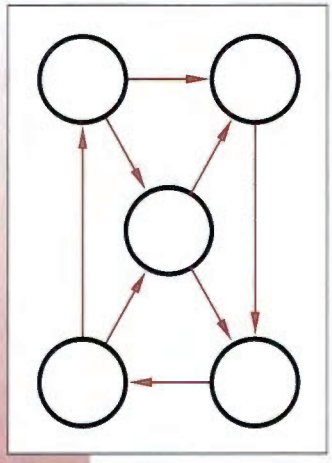


chapter 7

Sequential Logic Design Principles

Logic circuits are classified into two types, "combinational" and "sequential." A *combinational* logic circuit is one whose outputs depend only on its current inputs. The heater-fan-speed selection knob in an older car is like a combinational circuit. Its "output" selects a fan speed based only on its current "input"—the position of the knob.

A *sequential* logic circuit is one whose outputs depend not only on its current inputs, but also on the past sequence of inputs, possibly arbitrarily far back in time. The circuit controlled by up and down fan-speed buttons in a newer car is a sequential circuit—the current fan speed depends on an arbitrarily long sequence of up/down pushes, beginning when you first turned on the heater.

So it is inconvenient, and often impossible, to describe the behavior of a sequential circuit using a table that lists outputs as a function of a limited-length input sequence that has been received up until the current time. The circuit may have been operating for a *long* time, so there is no practical bound. With the sequential fan-speed circuit, it's not always possible to determine the current fan speed by looking only at a predetermined number of previous up/down pushes, whether that number is 1, 10, or 1000; the circuit may have received even more up/down pushes.

Instead, we can determine what the output of the fan-speed circuit should be if we know its current "state," and this can be done very concisely.

The best definition of "state" that I've seen appeared in Herbert Hellerman's book on *Digital Computer System Principles* (McGraw-Hill, 1967):

state
state variable

> The *state* of a sequential circuit is a collection of *state variables* whose values at any one time contain all the information about the past necessary to account for the circuit's future behavior.

In the present example, the fan speed is the current state. Inside a three-speed fan, this state might be stored as two binary state variables representing a decimal number between 0 and 3, with 0 corresponding to "off" and 3 to the highest speed. Given the current state (speed 0–3), we can always predict the next state as a function of the inputs (presses of the up/down pushbuttons).

Another example of a sequential circuit is the channel selector on a typical TV using, say, 7 bits to encode 128 channels. Here, one highly visible output of the circuit is an encoding of the state itself—the channel-number display. Other outputs, internal to the TV, may be combinational functions of the state alone (e.g., VHF/UHF/cable tuner selection) or of both state and input (e.g., turning off the TV if the current state is 0 or 1 and the "down" button is pressed).

State variables need not have direct physical significance, and there are often many ways to choose them to describe a particular sequential circuit. For example, in the TV-channel selector, instead of using 7 bits, the state might be stored as three BCD digits (12 bits), with many of the 4096 possible bit combinations going unused.

In a digital logic circuit, state variables are binary values, corresponding to certain logic signals in the circuit, as we'll see in later sections. A circuit with n binary state variables has 2^n possible states. As large as it might be, 2^n is always finite, never infinite, so sequential circuits are sometimes called *finite-state machines*.

finite-state machine

The state changes of most sequential circuits occur at times specified by a free-running *clock* signal. Figure 7-1 gives timing diagrams and nomenclature for typical clock signals. By convention, a clock signal is active high if state changes occur at the clock's rising edge or when the clock is HIGH, and active low in the complementary case. The *clock period* is the time between successive transitions in the same direction, and the *clock frequency* is the reciprocal of the period. The first edge or pulse in a clock period or sometimes the period itself is called a *clock tick*. The *duty cycle* is the percentage of time that the clock signal

clock
clock period
clock frequency
clock tick
duty cycle

NON-FINITE-STATE MACHINES

A group of mathematicians recently proposed a non-finite-state machine, but they're still busy listing its states. . . . Sorry, that's just a joke. There *are* mathematical models for infinite-state machines, such as Turing machines. They typically contain a small finite-state-machine control unit, and an infinite amount of auxiliary memory, such as an endless tape.

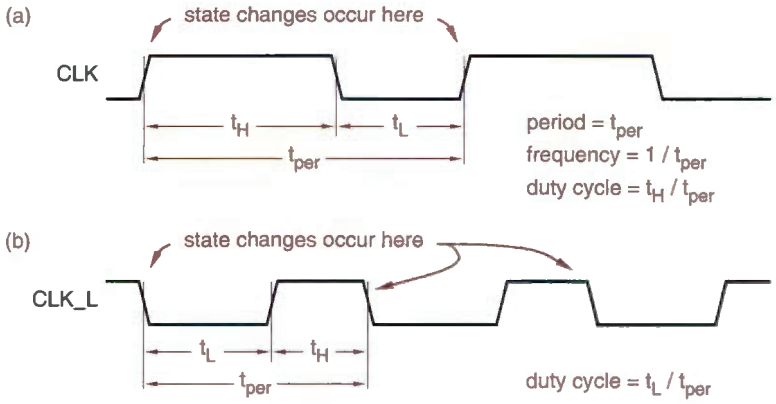


Figure 7-1
Clock signals: (a) active high; (b) active low.

is at its asserted level. Typical digital systems, from digital watches to supercomputers, use a quartz-crystal oscillator to generate a free-running clock signal. Clock frequencies might range from 32.768 kHz (for a watch) to 4 GHz (for a CMOS microprocessor with a cycle time of 250 ps); "typical" systems using TTL and CMOS parts have clock frequencies in the 5–500 MHz range.

