Example 10.7

If the ALU self-bypass path from Figure 10.6 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 (10.12), 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 $t_{cd} - t_{\text{hold}} + t_{ccq} = 45 - (-10) + 75 = 130$ ps.

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 10.7).

$$t_{pd} \le T_c - \underbrace{\max\left(t_{pdq},\ t_{pcq} + t_{\text{setup}} - t_{pw} + t_{\text{skew}}\right)}_{\text{sequencing overhead}} \tag{10.17}$$

$$t_{cd} \ge t_{\text{hold}} + t_{pw} - t_{ccq} + t_{\text{skew}} \tag{10.18}$$

$$t_{\text{borrow}} \le t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right) \tag{10.19}$$

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 advantage of the window of transparency to tolerate some clock skew without increasing the sequencing overhead. Clock skew will be addressed further in Section 13.4. 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.

10.3 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.

10.3.1 Conventional CMOS Latches

Figure 10.17(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 V_{DD}); 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 9.3.3). *D* drives the *diffusion input* of a pass transistor directly, leading to potential noise issues (see Section 9.3.9) 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.

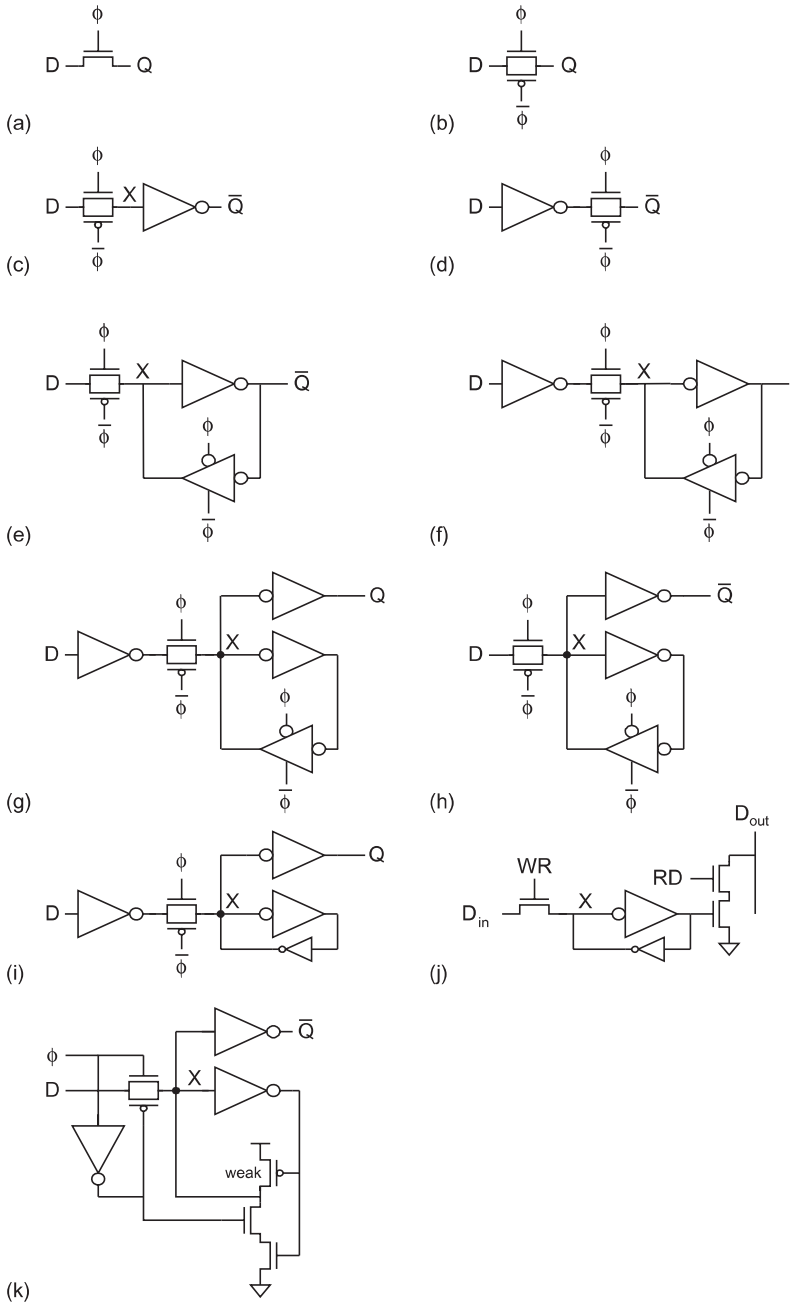

FIGURE 10.17 Transparent latches

Figure 10.17(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 $\overline{\phi}$, which can be provided as an additional input or locally generated from ϕ through an inverter. Figure 10.17(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 10.17(d) also behaves as an inverting latch with a buffered input but unbuffered output. As discussed in Section 9.2.5.1, the inverter followed by a 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. Figure 10.17(c) and (d) are both 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 