contents are being shifted. A circuit in which data can be loaded in series and then accessed in parallel is called a series-to-parallel converter. Similarly, the opposite type of circuit is a parallel-to-series converter. The circuit in Figure 7.19 can perform both of these functions.

7.9 Counters

In Chapter 5 we dealt with circuits that perform arithmetic operations. We showed how adder/subtractor circuits can be designed, either using a simple cascaded (ripple-carry) structure that is inexpensive but slow or using a more complex carry-lookahead structure that is both more expensive and faster. In this section we examine special types of addition and subtraction operations, which are used for the purpose of counting. In particular, we want to design circuits that can increment or decrement a count by 1. Counter circuits are used in digital systems for many purposes. They may count the number of occurrences of certain events, generate timing intervals for control of various tasks in a system, keep track of time elapsed between specific events, and so on.

Counters can be implemented using the adder/subtractor circuits discussed in Chapter 5 and the registers discussed in section 7.8. However, since we only need to change the contents of a counter by 1, it is not necessary to use such elaborate circuits. Instead, we can use much simpler circuits that have a significantly lower cost. We will show how the counter circuits can be designed using T and D flip-flops.

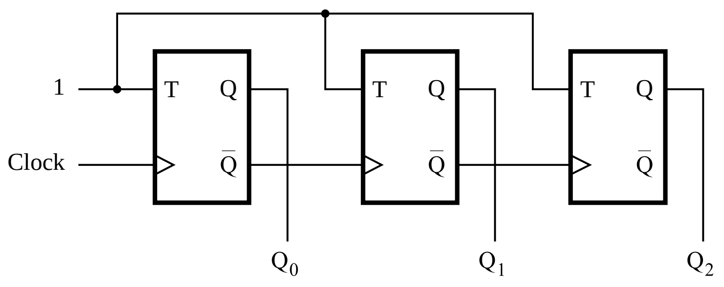
7.9.1 Asynchronous Counters

The simplest counter circuits can be built using T flip-flops because the toggle feature is naturally suited for the implementation of the counting operation.

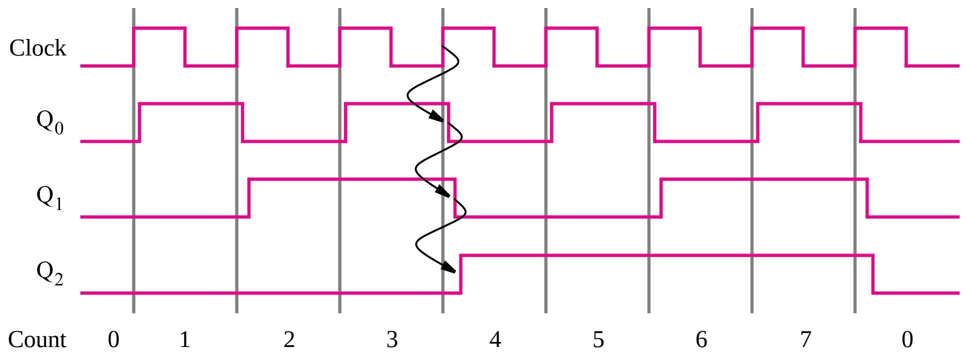
Up-Counter with T Flip-Flops

Figure 7.20*a* gives a three-bit counter capable of counting from 0 to 7. The clock inputs of the three flip-flops are connected in cascade. The *T* input of each flip-flop is connected to a constant 1, which means that the state of the flip-flop will be reversed (toggled) at each positive edge of its clock. We are assuming that the purpose of this circuit is to count the number of pulses that occur on the primary input called *Clock*. Thus the clock input of the first flip-flop is connected to the *Clock* line. The other two flip-flops have their clock inputs driven by the $\overline{Q}$ output of the preceding flip-flop. Therefore, they toggle their state whenever the preceding flip-flop changes its state from $Q = 1$ to $Q = 0$, which results in a positive edge of the $\overline{Q}$ signal.

Figure 7.20*b* shows a timing diagram for the counter. The value of Q_0 toggles once each clock cycle. The change takes place shortly after the positive edge of the *Clock* signal. The delay is caused by the propagation delay through the flip-flop. Since the second flip-flop is clocked by $\overline{Q}_0$, the value of Q_1 changes shortly after the negative edge of the Q_0 signal. Similarly, the value of Q_2 changes shortly after the negative edge of the Q_1 signal. If we look at the values $Q_2Q_1Q_0$ as the count, then the timing diagram indicates that the counting sequence is 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, and so on. This circuit is a modulo-8 counter. Because it counts in the upward direction, we call it an *up-counter*.



(a) Circuit



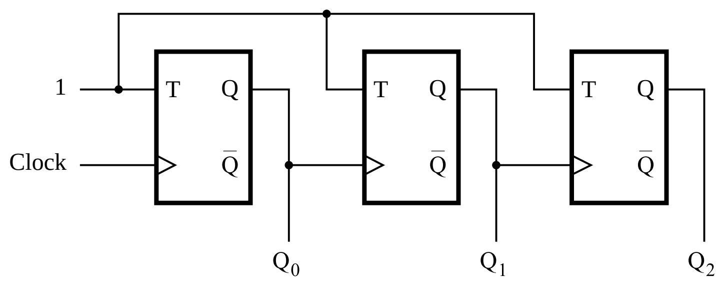
(b) Timing diagram

Figure 7.20 A three-bit up-counter.

