ECE 2372 Modern Digital Design July 17

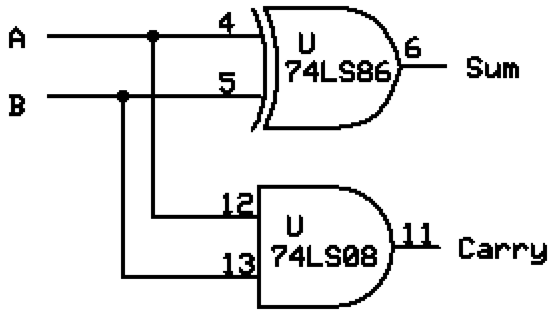
Reading assignment for today: Ch 3: section 3-9, 3-10, 3-11 Binary Adders, Ch 4: 4-1 thru 4-3

Last time we saw how to make a simple half adder circuit using ordinary logic gates which will provide a sum and a carry output given two binary operands A and B to be added together. From analysis of a truth table for Sum and Carry, we determined that we needed an XOR (exclusive OR) gate for the Sum and an AND gate for the Carry. However, we also realized that the half adder was not sufficient, even though it was a good start toward what we needed. The half adder did not provide a mechanism for a carry-in from a prior stage. Although that would generally be OK for the first state in an adder, it would not be adequate for subsequent stages of an adder. So, we derived a full adder by combining two half adders and an OR gate.

Quickly, let's review the half adder truth table and the half adder given below:

| operands | | AND | XOR |
|---|---|---|---|
| A | B | Carry | Sum |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

We can form what is referred to as a half-adder with simply an XOR gate and an AND gate as shown below:

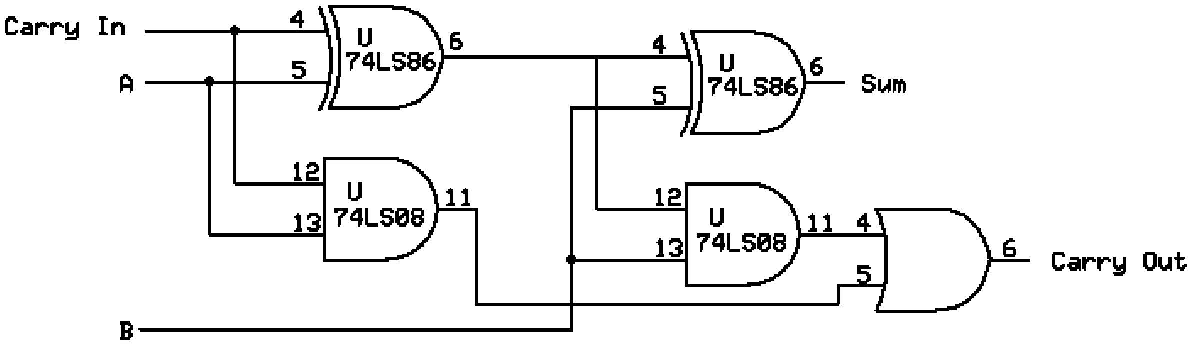


Now let's review the truth table for a full adder

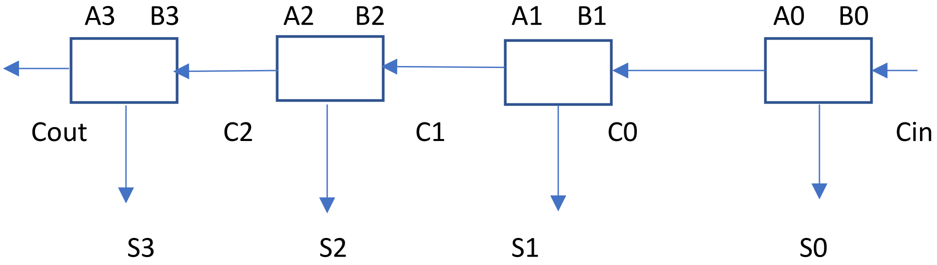
| Carry In | operands | | AND | XOR |
|---|---|---|---|---|
| | A | B | Carry out | Sum |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Referring to your textbook on page 158, this truth table can be reduced to:
Sum = ~C~AB + ~CA~B + C~A~B + CAB
Carry = CA +CB + AB

