6 SEQUENTIAL CIRCUIT ELEMENTS

OVERVIEW

In this chapter we adapt our knowledge of combinational-logic circuits and physical gates to create a different type of logic circuit: latches and flip-flops. We create these because we need them to implement sequential-logic designs.

The solution to many problems requires a sequence of events to take place. In this way, time enters logic design. If the solution is available for use on command then it is in storage somewhere. Once stored, the solution can be used over and over again. A circuit that stores information in effect remembers the information.

Combinational-circuit outputs are simply functions of the present inputs. When the inputs change, so do the outputs. We need something new. If the inputs change, remembering requires an output to sustain itself. For this reason, feedback of the output to an input comes into play. We learn that feedback is remembering as we start with one AND gate and one OR gate. We discover this two-gate circuit (latch) has two stable states; that is, it is bistable. An input level becomes active and then inactive as the output goes to H voltage and stays there. Later, another input level becomes active and then inactive as the output goes to L voltage and stays there. And so we learn that latches respond to level changes. The first bistable circuit we study is the RS latch. We learn the defining equation, the related truth table, and the properties of the latch. Then we deal with the bistable latch.

However, latches have their intrinsic problems that make them unusable in many applications. The work that solved these problems gradually transformed the latch circuit into something known as the *edge-triggered flip-flop,* which responds to fast transitions, known as edges, from L to H or vice versa. That work, however, is not used today, so we simply skip over most of it. We study in detail the two types of commercially available flip-flops: the D flip-flop and the JK flip-flop. The T flip-flop is also studied. Then we apply out knowledge to build asynchronous ripple counters using D and JK flip-flops.

INTRODUCTION

In combinational-logic circuits, current outputs are functions of current inputs. There is no remembering or memory of the past. In sequential-logic circuits, there is memory of the past, so that present outputs are functions of the present and past inputs. Another way to put this is that the next state of a sequential circuit depends on both the present state and the present inputs.

"Memory" in a circuit means that the present state values remain constant until a change-to-the-next-state command is given. We need to introduce time as a variable in order to understand what the words *present* and *next* imply. The concept of *state* in a synchronous-state machine means essentially that the circuit will hold present values until the next clock edge occurs. Each clock edge is a change-to-the-next-state command in synchronous-state machines. The concept of *state* in an asynchronous-state machine means essentially that the circuit will hold present values until one or more inputs change. We focus only on synchronous machines at this time.

A modern sequential-circuit design process is set forth in the next chapter. There the business of how one moves from the present state to the next state is resolved. In order to design and build those sequential circuits we need to add to our tool kit the necessary sequential-circuit elements.

6.1 Feedback Is Remembering

Remembering requires feedback from output to input.

A combinational circuit acquires memory capability when a circuit output is connected (fed back) to a circuit input. The minimum circuit has one gate, which can be AND (Figure 6.1) or OR (Figure 6.2). The AND circuit does not remember that the input was high, as shown in the timing diagram, because when input h goes low the feedback loop is broken, and it cannot be restored. The AND circuit *does* remember that its input was low. The OR circuit provides a solution for high inputs. The timing diagram shows that q remains high after d goes low, which demonstrates that the loop is not broken. The circuit remembers that d was high in the past. The problem with this simple OR memory circuit is that we cannot reset it to q = L. So AND or OR alone are not satisfactory; however, together they are.

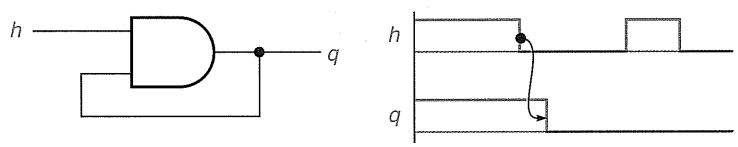


FIGURE 6.1
AND gate with feedback

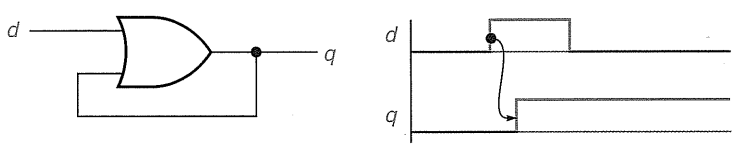


FIGURE 6.2
OR gate with feedback

Memory capability: one OR and one AND in a feedback loop.

OR provides memory (remembering) in a loop we cannot break, whereas we can break a feedback loop using AND. Let us combine them into one circuit by inserting the AND in the otherwise-unbreakable OR feedback loop (Figure 6.3). This circuit has two inputs, which we call data (d) and hold (h). When h is low, the loop is broken, so that output q follows data: $q = d$. We say the circuit is *transparent;* except for propagation delay, the transparent circuit can be omitted. *Propagation delay* is the time required for input changes to appear at an output. When h is high, the AND-OR circuit is in the hold mode (Figure 6.3) because q remains high after d goes low.

When h goes high, the value of d at that instant is *captured;* that is, the circuit remembers that value. The case when d is low as h goes high is illustrated in Figure 6.3. And later, when d goes high, q is driven high. The circuit can be *set* anytime h is high. When d goes low, the high value of d is remembered because the feedback loop is active. Still later, when h goes low, the feedback loop is broken again and q follows d. The circuit is *reset* because d happens to be low. This elementary circuit has many applications, one of which is the RS latch.

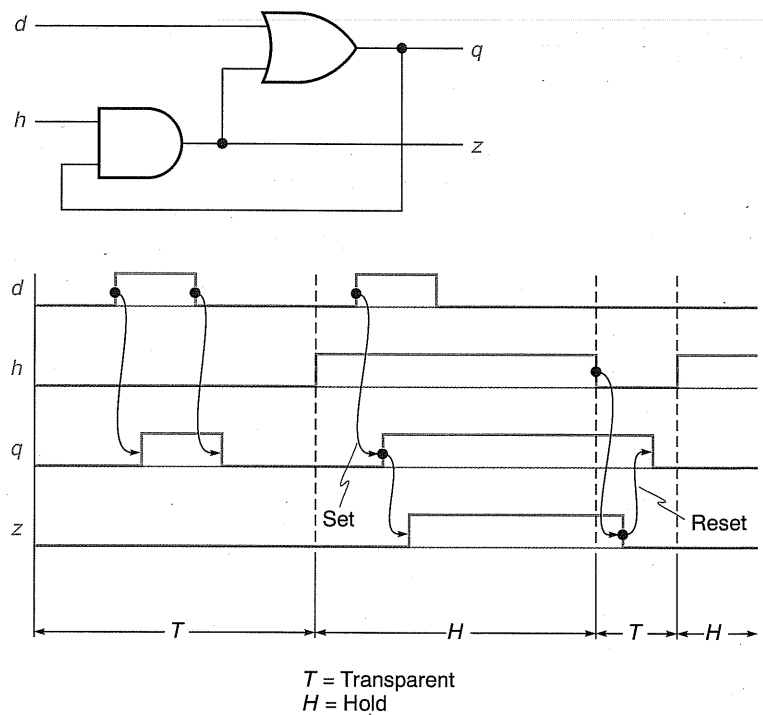


FIGURE 6.3
AND/OR gates with feedback

6.2 RS Latch

The RS (shorthand for "reset/set") latch has two simple forms: cross-coupled 00 gates (Figure 6.4) and cross-coupled 02 gates (Figure 6.5). Observe how mixed logic allows us to redraw the circuits as AND-OR pairs, revealing the function of each gate when we associate the s input with d (for "data") and the r input with h (for "hold") (Figure 6.3). Also note that q and z are complementary outputs. The latch has two stable states: one when $q = H$, and the other when $q = L$. To "prove" this we need only trace levels around the feedback loop. The stable states have the names *set* and *reset*. All of these facts follow from the defining equation.

