7.1 Introduction

The previous chapter addressed combinational circuits in which the output is a function of the current inputs. This chapter discusses sequential circuits in which the output depends on previous as well as current inputs; such circuits are said to have state. Finite state machines and pipelines are two important examples of sequential circuits.

Sequential circuits are usually designed with flip-flops or latches, which are sometimes called memory elements, that hold data called tokens. The purpose of these elements is not really memory; instead, it is to enforce sequence, to distinguish the current token from the previous or next token. Therefore, we will call them sequencing elements [Harris01]. Without sequencing elements, the next token might catch up with the previous token, garbling both. Sequencing elements delay tokens that arrive too early, preventing them from catching up with previous tokens. Unfortunately, they inevitably add some delay to tokens that are already critical, decreasing the performance of the system. This extra delay is called sequencing overhead.

This chapter considers sequencing for both static and dynamic circuits. Static circuits refer to gates that have no clock input, such as complementary CMOS, pseudo-nMOS, or pass transistor logic. Dynamic circuits refer to gates that have a clock input, especially domino logic. To complicate terminology, sequencing elements themselves can be either static or dynamic. A sequencing element with static storage employs some sort of feedback to retain its output value indefinitely. An element with dynamic storage generally maintains its value as charge on a capacitor that will leak away if not refreshed for a long period of time. The choices of static or dynamic for gates and for sequencing elements can be independent.

Sections 7.2–7.4 explore sequencing elements for static circuits, particularly flip-flops, 2-phase transparent latches, and pulsed latches. Section 7.5 delves into a variety of ways to sequence dynamic circuits. A periodic clock is commonly used to indicate the timing of a sequence. Section 7.6 describes how external signals can be synchronized to the clock and analyzes the risks of synchronizer failure. Wave pipelining is discussed in Section 7.7. Clock generation and distribution will be examined further in Section 12.5.

The choice of sequencing strategy is intimately tied to the design flow that is being used by an organization. Thus, it is important before departing on a design direction to ensure that all phases of design capture, synthesis, and verification can be accommodated.
This includes such aspects as cell libraries (Are the latch or flip-flop circuits and models available?); tools such as timing analyzers (Can timing closure be achieved easily?); and automatic test generation (Can self-test elements be inserted easily?).

7.2 Sequencing Static Circuits

Recall from Section 1.4.9 that latches and flip-flops are the two most commonly used sequencing elements. Both have three terminals: data input (D), clock (clk), and data output (Q). The latch is transparent when the clock is high and opaque when the clock is low; in other words, when the clock is high, D flows through to Q as if the latch were just a buffer, but when the clock is low, the latch holds its present Q output even if D changes. The flip-flop is an edge-triggered device that copies D to Q on the rising edge of the clock and ignores D at all other times. These are illustrated in Figure 7.1. The unknown state of Q before the first rising clock edge is indicated by the pair of lines at both low and high levels.

This section explores the three most widely used methods of sequencing static circuits with these elements: flip-flops, 2-phase transparent latches, and pulsed latches [Unger86, Harris01]. An ideal sequencing methodology would introduce no sequencing overhead, allow sequencing elements back-to-back with no logic in between, grant the designer flexibility in balancing the amount of logic in each clock cycle, tolerate moderate amounts of clock skew without degrading performance, and consume zero area and power. We will compare these methods and explore the tradeoffs they offer. We will also examine a number of transistor-level circuit implementations of each element.
7.2 SEQUENCING STATIC CIRCUITS

7.2.1 Sequencing Methods

Figure 7.2 illustrates three methods of sequencing blocks of combinational logic. In each case, the clock waveforms, sequencing elements, and combinational logic are shown. The horizontal axis corresponds to the time at which a token reaches a point in the circuit. For example, the token is captured in the first flip-flop on the first rising edge of the clock. It propagates through the combinational logic and reaches the second flip-flop on the second rising edge of the clock. The dashed vertical lines indicate the boundary between one clock cycle and the next. The clock period is $T_c$. In a 2-phase system, the phases may be separated by $t_{\text{nonoverlap}}$. In a pulsed system, the pulse width is $t_{\text{pulse}}$.

Flip-flop-based systems use one flip-flop on each cycle boundary. Tokens advance from one cycle to the next on the rising edge. If a token arrives too early, it waits at the flip-flop until the next cycle. Recall that the flip-flop can be viewed as a pair of back-to-back
latches using $clk$ and its complement, as shown in Figure 7.3. If we separate the latches, we can divide the full cycle of combinational logic into two phases, sometimes called half-cycles. The two latch clocks are often called $\phi_1$ and $\phi_2$. They may correspond to $clk$ and its complement $\overline{clk}$ or may be nonoverlapping ($t_{\text{nonoverlap}} > 0$). At any given time, at least one clock is low and the corresponding latch is opaque, preventing one token from catching up with another. The two latches behave in much the same manner as two watertight gates in a canal lock [Mead80]. Pulsed latch systems eliminate one of the latches from each cycle and apply a brief pulse to the remaining latch. If the pulse is shorter than the delay through the combinational logic, we can still expect that a token will only advance through one clock cycle on each pulse.

Table 7.1 defines the delays and timing constraints of the combinational logic and sequencing elements. These delays may differ significantly for rising and falling transitions and can be distinguished with an $r$ or $f$ suffix. For brevity, we will use the overall maximum and minimum.
Figure 7.4 illustrates these delays in a timing diagram. In a timing diagram, the horizontal axis indicates time and the vertical axis indicates logic level. A single line indicates that a signal is high or low at that time. A pair of lines indicates that a signal is stable but that we don’t care about its value. Criss-crossed lines indicate that the signal might change at that time. A pair of lines with cross-hatching indicates that the signal may change once or more over an interval of time.

Figure 7.4(a) shows the response of combinational logic to the input $A$ changing from one arbitrary value to another. The output $Y$ cannot change instantaneously. After the contamination delay $t_{cd}$, $Y$ may begin to change or glitch. After the propagation delay $t_{pd}$, $Y$ must have settled to a final value. The contamination delay and propagation delay may be very different because of multiple paths through the combinational logic. Figure 7.4(b) shows the response of a flip-flop. The data input must be stable for some window around the rising edge of the flop if it is to be reliably sampled. Specifically, the input $D$ must have settled by some setup time $t_{setup}$ before the rising edge of $clk$ and should not change again until a hold time $t_{hold}$ after the clock edge. The output begins to change after a clock-to-$Q$ contamination delay $t_{ccq}$ and completely settles after a clock-to-$Q$ propagation delay $t_{pcq}$. Figure 7.4(c) shows the response of a latch. Now the input $D$ must set up and hold around the falling edge that defines the end of the sampling period. The output initially changes $t_{ccq}$ after the latch becomes transparent on the rising edge of the clock and settles by $t_{pcq}$. Section 7.4.4 discusses how to measure the setup and hold times and propagation delays in simulation.
Ideally, the entire clock cycle would be available for computations in the combinational logic. Of course, the sequencing overhead of the latches or flip-flops cuts into this time. If the combinational logic delay is too great, the receiving element will miss its setup time and sample the wrong value. This is called a setup time failure or max-delay failure. It can be solved by redesigning the logic to be faster or by increasing the clock period. This section computes the actual time available for logic and the sequencing overhead of each of our favorite sequencing elements: flip-flops, two-phase latches, and pulsed latches.

Figure 7.5 shows the max-delay timing constraints on a path from one flip-flop to the next, assuming ideal clocks with no skew. The path begins with the rising edge of the clock triggering $F_1$. The data must propagate to the output of the flip-flop $Q_1$ and through the combinational logic to $D_2$, setting up at $F_2$ before the next rising clock edge. This implies that the clock period must be at least

$$T_c \geq t_{pq} + t_{pd} + t_{setup}$$  \hspace{1cm} (7.1)

Alternatively, we can solve for the maximum allowable logic delay, which is simply the cycle time less the sequencing overhead introduced by the propagation delay and setup time of the flip-flop.

$$t_{pd} \leq T_c - \left( t_{setup} + t_{pq} \right)$$  \hspace{1cm} (7.2)
Example

The Arithmetic/Logic Unit (ALU) self-bypass path limits the clock frequency of some pipelined microprocessors. For example, the Integer Execution Unit (IEU) of the Itanium 2 contains self-bypass paths for six separate ALUs, as shown in Figure 7.6(a) [Fetzer02]. The path for one of the ALUs begins at registers containing the inputs to an adder, as shown in Figure 7.6(b). The adder must compute the sum (or difference, for subtraction). A result multiplexer chooses between this sum, the output of the logic unit, and the output of the shifter. Then a series of bypass multiplexers selects the inputs to the ALU for the next cycle. The early bypass multiplexer chooses among results of ALUs from previous cycles and is not on the critical path. The 8:1 middle bypass multiplexer chooses a result from any of the six ALUs, the early bypass mux, or the register file. The 4:1 late bypass multiplexer chooses a result from either of two results returning from the data cache, the middle bypass mux result, or the immediate operand specified by the next instruction. The late bypass mux output is driven back to the ALU to use on the next cycle. Because the six ALUs and the bypass multiplexers occupy a significant amount of area, the critical path also involves 2-mm wires from the result mux to middle bypass mux and from the middle bypass mux back to the late bypass mux. (Note: In the Itanium 2, the ALU self-bypass path is built from four-phase skew-tolerant domino circuits. For the purposes of these examples, we will hypothesize instead that it is built from static logic and flip-flops or latches.)

For our example, the propagation delays and contamination delays of the path are given in Table 7.2. Suppose the registers are built from flip-flops with a setup time of 62 ps, hold time of –10 ps, propagation delay of 90 ps, and contamination delay of 75 ps. Calculate the minimum cycle time $T_c$ at which the ALU self-bypass path will operate correctly.

<table>
<thead>
<tr>
<th>Table 7.2 Combinational logic delays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
</tr>
<tr>
<td>Adder</td>
</tr>
<tr>
<td>Result Mux</td>
</tr>
<tr>
<td>Early Bypass Mux</td>
</tr>
<tr>
<td>Middle Bypass Mux</td>
</tr>
<tr>
<td>Late Bypass Mux</td>
</tr>
<tr>
<td>2-mm wire</td>
</tr>
</tbody>
</table>

continued
Figure 7.7 shows the analogous constraints on a path using two-phase transparent latches. Let us assume that data $D_1$ arrives at $L_1$ while the latch is transparent ($\phi_1$ high). The data propagates through $L_1$, the first block of combinational logic, $L_2$, and the second block of combinational logic. Technically, $D_3$ could arrive as late as a setup time.

Solution: The critical path involves propagation delays through the adder (590 ps), result mux (60 ps), middle bypass mux (80 ps), late bypass mux (70 ps), and two 2-mm wires (100 ps each), for a total of $t_{pd} = 1000$ ps. According to Eq(7.1), the cycle time $T_c$ must be at least $90 + 1000 + 62 = 1152$ ps.
before the falling edge of $\phi_1$ and still be captured correctly by $L_3$. To be fair, we will insist that $D_3$ nominally arrive no more than one clock period after $D_1$ because, in the long run, it is impossible for every single-cycle path in a design to consume more than a full clock period. Certain paths may take longer if other paths take less time; this technique is called time borrowing and will be addressed in Section 7.2.4. Assuming the path takes no more than a cycle, we see the cycle time must be

$$T_c \geq t_{pd1} + t_{pd1} + t_{pd2} + t_{pd2}$$

(7.3)

Once again, we can solve for the maximum logic delay, which is the sum of the logic delays through each of the two phases. The sequencing overhead is the two latch propagation delays. Notice that the nonoverlap between clocks does not degrade performance in the latch-based system because data continues to propagate through the combinational logic between latches even while both clocks are low. Realizing that a flip-flop can be made from two latches whose delays determine the flop propagation delay and setup time, we see Eq. (7.4) is closely analogous to Eq. (7.2).

$$t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \frac{2t_{pd0}}{\text{sequencing overhead}}$$

(7.4)

\[ \text{FIG 7.7} \quad \text{Two-phase latch max-delay constraint} \]
If the pulse is wide enough, the max-delay constraint for pulsed latches is similar to that of two-phase latches except that only one latch is in the critical path, as shown in Figure 7.8(a). However, if the pulse is narrower than the setup time, the data must set up before the pulse rises, as shown in Figure 7.8(b). Combining these two cases gives

\[ T_c \geq \max\{t_{pdq} + t_{pdq} + t_{pd} + t_{setup} - t_{pu}\} \]  

(7.5)

Solving for the maximum logic delay shows that the sequencing overhead is just one latch delay if the pulse is wide enough to hide the setup time

\[ t_{pd} \leq T_c - \max\{t_{pdq} + t_{pq} + t_{setup} - t_{pu}\} \]  

(7.6)

Ideally, sequencing elements can be placed back to back without intervening combinational logic and still function correctly. For example, a pipeline can use back-to-back registers to sequence along an instruction opcode without modifying it. However, if the hold time is large and the contamination delay is small, data can incorrectly propagate through two successive elements on one clock edge, corrupting the state of the system. This is
7.2 SEQUENCING STATIC CIRCUITS

It can only be fixed by redesigning the logic, not by slowing the clock. Therefore, designers should be very conservative in avoiding such failures because modifying and refabricating a chip is very expensive and time-consuming.

Figure 7.9 shows the min-delay timing constraints on a path from one flip-flop to the next assuming ideal clocks with no skew. The path begins with the rising edge of the clock triggering \( F_1 \). The data may begin to change at \( Q_1 \) after a \( \text{clk-to-Q} \) contamination delay, and at \( D_2 \) after another logic contamination delay. However, it must not reach \( D_2 \) until at least the hold time \( t_{\text{hold}} \) after the clock edge, lest it corrupt the contents of \( F_2 \). Hence we solve for the minimum logic contamination delay:

\[
 t_{\text{cd}} \geq t_{\text{hold}} - t_{\text{cq}}
\]  

(7.7)

Example

Recompute the ALU self-bypass path cycle time if the flip-flop is replaced with a pulsed latch. The pulsed latch has a pulse width of 150 ps, a setup time of 40 ps, a hold time of 5 ps, a \( \text{clk-to-Q} \) propagation delay of 82 ps and contamination delay of 52 ps, and a \( D\)-to-\( Q \) propagation delay of 92 ps.

Solution: \( t_{\text{ps}} \) is still 1000 ps. According to Eq. (7.5), the cycle time must be at least \( 92 + 1000 = 1092 \) ps.
If the contamination delay through the flip-flop exceeds the hold time, you can safely use back-to-back flip-flops. If not, you must explicitly add delay between the flip-flops (e.g., with a buffer) or use special slow flip-flops with greater than normal contamination delay on paths that require back-to-back flops. Scan chains are a common example of paths with back-to-back flops.

Figure 7.10 shows the min-delay timing constraints on a path from one transparent latch to the next. The path begins with data passing through \( L_1 \) on the rising edge of \( \phi_1 \). It must not reach \( L_2 \) until a hold time after the previous falling edge of \( \phi_2 \) because \( L_2 \) should have become safely opaque before \( L_1 \) becomes transparent. As the edges are separated by \( t_{\text{nonoverlap}} \), the minimum logic contamination delay through each phase of logic is

\[
    (7.8) \quad t_{\text{cont}} = t_{\text{hold}} - t_{\text{lag}} - t_{\text{nonoverlap}}
\]

(Note that our derivation found the minimum delay through the first half-cycle, but that the second half-cycle has the same constraint.)

This result shows that by making \( t_{\text{nonoverlap}} \) sufficiently large, hold time failure can be avoided entirely. However, generating and distributing nonoverlapping clocks is very challenging at high speeds. Therefore, most commercial transparent latch-based systems use the clock and its complement. In this case, \( t_{\text{nonoverlap}} = 0 \) and the contamination delay constraint is the same between the latches and flip-flops.

This leads to an apparent paradox: The contamination delay constraint applies to each phase of logic for latch-based systems, but to the entire cycle of logic for flip-flops. Therefore, latches seem to require twice the overall logic contamination delay as compared to flip-flops. Yet flip-flops can be built from a pair of latches! The paradox is resolved by observing that a flip-flop has an internal race condition between the two latches. The flip-flop must be carefully designed so that it always operates reliably.

Figure 7.11 shows the min-delay timing constraints on a path from one pulsed latch to the next. Now data departs on the rising edge of the pulse but must hold until after the falling edge of the pulse. Therefore, the pulse width effectively increases the hold time of the pulsed latch as compared to a flip-flop.

\[
    (7.9) \quad t_{\text{cont}} = t_{\text{hold}} - t_{\text{lag}} + t_{\text{pw}}
\]
7.2 SEQUENCING STATIC CIRCUITS

Two-phase latch min-delay constraint

Pulsed latch min-delay constraint
In a system using flip-flops, data departs the first flop on the rising edge of the clock and must set up at the second flop before the next rising edge of the clock. If the data arrives late, the circuit produces the wrong result. If the data arrives early, it is blocked until the clock edge, and the remaining time goes unused. Therefore, we say the clock imposes a hard edge because it sharply delineates the cycles.

In contrast, when a system uses transparent latches, the data can depart the first latch on the rising edge of the clock, but does not have to set up until the falling edge of the clock on the receiving latch. If one half-cycle or stage of a pipeline has too much logic, it can borrow time into the next half-cycle or stage, as illustrated in Figure 7.12(a) [Bernstein98, Harris01]. Time borrowing can accumulate across multiple cycles. However, in systems with feedback, the long delays must be balanced by shorter delays so that the overall loop completes in the time available. For example, Figure 7.12(b) shows a single-cycle self-bypass loop in which time borrowing occurs across half-cycles, but the entire path must fit in one cycle. A typical example of a self-bypass loop is the execution stage of a pipelined processor in which an ALU must complete an operation and bypass the result back for use in the ALU on a dependent instruction. Most critical paths in digital systems occur in self-bypass loops because otherwise latency does not matter.

Figure 7.13 illustrates the maximum amount of time that a two-phase latch-based system can borrow (beyond the \( T_{c}/2 - t_{cd} \) nominally available to each half-cycle of logic). Because data does not have to set up until the falling edge of the receiving latch’s clock, one phase can borrow up to half a cycle of time from the next (less setup time and non-overlap):

\[
t_{\text{borrow}} \leq \frac{T_{c}}{2} - \left( t_{\text{setup}} + t_{\text{nonoverlap}} \right)
\]  

7.2.4 Time Borrowing

In a system using flip-flops, data departs the first flop on the rising edge of the clock and must set up at the second flop before the next rising edge of the clock. If the data arrives late, the circuit produces the wrong result. If the data arrives early, it is blocked until the clock edge, and the remaining time goes unused. Therefore, we say the clock imposes a hard edge because it sharply delineates the cycles.