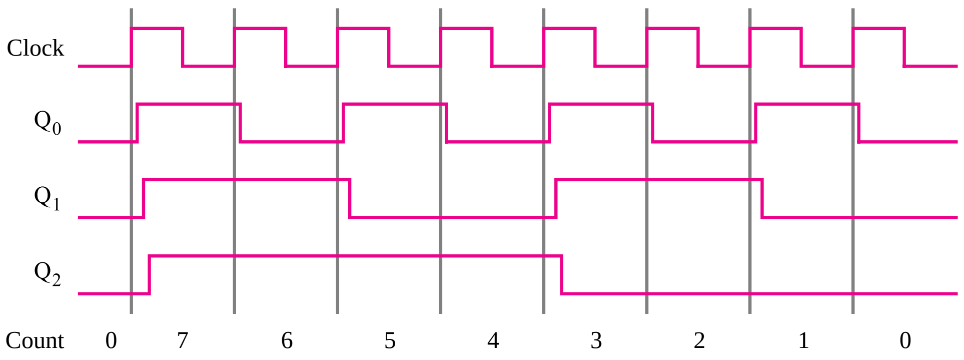
The counter in Figure 7.20*a* has three *stages*, each comprising a single flip-flop. Only the first stage responds directly to the *Clock* signal; we say that this stage is *synchronized* to the clock. The other two stages respond after an additional delay. For example, when *Count* = 3, the next clock pulse will cause the *Count* to go to 4. As indicated by the arrows in the timing diagram in Figure 7.20*b*, this change requires the toggling of the states of all three flip-flops. The change in Q_0 is observed only after a propagation delay from the positive edge of *Clock*. The Q_1 and Q_2 flip-flops have not yet changed; hence for a brief time the count is $Q_2Q_1Q_0 = 010$. The change in Q_1 appears after a second propagation delay, at which point the count is 000. Finally, the change in Q_2 occurs after a third delay, at which point the stable state of the circuit is reached and the count is 100. This behavior is similar to the rippling of carries in the ripple-carry adder circuit of Figure 5.6. The circuit in Figure 7.20*a* is an *asynchronous counter*, or a *ripple counter*.

Down-Counter with T Flip-Flops

A slight modification of the circuit in Figure 7.20*a* is presented in Figure 7.21*a*. The only difference is that in Figure 7.21*a* the clock inputs of the second and third flip-flops are driven by the Q outputs of the preceding stages, rather than by the $\overline{Q}$ outputs. The timing diagram, given in Figure 7.21*b*, shows that this circuit counts in the sequence 0, 7, 6, 5, 4,



(a) Circuit



(b) Timing diagram

Figure 7.21 A three-bit down-counter.

3, 2, 1, 0, 7, and so on. Because it counts in the downward direction, we say that it is a *down-counter*.

It is possible to combine the functionality of the circuits in Figures 7.20*a* and 7.21*a* to form a counter that can count either up or down. Such a counter is called an *up/down-counter*. We leave the derivation of this counter as an exercise for the reader (problem 7.16).

7.9.2 Synchronous Counters

The asynchronous counters in Figures 7.20*a* and 7.21*a* are simple, but not very fast. If a counter with a larger number of bits is constructed in this manner, then the delays caused by the cascaded clocking scheme may become too long to meet the desired performance requirements. We can build a faster counter by clocking all flip-flops at the same time, using the approach described below.

Synchronous Counter with T Flip-Flops

Table 7.1 shows the contents of a three-bit up-counter for eight consecutive clock cycles, assuming that the count is initially 0. Observing the pattern of bits in each row of

Table 7.1 Derivation of the synchronous up-counter.

Clock cycle	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

Q_1 changes

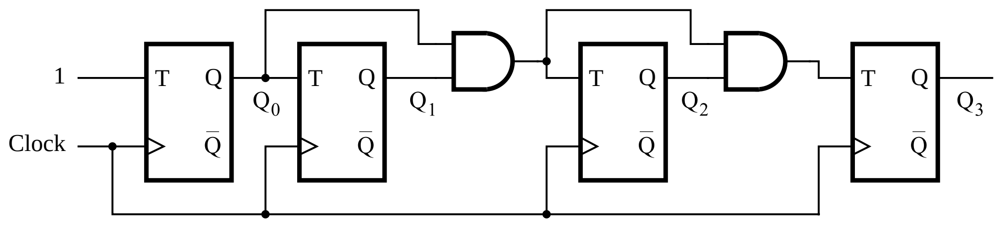
Q_2 changes

the table, it is apparent that bit Q_0 changes on each clock cycle. Bit Q_1 changes only when $Q_0 = 1$. Bit Q_2 changes only when both Q_1 and Q_0 are equal to 1. In general, for an n-bit up-counter, a given flip-flop changes its state only when all the preceding flip-flops are in the state $Q = 1$. Therefore, if we use T flip-flops to realize the counter, then the T inputs are defined as