The R and S input levels reset or set the latch.

In Figure 6.4 or 6.5 observe that

$$q = s + z \qquad \text{and} \qquad z = r'q$$

Therefore

$$q = s + r'q$$

In this equation q is an input (right side) and an output (left side). What does the equation with q on both sides mean?

Suppose one or more of the input variables r, s change. What is q? One answer uses the "right-now" (present) values of r, s, and q to evaluate $s + r'q$. Then the calculated value is assigned to the output q *after* changes to inputs propagate through the circuit. This new value of q is called q^+ (q-plus). We conclude that the left-side q does not equal the right-side q and that the left-side q is replaced by q^+.

So q^+ is the next state of q—next in the sense that q^+ is the output value when an input changes and after signal propagation times expire. (Signal propagation is created by changes in the inputs r or s.) The next-state equation for q^+ is evaluated for all r, s combinations of values in Figure 6.6. The latch functions listed in the table need to be verified by an H, L voltage analysis.

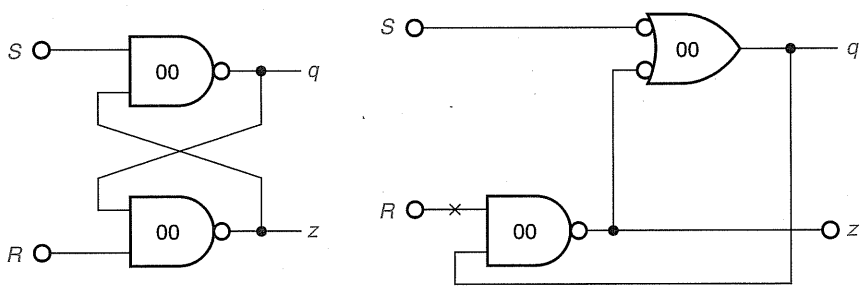


FIGURE 6.4
RS latch using 00 gates

FIGURE 6.5
RS latch using 02 gates

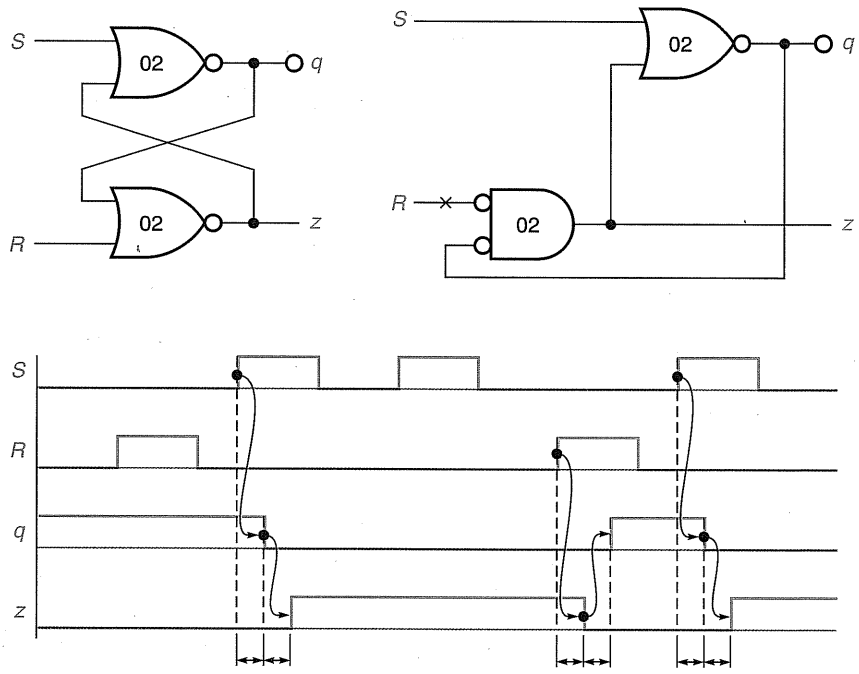


FIGURE 6.6
RS latch definition

Defining equation: $q^+ = s + r'q$

If:

$rs = 00$ the next state is q

$rs = 01$ the next state is 1

$rs = 10$ the next state is 0

$rs = 11$ the next state is 1

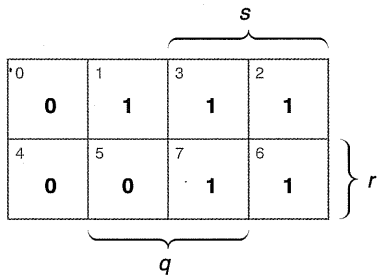
Excitation table (with q as a table-entered variable):

Row	r	s	q^+	Latch Function
0	0	0	q	hold
1	0	1	1	set
2	1	0	0	reset
3	1	1	1	(do not use)

FIGURE 6.7
Next-state K map for the RS latch

Next-state K map for $q^+ = s + r'q$:

$q^+(r, s, q)$

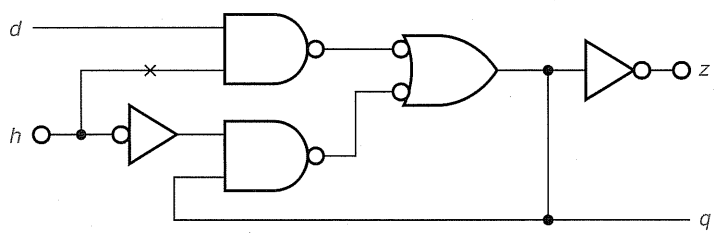


State of latch outputs depends upon past levels of the inputs.

Consider the 02 form of the RS latch (Figure 6.5). When both r and s are at the inactive L level, then the latch cannot change state. The latch *holds* the present value (and $z = q'$). When s is at the active H level while r is kept at L, the q output is driven active low and z is driven high, so that again $z = q'$. The latch is set. When r is at the active H level while s is kept at L, the z output is driven low and q is driven inactive high, so that once again $z = q'$. The latch is reset. When *both* r and s are at the active H level, then *both* outputs q and z are driven low and z does not equal q'. If *either* r or s falls to L, the complementary relationship of z and q is restored. If *both* r and s fall to low *simultaneously,* the next latch state is not predictable. The next state could be set or reset, or the latch could burst into oscillation. This is why the latch function entry is "do not use." In practice, the final value is determined by the last input to remain active. The r and s input signals provide *excitation* for the latch. The *excitation table* is generated by the defining equation, because it defines the output for various input excitations. The next-state K map for the RS latch is found in Figure 6.7.

Now we modify the circuit to get what we call the data-and-hold form of the RS latch (Figure 6.8). This form is the circuit of standard logic family members 75 and 375.

FIGURE 6.8
RS latch, DH format



EXERCISE 6.1

Derive the defining equation for the circuit in Figure 6.8. Make a truth table with d and q as table-entered variables.

Answer: $q^+ = hq + h'd$

h	q^+
0	d
1	q

■

6.3 Bistable Latch

Bistable latch has four modes: hold, set, reset, and toggle.

The RS latch does not have a *toggle function* (that is, the next state becomes the 1 state if the present state is 0, and vice versa). The equation for the toggle function is $q^+ = q'$. The following is one example of a toggle function.

> The caps-lock key on a keyboard is a toggle key. We press it to lock the keyboard in uppercase; we press it again to release to the upper/lowercase mode. Thus, the computer keyboard caps-lock key toggles the keyboard between the two typing modes.