In this chapter we'll discuss two types of sequential circuits that account for the majority of practical discrete designs. A *feedback sequential circuit* uses ordinary gates and feedback loops to obtain memory in a logic circuit, thereby creating sequential-circuit building blocks such as latches and flip-flops that are used in higher-level designs. A *clocked synchronous state machine* uses these building blocks, in particular edge-triggered D flip-flops, to create circuits whose inputs are examined and whose outputs change in accordance with a controlling clock signal. Other sequential circuit types, such as general fundamental mode, multiple-pulse mode, and multiphase circuits, are sometimes useful in high-performance systems and VLSI and are discussed in advanced texts.

feedback sequential circuit

clocked synchronous state machine

7.1 Bistable Elements

The simplest sequential circuit consists of a pair of inverters forming a feedback loop, as shown in Figure 7-2. It has *no* inputs and two outputs, Q and Q_L.

7.1.1 Digital Analysis

The circuit of Figure 7-2 is often called a *bistable*, since a strictly digital analysis shows that it has two stable states. If Q is HIGH, then the bottom inverter has a HIGH input and a LOW output, which forces the top inverter's output HIGH as we assumed in the first place. But if Q is LOW, then the bottom inverter has a LOW input and a HIGH output, which forces Q LOW, another stable situation. We could use a single state variable, the state of signal Q, to describe the state of the circuit; there are two possible states, Q = 0 and Q = 1.

bistable

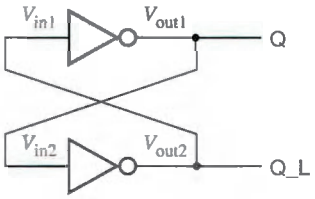


Figure 7-2
A pair of inverters forming a bistable element.

The bistable element is so simple that it has no inputs and therefore no way of controlling or changing its state. When power is first applied to the circuit, it randomly comes up in one state or the other and stays there forever. Still, it serves our illustrative purposes very well, and we *will* actually show a couple of applications for it in Sections 8.2.3 and 8.2.4.

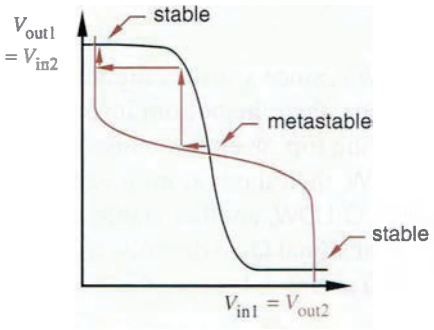
7.1.2 Analog Analysis

The analysis of the bistable has more to reveal if we consider its operation from an analog point of view. The dark line in Figure 7-3 shows the steady-state (DC) transfer function T for a single inverter; the output voltage is a function of input voltage, $V_{out} = T(V_{in})$. With two inverters connected in a feedback loop as in Figure 7-2, we know that $V_{in1} = V_{out2}$ and $V_{in2} = V_{out1}$; therefore, we can plot the transfer functions for both inverters on the same graph with an appropriate labeling of the axes. Thus, the dark line is the transfer function for the top inverter in Figure 7-2, and the colored line is the transfer function for the bottom one.

Considering only the steady-state behavior of the bistable's feedback loop, and not dynamic effects, the loop is in equilibrium if the input and output voltages of both inverters are constant DC values consistent with the loop connection and the inverters' DC transfer function. That is, we must have

$$\begin{aligned} V_{in1} &= V_{out2} \\ &= T(V_{in2}) \\ &= T(V_{out1}) \\ &= T(T(V_{in1})) \end{aligned}$$

Figure 7-3
Transfer functions for inverters in a bistable feedback loop.



Transfer function:

$V_{out1} = T(V_{in1})$

$V_{out2} = T(V_{in2})$

Likewise, we must have

$$V_{in2} = T(T(V_{in2}))$$

We can find these equilibrium points graphically from Figure 7-3; they are the points at which the two transfer curves meet. Surprisingly, we find that there are not two but *three* equilibrium points. Two of them, labeled *stable*, correspond to the two states that our "strictly digital" analysis identified earlier, with Q either 0 (LOW) or 1 (HIGH).

stable

The third equilibrium point, called a *metastable state*, occurs with V_{out1} and V_{out2} about halfway between a valid logic 1 voltage and a valid logic 0 voltage; so Q and Q_L are not valid logic signals at this point. Yet the loop equations are satisfied; if we can get the circuit to operate at the metastable point, it could theoretically stay there indefinitely. This behavior is called *metastability.*

metastable state

metastability

7.1.3 Metastable Behavior

Closer analysis of the situation at the metastable point shows that it is aptly named. It is not truly stable, because random noise will tend to drive a circuit that is operating at the metastable point toward one of the stable operating points, as we'll now demonstrate.

Suppose the bistable is operating precisely at the metastable point in Figure 7-3. Now let us assume that a small amount of circuit noise reduces V_{in1} by a tiny amount. This tiny change causes V_{out1} to *increase* by a small amount. But since V_{out1} produces V_{in2}, we can follow the first horizontal arrow from near the metastable point to the second transfer characteristic, which now demands a lower voltage for V_{out2}, which is V_{in1}. Now we're back where we started, except we have a much larger change in voltage at V_{in1} than the original noise produced, and the operating point is still changing. This "regenerative" process continues until we reach the stable operating point at the upper lefthand corner of Figure 7-3. However, if we perform a "noise" analysis for either of the stable operating points, we find that feedback brings the circuit back toward the stable operating point, rather than away from it.

