptably low global skew across the entire die at reasonable cost. Instead of abandoning the synchronous paradigm entirely for an asynchronous approach, designers will divide the die into local clock domains offering tolerable amounts of skew within each domain. Communication between clock domains may occur at reduced frequency, through source-synchronous FIFOs, or with via an asynchronous interface [9], [20].

The more one examines clock skew, the more special cases can be found to less conservatively budget skew. Of course, local skew is smaller than global skew. Skew between common edges of a clock is smaller than skew between different edges because of duty cycle variation in the clock network. Skews between clocks on a particular cycle are smaller than skews between consecutive cycles because of jitter; this is particularly important because it reduces the skews that might cause min-delay violations. Much like process variation, skew is a statistical effect. Therefore, for max-delay analysis one may budget expected, rather than worst case skew, just as one designs to typical, rather than SS processing, when seeking to get reasonable yield at target frequency. However, min-delay analysis should more conservatively budget worst case skew because min-delay violations prevent the chip from operating at any frequency.

In order to take advantage of different skew budgets between different pairs of clocks, improved timing analysis tools are necessary. Arrival times of data cease to have absolute meaning in such systems. Instead, arrival times must be specified relative to a particular launching clock which determines the skew to the receiver. Timing analysis requires a vector of arrival times at each latch with respect to different launching clocks. Skew hierarchies can be used to reduce the size of these vectors at the expense of some conservatism in timing analysis.

By adopting skew-tolerant circuit techniques and more carefully modeling skews between communicating blocks, designers will continue building clocked systems into the multi-GHz regime. Ultimately, systems may be limited more by power consumption than clocking: only a finite amount of heat can be removed from a die with a reasonably priced cooling system. Improvements in clock gating will disable some inactive domino gates to reduce the activity factor, but domino will continue to burn large amounts of power. Will domino gates become too expensive from a power perspective, or will designers find it better to build simpler machines with fewer transistors running at extreme domino speeds than very complex machines churning along at the speed of static logic? Domino presents other challenges as well. The aspect ratios of wires continues to grow, making coupling problems greater. The design time of domino also can be high, possibly increasing time to market. Will static circuits become a better choice for teams with finite resources, or will advances in CAD tools improve domino productivity? Circuit design should remain an exciting field as these issues are explored in the coming decade.

## Bibliography

- [1] D. Bailey and B. Benschneider, "Clocking Design and Analysis for a 600-MHz Alpha Microprocessor," *IEEE J. Solid-State Circuits*, Nov. 1998, vol. 33, no. 11, pp. 1627-1633.
- [2] D. Bearden, *et al.*, "A 133 MHz 64b Four-Issue CMOS Microprocessor," *ISSCC Dig. Tech. Papers*, pp. 174-175, Feb. 1995.
- [3] L. Boonstra, *et al.*, "A 4096-b One-Transistor per Bit Random-Access Memory with Internal Timing and Low Dissipation," *IEEE J. Solid-State Circuits*, Oct. 1973, vol. SC-8, no. 5, p. 305-10.
- [4] W. Bowhill *et al.*, "A 300 MHz 64 b Quad-Issue CMOS Microprocessor," *ISSCC Dig. Tech. Papers*, Feb. 1995, pp. 182-183.
- [5] W. Bowhill, *et al.*, "Circuit Implementation of a 300-MHz 64-bit Second-generation CMOS Alpha CPU," *Digital Technology Journal*, vol. 7, no. 1, pp. 100-119, 1995.
- [6] T. Burks, K. Sakallah, and T. Mudge, "Critical Paths in Circuits with Level-Sensitive Latches," *IEEE Trans. VLSI Sys.*, vol. 3, no. 2, pp. 273-291, June 1995.
- [7] A. Champernowne, L. Bushard, J. Rusterholtz, and J. Schomburg, "Latch-to-Latch Timing Rules," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 798-808, June 1990.
- [8] T. Chao, *et al.*, "Zero Skew Clock Routing with Minimum Wirelength," *IEEE Trans. Circuits Syst.-II*, vol. 39, no. 11, pp. 799-814, Nov. 1992.
- [9] D. Chapiro, *Globally-Asynchronous Locally Synchronous Systems*, Ph.D. dissertation, CS Department, Stanford University, Stanford, CA, May 1984.
- [10] K. Chu and D. Pulfrey, "Design Procedures for Differential Cascode Voltage Switch Circuits," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 6, pp. 1082-1087, Dec. 1986.
- [11] R. Colwell and R. Steck, "A 0.6 $\mu$ m BiCMOS Processor with Dynamic Execution," *ISSCC Dig. Tech. Papers*, pp. 176-177, Feb. 1995.