The textbook shows that combining two half adder circuits in the correct way forms a full adder that has all the capability we need. The full adder, formed from two half adders and an extra OR gate, is shown below:



We can connect several of these units (full binary adders) together to form a multi-bit binary adder. For example, we could connect 4 of these together to for a 4-bit ripple carry adder.



This is called a ripple carry adder because the value of the Carry bit at any stage has to be determined over a small period of time necessary for the signals (A, B operands, and Carry bits) to propagate thru the logic gates, and then ripple thru to the next stage. This type of adder can be rather slow, but it is easy to understand and easy to implement. As mentioned last time, a setup like this extended to 32 bits, with 10 nS propagation time of the signals inside the logic gates, it would take almost a full 1 microSecond to add two 32 bit binary numbers. That is exceedingly slow in the world of computers. That problem can be solved by faster gates and more sophisticated approaches to solving the problem – methods that are beyond the scope of this introductory course.

I mentioned last time that we could apply some concepts from 2's complement signed binary numbers to form a subtractor from an adder. We shall see how to do this in a moment. Remember that to convert a binary number into its negative, we invert all the bits and add 1. Inverting all the bits of a particular operand is easy, just feed all those signals thru an invertor or NOT gate and the signals are all inverted. To add 1, simply apply 1 to the Carry-in signal of the first stage. So, if we have two operands, A and B, and we want to subtract B from A, we can achieve that by converting B to its negative, and then adding B to A. The result is A – B.

A – B = A + (-B)

B, a binary number is converted to -B by inverting all the bits and adding 1 (2’s complement conversion).

On page 166 of your textbook is figure 3-45 which shows a 4-bit ripple adder converted into a combination adder/subtractor. Notice that XOR gates have been placed in the path of the B operand. Recall the operation of an XOR gate:

| operands | | XOR |
|---|---|---|
| A | B | |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

If we relabel the entries in this table, so that A is labeled “passthru”, we can get this:

| passthru | | XOR | |
|---|---|---|---|
| signal | B | | |
| 0 | 0 | 0 | B is “passed thru intact” |
| 0 | 1 | 1 | B is “passed thru intact” |
| 1 | 0 | 1 | B is “inverted” |
| 1 | 1 | 0 | B is “inverted” |

So, we see from this truth table that an XOR gate can be made into a selective “passthru vs invert function. This is extremely convenient and allows us to selectively “invert” the B operand. Notice in the figure below from your textbook that the “passthru/invert” control signal is routed to a signal labeled ‘S’ for subtract. Notice the ‘S’ signal also is sent to the Carry-in of the first stage. Thus, by placing a logic 1 on the ‘S’ signal, we invert all the bits on the B operand, and we also add 1 via the Carry-in bit. Thus, the B operand is converted into its own negative, and when it is added to the A operand, the end result is B is subtracted from A, yielding A-B by way of doing it as A + (-B). Designing and building a subtractor could be done by another approach, but the approach shown here is relatively simple and easy to understand.

From your book, the ripple adder/subtractor circuit:

## Sequential Circuits:

Before we learn about sequential circuits, we need to learn about a simple method of storing information in a logic circuit. This type of binary storage element is called a "flip-flop" and can be in one of two states, either storing a 1 or storing a 0.

On page 201 of your textbook, in figure 4-4 we are shown an S-R latch made from two cross coupled NOR gates. The name S-R latch comes from the terms "Set" and "Reset". The latch can be set into the true state (with Q output = 1, and ~Q output = 0), or it can be reset into the false state with Q output = 0 and ~Q output = 1). I am not sure where the term Q and ~Q originated from, but that terminology has been associated with flip-flops and latches for as long as I have known about them.