Metastable behavior of a bistable can be compared to the behavior of a ball dropped onto a hill, as shown in Figure 7-4. If we drop a ball from overhead, it will probably roll down immediately to one side of the hill or the other. But if it lands right at the top, it may precariously sit there for a while before random

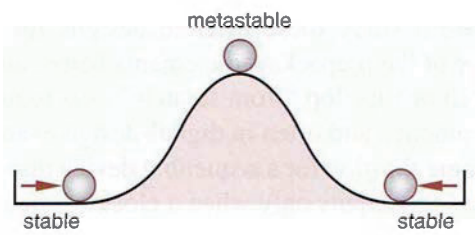


Figure 7-4
Ball and hill analogy for metastable behavior.

forces (wind, rodents, earthquakes) start it rolling down the hill. Like the ball at the top of the hill, the bistable may stay in the metastable state for an unpredictable length of time before nondeterministically settling into one stable state or the other.

If the *simplest* sequential circuit is susceptible to metastable behavior, you can be sure that *all* sequential circuits are susceptible. And this behavior is not something that only occurs at power-up.

Returning to the ball-and-hill analogy, consider what happens if we try to kick the ball from one side of the hill to the other. Apply a strong force (Superman), and the ball goes right over the top and lands in a stable resting place on the other side. Apply a weak force (Clark Kent), and the ball falls back to its original starting place. But apply a wishy-washy force (Charlie Brown), and the ball goes to the top of the hill, teeters, and eventually falls back to one side or the other.

This behavior is completely analogous to what happens to latches and flip-flops under marginal triggering conditions. For example, we'll soon study S-R latches, where a pulse on the S input forces the latch from the 0 state to the 1 state. A minimum pulse width is specified for the S input. Apply a pulse of this width or longer, and the latch immediately goes to the 1 state. Apply a very short pulse, and the latch stays in the 0 state. Apply a pulse just under the minimum width, and the latch may go into the metastable state. Once the latch is in the metastable state, its operation depends on "the shape of its hill." Latches and flip-flops built from high-gain, fast transistors tend to come out of metastability faster than ones built from low-performance technologies.

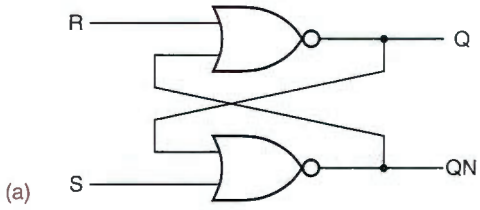
We'll say more about metastability in the next section in connection with specific latch and flip-flop types, and in Section 8.9 with respect to synchronous design methodology and synchronizer failure.

7.2 Latches and Flip-Flops

Latches and flip-flops are the basic building blocks of most sequential circuits. Typical digital systems use latches and flip-flops that are prepackaged, functionally specified devices in a standard integrated circuit. In ASIC design environments, latches and flip-flops are typically predefined cells specified by the ASIC vendor. However, within a standard IC or an ASIC, each latch or flip-flop cell is typically designed as a feedback sequential circuit using individual logic gates and feedback loops. We'll study these discrete designs for two reasons—to understand the behavior of the prepackaged elements better, and to gain the capability of building a latch or flip-flop "from scratch," as is required very occasionally in digital-design practice and often in digital-design exams.

flip-flop

All digital designers use the name *flip-flop* for a sequential device that normally samples its inputs and changes its outputs only when a clocking signal is



(b)

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

Figure 7-5
S-R latch: (a) circuit design using NOR gates; (b) function table.

changing. On the other hand, most digital designers use the name *latch* for a sequential device that watches its inputs continuously and can change its outputs at any time (although in some cases requiring an enable signal to be asserted). We follow this standard convention in this text. However, some textbooks and digital designers may (incorrectly) use the name "flip-flop" for a device that we call a "latch."

latch

In any case, because the functional behaviors of latches and flip-flops are quite different, it is important for the logic designer to know which type is being used in a design, either from the device's part number (e.g., 74x373 vs. 74x374) or from other contextual information. We discuss the most commonly used types of latches and flip-flops in the following subsections.

7.2.1 S-R Latch

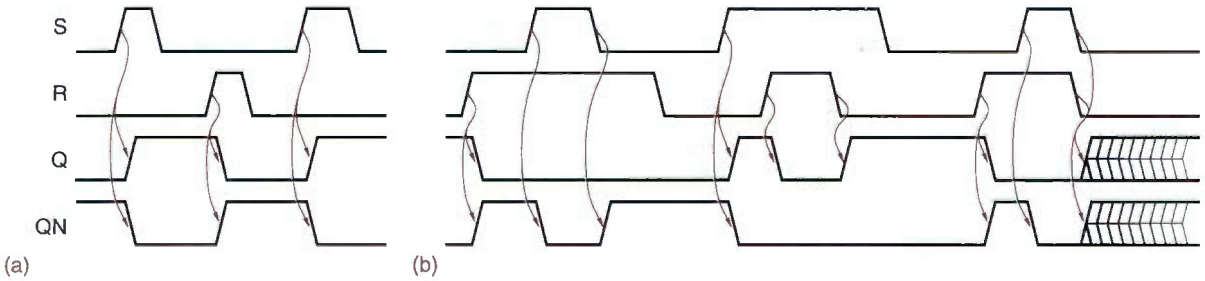
An *S-R (set-reset) latch* based on NOR gates is shown in Figure 7-5(a). The circuit has two inputs, S and R, and two outputs, labeled Q and QN, where QN is normally the complement of Q. Signal QN is sometimes labeled $\overline{Q}$ or Q_L.

S-R latch

If S and R are both 0, the circuit behaves like the bistable element—we have a feedback loop that retains one of two logic states, Q = 0 or Q = 1. As shown in Figure 7-5(b), either S or R may be asserted to force the feedback loop to a desired state. S *sets* or *presets* the Q output to 1; R *resets* or *clears* the Q output to 0. After the S or R input is negated, the latch remains in the state that it was forced into. Figure 7-6(a) shows the functional behavior of an S-R latch for a typical sequence of inputs. Colored arrows indicate causality, that is, which input transitions cause which output transitions.

set
preset
reset
clear

Figure 7-6 Typical operation of an S-R latch: (a) "normal" inputs; (b) S and R asserted simultaneously.