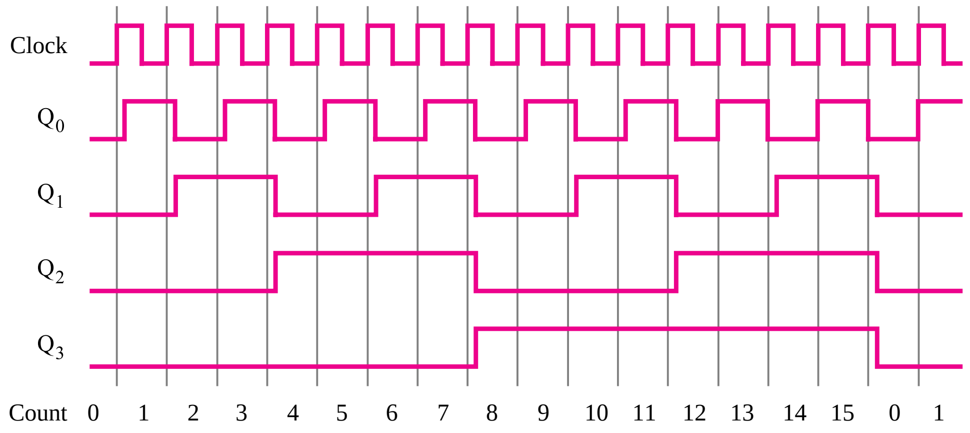
$$\begin{aligned}
T_0 &= 1\\
T_1 &= Q_0\\
T_2 &= Q_0Q_1\\
T_3 &= Q_0Q_1Q_2\\
&\cdot\\
&\cdot\\
&\cdot\\
T_n &= Q_0Q_1 \cdots Q_{n-1}
\end{aligned}$$

An example of a four-bit counter based on these expressions is given in Figure 7.22*a*. Instead of using AND gates of increased size for each stage, which may lead to fan-in problems, we use a factored arrangement, as shown in the figure. This arrangement does not slow down the response of the counter, because all flip-flops change their states after a propagation delay from the positive edge of the clock. Note that a change in the value of Q_0 may have to propagate through several AND gates to reach the flip-flops in the higher stages of the counter, which requires a certain amount of time. This time must not exceed the clock period. Actually, it must be less than the clock period minus the setup time for the flip-flops.

Figure 7.22*b* gives a timing diagram. It shows that the circuit behaves as a modulo-16 up-counter. Because all changes take place with the same delay after the active edge of the *Clock* signal, the circuit is called a *synchronous counter*.



(a) Circuit



(b) Timing diagram

Figure 7.22 A four-bit synchronous up-counter.

Enable and Clear Capability

The counters in Figures 7.20 through 7.22 change their contents in response to each clock pulse. Often it is desirable to be able to inhibit counting, so that the count remains in its present state. This may be accomplished by including an *Enable* control signal, as indicated in Figure 7.23. The circuit is the counter of Figure 7.22, where the *Enable* signal controls directly the T input of the first flip-flop. Connecting the *Enable* also to the AND-gate chain means that if $Enable = 0$, then all T inputs will be equal to 0. If $Enable = 1$, then the counter operates as explained previously.

In many applications it is necessary to start with the count equal to zero. This is easily achieved if the flip-flops can be cleared, as explained in section 7.4.3. The clear inputs on all flip-flops can be tied together and driven by a *Clear* control input.

Synchronous Counter with D Flip-Flops

While the toggle feature makes T flip-flops a natural choice for the implementation of counters, it is also possible to build counters using other types of flip-flops. The JK

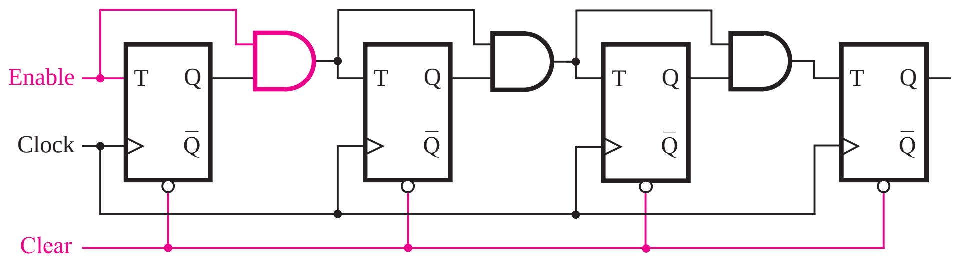


Figure 7.23 Inclusion of Enable and Clear capability.

flip-flops can be used in exactly the same way as the T flip-flops because if the J and K inputs are tied together, a JK flip-flop becomes a T flip-flop. We will now consider using D flip-flops for this purpose.

It is not obvious how D flip-flops can be used to implement a counter. We will present a formal method for deriving such circuits in Chapter 8. Here we will present a circuit structure that meets the requirements but will leave the derivation for Chapter 8. Figure 7.24 gives a four-bit up-counter that counts in the sequence 0, 1, 2, ..., 14, 15, 0, 1, and so on. The count is indicated by the flip-flop outputs $Q_3Q_2Q_1Q_0$. If we assume that $Enable = 1$, then the D inputs of the flip-flops are defined by the expressions

$$\begin{aligned} D_0 &= \overline{Q}_0 = 1 \oplus Q_0 \\ D_1 &= Q_1 \oplus Q_0 \\ D_2 &= Q_2 \oplus Q_1Q_0 \\ D_3 &= Q_3 \oplus Q_2Q_1Q_0 \end{aligned}$$

For a larger counter the ith stage is defined by

$$D_i = Q_i \oplus Q_{i-1}Q_{i-2}\cdots Q_1Q_0$$

We will show how to derive these equations in Chapter 8.