- [12] W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.
- [13] S. DasGupta, E. Eichelberger, and T. Williams, "LSI Chip Design for Testability," *ISSCC Dig. Tech. Papers*, pp. 216-217, Feb. 1978.
- [14] D. Dobberpuhl *et al.*, "A 200 MHz 64 b Dual-Issue CMOS Microprocessor," *IEEE J. Solid-State Circuits*, vol. 27, no. 11, pp. 1555-1567, Nov. 1992.
- [15] D. Dobberpuhl, *et al.*, "A 200-MHz 64-bit Dual-issue CMOS Microprocessor," *Digital Technology Journal*, vol. 4, no. 4, pp. 35-50, 1992.
- [16] H. Fair and D. Bailey, "Clocking Design and Analysis for a 600 MHz Alpha Microprocessor," *ISSCC Dig. Tech. Papers*, pp. 398-399, Feb. 1998.
- [17] E. Friedman, editor, *Clock Distribution Networks in VLSI Circuits and Systems*, New York: IEEE Press, 1995.
- [18] N. Gaddis and J. Lotz, "A 64-b Quad-Issue CMOS RISC Microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, Nov. 1996, pp. 1697-1702.
- [19] B. Gieseke, *et al.*, "A 600 MHz Superscalar RISC Microprocessor with Out-of-Order Execution," *ISSCC Dig. Tech. Papers*, pp. 176-177, Feb. 1997.
- [20] R. Ginosar and R. Kol, "Adaptive Synchronization," *Proc. Intl. Conf. Comp. Design*, pp. 188-189, Oct. 1998.
- [21] N. Gonclaves and H. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures," *IEEE J. Solid-State Circuits*, vol. SC-18, no. 3, pp. 261-266, June 1983.
- [22] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, Sept. 1996, vol. 31, no. 9, pp. 1277-84.
- [23] P. Gronowski and B. Bowhill, "Dynamic Logic and Latches - Part II," in *Proceedings of VLSI Circuits Workshop*, VLSI Circuits Symp., June 1996.



- [24] P. Gronowski, *et al.*, "A 433-MHz 64-b Quad-Issue RISC Microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1687-1696, Nov. 1996.
- [25] P. Gronowski, *et al.*, "High-Performance Microprocessor Design," *IEEE J. Solid-State Circuits*, vol. 33, no. 5, pp. 676-686, May 1998.
- [26] A. Hall, *Synthesis of Double Rank Sequential Circuits*, Tech. Report #53, EE Digital Systems Lab, Princeton University, Dec. 1966.
- [27] D. Harris, S. Oberman, and M. Horowitz, "SRT Division Architectures and Implementations," in *Proc. 13th IEEE Symposium on Computer Arithmetic*, July 1997.
- [28] D. Harris and M. Horowitz, "Skew-Tolerant Domino Circuits," *IEEE J. Solid-State Circuits*, vol. 32, no. 1, pp. 1702-1711, Nov. 1997.
- [29] C. Heikes, "A 4.5 mm<sup>2</sup> Multiplier Array for a 200MFLOP Pipelined Coprocessor," *ISSCC Dig. Tech. Papers*, pp. 290-291, Feb. 1994.
- [30] C. Heikes and G. Colon-Bonet, "A Dual Floating Point Coprocessor with an FMAC Architecture," *ISSCC Dig. Tech. Papers*, pp. 354-355, Feb. 1996
- [31] L. Heller, *et al.*, "Cascode Voltage Switch Logic: a Differential CMOS Logic Family," *ISSCC Dig. Tech. Papers*, pp. 16-17, Feb. 1984.
- [32] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, San Francisco: Morgan Kaufman, 1999, Chapter 1.
- [33] R. Hitchcock, "Timing Verification and Timing Analysis Program," in *25 Years of Electronic Design Automation*, IEEE/ACM, 1988.
- [34] M. Horowitz, "High Frequency Clock Distribution," in *Proceedings of VLSI Circuits Workshop*, VLSI Circuits Symp., June 1996.
- [35] Intel Corporation, *Opportunistic Time-Borrowing Domino Logic*, US Patent #5,517,136, May 14, 1996.