$\overline{Q}$ VERSUS QN In most applications of an S-R latch, the QN (a.k.a. $\overline{Q}$) output is always the complement of the Q output. However, the $\overline{Q}$ name is not quite correct, because there is one case where this output is not the complement of Q. If both S and R are 1, as they are in several places in Figure 7-6(b), then both outputs are forced to 0. Once we negate either input, the outputs return to complementary operation as usual. However, if we negate both inputs simultaneously, the latch goes to an unpredictable next state, and it may in fact oscillate or enter the metastable state. Metastability may also occur if a 1 pulse that is too short is applied to S or R.

Three different logic symbols for the same S-R latch circuit are shown in Figure 7-7. The symbols differ in the treatment of the complemented output. Historically, the first symbol was used, showing the active-low or complemented signal name inside the function rectangle. However, in bubble-to-bubble logic design the second form of the symbol is preferred, showing an inversion bubble outside the function rectangle. The last form of the symbol is obviously wrong.

propagation delay

Figure 7-8 defines timing parameters for an S-R latch. The *propagation delay* is the time it takes for a transition on an input signal to produce a transition on an output signal. A given latch or flip-flop may have several different propagation-delay specifications, one for each pair of input and output signals. Also, the propagation delay may be different depending on whether the output makes a LOW-to-HIGH or HIGH-to-LOW transition. With an S-R latch, a LOW-to-HIGH transition on S can cause a LOW-to-HIGH transition on Q, so a propagation delay $t_{pLH(SQ)}$ occurs, as shown in transition 1 in the figure. Similarly, a LOW-to-HIGH transition on R can cause a HIGH-to-LOW transition on Q, with propagation delay $t_{pHL(RQ)}$ as shown in transition 2. Not shown in the figure are the corresponding transitions on QN, which would have propagation delays $t_{pHL(SQN)}$ and $t_{pLH(RQN)}$.

minimum pulse width

Minimum-pulse-width specifications are usually given for the S and R inputs. As shown in Figure 7-8, the latch may go into the metastable state and remain there for a random length of time if a pulse shorter than the minimum width $t_{pw(min)}$ is applied to S or R. The latch can be deterministically brought out of the metastable state only by applying a pulse to S or R that meets or exceeds the minimum-pulse-width requirement.

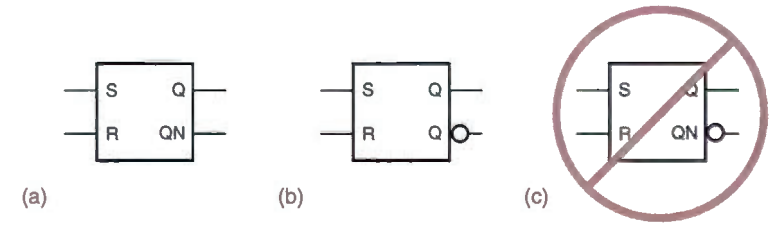


Figure 7-7
Symbols for an S-R latch: (a) without bubble; (b) preferred for bubble-to-bubble design; (c) incorrect because of double negation.

HOW CLOSE IS CLOSE?

As mentioned in the previous boxed note, an S-R latch may go into the metastable state if S and R are negated simultaneously. Often, but not always, a commercial latch's specifications define "simultaneously" (e.g., S and R negated within 10 ns of each other). This parameter is sometimes called the *recovery time*, t_{rec}. It is the minimum delay between negating S and R for them to be considered nonsimultaneous and it is closely related to the minimum-pulse-width specification. Both specifications are measures of how long it takes for the latch's feedback loop to stabilize during a change of state.

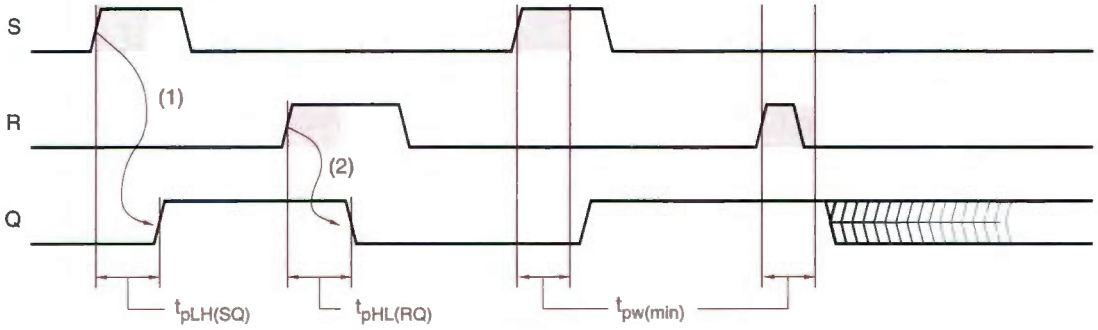


Figure 7-8 Timing parameters for an S-R latch.

7.2.2 $\overline{S}$-$\overline{R}$ Latch

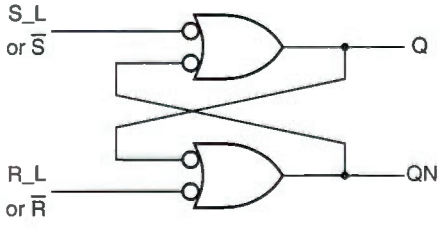
S-R latch

An $\overline{S}$-$\overline{R}$ *latch* (read "S-bar-R-bar latch") with active-low set and reset inputs may be built from NAND gates as shown in Figure 7-9(a). In CMOS and TTL logic families, $\overline{S}$-$\overline{R}$ latches are used much more often than S-R latches because NAND gates are preferred over NOR gates.