The RS latch equation is $q^+ = s + r'q$. If $rs = 11$, then $q^+ = 1 + 0 \times q = 1$. If we want $q^+ = q'$ when $rs = 11$, then $q^+ = sq' + r'q$. Having made this change we need to verify by evaluating the modified equation that q^+ for the other three rs combinations (00, 01, and 10) has not changed. As shown in Figure 6.9, this is indeed the case. The bistable latch has four functions: hold, set, reset, and toggle. The next-state K map for the bistable latch is given in Figure 6.10.

We did not design the RS latch circuit; we just presented it. Nevertheless, given the defining equation, how *would* we design a bistable latch? The design of any latch circuit requires an asynchronous design

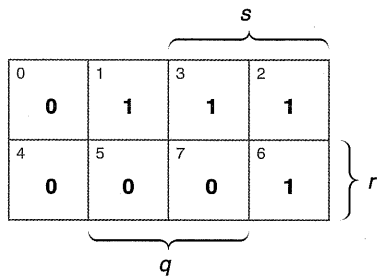
FIGURE 6.9
Bistable latch

Defining equation: $q^+ = sq' + r'q$

Excitation table (with q as a table-entered variable):

Row	r	s	q^+	Latch Function
0	0	0	q	hold
1	0	1	1	set
2	1	0	0	reset
3	1	1	q'	toggle

FIGURE 6.10
Next-state K map for the bistable latch



A latch is an asynchronous circuit.

process, which is not covered in detail in this text until Chapter 11. For now we will proceed intuitively.

First we synthesize the bistable q^+ equation as a combinational circuit, assuming there are no feedback connections (Figure 6.11). Then we add feedback lines to provide sources for the required active high q and active low q inputs, creating the bistable latch (Figure 6.12).

6.4 Sequential Circuit Timing

Synchronous circuits require a system clock to define precisely when events can occur. Events can occur once per clock period at a specified point in the clock period known as the *triggering edge*. A synchronous circuit performs reliably if inputs are stable for specified times (setup and hold) before and after every triggering edge. When these conditions are violated, unpredictable events can occur.

6.4.1 Clock

The *clock period* is the time between successive transitions in the same direction in a clock waveform (Figure 6.13a). The *clock frequency* is the number of successive transitions per second in the same direction in the clock waveform. Thus the clock frequency is the

FIGURE 6.11
Bistable circuit without feedback

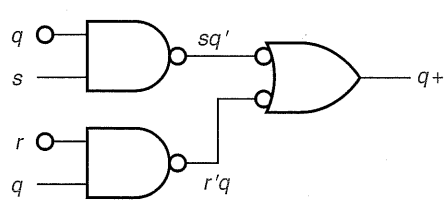
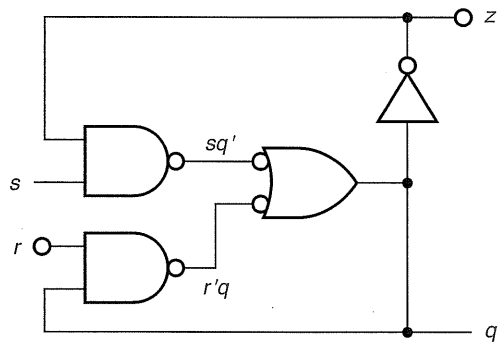


FIGURE 6.12
Bistable latch, RS format



reciprocal of the clock period. A useful abstraction of the clock waveform is a series of up arrows and down arrows representing corresponding clock transitions (Figure 6.13b). In a synchronous system, only one set of arrows (representing edges) is used to trigger events. The selected clock edges (up or down but not both) are referred to as the *active clock edges*.

Clock transitions can occur periodically or aperiodically. Clock transition rates range from single pulses initiated manually by pressing a push button to a maximum periodic rate set by hardware limitations (Figure 6.13c).

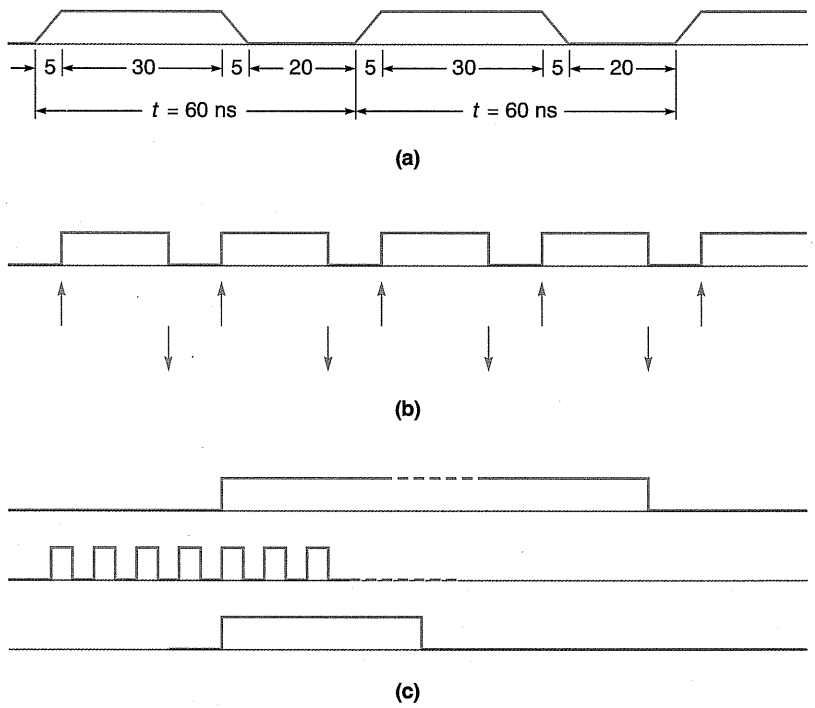
6.4.2 Setup Time and Hold Time

Synchronous circuits require input signals to meet *setup time and hold time* specifications. Failure to meet these specifications usually leads to uncertain next-state and output results. Reliable results follow when input variables are stable during setup and hold times.

Any flip-flop is an asynchronous circuit, because it is assembled from AND and OR gates.* An important assumption in the design of any flip-flop asynchronous circuit is that only one input changes at a time. In other words, the circuit must be quiescent before any input changes. From the point of view of an input, looking backward and forward in time, all other inputs must be quiescent. For example, if the *d* input to the D flip-flop discussed in the next section is changing, the clock input must be held constant, and vice versa. But for how long before and after any *d* input change must the clock input be held

* This may surprise you. In effect, setup time and hold time specifications allow us to think of, and use, edge-triggered flip-flops as if they themselves are synchronous circuits.

FIGURE 6.13
Clock timing diagrams



constant? *After* comes into play because an input change must propagate through the flip-flop circuit before another input changes. As it is doing so there can be no interferences from changes in other inputs starting to propagate through the circuit. In data books, *before* and *after* have the names t_{setup} (t_{su}) and t_{hold} (t_h).

The series of periodic up arrows in Figure 6.14 are an abstract representation of the clock waveform L-to-H transitions. The up arrows represent active clock edges. A time window defined by the parameters t_{su} and t_h straddles any active clock edge. A window is illustrated in Figure 6.14.

6.4.3 Synchronous Circuit Timing Diagrams

In a *synchronous sequential circuit* all flip-flops are clocked by the same clock signal. In principle, all flip-flop outputs change some time *after* every active clock edge. Any q waveform transitions occur after the active clock edge. The delay is the time required by the active clock edge to propagate through a flip-flop circuit and do its work. This is the active-clock-edge-to-output-q propagation time.

$f = 1/\tau$

Setup and hold times imply only one input changes at a time.

One clock drives all flip-flops in a synchronous circuit.