Notice in the table, the first two rows are the Set state. Notice Q = 1 and ~Q = 0 for that state. We can get to this state by applying a 1 to the S (Set) input and a 0 at the R (Reset) input. Once the latch has been "set", we can now bring the S input back to a 0 state and the latch will remain "set" with Q = 1 and ~Q = 0.

We can now "reset" the latch, by applying a 1 to the R (Reset) input and a 0 at the S (Set) input. Once the latch has been reset, we can now take the R (Reset) input back to the 0 state and the latch will remain in the "reset" state.

This is a very simple latch, appropriately called an S-R latch. Notice that to "Set" the latch, the S (Set) input is brought to 1 state (True) and the R (Reset) input is kept in the 0 (False) state. Once this has been done, the S (Set) input can be brought back to the 0 state and the latch will stay latched in the Set state.

To reset the latch, we do the opposite. We bring the R (Reset) input to the 1 state and the S (Set) input to the 0 state, and the latch will be reset. We can now bring the R (Reset) input back to the 0 state and the latch will remain in the reset state.

Notice that the truth table shows the state of S and R both being in the 1 state at the same time as an invalid condition.

So, how does this clever little circuit work? By feedback. Feedback is an awesome concept in electronics and you will see it many places. Let us suppose initially that the latch is in the Reset state, i.e. Q = 0 and ~Q = 1. Notice that the ~Q signal is fed back from ~Q to one of the inputs on the top NOR gate. That ~Q state (1) is fed back as an input (1) to one input of the top NOR gate. Reviewing the truth table for a NOR gate:

| operands | | NOR |
|---|---|---|
| A | B | |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Notice that if you have a 1 on either input of a NOR gate, the output is 0. Thus, with ~Q = 1 fed back to one input of the top NOR gate, the output of that gate (Q) is forced to = 0. That 0, on Q output, is fed back to one input of the bottom NOR gate. The S (Set) input has already been defined to be at 0 at this point, so with two 0 inputs on the bottom NOR gate, its output must 1 at the ~Q output. But, this was already established earlier.

Now, how do we set the latch? Suppose we are in the reset state, S = 0 and R = 0 and the latch is latched with Q = 0 and ~Q = 1. Now, bring the S (Set) input to the high state (1). Having a 1 on either input of the bottom NOR gate causes its output (~Q) to go to the 0 state. That 0 state is fed back to one input of the top NOR gate. It already has a 0 on the R input based on earlier statements. With two 0 inputs, the top NOR gate now has an output state of 1 (Q), and this is fed back now to the bottom NOR gate, latching the bottom NOR gate into the state with ~Q = 0. This is the "set" state of the latch.

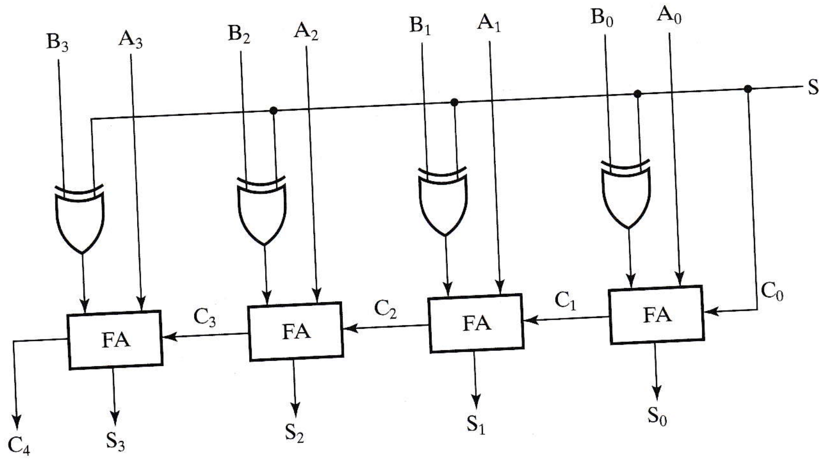
On page 203 of your textbook, in figure 4-6 we see an ~S-~R latch made with NAND gates. Notice this has active LOW input signals. The S-R latch made with NOR gates had active HIGH signals for Set and Reset, but the ~S-~R latch made with NAND gates has active LOW input signals for S and R.