As shown by the function table, Figure 7-9(b), operation of the $\overline{S}$-$\overline{R}$ latch is similar to that of the S-R, with two major differences. First, $\overline{S}$ and $\overline{R}$ are active low, so the latch remembers its previous state when $\overline{S} = \overline{R} = 1$; the active-low inputs are clearly indicated in the symbols in (c). Second, when both $\overline{S}$ and $\overline{R}$ are asserted simultaneously, both latch outputs go to 1, not 0 as in the S-R latch. Except for these differences, operation of the $\overline{S}$-$\overline{R}$ is the same as the S-R, including timing and metastability considerations.

Figure 7-9 $\overline{S}$-$\overline{R}$ latch: (a) circuit design using NAND gates; (b) function table; (c) logic symbol.

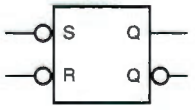
(a)

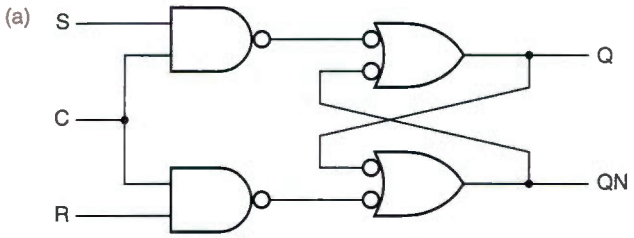


(b)

S_L	R_L	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	last Q	last QN

(c)





(b)

S	R	C	Q	QN
0	0	1	last Q	last QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	last Q	last QN

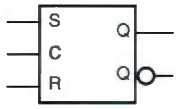


Figure 7-10 S-R latch with enable: (a) circuit using NAND gates; (b) function table; (c) logic symbol.

7.2.3 S-R Latch with Enable

S-R latch with enable

An S-R or $\overline{S}$-$\overline{R}$ latch is sensitive to its S and R inputs at all times. However, it may easily be modified to create a device that is sensitive to these inputs only when an enabling input C is asserted. Such an *S-R latch with enable* is shown in Figure 7-10. As shown by the function table, the circuit behaves like an S-R latch when C is 1, and retains its previous state when C is 0. The latch's behavior for a typical set of inputs is shown in Figure 7-11. If both S and R are 1 when C changes from 1 to 0, the circuit behaves like an S-R latch in which S and R are negated simultaneously—the next state is unpredictable and the output may become metastable.

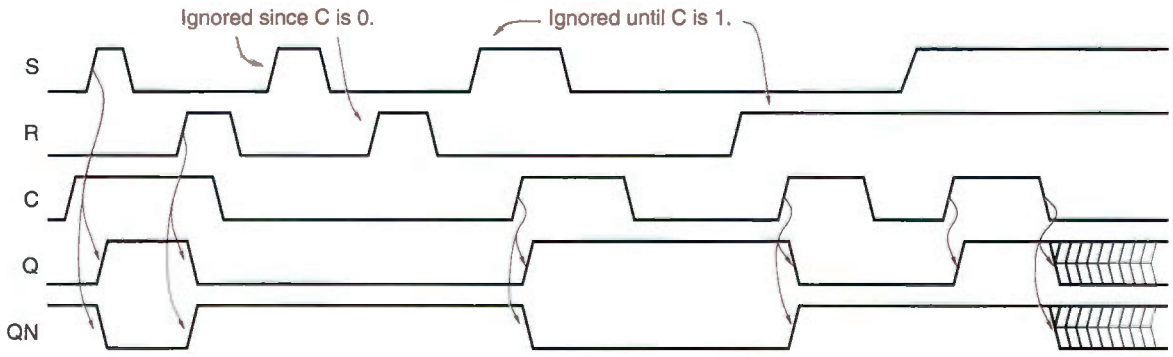
7.2.4 D Latch

D latch

S-R latches are useful in control applications, where we often think in terms of setting a flag in response to some condition and resetting it when conditions change. So, we control the set and reset inputs somewhat independently. However, we often need latches simply to store bits of information—each bit is presented on a signal line, and we'd like to store it somewhere. A *D latch* may be used in such an application.

Figure 7-12 shows a D latch. Its logic diagram is recognizable as that of an S-R latch with enable, with an inverter added to generate S and R inputs from the

Figure 7-11 Typical operation of an S-R latch with enable.



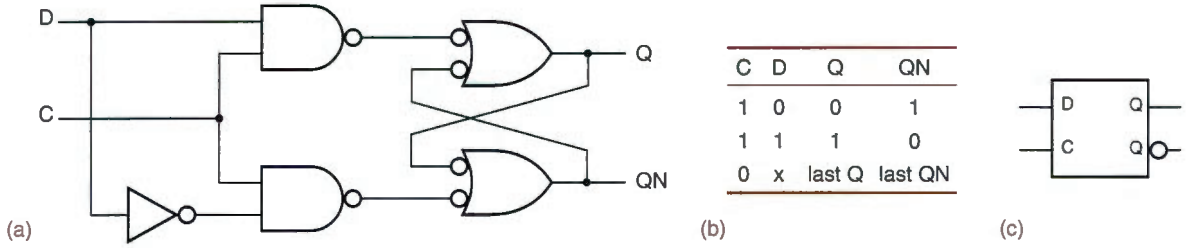


Figure 7-12 D latch: (a) circuit design using NAND gates; (b) function table; (c) logic symbol.

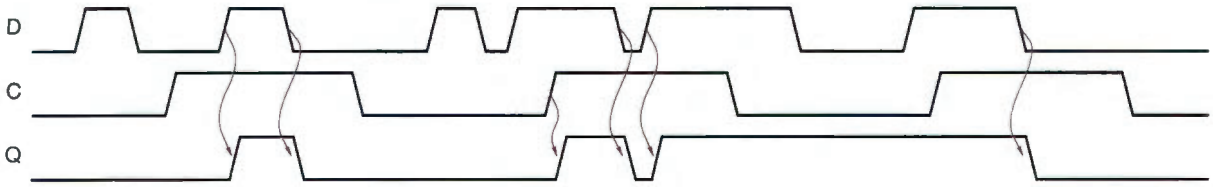


Figure 7-13 Functional behavior of a D latch for various inputs.