We have included the *Enable* control signal so that the counter counts the clock pulses only if $Enable = 1$. In effect, the above equations are modified to implement the circuit in the figure as follows

$$\begin{aligned} D_0 &= Q_0 \oplus Enable \\ D_1 &= Q_1 \oplus Q_0 \cdot Enable \\ D_2 &= Q_2 \oplus Q_1 \cdot Q_0 \cdot Enable \\ D_3 &= Q_3 \oplus Q_2 \cdot Q_1 \cdot Q_0 \cdot Enable \end{aligned}$$

The operation of the counter is based on our observation for Table 7.1 that the state of the flip-flop in stage i changes only if all preceding flip-flops are in the state $Q = 1$. This makes the output of the AND gate that feeds stage i equal to 1, which causes the output of the XOR gate connected to D_i to be equal to $\overline{Q}_i$. Otherwise, the output of the XOR gate provides $D_i = Q_i$, and the flip-flop remains in the same state. This resembles the carry propagation in a carry-lookahead adder circuit (see section 5.4); hence the AND-gate chain

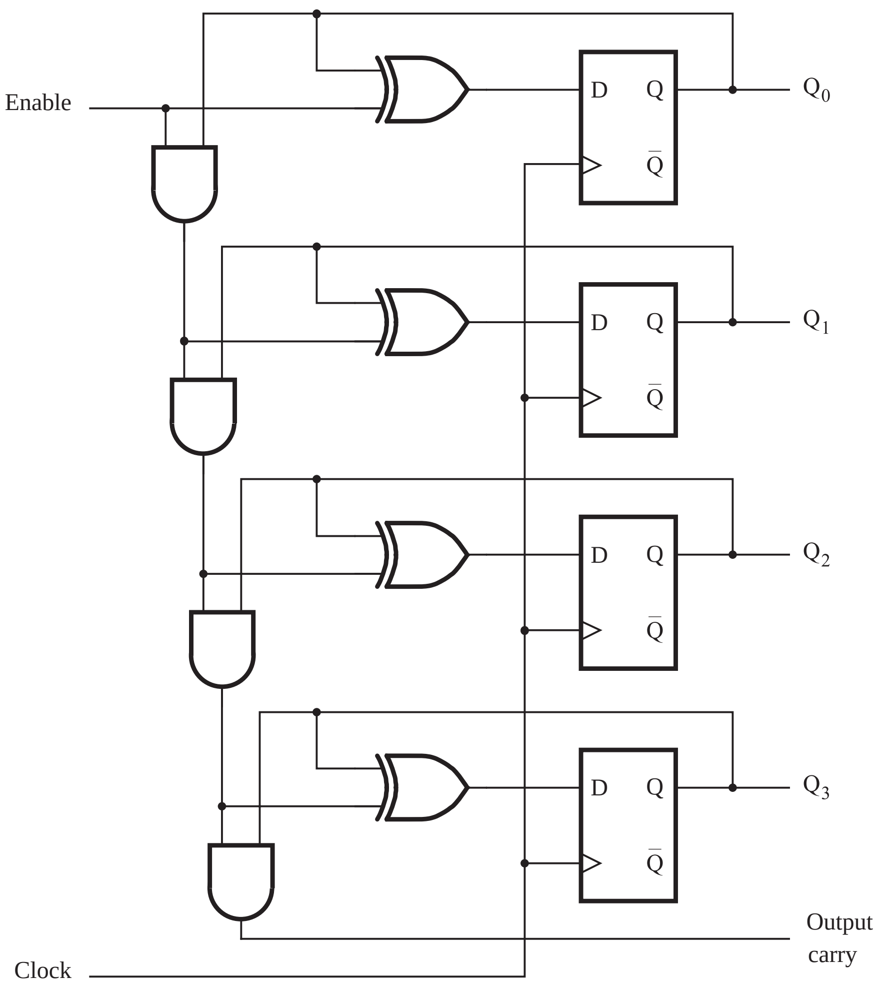


Figure 7.24 A four-bit counter with D flip-flops.

can be thought of as the *carry chain*. Even though the circuit is only a four-bit counter, we have included an extra AND gate that produces the "output carry." This signal makes it easy to concatenate two such four-bit counters to create an eight-bit counter.

Finally, the reader should note that the counter in Figure 7.24 is essentially the same as the circuit in Figure 7.23. We showed in Figure 7.16*a* that a T flip-flop can be formed from a D flip-flop by providing the extra gating that gives

$$\begin{aligned} D &= \mathrm{Q}\overline{T} + \overline{\mathrm{Q}}T \\ &= \mathrm{Q} \oplus T \end{aligned}$$

Thus in each stage in Figure 7.24, the D flip-flop and the associated XOR gate implement the functionality of a T flip-flop.

7.9.3 Counters with Parallel Load

Often it is necessary to start counting with the initial count being equal to 0. This state can be achieved by using the capability to clear the flip-flops as indicated in Figure 7.23. But sometimes it is desirable to start with a different count. To allow this mode of operation, a counter circuit must have some inputs through which the initial count can be loaded. Using the *Clear* and *Preset* inputs for this purpose is a possibility, but a better approach is discussed below.

The circuit of Figure 7.24 can be modified to provide the parallel-load capability as shown in Figure 7.25. A two-input multiplexer is inserted before each *D* input. One input to the multiplexer is used to provide the normal counting operation. The other input is a data bit that can be loaded directly into the flip-flop. A control input, *Load*, is used to choose the mode of operation. The circuit counts when $Load = 0$. A new initial value, $D_3D_2D_1D_0$, is loaded into the counter when $Load = 1$.

7.10 Reset Synchronization

We have already mentioned that it is important to be able to clear, or *reset*, the contents of a counter prior to commencing a counting operation. This can be done using the clear capability of the individual flip-flops. But we may also be interested in resetting the count to 0 during the normal counting process. An n-bit up-counter functions naturally as a modulo-2^n counter. Suppose that we wish to have a counter that counts modulo some base that is not a power of 2. For example, we may want to design a modulo-6 counter, for which the counting sequence is 0, 1, 2, 3, 4, 5, 0, 1, and so on.