10.17(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 10.17(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 10.17(g) is a 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 10.17(h) may be preferable because it is faster and more compact. Intel uses this as a standard datapath latch [Karnik01]. Figure 10.17(i) shows the *jamb latch*, a variation of Figure 10.17(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 10.17(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 latch is enabled at any given time with its *RD* signal. The Itanium 2 processor uses the latch shown in Figure 10.17(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.

The dynamic latch of Figure 10.17(d) can also be drawn as a clocked tristate, as shown in Figure 10.18(a). Such a form is sometimes called *clocked CMOS* (C^2MOS) [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^2MOS is slightly smaller because it eliminates two contacts. Figure 10.18(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].

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

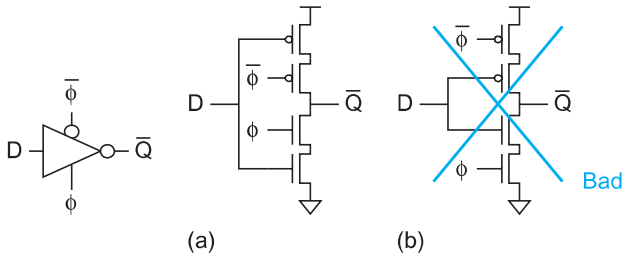

FIGURE 10.18 C^2MOS Latch

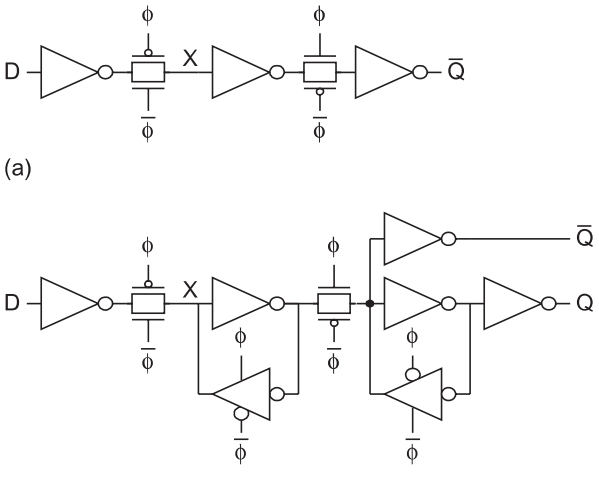

FIGURE 10.19 Flip-flops

10.3.2 Conventional CMOS Flip-Flops

Figure 10.19(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 10.19(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 $\overline{Q}$ output [Gerosa94]. Most standard cell libraries employ this design because it is simple, robust, compact, and energy-efficient [Stojanovic99]. However, some of the alternatives described later are faster.

Flip-flops usually take a single clock signal ϕ and locally generate its complement $\overline{\phi}$. 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. However, the clock load should be kept as small as possible because it has an activity factor of 1 and thus accounts for much of the power consumption in the flip-flop.