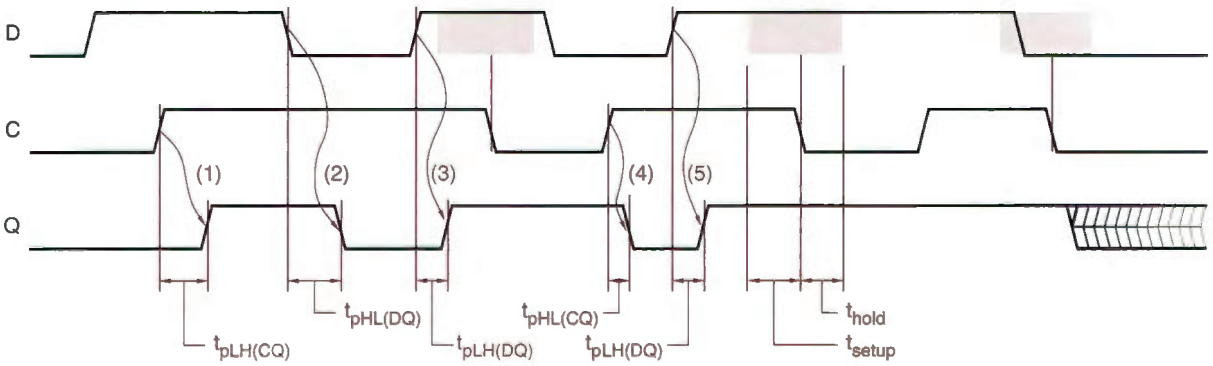
single D (data) input. This eliminates the troublesome situation in S-R latches, where S and R may be asserted simultaneously. The control input of a D latch, labeled C in (c), is sometimes named ENABLE, CLK, or G. It is active low in some D-latch designs, and always has a minimum pulse-width requirement.

An example of a D latch's functional behavior is given in Figure 7-13. When the C input is asserted, the Q output follows the D input. In this situation, the latch is said to be "open" and the path from D input to Q output is "transparent"; the circuit is often called a *transparent latch* for this reason. When the C input is negated, the latch "closes"; the Q output retains its last value and no longer changes in response to D, as long as C remains negated.

transparent latch

More detailed timing behavior of the D latch is shown in Figure 7-14. Four different delay parameters are shown for signals that propagate from the C or D input to the Q output. For example, at transitions 1 and 4 the latch is initially

Figure 7-14 Timing parameters for a D latch.



"closed" and the D input is the opposite of Q output, so that when C goes to 1 the latch "opens up" and the Q output changes after delay $t_{pLH(CQ)}$ or $t_{pHL(CQ)}$. At transitions 2 and 3 the C input is already 1 and the latch is already open, so that Q transparently follows the transitions on D with delay $t_{pHL(DQ)}$ and $t_{pLH(DQ)}$. Four more parameters specify the delay to the QN output, not shown.

Although the D latch eliminates the S = R = 1 problem of the S-R latch, it does not eliminate the metastability problem. As shown in Figure 7-14, there is a (shaded) window of time around the falling edge of C when the D input must not change. This window begins at time t_{setup} before the falling (latching) edge of C; t_{setup} is called the *setup time*. The window ends at time t_{hold} afterward; t_{hold} is called the *hold time*. If D changes at any time during the setup- and hold-time window, the output of the latch is unpredictable and may become metastable, as shown for the last latching edge in the figure.

setup time

hold time

7.2.5 Edge-Triggered D Flip-Flop

positive-edge-triggered D flip-flop

A *positive-edge-triggered D flip-flop* combines a pair of D latches, as shown in Figure 7-15, to create a circuit that samples its D input and changes its Q and QN outputs only at the rising edge of a controlling CLK signal. The first latch is called the *master*; it is open and follows the input when CLK is 0. When CLK goes to 1, the master latch is closed and its output is transferred to the second latch, called the *slave*. The slave latch is open all the while that CLK is 1, but changes only at the beginning of this interval, because the master is closed and unchanging during the rest of the interval.

master

slave

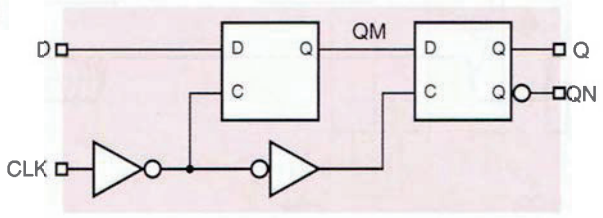
The triangle on the D flip-flop's CLK input indicates edge-triggered behavior and is called a *dynamic-input indicator*. Examples of the flip-flop's functional behavior for several input transitions are presented in Figure 7-16. The QM signal shown is the output of the master latch. Notice that QM changes only when CLK is 0. When CLK goes to 1, the current value of QM is transferred to Q, and QM is prevented from changing until CLK goes to 0 again.

dynamic-input indicator

Figure 7-17 shows more detailed timing behavior for the D flip-flop. All propagation delays are measured from the rising edge of CLK, since that's the only event that causes an output change. Different delays may be specified for LOW-to-HIGH and HIGH-to-LOW output changes.

Figure 7-15 Positive-edge-triggered D flip-flop: (a) circuit design using D latches; (b) function table; (c) logic symbol.