Notice there will be some time delay from the time an input signal changes until the outputs of Q and ~Q are "settled". If the gates have 10 nS delay, then time delays of that order are to be expected.

A somewhat improved D latch is shown on page 204 in figure 4-8. This improved topology gets around the undefined condition of S and R both being 1, and it introduces a "clocking" mechanism to activate the latch. A further improvement is shown on page 205 in figure 4-9, forming an edge triggered D flip-flop. The one shown is a negative edge triggered flip-flop. On the next page is shown a positive edge triggered D flip-flop. These are used quite often. In fact, there are commercial components available containing these and more advanced flip-flops.

We will stop here for today. Next time we will look more closely at a particular commercial edge triggered D flip-flop. If you would like to get a look ahead, go to Mouser.com and look up the component 74LS74, which we will look at next time.

Some material from your textbook is included below. Still a little behind on grading, should get caught up in the next couple of days.



□ **FIGURE 3-45**
Adder-Subtractor Circuit

full adder. A 4-bit adder–subtractor circuit is shown in Figure 3-45. Input $S$ controls the operation. When $S = 0$ the circuit is an adder, and when $S = 1$ the circuit becomes a subtractor. Each exclusive-OR gate receives input $S$ and one of the inputs of $B$, $B_i$. When $S = 0$, we have $B_i \oplus 0$. If the full adders receive the value of $B$, and the input carry is 0, the circuit performs $A$ plus $B$. When $S = 1$, we have $B_i \oplus 1 = \overline{B}_i$ and $C_0 = 1$. In this case, the circuit performs the operation $A$ plus the 2s complement of $B$.

## Signed Binary Numbers

In the previous section, we dealt with the addition and subtraction of unsigned numbers. We will now extend this approach to signed numbers, including a further use of complements that eliminates the correction step.

Positive integers and the number zero can be represented as unsigned numbers. To represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with 1s and 0s, including the sign of a number. As a consequence, it is customary to represent the sign with a bit placed in the most significant position of an $n$-bit number. The convention is to make the sign bit 0 for positive numbers and 1 for negative numbers.

It is important to realize that both signed and unsigned binary numbers consist of a string of bits when represented in a computer. The user determines whether the number is signed or unsigned. If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number. If the binary number is assumed to be unsigned, then the leftmost bit is the most significant bit of the number. For example, the string of bits 01001 can be considered as 9 (unsigned binary) or +9 (signed binary), because the leftmost bit is 0. Similarly, the string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned

## 4-2 Latches

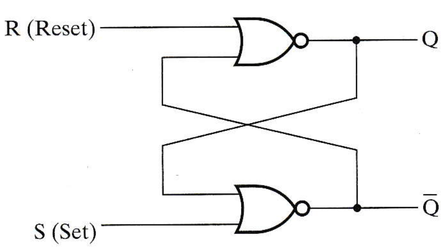
A storage element can maintain a binary state indefinitely (as long as power is delivered to the circuit), until directed by an input signal to switch states. The major differences among the various types of latches and flip-flops are the number of inputs they possess and the manner in which the inputs affect the binary state. The most basic storage elements are latches, from which flip-flops are usually constructed. Although latches are most often used within flip-flops, they can also be used with more complex clocking methods to implement sequential circuits directly. The design of such circuits is, however, beyond the scope of the basic treatment given here. In this section, the focus is on latches as basic primitives for constructing storage elements.

### *SR* and $\overline{S}\,\overline{R}$ Latches

The *SR* latch is a circuit constructed from two cross-coupled NOR gates. It is derived from the single-loop storage element in Figure 4-2(d) by simply replacing the inverters with NOR gates, as shown in Figure 4-4(a). This replacement allows the stored value in the latch to be changed. The latch has two inputs, labeled *S* for set and *R* for reset, and two useful states. When output $Q = 1$ and $\overline{Q} = 0$, the latch is said to be in the *set state*. When $Q = 0$ and $\overline{Q} = 1$, it is in the *reset state*. Outputs $Q$ and $\overline{Q}$ are normally the complements of each other. When both inputs are equal to 1 at the same time, an undefined state with both outputs equal to 0 occurs.

Under normal conditions, both inputs of the latch remain at 0 unless the state is to be changed. The application of a 1 to the *S* input causes the latch to go to the set (1) state. The *S* input must go back to 0 before *R* is changed to 1 to avoid occurrence of the undefined state. As shown in the function table in Figure 4-4(b), two input conditions cause the circuit to be in the set state. The initial condition is $S = 1$, $R = 0$, to bring the circuit to the set state. Applying a 0 to *S* with $R = 0$ leaves the circuit in the same state. After both inputs return to 0, it is possible to enter the reset state by applying a 1 to the *R* input. The 1 can then be removed from *R*, and the circuit remains in the reset state. Thus, when both inputs are equal to 0, the latch can be in either the set or the reset state, depending on which input was most recently a 1.

If a 1 is applied to both the inputs of the latch, both outputs go to 0. This produces an undefined state, because it violates the requirement that the outputs be the



(a) Logic diagram

| S | R | Q | $\overline{Q}$ | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | Set state |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 1 | Reset state |
| 0 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 0 | Undefined |

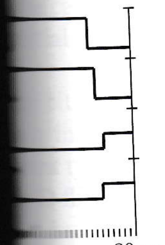
(b) Function table

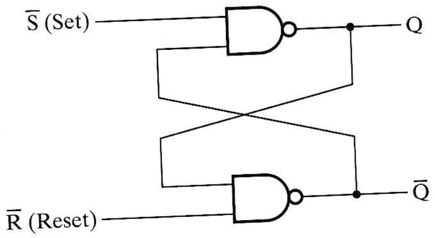
□ **FIGURE 4-4**
*SR* Latch with NOR Gates

…ctable next
…se problems
…usly.
…s illustrated
…ly the inputs
…y between 0
…ng 0 and, in
…, the latch
…R at 0, the
The delays in
…e delays of
…0, the latch
…S equal to 0,
…1. The latch
…Q and Q_b
…Q_b take on
…sequence of
…equal gate
…delays and
…Regardless
…undesirable,
…nges only in

…Figure 4-6. It
…re changed.
…e latch in the
…state. With
…R input. This
…puts return
…inputs are
…d.

…put signals
…R. Because





(a) Logic diagram

| $\overline{S}$ | $\overline{R}$ | Q | $\overline{Q}$ | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Set state |
| 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Reset state |
| 1 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | Undefined |

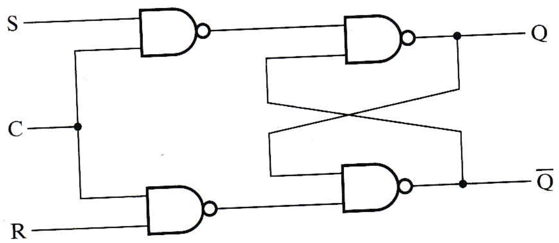
(b) Function table

□ **FIGURE 4-6**
$\overline{S}\,\overline{R}$ Latch with NAND Gates

the NAND latch requires a 0 signal to change its state, it is referred to as an $\overline{S}\,\overline{R}$ latch. The bar above the letters designates the fact that the inputs must be in their complement form in order to act upon the circuit state.

The operation of the basic NOR and NAND latches can be modified by providing an additional control input that determines when the state of the latch can be changed. An *SR* latch with a control input is shown in Figure 4-7. It consists of the basic NAND latch and two additional NAND gates. The control input *C* acts as an enable signal for the other two inputs. The output of the NAND gates stays at the logic-1 level as long as the control input remains at 0. This is the quiescent condition for the $\overline{S}\,\overline{R}$ latch composed of two NAND gates. When the control input goes to 1, information from the *S* and *R* inputs is allowed to affect the $\overline{S}\,\overline{R}$ latch. The set state is reached with $S = 1$, $R = 0$, and $C = 1$. To change to the reset state, the inputs must be $S = 0$, $R = 1$, and $C = 1$. In either case, when *C* returns to 0, the circuit remains in its current state. Control input $C = 0$ disables the circuit so that the state of the output does not change, regardless of the values of *S* and *R*. Moreover, when $C = 1$ and both the *S* and *R* inputs are equal to 0, the state of the circuit does not change. These conditions are listed in the function table accompanying the diagram.

An undefined state occurs when all three inputs are equal to 1. This condition places 0s on both inputs of the basic $\overline{S}\,\overline{R}$ latch, giving an undefined state. When the

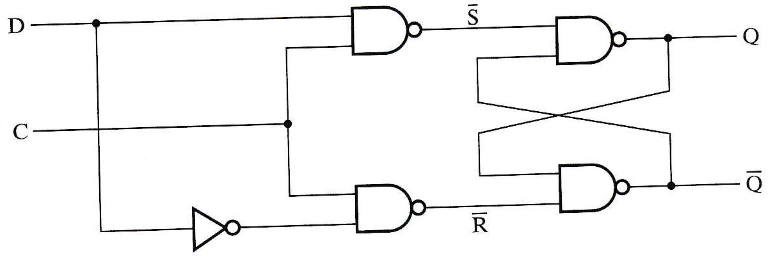


(a) Logic diagram

| C | S | R | Next state of Q |
|---|---|---|---|
| 0 | X | X | No change |
| 1 | 0 | 0 | No change |
| 1 | 0 | 1 | Q = 0; Reset state |
| 1 | 1 | 0 | Q = 1; Set state |
| 1 | 1 | 1 | Undefined |

(b) Function table

□ **FIGURE 4-7**
*SR* Latch with Control Input



(a) Logic diagram

| C | D | Next state of Q |
|---|---|---|
| 0 | X | No change |
| 1 | 0 | Q = 0; Reset state |
| 1 | 1 | Q = 1; Set state |

(b) Function table

□ **FIGURE 4-8**
*D* Latch

control input goes back to 0, one cannot conclusively determine the next state, since the $\overline{S}\,\overline{R}$ latch sees inputs (0, 0) followed by (1, 1). The *SR* latch with control input is an important circuit, because other latches and flip-flops are constructed from it. Sometimes the *SR* latch with control input is referred to as an *SR* (or *RS*) flip-flop—however, according to our terminology, it does not qualify as a flip-flop, since the circuit does not fulfill the flip-flop requirements presented in the next section.

### *D* Latch

One way to eliminate the undesirable undefined state in the *SR* latch is to ensure that inputs *S* and *R* are never equal to 1 at the same time. This is done in the *D* latch, shown in Figure 4-8. This latch has only two inputs: *D* (data) and *C* (control). The complement of the *D* input goes directly to the $\overline{S}$ input, and *D* is applied to the $\overline{R}$ input. As long as the control input is 0, the $\overline{S}\,\overline{R}$ latch has both inputs at the 1 level, and the circuit cannot change state regardless of the value of *D*. The *D* input is sampled when $C = 1$. If *D* is 1, the *Q* output goes to 1, placing the circuit in the set state. If *D* is 0, output *Q* goes to 0, placing the circuit in the reset state.

The *D* latch receives its designation from its ability to hold *data* in its internal storage. The binary information present at the data input of the *D* latch is transferred to the *Q* output when the control input is enabled (1). The output follows changes in the data input, as long as the control input is enabled. When the control input is disabled (0), the binary information that was present at the data input at the time the transition in *C* occurred is retained at the *Q* output until the control input *C* is enabled again.

## 4-3 Flip-Flops

A change in value on the control input allows the state of a latch in a flip-flop to switch. This change is called a *trigger*, and it enables, or triggers, the flip-flop. The *D*

state, since
input is
from it.
flip-flop—
since the
tion.

to ensure
e D latch,
ntrol). The
the $\overline{R}$ in-
level, and
is sampled
state. If $D$

ernal stor-
rred to the
in the data
led (0), the
sition in $C$
ain.

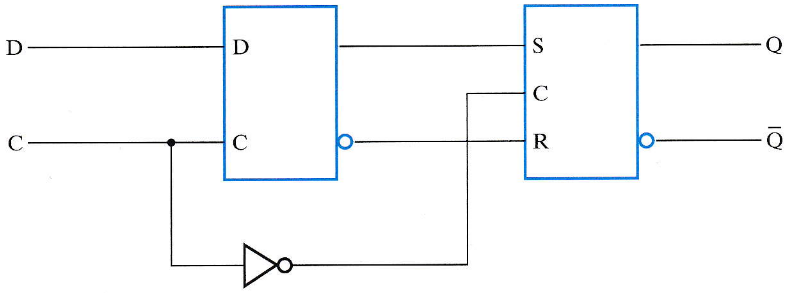
flip-flop to
op. The $D$

latch with clock pulses on its control input is triggered every time a pulse to the logic-1 level occurs. As long as the pulse remains at the active (1) level, any changes in the data input will change the state of the latch. In this sense, the latch is *transparent*, since its input value can be seen from the outputs while the control input is 1.

As the block diagram of Figure 4-3 shows, a sequential circuit has a feedback path from the outputs of the flip-flops to the combination circuit. As a consequence, the data inputs of the flip-flops are derived in part from the outputs of the same and other flip-flops. When latches are used for the storage elements, a serious difficulty arises. The state transitions of the latches start as soon as the clock pulse changes to the logic-1 level. The new state of a latch may appear at its output while the pulse is still active. This output is connected to the inputs of some of the latches through a combinational circuit. If the inputs applied to the latches change while the clock pulse is still in the logic-1 level, the latches will respond to *new state values* of other latches instead of the *original state values*, and a succession of changes of state instead of a single one may occur. The result is an unpredictable situation, since the state may keep changing and continue to change until the clock returns to 0. The final state depends on how long the clock pulse stays at the logic-1 level. Because of this unreliable operation, the output of a latch cannot be applied directly or through combinational logic to the input of the same or another latch when all the latches are triggered by a single clock signal.

Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential circuit that employs a single clock. Note that the problem with the latch is that it is transparent: As soon as an input changes, shortly thereafter the corresponding output changes to match it. This transparency is what allows a change on a latch output to produce additional changes at other latch outputs while the clock pulse is at logic 1. The key to the proper operation of flip-flops is to prevent them from being transparent. In a flip-flop, before an output can change, the path from its inputs to its outputs is broken. So a flip-flop cannot "see" the change of its output or of the outputs of other, similar flip-flops at its input during the same clock pulse. Thus, the new state of a flip-flop depends only on the immediately preceding state, and the flip-flops do not go through multiple changes of state.

A common way to create a flip-flop is to connect two latches as shown in Figure 4-9, which is often referred to as a *master–slave* flip-flop. The left latch, the master, changes its value based upon the input while the clock is high. That value is



□ **FIGURE 4-9**
Negative-Edge-Triggered D Flip-Flop