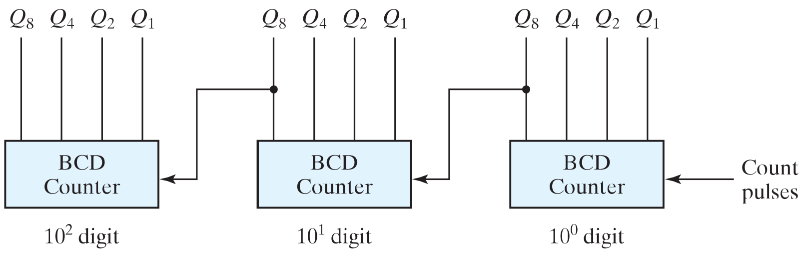


**FIGURE 6.11**
**Block diagram of a three-decade decimal BCD counter**

To verify that these conditions result in the sequence required by a BCD ripple counter, it is necessary to verify that the flip-flop transitions indeed follow a sequence of states as specified by the state diagram of Fig. 6.9. $Q_1$ changes state after each clock pulse. $Q_2$ complements every time $Q_1$ goes from 1 to 0, as long as $Q_8 = 0$. When $Q_8$ becomes 1, $Q_2$ remains at 0. $Q_4$ complements every time $Q_2$ goes from 1 to 0. $Q_8$ remains at 0 as long as $Q_2$ or $Q_4$ is 0. When both $Q_2$ and $Q_4$ become 1, $Q_8$ complements when $Q_1$ goes from 1 to 0. $Q_8$ is cleared on the next transition of $Q_1$.

The BCD counter of Fig. 6.10 is a *decade* counter, since it counts from 0 to 9. To count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter. Multiple decade counters can be constructed by connecting BCD counters in cascade, one for each decade. A three-decade counter is shown in Fig. 6.11. The inputs to the second and third decades come from $Q_8$ of the previous decade. When $Q_8$ in one decade goes from 1 to 0, it triggers the count for the next higher order decade while its own decade goes from 9 to 0.

## 6.4 SYNCHRONOUS COUNTERS

Synchronous counters are different from ripple counters in that clock pulses are applied to the inputs of all flip-flops. A common clock triggers all flip-flops simultaneously, rather than one at a time in succession as in a ripple counter. The decision whether a flip-flop is to be complemented is determined from the values of the data inputs, such as $T$ or $J$ and $K$ at the time of the clock edge. If $T = 0$ or $J = K = 0$, the flip-flop does not change state. If $T = 1$ or $J = K = 1$, the flip-flop complements.

The design procedure for synchronous counters was presented in Section 5.8, and the design of a three-bit binary counter was carried out in conjunction with Fig. 5.31. In this section, we present some typical synchronous counters and explain their operation.

### Binary Counter

The design of a synchronous binary counter is so simple that there is no need to go through a sequential logic design process. In a synchronous binary counter, the flip-flop in the least significant position is complemented with every pulse. *A flip-flop in any other* *position is complemented when all the bits in the lower significant positions are equal to 1*. For example, if the present state of a four-bit counter is $A_3A_2A_1A_0 = 0011$, the next count is 0100. $A_0$ is always complemented. $A_1$ is complemented because the present state of $A_0 = 1$. $A_2$ is complemented because the present state of $A_1A_0 = 11$. However, $A_3$ is not complemented, because the present state of $A_2A_1A_0 = 011$, which does not give an all-1's condition.

Synchronous binary counters have a regular pattern and can be constructed with complementing flip-flops and gates. The regular pattern can be seen from the four-bit counter depicted in Fig. 6.12. The $C$ inputs of all flip-flops are connected to a common clock. The counter is enabled by *Count_enable*. If the enable input is 0, all $J$ and $K$ inputs are equal to 0 and the clock does not change the state of the counter. The first stage, $A_0$, has its $J$ and $K$ equal to 1 if the counter is enabled. The other $J$ and $K$ inputs are equal to 1 if all previous least significant stages are equal to 1 and the count is enabled. The chain of AND gates generates the required logic for the $J$ and $K$ inputs in each stage. The counter can be extended to any number of stages, with each stage having an additional flip-flop and an AND gate that gives an output of 1 if all previous flip-flop outputs are 1.

Note that the flip-flops trigger on the positive edge of the clock. The polarity of the clock is not essential here, but it is with the ripple counter. The synchronous counter can be triggered with either the positive or the negative clock edge. The complementing flip-flops in a binary counter can be of either the $JK$ type, the $T$ type, or the $D$ type with XOR gates. The equivalency of the three types is indicated in Fig. 5.13.