The most straightforward approach is to recognize when the count reaches 5 and then reset the counter. An AND gate can be used to detect the occurrence of the count of 5. Actually, it is sufficient to ascertain that $Q_2 = Q_0 = 1$, which is true only for 5 in our desired counting sequence. A circuit based on this approach is given in Figure 7.26*a*. It uses a three-bit synchronous counter of the type depicted in Figure 7.25. The parallel-load feature of the counter is used to reset its contents when the count reaches 5. The resetting action takes place at the positive clock edge after the count has reached 5. It involves loading $D_2D_1D_0 = 000$ into the flip-flops. As seen in the timing diagram in Figure 7.26*b*, the desired counting sequence is achieved, with each value of the count being established for one full clock cycle. Because the counter is reset on the active edge of the clock, we say that this type of counter has a *synchronous reset*.

Consider now the possibility of using the clear feature of individual flip-flops, rather than the parallel-load approach. The circuit in Figure 7.27*a* illustrates one possibility. It uses the counter structure of Figure 7.22*a*. Since the clear inputs are active when low, a

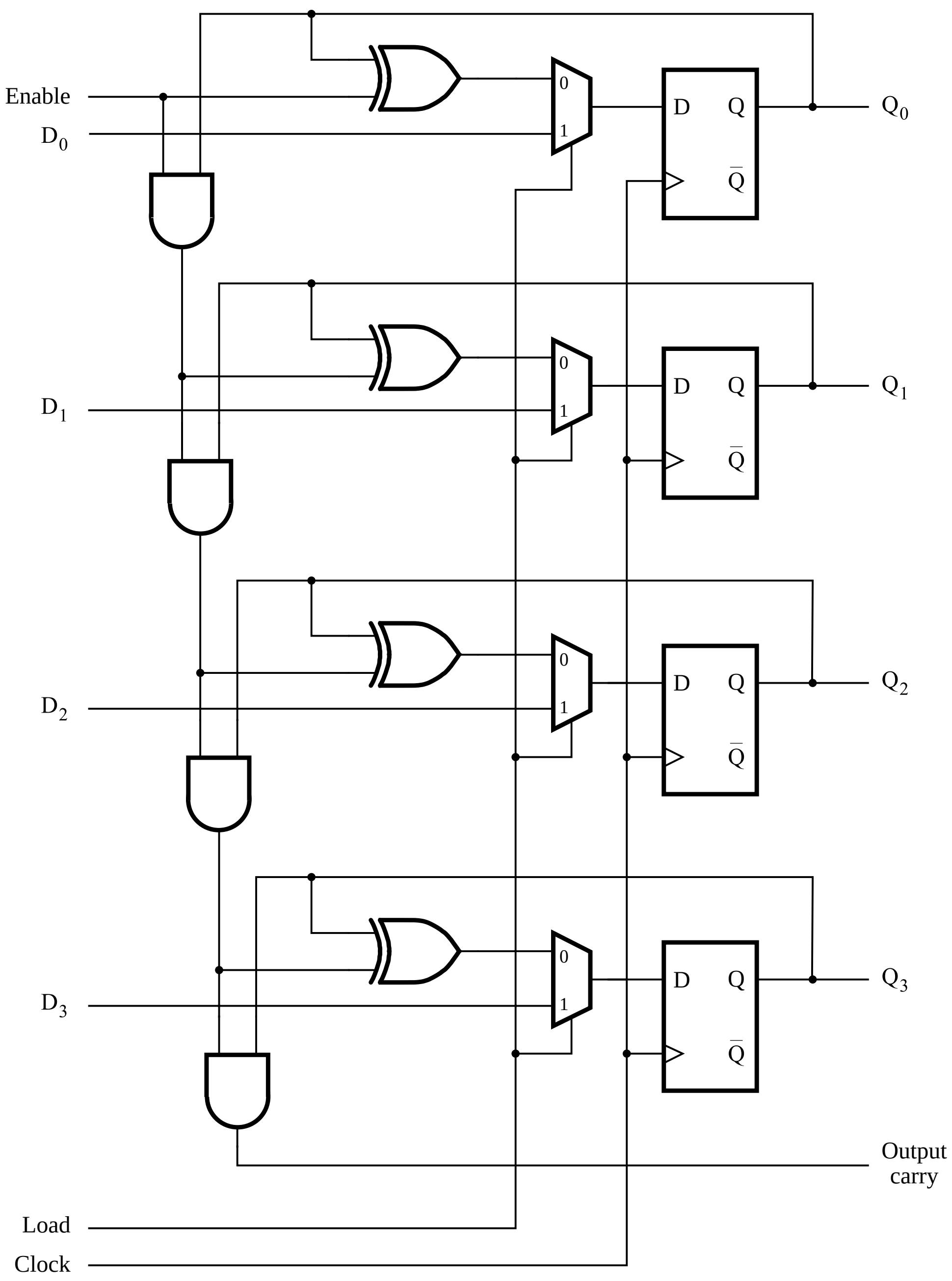
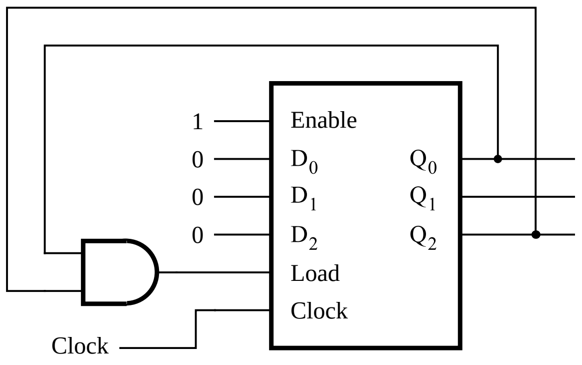
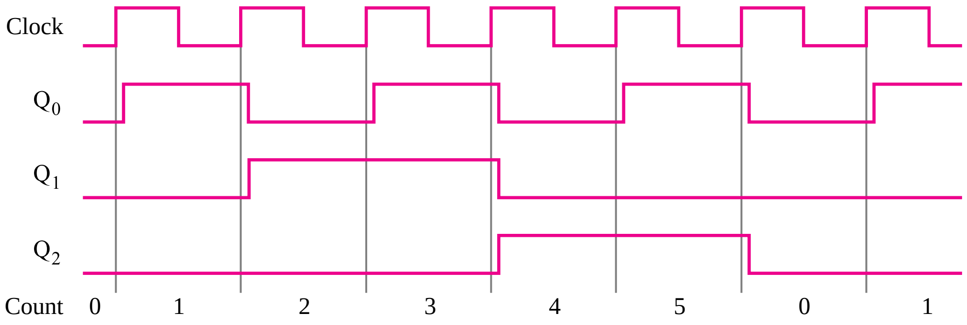