Actual setup time is clock period τ minus t_{PHL}.

When a high flip-flop output is switched to low by the active clock edge, the time delay from the active clock edge to the flip-flop output transition is the time to propagate and implement an H-to-L transition, or t_{PHL}. When a low flip-flop output is switched to high by the active clock edge, the time delay from the clock edge to the flip-flop output transition is the time to propagate and implement an L-to-H transition, or t_{PLH}. A realistic (delayed) output waveform is illustrated in Figure 6.15a. Waveforms like these that are synchronized to a clock are known as *synchronous waveforms*. In a sequential circuit, one flip-flop's output is in effect another flip-flop's input (Figure 6.15a). This is why, *within* a sequential circuit, all input and output waveforms are synchronous waveforms. (External inputs may be asynchronous. This complex issue is discussed in Section 6.5.3 and in Chapter 7.)

If we assume that the synchronous waveform in Figure 6.15a is an input to a flip-flop, then we can say that the *actual* hold time for the active clock edge ending clock cycle 1 is the propagation time delay t_{PLH}. And the *actual* hold time for the active clock edge ending clock cycle 2 is the propagation time delay t_{PHL}. The *actual* setup time in clock cycle 2 is the time interval from the L-to-H input waveform edge to the active clock edge ending cycle 2. Actual setup and hold times must exceed the flip-flop setup time and hold time requirements. The unavoidable propagation times t_{PHL} and t_{PLH} allow the system to work by providing holding time. A zero t_{PHL} or t_{PLH} provides zero hold time, which is unsatisfactory for most circuits.

When the clock frequency is increased, the period decreases until the period equals t_{setup} plus t_{hold}. This minimum possible clock period is illustrated in Figure 6.15b.

6.4.4 Metastability

Metastability is important to study because it can yield performance that is unreliable. A *metastable* state is a peculiar state of pseudoequilibrium in which the energy content of the state is either more than or less than that required for a stable state.

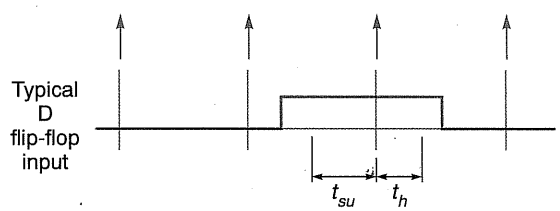
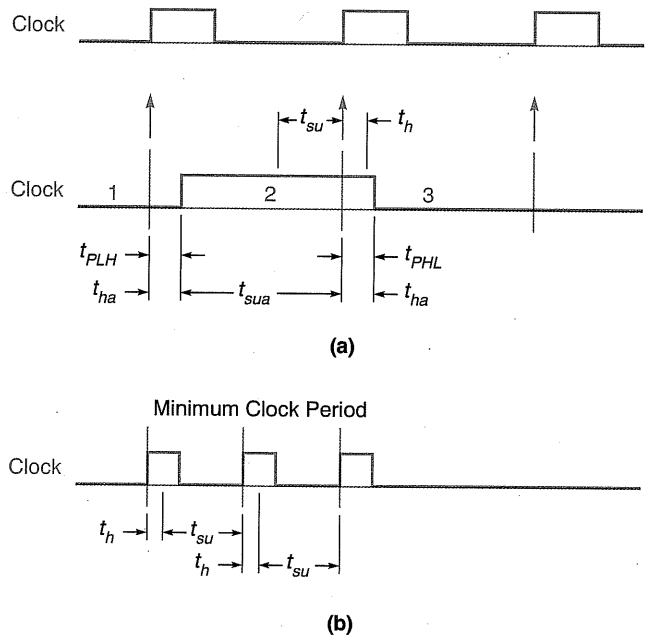


FIGURE 6.14
Setup and hold

FIGURE 6.15
Synchronous circuit timing diagram



The RS latch assembled with 00 gates is set when the s input is asserted by falling to the L level. Suppose the s input returns to the H level after time T has elapsed. The s waveform is now a *pulse* of duration T. The *energy* E in the pulse is essentially the dV = H − L voltage differential multiplied by the time T. As the pulse duration T decreases, the energy E decreases. At some point (T_1) the energy in the s pulse is insufficient to set the latch.

If the duration T is increased from below T_1, then the energy in the pulse increases; and at some point T_2 (greater than T_1) and beyond, the s pulse will set the latch. Pulses with duration in the range T_1 to T_2 may put the latch in a metastable state. A setup time specification guarantees prior-to-the-clock-edge input pulse durations greater than T_2. This is why asynchronous inputs must be synchronized.

Asynchronous inputs can induce metastability in a flip-flop.

The possibility of a latch's entering a metastable state is revealed by superposition of two inverter transfer functions. In effect, two cascaded inverters represent any latch's feedback loop. The inverter transfer function for inverters 1 and 2 is illustrated in Figure 6.16a. Two cascaded inverters establish relationships between their inputs and outputs: $V_{in1} = V_{out2}$, $V_{in2} = V_{out1}$. This is why their transfer functions may be superimposed, as shown in Figure 6.16b. The two

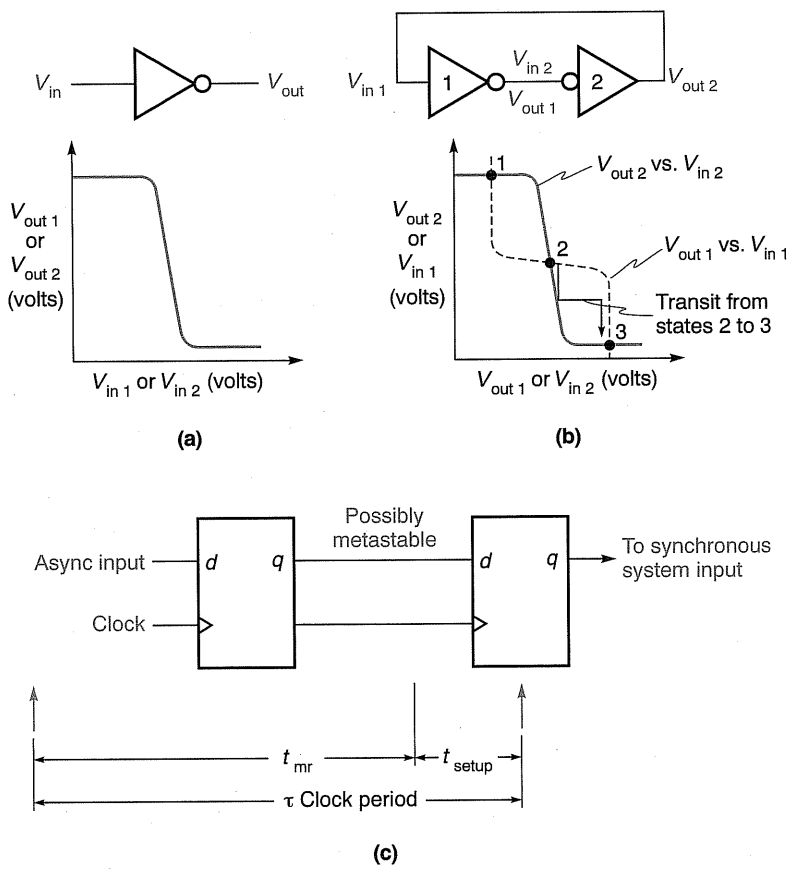
transfer functions intersect at three points: points 1 and 3 represent stable states set and reset, point 2 represents the metastable state. The intrinsic instability of point 2 is illustrated in Figure 6.16b by a slight perturbation of V_{in2}, which "jumps" the latch to stable point 3 (reset).