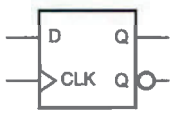
(a)



(b)

D	CLK	Q	QN
0	↑	0	1
1	↑	1	0
x	0	last Q	last QN
x	1	last Q	last QN

(c)



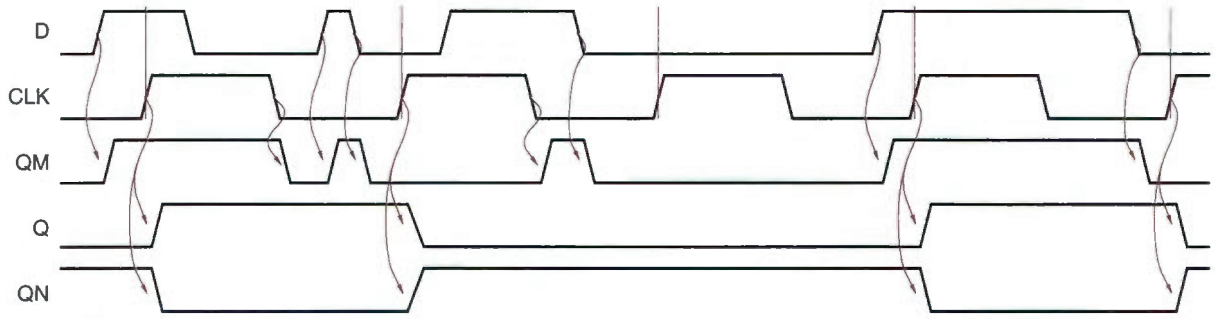


Figure 7-16 Functional behavior of a positive-edge-triggered D flip-flop.

Like a D latch, the edge-triggered D flip-flop has a setup- and hold-time window during which the D inputs must not change. This window occurs around the triggering edge of CLK, and is indicated by shaded color in Figure 7-17. If the setup and hold times are not met, the flip-flop output will usually go to a stable, though unpredictable, 0 or 1 state. In some cases, however, the output will oscillate or go to a metastable state halfway between 0 and 1, as shown at the second-to-last clock tick in the figure. If the flip-flop goes into the metastable state, it will return to a stable state on its own only after a probabilistic delay, as explained in Section 8.9. It can also be forced into a stable state by applying another triggering clock edge with a D input that meets the setup- and hold-time requirements, as shown at the last clock tick in the figure.

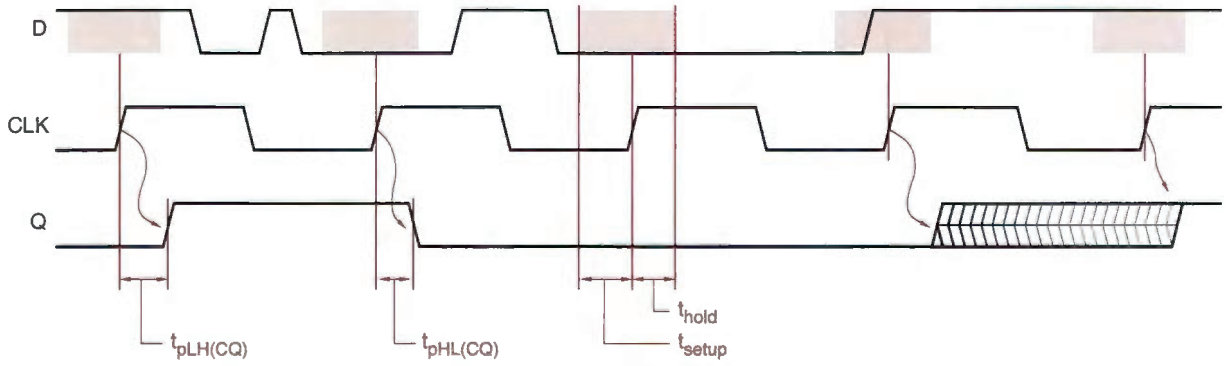
A *negative-edge-triggered D flip-flop* simply inverts the clock input, so that all the action takes place on the falling edge of CLK_L; by convention, a falling-edge trigger is considered to be active low. The flip-flop's function table and logic symbol are shown in Figure 7-18.

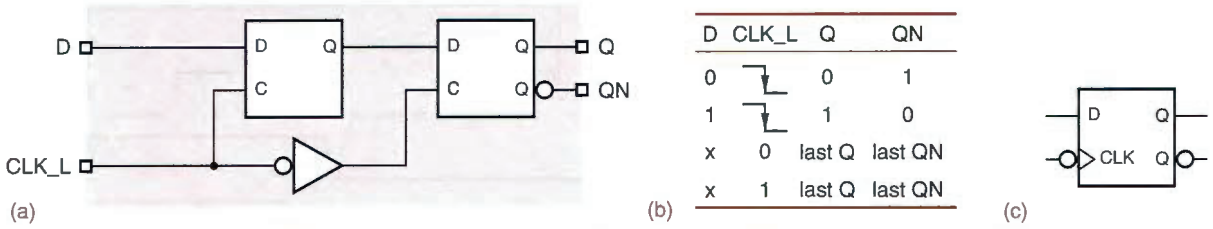
negative-edge-triggered D flip-flop

Some D flip-flops have *asynchronous inputs* that may be used to force the flip-flop to a particular state independent of the CLK and D inputs. These inputs, typically labeled PR (*preset*) and CLR (*clear*), behave like the set and reset

asynchronous inputs
preset
clear

Figure 7-17 Timing behavior of a positive-edge-triggered D flip-flop.





D	CLK_L	Q	QN
0	↓	0	1
1	↓	1	0
x	0	last Q	last QN
x	1	last Q	last QN

Figure 7-18 Negative-edge triggered D flip-flop: (a) circuit design using D latches; (b) function table; (c) logic symbol.