In contrast, when a system uses transparent latches, the data can depart the first latch on the rising edge of the clock, but does not have to set up until the falling edge of the clock on the receiving latch. If one half-cycle or stage of a pipeline has too much logic, it can borrow time into the next half-cycle or stage, as illustrated in Figure 7.12(a) [Bernstein98, Harris01]. Time borrowing can accumulate across multiple cycles. However, in systems with feedback, the long delays must be balanced by shorter delays so that the overall loop completes in the time available. For example, Figure 7.12(b) shows a single-cycle self-bypass loop in which time borrowing occurs across half-cycles, but the entire path must fit in one cycle. A typical example of a self-bypass loop is the execution stage of a pipelined processor in which an ALU must complete an operation and bypass the result back for use in the ALU on a dependent instruction. Most critical paths in digital systems occur in self-bypass loops because otherwise latency does not matter.

Figure 7.13 illustrates the maximum amount of time that a two-phase latch-based system can borrow (beyond the \( T_{c}/2 - t_{cd} \) nominally available to each half-cycle of logic). Because data does not have to set up until the falling edge of the receiving latch’s clock, one phase can borrow up to half a cycle of time from the next (less setup time and non-overlap):

\[
t_{\text{borrow}} \leq \frac{T_{c}}{2} - \left( t_{\text{setup}} + t_{\text{nonoverlap}} \right)
\]  

Example

If the ALU self-bypass path uses pulsed latches in place of flip-flops, will it have any hold-time problems?

Solution: Yes. The late bypass mux has \( t_{dy} = 45 \) ps. The pulsed latches have \( t_{pu} = 150 \) ps, \( t_{hold} = 5 \) ps, and \( t_{dy} = 52 \) ps. Hence, EQ (7.9) is badly violated. Src1 may receive imm from the next instruction rather than the current instruction. The problem could be solved by adding buffers after the imm pulsed latch. The buffers would need to add a minimum delay of \( t_{hold} - t_{pu} + t_{dy} = 58 \) ps. Alternatively, the imm pulsed latch could be replaced with a flip-flop without slowing the critical path. If the flip-flop were designed with a very long (110 ps) contamination delay, the race would be avoided.
7.2 SEQUENCING STATIC CIRCUITS

Loops may borrow time internally but must complete within the cycle.

**FIG 7.12** Time borrowing

**FIG 7.13** Maximum amount of time borrowing
Pulsed latches can be viewed as transparent latches with a narrow pulse. If the pulse is wider than the setup time, pulsed latches are also capable of a small amount of time borrowing from one cycle to the next.

Time borrowing has two benefits for the system designer. The most obvious is intentional time borrowing, in which the designer can more easily balance logic between half-cycles and pipeline stages. This leads to potentially shorter design time because the balancing can take place during circuit design rather than requiring changes to the microarchitecture to explicitly move functions from one stage to another. The other is

Example

Suppose the ALU self-bypass path is modified to use two-phase transparent latches. A mid-cycle $\phi_2$ latch is placed after the adder, as shown in Figure 7.14. The latches have a setup time of 40 ps, a hold time of 5 ps, a $clk$-to-$Q$ propagation delay of 82 ps and contamination delay of 52 ps, and a $D$-to-$Q$ propagation delay of 92 ps. Compute the minimum cycle time for the path. How much time is borrowed through the mid-cycle latch at this cycle time? If the cycle time is increased to 2000 ps, how much time is borrowed?

Solution:

According to Eq. (7.3), the cycle time is $T_c = 82 + 590 + 82 + 410 = 1164$ ps. The first half of the cycle involves the latch and adder delays and consumes $82 + 590 = 672$ ps. The nominal half-cycle time is $T_c/2 = 582$ ps. Hence, the path borrows 90 ps from the second half-cycle. If the cycle time increases to 2000 ps and the nominal half-cycle time becomes 1000 ps, time borrowing no longer occurs.
7.2 SEQUENCING STATIC CIRCUITS

### Opportunistic Time Borrowing

Even if the designer carefully equalizes the delay in each stage at design time, the delays will differ from one stage to another in the fabricated chip because of process and environmental variations and inaccuracies in the timing model used by the CAD system. In a system with hard edges, the longest cycle sets the minimum clock period. In a system capable of time borrowing, the slow cycles can opportunistically borrow time from faster ones and average out some of the variation.

Some experienced design managers forbid the use of intentional time borrowing until the chip approaches tapeout. Otherwise designers are overly prone to assuming that their pipeline stage can borrow time from adjacent stages. When many designers make this same assumption, all of the paths become excessively long. Worse yet, the problem may be hidden until full chip timing analysis begins, at which time it is too late to redesign so many paths. Another solution is to do full-chip timing analysis starting early in the design process.

### 7.2.5 Clock Skew

The analysis so far has assumed ideal clocks with zero skew. In reality clocks have some uncertainty in their arrival times that can cut into the time available for useful computation, as shown in Figure 7.15(a). The bold $clk$ line indicates the latest possible clock arrival time. The hashed lines show that the clock might arrive over a range of earlier times because of skew. The worst scenario for max delay in a flip-flop-based system is that the launching flop receives its clock late and the receiving flop receives its clock early. In this case, the clock skew is subtracted from the time available for useful computation and appears as sequencing overhead. The worst scenario for min delay is that the launching flop receives its clock early and the receiving clock receives its clock late, as shown in Figure 7.15(b). In this case, the clock skew effectively increases the hold time of the system.

$$t_{pd} \leq T_c - (t_{pg} + t_{swap} + t_{skew}) \quad (7.12)$$

$$t_{hd} \geq t_{hold} - t_{coy} + t_{skew} \quad (7.13)$$
In the system using transparent latches, clock skew does not degrade performance. Figure 7.16 shows how the full cycle (less two latch delays) is available for computation even when the clocks are skewed because the data can still arrive at the latches while they are transparent. Therefore, we say that transparent latch-based systems are skew-tolerant. However, skew still effectively increases the hold time in each half-cycle. It also cuts into the window available for time borrowing.

\[
\begin{align*}
\tau_{pd} &\leq T_c - \left( 2 \tau_{pld} \right) \\
\tau_{pd1} + \tau_{pd2} &\geq \tau_{hold} - \tau_{overlap} + \tau_{skew}
\end{align*}
\]

(7.14)  (7.15)
7.2 SEQUENCING STATIC CIRCUITS

Pulsed latches can tolerate an amount of skew proportional to the pulse width. If the pulse is wide enough, the skew will not increase the sequencing overhead because the data can arrive while the latch is transparent. If the pulse is narrow, skew can degrade performance. Again, skew effectively increases the hold time and reduces the amount of time available for borrowing (see Exercise 7.7).

In summary, systems with hard edges (e.g., flip-flops) subtract clock skew from the time available for useful computation. Systems with softer edges (e.g., latches) take advan-

Example

If the ALU self-bypass path from Figure 7.6 can experience 50 ps of skew from one cycle to the next between flip-flops in the various ALUs, what is the minimum cycle time of the system? How much clock skew can the system have before hold time failures occur?

Solution: According to Eq. (7.12), the cycle time should increase by 50 ps to 1202 ps. The maximum skew for which the system can operate correctly at any cycle time is 

\[ t_{skew} \leq \frac{T}{2} \left( t_{setup} + t_{nonoverlap} + t_{skew} \right) \]  

(7.16)

![Clock skew and transparent latches](Image)

Pulsed latches can tolerate an amount of skew proportional to the pulse width. If the pulse is wide enough, the skew will not increase the sequencing overhead because the data can arrive while the latch is transparent. If the pulse is narrow, skew can degrade performance. Again, skew effectively increases the hold time and reduces the amount of time available for borrowing (see Exercise 7.7).

\[ t_{pd} \leq T - \max\{t_{pd} + t_{pq} + t_{setup} - t_{pu} + t_{skew}\} \]  

(7.17)

\[ t_{skew} \leq t_{hold} + t_{pu} - t_{pq} + t_{skew} \]  

(7.18)

\[ t_{borrow} \leq t_{pu} - \left( t_{setup} + t_{skew} \right) \]  

(7.19)

In summary, systems with hard edges (e.g., flip-flops) subtract clock skew from the time available for useful computation. Systems with softer edges (e.g., latches) take advan-
tage of the window of transparency to tolerate some clock skew without increasing the sequencing overhead. Clock skew will be addressed further in Section 12.5. In particular, different amounts of skew can be budgeted for min-delay and max-delay checks. Moreover, nearby sequential elements are likely to see less skew than elements on opposite corners of the chip. Current automated place & route tools spend considerable effort to model clock delays and insert buffer elements to minimize clock skew, but skew is a growing problem for systems with aggressive cycle times.

### 7.3 Circuit Design of Latches and Flip-flops

Conventional CMOS latches are built using pass transistors or tristate buffers to pass the data while the latch is transparent and feedback to hold the data while the latch is opaque. We begin by exploring circuit designs for basic latches, then build on them to produce flip-flops and pulsed latches. Many latches accept reset and/or enable inputs. It is also possible to build logic functions into the latches to reduce the sequencing overhead.

A number of alternative latch and flip-flop structures have been used in commercial designs. The True Single Phase Clocking (TSPC) technique uses a single clock with no inversions to simplify clock distribution. The Klass Semidynamic Flip-Flop (SDFF) is a fast flip-flop using a domino-style input stage. Differential flip-flops are good for certain applications. Each of these alternatives are described and compared.

#### 7.3.1 Conventional CMOS Latches

Figure 7.17(a) shows a very simple transparent latch built from a single transistor. It is compact and fast but suffers four limitations. The output does not swing from rail-to-rail (i.e., from GND to VDD); it never rises above $V_{DD} - V_t$. The output is also dynamic; in other words, the output floats when the latch is opaque. If it floats long enough, it can be disturbed by leakage (see Section 6.3.3). D drives the diffusion input of a pass transistor directly, leading to potential noise issues (see Section 6.3.9) and making the delay harder to model with static timing analyzers. Finally, the state node is exposed, so noise on the output can corrupt the state. The remainder of the figures illustrate improved latches using more transistors to achieve more robust operation.

Figure 7.17(b) uses a CMOS transmission gate in place of the single nMOS pass transistor to offer rail-to-rail output swings. It requires a complementary clock $\bar{\phi}$, which can be provided as an additional input or locally generated from $\phi$ through an inverter. Figure 7.17(c) adds an output inverter so that the state node $X$ is isolated from noise on the output. Of course, this creates an inverting latch. Figure 7.17(d) also behaves as an inverting latch with a buffered input but unbuffered output. As discussed in Sections 2.5.6 and 6.2.5.1, the inverter followed by transmission gate is essentially equivalent to a tristate inverter but has a slightly lower logical effort because the output is driven by both transistors of the transmission gate in parallel. Both (c) and (d) are fast dynamic latches.
In modern processes, subthreshold leakage is large enough that dynamic nodes retain their values for only a short time, especially at the high temperature and voltage encountered during burn-in test. Therefore, practical latches need to be staticized, adding feedback to prevent the output from floating, as shown in Figure 7.17(e). When the clock is ‘1,’ the input transmission gate is ON, the feedback tristate is OFF, and the latch is transparent. When the clock is ‘0,’ the input transmission gate turns OFF. However, the feedback tristate turns ON, holding \( X \) at the correct level. Figure 7.17(f) adds an input inverter so the input is a transistor gate rather than unbuffered diffusion. Unfortunately, both (e) and (f) reintroduced output noise sensitivity: A large noise spike on the output can propagate backward through the feedback gates and corrupt the state node \( X \). Figure 7.17(g) is a very robust transparent latch that addresses all of the deficiencies mentioned so far: The latch is static, all nodes swing rail-to-rail, the state noise is isolated from output noise, and the input drives transistor gates rather than diffusion. Such a latch is widely used in standard cell applications including the Artisan standard cell library [Artisan02]. It is recommended for all but the most performance- or area-critical designs.

In semicustom datapath applications where input noise can be better controlled, the inverting latch of Figure 7.17(h) may be preferable because it is faster and more compact; for example, Intel uses this as a standard datapath latch [Karnik01]. Figure 7.17(i) shows the jamb latch, a variation of (g) that reduces the clock load and saves two transistors by using a weak feedback inverter in place of the tristate. This requires careful circuit design to ensure that the tristate is strong enough to overpower the feedback inverter in all process corners. Figure 7.17(j) shows another jamb latch commonly used in register files and Field Programmable Gate Array (FPGA) cells. Many such latches read out onto a single \( D_{out} \) wire and only one is enabled at any given time with its \( RD \) signal. The Itanium 2 processor uses the latch shown in Figure 7.17(k) [Naffziger02]. In the static feedback, the pulldown stack is clocked, but the pullup is a weak pMOS transistor. Therefore, the gate driving the input must be strong enough to overcome the feedback. The Itanium 2 cell library also contains a similar latch with an additional input inverter to buffer the input when the previous gate is too weak or far away. With the input inverter, the latch can be viewed as a cross between the designs shown in (g) and (i). Some latches add one more inverter to provide both true and complementary outputs.

Figure 8.53 shows layouts for the latch of Figure 7.17(h) with a built-in clock inverter. The state node \( X \) can be shared between the transmission gate and tristate diffusions.

The dynamic latch of Figure 7.17(d) can also be drawn as a clocked tristate, as shown in Figure 7.18(a). Such a form is sometimes called clocked CMOS \((C^*MOS)\) [Suzuki73]. The conventional form using the inverter and transmission gate is slightly faster because the output is driven through the nMOS and pMOS working in parallel. \( C^*MOS \) is slightly smaller because it eliminates two contacts. Figure 7.18(b) shows another form of the tristate that swaps the data and clock terminals. It is logically equivalent but electrically inferior because toggling \( D \) while the latch is opaque can cause charge sharing noise on the output node [Suzuki73].
CHAPTER 7  SEQUENTIAL CIRCUIT DESIGN

(a)  
(b)  
(c)  
(d)  
(e)  
(f)  
(g)  
(h)  
(i)  
(j)  
(k)  

FIG 7.17  Transparent latches
All of the latches shown so far are transparent while $\phi$ is high. They can be converted to active-low latches by swapping $\phi$ and $\bar{\phi}$.

### 7.3.2 Conventional CMOS Flip-flops

Figure 7.19(a) shows a dynamic inverting flip-flop built from a pair of back-to-back dynamic latches [Suzuki73]. Either the first or the last inverter can be removed to reduce delay at the expense of greater noise sensitivity on the unbuffered input or output. Figure 7.19(b) adds feedback and another inverter to produce a noninverting static flip-flop. The PowerPC 603 microprocessor datapath used this flip-flop design without the input inverter or $\bar{Q}$ output [Gerosa94].

Flip-flops usually take a single clock signal $\phi$ and locally generate its complement $\bar{\phi}$. If the clock rise/fall time is very slow, it is possible that both the clock and its complement will simultaneously be at intermediate voltages, making both latches transparent and increasing the flip-flop hold time. In ASIC standard cell libraries (such as the Artisan
library), the clock is both complemented and buffered in the flip-flop cell to sharpen up the edge rates at the expense of more inverters and clock loading.

Recall that the flip-flop also has a potential internal race condition between the two latches. This race can be exacerbated by skew between the clock and its complement caused by the delay of the inverter. Figure 7.20(a) redraws Figure 7.19(a) with a built-in clock inverter. When \( \phi \) falls, both the clock and its complement are momentarily low as shown in Figure 7.20(b), turning on the clocked pMOS transistors in both transmission gates. If the skew (i.e., inverter delay) is too large, the data can sneak through both latches on the falling clock edge, leading to incorrect operation. Figure 7.20(c) shows a C MOS dynamic flip-flop built using C MOS latches rather than inverters and transmission gates [Suzuki73]. Because each stage inverts, data passes through the nMOS stack of one latch and the pMOS of the other, so skew that turns on both clocked pMOS transistors is not a hazard. However, the flip-flop is still susceptible to failure from very slow edge rates that turn both transistors partially ON. The same skew advantages apply even when an even number of inverting logic stages are placed between the latches; this technique is sometimes called NO RAce (NORA) [Gonclaves83]. In practice, most flip-flop designs carefully control the delay of the clock inverter so the transmission gate design is safe and slightly faster than C MOS [Chao89].

All of these flip-flop designs still present potential min-delay problems between flip-flops, especially when there is little or no logic between flops and the clock skew is large or poorly analyzed. For VLSI class projects where careful clock skew analysis is too much work and performance is less important, a reasonable alternative is to use a pair of two-phase nonoverlapping clocks instead of the clock and its complement, as shown in Figure 7.21. The flip-flop captures its input on the rising edge of \( \phi_1 \). By making the nonoverlap large enough, the circuit will work despite large skews. However, the nonoverlap time is
7.3 CIRCUIT DESIGN OF LATCHES AND FLIP-FLOPS

not used by logic, so it directly increases the setup time and sequencing overhead of the flip-flop (see Exercise 7.8).

7.3.3 Pulsed Latches

A pulsed latch can be built from a conventional CMOS transparent latch driven by a brief clock pulse. Figure 7.22(a) shows a simple pulse generator, sometimes called a clock chopper or one-shot [Harris01].

The Naffziger pulsed latch used on the Itanium 2 processor consists of the latch from Figure 7.17(j) driven by even shorter pulses produced by the generator of Figure 7.22(b) [Naffziger02]. This pulse generator uses a fairly slow (weak) inverter to produce a pulse with a nominal width of about one-sixth of the cycle (125 ps for 1.2 GHz operation). When disabled, the internal node of the pulse generator floats high momentarily, but no keeper is required because the duration is short. Of course, the enable signal has setup and hold requirements around the rising edge of the clock, as shown in Figure 7.22(c).

Figure 7.22(d) shows yet another pulse generator used on an NEC RISC processor [Kozu96] to produce substantially longer pulses. It includes a built-in dynamic transmission-gate latch to prevent the enable from glitching during the pulse.

Many designers consider short pulses risky. The pulse generator should be carefully simulated across process corners and possible RC loads to ensure the pulse is not degraded too badly by process variation or routing. However, the Itanium 2 team found that the pulses could be used just as regular clocks as long as the pulse generator had adequate drive.