Once in a metastable state, the output can "oscillate" for an indefinite time, or suddenly switch after an indefinite time. The output becomes unpredictable. These events can generate what the uninformed perceive as mysterious random failures in a digital system. This is why, in practice, metastability is a source of disasters.

Synchronize all asynchronous inputs.

Two cascaded flip-flops can synchronize an asynchronous input if the clock period is sufficiently long in duration (Figure 6.16c). The parameter *metastable resolution time* (t_{mr}) is the "maximum" time a flip-flop will remain in a metastable state. If a metastable state is

FIGURE 6.16
Stable and metastable states



resolved in less than t_{mr} seconds, then a sufficiently long clock period is $t_{meta} + t_{setup}$. For more information on this complex subject, consult Chaney 1983.

6.5 Edge-Triggered Flip-Flops: General Comments

Let us assume that the bistable latch h_L input (see Figure 6.8) is a clock. When the clock is low, the latch holds the present value of q. The latch is in the *hold mode* when the clock is low. When the clock is high, the latch is in the *transparent mode: q* will follow changes in d, because the feedback loop is broken. The circuit is transparent in the sense that the output follows the input when the clock is high. For q not to change while the clock is high, input d must be held constant. Furthermore, any glitches (see Section 5.1.3, page 227) at the d input while the clock is high will be captured by q according to the glitch time duration and direction. The time constraint on d and the vulnerability to glitches are undesirable. This is why latches are replaced in many applications by edge-triggered flip-flops, which capture the value of d when the clock transitions from H to L (or vice versa). The vulnerability to glitches decreases enormously, and d need only have the correct value in a suitable time window that includes the transition.

Any flip-flop has two states, known as set and reset, and the symbol for the output is usually labeled q. The output q reports the state of the flip-flop, and so q is called a *state variable*. Clocked flip-flops are driven by a clock, which is a periodic digital waveform of frequency f and period $t = 1/f$. A 25-megahertz (MHz) waveform has a period of 40 nanoseconds (ns). A digital waveform has only levels H and L. Waveform transitions from H to L or L to H are called *waveform edges* or simply *edges*.

Edge-triggered flip-flops change state only at active clock edges.

Edge-triggered flip-flops are set and reset by either the negative-going (H to L) or the positive-going (L to H) clock waveform edges, but not both. Hence a clocked flip-flop is triggered only once per clock cycle. A 25-MHz clock triggers a flip-flop once per 40 ns. *Every* triggering edge (positive- or negative-going but not both) stores logical 0 or 1 in the flip-flop according to rules derived from the flip-flop defining equation. So a flip-flop state is changed only once per clock period, and the state is constant during each clock period. Edge-triggered flip-flops do not have transparent modes. This is basically why they are different from latches. The present inputs and the present state in the current clock period determine the next state in the next clock period. Finally, we note that flip-flop names are taken from their input terminals, such as D, JK, and T.

6.6 Edge-Triggered D Flip-Flop

D flip-flop defining equation is $q^+ = d$.

The edge-triggered D flip-flop has one input called *d* (for "data") and another input called *clock*. Some D flip-flops have one output line *q*. Others have two output lines for *q* that are voltage complements; if one is H, the other is L, and vice versa (Figure 6.17). The key fact to know about a flip-flop is the defining equation: everything is derived from that equation. Flip-flop inputs provide *excitation* for the flip-flop at each clock edge. The defining equation $q^+ = d$ is the source of the flip-flop excitation table in Figure 6.17.

In practice, flip-flops have additional (asynchronous) inputs that preset and clear the flip-flop. Preset (PRE) and clear (CLR) are aliases for set and reset. The commercial function (truth) table for the 74 includes these inputs and the clock, as shown in Figure 6.18. The bar over PRE, CLR, and one of the q symbols does not make these the complement operator. In data books the bar means the variable is given the active low (− −) assignment. (Data books make extensive use of mixed logic.) We do not use the bar in our circuit diagrams. The bubble at the corresponding input replaces the bar. This statement is verified in the first line of the table, which shows that the 74 is set to (q =) H when PRE is active low. The second line shows that an active low CLR resets the flip-flop to (q =) L. The third line reveals the indefinite RS latch behavior (see Figure 6.6) when both PRE and CLR are active low. The dashes mean "don't care." The clock (CLK) and the D input are disabled when either PRE or CLR is active. In the next two lines, with PRE and CLR inactive, the up arrow in the CLK column means the $q^+ = d$ equation is executed when the clock transitions from L to H (Q is q^+ in this table).

The equation $q^+ = d$ implies that *q* is a copy of the *d* waveform delayed by one clock period (Figure 6.19a). (A one-clock-period slip

FIGURE 6.17
D flip-flop (74LS74)

Defining equation: $q^+ = d$

Excitation table:

d	q^+	Function
0	0	reset
1	1	set

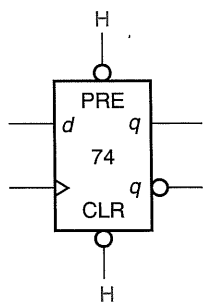


FIGURE 6.18
D flip-flop commercial function table

INPUTS				OUTPUTS		
$\overline{PRE}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$	
L	H	–	–	H	L	
H	L	–	–	L	H	
L	L	–	–	H[1]	H[1]	
H	H	↑	H	H	L	set
H	H	↑	L	L	H	reset
H	H	L	–	Q_0	$\overline{Q_0}$	hold

[1] This configuration is not stable.

D flip-flop output follows input with a one clock period delay.

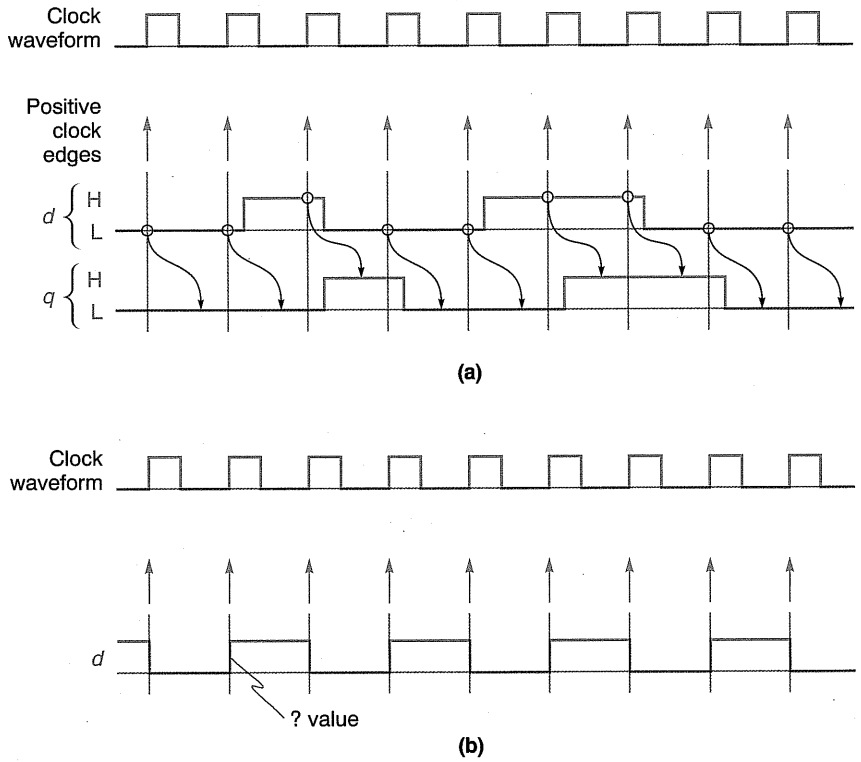
assumes the d input is synchronous with the clock.) Observe that there is no q^+ waveform, because in the next state q^+ becomes the present-state q. The value of d in clock period n determines the value of q in the next clock period, $n + 1$. Therefore, during clock period n, q^+ is shorthand for the value of q in clock period $n + 1$. The next-state q^+ is determined by the value of the d input at the time the L-to-H clock edge occurs. (The clock edge executes the q^+ equation.) The coincidence of these values and the clock edge are marked by circles in Figure 6.19a. They are the cause, and the arrows show the effect. (Some flip-flops are designed to have the negative-going H-to-L edge trigger the flip-flop.)

