for a collection of logic gates and flops that are controlled by a common clock. The ripple counter is not synchronous, even though it is controlled by a clock, because each flop has its own clock, which leads to the undesirable ripple output characteristic previously mentioned. A synchronous circuit has all of its flops transition at the same time so that they settle at the same time, with a resultant improvement in performance. Another benefit of synchronous logic is easier circuit analysis, because all flops change at the same time.

Designing a synchronous counter requires the addition of logic to calculate the next count value based on the current count value. Figure 1.15 shows a high-level block diagram of a synchronous counter and is also representative of synchronous logic in general. Synchronous circuits consist of state-full elements (flops), with *combinatorial logic* providing feedback to generate the next state based on the current state. *Combinatorial logic* is the term used to describe logic gates that have no state on their own. Inputs flow directly through combinatorial logic to outputs and must be captured by flops to preserve their state.

An example of synchronous logic design can be made of converting the three-bit ripple counter into a synchronous equivalent. Counters are a common logic structure, and they can be designed in a variety of ways. The Boolean equations for small counters may be directly solved using a truth table and K-map. Larger counters may be assembled in regular structures using binary adders that generate the next count value by adding 1 to the current value. A three-bit counter is easily handled with a truth-table methodology. The basic task is to create a truth table relating each possible current state to a next state as shown in Table 1.12.

TABLE 1.12 Three-Bit Counter Truth Table

Reset	Current State	Next State
1	XXX	000
0	000	001
0	001	010
0	010	011
0	011	100
0	100	101
0	101	110
0	110	111
0	111	000

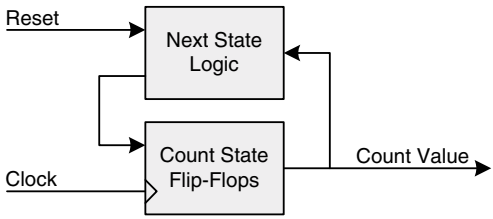

FIGURE 1.15 Synchronous counter block diagram.

Three Boolean equations are necessary, one for each bit that feeds back to the count state flops. If the flop inputs are labeled D[2:0], the outputs are labeled Q[2:0], and an active-high synchronous reset is defined, the following equations can be developed:

$$D[0] = \overline{Q[0]}\&\overline{RESET}$$

$$D[1] = \{(\overline{Q[0]}\&Q[1]) + (Q[0]\&\overline{Q[1]})\}\&\overline{RESET} = (Q[0] \oplus Q[1])\&\overline{RESET}$$

$$D[2] = \{(\overline{Q[2]}\&Q[1]\&Q[0]) + (Q[2]\&\overline{Q[1]}) + (Q[2]\&\overline{Q[0]})\}\&\overline{RESET}$$

Each equation's output is forced to 0 when RESET is asserted. Otherwise, the counter increments on each rising clock edge. Synchronous logic design allows any function to be implemented by changing the feedback logic. It would not be difficult to change the counter logic to count only odd or even numbers, or to count only as high as 5 before rolling over to 0. Unlike the ripple counter, whose structure supports a fixed counting sequence, next state logic can be defined arbitrarily according to an application's needs.

1.10 SYNCHRONOUS TIMING ANALYSIS

Logic elements, including flip-flops and gates, are physical devices that have finite response times to stimuli. Each of these elements exhibits a certain propagation delay between the time that an input is presented and the time that an output is generated. As more gates are chained together to create more complex logic functions, the overall propagation delay of signals between the end points increases. Flip-flops are triggered by the rising edge of a clock to load their new state, requiring that the input to the flip-flop is stable prior to the rising edge. Similarly, a flip-flop's output stabilizes at a new state some time after the rising edge. In between the output of a flip-flop and the input of another flip-flop is an arbitrary collection of logic gates, as seen in the preceding synchronous counter circuit. Synchronous timing analysis is the study of how the various delays in a synchronous circuit combine to limit the speed at which that circuit can operate. As might be expected, circuits with lesser delays are able to run faster.

A clock breaks time into discrete intervals that are each the duration of a single clock period. From a timing analysis perspective, each clock period is identical to the last, because each rising clock edge is a new flop triggering event. Therefore, timing analysis considers a circuit's delays over one clock period, between successive rising (or falling) clock edges. Knowing that a wide range of clock frequencies can be applied to a circuit, the question of time arises of how fast the clock can go before the circuit stops working reliably. The answer is that the clock must be slow enough to allow sufficient time for the output of a flop to stabilize, for the signal to propagate through the combinatorial logic gates, and for the input of the destination flop to stabilize. The clock must also be slow enough for the flop to reliably detect each edge. Each flop circuit is characterized by a minimum clock pulse width that must be met. Failing to meet this minimum time can result in the flop missing clock events.

Timing analysis revolves around the basic timing parameters of a flop: *input setup time* (t_{SU}), *input hold time* (t_H), and *clock-to-out time* (t_{CO}). Setup time specifies the time immediately preceding the rising edge of the clock by which the input must be stable. If the input changes too soon before the clock edge, the electrical circuitry within the flop will not have enough time to properly recognize the state of the input. Hold time places a restriction on how soon after the clock edge the input