The Partovi pulsed latch in Figure 7.23(a) eliminates the need to distribute the pulse by building the pulse generator into the latch itself [Partovi96, Draper97]. The weak cross-coupled inverters in the dashed box staticize the circuit, although the latch is susceptible to back-driven output noise on $Q$ or $\overline{Q}$ unless an extra inverter is used to buffer the output. The Partovi pulsed latch was used on the AMD K6 and Athlon [Golden99], but is slightly slower than a simple latch [Naffziger02]. It was originally called an Edge Triggered Latch (ETL), but strictly speaking is a pulsed latch because it has a brief window of transparency.
Most practical sequencing elements require a reset signal to enter a known initial state on startup. Figure 7.24 shows latches and flip-flops with reset inputs. There are two types of reset: synchronous and asynchronous. Asynchronous reset forces $Q_{\text{low}}$ immediately, while synchronous reset waits for the clock. Synchronous reset signals must be stable for a setup and hold time around the clock edge while asynchronous reset is characterized by a propagation delay from reset to output. Synchronous reset simply requires ANDing the input $D$ with reset. Asynchronous reset requires gating both the data and the feedback to force the

### 7.3.4 Resettable Latches and Flip-flops

Most practical sequencing elements require a reset signal to enter a known initial state on startup. Figure 7.24 shows latches and flip-flops with reset inputs. There are two types of reset: synchronous and asynchronous. Asynchronous reset forces $Q_{\text{low}}$ immediately, while synchronous reset waits for the clock. Synchronous reset signals must be stable for a setup and hold time around the clock edge while asynchronous reset is characterized by a propagation delay from reset to output. Synchronous reset simply requires ANDing the input $D$ with reset. Asynchronous reset requires gating both the data and the feedback to force the
reset independent of the clock. The tristate NAND gate can be constructed from a NAND gate in series with a clocked transmission gate.

Settable latches and flip-flops force the output high instead of low. They are similar to resettable elements but replace NAND with NOR and reset with set. Figure 7.25 shows a flip-flop combining both asynchronous set and reset.

**FIG 7.24** Resettable latches and flip-flops

**FIG 7.25** Flip-flop with asynchronous set and reset
7.3.5 **Enabled Latches and Flip-flops**
Sequencing elements also often accept an enable input. When enable $en$ is low, the element retains its state independently of the clock. The enable can be performed with an input multiplexer or clock gating, as shown in Figure 7.26. The input multiplexer feeds back the old state when the element is disabled. The multiplexer adds area and delay. Clock gating does not affect delay from the data input and the AND gate can be shared among multiple clocked elements. Moreover, it significantly reduces power consumption because the clock on the disabled element does not toggle. However, the AND gate delays the clock, potentially introducing clock skew. Section 12.5.5 addresses techniques to minimize the skew by building the AND gate into the final buffer of the clock distribution network. $en$ must be stable while the clock is high to prevent glitches on the clock, as will be discussed further in Section 7.4.3.

![Diagram of enabled latches and flip-flops](image)

7.3.6 **Incorporating Logic into Latches**
Another way to reduce the sequencing overhead of latches is to replace some of the inverters in the latch with gates that perform useful computation. Figure 7.27 shows two ways to do this in dynamic latches. The DEC Alpha 21164 used an assortment of latches built from a clocked transmission gate preceded and followed by inverting static CMOS gates.
such as NANDs, NORs, or inverters [Bowhill95]. This provides the low overhead of the
transmission gate latch while preserving the buffered inputs and outputs. The max-latch
consists of two transmission gates in parallel controlled by clocks gated with the corre-
sponding select signals. It integrates the multiplexer function with no extra delay from the
$D$ inputs to the $Q$ outputs except the small amount of extra diffusion capacitance on the
state node. Note that the setup time on the select inputs is relatively high. The clock gat-
ing will introduce skew unless the clocking methodology systematically plans to gate all
clocks. The same principles extend to static latches and flip-flops.

The Klass semidynamic flip-flop (SDFF) [Klass99] shown in Figure 7.28 is a cross between
a pulsed latch and a flip-flop. Like the Partovi pulsed latch, it operates on the principle of
intersecting pulses. However, it uses a dynamic NAND gate in place of the static NAND.
While the clock is low, $X$ precharges high and $Q$ holds its old state. When the clock rises,
the dynamic NAND evaluates. If $D$ is ’0,’ $X$ remains high and the top nMOS transistor
turns OFF. If $D$ is ’1’ and $X$ starts to fall low, the transistor remains ON to finish the tran-
sition. This allows for a very short pulse and short hold time. The weak cross-coupled
inverters staticize the flip-flop and the final inverter buffers the output node.

Like a pulsed latch, the SDFF accepts rising inputs slightly after the rising clock
edge. Like a flip-flop, falling inputs must set up before the rising clock edge. It is called
semidynamic because it combines the dynamic input stage with static operation. The
SDFF is slightly faster than the Partovi pulsed latch but loses the skew tolerance and time
borrowing capability. The Sun UltraSparc III built logic into the SDFF very efficiently by
replacing the single transistor connected to $D$ with a collection of transistors performing
the OR or multiplexer functions [Heald00].
Differential flip-flops accept true and complementary inputs and produce true and complementary outputs. They are built from a clocked sense amplifier so they can rapidly respond to small differential input voltages. While they are larger than an ordinary single-ended flip-flop—having an extra inverter to produce the complementary output—they work well with low-swing inputs such as register file bitlines (Section 11.2.3) and low-swing busses (Section 4.6.7).

Figure 7.29(a) shows a differential sense-amplifier flip-flop (SA-F/F) receiving differential inputs and producing a differential output [Matsui94]. When the clock is low, the internal nodes $X$ and $\bar{X}$ precharge. When the clock rises, one of the two nodes is pulled down, while the cross-coupled pMOS transistors act as a keeper for the other node. The SR latch formed by the cross-coupled NAND gates behaves as a slave stage, capturing the output and holding it through precharge. The flip-flop can amplify and respond to small differential input voltages, or it can use an inverter to derive the complementary input from $D$. This flip-flop was used in the Alpha 21264 [Gronowski98]. It has a small clock load and avoids the need for an inverted clock. If the two input transistors are replaced by true and complementary nMOS logic networks, the SA-F/F can also perform logic functions at the expense of greater setup time [Klass99].

The original SA-F/F suffers from the possibility that one of the internal nodes will float low if the inputs switch while the clock is high. The StrongArm 110 processor [Montanaro96] adds the weak nMOS transistor shown in Figure 7.29(a) to fully staticize the flip-flop at the expense of a small amount more internal loading and delay.

Although the sense amplifier stage is fast, the propagation delay through the two cross-coupled NAND gates hurts performance. The NAND gates serve as a slave SR latch and are only necessary to convert the monotonically falling pulsed $X$ signals to static $Q$ outputs; they can be replaced by HI-skew inverters when $Q$ drives domino gates. Alternatively, Figure 7.29(b) shows how to build a faster slave latch to replace the cross-coupled NANDs at the expense of eight more transistors [Nikoli00]. When the $X$ signal falls, it turns on the pMOS transistor to immediately pull the $Q$ output high. It also drives one of the inverters high, which pulls down the $\bar{Q}$ output through the opposite nMOS transistor.
The four blue transistors serve as small cross-coupled tristate keepers that hold the outputs after the master stage precharges, but turn off to avoid contention when the outputs need to switch. This slave latch still involves two gate delays for the falling output, but the delays are faster because the gates avoid contention, have lower logical effort, and are skewed to favor the critical edges.

The AMD K6 used another differential flip-flop shown in Figure 7.29(c) at the interface from static to self-resetting domino logic [Draper97]. The master stage consists of a self-resetting dual-rail domino gate. Assume the internal nodes are initially precharged. On the rising edge of the clock, one of the two will pull down and drive the corresponding output high. The OR gate detects this and produces a done signal that precharges the internal nodes and resets the outputs. Therefore, the flip-flop produces pulsed outputs primarily suitable for use in subsequent self-resetting domino gates (see Section 7.5.2.4). The cross-coupled pMOS transistors improve the noise immunity while the cross-coupled inverters staticize the internal nodes.

![Differential flip-flops](image)

**Figure 7.29** Differential flip-flops
CHAPTER 7  SEQUENTIAL CIRCUIT DESIGN

214

7.3.9  True Single-phase-clock (TSPC) Latches and Flip-flops

Conventional latches require both true and complementary clock signals. In modern CMOS systems, the complement is normally generated locally with an inverter in the latch cell. In the late 1980s, some researchers worked to avoid the complementary signal. The True Single-Phase-Clock (TSPC) latches and flip-flops replace the inverter-transmission gate or C′MOS stage with a pair of stages requiring only the clock, not its complement [Ji-Ren87, Yuan89]. Figure 7.30(a and b) show active high and low TSPC dynamic latches. Figure 7.30(c) shows a TSPC dynamic flip-flop. Note that this flip-flop produces a momentary glitch on \( Q \) after the rising clock edge when \( D \) is low for multiple cycles; this increases the activity factor of downstream circuits and costs power. [Afghahi90] extends the TSPC principle to handle domino, RAMs, and other precharged circuits.

![TSPC latches and flip-flops](image)

The dynamic TSPC latches were used on the groundbreaking Alpha 21064 microprocessor [Dobberpuhl92]. Logic can be built into the first stage of each latch. The latch is not easy to staticize [Afghahi90]. In any case, the clock must also be reasonably sharp to prevent races when both transistors are partially ON [Larsson94]. The Alpha 21164 reverted to conventional dynamic latches for an estimated 10% speed improvement [Bowhill95]. In summary, TSPC is primarily of historic interest.

7.4 Static Sequencing Element Methodology

This section examines a number of issues designers must address when selecting a sequencing element methodology. We begin with general issues, and then proceed to techniques specific to flip-flops, pulsed latches, and transparent latches.

Until the 0.5 or 0.35 \( \mu m \) generation, leakage was relatively low and thus dynamic latches held their state for acceptably long times. The DEC Alpha 21164 was one of the last major microprocessors to use a dynamic latching methodology in a 0.35 \( \mu m \) process in the mid-1990s. It required a minimum operating frequency of 1/10th full speed to retain state, even during testing. Modern systems generally require static sequencing elements to hold state when clocks are gated or the system is tested at a moderate frequency. Leakage
is usually worst during burn-in testing at elevated temperature and voltage, where the chip must still function correctly to ensure good toggle coverage. Static elements are larger and somewhat slower than their dynamic counterparts.

Similarly, the growing difficulty and cost of debugging and testing has forced engineers to build design-for-test (DFT) features into the sequencing elements. The most important feature is scan, a special mode in which the latches or flip-flops can be chained together into a large shift register so that they can be read and written under external control during testing. This technique is discussed further in Section 9. Scan has become particularly important because chips have so many metal layers that most internal signals cannot be directly reached with probes. Moreover, some flip-chips are mounted upside down, making physical access even more difficult. Scan can dramatically decrease the time required to debug a chip and reduce the cost of testing, so most design methodologies dictate that all sequencing elements must be scannable despite the extra area this entails. The Alpha 21264 did not support full scan and was very difficult to debug, leading to a later-than-desired release.

Clock distribution is another key challenge. As we will see in Section 12.5, it is very difficult to distribute a single clock across a large die in a fashion that gets it to all sequencing elements at nearly the same time. Controlling the clock skew on more than one clock is even more difficult, so almost all modern designs distribute a single high-speed clock. Other signals such as complementary clocks, pulses, and delayed clocks are generated locally where they are needed. The clock edge rates must be relatively sharp to avoid races in which both the master and slave latches are partially on simultaneously. The global clock may have slow edge rates after propagating along long wires, so it is typically buffered locally (either in each sequencing element or in a buffer cell serving a bank of elements) to sharpen the edge rates. Clock power, from the clock distribution network and the clocked loads, typically accounts for one third to one half of the total chip power consumption. Therefore, clocks are often gated with an AND gate in place of the local clock buffer to turn off the sequencing elements for inactive units of the chip.

All bistable elements are subject to soft errors from alpha particles or cosmic rays striking the circuits and injecting charge onto sensitive nodes (see Section 4.8.7). Sequencing elements require relatively high capacitance on the state node to achieve low soft error rates. This can set a lower bound on the minimum transistor sizes on that node.

### 7.4.1 Choice of Elements

Flip-flops, pulsed latches, and transparent latches offer tradeoffs in sequencing overhead, skew tolerance, and simplicity.

#### 7.4.1.1 Flip-flops

As we have seen, flip-flops have fairly high sequencing overhead but are popular because they are so simple. Nearly all engineers understand how flip-flops work. Some synthesis tools and timing analyzers handle flip-flops much more gracefully than transparent latches. Many ASIC methodologies use flip-flops exclusively for pipelines and state machines. If performance requirements are not near the cutting edge of a process, flip-flops are clearly the right choice in today’s CAD flows.
7.4.1.2 Pulsed Latches

Pulsed latches are faster than flip-flops and offer some time-borrowing capability at the expense of greater hold times. They have fewer clocked transistors and hence lower power consumption. If intentional time borrowing is not necessary, you can model a pulsed latch as a flip-flop triggered on the rising edge of the pulse with a lower delay but a lengthy hold time. This makes pulsed latches relatively easy to integrate into flip-flop-based CAD flows. Moreover, the pulsed latches still offer opportunistic time borrowing to compensate for modeling inaccuracies even if the intentional time borrowing is not used.

The long hold times make pulsed latches unsuitable for use in pipelines with no logic between pipeline stages. One solution is to use ordinary flip-flops in place of the pulsed latches in these circumstances where speed is not important. Unfortunately, some pulsed latches fan out to multiple paths, some of which are short and others long. The Itanium 2 processor used the clocked deracer in conjunction with Naffziger pulsed latches, as shown in Figure 7.31 [Naffziger02]. These were placed before the receiving latches on short paths and block incoming paths while the receiving latch is transparent. They automatically adapt to pulse with variation and hence have a shorter nominal propagation delay than buffers, but also consume more power than buffers because of the clock loading [Rusu03].

7.4.1.3 Transparent Latches

Transparent latches also have lower sequencing overhead than flip-flops and are attractive because they permit nearly half a cycle of time borrowing. One latch must be placed in each half-cycle. Data can arrive at the latch any time the latch is transparent. A convenient design approach is to nominally place the latch at the beginning of each half-cycle. Data can arrive at the latch any time the latch is transparent. Then time borrowing occurs when the logic in one half-cycle is longer than nominal and data does not arrive at the next latch until some time into the next half-cycle.

Figure 7.32 illustrates pipeline timing for short and long logic paths between latches. When the path is short (a), the data arrives at the second latch early and is delayed until the rising edge of $\phi_2$. Therefore, it is natural to consider latches residing at the beginning of their half-cycle because short paths automatically adjust to operate this way. When the path is longer (b), it borrows time from the first half-cycle into the second. Notice how
clock skew does not slow long paths because the data does not arrive at the latch until after the latest skewed rising edge.

Logic blocks involving multiple signals must ensure that each signal path passes through two latches in each cycle. Signals can be classified as phase 1 or phase 2 and logic gates must receive all their inputs from the same phase. Section 7.4.3 develops a formal notation of timing types to track when signals are safe to use.

All of the techniques mentioned in Section 6.5 can be used in common to create low-power libraries for sequential elements. However, because flip-flops and latches are normally clocked every cycle, both the core latch design and the clock distribution network must be carefully scrutinized to achieve low dynamic power. Keep device sizes small inside the core latch and minimize the number of clocked transistors. Pulsed latches are attractive because they have fewer clocked transistors than two-phase latches or flip-flops. The conventional flip-flop of Figure 7.19(b) is also power-efficient because it is simple. [Stojanovic99] presents an extensive study of power and delay in sequential elements.

Scan latches and flip-flops (see Chapter 9) used for testing increase the internal sequential element switched load. Unfortunately, there is a tradeoff between testability and power consumption. The decision is normally toward testability.

Clock gating can be effectively used to turn off sections of circuitry that are not required during certain time intervals. When clock gaters are inserted, the relative delay between blocks must be carefully monitored to ensure that no clock races occur (see Section 12.5.5).
As discussed in Section 7.2.3, latches with two-phase nonoverlapping clocks ($\phi_1$ and $\phi_2$) are attractive for class projects because with an adequately long clock period and sufficiently great nonoverlap, they are guaranteed to be safe from both setup and hold problems as long as they are used correctly. Logic must be divided into phases 1 and 2. Signals can only interact with other signals in the same phase. Passing through a latch changes the phase of the signal. The situation becomes slightly more complicated when gated clocks

### Example

Figure 7.33 shows a simplified block diagram of an IEEE 802.11a Wireless Local Area Network receiver [Ryan01]. Being a packet-based system, the clocking takes advantage of the fact that the clocks only have to be applied to modules when a packet arrives. The packet activity is monitored by the packet detector, which is carefully designed to dissipate the least dynamic power as it is “listening” constantly. The rest of the circuitry is usually idle and all clocks to the remaining modules are turned off. When a packet is detected, a control block issues a start of packet signal to the clock gater that controls the synchronization module (Synch). The control block also figures the length of the packet and thus, in conjunction with the pipeline delay through modules, knows how long to turn the local clock to each module in the data pipe. The Fast Fourier Transform (FFT) and Viterbi Decoder modules are similarly controlled. The Viterbi Decoder uses a register exchange technique, which has the ramifications that it has a large number of registers (128 • 64 bits). During clocking each register clocks constantly and so the decoder dissipates a significant portion of the dynamic power in the signal processing chain. Fortunately, the decoder contains very little logic between registers. Thus, it can be operated on a lower supply voltage to reduce dynamic power.

![Block Diagram](image)

**FIG 7.33** Wireless system with clock gating and variable power supply

### 7.4.3 Two-phase Timing Types

As discussed in Section 7.2.3, latches with two-phase nonoverlapping clocks ($\phi_1$ and $\phi_2$) are attractive for class projects because with an adequately long clock period and sufficiently great nonoverlap, they are guaranteed to be safe from both setup and hold problems as long as they are used correctly. Logic must be divided into phases 1 and 2. Signals can only interact with other signals in the same phase. Passing through a latch changes the phase of the signal. The situation becomes slightly more complicated when gated clocks...
and domino circuits are mixed with the latches. [Noice83] describes a method of timing types that can be appended to signal names to keep track of which signals can be safely combined at inputs to gates and latches.