If t_{PHL} and t_{PLH} are zero, then the d and q waveform transitions are coincident with the clock edge (Figure 6.19b). The equation $q^+ = d$ in effect asks the question "What is the value of d at the clock edge?" Suppose d transitions from L to H coincident with the clock edge. What value does the circuit use when executing the q^+ equation? Is d low, or is it high? When propagation delays equal zero there is no definite answer. Showing the d and q transitions coincident with the clock edge has no meaning. This is why nonzero propagation delays are necessary for synchronous operation.

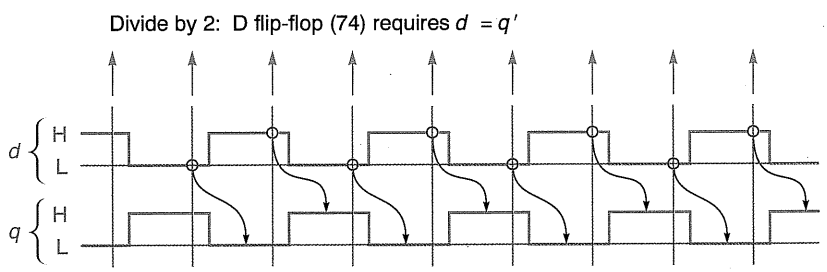
EXAMPLE 6.1 Divide-by-2 Circuit

One application of any flip-flop is dividing by 2, or toggling. To divide by 2 means to divide the clock frequency by 2, thereby doubling the clock period. Here is one way to deduce what connections must be

FIGURE 6.19
D flip-flop d and q waveforms



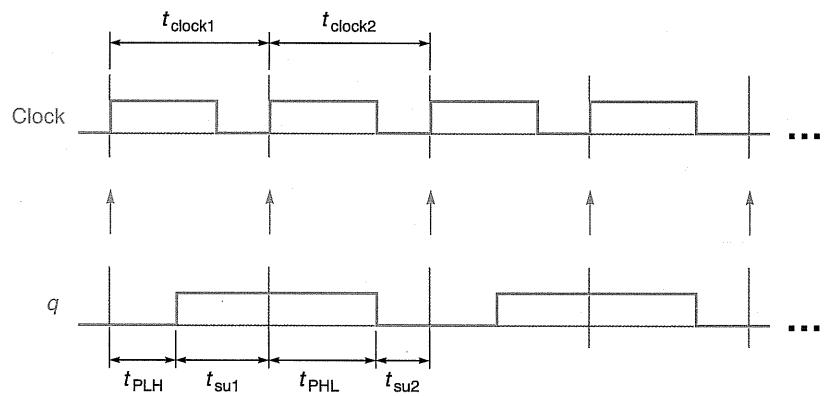
made so that a D flip-flop divides by 2. We start by drawing the desired digital waveform for q with a period twice the clock period. Given q and the defining equation $q^+ = d$, we deduce that the d waveform must be the same q waveform slipped back one clock period. (A one-clock-period slip assumes the d input is synchronous with the clock.) This conclusion is verified by inspection of each clock period (as a present state) of the d and q waveforms shown below. In each clock period the waveforms show that $d = q'$. Substituting for d, the defining equation



becomes $q^+ = q'$. That is, q-next is the complement of q-present, which is what we want.

EXAMPLE 6.2 Divide by 2: Calculating the Minimum Clock Period

The minimum clock period for reliable division by 2 is derived from the 74LS74 switching characteristics and from the clock, setup, and hold specifications. A drawing of the clock and q waveforms starts the process.



From the data book:

$t_{PLH} = 25$ ns max, $t_{PHL} = 40$ ns max, $t_{su} = 20$ ns min, $t_h = 5$ ns min

From the waveforms:

$$t_{clock1} = t_{PLH} + t_{su1} = 25 + 20 = 45 \quad \text{or} \quad t_{clock2} = t_{PHL} + t_{su2} = 40 + 20 = 60$$

Therefore t_{clock} must be greater than or equal to 60 ns in a divide-by-2 circuit.

Check:

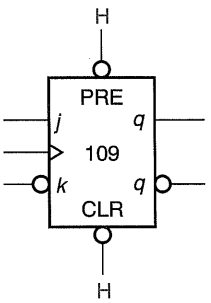
$$t_{clock\text{-}min} = 60 \text{ ns} \Rightarrow f_{max} = 16.667 \text{ MHz}$$

FIGURE 6.20
JK flip-flop

Defining equation: $q^+ = jq' + k'q$

Excitation table:

Row	+ + / + + j	+ + / − − k	q^+	Function
	← Assignments for 73, 112	← Assignments for 109		
0	0	0	q	hold
1	0	1	0	reset
2	1	0	1	set
3	1	1	q'	toggle



6.7 Edge-Triggered JK Flip-Flop

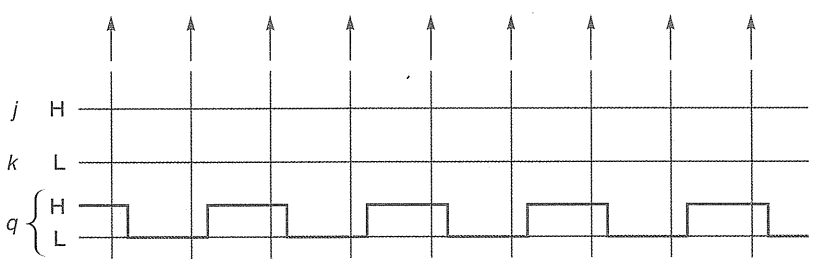
JK flip-flop defining equation is $q^+ = jq' + k'q$.

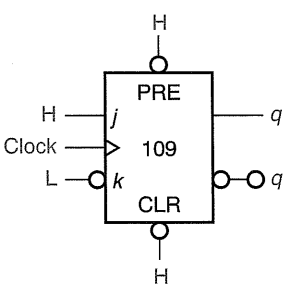
The JK flip-flop has two data inputs called j and k and a third input called *clock* (Figure 6.20). Some JK flip-flops have one output line q. Others have two output lines for q that are voltage complements; if one is H the other is L, and vice versa. The 109 is our preferred version of the JK flip-flop because input circuits are simplified in many applications when the active levels of j and k are different.

EXAMPLE 6.3 Divide by 2 with the JK Flip-Flop

The divide-by-2 equation is $q^+ = q'$. This is derived from the JK defining equation, $q^+ = jq' + k'q$, by setting $j = k = 1$ so that

$$q^+ = 1 \times q' + 1' \times q = q'$$





The JK truth table (Figure 6.21) is rearranged to emphasize other JK properties, which are applied in Chapter 8. When $q = 0$, the k input is disabled:

$$q^+ = j0' + k'0 = j1 + k'0 = j$$

When $q = 1$, the j input is disabled:

$$q^+ = j1' + k'1 = j0 + k'1 = k'$$

EXERCISE 6.2 Let $x = q_0'$ in the circuit in Figure P6.2 (page 326). Make a present-state, next-state (PS, NS) truth table where $q_1 q_0$ is the present state and $q_1^+ q_0^+$ is the next state. Derive the in-circuit defining equations.

Answer:

q_1	q_0	q_1^+	q_0^+		
0	0	0	1	$q_1^+ = q_0$	$q_0^+ = q_0'$
0	1	1	0		
1	0	0	1		
1	1	1	0		

■

EXERCISE 6.3 Refer to the circuit in Figure P6.8 (p. 327). Make a present-state, next-state (PS, NS) truth table where $q_1 q_0$ is the present state and $q_1^+ q_0^+$ is the next state. Derive the in-circuit defining equations.

Answer:

q_1	q_0	q_1^+	q_0^+		
0	0	0	1	$q_1^+ = q_1'$ xor q_0	$q_0^+ = q_0'$
0	1	1	0		
1	0	1	1		
1	1	0	0		

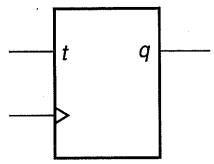
■

FIGURE 6.21
JK flip-flop revisited

Row	j	k	q	q^+	Function
0	0	0	0	0	$q^+ = j$
2	0	1	0	0	
4	1	0	0	1	
6	1	1	0	1	
1	0	0	1	1	$q^+ = k'$
3	0	1	1	0	
5	1	0	1	1	
7	1	1	1	0	

FIGURE 6.22
T flip-flop

Defining equation: $q^+ = tq' + t'q = t \text{ xor } q$



Excitation table:

Row	t	q	q^+	Function
0	0	0	0	hold value of q when t is False
1	0	1	1	$q^+ = q$
2	1	0	1	toggle q when t is True
3	1	1	0	$q^+ = q'$

With q as a table-entered variable:

Row	t	q	q^+	Function
0	0	0	q	hold
1	1	1	q'	toggle

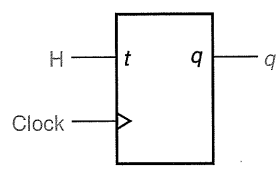
6.8 Edge-Triggered T Flip-Flop

The T flip-flop is the special case of the JK flip-flop where $t = j = k$ (Figure 6.22). There is no T flip-flop in the standard logic family. T stands for "toggle."

EXAMPLE 6.4 Divide-by-Two with the T Flip-Flop

The divide-by-2 equation, $q^+ = q'$, is derived from the T defining equation, $q^+ = tq' + t'q$, by setting $t = 1$.

$$q^+ = 1 \text{ xor } q = q' \quad \text{Toggle mode when } t = 1$$



6.9 Converting Flip-Flops

The standard logic family includes only edge-triggered D and JK flip-flops, because the D and JK are readily converted to other types. The conversions are guided by the defining equations.

Converting the JK flip-flop equation to the D flip-flop equation:

$$q^+ = jq' + k'q \quad \text{and} \quad q^+ = d = d(q' + q) = dq' + dq$$

imply

$$jq' + k'q = dq' + dq$$

Equating coefficients of q and q' we get $j = d$ and $k' = d$ (Figure 6.23a).

Converting the JK flip-flop equation to the T flip-flop equation:

$$q^+ = jq' + k'q \quad \text{and} \quad g^+ = tq' + t'q$$

imply

$$jq' + k'q = tq' + t'q$$

Equating coefficients of q and q' we get $j = t$ and $k' = t'$. Therefore $t = j = k$ (Figure 6.23b).

Converting the D flip-flop equation to the JK flip-flop equation:

$$q^+ = d \quad \text{and} \quad q^+ = jq' + k'q$$

imply

$$d = jq' + k'q$$

A conventional AND/OR circuit implements this equation for d. The AND/OR circuit is the circuit of a two-to-one multiplexer with q as the multiplexing variable (Figure 6.24a). Also, note that the d equation is in the form of a defining equation for a two-to-one mux with $a_0 = j$ and $a_1 = k'$.

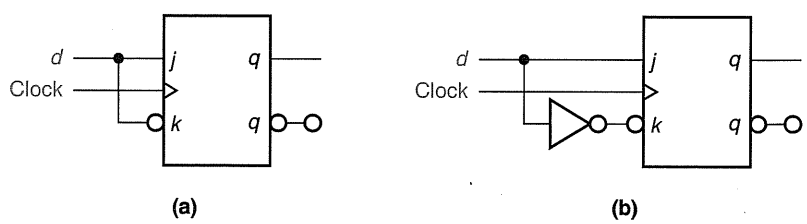


FIGURE 6.23
Converting a JK flip-flop to T and D flip-flops

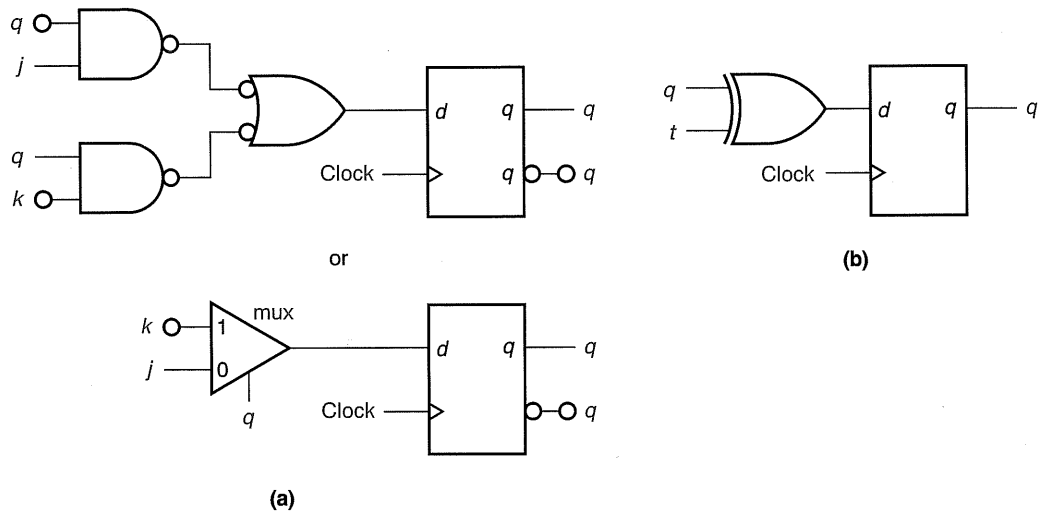


FIGURE 6.24
Converting a D flip-flop to JK and T flip-flops

Converting the D flip-flop equation to the T flip-flop equation:

$$q^+ = d \quad \text{and} \quad q^+ = tq' + t'q$$

imply

$$d = tq' + t'q$$

$$d = t \text{ xor } q$$

Here d is the xor output (Figure 6.24b).

6.10 Asynchronous Ripple Counters

Although synchronous counters (see Chapter 8) are far superior in performance, you should become aware of ripple counters.

A *digital counter* is a group of flip-flops wired to count clock edges, and to store in the flip-flops the number of clock edges counted. The counter number can be wired to be in any radix. We, however, are interested in the case in which the number is a radix-2 binary number. How do we wire a group of flip-flops to count up or down in binary? Chapter 7 presents a general method for answering such questions. At this point, however, let us proceed intuitively.

First we write down a sequence of binary numbers:

q_{3210}	Count by 1's	Count by 0's
0000	0	15 (decimal)
0001	1	14
0010	2	13
0011	3	12
0100	4 up	11 down
0101	5 ↓	10 ↓
0110	6	9
0111	7	8
1000	8	7
1001	9	6
1010	10	5
1011	11	4
1100	12	3
1101	13	2
1110	14	1
1111	15 (decimal)	0

Count increments when a variable changes from 1 to 0.