- [36] Intel Corporation, "Intel Microprocessor Quick Reference Guide," courtesy of Intel Museum, Santa Clara, CA, 1997.
- [37] Special issue on soft errors, *IBM J. Research & Dev.*, vol. 40, no. 1, Jan. 1996.
- [38] A. Ishii, C. Leiserson, and M. Papaefthymiou, "Optimizing Two-Phase, Level-Clocked Circuitry," in *J. ACM*, vol. 44, no. 1, pp. 148-199, Jan. 1997.
- [39] N. Jouppi, *Timing Verification and Performance Improvement of MOS VLSI Designs*, Ph.D. thesis, Stanford University, 1984.
- [40] V. von Kaenel, *et al.*, "A 600 MHz CMOS PLL Microprocessor Clock Generator with a 1.2 GHz VCO," *ISSCC Dig. Tech. Papers*, pp. 396-397, Feb. 1998.
- [41] F. Klass, "Semi-Dynamic and Dynamic Flip-Flops with Embedded Logic," *Symposium on VLSI Circuits Dig. Tech. Papers*, pp. 108-109, June 1998.
- [42] R. Krambeck, C. Lee, and H. Law, "High-Speed Compact Circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. SC-17, no. 3, pp. 614-619, 1982.
- [43] J. Kuskin, *et al.*, "The Stanford FLASH Multiprocessor," in *Proc. Intl. Symp. Comp. Arch.*, pp. 302-313, Apr. 1994.
- [44] P. Larsson and C. Svensson, "Noise in Digital Dynamic CMOS Circuits," *IEEE J. Solid-State Circuits*, vol. 29, no. 6, June 1994.
- [45] L. Lev, *et al.*, "A 64-b Microprocessor with Multimedia Support," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, Nov. 1995.
- [46] L. Lev, "Signal and Power Network Integrity," in *Proceedings of VLSI Circuits Workshop, VLSI Circuits Symp.*, June 1996.
- [47] J. Lotz, *et al.*, "A Quad-Issue Out-of-Order RISC CPU," in *ISSCC Dig. Tech. Papers*, Feb. 1996, pp. 210-211.

- [48] M. Matsui, *et al.*, "A 200-MHz 13 mm<sup>2</sup> 2-D DCT Macrocell Using Sense-Amplifier Pipeline Flip-Flop scheme," *IEEE J. Solid-State Circuits*, vol. 29, no. 12, pp. 1482-91, Dec. 1994.
- [49] C. Mead and L. Conway, *Introduction to VLSI Systems*, Reading, MA: Addison-Wesley, 1980.
- [50] *Microprocessor Report*, Sebastopol, CA: MicroDesign Resources, 1995-1998.
- [51] J. Montanaro, *et al.*, "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, 1703-14, Nov. 1996.
- [52] A. Mukherjee, *Introduction to nMOS and CMOS VLSI Systems Design*, Englewood Cliffs, NJ: Prentice-Hall, 1986.
- [53] K. Nowka and T. Galambos, "Circuit Design Techniques for a Gigahertz Integer Microprocessor," in *Proc. Intl. Conf. Comp. Design*, pp. 11-16, October 1998.
- [54] D. Noice, *A Clocking Discipline for Two-Phase Digital Integrated Circuits*, Stanford University Technical Report, Jan. 1983.
- [55] J. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 336-349, July 1985.
- [56] H. Partovi, *et al.*, "Flow-Through Latch and Edge-Triggered Flip-flop Hybrid Elements," *ISSCC Dig. Tech. Papers*, pp. 138-139, Feb. 1996.
- [57] W. Penny and L. Lau, *MOS Integrated Circuits: Theory, Fabrication, Design and Systems Applications of MOS LSI*, New York: Van Nostrand, Reinhold, 1973, Chapter 5.
- [58] J. Rabaey, *Digital Integrated Circuits*, Upper Saddle River, NJ: Prentice Hall, 1996.
- [59] B. Razavi, ed., *Monolithic Phase-Locked Loops and Clock Recovery Circuits*, New York: IEEE Press, 1996.