In the two-phase timing discipline, a signal can belong to either phase 1 or phase 2 and be of one of three classes: stable, valid, or qualified clock. A signal is said to be stable during phase 1 (\(s_1\)) if it settles to a value before \(\phi_1\) rises and remains constant until after \(\phi_1\) falls. It is said to be valid during phase 1 (\(v_1\)) if it settles to a value before \(\phi_1\) falls and remains at that value until after \(\phi_1\) falls. It is said to be a phase 1 gated or qualified clock (\(q_1\)) if it either rises and falls like \(\phi_1\) or remains low for the entire cycle. By definition, \(q_1\) is a \(g_1\) signal. Phase 2 signals are analogous. Figure 7.34 illustrates the timing of each of these types.

---

**Figure 7.34** Timing types

Latches must take qualified clocks (either \(g_1\) or \(g_2\) signals) to their clock inputs. A phase 1 latch requires a \(s_1\) or \(v_1\) input (so that the input satisfies setup and hold times around the falling edge of \(\phi_1\)) and produces a \(s_2\) output because the output settles while
φ₁ is high (before φ₂ rises) and does not change again until the next time φ₁ is high (after φ₂ falls). A phase 2 latch requires a _s2 or _v2 input and produces a _s1 output. Qualified clocks are formed as the AND of a clock phase or another qualified clock with a stable signal belonging to the same phase. The qualifying signal must be stable to ensure there are no glitches in the clock. Qualified clocks are only used at the clock terminals of latches or dynamic logic. A block of static CMOS combinational logic requires that all inputs belong to the same phase. If all inputs are stable, the output is also stable. If any are valid, the output is valid. The phase of a domino gate is defined by the clock or qualified clock driving its evaluation transistor. The precharge transistor accepts the complement of the other phase. The inputs must be stable or valid during the evaluation phase and the output is valid during that phase because it settles before the end of the phase and does not change until precharge at the beginning of the next phase. All of these rules are illustrated in Figure 7.35. The definitions are based on the assumption that the propagation delays are short compared to the cycle time so that no time borrowing takes place; however, the connections continue to be safe even if time borrowing does occur.

Figure 7.38(a) redraws the flip-flop of Figure 7.21 built from master and slave latches using two-phase nonoverlapping clocking. The flip-flop changes its output on the rising edge of φ₁. Both input and output are _s2 signals. Figure 7.38(b) shows an enabled version of the flip-flop using clock gating. The enable signal to the slave must be _s1 to prevent glitches on the qualified clock; in other words, the enable must not change while φ₁ is high. If the system is built primarily from flip-flops with _s2 outputs, the enable must be delayed through a phase 2 latch to become _s1. Alternatively, the master (φ₂) latch could be enabled, but this requires that the enable set up half a cycle earlier.
Example

Annotate each of the signals in Figure 7.36 with its timing type. If the circuit contains any illegal connections, identify the problems and explain why the connections could cause malfunctions.

Solution: Figure 7.37 shows the timing types of each signal. $t_{??}$ is the OR of $h_s1$ and $r_s2$. Hence, it might change after the rising edge of $\phi_2$ or $\phi_1$. Excessive clock skew on $\phi_2$ could cause a hold time violation, affecting the result seen at $u_s1$. Skew between these clocks could cause hold time violation.

Annotated circuit showing timing types
Even when conventional two-phase latches with 50% duty cycles are used, the timing types are still convenient to track which signals can interact. Typically, one distributes a single 50% duty cycle clock $\phi_1$ and locally generates its complement $\phi_2$. In such a case, $\phi_1$ plays the role of $\phi_1$ and $\phi_2$ while $\phi_2$ plays the role of $\phi_1$ and $\phi_2$. This means that both the precharge and evaluate transistors of dynamic gates receive the same signal. Because there is no nonoverlap, you must analyze each path to ensure no hold problems exist. In particular, be careful to guarantee a stable enable signal for gated clocks.

Previous sections have derived sequencing element performance in terms of the setup and hold times and propagation and contamination delays. These delays are interrelated and are used for budgeting purposes. For example, a flip-flop might still capture its input properly if the data changes slightly less than a setup time before the clock edge. However, the clock-to-$Q$ delay might be quite long in this situation. If we call $t_{DC}$ the time that the data actually sets up before the clock edge and $t_{CQ}$ the actual delay from clock to $Q$, we could define $t_{setup}$ as the smallest value of $t_{DC}$ such that $t_{CQ} \leq t_{pq}$. Moreover, we could choose $t_{pq}$ to minimize the sequencing overhead $t_{setup} + t_{pq}$. In this section we will explore how to characterize these delays through simulation.

![Figure 7.38](image)

**Figure 7.38** Flip-flops using two-phase nonoverlapping clocks

Even when conventional two-phase latches with 50% duty cycles are used, the timing types are still convenient to track which signals can interact. Typically, one distributes a single 50% duty cycle clock $\phi_1$ and locally generates its complement $\phi_2$. In such a case, $\phi_1$ plays the role of $\phi_1$ and $\phi_2$ while $\phi_2$ plays the role of $\phi_1$ and $\phi_2$. This means that both the precharge and evaluate transistors of dynamic gates receive the same signal. Because there is no nonoverlap, you must analyze each path to ensure no hold problems exist. In particular, be careful to guarantee a stable enable signal for gated clocks.

### 7.4.4 Characterizing Sequencing Element Delays

Previous sections have derived sequencing element performance in terms of the setup and hold times and propagation and contamination delays. These delays are interrelated and are used for budgeting purposes. For example, a flip-flop might still capture its input properly if the data changes slightly less than a setup time before the clock edge. However, the clock-to-$Q$ delay might be quite long in this situation. If we call $t_{DC}$ the time that the data actually sets up before the clock edge and $t_{CQ}$ the actual delay from clock to $Q$, we could define $t_{setup}$ as the smallest value of $t_{DC}$ such that $t_{CQ} \leq t_{pq}$. Moreover, we could choose $t_{pq}$ to minimize the sequencing overhead $t_{setup} + t_{pq}$. In this section we will explore how to characterize these delays through simulation.

Figure 7.39 shows the timing of a conventional static edge-triggered flip-flop from Figure 7.19(b). Delays are normalized to a 75 ps FO4 inverter. The actual $\phi_1$-to-$Q$ ($t_{CQ}$) and $D$-to-$Q$ ($t_{DQ}$) delays for a rising input are plotted against the $D$-to-$\phi_1$ ($t_{DC}$) delay, i.e., how long the data arrived before the clock rises. If the data arrives long before the clock, $t_{CQ}$ is short and essentially independent of $t_{DC}$ delay. $t_{DQ} = t_{DC} + t_{pq}$ so it increases linearly as data arrives earlier because the data is blocked and waits for the clock before proceeding. As the data arrives closer to the clock, $t_{CQ}$ begins to rise. However, $t_{DQ}$ initially decreases and reaches a minimum when $t_{pq}$ has a slope of $-1$ (note the axes are not to scale). Therefore, let us define the setup time $t_{setup}$ as $t_{DC}$ at which this minimum $t_{DQ}$ occurs and the propagation delay $t_{pq}$ as $t_{CQ}$ at this time. The contamination delay $t_{pq}$ is the minimum $t_{CQ}$ that occurs when the input arrives early.
In general, the delays can differ for inputs of '0' and '1.' Figure 7.40 plots $t_{CQ}$ vs. $t_{DC}$ for the four combinations of rising and falling $D$ and $Q$. The setup times $\Delta_{DC0}$ and $\Delta_{DC1}$ are the times that $D$ must fall or rise, respectively, before the clock so that the data is properly captured with the least possible $t_{DQ}$. Observe that this flip-flop has a longer setup time but shorter propagation delay for low inputs than high inputs. The hold times $t_{hold0}$ and $t_{hold1}$ are the times that $D$ must rise or fall, respectively, after the clock so that the old value of '0' or '1' is captured instead of the new value. Observe that the hold times are typically negative. The contamination delay $t_{ccq0/1}$ again is the lowest possible $t_{CQ}$ and occurs when the input changes well before the clock edge. When only one delay is quoted for a flip-flop timing parameter, it is customarily the worst of the '0' and '1' delays.

The aperture width $t_a$ is the width of the window around the clock edge during which the data must not transition if the flip-flop is to produce the correct output with a propagation delay less than $t_{pdc}$. The aperture times for rising and falling inputs are

$$t_{ar} = t_{amp1} + t_{hold1}$$

$$t_{af} = t_{amp0} + t_{hold0} \quad (7.20)$$

If the data transitions within the aperture, $Q$ can become metastable and take an unbounded amount of time to settle. Metastability is discussed further in Section 7.6.1.

If $D$ is a very short pulse, the flip-flop may fail to capture it even if $D$ is stable during the setup and hold times around the rising clock edge. Similarly, if the clock pulse is too short, the flip-flop may fail to capture stable data. Well-characterized libraries sometimes specify minimum pulse widths for the clock and/or data as well as setup and hold times.
Level-sensitive latches have somewhat different timing constraints because of their transparency, as shown in Figure 7.41 for a conventional static latch from Figure 7.17(g) using a pulse width of 4 FO4 inverter delays. As with an edge-triggered flip-flop, if the data arrives before the clock rises (\(t_{DCr} > 0\)), it must wait for the clock. In this region the clock-to-\(Q\) delay is nearly constant and \(t_{DCr}\) increases as the data arrives earlier. If the data arrives after the clock rises while the latch is transparent, \(t_{DCr}\) is essentially independent of the arrival time. The data must set up before the falling edge of the clock. The second set of labels on the X-axis indicates the D-to-clk fall time \(t_{DCF}\). As the data arrives too close to the falling edge, \(t_{DCr}\) increases. Now to achieve low \(t_{DCr}\), we choose the setup time before the knee of the curve, e.g., 5% greater than its minimum value. The setup time is measured relative to the falling edge of the clock. If the data changes less than a hold time after the falling edge of the clock, \(Q\) may momentarily glitch. Thus, the hold time \(t_{hold}\) for a latch is defined to be \(-t_{DCF}\) for which \(Q\) displays a negligible glitch.

Pulsed latches have setup and hold times measured around the falling edge of the clock. However, designers often wish to treat pulsed latches as edge-triggered flip-flops from the perspective of timing analysis. Therefore, we can define "virtual" setup and hold times relative to the rising clock edge [Stojanovic99]. For example, the pulsed latch in
Figure 7.41 has $t_{\text{setup-virtual}} = t_{\text{setup}} = -2.4$ FO4 but $t_{\text{pq-virtual}} = t_{\text{pq}} + (t_{\text{pw}} - t_{\text{setup}}) = 4.06$ FO4 so the total sequencing overhead of $t_{\text{del}} = t_{\text{setup-virtual}} + t_{\text{pq-virtual}}$ is unaffected by the change of reference or pulse width. The virtual hold time is now $t_{\text{hold-virtual}} = t_{\text{hold}} + t_{\text{pw}} = 2.6$ FO4, which is positive as one should expect because the input must hold long after the rising edge of the clock.

The delays vary with input slope, voltage, and temperature. The contamination delay should be measured in the environment where it is shortest while the setup and hold times and propagation delay should be measured in the environment where it is longest.

The designer can trade off setup time, hold time, and propagation delay. Figure 7.42 shows the effects of adding delay $t_{\text{buf}}$ to the clock, $D$, or $Q$ terminals of a flip-flop. Recall that the sequencing overhead depends on the sum of the setup time and propagation delay while the minimum delay between flip-flops depends on the hold time less the contamination delay. Adding delay on either the input or output eases min-delay at the expense of sequencing overhead. Many standard cell libraries intentionally use slow flip-flops so that logic designers do not have to worry about hold time violations. Adding delay on the clock simply shifts when the flop activates. The sequencing overhead does not change, but the system can accommodate more logic in the previous cycle and less in the next cycle. This is similar to time borrowing in latch-based systems, but must be done intentionally by adjusting the clock rather than opportunistically by taking advantage of transparency. Some authors refer to delaying the clock as *intentional clock skew*. This book reserves the term *clock skew* for uncertainty in the clock arrival times.
7.5 Sequencing Dynamic Circuits

Dynamic and domino circuits operate in two steps: precharge and evaluation. Ideally, the delay of a path should be the sum of the evaluation delays of each gate along the path. This requires some careful sequencing to hide the precharge time. Traditional domino circuits discussed in Section 7.5.1 divide the cycle into two half-cycles. One phase evaluates while the other precharges, then the other evaluates while the first precharges. Transparent latches hold the result of each phase while it precharges. This scheme hides the precharge time but introduces substantial sequencing overhead because of the latch delays and setup time. A variety of skew-tolerant domino circuit schemes described in Section 7.5.2 use overlapping clocks to eliminate the latches and the sequencing overhead. Section 7.5.3 expands on skew-tolerant domino clocking for unfooted dynamic gates.

Recall that dynamic gates require that inputs be monotonically rising during evaluation. They produce monotonically falling outputs. Domino gates consist of dynamic gates followed by inverting static gates to produce monotonically rising outputs. Because of these two levels of inversion, domino gates can only compute noninverting logic functions. We have seen that dual-rail domino gets around this problem by accepting both true and complementary inputs and producing both true and complementary outputs. Dual-rail domino is not always practical. For example, dynamic logic is very efficient for building wide NOR structures because the logical effort is independent of the number of inputs. However, the complementary structure is a tall NAND, which is quite inefficient. When inverting functions are required, an alternative is to use a dynamic gate that produces monotonically falling outputs, but delays the clock to the subsequent dynamic gate so that the inputs are stable by the time the gate enters evaluation. Section 7.5.4 explores a selection of these nonmonotonic techniques.
7.5.1 Traditional Domino Circuits

Figure 7.43(a) shows a traditional domino clocking scheme. While the clock is high, the first half-cycle evaluates and the second precharges. While the clock is low, the second evaluates and the first precharges. With this ping-pong approach, the precharge time does not appear in the critical path. The inverting latches hold the result of one half-cycle while that half-cycle precharges and the next evaluates. The data must arrive at the first half-cycle latch a setup time before the clock falls. It propagates through the latch, so the overhead of each latch is the maximum of its setup time and D-to-Q propagation delay [Harris01]. Assuming the propagation delay is longer, the time available for computation in each cycle is

\[ t_{pd} = T_c - 2t_{pdq} \]  

(7.21)
Figure 7.43(b) shows the pipeline with clock skew. Data is launched into the first dynamic gate of each cycle on the rising edge of the clock and must set up before the falling edge. Hence, clock skew cuts into the time available for computation in each half-cycle. This is even worse than flip-flops, which pay clock skew once per cycle. Assuming the skew and setup time are greater than the propagation delay, the time for computation becomes

\[ t_{pd} = T_c - 2t_{setup} - 2t_{skew} \]  

(7.22)

Moreover, like flip-flops, traditional domino circuits suffer from imbalanced logic. Gates cannot borrow time into the next half-cycle, so a fraction of a gate delay at the end of each half-cycle may be wasted. This penalty is hard to quantify, but clearly the ability to borrow time intentionally or opportunistically would help performance.

In summary, traditional domino circuits have high sequencing overhead from latch delay, clock skew, and imbalanced logic. For heavily pipelined systems with short cycle times, this overhead can be such a large fraction of the cycle time that it wipes out the performance advantage that domino was intended to bring. Therefore, many system designers have developed skew-tolerant domino sequencing techniques with lower overhead. The next section is devoted to these techniques.

### 7.5.2 Skew-tolerant Domino Circuits

Traditional domino circuits have such high sequencing overhead because they have a hard edge in each half-cycle: The first domino gate does not begin evaluating until the rising edge of the clock, but the result must set up at the latch before the falling edge of the clock. If we could remove the latch, we could soften the falling edge and cut the overhead.

The latch serves two functions: (1) to prevent nonmonotonic signals from entering the next domino gate while it evaluates and (2) to hold the results of the half-cycle while it precharges and the next half-cycle evaluates. Within domino pipelines, all the signals are monotonic, so the first function is unnecessary. Moreover, as long as the next half-cycle has sufficient time to evaluate using the results of the first half-cycle, the first half-cycle can precharge without impacting the output of the next.

Figure 7.44 illustrates the implications of eliminating the latch. In general, let logic be divided into \( N \) phases rather than two half-cycles. Figure 7.44(a) shows the last domino gate in phase 1 driving the first gate in phase 2. Figure 7.44(b) shows that the circuit fails if the clocks are nonoverlapping. When \( \phi_1 \) falls, nodes \( a \) and \( b \) precharge high and low, respectively. When \( \phi_2 \) rises, the input to the first domino gate in this phase has already fallen, so \( c \) will never discharge and the circuit loses information. Figure 7.44(c) shows that the second dynamic gate receives the correct information if the clocks overlap. Now \( \phi_1 \) rises while \( b \) still holds its correct value. Therefore, the first domino gate phase 2 can evaluate using the results of the first phase. When \( \phi_1 \) falls and \( b \) precharges low, \( c \) holds its value. Without a keeper, \( c \) can float either high or low. Figure 7.45 shows a full keeper consisting of weak cross-coupled inverters to hold the input either high or low. In summary,
the latches can be eliminated at phase boundaries as long as the clocks overlap and the first
dynamic gate of each phase uses a full keeper.

In general, as long as the clock overlap is long enough that the second phase can evaluate
before the first precharges, the latch between phases is unnecessary. Let us define $t_{\text{hold}}$
as the required overlap so that the second phase can evaluate before the first precharges. It
is typically a small negative number because the dynamic gate evaluates quickly, but pre-
charge is slower and must ripple through the static stage. The clocks must overlap enough
such that they still overlap by $t_{\text{hold}}$ even under worst-case clock skew. The sequencing
overhead is zero because data propagates from one domino gate to the next without wait-
ing at any sequencing elements. Therefore, we use the generic name skew-tolerant domino

---

Weste07v4.fm Page 229 Monday, January 5, 2004 2:30 AM

1Do not confuse this $t_{\text{hold}}$ with $t_{\text{hold}}$ on a sequencing element, the time that the data must remain stable after the clock edge.
for domino circuits with overlapping clocks that eliminate the latches between phases [Harris01]. Using more clock phases also helps spread the power consumption across the cycle rather than drawing large noisy current spikes on the two clock edges.

Skew-tolerant domino circuits can also borrow time from one phase into the next, as illustrated in Figure 7.46. Nominally each phase occupies half the cycle in this example. However, a $\phi_1$ dynamic gate can borrow time into phase 2 if that is convenient because both clocks are simultaneously high. If one phase overlaps the next by $t_{\text{overlap}}$ less any clock skew, the maximum time that gates in one phase can borrow into time nominally allocated for the next is

$$t_{\text{borrow}} = t_{\text{overlap}} - t_{\text{hold}} - t_{\text{skew}}$$ (7.23)

[Williams91] observed that self-timed pipelines could use overlapping clocks to eliminate latches, but such asynchronous design has not been widely adopted. The Alpha 21164 overlapped clocks in the ALU to eliminate the mid-cycle latch and improve performance [Bowhill95]. Since then, most high-performance synchronous systems using domino have employed some form of skew-tolerant domino to avoid the high sequencing overhead of traditional domino.