The list assigns $q_3q_2q_1q_0$ to be active high variables to count by 1's. The count increments by one as each clock edge is counted. The counter counts clock edges. However, if we use active low variables, in effect we focus on the zeros, and the count decrements by one as each clock edge is counted. Observe that the least significant digit q_0 toggles as clock edges are counted. Also, each time q_0 changes from 1 to 0, digit q_1 changes state (0 to 1 or 1 to 0). Furthermore, q_0's period is twice the clock period, and q_1's period is twice that of q_0. Observe also that each time q_1 changes from 1 to 0, digit q_2 changes state (0 to 1 or 1 to 0). Furthermore, q_2's period is twice q_1's period, just as q_1's period is twice that of q_0. Finally, observe that q_3's period is twice q_2's period.

So we say that

clock divided by 2 produces q_0

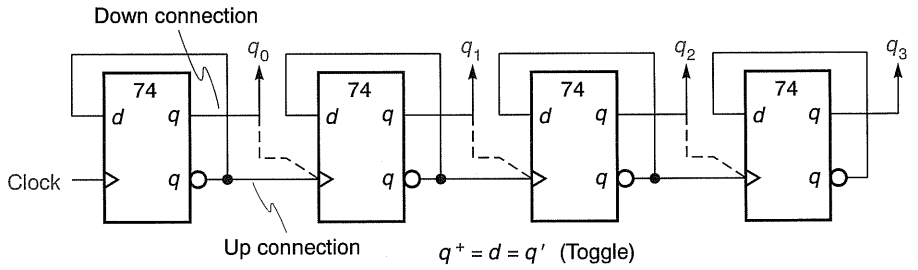
q_0 divided by 2 produces q_1

q_1 divided by 2 produces q_2

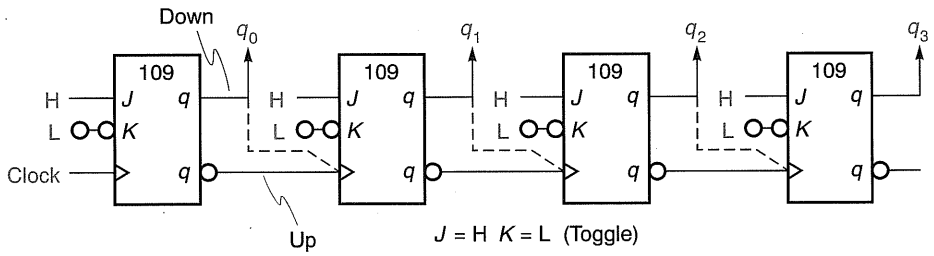
q_2 divided by 2 produces q_3

and we conclude that one way to wire a group of flip-flops as a binary

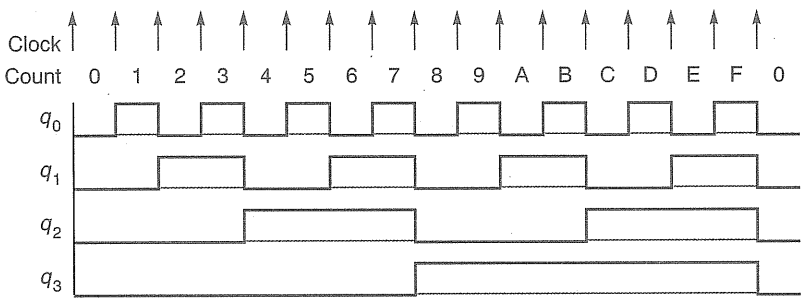
FIGURE 6.25
Ripple counters



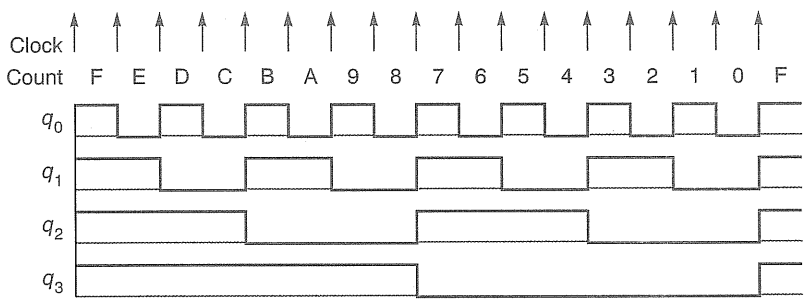
(a) *D* flip-flop ripple counter (asynchronous)



(b) *JK* flip-flop ripple counter (asynchronous)



(c) Up waveforms



(d) Down waveforms

counter is to use a cascade of divide-by-2 circuits. The cascade of dividers may be wired to count up or down. A cascade may use D flip-flops (Fig. 6.25a) or JK flip-flops (Figure 6.25b). The cascade of dividers is called a ripple counter because higher-order digits cannot change state until the lower-order digits change state and the change of state propagates through the cascade. Now, are we counting by 1's or by 0's?

The q_{j+1} waveforms (Figure 6.25c) for an up counter change state when the q_j waveforms transition from H to L, but the D flip-flops trigger on the L-to-H transitions. The active low q_j flip-flop outputs transition from L to H when the active high q_j outputs transition from H to L. Thus the active low q of any bit should be the clock input to the next bit. The input clock is not changed.

Count decrements when a variable changes from 0 to 1.

Down-counter waveforms (Figure 6.25d) change state when the q_j waveforms go from L to H. Thus the active high q output of any bit should be the clock input to the next bit. Figures 6.25a and 6.25b show the up and down ripple counter connections for D and JK ripple counters, respectively.

SUMMARY

Latch

Defining equations:

RS $q^+ = s + r'q$

Bistable $q^+ = sq' + r'q$

Timing

Any flip-flop has two states, known as set and reset; the symbol for the output is usually labeled q. Since the output q reports the state of the flip-flop, q is called a state variable.

Synchronous circuits require input signals to meet setup time and hold time specifications. A time window defined by the parameters t_{setup} and t_{hold} straddles any clock edge. A window is illustrated in Figure 6.14. Actual setup and hold times must exceed the flip-flop setup and hold time requirements.

In a synchronous sequential circuit all flip-flops are clocked by the same clock signal. In principle, all flip-flop outputs change some time after every clock edge. All q waveform transitions occur after the clock edge. The delay is the time required by the clock edge to propagate through a flip-flop circuit and do its work. This is the clock-edge-to-output-q propagation time. The unavoidable propagation times t_{PHL} and t_{PLH} allow the system to work by providing holding time. A zero t_{PHL} or t_{PLH} provides zero hold time, which is unsatisfactory for most circuits.

Edge-Triggered Flip-flops

The defining equation is the next-state equation. The defining equation calculates the next state given both the present state and the inputs.

Defining equations:

D $q^+ = d$

JK $q^+ = jq' + k'q$

T $q^+ = tq' + t'q = t \text{ xor } q$

Excitation tables:

D

Row	d	q^+	Function
0	0	0	reset
1	1	1	set

JK

Row	j	k	q^+	Function
0	0	0	q	hold
1	0	1	0	reset
2	1	0	1	set
3	1	1	q'	toggle

T

Row	t	q^+	Function
0	0	q	hold
1	1	q'	toggle

Asynchronous Ripple Counters

A digital counter is a group of flip-flops wired to count rising or falling clock edges. The counter's purpose is to store the number of clock edges counted. The counter number can be wired to be in any radix.

Ripple counters exist because their circuits are simple. This rationale no longer justifies designing them, however, because synchronous