## Up–Down Binary Counter

A synchronous countdown binary counter goes through the binary states in reverse order, from 1111 down to 0000 and back to 1111 to repeat the count. It is possible to design a countdown counter in the usual manner, but the result is predictable by inspection of the downward binary count. The bit in the least significant position is complemented with each pulse. *A bit in any other position is complemented if all lower significant bits are equal to 0.* For example, the next state after the present state of 0100 is 0011. The least significant bit is always complemented. The second significant bit is complemented because the first bit is 0. The third significant bit is complemented because the first two bits are equal to 0. But the fourth bit does not change, because not all lower significant bits are equal to 0.

A countdown binary counter can be constructed as shown in Fig. 6.12, except that the inputs to the AND gates must come from the complemented outputs, instead of the normal outputs, of the previous flip-flops. The two operations can be combined in one circuit to form a counter capable of counting either up or down. The circuit of an up–down binary counter using $T$ flip-flops is shown in Fig. 6.13. It has an up control input and a down control input. When the up input is 1, the circuit counts up, since the $T$ inputs receive their signals from the values of the previous normal outputs of the flip-flops. When the down input is 1 and the up input is 0, the circuit counts down, since the complemented outputs of the previous flip-flops are applied to the $T$ inputs. When the up and down inputs are both 0, the circuit does not change state and remains