There are many ways to produce overlapping clocks. In general, you can use $N$ separate clock phases. Each phase can use 50% duty-cycle waveforms or can stretch the falling edge for even greater overlap. Generating multiple overlapping clocks with low skew is a challenge. The remainder of this section describes a number of methods that have been used successfully.

7.5.2.1 Two-phase Skew-tolerant Domino and OTB Domino Figure 7.47 shows a clock generator for the two-phase skew-tolerant domino system from Figure 7.46. The generator uses clock choppers that stretch the falling edge to provide the overlap. A potential problem with two-phase systems is that if a phase of logic has short contamination delay,
7.5.2.2 Four-phase Skew-tolerant Domino

Figure 7.49 shows a four-phase skew-tolerant domino system. Each of the phases has a 50% duty cycle and is spaced a quarter cycle after the previous one, so the nominal overlap is a quarter cycle. The clocks are never all simultaneously high so race problems are solved unless skew approaches a quarter cycle. According to EQ (7.23), the maximum time available for borrowing from one phase to the next is

\[ t_{\text{borrow}} = T_P / 4 - t_{\text{hold}} - t_{\text{skew}} \]  \hspace{1cm} (7.24)

Figure 7.50(a) shows a local clock generator producing the four phases. \( \phi_1 \) and \( \phi_2 \) are produced directly from the global clock and its complement. \( \phi_3 \) and \( \phi_4 \) are delayed by buffers with nominal quarter cycle latency. By using both clock edges, each phase is guaranteed to overlap the next phase independent of clock frequency. Variations in these buffer delays with
process, voltage, and temperature can reduce the overlap and available time for borrowing. To avoid excessive pessimism, remember that in the fast corner where overlaps are short, the rest of the gates are also faster. The clock generator also includes a built-in enable.

In general, clock choppers can be used to produce even greater overlap at the expense
of greater race concerns. The Itanium II uses four-phase skew-tolerant domino with duty cycles exceeding 50% [Naffziger02]. Figure 7.50(b) shows a four-phase clock generator with clock choppers to provide longer duty cycles. [Harris01] describes four-phase circuit methodology in much more detail, including testability and a generalization of timing types from Section 7.4.3.

7.5.2.3 \(N\)-phase Skew-tolerant Domino Another approach to domino clocking is to use a chain of buffers to produce a unique phase for each level of logic in a cycle. Figure 7.51 shows two ways of producing these phases. In Figure 7.51(a), half the phases are generated off the rising edge of the clock and half off the falling edge. In this way, each phase is guaranteed to overlap the next independent of cycle time. However, the phase length is sensitive to duty cycle error in the clock distribution network. In Figure 7.51(b), all of the phases are generated off the rising edge. If the clock period is long, the final phase must delay its falling edge to guarantee it will still overlap the first phase of the next cycle. The NAND gate ensures that the last phase, \(\phi_6\), will not rise until after \(clk\) falls (to avoid min-delay problems) and will not fall until after \(clk\) rises (to ensure overlap of \(\phi_1\)).

A number of design teams have independently developed these techniques. The approach of one phase for each level of logic has been called Delayed Reset (IBM [Nowka98]), Cascaded Reset (IBM [Silberman98]), and Delayed Clocking (Sun [Heald00]). The phase generator for Cascaded Reset domino is well suited to driving footless dynamic gates and will be discussed further in Section 7.5.3.

7.5.2.4 Self-resetting (Postcharge) Domino In the methods examined so far, the timing of the precharge operation has been controlled by the clock generator. An alternative approach, called Self-Resetting or Postcharge Domino, is to control the precharge based on the output of the domino gate. Figure 7.52 shows a simple self-resetting domino gate. When the domino gate evaluates and the output rises, a timing chain produces a precharge signal \(reset\) to precharge the dynamic stage (and possibly assist pulling the HI-skew inverter low, particularly if the inverter is highly skewed). Once the output has fallen, the precharge signal turns off the precharge transistors and the gate is ready to evaluate again. The input must have fallen before the gate reenters evaluation so the gate does not repeatedly pulse on a steady input. Therefore, self-resetting gates accept input pulses and produce output pulses whose duration of five gate delays is determined by the delay of the timing chain. As long as the first inverter in the timing chain is small compared to the rest of the load on node \(Y\), its extra loading has negligible impact on performance.

Self-resetting gates save power because they reduce the loading on the clock. Moreover, they only toggle the precharge signal when the gate evaluates low. In Section 11.2.2, we will see that this is particularly useful for RAM decoders. Only one of many word lines in a RAM will rise on each cycle, so a self-resetting decoder saves power by resetting only that line without applying precharge to the other word line drivers. For example, an IBM SRAM [Chappel91], the Intergraph Clipper cache [Heald93], and the Sun UltraSparc I cache [Heald98] use self-resetting gates\(^2\).

\(^2\)Confusingly referred to as "delayed reset" by Sun in [Lev95, Heald98].
Self-resetting AND gates in these decoders often receive the address inputs as static levels rather than pulses. **Predicated self-resetting AND gates** [Amrutur01] wait for the input to fall before precharging the output to stretch the pulse width and prevent multiple output pulses when the input is held high, as shown in Figure 7.53. The first inverter in the timing chain is replaced by a **generalized Muller C-element**, shown in blue, whose output does not rise until both the output and one of the inputs have fallen. This only works for functions such as AND or OR-AND where one of the inputs is in series with all of the others.
[Probsting91] applies self-resetting techniques to NORA gates for buffers and memory decoders. Figure 7.54 shows an example of a postcharged buffer for a memory chip. It rapidly amplifies the chip select signal CS and provides a series of pulses that serve as clocks for large (multi-pF) loads across the chip. The clock chopper produces a pulse to trigger the first stage of the buffer. The buffer consists of alternating extremely HI- and LO-skew inverters with logical efforts of approximately 2/3 and 1/3, respectively. Each inverter also receives a postcharge signal from a subsequent stage to assist the weak device in resetting the gate. The very small transistor serves as a keeper, so the gates can be viewed as unfooted NTP dynamic nMOS and pMOS inverters. Forward moving pulses trigger each gate. Signals from four stages ahead feed back to postcharge the gate. The buffer is roughly twice as fast as an ordinary chain of inverters because of the lower logical efforts. It also avoids the need for an external clock to precharge the dynamic gates.
IBM has developed an extensive methodology for self-resetting domino gates called SRCMOS [Haring96] that has been applied to circuits including a register file [Hwang99], 64-bit adder [Hwang99-2], and the S/390 G4 CPU cache [Webb97]. SRCMOS gates are typically unfooted dynamic gates followed by highly skewed static inverters, as shown in Figure 7.55. True and complementary reset signals precharge the dynamic stage and help pull the output low. An additional weak static evaluation transistor converts the gate into pseudo-nMOS when the global se signal is asserted to assist with testing and low-frequency debug. The inputs and outputs are pulses. The reset signals are generated from the gate outputs or from a global reset.

To avoid the overhead and timing constraints of reset circuitry on every gate, the reset signals can be derived from the output of the first gate in a pipeline and delayed through buffers to reset subsequent gates. Figure 7.56 shows an example of an SRCMOS macro adapted from [Hwang99-2]. The upper portion represents an abstract
Datapath. None of the keepers or static evaluation devices are shown. The center is a timing chain that provides reset pulses to each gate. These pulses may be viewed as $N$-phase skew-tolerant domino clocks. The bottom shows a pulse generator. In normal operation, the power-on reset signal is low and the static evaluation signal $se$ high. Assume that all of the gates have been precharged. When the input pulse arrives at $A$, the datapath will begin evaluating. The first stage must use dual-rail (or in general, 1-of-$N$ hot) encoding so that $Y_{1\_h}$ or $Y_{1\_l}$ will rise when the stage has completed. This triggers the pulse generator, which raises the done signal and initiates a reset. A wave of low-going reset pulses propagates along the timing chain to precharge each gate. One of the reset pulses also precharges the pulse generator, terminating the reset operation. At this point, the datapath can accept a new input pulse. If the data idles low, none of the nodes toggle and the circuit consumes no dynamic power.

![Datapath Diagram](image)

**Fig 7.56** SRCMOS Data Path
The power-on reset forces done and reset high to initialize the pipeline at startup. When the static evaluation signal is asserted, the reset pulses are inhibited. In this mode, the datapath gates behave as pseudo-nMOS rather than dynamic, permitting low-frequency test and debug.

Self-resetting gates require very careful design because they act on pulses rather than static levels. Some of the timing checks include [Narayanan96]:

- Pulse overlap constraints
  - Pulses arriving at series transistors must overlap so the dynamic gate can pull down through all the transistors
- Pulse width constraints
  - Pulses must be wide enough for a gate to evaluate
- Collision avoidance constraints
  - Pulses must not arrive at dynamic gates while the gates are being precharged

The Pentium 4 uses yet another form of self-resetting domino called Globally-Reset Domino with Self-Terminating Precharge (Global STP) to achieve very fast cycle times [Hinton01]. The first design operated at 2 GHz in a 180 nm process (< 16 FO4 inverter delays / cycle). More remarkably, the integer execution was double-pumped to 4 GHz using Global STP domino. Each cycle has time for only eight gate delays: four dynamic gates and four static gates.

Figure 7.57 illustrates the Global STP circuits. A frequency doubler generates pulses off both edges of the clock to drive the datapath. Each stage of the datapath is a domino gate with a keeper (k) and precharge transistor (p). The gates are shown using HI-skew inverters but could use any HI-skew inverting static gate. The small NAND gates save power by only turning on the precharge transistor if the dynamic gate had evaluated low. The first stage requires a foot to only sample the input while φ₁ is high. The last stage also uses a foot, a full keeper, and more complex reset circuitry to stretch the width of the output pulse so it is compatible with static logic. The reset timing chain must be carefully designed to produce precharge clocks properly aligned to the data. For example, φ₁ should be timed to rise close to the time Y₁ evaluates high to prevent contention between the precharge transistor and the pulldown network. Global STP circuit design can be a very labor-intensive process. IBM used a similar timing chain without the frequency doubler on an experimental 1 GHz PowerPC chip and called the method cascaded reset [Silberman98].

7.5.3 Unfooted Domino Gate Timing

Unfooted domino gates have a lower logical effort than footed gates because they eliminate the clocked evaluation transistor. They also reduce clock loading, which can save power. However, at least one input in each series stack must be OFF during precharge to prevent crowbar current flowing from $V_{DD}$ to GND through the precharge device and ON stack.
The easiest way to ensure this is to require that the input come from a previous domino gate that has completed precharge before the footless gate begins precharge. Moreover, the previous gate must not output a '1' again until the unfooted gate is in evaluation.

One way to ensure these constraints is to delay the falling edge of clocks to footless gates as shown in Figure 7.58(a). The first domino gate is footed to accept static inputs that might be high during precharge. The subsequent unfooted gates begin evaluating at the same time but have their precharge delayed until the previous gate has precharged. Multiple delayed clocks can be used to allow multiple stages of unfooted gates. For example, the Itanium II processor uses one footed gate followed by four unfooted gates in the first half-cycle of the execution stage for the 64-bit adder [Fetzer02]. If the falling edge is
delayed too much in a system with a short clock period, the clock may not be low long enough to fully precharge the gate. Figure 7.58(b) shows an OTB domino system that uses only one delayed clock but allows every other domino gate to be footless. The delayed clocks can be produced with clock choppers as was shown in Figure 7.47.

The precharge time on each of the delayed phases in Figure 7.58(a) becomes shorter because the falling edge is delayed but the rising edge is not. It is not strictly necessary for all the rising edges to coincide; some delay can be accepted so long as the delayed clock is in evaluation by the time the input arrives at its unfooted gate. Figure 7.59 shows a delayed precharge clock buffer [Colwell95] used on the Pentium II. The delayed clocks are produced with skewed buffers that have fast rising edges but slower falling edges.
Self-resetting domino also works well with unfooted gates. The inputs are pulses rather than levels. As long as the pulses are only high while the gate is in evaluation, no precharge contention will occur. For example, Figure 7.54, Figure 7.56, and Figure 7.57 illustrate self-resetting circuits with unfooted gates in some or all of the stages.

The consequence of precharging an unfooted gate before its input has fully fallen low is excess power consumption rather than outright circuit failure. Therefore, delays can be set to nominally avoid precharge contention, yet accept that, under worst-case clock skew, contention may occur in a few places.

7.5.4 Nonmonotonic Techniques

The monotonicity requirement forces domino gates to perform only noninverting functions. Dual-rail domino accepts true and complementary inputs and produces true and complementary outputs. This works reasonably well for circuits such as XORs at the expense of twice the hardware. However, domino is particularly poorly suited to wide NOR functions. Figure 7.60 compares a dual-rail domino 4-input OR/NOR gate to a 4-input dynamic NOR. The dual-rail design tends to be very slow because the complementary gate is a tall NAND with a logical effort of 5/3. On the other hand, a dynamic wide NOR is very compact and has a logical effort of only 2/3. The problem is exacerbated for wider gates.

The output of a dynamic gate is monotonically falling so it cannot directly drive another dynamic gate controlled by the same clock, as was shown in Figure 6.27. However, if the rising edge of the clock for the second gate is delayed until the first gate has fully evaluated, the second gate sees a stable input and will work correctly, as shown by Figure 7.61.

The primary tradeoff in such clock-blocked circuits is the amount of delay. If the delay is too short, the circuit will fail, but as the delay becomes longer, the circuit sacrifices the performance advantages that dynamic logic was supposed to provide. This challenge is exacerbated by process and environmental variations that require margins on the delay in the nominal case so that the circuit continues to operate correctly in the worst case.
Figure 7.61 also illustrates the *precharge race* problem. When $X$ precharges while $Y$ is still in evaluation, $Y$ will tend to start to fall. If $\phi_2$ falls too late, $Y$ will incorrectly glitch low. We can alleviate this problem by latching $Y$ before $X$ precharges or by delaying the falling edge of $\phi_1$.

This section addresses a number of nonmonotonic techniques using delayed clocks to directly cascade dynamic gates and examines the margins required for matched delays.

### 7.5.4.1 Delay Matching

Figure 7.62 shows a number of simple delay elements. The buffer delay can be set by adjusting gate widths. The buffer with transmission gates provides flexibility for longer delays. The current-starved inverter and switched capacitance designs use a reference voltage to adjust the delay externally. The digitally controlled current-starved inverter uses several digital signals rather than an analog voltage to adjust delay.
Section 4.7.6 showed that the delay of gates can vary by as much as 30% relative to an FO4 inverter across process, voltage, and temperature variations. Therefore, the delay line should provide some margin to guarantee it always is slower than the gate it must match. For example, [Yee00] uses a 20% margin. Many industrial designs use even more margin to ensure the circuit will have good yield in high-volume production. (Who wants to explain to the big boss why he or she wasted millions of dollars for the sake of saving a few picoseconds?) You should always make sure that the circuit works correctly in all process and environmental corners because it is not obvious which corner will cause the worst-case mismatches. Moreover, random device variations and inaccuracies in the parasitic extraction and device models cause further mismatch that cannot be captured through the design corner files. Yet another problem is that matching differs from one process to another, potentially requiring expensive redesign of circuits with matched delays when they are ported to the next process generation. Adjustable delay lines are attractive because the margin can be set more aggressively and increased after fabrication (as was done in [Vangal02]); however, generating and distributing a low-noise reference voltage can be challenging.

The key to good matching is to make the delay circuit behave like the gate it should match as much as possible. A good technique is to use a \textit{dummy gate} in the delay line, as shown in Figure 7.63 for a 2:1 dynamic multiplexer. The dummy gate replicates the gate being matched so that to first order, process and environmental variations will affect both identically. The input pattern is selected for worst-case delay.

You might be tempted to use longer-than-minimum length transistors to create long delays, but this is not good because transistor length variations will affect the delay circuit much differently than the gate it matches.

Despite all of these difficulties, delay matching has been used for decades in specialized circumstances that require wide NOR operation such as CAMs and PLAs (see Sections 11.6 and 11.7). [Yee00] proposes wider use of delay matching in datapath applications and names the practice \textit{Clock-Delayed (CD) Domino}.

\subsection{Clock-delayed Domino}

In the simplest CD Domino scheme, logic is \textit{levelized} as shown in Figure 7.64(a). The boxes represent domino gates annotated with their worst-case delay. Delay elements produce clocks tuned to the slowest gate in each level. The overall path delay is the sum of the delays of each element, which may be longer than the
actual critical path through logic. An alternative scheme is to clock each gate at a time matched to its latest input, as shown in Figure 7.64(b). This better matches the critical path at the expense of more delay elements and design effort. CD Domino is most effective for functions where high fan-in gates can be converted to wide dynamic NORs.

7.5.4.3 Race-based Nonmonotonic Logic The Itanium II processor uses a specialized nonmonotonic structure called an annihilation gate for high fan-in AND functions such as a 6-input decoder [Naffziger02]. An ordinary high fan-in AND gate requires many series transistors. Using DeMorgan’s Law, it can be converted to a wide NOR with complementary inputs. The annihilation gate in Figure 7.65 performs this NOR function very rapidly.
while generating a monotonically rising output suitable as an input to subsequent domino gates. It can be viewed as a dynamic NOR followed by a domino buffer with no clock delay. This introduces a race condition, but the two stages are carefully sized so the NOR will always win the race.

Initially, both $X$ and $W$ are precharged. The inputs must set up and hold around the rising edge of $\phi$. When $\phi$ rises and the gate evaluates, $W$ begins pulling down. If one or more of the inputs are asserted, $X$ will also pull down, cutting off the transistor that was discharging $W$. The keeper will restore $W$ back to a high level and the output $Y$ will remain low. If all of the inputs are low, $X$ will remain high, $W$ will discharge, and $Y$ will monotonically rise. The full keepers hold both $X$ and $W$ after evaluation. The gate has a built-in race that $X$ must fall quickly, so $W$ does droop too much and cause a glitch on $Y$. The annihilation gate requires very careful design and attention to noise sources, but is fast and compact.