Figure 7.25 A counter with parallel-load capability.



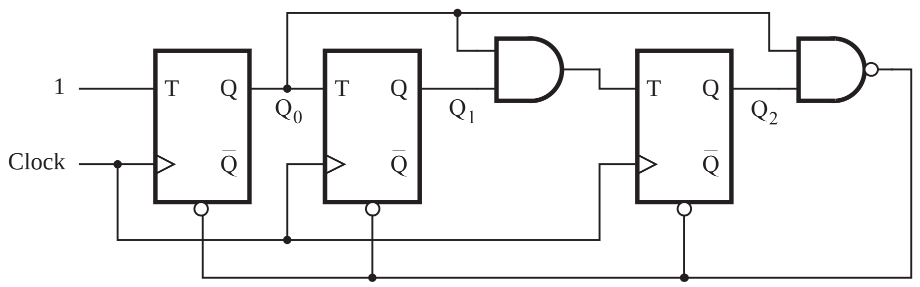
(a) Circuit



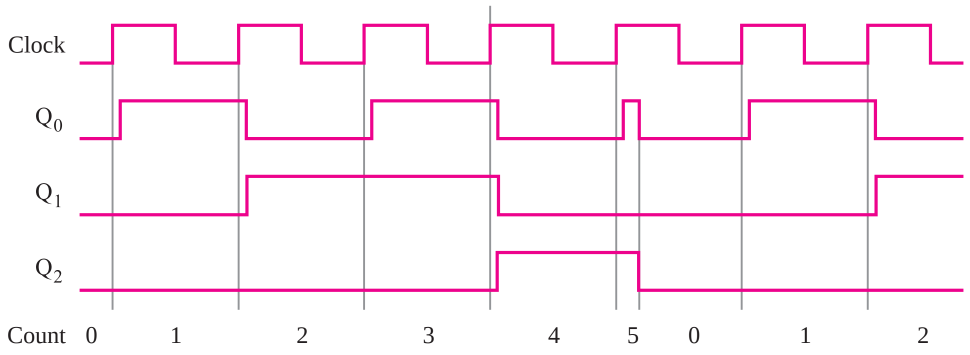
(b) Timing diagram

Figure 7.26 A modulo-6 counter with synchronous reset.

NAND gate is used to detect the occurrence of the count of 5 and cause the clearing of all three flip-flops. Conceptually, this seems to work fine, but closer examination reveals a potential problem. The timing diagram for this circuit is given in Figure 7.27*b*. It shows a difficulty that arises when the count is equal to 5. As soon as the count reaches this value, the NAND gate triggers the resetting action. The flip-flops are cleared to 0 a short time after the NAND gate has detected the count of 5. This time depends on the gate delays in the circuit, but not on the clock. Therefore, signal values $Q_2Q_1Q_0 = 101$ are maintained for a time that is much less than a clock cycle. Depending on a particular application of such a counter, this may be adequate, but it may also be completely unacceptable. For example, if the counter is used in a digital system where all operations in the system are synchronized by the same clock, then this narrow pulse denoting *Count* $= 5$ would not be seen by the



(a) Circuit



(b) Timing diagram

Figure 7.27 A modulo-6 counter with asynchronous reset.

rest of the system. To solve this problem, we could try to use a modulo-7 counter instead, assuming that the system would ignore the short pulse that denotes the count of 6. This is not a good way of designing circuits, because undesirable pulses often cause unforeseen difficulties in practice. The approach employed in Figure 7.27*a* is said to use *asynchronous reset*.

The timing diagrams in Figures 7.26*b* and 7.27*b* suggest that synchronous reset is a better choice than asynchronous reset. The same observation is true if the natural counting sequence has to be broken by loading some value other than zero. The new value of the count can be established cleanly using the parallel-load feature. The alternative of using the clear and preset capability of individual flip-flops to set their states to reflect the desired count has the same problems as discussed in conjunction with the asynchronous reset.