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 10.20(a) redraws Figure 10.19(a) with a built-in clock inverter. When ϕ falls, both the clock and its complement are momentarily low as shown in Figure 10.20(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 10.20(c) shows a C^{2}MOS dynamic flip-flop built using C^{2}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^{2}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

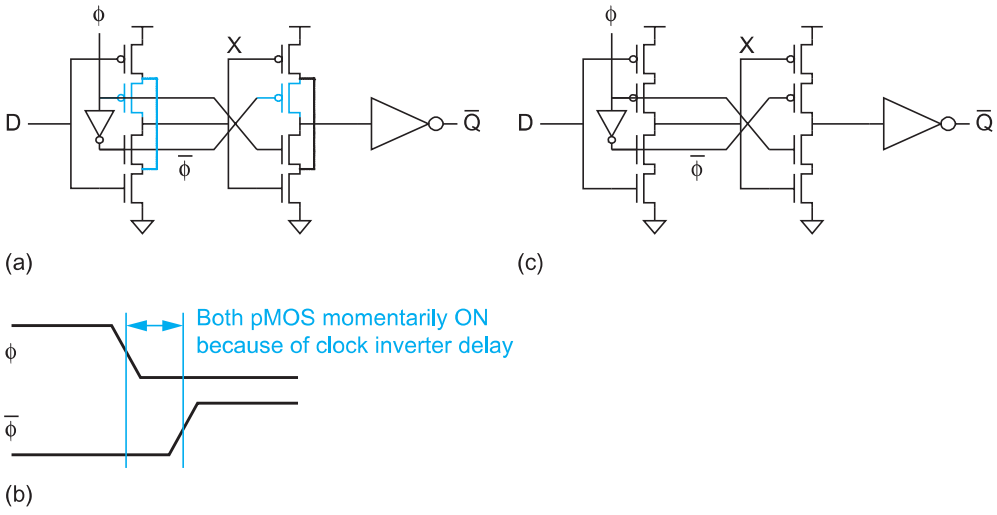

FIGURE 10.20 Transmission gate and NORA dynamic flip-flops

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 10.21. The flip-flop captures its input on the rising edge of ϕ_1. By making the nonoverlap large enough, the circuit will work despite large skews. However, the nonoverlap time is not used by logic, so it directly increases the setup time and sequencing overhead of the flip-flop (see Exercise 10.8). The layout for the flip-flop is shown on the inside front cover and is readily adapted to use a single clock. Observe how diffusion nodes are shared to reduce parasitic capacitance.

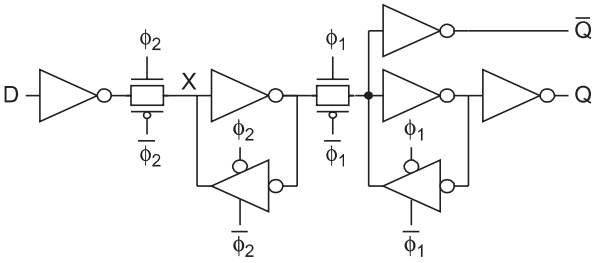

FIGURE 10.21 Flip-flop with two-phase nonoverlapping clocks

10.3.3 Pulsed Latches

A pulsed latch can be built from a conventional CMOS transparent latch driven by a brief clock pulse. Figure 10.22(a) shows a simple pulse generator, sometimes called a *clock chopper* or *one-shot* [Harris01a]. The pulsed latch is faster than a regular flip-flop because it involves a single latch rather than two and because it allows time borrowing. It can also consume less energy, although the pulse generator adds to the energy consumption (and is ideally shared across multiple pulsed latches for energy and area efficiency). The drawback is the increased hold time.

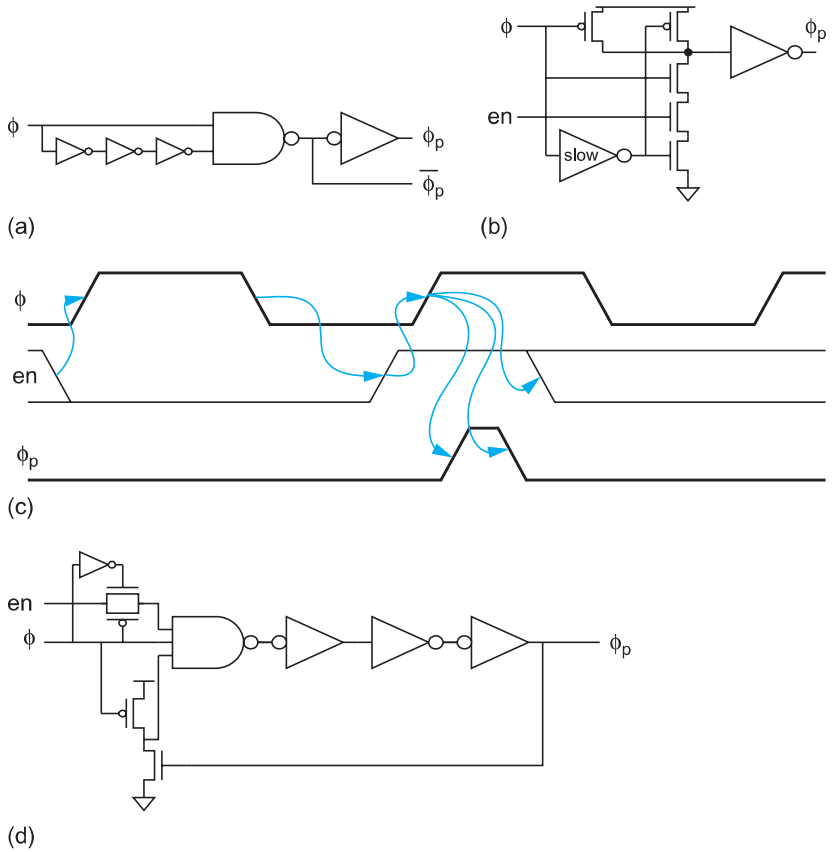

FIGURE 10.22 Pulse generators

The *Naffziger pulsed latch* used on the Itanium 2 processor consists of the latch from Figure 10.17(k) driven by even shorter pulses produced by the generator of Figure 10.22(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 10.22(c).

Figure 10.22(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 quad-core Itanium pulse generator selects between 1- and 3-inverter delay chains using a transmission gate multiplexer [Stackhouse09]. The wider pulse offers more robust latch operation across process and environmental variability and permits more time borrowing, but increases the hold time. The multiplexer select is software-programmable to fix problems discovered after fabrication.

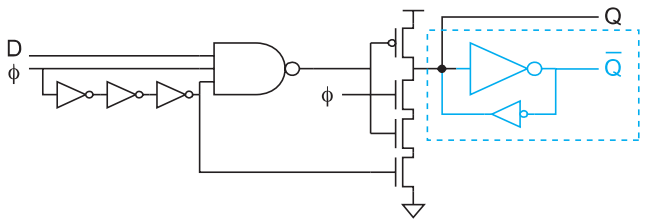

FIGURE 10.23 Partovi pulsed latch

The *Partovi pulsed latch* in Figure 10.23 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.

10.3.4 Resettable Latches and Flip-Flops

Most practical sequencing elements require a reset signal to enter a known initial state on startup and ensure deterministic behavior. Figure 10.24 shows latches and flip-flops with reset inputs. There are two types of reset: *synchronous* and *asynchronous*. Asynchronous reset forces Q 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 $\overline{reset}$. 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 of Figure 10.24 but replace NAND with NOR and $\overline{reset}$ with *set*. Figure 10.25 shows a flip-flop combining both asynchronous set and reset.

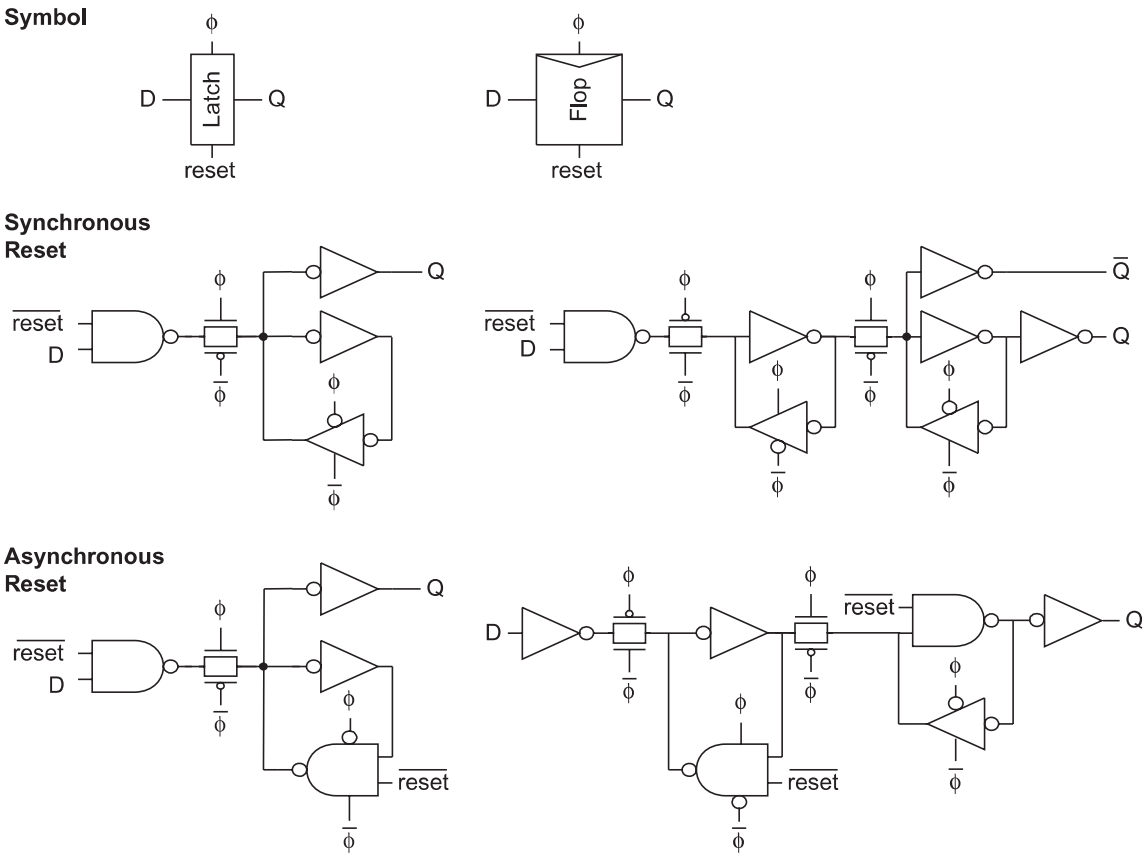

FIGURE 10.24 Resettable latches and flip-flops

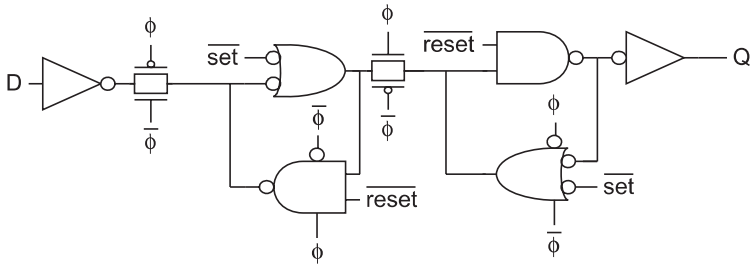

FIGURE 10.25 Flip-flop with asynchronous set and reset

10.3.5 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 10.26. 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