The annihilation gate is a new incarnation of a long-lost circuit called Latched Domino [Pretorius86] shown in Figure 7.66. The Latched Domino gate adds a cross-coupled nMOS transistor to help pull down node $X$. It also replaces the full keepers with ordinary keepers. As long as the glitches on $X$ and $W$ are small enough, $Y_h$ and $Y_l$ are good monotonic dual-rail outputs.

![Annihilation gate](image1)

![Latched domino gate](image2)
Intel uses a similar gate called a Complementary Signal Generator (CSG) shown in Figure 7.67 to produce dual-rail outputs from single rail inputs in a 5 GHz ALU [Vangal02]. Again, nodes $X$ and $W$ precharge and the inputs must set up before the rising edge of $\phi$. When $\phi$ rises, $W$ begins to discharge. If any of the inputs are true, $X$ also begins to discharge. The pulldown and keeper strengths must be chosen so that $X$ falls much faster than $W$. Once one of these nodes falls, it turns on the cross-coupled pMOS pull-ups to restore the other node to full levels. These strong pull-ups also help fight leakage, permitting wide fan-in logic functions. The CSG was designed so the glitch on $W$ would not exceed 10% of $V_{DD}$. In a dual-$V_t$ process, low $V_t$ transistors were used on all but the noise-sensitive input transistors.

The CSG is very effective in circuits that can use single-rail signals through most of the path but that require dual-rail monotonic inputs to the last stage for functions such as XOR. They can be much faster and more compact than dual-rail domino but suffer from the very delicate race. The clock does impose a hard edge before which the inputs must set up so that skew and delay mismatches on this clock appear as sequencing overhead.

![Complementary signal generator](image)

**7.5.4.4 Output Prediction Logic** Clock-delayed and race-based dynamic logic represent two extremes in nonmonotonic logic. Both consist of two cascaded dynamic gates. CD Domino delays the clock to the second gate until the first has had time to fully discharge so the second gate will not glitch. Race-based logic such as annihilation gates and CSGs do not delay the clock but use transistor and keeper sizing to ensure the glitch on the second gate remains acceptably small. Output Prediction Logic (OPL) fits between these two extremes, delaying the clock by a moderate amount and accepting modest glitches [McMurchie00]. The delay is chosen as a compromise between performance and glitch size.
Figure 7.68 shows a basic OPL gate consisting of a Noise-Tolerant Precharge dynamic stage that was discussed in Section 6.2.4.3. You can view it either as a complementary CMOS structure with clocked evaluation and precharge transistors or as a dynamic gate plus a complementary pMOS pullup network. Like an ordinary dynamic gate, the output precharges high while the clock is low, then evaluates low when the clock rises and the appropriate inputs are asserted. However, like a static CMOS gate, the output can pull back high through the pMOS network to recover from output glitches.

Figure 7.69 shows a chain of OPL 2-input NAND gates. Each receives a clock delayed from the previous stage. As the stages are inverting, it resembles a chain of CD Domino gates. The amount of delay is critical to the circuit operation. Suppose $A$ is ‘1’ and all the unnamed outer inputs are also ‘1’ so $B, D$, and $F$ should pull low and $C$ and $E$ stay high. OPL precharges all the outputs to predict each output will remain high. The gates can be very fast because only half of the outputs have to transition.

Figure 7.70 shows three cases of short (a), long (b), and medium (c) clock delays between a pair of OPL inverters. Simulating OPL is tricky because if all the gates are identical, the outputs will tend to settle at a metastable point momentarily, then diverge as the previous gate transitions. To break this misleading symmetry, a small parasitic capacitance $C_p$ was added to node $B$. 

---

**FIG 7.68** OPL gate

**FIG 7.69** Chain of OPL gates

**FIG 7.70** OPL waveforms for various clock delays
In Figure 7.70(a), all the clocks rise simultaneously. \( \phi_2 \) arrives at the second stage while the input \( B \) is still high so \( C \) pulls most of the way low. When \( B \) falls, \( C \) rises back up. This causes \( D \) to fall, \( E \) to rise, and \( F \) to fall. In this mode of operation, the data ripples through the gates much as in static CMOS and the circuit delay is rather slow.

In Figure 7.70(b), the clock spacing is 50 ps. \( \phi_2 \) arrives at the second stage after the input \( B \) has pulled most of the way low so \( C \) remains high. After another delay, \( \phi_3 \) rises, \( D \) falls, and so forth. In this mode of operation, the OPL chain behaves in clock-blocked mode just like clock-delayed domino. The path delay is the sum of the clock delays plus the propagation delay of the final stage, which again is rather slow because the clock delay is lengthy.

In Figure 7.70(c), the clock spacing is 15 ps. \( \phi_2 \) arrives at the second stage as the input \( B \) is falling so \( C \) glitches slightly, then returns to a good high value. After another delay, \( D \) falls. Again, the path delay is essentially the sum of the clock delays and final stage delay, but it is now faster because the clock delay is shorter than required for CD domino. The extra speed comes at the expense of some glitching.

A challenge in designing OPL gates is to choose just the right clock spacing. It should be as short as possible but not too short. Figure 7.71 plots the delay from \( A \) to \( F \) against the spacing between clocks. The nMOS transistors are 2 units wide and the figure compares the performance for pMOS of 1, 3, or 5 units. Wider pMOS transistors have slower evaluation delays but recover better from glitches. The lowest path delay occurs with a clock spacing of 10–15 ps. The path slows significantly if the clock spacing is too short, so the designer should nominally provide some margin in clock delay to ensure the worst case is still long enough. In comparison, a chain of static CMOS NAND gates has a delay of 213 ps.

The basic OPL technique was illustrated for modified complementary CMOS gates that are relatively slow but recover quickly from large glitches. It also applies to other circuit families that have faster evaluation delays for high fan-in NOR structures such as...
pseudo-nMOS or dynamic gates, as illustrated in Figure 7.72(a and b). Pseudo-nMOS OPL is faster at evaluating because of the lower logical effort, but slower at recovery if the glitch is large. Dynamic OPL gates evaluate even faster but cannot recover at all if the glitch is large enough to flip the keeper. Using a low-skew feedback inverter improves the glitch tolerance for the keeper. As the best delay between clocks is a function of both evaluation delay and glitch tolerance, pseudo-nMOS and dynamic OPL are comparable in performance. Dynamic gates dissipate less power than pseudo-nMOS but may fail entirely if the clock delay is too short. Figure 7.72(c) shows a differential OPL gate using cross-coupled pMOS keepers that do not fight the initial transition and that can recover from arbitrarily large glitches [Kio01]. The inventors found that this was the fastest family of all, nearly five times faster than static CMOS.

7.5.5 Static-to-domino Interface

Static CMOS gates require inputs that are levels and may produce nonmonotonic glitches on the outputs. Domino gates require inputs that are monotonic during evaluation and produce pulses on the outputs. Therefore, interface circuitry is necessary at the static-to-dynamic interface to avoid glitches and circuitry at the dynamic-to-static interface to convert the pulses into levels.

7.5.5.1 Static-to-domino Interface

Falling static inputs to domino gates must set up by the time the gate begins evaluation and should not change until evaluation is complete. This imposes a hard edge and the associated clock skew penalties, so the static-to-domino interface is relatively expensive. High-performance skew-tolerant domino pipelines build entire loops out of domino to avoid paying the skew at the static-to-domino interface.

A simple solution to avoiding glitches at the interface is to latch the static signals as shown in Figure 7.73(a). The latch is opaque while the domino gates evaluate. Figure 7.73(b) shows that the latch does not even need to be placed at the end of the previous half-cycle. The static logic must be designed to set up before domino gates enter evaluation. The latch only acts to prevent the next token from arriving too early and upsetting the domino input.
In systems using flip-flops or pulsed latches, another approach is to capture the input on the clock edge with a flop or latch that produces monotonically rising outputs, as shown in Figure 7.74. The SA/F-F produces dual-rail monotonic outputs if the SR latch is replaced by HI-skew inverters. The K6 differential flip-flop also produces dual-rail monotonic pulsed outputs suitable for self-resetting logic that requires pulsed inputs. In any of these cases, you can build logic into the latch or flip-flop. For example, Figure 7.75 shows a single-rail pulsed domino flip-flop or entry latch (ELAT) with integrated logic used on UltraSparc and Itanium 2 [Klass99, Naffziger02]. It can be viewed as a fully dynamic version of the Klass SDFF. Falling inputs must set up before the clock edge, but rising inputs can borrow a small amount of time after the edge. The output is a monotonically rising signal suitable as an input to subsequent domino gates. The pulsed domino flip-flop can also use a single pulsed nMOS transistor in place of the two clocked devices [Mehta99].

7.5.5.2 Domino-to-static Interface Domino outputs are pulses that terminate when the gates precharge. Static logic requires levels that remain stable until they are sampled, independent of the clock period. At the domino-to-static interface, another latch is required as a pulse-to-level converter. The output of this latch can borrow time into subsequent static logic, so the latch does not impose a hard edge.
Figure 7.76 shows a domino gate with a simple built-in output latch. The HI-skew inverter is replaced with a clocked inverter. The critical path still passes through only the pMOS transistor, so the latch is nearly as fast as a simple inverter. On the falling edge of the clock, the latch locks out the precharge, holding the result of the domino gate until the next rising edge of the clock. A weak inverter staticizes the \( Y \) output. \( Y \) should typically be buffered before driving long wires to prevent noise from backdriving the latch. Note that \( Y \) does glitch low shortly after the rising edge of the clock. The glitch can cause excess power dissipation in the static logic. Dual-rail domino outputs can avoid the glitch at the cost of greater delay by using the SR latch shown in Figure 7.29(a or b).
The Itanium 2 uses a dynamic latch converter (DLC) on the last domino gate in each half-cycle to hold the output by delaying the precharge until the next rising clock edge. This provides greater skew tolerance in domino paths and allows the output to drive static logic. An ordinary dynamic gate receives the same clock for the precharge ($RCLK$) and evaluation ($ECLK$) transistors and has a weak pMOS keeper. Figure 7.77 shows a DLC that is a “bolt-on” block consisting of a delayed clock generator and an extra nMOS keeper to make a full keeper. The $RCLK$ generator produces a brief low-going precharge pulse on the rising edge of the clock. Although the precharge and evaluate transistors may be on momentarily, this is not a large concern because the DLC operates the last gate of the half-cycle so the inputs do not arrive until several gate delays after the clock edge. The DLC also may include scan circuitry illustrated in Section 9.

In self-resetting domino, the reset pulse for the last gate can also be delayed so that the domino output is compatible with static logic. For example, Figure 7.57 showed such a pulse generator for Global STP domino.
7.5.6 Delayed Keepers

Dynamic gates with high leakage current will eventually discharge to an invalid logic level unless they have strong keepers. The problem is especially severe when the inputs use many parallel low-\(V_t\) transistors. Unfortunately, the strong keeper slows the dynamic gate, reducing the performance advantage it was supposed to provide. As discussed in Section 6.2.4.3 for the burn-in keeper, this problem can be addressed by breaking the keeper into two parts. One part operates in the typical fashion. The second part turns on after some delay when the gate has had adequate time to evaluate. This combines the advantage of fast initial evaluation from the smaller keeper with better long-term leakage immunity from the two keepers in parallel.

Figure 7.78(a) shows such a conditional keeper [Alvandpour02]. \(P_2\) is the conventional feedback keeper. \(P_1\) turns on three gate delays after \(\phi\) rises to help fight leakage. Figure 7.78(b) shows High-Speed Domino that leaves \(X\) floating momentarily until \(P_1\) turns ON [Allam00]. Skew-Tolerant High-Speed Domino uses two transistors in series as the second keeper [Jung01], as shown in Figure 7.78(c). The inverting delay logic (IDL) can be an inverter, three inverters in series, or some other inverting structure with greater delay.

A challenge with any of these delayed keeper techniques is to ensure that the second part of the keeper turns on at a suitable time after the input arrives, but before too much leakage occurs. They work best for the first gate after a phase boundary, where the inputs are known to set up by the time the clock rises [Alvandpour02].

7.6 Synchronizers

Sequencing elements are characterized by a setup and hold time. If the data input changes before the setup time, the output reflects the new value after a bounded propagation delay. If the data changes after the hold time, the output reflects the old value after a bounded propagation delay. If the data changes during the aperture between the setup and hold
time, the output may be unpredictable and the time for the output to settle to a good logic level may be unbounded. Properly designed synchronous circuits guarantee the data is stable during the aperture. However, many interesting systems must interface with data coming from sources that are not synchronized to the same clock. For example, the user can press a key at any time and data coming over a network can be aligned with a clock of differing phase or frequency.

A synchronizer is a circuit that accepts an input that can change at arbitrary times and produces an output aligned to the synchronizer’s clock. Because the input can change during the synchronizer’s aperture, the synchronizer has a nonzero probability of producing a metastable output. This section first examines the response of a latch to an analog voltage that can change near the sampling clock edge. The latch can enter a metastable state for some amount of time that is unbounded, although the probability of remaining metastable drops off exponentially with time. Therefore, you can build a simple synchronizer by sampling a signal, waiting until the probability of metastability is acceptably low, then sampling again. In certain circumstances, the relationship of the data and clock timing is more predictable, permitting more reliable synchronizers.

7.6.1 Metastability

A latch is a bistable device; i.e., it has two stable states (0 and 1). Under the right conditions, that latch can enter a metastable state in which the output is at an indeterminate level between 0 and 1. For example, Figure 7.79 shows a simple model for a static latch consisting of two switches (probably transmission gates in practice) and two inverters. While the latch is transparent, the sample switch is closed and the hold switch open (Figure 7.79(a)). When the latch goes opaque, the sample switch opens and the hold switch closes (Figure 7.79(b)). Figure 7.79(c) shows the DC transfer characteristics of the two inverters. Because $A = B$ when the latch is opaque, the stable states are $A = B = 0$ and $A = B = V_{DD}$. The metastable state is $A = B = V_{m}$, where $V_{m}$ is not a legal logic level. This point is called metastable because the voltages are self-consistent and can remain there indefinitely. However, any noise or other disturbance will cause $A$ and $B$ to switch to one of the two stable states. Figure 7.79(d) shows an analogy of a ball on a hill. The top of the hill is a metastable state. Any disturbance will cause the ball to roll down to one of the two stable states on the left or right side of the hill.

Figure 7.80(a) plots the output of the latch from Figure 7.17(g) as the data transitions near the falling clock edge. If the data changes at just the wrong time $t_m$ within the aperture, the output can remain at the metastable point for some time before settling to a valid logic level. Figure 7.80(b) plots $t_{10}$ vs. $t_{20} - t_m$ on a semilogarithmic scale for a rising input and output. The delay is less than or equal to $t_{10}$ for inputs that meet the setup time and increases for inputs that arrive too close to $t_m$. The points marked on the graph will be used in the example at the end of this section.
The cross-coupled inverters behave like a linear amplifier with gain $G$ when $A$ is near the metastable voltage $V_m$. The inverter delay can be modeled with an output resistance $R$ and load capacitance $C$. We can model the behavior in metastability by assuming that the initial voltage on node $A$ when the latch becomes opaque at time $t = 0$ is

$$A(0) = V_m + a(0)$$  \hspace{1cm} (7.25)
where \( a(0) \) is a small signal offset from the metastable point. Figure 7.81 shows a small-signal model for \( a(t) \). The behavior after time 0 is given by the first-order differential equation

\[
\frac{G_a(t) - a(t)}{R} = C \frac{da(t)}{dt}
\]  

(7.26)

Solving this equation shows that the positive feedback drives \( a(t) \) exponentially away from the metastable point with a time constant determined by the gain and RC delay of the cross-coupled inverter loop.

\[
a(t) = a(0) e^{\frac{t}{\tau_s}} \quad \tau_s = \frac{RC}{G-1}
\]  

(7.27)

Suppose the node is defined to reach a legal logic level when \( |a(t)| \) exceeds some deviation \( \Delta V \). The time to reach this level is

\[
t_{\text{DQ}} = \tau_s \left[ \ln \Delta V - \ln a(0) \right]
\]  

(7.28)

This shows that the latch propagation delay increases as \( A(0) \) approaches the metastable point and \( a(0) \) approaches 0. The delay approaches infinity if \( a(0) \) is precisely 0, but this can never physically happen because of noise. However, there is no upper bound on the possible waiting time \( t \) required for the signal to become valid. If the input \( A(t) \) is a ramp that passes through \( V_m \) at time \( t_m \), \( a(0) \) is proportional to \( t_m - t_{\text{DC}} \). Observe that Eq. (7.28) is a good fit to the log-linear portion of Figure 7.80(b). The time constant \( \tau_s \) is essentially the reciprocal of the gain-bandwidth product \( [\text{Flannagan85}] \). Therefore, the feedback loop in a latch should have a high gain-bandwidth product to resolve from metastability quickly.

Designers need to know the probability that latch propagation delay exceeds some time \( t' \). Longer propagation delays are less likely because they require \( a(0) \) to be closer to 0. This probability should decrease with the clock period \( T_c \) because a uniformly distributed input change is less likely to occur near the critical time. Projecting through Eq. (7.28) shows that it should also decrease exponentially with waiting time \( t' \). Theoretical and experimental studies \( [\text{Chaney83, Veendrick80, Horstmann89}] \) find that the probability can be expressed as

\[
P\{t_{\text{DQ}} > t'\} = \frac{T_c}{T} e^{-\frac{t'}{\tau}} \quad \text{for} \ t' > 0
\]  

(7.29)

where \( T_c \) and \( \tau \) can be extracted through simulation \( [\text{Baghini02}] \) or measurement. Intuitively, \( T_c/T \), describes the probability that the input would change during the aperture,
causing metastability, and the exponential term describes the probability that the output hasn’t resolved after \( t' \) if it did enter metastability. The model is only valid for sufficiently long propagation delays (\( \delta \) significantly greater than \( \Delta_{DQ} \)).

We have seen that a good synchronizer latch should have a feedback loop with a high-gain-bandwidth product. Conventional latches have data and clock transistors in series, increasing the delay (i.e., reducing the bandwidth). Figure 7.82 shows a synchronizer flip-flop in which the feedback loops simplify to cross-coupled inverter pairs [Dike99]. Furthermore, the flip-flop is reset to 0, and then is only set to 1 if \( D = 1 \) to minimize loading on the feedback loop.

The flip-flop consists of master and slave jamb latches. Each latch is reset to 0 while \( D = 0 \). When \( D \) rises before \( \phi \), the master output \( X \) is driven high. This in turn drives the slave output \( Q \) high when \( \phi \) rises. The pulldown transistors are just large enough to overpower the cross-coupled inverters, but should add as little stray capacitance to the feedback loops as possible. \( X \) and \( Q \) are buffered with small inverters so they do not load the feedback loops.