7.11 Other Types of Counters

In this section we discuss three other types of counters that can be found in practical applications. The first uses the decimal counting sequence, and the other two generate sequences of codes that do not represent binary numbers.

7.11.1 BCD Counter

Binary-coded-decimal (BCD) counters can be designed using the approach explained in section 7.10. A two-digit BCD counter is presented in Figure 7.28. It consists of two modulo-10 counters, one for each BCD digit, which we implemented using the parallel-load four-bit counter of Figure 7.25. Note that in a modulo-10 counter it is necessary to reset the four flip-flops after the count of 9 has been obtained. Thus the *Load* input to each

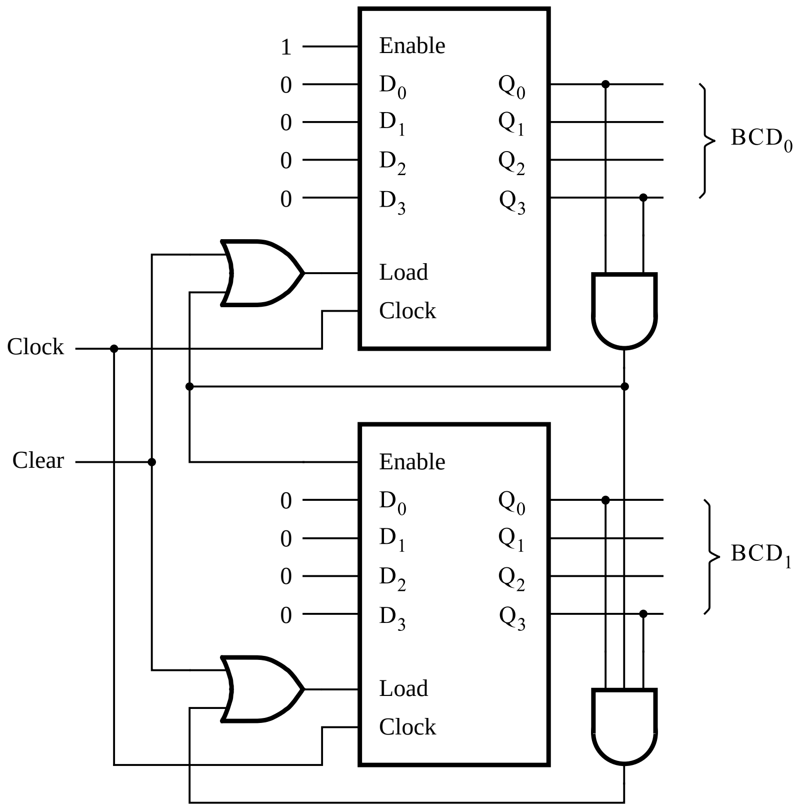


Figure 7.28 A two-digit BCD counter.

stage is equal to 1 when $Q_3 = Q_0 = 1$, which causes 0s to be loaded into the flip-flops at the next positive edge of the clock signal. Whenever the count in stage 0, BCD_0, reaches 9 it is necessary to enable the second stage so that it will be incremented when the next clock pulse arrives. This is accomplished by keeping the *Enable* signal for BCD_1 low at all times except when $BCD_0 = 9$.

In practice, it has to be possible to clear the contents of the counter by activating some control signal. Two OR gates are included in the circuit for this purpose. The control input *Clear* can be used to load 0s into the counter. Observe that in this case *Clear* is active when high. VHDL code for a two-digit BCD counter is given in Figure 7.77.

In any digital system there is usually one or more clock signals used to drive all synchronous circuitry. In the preceding counter, as well as in all counters presented in the previous figures, we have assumed that the objective is to count the number of clock pulses. Of course, these counters can be used to count the number of pulses in any signal that may be used in place of the clock signal.

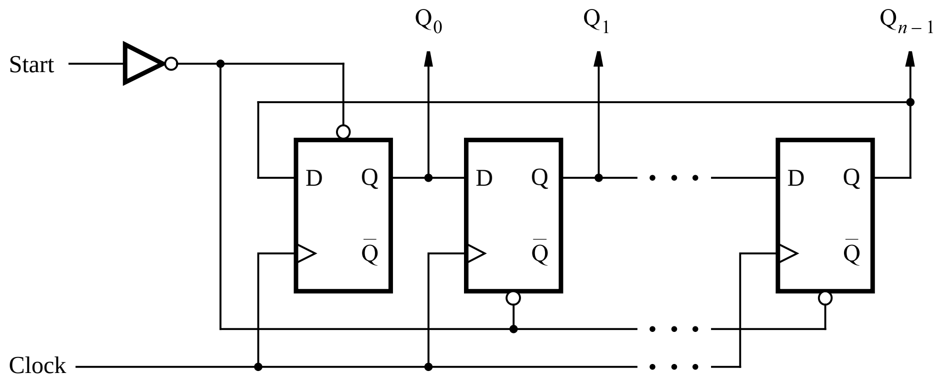
7.11.2 RING COUNTER

In the preceding counters the count is indicated by the state of the flip-flops in the counter. In all cases the count is a binary number. Using such counters, if an action is to be taken as a result of a particular count, then it is necessary to detect the occurrence of this count. This may be done using AND gates, as illustrated in Figures 7.26 through 7.28.