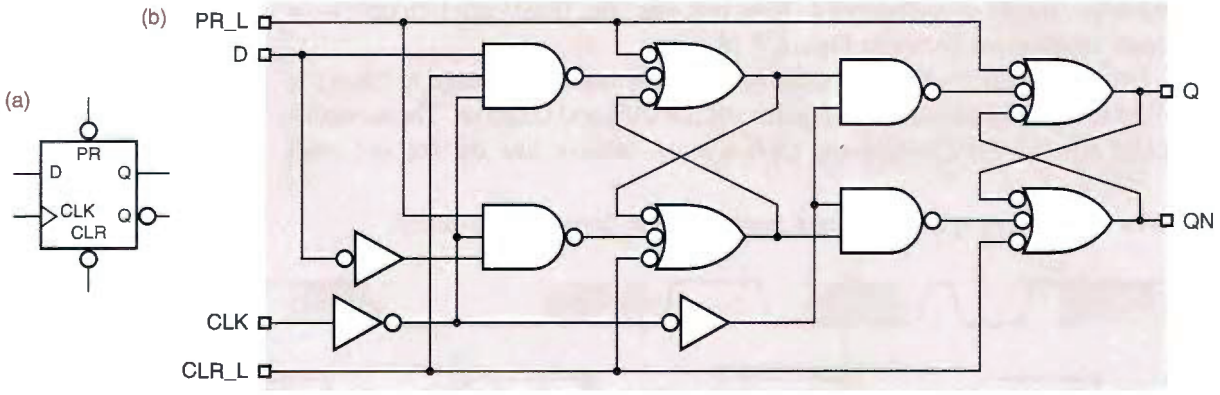
inputs on an S-R latch. The logic symbol and NAND circuit for an edge-triggered D flip-flop with these inputs is shown in Figure 7-19. Although asynchronous inputs are used by some logic designers to perform tricky sequential functions, they are best reserved for initialization and testing purposes, to force a sequential circuit into a known starting state; more on this when we discuss synchronous design methodology in Section 8.7.

7.2.6 Edge-Triggered D Flip-Flop with Enable

enable input
clock-enable input

A commonly desired function in D flip-flops is the ability to hold the last value stored, rather than load a new value, at the clock edge. This is accomplished by adding an *enable input*, called EN or CE (*clock enable*). While the name "clock enable" is descriptive, the extra input's function is not obtained by controlling

Figure 7-19 Positive-edge-triggered D flip-flop with preset and clear: (a) logic symbol; (b) circuit design using NAND gates.



TIME FOR A COMMERCIAL

Commercial TTL positive-edge-triggered D flip-flops do not use the master-slave latch design of Figure 7-15 or Figure 7-19. Instead, flip-flops like the 74LS74 use the six-gate design of Figure 7-20, which is smaller and faster. We'll show how to formally analyze the next-state behavior of both designs in Section 7.9.

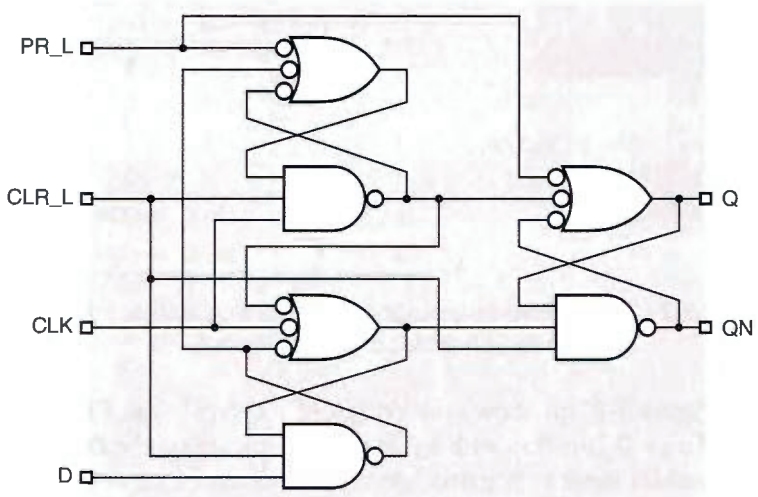


Figure 7-20 Commercial circuit for a positive-edge-triggered D flip-flop such as 74LS74.

the clock in any way whatsoever. Rather, as shown in Figure 7-21(a), a 2-input multiplexer controls the value applied to the internal flip-flop's D input. If EN is asserted, the external D input is selected; if EN is negated, the flip-flop's current output is used. The resulting function table is shown in (b). The flip-flop symbol is shown in (c); in some flip-flops, the enable input is active low, denoted by an inversion bubble on this input.

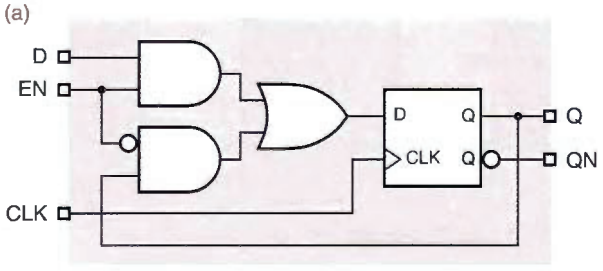
7.2.7 Scan Flip-Flop

scan capability

An important flip-flop function for ASIC testing is so-called *scan capability*. The idea is to be able to drive the flip-flop's D input with an alternate source of data during device testing. When all of the flip-flops are put into testing mode, a test pattern can be "scanned in" to the ASIC using the flip-flops' alternate data inputs. After the test pattern is loaded, the flip-flops are put back into "normal" mode, and all of the flip-flops are clocked normally. After one or more clock ticks, the flip-flops are put back into test mode, and the test results are "scanned out."

Figure 7-21 Positive-edge-triggered D flip-flop with enable: (a) circuit design; (b) function table; (c) logic symbol.

(a)



(b)

D	EN	CLK	Q	QN
0	1	↑	0	1
1	1	↑	1	0
x	0	↑	last Q	last QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN

(c)