Example

Find \( \tau_s \), \( T_0 \), and \( \delta \) for the latch using the data in Figure 7.80.

**Solution:** \( \delta \) is the propagation delay above which the data fits a good straight line on a log-linear scale. In Figure 7.80, this appears to be approximately 175 ps. The probability that the delay exceeds some \( t' \) is the chance that the input changing at a random time falls within the small aperture that leads to the high delay. We can choose two points on the linear portion of the plot and solve for the two unknowns. For example, choosing (0.1 ps, 290 ps) and (0.01 ps, 415 ps), we solve

\[
P(t_{DQ} > 290 \text{ ps}) = \frac{0.1 \text{ ps}}{h} T_0 \frac{290 \text{ ps}}{t'}
\]

\[
P(t_{DQ} > 415 \text{ ps}) = \frac{0.01 \text{ ps}}{h} T_0 \frac{415 \text{ ps}}{t'}
\]

(7.30)

\( T_0 \) drops out of the equations and we find \( \tau_s = 54 \text{ ps} \) and \( T_0 = 21 \text{ ps} \). Recall that this data was taken for a rising input. A conservative design should also consider the falling input and take data in the slow rather than typical environment.
A synchronizer accepts an input $D$ and a clock $\phi$. It produces an output $Q$ that ought to be valid some bounded delay after the clock. The synchronizer has an aperture defined by a setup and hold time around the rising edge of the clock. If the data is stable during the aperture, $Q$ should equal $D$. If the data changes during the aperture, $Q$ can be chosen arbitrarily. Unfortunately, it is impossible to build a perfect synchronizer because the duration of metastability can be unbounded. We define synchronizer failure as occurring if the output has not settled to a valid logic level after some time $t'$. Figure 7.83 shows a simple synchronizer built from a pair of flip-flops. $F1$ samples the asynchronous input $D$. The output $X$ may be metastable for some time, but will settle to a good level with high probability if we wait long enough. $F2$ samples $X$ and produces an output $Q$ that should be a valid logic level and be aligned with the clock. The synchronizer has a latency of one clock cycle, $T_c$. It can fail if $X$ has not settled to a valid level by a setup time before the second clock edge.

Each flip-flop samples on the rising clock edge when the master latch becomes opaque. The slave latch merely passes along the contents of the master and does not sig-
nificantly affect the probability of metastability. If the synchronizer receives an average of $N$ asynchronous input changes at $D$ each second, the probability of synchronizer failure in any given second is

$$P_{\text{failure}} = N \frac{T_0}{T_c} e^{-\left(\frac{T_c - \tau_s}{\tau_s}\right)}$$  \hspace{1cm} (7.32)

and the mean time between failures increases exponentially with cycle time

$$\text{MTBF} = \frac{1}{P_{\text{failure}}} = \frac{T_c}{N T_0}$$  \hspace{1cm} (7.33)

The acceptable MTBF depends on the application. For medical equipment where synchronizer reliability is crucial and latency is relatively unimportant, the MTBF can be chosen to be longer than the life of the universe ($\sim 10^{19}$ seconds) by waiting more than one clock cycle before using the data. For noncritical applications, the MTBF can be chosen to be merely longer than the designer’s expected duration of employment at the company!

Example

A particular synchronizer flip-flop in a 0.25 $\mu$m process has $\tau_s = 20$ ps and $T_0 = 15$ ps [Dike99]. Assuming the input toggles at $N = 50$ MHz and the setup time is negligible, what is the minimum clock period $T_c$ for which the MTBF exceeds one year?

Solution: $1\text{ year} = \pi \cdot 10^7$ seconds. Thus, we must solve

$$\pi \cdot 10^7 = \frac{T_c}{T_0} e^{\left(\frac{T_c - \tau_s}{\tau_s}\right)}$$

numerically for a minimum clock period of 625 ps (1.6 GHz).

Example

How much longer must we wait for a 1000-year MTBF?

Solution: Solving a similar equation gives 760 ps. Increasing the waiting time by 135 ps improved MTBF by a factor of 1000.
A common application of synchronizers is in communication between asynchronous clock domains, i.e., blocks of circuits that do not share a common clock. Suppose System A is controlled by $\text{clk}_A$ that needs to transmit $N$-bit data words to System B, which is controlled by $\text{clk}_B$, as shown in Figure 7.84. The systems can represent separate chips or separate units within a chip using unrelated clocks. Each word should be received by system B exactly once. System A must guarantee that the data is stable while the flip-flops in System B sample the word. It indicates when new data is valid by using a request signal ($\text{Req}$), so System B receives the word exactly once rather than zero or multiple times. System B replies with an acknowledge signal ($\text{Ack}$) when it has sampled the data so System A knows when the data can safely be changed. If the relationship between $\text{clk}_A$ and $\text{clk}_B$ is completely unknown, a synchronizer is required at the interface.

### 7.6.3 Communicating Between Asynchronous Clock Domains

A common application of synchronizers is in communication between asynchronous clock domains, i.e., blocks of circuits that do not share a common clock. Suppose System A is controlled by $\text{clk}_A$ that needs to transmit $N$-bit data words to System B, which is controlled by $\text{clk}_B$, as shown in Figure 7.84. The systems can represent separate chips or separate units within a chip using unrelated clocks. Each word should be received by system B exactly once. System A must guarantee that the data is stable while the flip-flops in System B sample the word. It indicates when new data is valid by using a request signal ($\text{Req}$), so System B receives the word exactly once rather than zero or multiple times. System B replies with an acknowledge signal ($\text{Ack}$) when it has sampled the data so System A knows when the data can safely be changed. If the relationship between $\text{clk}_A$ and $\text{clk}_B$ is completely unknown, a synchronizer is required at the interface.

The request and acknowledge signals are called *handshaking* lines. Figure 7.85 illustrates two-phase and four-phase handshaking protocols. The four-phase handshaking is level-sensitive while the two-phase handshaking is edge-triggered. In the four-phase handshake, system A places data on the bus. It then raises $\text{Req}$ to indicate that the data is valid. System B samples the data when it sees a high value on $\text{Req}$ and raises $\text{Ack}$ to indicate that the data has been captured. System A lowers $\text{Req}$, then system B lowers $\text{Ack}$. This protocol requires four transitions of the handshake lines. In the two-phase handshake, system A places data on the bus. Then it changes $\text{Req}$ (low to high or high to low) to indicate that the data is valid. System B samples the data when it detects a change in the level of $\text{Req}$ and toggles $\text{Ack}$ to indicate that the data has been captured. This protocol uses fewer transitions (and thus possibly less time and energy), but requires circuitry that responds to edges rather than levels.
Req is not synchronized to clkB. If it changes at the same time clkB rises, System B may receive a metastable value. Thus, System B needs a synchronizer on the Req input. If the synchronizer waits long enough, the request will resolve to a valid logic level with very high probability. The synchronizer may resolve high or low. If it resolves high, the rising request was detected and System B can sample the data. If it resolves low, the rising request was just missed. However, it will be detected on the next cycle of clkB, just as it would have been if the rising request occurred just slightly later. Ack is not synchronized to clkA, so it also requires a synchronizer.

Figure 7.86 shows a typical two-phase handshaking system [Crews03]. clkA and clkB operate at unrelated frequencies and each system may not know the frequency of its counterpart. Each system contains a synchronizer, a level-to-pulse converter, and a pulse-to-level converter. System A asserts ReqA for one cycle when DataA is ready. We will refer to this as a pulse. The XOR and flip-flop form a pulse-to-level converter that toggles the level of Req. This level is synchronized to clkB. When an edge is detected, the level-to-pulse converter produces a pulse on ReqB. This pulse in turn toggles Ack. The acknowledge level is synchronized to clkA and converted back to a pulse on AckA. The synchronizers add significant latency so the throughput of asynchronous communication can be much lower than that of synchronous communication.

7.6.4 Common Synchronizer Mistakes

Although a synchronizer is a simple circuit, it is notoriously easy to misuse. For example, the AMD 9513 system timing controller, AMD 9519 interrupt controller, Zilog Z-80 Serial I/O interface, Intel 8048 microprocessor, and AMD 29000 microprocessor are all said to have suffered from metastability problems [Wakerly00].

One way to build a bad synchronizer is to use a bad latch or flip-flop. The synchronizer depends on positive feedback to drive the output to a good logic level. Therefore, dynamic latches without feedback such as Figure 7.17(a–d) do not work. The probability of failure grows exponentially with the time constant of the feedback loop. Therefore, the loop should be lightly loaded. The latch from Figure 7.17(f) is a poor choice because a large capacitive load on the output will increase the time constant; Figure 7.17(g) is a much better choice.
Another error is to capture inconsistent data. For example, Figure 7.87(a) shows a single signal driving two synchronizers (each consisting of a pair of back-to-back flip-flops). If the signal is stable through the aperture, \( Q_1 \) and \( Q_2 \) will be the same. However, if the signal changes during the aperture, \( Q_1 \) and \( Q_2 \) might resolve to different values. If the system requires that \( Q_1 \) and \( Q_2 \) be identical representations of the data input, they must come from a single synchronizer.

Another example is to synchronize a multi-bit word where more than one bit might be changing at a time. For example, if the word in Figure 7.87(b) is transitioning from 0000 to 1111, the synchronizer might produce a value such as 0101 that is neither the old nor the new data word. For this reason, the system in Figure 7.86 synchronized only the \( \text{Req}/\text{Ack} \) signals and used them to indicate that data was stable to sample or finished being sampled. Gray codes (see Section 10.7.3) are also useful for counters whose outputs must be synchronized because exactly one bit changes on each count so the synchronizer is guaranteed to find either the old or the new data value.
In general, synchronizer bugs are intermittent and very difficult to locate and diagnose. For this reason, the number of synchronizers in a system should be strictly limited.

### 7.6.5 Arbiters

The arbiter of Figure 7.88(a) is closely related to the synchronizer. It determines which of two inputs arrived first. If the spacing between the inputs exceeds some aperture time, the first input should be acknowledged. If the spacing is smaller, exactly one of the two inputs should be acknowledged, but the choice is arbitrary. For example, in a television game show, two contestants may pound buttons to answer a question. If one presses the button first, she should be acknowledged. If both press the button at times too close to distinguish, the host may choose one of the two contestants arbitrarily.

Figure 7.88(b) shows an arbiter built from a SR latch and a four-transistor metastability filter. If one of the request inputs arrives well before the other, the latch will respond appropriately. However, if they arrive at nearly the same time, the latch may be driven into metastability, as shown in Figure 7.88(c). The filter keeps both acknowledge signals low until the voltage difference between the internal nodes \( n_1 \) and \( n_2 \) exceeds \( V_t \), indicating that a decision has been made. Such an asynchronous arbiter will never produce metastable outputs. However, the time required to make the decision can be unbounded, so the acknowledge signals must be synchronized before they are used in a clocked system.
Arbiters can be generalized to select 1-of-\(N\) or \(M\)-of-\(N\) inputs. However, such arbiters have multiple metastable states and require careful design [van Berkel99].

### 7.6.6 Degrees of Synchrony

The simple synchronizer from Section 7.6.2 accepts inputs that can change at any time, but has a nonzero probability of failure. In practice, many inputs may not be aligned to a single system clock, but they may still be predictable. Table 7.3 provides a classification of degrees of synchrony between input signals and the receiver system clock [Messerschmitt90] based on the difference in phase \(\Delta \phi\) and frequency \(\Delta f\).

[Dally98] describes a number of synchronizers that have zero failure probability and possibly lower latency when the input is predictable. They are based on the observation that either the signal or a copy of the signal delayed by \(t_a\) will be stable throughout the aperture. Hence, a synchronizer that can predict the input arrival time can choose the signal or its delayed counterpart to safely sample. Mesochronous signals are synchronized by measuring the phase difference and delaying the input enough to ensure it falls outside the aperture. Plesiochronous signals can be synchronized in a similar fashion, but the phase difference slowly varies so the delay must be occasionally adjusted. Because the frequencies differ, the synchronizer requires some control flow to handle the missing or extra data items. Periodic signals also require control flow and use a clock predictor to calculate where the next clock edge will occur and whether the signal must be delayed to avoid falling in the aperture.

### 7.7 Wave Pipelining

Recall that sequencing elements are used in pipelined systems to prevent the current token from overtaking the next token or from being overtaken by the previous token in the pipeline. If the elements propagate through the pipeline at a fairly constant rate, explicit sequencing elements may not be necessary to maintain sequence. As an analogy, fiber optic cables carry data as a series of light pulses. Many pulses enter the cable before the first one reaches the end, yet the cable does not need internal latches to keep the pulses separated because they propagate along the cable at a well-controlled velocity. The maximum data rate is limited by the dispersion along the line that causes pulses to smear over time and blur into one another if they become too short.

Figure 7.89 compares traditional pipelining with wave pipelining. In both cases, the pipeline contains combinational logic separated by registers (Figure 7.89(a)). The registers \(F1\) and \(F2\) receive clocks \(clk1\) and \(clk2\) that are nominally identical, but might experience skew. Figure 7.89(b) shows traditional pipelining. The data is launched on the rising edge of \(clk1\). Its propagation is indicated by the hashed cone. \(D2\) becomes stable somewhere between the contamination and propagation delays after the clock edge (neglecting the flip-flop \(clk\)-to-\(Q\) delay). \(D2\) must not change during the setup and hold aperture around \(clk2\), marked with the gray box. The figure shows two successive cycles in which tokens \(i\)
and \( i + 1 \) move through the pipeline. Each token passes through the combinational logic in a single cycle. Figure 7.89(c) shows wave pipelining with a clock of twice the frequency. Token \( i \) enters the combinational logic, but takes two cycles to reach \( F2 \). Meanwhile, token \( i + 1 \) enters the logic a cycle later. As long as each token is stable to sample at \( F2 \) and the cones do not overlap, the pipeline will operate correctly with the same latency but twice the throughput.

[Burleson98] gives a tutorial on wave pipelining and derives the timing constraints. In general, a wave pipeline can contain \( N \) tokens between each pair of registers. The maximum value of \( N \) is limited by the ratio of propagation delay to dispersion of the logic cones

\[
N < \frac{t_{pd}}{t_{pd} - t_{id}} \tag{7.34}
\]
If the contamination and propagation delays are nearly equal, the combinational logic can contain many tokens simultaneously. In practice, the delays tend to be widely variable because of voltage, temperature, and processing as well as differences in path lengths through the logic. Clock skew and sequencing overhead also eat into the timing budgets. In practice, even achieving $N = 2$ simultaneous tokens can be difficult and wave pipelining has not achieved widespread popularity for general-purpose logic.
7.8 Pitfalls and Fallacies

**Incompletely reporting flip-flop delay**
The effective delay of a flip-flop is its minimum D-to-Q time. This is the sum of the setup time \( t_{\text{setup}} \) and the clk-to-Q delay \( t_{\text{pdq}} \) if these delays are defined to minimize the sum. Some engineers focus on only the clk-to-Q delay or define setup and clk-to-Q delays in a way that does not minimize the sum.

**Failing to check hold times**
One of the leading reasons that chips fail to operate even though they appear to simulate correctly is hold time violations, especially violations caused by unexpected clock skew. Unless a design uses two-phase non-overlapping clocks, the clock skew should be carefully modeled and the hold times should be checked with a static timing analyzer. These checks should happen as soon as a block is designed so that errors can be corrected immediately. For example, a large microprocessor used a wide assortment of delayed clocks to solve setup time problems on long paths. Hold times were not checked until shortly before tapeout, leading to a significant schedule slip when many violations were found.

**Choosing a sequencing methodology too late in the design cycle**
Designers may choose from many sequencing methodologies, each of which has tradeoffs. The best methodology for a particular application is very debatable, and engineers love a good debate. If the sequencing methodology is not settled at the beginning of the project, experience shows that engineers will waste tremendous amounts of time redoing work as the method changes, or supporting and verifying multiple methodologies. Projects need a strong technical manager to demand that a team choose one method at the beginning and stick with it.

**Failing to synchronize asynchronous inputs**
Unsynchronized inputs can cause strange and wonderful sporadic system failures that are very difficult to locate. For example, a finite state machine running off one clock received a READY input from a UART running on another clock when the UART had data available, as shown in Figure 7.90. The designer reasoned that synchronizing the READY signal was unimportant because if it changed near the clock edge of the FSM, she did not care whether it was detected in one cycle or the next. Moreover, the clock was so slow that metastability would have time to resolve. However, the FSM occasionally failed by jumping to seemingly random states that could never legally occur. After two months of debugging, she realized that the problem was triggered if the asynchronous READY signal was asserted a few gate delays before the FSM clock edge. The propagation delay through the combinational logic was different for various bits of the next state logic. Some bits had changed to their new values while others were still at their old values, so the FSM could jump to an undefined state. Registering the READY signal with the FSM clock before it drove the combinational logic solved the problem.

**Building faulty synchronizers**
Designers have found many ways to build faulty synchronizers. For example, if an asynchronous input drives more than one synchronizer, the two synchronizers
7.9 Case Study: Pentium 4 and Itanium 2

Sequencing Methodologies

The Pentium 4 and Itanium 2 represent two philosophies of high-performance microprocessor design sometimes called *Speed Demon* and *Braniac*, respectively. The Pentium 4 was designed by Intel for server and desktop applications and has migrated into laptop computers as well. The Itanium 2 was jointly designed by Hewlett-Packard and Intel for high-end server applications. Figure 7.91 shows the date of introduction and the performance of several generations of these processors.

The Pentium 4 uses a very long (20+ stage) pipeline with few stages of logic per cycle to achieve extremely high frequencies. It issues up to three instructions per cycle, but the long pipeline causes severe penalties for branch mispredictions and cache misses, so the overall average number of instructions executed per cycle is relatively low. Figure 4.76 showed a die photo of the 42-million transistor Pentium 4. The chip consumes around 55 watts. A top-of-the-line Pentium 4 sold in 1000-unit quantities for around $400–$600 (depending on price pressure from competitor AMD). The chip has aggressively migrated...
into Intel’s most advanced processes both to achieve high performance and to reduce the
die size and manufacturing cost. The Speed Demon approach also gives Intel bragging
rights to the highest clock frequency microprocessors, which is important because many
consumers compare processors on clock frequency rather than benchmark performance.

In contrast, the Itanium 2 focuses on executing many instructions per cycle at a lower
clock rate. It uses an 8-stage integer pipeline clocked at about half the rate of the Pentium
4 in the same process, so each cycle accommodates about twice as many gate delays
(roughly 20–24 FO4 inverter delays, compared to roughly 10–12 for the Pentium 4). However, it issues up to six instructions per cycle and has a very high-bandwidth memory
and I/O system to deliver these instructions and their data. As a result, it achieves nearly
the same integer performance and much better floating-point benchmark results than the
Pentium 4. Moreover, it also performs well on multiprocessor and transaction processing
tasks typical of high-end servers. Figure 7.92 shows a die photo of the Itanium 2 with a
3MB level 3 (L3) cache; notice that the three levels of cache occupy most of the die area
and most of the 221 million transistors. The 1.5 GHz model with 6MB cache bumps the
transistor count to 410 million and further dwarfs the processor core. The chip consumes
about 130 watts, limited by the cost of cooling multiprocessor server boxes. A high-end
Itanium 2 sold for more than $4000 because the server market is much less price-sensitive.
The chip has lagged a year behind the Pentium 4 in process technology.
The Pentium 4 actually operates at three different internal clock rates [Hinton01, Kurd01]. In addition to the core clock that drives most of the logic, it has a double-speed fast clock for the ALU core and a half-speed slow clock for noncritical portions of the chip. The core clock is distributed across the chip using a triple spine, as will be shown in Section 12.5.4.3. These clocks drive pulsed latches, flip-flops, and self-resetting domino gates.

The ALU runs at a remarkable rate of twice the core clock frequency (about 6 FO4 inverter delays). To achieve this speed, it is stripped down to just the essential functions of the bypass multiplexer and the 16-bit add/subtract unit. Other less commonly used blocks such as the shifter and multiplier operate at core frequency. The ALU uses unfooted domino gates. The gates produce pulsed outputs and precharge in a self-timed fashion using the Global STP approach described in Section 7.5.2.4. These circuits demanded extensive verification by expert circuit designers to ensure the domino gates function reliably.

The Pentium 4 uses pulsed latches operating at all three clock speeds. Figure 7.93 shows pulse generators that receive the core clock and produce the appropriate output pulses. The medium-speed pulse generator produces a pulse on the rising edge of the core clock. The pulse width can be shaped by the adjustable delay buffer to provide both long pulses (offering more time borrowing) and short pulses (to prevent hold-time problems). The buffer is built from a digitally controlled current-starved inverter with four discrete settings. The pulse generator also accepts enable signals to gate the clock or save power on unused blocks. The fast pulse generator produces pulses on both the rising and falling edges of the core clock. Therefore, the core clock should have nearly equal high and low times, i.e., 50% duty cycle, so the pulses are equally spaced.

The Itanium 2 operates at a single primary clock speed, but also makes use of extensive domino logic and pulsed latches [Naffziger02, Fetzer02, Rusu03]. The clock is distributed across the chip using an H-tree, as will be shown in Section 12.5.4.2. The H-tree drives 33 second-level clock buffers distributed across the chip. These buffer outputs, called SLCBOs, in turn drive local clock gates that serve banks of sequencing elements within functional blocks. There are 24 different types of clock gates producing inverted, stretched, delayed, and pulsed clocks. Figure 7.94 shows some of these clocks. Each gater comes in many sizes and is tuned to drive different clock loads with low skew over regions of up to about 1000 µm. Section 12.5.6.3 analyzes the clock skew from this distribution network.
7.9 CASE STUDY: PENTIUM 4 AND ITANIUM 2 SEQUENCING METHODOLOGIES

![Diagram of pulse generators and clock gater waveforms]

**FIG 7.93** Pulse generators

**FIG 7.94** Clock gater waveforms
In the Itanium 2, 95% of the static logic blocks use Naffziger-pulsed latches with 125-ps wide pulses called $PCK$, as were described in Section 7.3.3. The pulsed latches are fast, permit a small amount of time borrowing, and present a small load to the clock. In situations where more time borrowing is needed, the gater may produce a wider pulsed clock $WPCK$. As discussed in Section 7.4.1.2, clocked demuxers using $NPCK$ can be inserted between back-to-back pulsed latches to prevent hold time violations.

The Itanium 2 uses extensive amounts of domino logic to achieve high performance at the expense of power consumption and careful design. Figure 7.95 shows a typical four-phase skew-tolerant domino pipeline from the Itanium 2. $CK$ and $NCK$ are clocks with a duty cycle slightly higher than 50% that are playing the roles of $\phi_1$ and $\phi_3$. They are delayed with buffers to produce $CKD$ and $NCKD$ ($\phi_2$ and $\phi_4$).

The last gate in each phase uses a dynamic latch converter (DLC) to hold the output so that it can drive static logic and retain its state when the clock stops, as was discussed in Section 7.5.5.2. The DLC also provides scan capability at each half-cycle boundary to help with test.

At static-to-dynamic interfaces, inputs pass through pulsed entry latches (ELATs) that capture the static signal and convert it into a single-rail monotonic dynamic signal. These ELATs were shown in Figure 7.75 and can perform logic as well as latching. Some ELATs use $PCK$, while others derive the pulse internally from $CK$.

In some especially critical paths, alternating stages use unfooted domino gates. The falling edge of the clocks for these stages is delayed further to avoid contention during precharge. Figure 7.96 shows an extreme example in which a footed gate is followed by three stages of unfooted domino with successively delayed precharge edges, as was done in the 64-bit Naffziger adder used in the integer execution units.
Summary

This chapter has examined the tradeoffs of sequencing with flip-flops, two-phase transparent latches, and pulsed latches. The ITRS forecasts cycle times dropping well below 10 FO4 delays (see Table 4.17). Minimizing sequencing overhead will be very important in these high-performance systems. Flip-flops are the simplest, but have the greatest sequencing overhead. Transparent latches are most tolerant of skew and allow the most time borrowing, but require greater design effort to partition logic into half-cycles instead of cycles. Pulsed latches have the lowest sequencing overhead, but are most susceptible to min-delay problems. Table 7.4 compares the sequencing overhead, minimum delay constraint, and time borrowing capability of each technique. All of the techniques are used in commercial products, and the designer's choice depends on the design constraints and CAD tools.

In class projects for introductory VLSI classes, timing analysis is often rudimentary or nonexistent. Using two-phase nonoverlapping clocks generated off chip is attractive because you can guarantee the chip will have no max-delay or min-delay failures if the clock period and nonoverlap are sufficiently large. However, it is not practical to generate and distribute two nonoverlapping phases on a large, high-performance commercial chip. The great majority of low- and mid-performance designs and some high-speed designs use flip-flops. Flip-flops are very easy to use and are well understood by most designers. Even more importantly, they are handled well by synthesis tools and timing analyzers. Unfortunately, in systems with few gate delays per cycle, the sequencing overhead can consume a large fraction of the cycle. Moreover, many standard cell flip-flops are intentionally rather slow to prevent hold time violations at the expense of greater sequencing overhead.

Most two-phase latch systems distribute a single clock and locally invert it to drive the second latch. These systems tolerate significant amounts of clock skew without loss of performance and can borrow time to balance delay intentionally or opportunistically. However, the systems require more effort to understand because time borrowing distrib-

<table>
<thead>
<tr>
<th>Table 7.4 Comparison of sequencing elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequencing overhead</strong></td>
</tr>
<tr>
<td>Flip-Flops: $t_{foo} + t_{setup} + t_{skew}$</td>
</tr>
<tr>
<td>Two-Phase Transparent Latches: $2t_{polq}$</td>
</tr>
<tr>
<td>Pulsed Latches: $\max(t_{polq} + t_{setup} - t_{pdq} + t_{skew})$</td>
</tr>
<tr>
<td><strong>Minimum logic delay</strong></td>
</tr>
<tr>
<td>Flip-Flops: $t_{hold} - t_{cyg} + t_{skew}$</td>
</tr>
<tr>
<td>Two-Phase Transparent Latches: $t_{hold} - t_{cyg} - t_{nonoverlap} + t_{skew}$</td>
</tr>
<tr>
<td>Pulsed Latches: $t_{hold} - t_{cyg} + t_{puw} + t_{skew}$</td>
</tr>
<tr>
<td><strong>Time borrowing</strong></td>
</tr>
<tr>
<td>Flip-Flops: 0</td>
</tr>
<tr>
<td>Two-Phase Transparent Latches: $\frac{T}{2} - (t_{setup} + t_{nonoverlap} + t_{skew})$</td>
</tr>
<tr>
<td>Pulsed Latches: $t_{puw} - (t_{setup} + t_{skew})$</td>
</tr>
</tbody>
</table>
utes the timing constraints across many stages of a pipeline rather than isolating them at each stage. Not all timing analyzers handle latches gracefully, especially when there are different amounts of clock skew between different clocks [Harris99]. Two-phase latches have been used in the Alpha 21064 and 21164 [Gronowski98], PowerPC 603 [Gerosa94], and many other IBM designs.

Pulsed latches have low sequencing overhead. They present a tradeoff when choosing pulse width: A wide pulse permits more time borrowing and skew tolerance, but makes min-delay constraints harder to meet. Pulsed latches are also popular because they can be modeled as fast flip-flops with a lousy hold time from the point of view of a timing analyzer (or novice designer) if intentional time borrowing is not permitted. The min-delay problems can be largely overcome by mixing pulsed latches for long paths and flip-flops for short paths. Unfortunately, many real designs have paths in which the propagation delay is very long but the contamination delay is very short, making robust design more challenging. Pulsed latches have been used on Itanium 2 [Naffziger03], Athlon [Draper97], and CRAY 1 [Unger86]. However, they can wreak havoc with conventional commercially available design flows and are best avoided unless the performance requirements are extreme.

Domino circuits are widely used in high-performance systems because they are 1.5-2x faster than static CMOS. Traditional domino circuits with latches have high sequencing overhead that wastes much of the potential speedup, so most designers have moved to skew-tolerant techniques. Static-to-domino interfaces impose hard edges and the associated sequencing overhead, motivating the use of domino throughout critical loops. Single-rail domino only computes noninverting functions, so most loops require dual-rail domino that consumes more area, wiring, and power and is ill-suited to wide NORs. An alternative is to push the inverting functions to the end of the pipeline, using single-rail domino through most of the pipeline and nonmonotonic static logic at the end. The area savings comes at the cost of one hard edge in the cycle.

Four-phase or delayed reset skew-tolerant domino circuits work well in datapaths because the clock generation is relatively simple. Self-resetting domino is ideally suited to memories where the decoder power consumption is greatly reduced by only precharging the output that switched and where the number of unique circuits to design is relatively small. It was also used on the Pentium 4, but was costly in terms of designer effort because so many pulse constraints must be satisfied.

Clock-delayed domino is used in wide dynamic NOR functions where the power consumption of pseudo-nMOS is unacceptable. For example, it is an important technique for CAMs and PLAs. The delay matching raises an unpleasant tradeoff between speed and correct operation, requiring significant margin for safe operation. The risk of race conditions deters many designers from using it more widely. Annihilation gates and complementary signal generators are interesting special cases in which no clock gate delay at all is required. Output prediction logic is also interesting, but has yet to be proven in a large application.

When inputs to a system arrive asynchronously, they cannot be guaranteed to meet setup or hold times at clocked elements. Even if we do not care whether an input arrived in one cycle or the next, we must ensure that the clocked element produces a valid logic
level. Unfortunately, if the element samples a changing input at just the wrong time, it may produce a metastable output that remains invalid for an unbounded amount of time. The probability of metastability drops off exponentially with time. Systems use synchronizers to sample the asynchronous input and hold it long enough to resolve to a valid logic level with very high probability before passing it onward.

Most synchronous VLSI systems use opaque sequencing elements to separate one token from the next. In contrast, many optical systems transmit data as pulses separated in time. As long as the propagation medium does not disperse the pulses too badly, they can be recovered at a receiver. Similarly, if a VLSI system has low dispersion, i.e., nearly equal contamination and propagation delays, it can send more than one wave of data without explicit latching. Such wave pipelining offers the potential of high throughput and low sequencing overhead. However, it is difficult to perform in practice because of the variability of data delay.

Exercises

Use the following timing parameters for the questions in this section.

| Table 7.5 Sequencing element parameters |
|-------------------------------|--------|---------|---------|---------|--------|
|                               | Setup Time | clk-to-Q Delay | D-to-Q Delay | Contamination Delay | Hold Time |
| Flip-flops                    | 65 ps      | 50 ps    | n/a      | 35 ps     | 30 ps   |
| Latches                       | 25 ps      | 50 ps    | 40 ps    | 35 ps     | 30 ps   |

7.1 For each of the following sequencing styles, determine the maximum logic propagation delay available within a 500 ps clock cycle. Assume there is zero clock skew and no time borrowing takes place.
   a) Flip-flops
   b) Two-phase transparent latches
   c) Pulsed latches with 80 ps pulse width

7.2 Repeat Exercise 7.1 if the clock skew between any two elements can be up to 50 ps.

7.3 For each of the following sequencing styles, determine the minimum logic contamination delay in each clock cycle (or half-cycle, for two-phase latches). Assume there is zero clock skew.
   a) Flip-flops
   b) Two-phase transparent latches with 50% duty cycle clocks
7.4 Repeat Exercise 7.3 if the clock skew between any two elements can be up to 50 ps.

7.5 Suppose one cycle of logic is particularly critical and the next cycle is nearly empty. Determine the maximum amount of time the first cycle can borrow into the second for each of the following sequencing styles. Assume there is zero clock skew.
   a) Flip-flops
   b) Two-phase transparent latches with 50% duty cycle clocks
   c) Two-phase transparent latches with 60 ps of nonoverlap between phases
   d) Pulsed latches with 80 ps pulse width

7.6 Repeat Exercise 7.5 if the clock skew between any two elements can be up to 50 ps.

7.7 Prove Eq (7.17).

7.8 Consider a flip-flop built from a pair of transparent latches using nonoverlapping clocks. Express the setup time, hold time, and clock-to-Q delay of the flip-flop in terms of the latch timing parameters and \( t_{\text{nonoverlap}} \).

7.9 For the path in Figure 7.97, determine which latches borrow time and if any setup time violations occur. Repeat for cycle times of 1200, 1000, and 800 ps. Assume there is zero clock skew and that the latch delays are accounted for in the propagation delay \( \Delta s \).
   a) \( \Delta 1 = 550 \text{ ps}; \Delta 2 = 580 \text{ ps}; \Delta 3 = 450 \text{ ps}; \Delta 4 = 200 \text{ ps} \)
   b) \( \Delta 1 = 300 \text{ ps}; \Delta 2 = 600 \text{ ps}; \Delta 3 = 400 \text{ ps}; \Delta 4 = 550 \text{ ps} \)

7.10 Determine the minimum clock period at which the circuit in Figure 7.98 will operate correctly for each of the following logic delays. Assume there is zero clock skew and that the latch delays are accounted for in the propagation delay \( \Delta s \).
   a) \( \Delta 1 = 300 \text{ ps}; \Delta 2 = 400 \text{ ps}; \Delta 3 = 200 \text{ ps}; \Delta 4 = 350 \text{ ps} \)
b) $\Delta_1 = 300 \text{ ps}; \Delta_2 = 400 \text{ ps}; \Delta_3 = 400 \text{ ps}; \Delta_4 = 550 \text{ ps}$

c) $\Delta_1 = 300 \text{ ps}; \Delta_2 = 900 \text{ ps}; \Delta_3 = 200 \text{ ps}; \Delta_4 = 350 \text{ ps}$

7.11 Repeat Exercise 7.10 if the clock skew is 100 ps.

7.12 Label the timing types of each signal in the circuit from Figure 7.97. The flip-flop is constructed with back-to-back transparent latches—the first controlled by $\text{clk}_b$ and the second by $\text{clk}$.

7.13 Using a simulator, compare the $D$-to-$Q$ propagation delays of a conventional dynamic latch from Figure 7.17(d) and a TSPC latch from Figure 7.30(a). Assume each latch is loaded with a fanout of 4. Use 4 $\lambda$-wide clocked transistors and tune the other transistor sizes for least propagation delay.

7.14 Using a simulator, find the setup and hold times of a TSPC latch under the assumptions of Exercise 7.13.

7.15 Determine the maximum logic propagation delay available in a cycle for a traditional domino pipeline using a 500 ps clock cycle. Assume there is zero clock skew.

7.16 Repeat Exercise 7.15 if the clock skew between any two elements can be up to 50 ps.

7.17 Determine the maximum logic propagation delay available in a cycle for a four-phase skew-tolerant domino pipeline using a 500 ps clock cycle. Assume there is zero clock skew.

7.18 Repeat Exercise 7.17 if the clock skew between any two elements can be up to 50 ps.

7.19 How much time can one phase borrow into the next in Exercise 7.18 if the clocks each have a 50% duty cycle?

7.20 Repeat Exercise 7.18 if the clocks have a 65% duty cycle.

7.21 Design a fast-pulsed latch. Make the gate capacitance on the clock and data inputs equal. Let the latch drive an output load of four identical latches. Simulate your latch and find the setup and hold times and clock-to-$Q$ propagation and contamination delays. Express your results in FO4 inverter delays.
7.22 Simulate the worst-case propagation delay of an 8-input dynamic NOR gate driving a fanout of 4. Report the delay in all 16 design corners (voltage, temperature, nMOS, pMOS). Also determine the delay of a fanout-of-4 inverter in each of these corners. By what percentage does the absolute propagation delay of the NOR gate vary across corners? By what percentage does its normalized delay vary (in terms of FO4 inverters)? Comment on the implications for circuits using matched delays.

7.23 A synchronizer uses a flip-flop with $\tau_s = 54$ ps and $T_0 = 21$ ps. Assuming the input toggles at 10 MHz and the setup time is negligible, what is the minimum clock period for which the mean time between failures exceeds 100 years?

7.24 Simulate the synchronizer flip-flop of Figure 7.82 and make a plot analogous to Figure 7.80. From your plot, find $\Delta_{DQ}$, $h$, $\tau$, and $T_0$.

7.25 InferiorCircuits, Inc., wants to sell you a perfect synchronizer that they claim never produces a metastable output. The synchronizer consists of a regular flip-flop followed by a high-gain comparator that produces a high output for inputs above $0.25 \cdot V_{DD}$ and a low output for inputs below that point. The VP of marketing argues that even if the flip-flop enters metastability, its output will hover near $V_{DD}/2$ so the synchronizer will produce a good high output after the comparator. Why wouldn't you buy this synchronizer?