- [60] P. Restle and A. Deutsch, "Designing the Best Clock Distribution Network," *Proc. VLSI Symp.*, pp. 2-5, June 1998.
- [61] J. Rubinstein, P. Penfield, and M. Horowitz, "Signal Delay in RC Tree Networks", in *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, pp. 202-211, July 1983.
- [62] K. Sakallah, T. Mudge, and O. Olukotun, "Analysis and Design of Latch-Controlled Synchronous Digital Circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 3, pp. 322-333, March 1992.
- [63] *The National Technology Roadmap for Semiconductors*, Austin, TX: SEMATECH, 1997. (<http://notes.sematech.org/97melec.htm>)
- [64] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "A Pseudo-Polynomial Algorithm for Verification of Clocking Schemes," in *Tau 92*, 1992.
- [65] N. Shenoy, *Timing Issues in Sequential Circuits*, Ph.D. dissertation, University of California, Berkeley, 1993.
- [66] K. Shepard, *et al.*, "Design Methodology for the S/390 Parallel Enterprise Server G4 Microprocessors," *IBM J. Research and Development*, vol. 41, no 4-5, pp. 515-547, July-Sept. 1997.
- [67] M. Shoji, "Electrical Design of BELLMAC-32A Microprocessor," *Proc. IEEE Int'l Conf. Circuits and Computers*, pp. 112-115, Sept. 1982.
- [68] M. Shoji, "Elimination of Process-Dependent Clock Skew in CMOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 5, pp. 875-880, Oct. 1986.
- [69] M. Shoji, *High-Performance CMOS Circuits*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [70] I. Sutherland, R. Sproull, and D. Harris, *Logical Effort*, San Francisco, CA: Morgan Kaufmann, 1999.

- [71] T. Szymanski, "LEADOUT: A Static Timing Analyzer for MOS Circuits," in *ICCAD-86 Dig. Tech. Papers*, 1986, pp. 130-133.
- [72] T. Szymanski, "Computing Optimal Clock Schedules," in *Proc. 29th Design Automation Conf.*, pp. 399-404, 1992.
- [73] T. Szymanski and N. Shenoy, "Verifying clock schedules," in *ICCAD Dig. Tech. Papers*, pp. 124-131, Nov. 1992.
- [74] T. Thorp, G. Yee, and C. Sechen, "Domino Logic Synthesis Using Complex Gates," *Proc. Intl. Conf. Computer-Aided Design*, Nov. 1998.
- [75] R. Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Trans. Computer-Aided Desi.*, vol. 12, no. 2, pp. 242-249, Feb. 1993.
- [76] S. Unger and C. Tan, "Clocking Schemes for High-speed Digital Systems," *IEEE Trans. Comput.*, vol. C-35, pp. 880-895, Oct 1986.
- [77] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Reading, MA: Addison-Wesley, 1993, p. 351.
- [78] T. Williams, *Self-timed rings and their application to division*, Ph.D. dissertation, EE Department, Stanford University, Stanford, CA, May 1991.
- [79] T. Williams and M. Horowitz, "A Zero-Overhead Self-Timed 160-ns 54-b CMOS Divider," *IEEE J. Solid-State Circuits*, vol. 26, no. 11, pp. 1651-1661, Nov. 1991.
- [80] S. Unger and C. Tan, "Clocking Schemes for High-Speed Digital Systems," *IEEE Trans. Comput.*, vol. C-35, no. 10, pp. 880-895, Oct. 1986.
- [81] N. Vasseghi, *et al.*, "200 MHz Superscalar RISC Processor," *IEEE J. Solid State Circuits*, vol. 31, No. 11, pp. 1675-1686, Nov. 1996.
- [82] A. Vittal and M. Marek-Sadowska, "Crosstalk Reduction for VLSI," *IEEE Transactions on CAD*, vol. 16, no. 3, pp. 290-298, March 1997.

- [83] Y. Ye, "Interior Point Algorithms: Theory and Analysis," New York: Wiley, 1997.
- [84] J. Yuan and C. Svensson, "High Speed CMOS Circuit Technique," *IEEE J. Solid-State Circuits*, vol. 24, no. 1, pp. 62-69, Feb. 1989.
- [85] J. Yuan, C. Svensson, and P. Larsson, "New domino logic precharged by clock and data," *Electronics Letters*, vol. 29, No. 25, pp.2188-2189, Dec. 1993.
- [86] J. Yuan and C. Svensson, "New Single-Clock SMOS Latches and Flipflops with Improved Speed and Power Savings," *IEEE J. Solid-State Circuits*, vol. 32, no. 1, pp. 62-69, Jan. 1997.