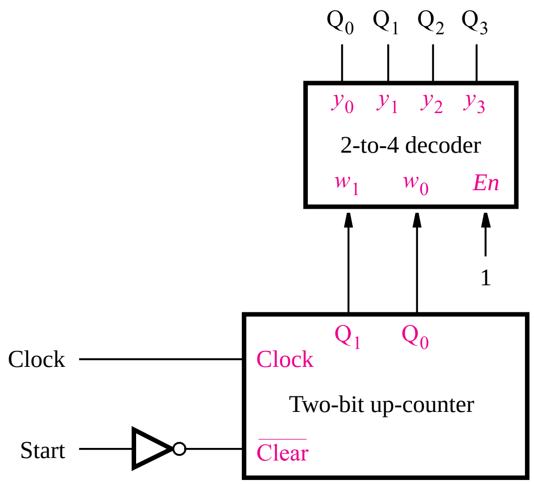
It is possible to devise a counterlike circuit in which each flip-flop reaches the state $Q_i = 1$ for exactly one count, while for all other counts $Q_i = 0$. Then Q_i indicates directly an occurrence of the corresponding count. Actually, since this does not represent binary numbers, it is better to say that the outputs of the flips-flops represent a code. Such a circuit can be constructed from a simple shift register, as indicated in Figure 7.29*a*. The Q output of the last stage in the shift register is fed back as the input to the first stage, which creates a ringlike structure. If a single 1 is injected into the ring, this 1 will be shifted through the ring at successive clock cycles. For example, in a four-bit structure, the possible codes $Q_0Q_1Q_2Q_3$ will be 1000, 0100, 0010, and 0001. As we said in section 6.2, such encoding, where there is a single 1 and the rest of the code variables are 0, is called a *one-hot code*.

The circuit in Figure 7.29*a* is referred to as a *ring counter*. Its operation has to be initialized by injecting a 1 into the first stage. This is achieved by using the *Start* control signal, which presets the left-most flip-flop to 1 and clears the others to 0. We assume that all changes in the value of the *Start* signal occur shortly after an active clock edge so that the flip-flop timing parameters are not violated.

The circuit in Figure 7.29*a* can be used to build a ring counter with any number of bits, n. For the specific case of $n = 4$, part (*b*) of the figure shows how a ring counter can be constructed using a two-bit up-counter and a decoder. When *Start* is set to 1, the counter is reset to 00. After *Start* changes back to 0, the counter increments its value in the normal way. The 2-to-4 decoder, described in section 6.2, changes the counter output into a one-hot code. For the count values 00, 01, 10, 11, 00, and so on, the decoder produces $Q_0Q_1Q_2Q_3 = 1000, 0100, 0010, 0001, 1000$, and so on. This circuit structure can be used for larger ring counters, as long as the number of bits is a power of two. We will give an example of a larger circuit that uses the ring counter in Figure 7.29*b* as a subcircuit in section 7.14.



(a) An *n*-bit ring counter



(b) A four-bit ring counter

Figure 7.29 Ring counter.

7.11.3 Johnson Counter

An interesting variation of the ring counter is obtained if, instead of the Q output, we take the $\overline{Q}$ output of the last stage and feed it back to the first stage, as shown in Figure 7.30. This circuit is known as a *Johnson counter*. An *n*-bit counter of this type generates a counting sequence of length $2n$. For example, a four-bit counter produces the sequence 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, and so on. Note that in this sequence, only a single bit has a different value for two consecutive codes.

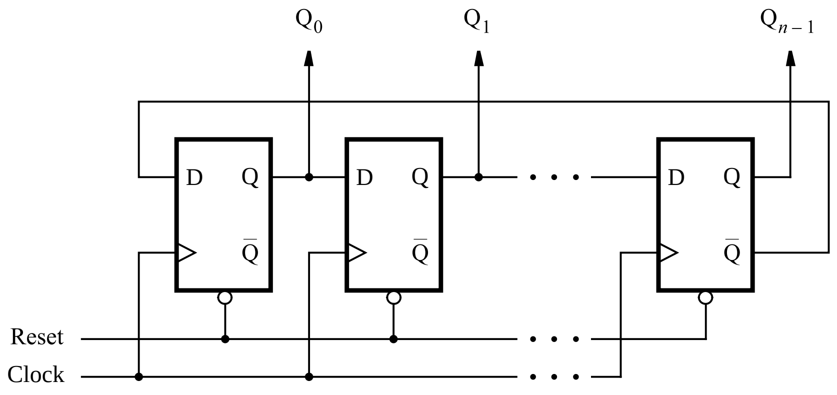


Figure 7.30 Johnson counter.

To initialize the operation of the Johnson counter, it is necessary to reset all flip-flops, as shown in the figure. Observe that neither the Johnson nor the ring counter will generate the desired counting sequence if not initialized properly.

7.11.4 Remarks on Counter Design

The sequential circuits presented in this chapter, namely, registers and counters, have a regular structure that allows the circuits to be designed using an intuitive approach. In Chapter 8 we will present a more formal approach to design of sequential circuits and show how the circuits presented in this chapter can be derived using this approach.

7.12 Using Storage Elements with CAD Tools

This section shows how circuits with storage elements can be designed using either schematic capture or VHDL code.

7.12.1 Including Storage Elements in Schematics

One way to create a circuit is to draw a schematic that builds latches and flip-flops from logic gates. Because these storage elements are used in many applications, most CAD systems provide them as prebuilt modules. Figure 7.31 shows a schematic created with a schematic capture tool, which includes three types of flip-flops that are imported from a library provided as part of the CAD system. The top element is a gated D latch, the middle element is a positive-edge-triggered D flip-flop, and the bottom one is a positive-edge-triggered T flip-flop. The D and T flip-flops have asynchronous, active-low clear and