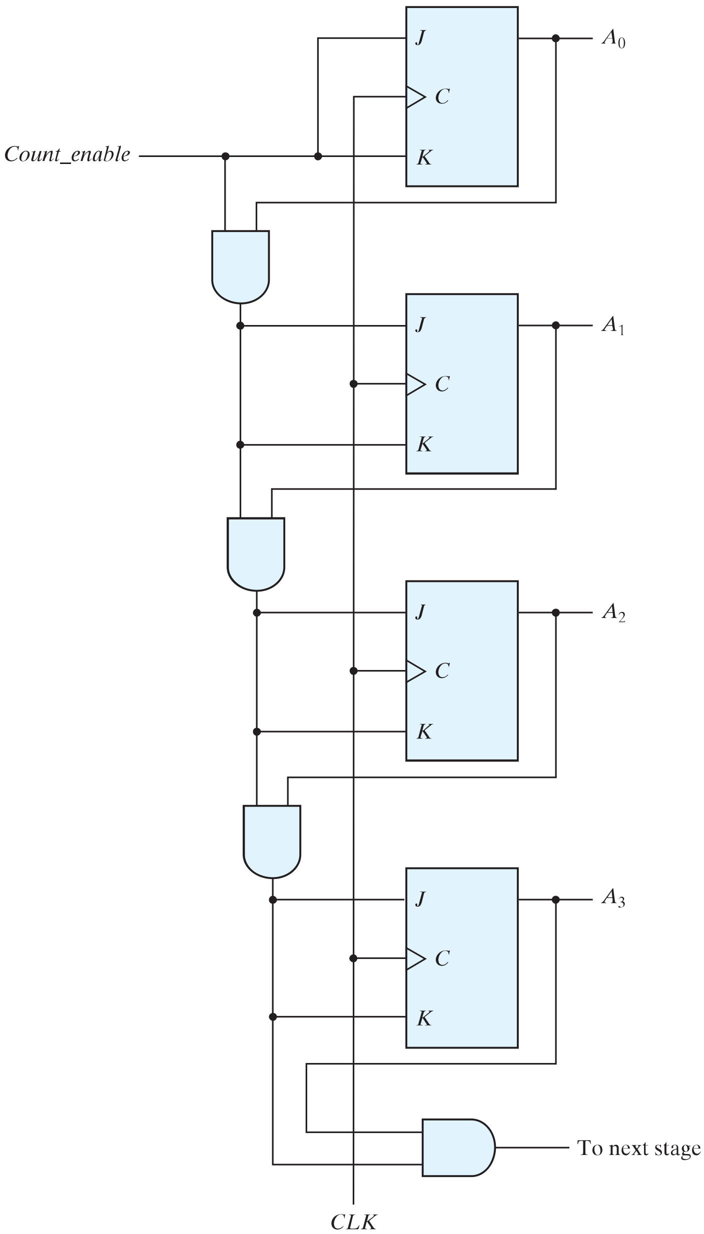


**FIGURE 6.12**
Four-bit synchronous binary counter

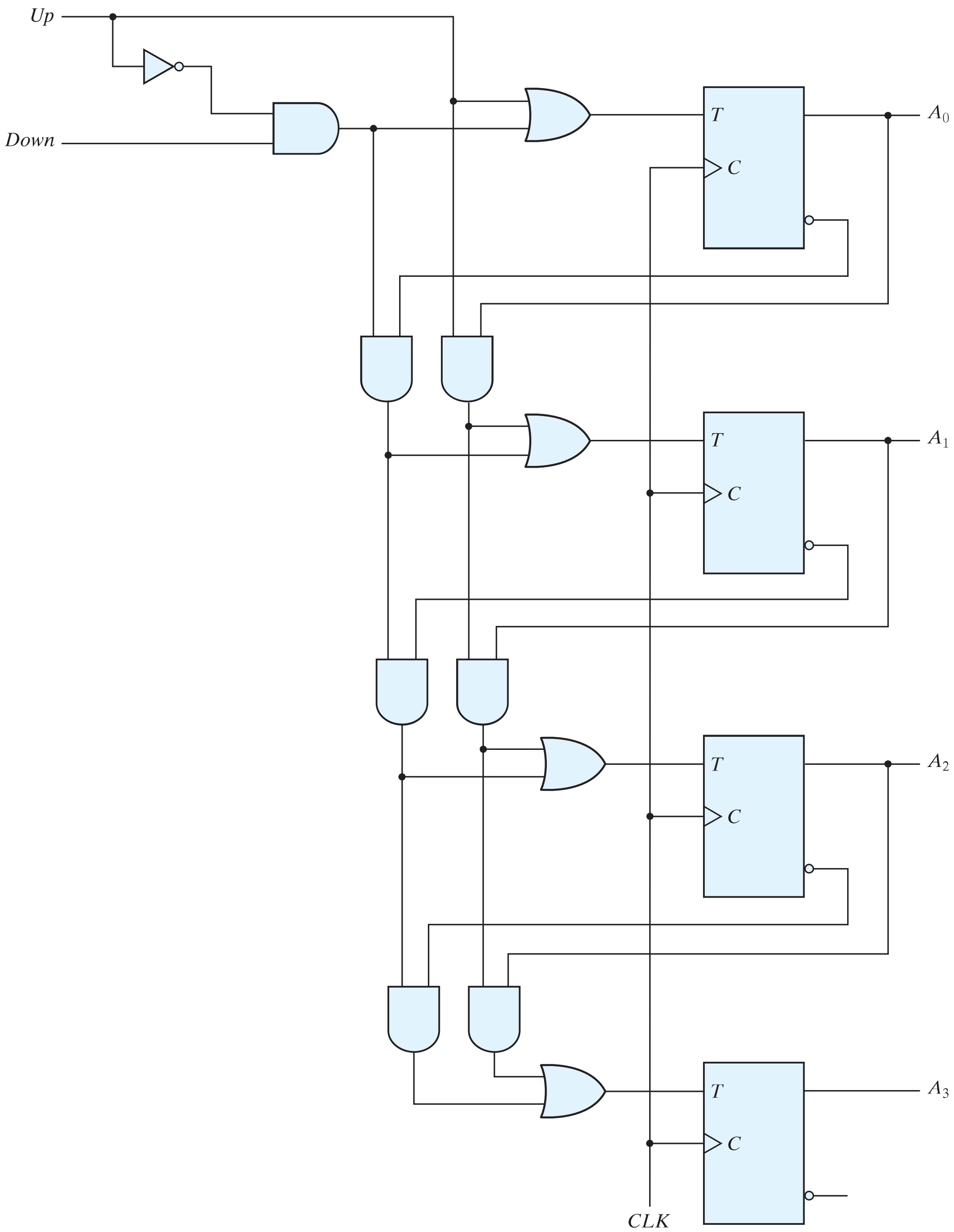


**FIGURE 6.13**
Four-bit up–down binary counter

in the same count. When the up and down inputs are both 1, the circuit counts up. This set of conditions ensures that only one operation is performed at any given time. Note that the up input has priority over the down input.

## BCD Counter

A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern, unlike a straight binary count. To derive the circuit of a BCD synchronous counter, it is necessary to go through a sequential circuit design procedure.

The state table of a BCD counter is listed in Table 6.5. The input conditions for the $T$ flip-flops are obtained from the present- and next-state conditions. Also shown in the table is an output $y$, which is equal to 1 when the present state is 1001. In this way, $y$ can enable the count of the next-higher significant decade while the same pulse switches the present decade from 1001 to 0000.

The flip-flop input equations can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are

$$
\begin{aligned}
T_{Q1} &= 1 \\
T_{Q2} &= Q'_8 Q_1 \\
T_{Q4} &= Q_2 Q_1 \\
T_{Q8} &= Q_8 Q_1 + Q_4 Q_2 Q_1 \\
y &= Q_8 Q_1
\end{aligned}
$$

The circuit can easily be drawn with four $T$ flip-flops, five AND gates, and one OR gate. Synchronous BCD counters can be cascaded to form a counter for decimal numbers of any length. The cascading is done as in Fig. 6.11, except that output $y$ must be connected to the count input of the next-higher significant decade.

**Table 6.5**
*State Table for BCD Counter*

| Present State | | | | Next State | | | | Output | Flip-Flop Inputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_8$ | $Q_4$ | $Q_2$ | $Q_1$ | $Q_8$ | $Q_4$ | $Q_2$ | $Q_1$ | $y$ | $TQ_8$ | $TQ_4$ | $TQ_2$ | $TQ_1$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |