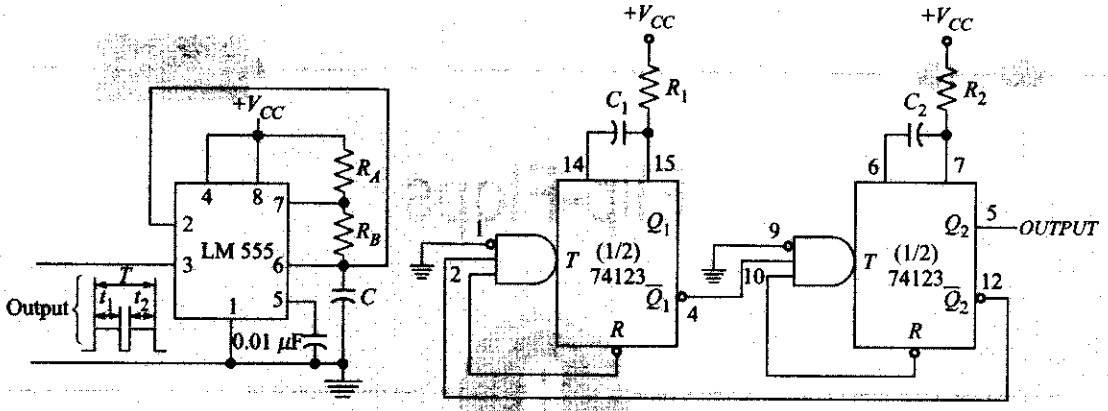


Work element: Study the working of IC 555 and 74123, and understand the different input outputs. From above relations, calculate the resistance and capacitance values. See the waveform in oscilloscope. Calculate duty cycle from the oscilloscope reading and compare with theoretical value. Conduct similar exercise for 74123 based circuit as shown. Repeat the experiment with other combinations of resistance and capacitance values.

Answers to Self-tests

1. An input is sensitive to PTs, and the circuit output changes synchronously with PTs. The circuit output changes in synchronism with NTs.
2. It means that a circuit input is sensitive to PTs. (See Fig. 7.6b.)
3. The logic symbol for an input sensitive to NTs is a bubble in front of a dynamic input indicator. (See Fig. 7.7b.)
4. A series mode offers low impedance at resonance, thus providing positive feedback for oscillation. A parallel mode offers high impedance at resonance, and thus provides insufficient feedback to produce oscillation.
5. Unnecessary. They simply simulate a load condition.
6. It means that the circuit has two input threshold voltage levels—an upper threshold and a lower threshold. By contrast, a simple inverter has only a single threshold voltage level.
7. Noninverting: the input and output are both high (or both low) at the same time (no phase shift). Inverting: 180° phase shift between input and output.
8. Schmitt triggers can be used to clean up a noisy signal or to change a signal having a slow rise time into one having a fast rise time.
9. A circuit has two output states, neither of which is stable.
10. True
11. Inversely
12. A circuit has two output states, one of which is stable.
13. True
14. The stable state is low.
15. Nonretriggerable
16. True
17. 0.69
18. Glitches are the unwanted pulses appearing at the output of a gate when two or more inputs change state simultaneously.
19. A strobe pulse is a pulse timed to eliminate glitches.

Flip-Flops

8

OBJECTIVES

- **Describe the operation of the basic RS flip-flop and explain the purpose of the additional input on the gated (clocked) RS flip-flop**
- **Show the truth table for the edge-triggered. RS flip-flop, edge-triggered D flip-flop, and edge-triggered JK flip-flop**
- **Discuss some of the timing problems related to flip-flops**
- **Draw a diagram of a JK master-slave flip-flop and describe its operation**
- **State the cause of contact bounce and describe a solution for this problem**
- **Describe characteristic equations of Flip-Flops and analysis techniques of sequential circuit**
- **Describe excitation table of Flip-Flops and explain conversion of Flip-Flops as synthesis example**

The outputs of the digital circuits considered previously are dependent entirely on their inputs. That is, if an input changes state, output may also change state. However, there are requirements for a digital device or circuit whose output will remain unchanged, once set, even if there is a change in input level(s). Such a device could be used to store a binary number. A flip-flop is one such circuit, and the characteristics of the most common types of flip-flops used in digital systems are considered in this chapter. Flip-flops are used in the construction of registers and counters, and in numerous other applications. The elimination of switch contact bounce is a clever application utilizing the unique operating characteristics of flip-flops. In a sequential logic circuit flip-flops serve as key memory elements. Analysis of such circuits are done through truth tables or characteristic equations of flip-flops. The analysis result is normally presented through state

table or state transition diagram and also through timing diagram. Conversion of flip-flop from one kind to another can be posed as a synthesis problem where flip-flop excitation tables are very useful.

8.1 *RS* FLIP-FLOPS

Any device or circuit that has two stable states is said to be *bistable*. For instance, a toggle switch has two stable states. It is either up or down, depending on the position of the switch as shown in Fig. 8.1a. The switch is also said to have *memory* since it will remain as set until someone changes its position.

A *flip-flop* is a bistable electronic circuit that has two stable states—that is, its output is either 0 or +5 Vdc as shown in Fig. 8.1b. The flip-flop also has memory since its output will remain as set until something is done to change it. As such, the flip-flop (or the switch) can be regarded as a memory device. In fact, any bistable device can be used to store one binary digit (bit). For instance, when the flip-flop has its output set at 0 Vdc, it can be regarded as storing a logic 0 and when its output is set at +5 Vdc, as storing a logic 1. The flip-flop is often called a *latch*, since it will hold, or latch, in either stable state.

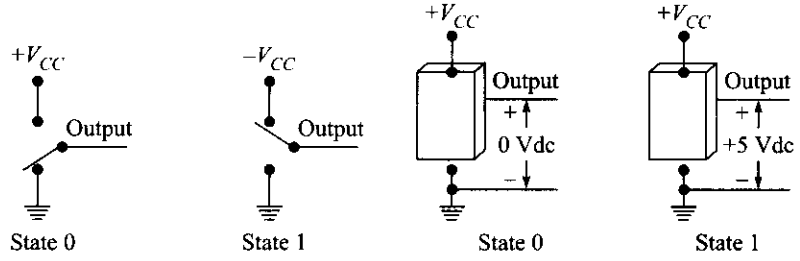


Fig. 8.1 Bistable devices

Basic Idea

One of the easiest ways to construct a flip-flop is to connect two inverters in series as shown in Fig. 8.2a. The line connecting the output of inverter *B* (*INV B*) back to the input of inverter *A* (*INV A*) is referred to as the *feedback line.*

For the moment, remove the feedback line and consider V_1 as the input and V_3 as the output as shown in Fig. 8.2b. There are only two possible signals in a digital system, and in this case we will define $L = 0 = 0$ Vdc and $H = 1 = +5$ Vdc. If V_1 is set to 0 Vdc, then V_3 will also be 0 Vdc. Now, if the feedback line shown in Fig. 8.2b is reconnected, the ground can be removed from V_1, and V_3, will remain at 0 Vdc. This is true since once the input of INV A is grounded, the output of INV B will go low and can then be used to hold the input of INV A low by using the feedback line. This is one stable state—$V_3 = 0$ Vdc.

Conversely, if V_1 is +5 Vdc, V_3 will also be +5 Vdc as seen in Fig. 8.2c. The feedback line can again be used to hold V_1 at +5 Vdc since V_3 is also at +5 Vdc. This is then the second stable state— $V_3 = +5$ Vdc.

NOR-Gate Latch

The basic flip-flop shown in Fig. 8.2a can be improved by replacing the inverters with either NAND or NOR gates. The additional inputs on these gates provide a convenient means for application of input signals to

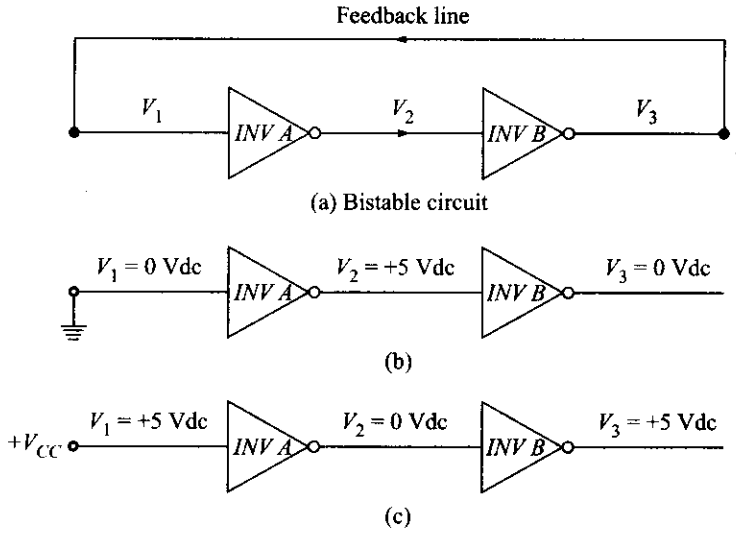


Fig. 8.2 Bistable circuit

switch the flip-flop from one stable state to the other. Two 2-input NOR gates are connected in Fig. 8.3a to form a flip-flop. Notice that if the two inputs labeled R and S are ignored, this circuit will function exactly as the one shown in Fig. 8.2a.

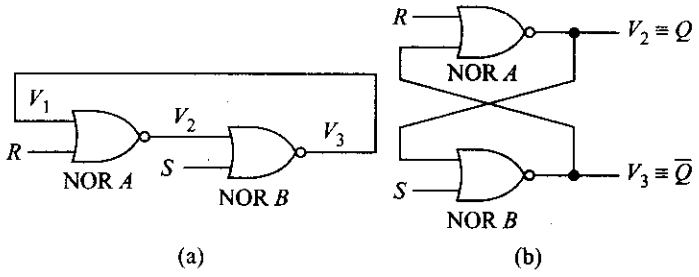


Fig. 8.3 NOR-gate flip-flop

This circuit is redrawn in a more conventional form in Fig. 8.3b. The flip-flop actually has two outputs, defined in more general terms as Q and $\overline{Q}$. It should be clear that regardless of the value of Q, its complement is $\overline{Q}$. There are two inputs to the flip-flop defined as R and S. The input/output possibilities for this RS flip-flop are summarized in the truth table in Fig. 8.4. To aid in understanding the operation of this circuit, recall that an $H = 1$ at any input of a NOR gate forces its output to an $L = 0$.

1. The first input condition in the truth table is $R = 0$ and $S = 0$. Since a 0 at the input of a NOR gate has no effect on its output, the flip-flop simply remains in its present state; that is, Q remains unchanged.
2. The second input condition $R = 0$ and $S = 1$ forces the output of NOR gate B low. Both inputs to NOR gate A are now low, and the NOR-gate output must be high. Thus a 1 at the S input is said to *SET* the flip-flop, and it switches to the stable state where $Q = 1$.

3. The third input condition is $R = 1$ and $S = 0$. This condition forces the output of NOR gate A low, and since both inputs to NOR gate B are now low, the output must be high. Thus a 1 at the R input is said to RESET the flip-flop, and it switches to the stable state where $Q = 0$ (or $\overline{Q} = 1$).
4. The last input condition in the table, $R = 1$ and $S = 1$, is forbidden, as it forces the outputs of both NOR gates to the low state. In other words, both $Q = 0$ and $\overline{Q} = 0$ at the same time! But this violates the basic definition of a flip-flop that requires Q to be the complement of $\overline{Q}$, and so it is generally agreed never to impose this input condition. Incidentally, if this condition is for some reason, imposed and the next input is $R = 0$, $S = 0$ then the resulting state Q depends on propagation delays of two NOR gates. If delay of gate A is less, i.e. it acts faster, then $Q = 1$ else it is 0. Such dependence makes the job of a design engineer difficult, as any replacement of a NOR gate will make Q unpredictable. That's why $R = 1$, $S = 1$ is forbidden and truth table entry is ?.

It is also important to remember that TTL gate inputs are quite noise-sensitive and therefore should never be left unconnected (floating). Each input must be connected either to the output of a prior circuit, or if unused, to GND or $+V_{CC}$.

R	S	Q	Action
0	0	Last state	No change
0	1	1	SET
1	0	0	RESET
1	1	?	Forbidden

Fig. 8.4 **Truth table for a NOR-gate *RS* flip-flop**

Example 8.1 Use the pinout diagram for a 54/7427 triple 3-input NOR gate and show how to connect a simple *RS* flip-flop.

Solution One possible arrangement is shown in Fig. 8.5. Notice that pins 3 and 4 are tied together, as are pins 10 and 11; thus no input leads are left unconnected and the two gates simply function as 2-input gates. The third NOR gate is not used. (It can be a spare or can be used elsewhere.)

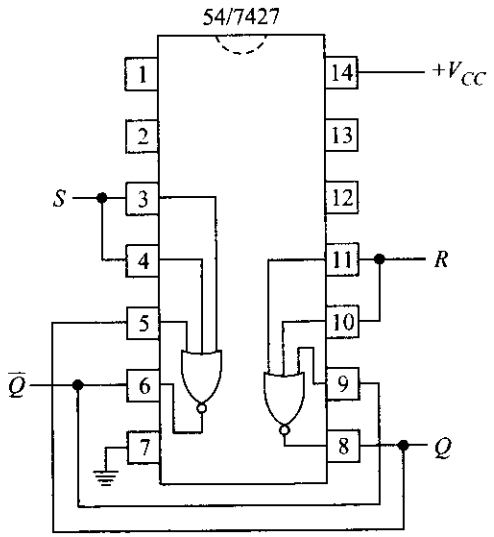
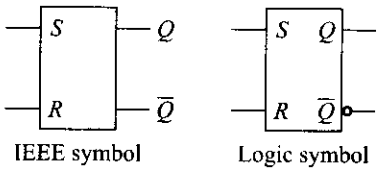


Fig. 8.5 **54/7427**

The standard logic symbols for an *RS* flip-flop are shown in Fig. 8.6 along with its truth table. The truth table is necessary since it describes exactly how the flip-flop functions.



(a)

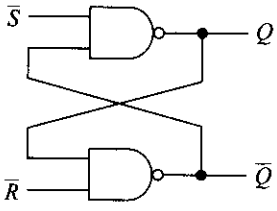
R	S	Q
0	0	Last state
0	1	1
1	0	0
1	1	? (Forbidden)

(b) Truth table

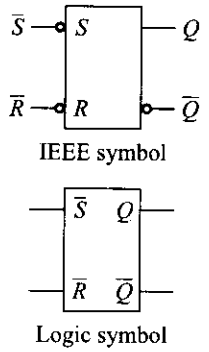
Fig. 8.6 ***RS* flip-flop**

NAND-Gate Latch

A slightly different latch can be constructed by using NAND gates as shown in Fig. 8.7. The truth table for this NAND-gate latch is different from that for the NOR-gate latch. We will call this latch an $\overline{R}\overline{S}$ flip-flop. To understand how this circuit functions, recall that *a low on any input to a NAND gate will force its output high*. Thus a low on the $\overline{S}$ input will set the latch ($Q = 1$ and $\overline{Q} = 0$). A low on the $\overline{R}$ input will reset it ($Q = 0$). If both $\overline{R}$ and $\overline{S}$ are high, the flip-flop will remain in its previous state. Setting both $\overline{R}$ and $\overline{S}$ low simultaneously is forbidden since this forces both Q and $\overline{Q}$ high.



(a) NAND gate latch



(b)

$\overline{R}$	$\overline{S}$	Q
1	1	Last state
1	0	1
0	1	0
0	0	? (Forbidden)

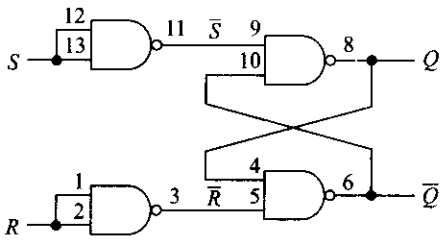
(c) Truth table

Fig. 8.7 ***$\overline{R}\overline{S}$* flip-flop**

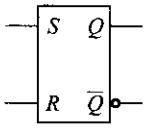
Example 8.2 Show how to convert the $\overline{R}\overline{S}$ flip-flop in Fig. 8.7 into an *RS* flip-flop.

Solution By placing an inverter at each input as shown in Fig. 8.8, the 2 inputs are now *R* and *S*, and the resulting circuit behaves exactly as the *RS* flip-flop in Fig. 8.6. A single 54/7400 (quad 2-input NAND gate) is used.

Simple latches as discussed in this section can be constructed from NAND or NOR gates or obtained as medium-scale integrated circuits (MSI). For instance, the 74LS279 is a *quad* $\overline{R}\overline{S}$ *latch*. The pinout and truth table for this circuit are given in Fig. 8.9. Study the truth table carefully, and you will see that the latches behave exactly like the $\overline{R}\overline{S}$ flip-flop discussed above.



(a) 54/7400

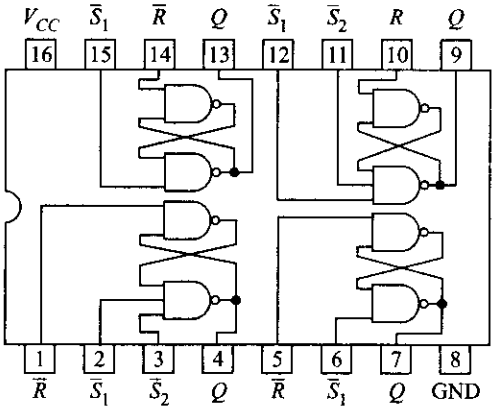


(b) Logic symbol

R	S	Q
0	0	Last state
0	1	1
1	0	0
1	1	? (Forbidden)

(c)

Fig. 8.8 An RS flip-flop (latch)



(a) Pinout 74S279A

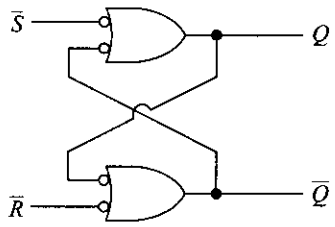
$\overline{S}_1$	$\overline{S}_2$	$\overline{R}$	Q
0	0	0	? Forbidden
0	X	1	1
X	0	1	1
1	1	0	0
1	1	1	? Forbidden

X = Don't care

(b) Truth table

Fig. 8.9 Quad SET-RESET latch

The NOR-gate flip-flop in Fig. 8.3 is seen to be an active-high circuit because an $H = 1$ at either the S or R input is required to change the output Q. On the other hand, the NAND-gate flip-flop in Fig. 8.7 can be considered an active-low circuit because an $L = 0$ at either input is required to change Q. The NAND gates in Fig. 8.7 can be changed to bubbled-input OR gates as shown in Fig. 8.10. This circuit is equivalent to the NAND-gate latch in Fig. 8.7 and functions in exactly the same way. However, the bubbled inputs more clearly express circuit operation.



$\overline{R}\,\overline{S}$ flip-flop

Fig. 8.10 Bubbled OR-gate equivalent of Fig. 8.7

SELF-TEST

1. What do the letters *R* and *S* stand for in the term "*RS* latch"?
2. A 74LS279 is a quad latch. What does quad mean?
3. Why is the NAND-gate latch considered active-low?

8.2 GATED FLIP-FLOPS

Two different methods for constructing an *RS* flip-flop were discussed in Sec. 8.1. The NOR-gate realization in Fig. 8.3b is an exact equivalent of the NAND-gate realization in Fig. 8.8a, and they both have the exact same symbol and truth table as given in Fig. 8.6. Both of these *RS* flip-flops, or latches, are said to be *transparent*; that is, any change in input information at R or S is transmitted immediately to the output at Q and $\overline{Q}$ according to the truth table.

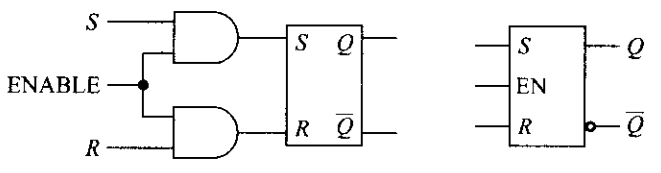
Clocked *RS* Flip-Flops

The addition of two AND gates at the R and S inputs as shown in Fig. 8.11 will result in a flip-flop that can be enabled or disabled. When the ENABLE input is low, the AND gate outputs must both be low and changes in neither R nor S will have any effect on the flip-flop output Q. The latch is said to be *disabled.*

When the ENABLE input is high, information at the R and S inputs will be transmitted directly to the outputs. The latch is said to be *enabled.* The output will change in response to input changes as long as the ENABLE is high. When the ENABLE input goes low, the output will retain the information that was present on the input when the high-to-low transition took place.

In this fashion, it is possible to *strobe* or *clock* the flip-flop in order to store information (set it or reset it) at any time, and then hold the stored information for any desired period of time. This flip-flop is called a *gated* or *clocked RS flip-flop.* The proper symbol and truth table are given in Fig. 8.11b. Notice that there are now three inputs—R, S, and the ENABLE or CLOCK input, labeled *EN.* Notice also that the truth-table output is not simply Q, but Q_{n+1}. This is because we must consider two different instants in time: the time before the ENABLE goes low Q_n and the time just after ENABLE goes low Q_{n+1}. When EN = 0, the flip-flop is disabled and R and S have no effect; thus the truth table entry for R and S is X (don't care).

Example 8.3 Explain the meaning of Q_n the truth table in Fig. 8.11b.



(a) Logic diagram

EN	S	R	Q_{n+1}
1	0	0	Q_n (no change)
1	0	1	0
1	1	0	1
1	1	1	? (Illegal)
0	X	X	Q_n (no change)

(b) IEEE symbol and truth table

Fig. 8.11 **Clocked *RS* flip-flop**

Solution For the flip-flop to operate properly, there must be a PT on the EN input. While EN is high, the information on R and S causes the latch to set or reset. Then when EN transitions back to low, this information is retained in the latch. When this NT occurred, both R and S inputs were low (0), and thus there was no change of state. In other words, the value of Q at time $n + 1$ is the same as it was at time n. Remember that time n occurs just before the NT on EN, and time $n + 1$, occurs just after the transition.

The logic diagrams shown in Fig. 8.12a and b illustrate two different methods for realizing a clock *RS* flip-flop. Both realizations are widely used in medium- and large-scale integrated circuits, and you will find them easy to recognize. You might like to examine the logic diagrams for a 54LS109 or a 54LS74, for instance.

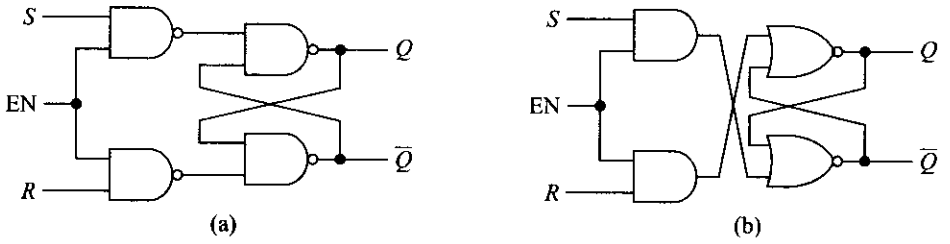


Fig. 8.12 **Two different realizations for a clocked *RS* flip-flop**

Example 8.4 Figure 8.13 shows the input waveforms *R*, *S*, and EN applied to a clocked *RS* flip-flop. Explain the output waveform *Q*.

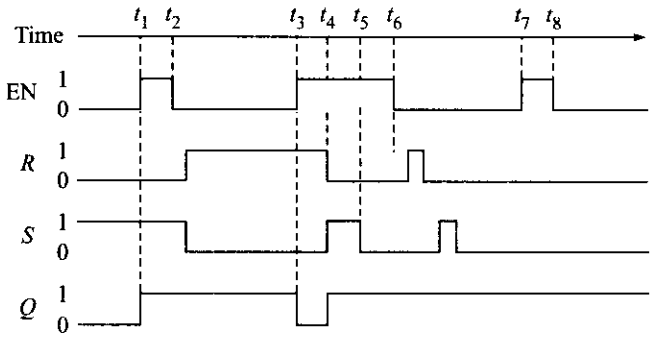


Fig. 8.13

Solution Between t_2 and t_3 both *R* and *S* change states, but since EN is low, the flip-flop is still disabled and *Q* remains at 1.

Between t_3 and t_6, the flip-flop will respond to any change in *R* and *S* since EN is high. Thus at t_3 *Q* goes low, and at t_4 it goes back high. No change occurs at t_5. At t_6 the value $Q = 1$ is latched and no changes in *Q* occur between t_6 and t_7 even though both *R* and *S* change.

Between t_7, and t_8 no change in *Q* occurs since both *R* and *S* are low. Initially, the flip-flop is reset ($Q = 0$). At time t_1 EN goes high; the flip-flop is now enabled, and it is immediately set ($Q = 1$) since $R = 0$ and $S = 1$. At time t_2 EN goes low and the flip-flop is disabled and latches in the stable state $Q = 1$.

Clocked D Flip-Flops

The *RS* flip-flop has two data inputs, *R* and *S*. To store a high bit, you need a high *S*; to store a low bit, you need a high *R*. Generation of two signals to drive a flip-flop is a disadvantage in many applications. Furthermore, the forbidden condition of both *R* and *S* high may occur inadvertently. This has led to the *D* flip-flop, a circuit that needs only a single data input.

Figure 8.14 shows a simple way to build a *D* (Data) flip-flop. This flip-flop is disabled when EN is low, but is *transparent* when EN is high. The action of the circuit is straightforward, as follows. When EN is low, both AND gates are disabled; therefore, *D* can change value without affecting the value of *Q*. On the other hand,

when EN is high, both AND gates are enabled. In this case, Q is forced to equal the value of D. When EN again goes low, Q retains or stores the last value of D.

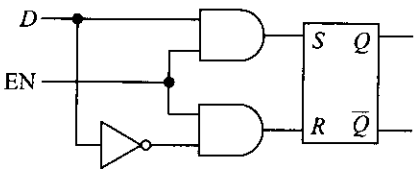
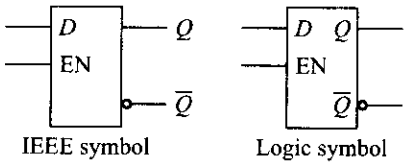


Fig. 8.14 **A *D* Flip-flop**

There are many ways to design D flip-flops. In general, a D flip-flop is a bistable circuit whose D input is transferred to the output when EN is high. Figure 8.15 shows the logic symbols used for any type of D flip-flop.

In this section we're talking about the kind of D flip-flop in which Q can follow the value of D while EN is high. In other words, if the data bit changes while EN is high, the last value of D before EN return low is the value of D that is stored. This kind of D flip-flop is often called a D latch.

Figure 8.15b shows the truth table for a D latch. While (EN) is low, D is a don't care (X); Q will remain latched in its last state. When EN is high, Q takes on the last value of D. If D is changing while EN is high, it is the last value of D that is stored.



(a) D flip-flop logic symbol

EN	D	Q_{n+1}
0	X	Q_n (last state)
1	0	0
1	1	1

(b) Truth table

Fig. 8.15 ***D* Flip-flop logic symbol**

The idea of data storage is illustrated in Fig. 8.16. Four D latches are driven by the same clock signal. When the clock goes high, input data is loaded into the flip-flops and appears at the output. Then when the clock goes low, the output retains the data. For instance, suppose that the data input is

$$D_3D_2D_1D_0 = 0111$$

When the clock goes high, this word is loaded into the D latches, resulting in an output of

$$Q_3Q_2Q_1Q_0 = 0111$$

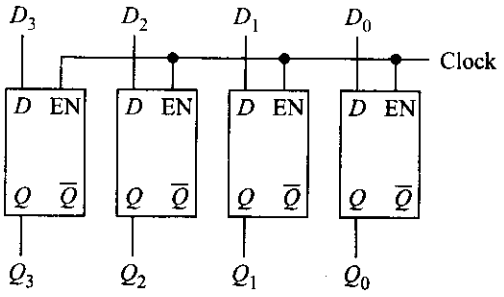


Fig. 8.16 **Storing a 4-bit word**

After the clock goes low, the output data is retained, or stored. As long as the clock is low, the D values can change without affecting the Q values.

The 7475 in Fig. 8.17 is a TTL MSI circuit that contains four D latches; it's called a *quad bistable latch*. The 7475 is ideal for handling 4-bit nibbles of data. If more than one 7475 is used, words of any length can be stored.

SELF-TEST

4. What does an entry X mean in a flip-flop truth table?
5. What could you do to disable the flip-flop in Fig. 8.11?
6. Which flip-flop is easier to use, the RS of the D, as a clocked or gated latch to store data?

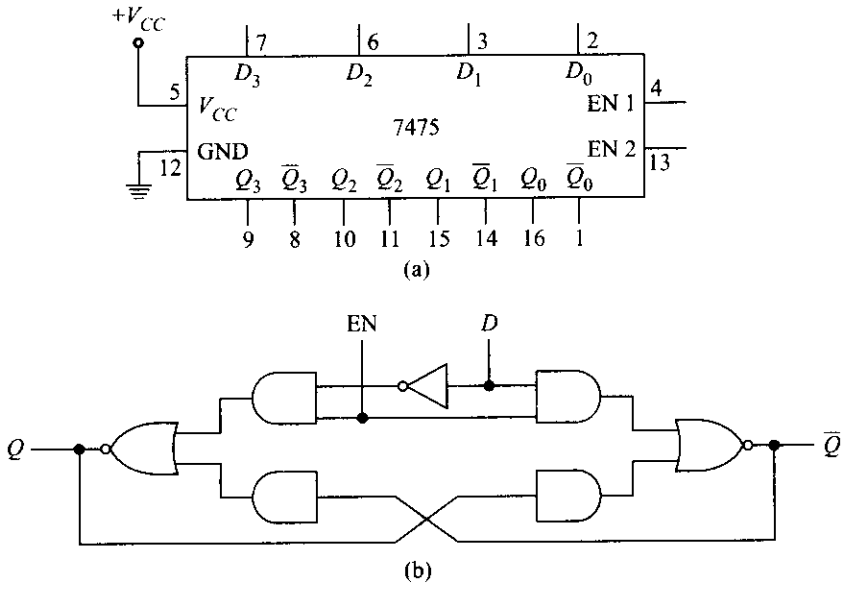


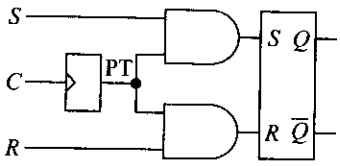
Fig. 8.17 **4-bit bistable latch: (a) Pinout, (b) Logic diagram (each latch)**

8.3 EDGE-TRIGGERED *RS* FLIP-FLOPS

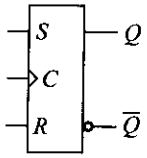
The simple latch-type flip-flops presented in Sec. 8.1 are completely *transparent*; that is, the output Q immediately follows *any* change of state at the input (R, S, or D). The gated or clocked RS and D flip-flops in Sec. 8.2 might be considered *semitransparent*. That is, the output Q will change state immediately provided that the EN input is high. If any of these flip-flops are used in a synchronous system, care must be taken to ensure that all flip-flop inputs change state in synchronism with the clock. One way of resolving the problem for gated flip-flops is to allow changes in R, S, and D input levels only when EN is low (or require fixed levels at R, S, and D any time EN is high). At the very least, these are highly inconvenient restrictions, and at the worst they may in fact be impossible to realize. From the previous chapter, we know that virtually all digital systems operate in a synchronous mode. Thus the *edge-triggered flip-flop* was developed to overcome these rather severe restrictions.

Positive-Edge-Triggered *RS* Flip-Flops

In Fig. 8.18a, the clock (C) is applied to a positive pulse-forming circuit (discussed in Sec. 7.1). The PTs developed are then applied to a gated RS flip-flop. The result is a positive-edge-triggered RS flip-flop, with the IEEE symbol given in Fig. 8.18b. The small triangle inside the symbol (dynamic input indicator) indicates that Q can change state only with PTs of the clock (C). Each PT of the clock in Fig. 8.18c produces a very narrow PT that is applied to the AND gates. The AND gates are active only while the PT is high (perhaps 25 ns), and thus Q can change state only during this short time period. In this manner Q changes state in synchronism with the PTs of the clock.



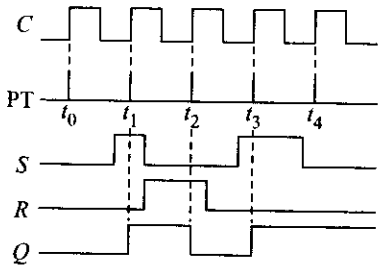
(a) Logic diagram



(b) IEEE symbol

C	S	R	Q_{n+1}	Action
↑	0	0	Q_n	No change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	?	Illegal

(c) Truth table



(d) Positive-edge-triggered *RS* flip-flop

Fig. 8.18 Positive-edge-triggered *RS* flip-flop

This flip-flop is easy to use in any synchronous system! Another way of expressing its behavior is to say the flip-flop is transparent only during PTs; it is not transparent for the remainder of the time. In other words, *S* and *R* inputs affect *Q* only while the positive pulse is high, and they need to be static only during this very short time.

The truth table for the edge-triggered *RS* flip-flop is given in Fig. 8.18c. The small vertical arrows under *C* (clock) mean that changes of state (*Q*) occur according to the *R* and *S* levels, but only during PTs of the clock. Look at the waveforms in Fig. 8.18d. Note that when *Q* changes state, it does so in exact synchronism with PTs of the clock *C*.

Example 8.5 Use the positive-edge-triggered *RS* flip-flop truth table to explain *Q* changes of state with time in Fig. 8.18d.

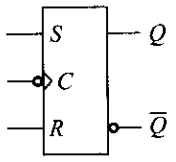
Solution Here's what happens at each point in time:

Time t_0: $S = 0$, $R = 0$, no change in Q (Q remains 0)
Time t_1: $S = 1$, $R = 0$, Q changes from 0 to 1
Time t_2: $S = 0$, $R = 1$, Q resets to 0
Time t_3: $S = 1$, $R = 0$, Q sets to 1
Time t_4: $S = 0$, $R = 0$, no change in Q (Q remains 1)

Notice that either *R* or *S*, or both, are allowed to change state at any time, whether *C* is high or low. The only time both *R* and *S* must be stable (unchanging) is during the short PTs of the clock.

Negative-Edge-Triggered *RS* Flip-Flops

The symbol in Fig. 8.19a is for a negative-edge-triggered *RS* flip-flop. The truth table in Fig. 8.19b shows that *Q* changes state according to the *R* and *S* inputs, but only during NTs of the clock. On the IEEE symbol, the small bubble on the clock input (*C*) means active-low. This bubble, along with the dynamic input indicator,



(a) IEEE symbol

C	S	R	Q_{n+1}	Action
↓	0	0	Q_n	No change
↓	0	1	0	RESET
↓	1	0	1	SET
↓	1	1	?	Illegal

(b) Truth table

Fig. 8.19 **Negative-edge-triggered *RS* flip-flop**

means negative-edge triggering. This flip-flop behaves exactly like the positive-edge-triggered *RS* flip-flop, except that changes in output Q are synchronized with NTs of the clock (C).

Example 8.6 Use the negative-edge-triggered *RS* flip-flop truth table to explain Q changes of state with time in Fig. 8.20.

Solution Here's what happens at each point in time:

Time t_0: $S = 0$, $R = 0$, no change in Q(Q remains 0)
Time t_1: $S = 1$, $R = 0$, Q changes from 0 to 1
Time t_2: $S = 0$, $R = 1$, Q resets to 0
Time t_3: $S = 1$, $R = 0$, Q sets to 1
Time t_4: $S = 0$, $R = 0$, no change in Q(Q remains 1)

Notice that either R or S, or both, are allowed to change state at any time, whether C is high or low. The only time both R and S must be stable (unchanging) is during the short NTs of the clock.

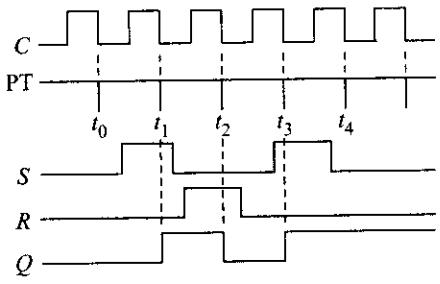


Fig. 8.20

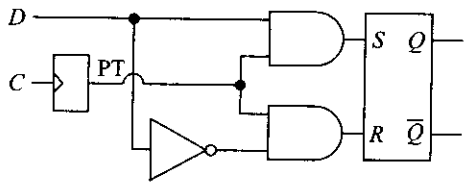
SELF-TEST

7. What does it mean to say that a flip-flop is transparent?
8. What is positive-edge triggering?
9. How does an *RS* latch differ from an edge-triggered *RS* flip-flop?

8.4 EDGE-TRIGGERED *D* FLIP-FLOPS

Although the D latch is used for temporary storage in electronic instruments, an even more popular kind of D flip-flop is used in digital computers and systems. This kind of flip-flop samples the data bit at a unique point in time.

Figure 8.21 shows a positive pulse-forming circuit at the input of a D latch. The narrow positive pulse (PT) enables the AND gates for an instant. The effect is to activate the AND gates during the PT of C, which is equivalent to sampling the value of D for an instant. At this unique point in time, D and its complement hit the flip-flop inputs, forcing Q to set or reset (unless Q already equals D). Again, this operation is called *edge triggering* because the flip-flop responds only when the clock is in transition between its two voltage states. The triggering in Fig. 8.21 occurs on the positive-going edge of the clock; this is why it's referred to as *positive-edge triggering*.



(a) Circuit diagram

C	D	Q_{n+1}
0	X	Q_n (last state)
↑	0	0
↑	1	1

(c) Truth table

Fig. 8.21 Positive-edge-triggered *D* flip-flop

The truth table in Fig. 8.21b summarizes the action of a positive-edge-triggered D flip-flop. When the clock is low, D is a don't care and Q is latched in its last state. On the leading edge of the clock (PT), designated by the up arrow, the data bit is loaded into the flip-flop and Q takes on the value of D.

When power is first applied, flip-flops come up in random states. To get some computers started, an operator has to push a RESET button. This sends a CLEAR or RESET signal to all flip-flops. Also, it's necessary in some digital systems to preset (synonymous with set) certain flip-flops.

Figure 8.22 shows how to include both functions in a D flip-flop. The edge triggering is the same as previously described. Depressing the RESET button will set Q to 1 with the first PT of the clock. Q will remain high as long as the button is held closed. The first PT of the clock after releasing the button will set Q according to the D input. Furthermore, the OR gates allow us to slip in a high PRESET or a high CLEAR when desired. A high PRESET forces Q to equal 1; a high CLEAR resets Q to 0.

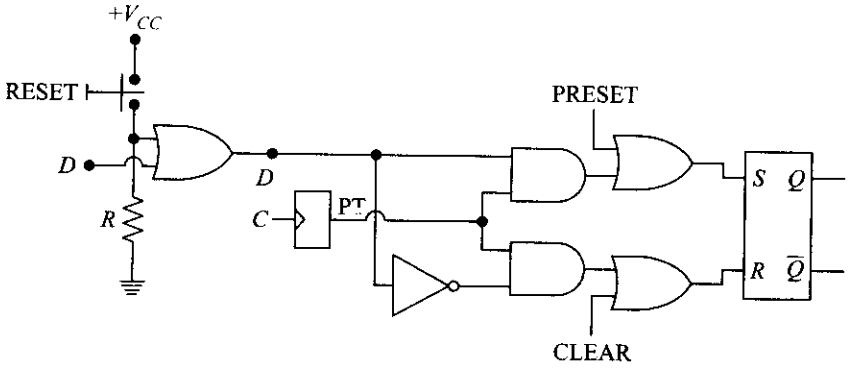


Fig. 8.22 PRESET and CLEAR functions

The PRESET and CLEAR are called *asynchronous inputs* because they activate the flip-flop independently of the clock. On the other hand, the D input is a synchronous input because it has an effect only with PTs of the clock.

Figure 8.23a is the IEEE symbol for a positive-edge-triggered D flip-flop. The clock input has a small triangle to serve as a reminder of edge triggering. When you see this symbol, remember what it means; the D input is sampled and stored on PTs of the clock.

Sometimes, triggering on NTs of the clock is better suited to the application. In this case, an internal inverter can complement the clock pulse before it reaches the AND gates. Figure 8.23b is the symbol for a negative-edge-triggered D flip-flop. The bubble and triangle symbolize the negative-edge triggering.

Figure 8.23c is another commercially available *D* flip-flop (the 54/74175 or 54/74LS175). Besides having positive-edge triggering, it has an inverted CLEAR input This means that a low CLR resets it. The 54/74175 has four of these *D* flip-flops in a single 16-pin dual in-line package (DIP), and it's referred to as a *quad D-type flip-flop with clear.*

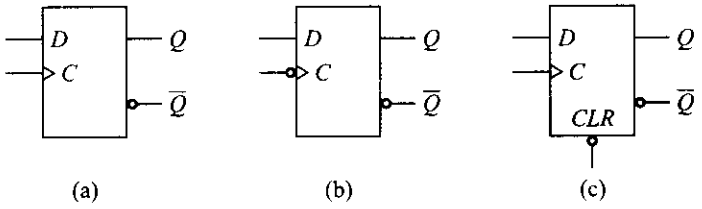


Fig. 8.23 ***D*flip-flopsymbols: (a) Positive-edge-triggered, (b) Negative-edge-triggered, (c) Positive-edge-triggered with active low clear**

SELF-TEST

10. The *C* input to the *D* flip-flop in Fig. 8.21 is held low. What effect does the *D* input have?
11. To preset the flip-flop in Fig. 8.22, what level is required at the preset input. What is the resulting state of *Q*?

8.5 EDGE-TRIGGERED *JK* FLIP-FLOPS

Setting $R = S = 1$ with an edge-triggered *RS* flip-flop forces both Q and $\overline{Q}$ to the same logic level. This is an *illegal* condition, and it is not possible to predict the final state of Q. The *JK* flip-flop accounts for this illegal input, and is therefore a more versatile circuit. Among other things, flip-flops can be used to build counters. Counters can be used to count the number of PTs or NTs of a clock. For purposes of counting, the *JK* flip-flop is the ideal element to use. There are many commercially available edge-triggered *JK* flip-flops. Let's see how they function.

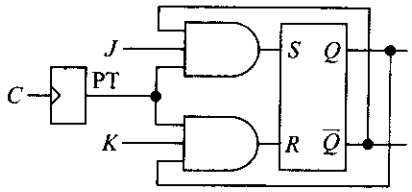
Positive-Edge-Triggered *JK* Flip-Flops

In Fig. 8.24, the pulse-forming box changes the clock into a series of positive pulses, and thus this circuit will be sensitive to PTs of the clock. The basic circuit is identical to the previous positive-edge-triggered *RS* flip-flop, with two important additions:

1. The Q output is connected back to the input of the lower AND gate.
2. The $\overline{Q}$ output is connected back to the input of the upper AND gate.

This cross-coupling from outputs to inputs changes the *RS* flip-flop into a *JK* flip-flop. The previous *S* input is now labeled *J*, and the previous *R* input is labeled *K*. Here's how it works:

1. When *J* and *K* are both low, both AND gates are disabled. Therefore, clock pulses have no effect. This first possibility is the initial entry in the truth table. As shown, when *J* and *K* are both 0s, Q retains its last value.
2. When *J* is low and *K* is high, the upper gate is disabled, so there's no way to set the flip-flop. The only possibility is reset. When Q is high, the lower gate passes a RESET pulse as soon as the next positive



(a) One way to implement a *JK* flip-flop

C	J	K	Q_{n+1}	Action
↑	0	0	Q_n (last state)	No change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	$\overline{Q}_n$ (toggle)	Toggle

(b) Truth table

Fig. 8.24 A positive-edge-triggered *JK* flip-flop

clock edge arrives. This forces Q to become low (the second entry in the truth table). Therefore, $J = 0$ and $K = 1$ means that the next PT of the clock resets the flip-flop (unless Q is already reset).

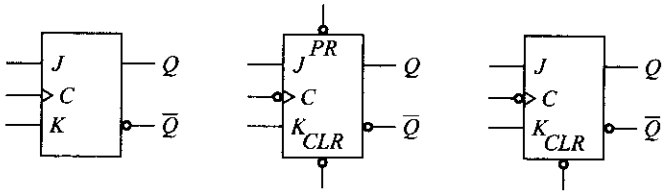
3. When J is high and K is low, the lower gate is disabled, so it's impossible to reset the flip-flop. But you can set the flip flop as follows. When Q is low, $\overline{Q}$ is high; therefore, the upper gate passes a SET pulse on the next positive clock edge. This drives Q into the high state (the third entry in the truth table). As you can see, $J = 1$ and $K = 0$ means that the next PT of the clock sets the flip-flop (unless Q is already high).
4. When J and K are both high (notice that this is the forbidden state with an *RS* flip-flop), it's possible to set or reset the flip-flop. If Q is high, the lower gate passes a RESET pulse on the next PT. On the other hand, when Q is low, the upper gate passes a SET pulse on the next PT. Either way, Q changes to the complement of the last state (see the truth table). Therefore, $J = 1$ and $K = 1$ mean the flip-flop will *toggle* (switch to the opposite state) on the next positive clock edge.

Propagation delay prevents the *JK* flip-flop from racing (toggling more than once during a positive clock edge). Here's why. In Fig. 8.24, the outputs change after the PT of the clock. By then, the new Q and $\overline{Q}$ values are too late to coincide with the PTs driving the AND gates. For instance, if t_p = 20 ns, the outputs change approximately 20 ns after the leading edge of the clock. If the PTs are narrower than 20 ns, the returning Q and $\overline{Q}$ arrive too late to cause false triggering.

Figure 8.25a shows a symbol for a *JK* flip-flop of any design. When you see this on a schematic diagram, remember that on the next PT of the clock:

1. J and K low: no change of Q.
2. J low and K high: Q is reset low.
3. J high and K low: Q is set high.
4. J and K both high: Q toggles to opposite state.

You can include OR gates in the design to accommodate PRESET and CLEAR as was done earlier. Figure 8.25b gives the symbol for a *JK* flip-flop with *PR* and *CLR*. Notice that it is negative-edge-triggered and requires a low *PR* to set it or a low *CLR* to reset it.



(a) Basic symbol (b) 74LS76A (c) 74LS73A

Fig. 8.25 *JK* flip-flop symbols

Figure 8.25c is another commercially available *JK* flip-flop. It is negative-edge-triggered and requires a low *CLR* to reset it. The output *Q* reacts immediately to a *PR* or *CLR* signal. Both *PR* and *CLR* are asynchronous, and they override all other input signals.

Example 8.7 Toggle flip-flop, popularly known as *T* flip-flop has following input-output relation. When input $T = 0$, the output *Q* does not change its state. For $T = 1$, the output *Q* toggles its value. Derive *T* flip-flop from *JK* flip-flop.

Solution From Fig. 8.24b we find for input $J = K = 0$, the output $Q_{n+1} = Q_n$, i.e. output does not change its state. And for $J = K = 1$, the output $Q_{n+1} = Q_n'$, i.e. output toggles. Thus, if we tie *J* and *K* inputs of *JK* flip-flop together and make a common input $T = J = K$, the resulting circuit will behave as *T* flip-flop.

SELF-TEST

12. What is the primary difference between a *JK* and an *RS* flip-flop?
13. How could you change an edge-triggered *RS* flip-flop into an edge-triggered *JK* flip-flop?

8.6 FLIP-FLOP TIMING

Diodes and transistors cannot switch states immediately. It always takes a small amount of time to turn a diode on or off. Likewise, it takes time for a transistor to switch from saturation to cutoff, and vice versa. For bipolar diodes and transistors, the switching time is in the nanosecond region.

Switching time is the main cause of propagation delay, designated t_p. This represents the amount of time it takes for the output of a gate or flip-flop to change states after the input changes. For instance, if the data sheet of an edge-triggered *D* flip-flop lists $t_p = 10$ ns, it takes about 10 ns for *Q* to change states after *D* has been sampled by the clock edge. This propagation delay time has been used to construct the "pulse-forming circuit" used with edge-triggered flip-flops. When flip-flops are used to construct counters, the propagation delay is often small enough to be ignored.

Stray capacitance at the *D* input (plus other factors) makes it necessary for data bit *D* to be at the input before the clock edge arrives. The setup time t_{setup} is the minimum amount of time that the data bit must be present before the clock edge hits. For instance, if a *D* flip-flop has a setup time of 15 ns, the data bit to be stored must be at the *D* input at least 15 ns before the clock edge arrives; otherwise, the manufacturer does not guarantee correct sampling and storing.

Furthermore, data bit *D* has to be held long enough for the internal transistors to switch states. Only after the transition is assured can we allow data bit *D* to change. Hold time t_{hold} is the minimum amount of time that data bit *D* must be present after the PT of the clock. For example, if $t_{setup} = 15$ ns and $t_{hold} = 5$ ns, the data bit has to be at the *D* input at least 15 ns before the clock edge arrives and held at least 5 ns after the clock PT.

Example 8.8 Typical waveforms for setting a 1 in a positive-edge-triggered flip-flop are shown in Fig. 8.26. Discuss the timing.

Solution The lower line in Fig. 8.26 is the time line with critical times marked on it. Prior to t_1, the data can be a 1 or a 0, or can be changing. This is shown by drawing lines for both high and low levels on *D*. From time t_1 to t_2, the

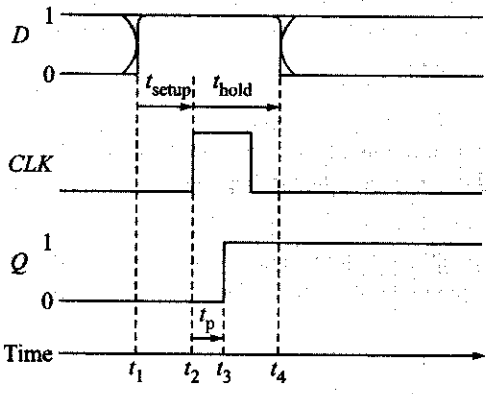


Fig. 8.26

data line D must be held steady (in this case a 1). This is the setup time t_{setup}. Data is shifted into the flip-flop at time t_2 but does not appear at Q until time t_3. The time from t_2 to t_3 is the propagation time t_p. In order to guarantee proper operation, the data line must be held steady from time t_2 until t_4; this is the hold time t_{hold}. After t_4, D is free to change states—shown by the double lines.

8.7 EDGE TRIGGERING THROUGH INPUT LOCK OUT

We have seen how edge triggering of flip-flops can be achieved by pulse forming circuit (Section 7.1). This requires application of a very narrow pulse which is generated using differential propagation delays of two signal flow paths while the flip-flops themselves are level triggered. An alternate way of achieving edge triggering is to implement a kind of lock out of the input so that it is not able to enforce a change at output which itself is level triggered. This is to say that the effect of change in input is allowed only at the edge and not after the edge. Let us see how this is possible by implementing a positive edge triggered D type flip-flop. This requires three NAND latches as shown in Fig. 8.27 with one NAND gate (number 3) having three inputs and the rest are all two input NAND gates. Note that for a NAND gate output to be 0, all the inputs must be at 1, else the output is 1. The output latch behaves like an SR flip flop where no change in output occurs if $S = 1, R = 1$.

Now, if the clock input is held at 0 then irrespective of what is present at D input, the NAND logic makes both $S = 1, R = 1$ and thus there could be no change in the output. If Clock = 1 then SR can always change if other inputs of NAND gates 2 and 3 change and thus the output is essentially level triggered. We will now explain how input lock out makes the circuit as a whole a positive edge triggered circuit.

Consider the case when Clock = 0 and $D = 0$ (Fig. 8.27a). Since, for a NAND gate, 0 is the forcing input, the intermediate outputs are $S = 1, R = 1$ and $A = 1$ which make $B = 0$. Now, clock makes a transition from 0 $\rightarrow$ 1. $D = 0$ forces $A = 1$ and $B = 0$ keeps $R = 1$. Thus, after this transition, $S = 0, R = 1, A = 1$ and $B = 0$. This makes $Q = 0$ irrespective of the previous state and one can see that the value at D, i.e. 0 is transferred to Q after the clock trigger. Next, we see if at Clock = 1, D is changed, then whether Q is changed. This is shown in Fig. 8.27b as a follow-up of Fig. 8.27a. Before D makes a transition Clock = 1, $D = 0$ and intermediate outputs $S = 0, R = 1, A = 1, B = 0$ and $Q = 0$. When D goes to 1, 4th NAND gate is only directly affected as D is not connected elsewhere. However, the output A of this gate does not change as it is kept held at 1 by the

other input coming from $S = 0$. Thus, $S = 0$, $R = 1$, $A = 1$, $B = 0$ and $Q = 0$. This is the lock out of input we were referring to. Note that clock going from 1 to 0 does not change Q as that transition makes $S = 1$, $R = 1$.

Next, consider the case when Clock = 0 and $D = 1$. This is shown in Fig. 8.27c. $S = 1$, $D = 1$ make $A = 0$ which in turn makes $B = 1$. Now, clock makes a transition from $0 \rightarrow 1$. $A = 0$ maintains $S = 1$. Both the inputs of 2nd NAND gate being 1, $R = 0$. $S = 1$, $R = 0$ make $Q = 1$ irrespective of previous state and thus after positive clock trigger, the logic value of D arrives at Q for $D = 1$ case, too. With Clock = 1, if input D changes from 1 to 0, will the output Q change? This 4th possibility is shown in Fig. 8.27d. $D = 0$ makes $A = 1$ but $R = 0$ maintains $B = 1$ and $S = 1$. Thus, after the transition, SR remains at where it was and input D remaines locked out, i.e. unable to effect any change in the output at Clock = 1.

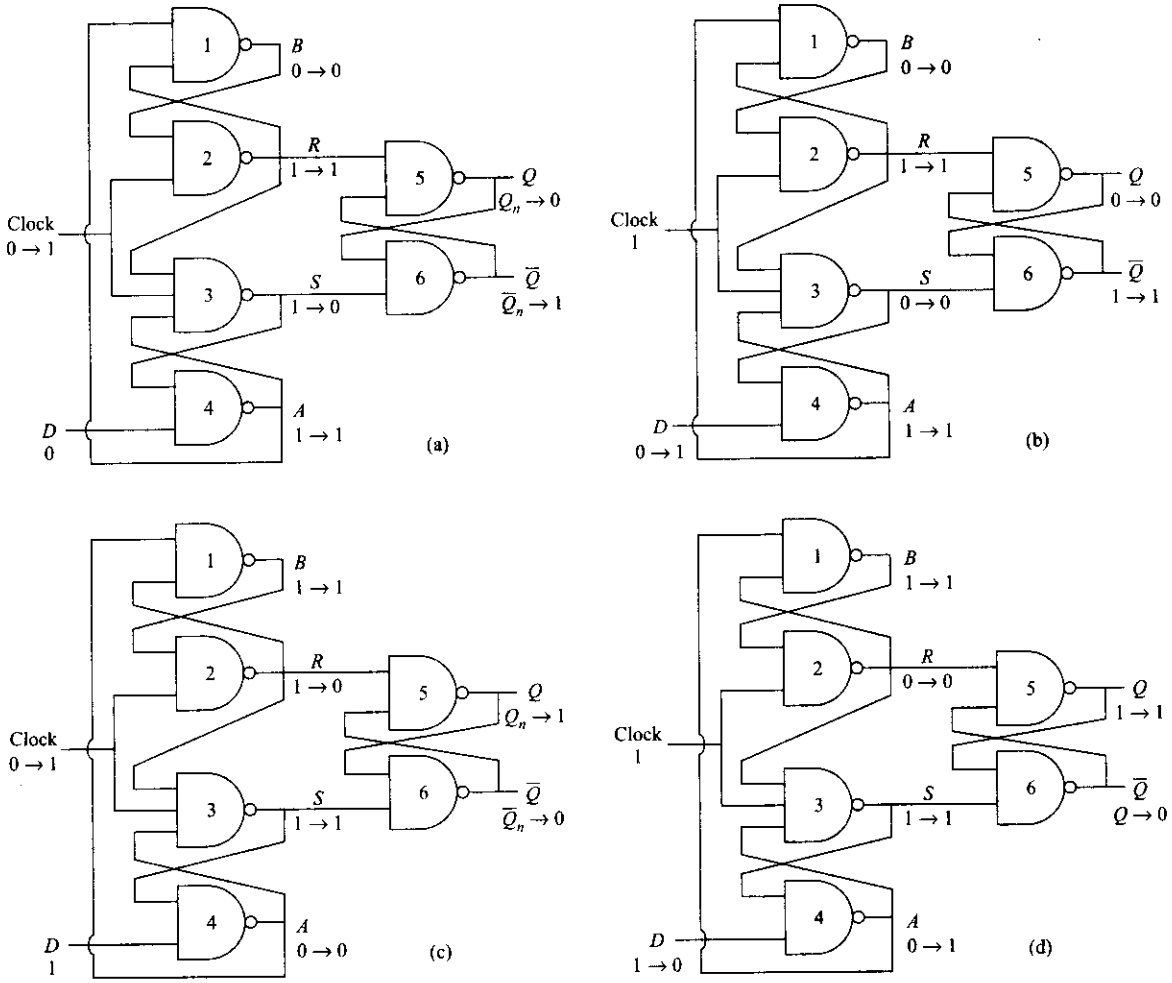


Fig. 8.27 **Positive edge triggering of *D* type flip-flop through input lock out**

8.8 *JK* MASTER-SLAVE FLIP-FLOPS

Figure 8.28 shows one way to build a *JK* master-slave flip-flop. Here's how it works.

1. To begin with, the master is positive-level-triggered and the slave is negative-level-triggered. Therefore, the master responds to its J and K inputs before the slave. If $J = 1$ and $K = 0$, the master sets on the positive clock transition. The high Q output of the master drives the J input of the slave, so on the negative clock transition, the slave sets, copying the action of the master.
2. If $J = 0$ and $K = 1$, the master resets on the PT of the clock. The high $\overline{Q}$ output of the master goes to the K input of the slave. Therefore, the NT of the clock forces the slave to reset. Again, the slave has copied the master.
3. If the master's J and K inputs are both high, it toggles on the PT of the clock and the slave then toggles on the clock NT. Regardless of what the master does, therefore, the slave copies it: if the master sets, the slave sets; if the master resets, the slave resets.
4. If $J = K = 0$, the flip-flop is disabled and Q remains unchanged.

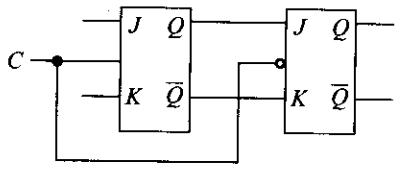
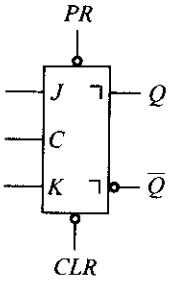


Fig. 8.28 Master-slave flip-flop

The symbol for a 7476 master-slave flip-flop is shown in Fig. 8.29. Either it can be preset to $Q = H$ by taking PR low, or it can be reset to $Q = L$ by taking CLR low. These two inputs take precedence over all other signals!

There is something different however. First of all, notice that the clock (C) is not edge-triggered. The master does in fact change state when C goes high. However, while the clock is high, any change in J or K will immediately affect the master flip-flop. In other words, the master is transparent while the clock is high, and thus J and K must be static during this time. The truth table in Fig. 8.29b reveals this action by means of the pulse symbol (⎍⎍).



(a) Symbol

C	J	K	Q_{n+1}	Action
⎍⎍	L	L	Q_n	No change
⎍⎍	L	H	L	RESET
⎍⎍	H	L	H	SET
⎍⎍	H	H	$\overline{Q}_n$	Toggle

(b) Truth table

Fig. 8.29 7476 JK master flip-flop.

Second, the symbol ⌝ appearing next to the Q and the $\overline{Q}$ outputs is the IEEE designation for a *postponed output*. In this case, it means Q does not change state until the clock makes an NT. In other words, the contents of the master are shifting into the slave on the clock NT, and at this time Q changes state.

To summarize: The master is set according to J and K while the clock is high; the contents of the master are then shifted into the slave (Q changes state) when the clock goes low. This particular flip-flop might be referred to as *pulse-triggered*, to distinguish it from the edge-triggered flip-flops previously discussed.

There are numerous pulse-triggered master-slave flip-flops in use today. However, because edge-triggered flip-flops have overcome the restriction of holding J and K static when the clock is high, most new designs incorporate edge-triggered devices. Some of the more popular pulse-triggered flip-flops you might encounter include the 7473, 7476, and 7478. Their more modern, edge-triggered counterparts include the 74LS73A, the 74LS76A, and the 74LS78A.

Example 8.9 The *JK* master-slave flip-flop in Fig. 8.29 has its J and K inputs tied to $+V_{CC}$ and a series of pulses (actually a square wave) are applied to its C input. Describe the waveform at Q.

Solution Since $J = K = 1$, the flip-flop simply toggles each time the clock goes low. The wave-form at Q has a period twice that of the C waveform. In other words, the frequency of Q is only one-half that of C. This circuit acts as a frequency divider—the output frequency is equal to the input frequency divided by 2. Note that Q changes state on NTs of the clock. The waveforms are given in Fig. 8.30.

Fig. 8.30

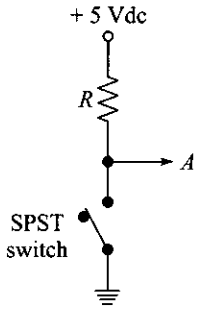
SELF-TEST

14. What is the main difference between an edge-triggered and a pulse-triggered *JK* flip-flop?
15. Explain the operation of the master-slave flip-flop in Fig. 8.29.

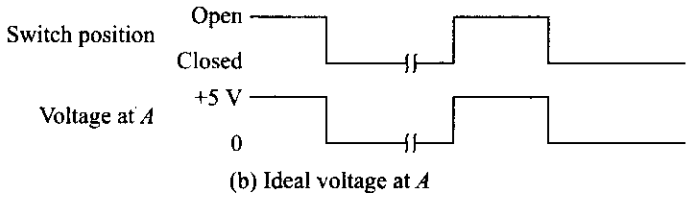
8.9 SWITCH CONTACT BOUNCE CIRCUITS

In nearly every digital system there will be occasion to use mechanical contacts for the purpose of conveying an electrical signal; examples of this are the switches used on the keyboard of a computer system. In each case, the intent is to apply a high logic level (usually +5 Vdc) or a low logic level (0 Vdc). The single-pole–single-throw (SPST) switch shown in Fig. 8.31a is one such example. When the switch is open, the voltage at point A is +5 Vdc; when the switch is closed, the voltage at paint A is 0 Vdc. Ideally, the voltage waveform at A should appear as shown in Fig. 8.31b as the switch is moved from open to closed, or vice versa.

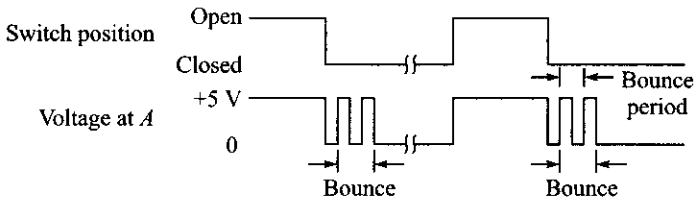
In actuality, the waveform at point A will appear more or less as shown in Fig. 8.31c, as the result of a phenomenon known as *contact bounce*. Any mechanical switching device consists of a moving contact arm restrained by some sort of a spring system. As a result, when the arm is moved from one stable position to the other, the arm bounces, much as a hard ball bounces when dropped on a hard surface. The number of bounces that occur and the period of the bounce differ for each switching device. Notice carefully that in this particular instance, even though actual physical contact bounce occurs each time the switch is opened or closed, contact bounce appears in the voltage level at point A only when the switch is closed.



(a)



(b) Ideal voltage at A



(c) Voltage at A showing contact bounce

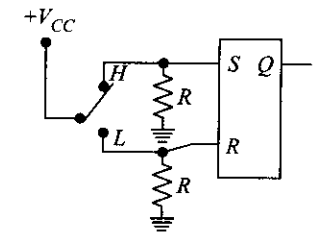
Fig. 8.31

If the voltage at point A is applied to the input of a TTL circuit, the circuit will respond properly when the switch is opened, since no contact bounce occurs. However, when the switch is closed, the circuit will respond as if multiple signals were applied, rather than the single-switch closure intended—the undesired result of mechanical contact bounce. There is a need here for some sort of electronic circuit to eliminate the contact bounce problem.

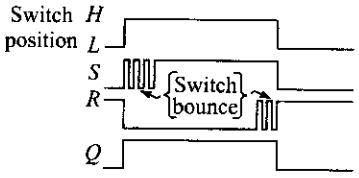
A Simple *RS* Latch Debounce Circuit

The *RS* latch in Fig. 8.32 will remove any contact bounce due to the switch. The output (Q) is used to generate the desired switch signal.

When the switch is moved to position H, $R = 0$ and $S = 1$. Bouncing occurs at the S input due to the switch. The flip-flop "sees" this as a series of high and low inputs, settling with a high level. The flip-flop will immediately be set with $Q = 1$ at the first high level on S. When the switch bounces, losing contact, the input signals are $R = S = 0$, therefore the flip-flop remains set ($Q = 1$). When the switch regains contact, $R = 0$ and $S = 1$; this causes an attempt to again set the flip-flop. But since the flip-flop is already set, no changes occur at Q. The result is that the flip-flop responds to the first, and only to the first, high level at its S input, resulting in a "clean" low-to-high signal at its output (Q).



(a) Switch contact bounce eliminator



(b) Switch bounce

Fig. 8.32 **Debounce circuit**

When the switch is moved to position L, $S = 0$ and $R = 1$. Bouncing occurs at the R input due to the switch. Again, the flip-flop "sees" this as a series of high and low inputs. It simply responds to the *first* high level, and ignores all following transitions. The result is a "clean" high-to-low signal at the flip-flop output. The waveforms in Fig. 8.32b illustrate the behavior.

SELF-TEST

16. What is *switch contact bounce*?
17. Why is switch contact bounce important to account for in a digital system?

8.10 VARIOUS REPRESENTATIONS OF FLIP-FLOPS

There are various ways a flip-flop can be represented, each one suitable for certain application. Considering basic flip-flop truth table as starting point, this section derives these representations.

Characteristic Equations of Flip-flops

The characteristic equations of flip-flops are useful in analyzing circuits made of them. Here, next output Q_{n+1} is expressed as a function of present output Q_n and input to flip-flops. Karnaugh Map can be used to get the optimized expression and truth table of each flip-flop is mapped into it. This is shown in Fig. 8.33 for all types of flip-flops. The logic equations are presented in SOP form by forming largest group of 1's for each

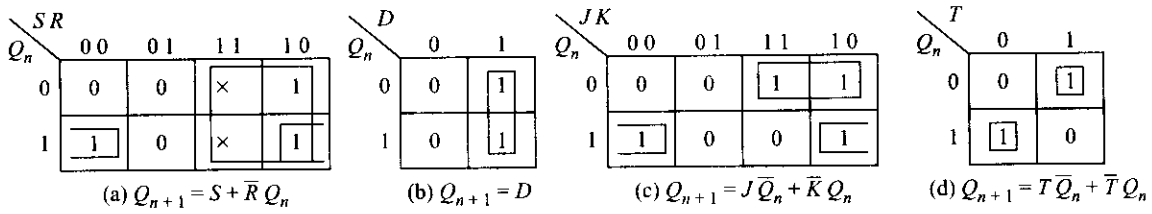


Fig. 8.33 **Characteristic equations of (a) *SR* flip-flop, (b) *D* flip-flop, (c) *JK* flip-flop, (d) *T* flip-flop**

flip-flop. For *SR* flip-flop, since $S = R = 1$ input is not allowed we have don't care states in corresponding locations in Karnaugh Map. This means, it does not matter if Q_{n+1} is 0 or 1 if $SR = 11$ as such a combination at input side will never arise.

The equation for *SR* flip-flop and all others thus can be represented in a summarized form as

SR flip-flop: $Q_{n+1} = S + R'Q_n$

JK flip-flop: $Q_{n+1} = JQ_n' + K'Q_n$

D flip-flop: $Q_{n+1} = D$

T flip-flop: $Q_{n+1} = TQ_n' + T'Q_n$

Flip-Flops as Finite State Machine

In a sequential logic circuit the value of all the memory elements at a given time define the *state* of that circuit at that time. Finite State Machine (FSM) concept offers a better alternative to truth table in understanding progress of sequential logic with time. For a complex circuit a truth table is difficult to read as its size becomes too large. In FSM, functional behavior of the circuit is explained using finite number of states. State transition diagram is a very convenient tool to describe an FSM. In Fig. 8.34 all the flip-flops are represented as finite state machine through their state transition diagrams.

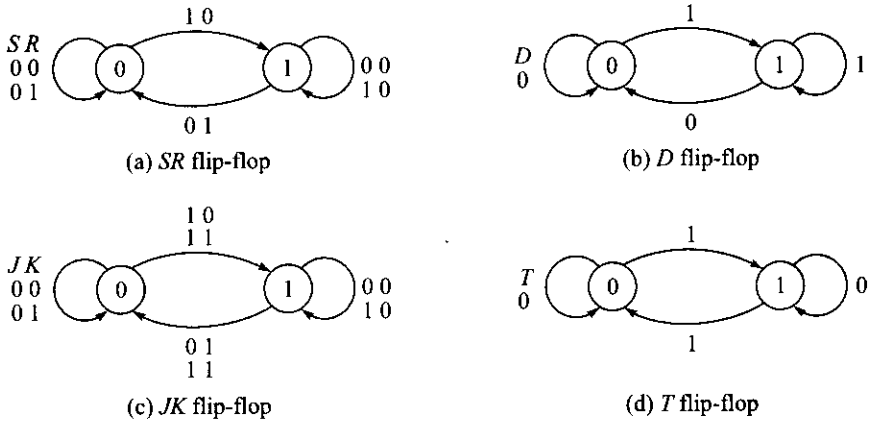


Fig. 8.34 **State transition diagram of (a) *SR* flip-flop, (b) *D* flip-flop, (c) *JK* flip-flop, (d) *T* flip-flop**

Let us see how state transition diagram for *SR* flip-flop is developed from its truth table or characteristic equation. Each flip-flop can be at either of 0 or 1 state defined by its stored value at any given time. Application of input may change the stored value, i.e. state of the flip-flop. This is shown by directional arrow and the corresponding input is written alongside. If *SR* flip-flop stores 0, then for *SR* = 00 or 01 the stored value does not change. For *SR* = 10, flip-flop output changes to 1. Note that, *SR* = 11 is not allowed in *SR* flip-flop. When *SR* flip-flop stores 1, application of *SR* = 00 or 10 does not change its value and only when *SR* = 01, output changes to 0. State transitions on application of all possible combination of inputs at every state are shown in Fig. 8.34(a) for *SR* flip-flop. The state transition diagrams are developed in a similar way for *D*, *JK*, *T* flip-flops and are shown in Figs. 8.34 (b), (c), (d) respectively. We see, the timing relation implicit in flip-flop truth tables are brought to the forefront by FSM concept and state transition diagram.

Flip-Flop Excitation Table

In synthesis or design problem excitation tables are very useful and its importance is analogous to that of truth table in analysis problem. Excitation table of a flip-flop is looking at its truth table in a reverse way. Here, flip-flop input is presented as a dependent function of transition $Q_n \rightarrow Q_{n+1}$ and comes later in the table. This is derived from flip-flop truth table or characteristic equation but more directly from its state transition diagram. Figure 8.35 gives a summary presentation of excitation tables of all the flip-flops.

$Q_n \rightarrow Q_{n+1}$		S	R	J	K	D	T
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	0	1
1	1	×	0	×	0	1	0

Fig. 8.35 **Excitation table of flip-flops**

From Fig. 8.34(a), one can see if present state is 0 application of *SR* = 0× does not alter its value where '×' denotes don't care condition in *R* input. State 0 to 1 transition occurs when *SR* = 10 is present at the input side while state 1 to 0 transition occurs if *SR* = 01. Present state 1 is maintained if *SR* = 0, i.e. *SR* = 00 or *SR* = 01. This is shown in Fig. 8.35 along *SR* column. Excitation table for other flip-flops are obtained in a similar way.

Note that, *JK* flip-flop has maximum number of don't care '×' conditions and *D* flip-flop input simply follows the value to which transition is made.

SELF-TEST

18. What is characteristic equation of a flip-flop?
19. What is a Finite State Machine?
20. How is excitation table different from flip-flop truth table?

Example 8.10 A fictitious flip-flop with two inputs *A* and *B* functions like this. For *AB* = 00 and 11 the output becomes 0 and 1 respectively. For *AB* = 01, flip-flop retains previous output while output complements for *AB* = 10. Draw the truth table and excitation table of this flip-flop.

Solution The truth table and corresponding excitation tables are presented in Figs. 8.36(a) and (b) respectively. For 0→0 transition we see *AB* need to be 00 or 01. Hence, we write *AB* = 0× in that place and similarly for other transitions.

A	B	Q_{n+1}
0	0	0
0	1	Q_n
1	0	$\overline{Q}_n$
1	1	1

(a)

$Q_n \rightarrow Q_{n+1}$	A	B
$0 \rightarrow 0$	0	×
$0 \rightarrow 1$	1	×
$1 \rightarrow 0$	×	0
$1 \rightarrow 1$	×	1

(b)

Fig. 8.36 Solution for example 8.10: (a) Truth table, (b) Excitation table

8.11 ANALYSIS OF SEQUENTIAL CIRCUITS

A sequential logic circuit contains flip-flops as memory elements and may also contain logic gates as combinatorial circuit elements. Analysis of a circuit helps to explain its performance. We may use truth tables of each building block or corresponding equations for this purpose. In this section, we look at important issues in an analysis problem through an example. In subsequent chapters, more analysis examples will be taken up.

Example 8.11 Consider, the sequential circuit shown in Fig. 8.37. It has only input *CLK* in the form of fixed frequency binary pulses that triggers both the flip-flops. An output X is generated from flip-flop outputs as shown. Analysis of this circuit will give how flip-flop values (or states) and more importantly output X change with input *CLK*. The steps are as follows.

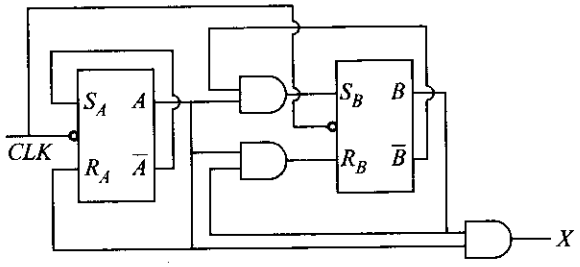


Fig. 8.37 A sequential logic circuit for analysis purpose

Note from the circuit diagram flip-flop input relations: $S_A = A_n'$, $R_A = A_n$ and $S_B = A_nB_n'$, $R_B = A_nB_n$.

Next, using characteristic equation of *SR* flip-flop (Section 8.9) we can write,

for flip-flop A

$$\begin{aligned} A_{n+1} &= S_A + R_A'A_n \\ &= A_n' + A_n'A_n \text{ (Substituting } S_A = A_n' \text{ and } R_A = A_n) \\ &= A_n' \end{aligned}$$

and for flip-flop B

$$\begin{aligned} B_{n+1} &= S_B + R_B'B_n \\ &= A_nB_n' + (A_nB_n)'B_n \text{ (Substituting } S_B = A_nB_n' \text{ and } R_B = A_nB_n) \\ &= A_nB_n' + (A_n' + B_n')B_n \text{ (Following De Morgan's Theorem)} \end{aligned}$$

$= A_n B_n' + A_n' B_n$

$= A_n \oplus B_n$

Now the output from the given circuit, $X_n = A_n B_n$

The equation shows that present (given by time index n) values of A and B flip-flop, also called *states* of the sequential circuit determine present output and next (given by time index $n + 1$) flip-flop values or state of the circuit. Thus, if present state is $B_n = 0$, $A_n = 0$ then present output $X_n = A_n B_n = 0.0 = 0$ and at the end of first clock cycle we get next state is $B_{n+1} = 0 \oplus 0 = 0$, $A_{n+1} = 0' = 1$. In next clock cycle present state is nothing but next state of previous cycle or $B_n = 0$, $A_n = 1$. The output now is generated as $X_n = 0.1 = 0$ and next state is determined as $B_{n+1} = 0 \oplus 1 = 1$, $A_{n+1} = 1' = 0$. Continuing this exercise we arrive at *state analysis table* also called state table as shown in Table 8.1.

Table 8.1 **State Analysis Table for Analysis Example**

Present State		*Present Input*			*Next State*			*Present Output*
B_n	A_n	$S_B = A_n B_n'$	$R_B = A_n B_n$	$S_A = A_n'$	$R_B = A_n$	$B_{n+1} = A_n \oplus B_n$	$A_{n+1} = A_n'$	$X = A_n B_n$
0	0	0	0	1	0	0	1	0
0	1	1	0	0	1	1	0	0
1	0	0	0	1	0	1	1	0
1	1	0	1	0	1	0	0	1
0	0	0	0	1	0	0	1	0
0	1	...	...				...	Repeats

We find that the states as well as output of the above circuit repeat after every four clocking periods and at every fourth clock period the output remains 1 for one clock period. The circuit thus behaves like a counter that counts number of clock pulses that has arrived at its input and signals when there is a count of four. A pictorial presentation of the performance of the circuit showing state transitions with each clock is shown in Fig. 8.38. The values within the circle follow syntax: $B_n A_n / X_n$. Flip-flop outputs defining current state is shown to the left of '/' and current output appears at right. Such circuit where output is directly derived from current state only and not from current inputs are called Moore circuit. If current inputs are also used in output forming logic it is called Mealy circuit. More about these are discussed in Chapter 11. Often, a state transition diagram of a sequential circuit serves better than the word description and is presented as final output of an analysis exercise.

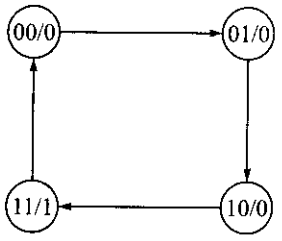


Fig. 8.38 **State transition diagram of the sequential circuit given in Fig. 8.37**

Analysis of a sequential circuit can also be done through timing diagram where all the input, output and if necessary intermediate variables are plotted against some reference signal say, clock input. The timing diagram obtained by analyzing circuit of Fig. 8.37 is shown in Fig. 8.39. The method followed is given next.

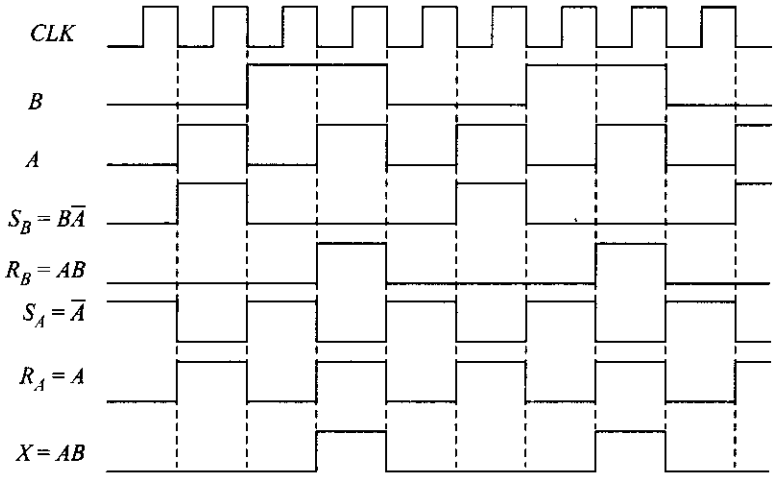


Fig. 8.39 **Timing diagram of the circuit given in Fig. 8.36**

We start with an initial state $B = 0, A = 0$ and note that this state can only change when negative edge of the clock comes. The next state values of B and A are dependent on current inputs S_B, R_B and S_A, R_A at the time of clock trigger. As done before, these input values are derived following relations given in the circuit diagram, i.e. $S_A = A'$, $R_A = A$ and $S_B = AB'$, $R_B = A_B$ (suffix n can be ignored). For $B = 0, A = 0$ we get $S_A = 1, R_A = 0, S_B = 0$ and $R_B = 1$ and these values can change only when B and A change, i.e. in next clock cycle. Thus above values of SR inputs of two flip-flops continue till next negative edge of the clock. For $S_B = 0$ and $R_B = 1$, at the negative edge of clock B remains at 0 (from truth table of SR flip-flop). Similarly for $S_A = 1, R_A = 0$ flip-flop A moves to 1. Thus we get B and A value of next clock cycle. Following above relation we now calculate SR input values of these flip-flops as $S_A = 0$, $R_A = 1$, $S_B = 1$ and $R_B = 0$ and these again remain constant up to next negative edge of the clock. Here as $S_B = 1$ and $R_B = 0$, B moves to 1 and as $S_A = 0, R_A = 1, A$ moves to 0 and remains constant till next clock trigger. SR inputs are again calculated and this process is continued for subsequent clock cycles. In each of these clock cycles we calculate and draw the output following relation $X = AB$. The timing diagram shows the states get repeated as $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$, and so on. Repetition occurs after every fourth clock cycle. The output $X = AB$, accordingly shows repetition as $0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0$ and remains high for one clock period every time flip-flop output becomes $B = 1, A = 1$.

A detailed analysis of various configurations of counter and its timing diagram will be presented in Chapter 10.

SELF-TEST

21. What is analysis of sequential circuit?
22. Which of truth table and excitation table is useful for analysis of a sequential circuit?

Example 8.12 Explain the function of the circuit shown in Fig. 8.40 through state transition diagram.

Solution The D flip-flop input can be written as $D = X \oplus Q_n$ and output $Y = XQ_n'$. Figure 8.41(a) shows the state table and Fig. 8.41(b) its state transition diagram. Note that, the circuit follows Mealy model and at any given state output is

generated from input to that state. Thus, outputs are shown by the side of the input in state transition diagram to right of the input and is separated by a '/' sign.

On careful observation, we can see something interesting in above circuit. If we ignore Y, then the D flip-flop block with Ex-OR gate as connected behaves like a T flip-flop where $T = X$.

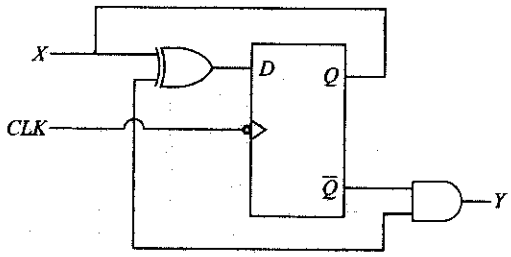
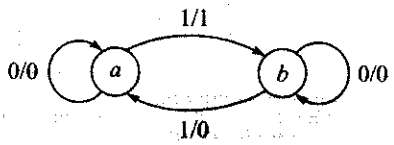


Fig. 8.40 **State transition diagram of Example 8.11**

Q_n	X	$D = X \oplus Q_n$	Q_{n+1}	$Y = X\overline{Q}_n$
0	0	0	0	0
0	1	1	1	1
1	0	1	1	0
1	1	0	0	0

(a)



(b)

Fig. 8.41 **Solution to Example 8.11: (a) State table, (b) State transition diagram**

8.12 CONVERSION OF FLIP-FLOPS: A SYNTHESIS EXAMPLE

Knowledge about how flip-flop of one type can be converted to another may be useful on various count. Say, when we have designed the circuit with one type and for implementation we get a different type from the store or the market. Redesign of the problem with available type of flip-flops may take considerable amount of time if the circuit is very complex. Instead one can convert the available type using few basic gate to the type in which design is done and implement the existing design.

Conversion of JK to SR, D, T is fairly straightforward as we see from their respective truth tables or characteristic equations. For example, one need not do anything extra to replace SR flip-flop from a design if SR flip-flop is not available, by JK flip-flop. This is because their truth tables are same except for input combination 11, which in design with SR flip-flop is taken care of not to appear in the input side. Hence, replacing SR with JK flip-flop does not pose any problem. However, the reverse is not true. In design with JK flip-flop there remains possibility of 11 appearing at input side and that combination of input is forbidden for SR flip-flop. Again, comparing truth tables or characteristic equations of JK and D flip-flops we see that putting an inverter from J to K ($K = J'$) we get D flip-flop from JK flip-flop where $J = D$. T flip-flop can be obtained from JK flip-flop by making $T = J = K$.

We show here how to convert an SR flip-flop to a JK flip-flop through a systematic approach, as a general methodology for synthesis or design of sequential logic circuit. A detailed study on various design problems and related issues are presented in Chapter 11.

In step one of this method, we look into JK flip-flop truth table and specifically note, $Q_n \rightarrow Q_{n+1}$ transitions for a given combination of inputs JK and present state Q_n. Since the synthesis element is SR flip-flop we shall

refer to its excitation table to identify *SR* input combination for a required $Q_n \rightarrow Q_{n+1}$ transition. Table 8.2 shows truth table of *JK* flip-flop as well as necessary *SR* inputs for $Q_n \rightarrow Q_{n+1}$ transitions. Such tables are also known as *state synthesis table.*

Table 8.2 State Synthesis Table for *SR* to *JK* Flip-Flop Conversion

J_n	K_n	Q_n	$\rightarrow Q_{n+1}$	S_n	R_n
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1

The next step is to write *SR* inputs as a function of *JK* inputs and present state Q_n. Karnaugh Map derived from Table 8.2 for *SR* inputs are shown in Fig. 8.42 along with their design equations.

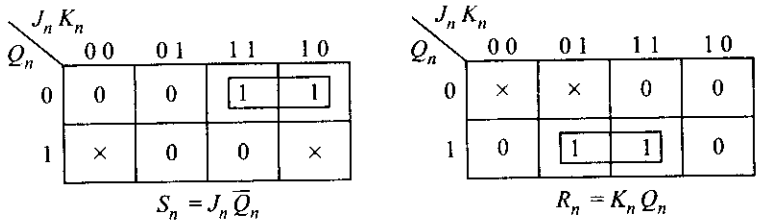


Fig. 8.42 Karnaugh Map and Design equations for *SR* inputs

The final synthesized circuit developed from these equations, are shown in Fig. 8.43. The functional block within dotted lines made up of an *SR* flip-flop and two AND gates, behave like a *JK* flip-flop.

Thus conversion between flip-flops, in simple cases can be done comparing their respective truth tables. For other cases, the steps shown above can be followed. Refer to Example 8.13 and Problems 8.30 to 8.32.

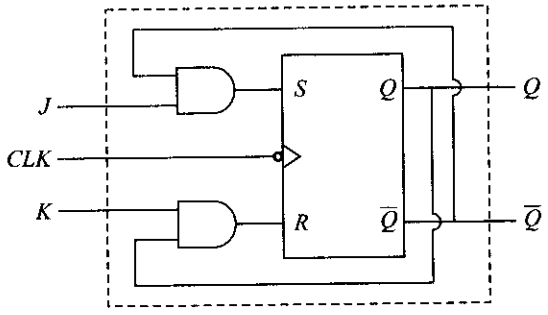


Fig. 8.43 Conversion of *SR* flip-flop to *JK* flip-flop.

SELF-TEST

23. Why flip-flop conversion is needed?
24. What is the basic difference between analysis and synthesis steps?
25. What is the difference between state analysis table and state synthesis table?

Example 8.13 Show how a *D* flip-flop can be converted to *SR* flip-flop.

Solution Note characteristic equation of two flip-flops.

For *SR* flip-flop: $Q_{n+1} = S + R'Q_n$ and for *D* flip-flop: $Q_{n+1} = D$

Thus with $D = S + R'Q_n$ we get circuit shown in Fig. 8.44 which behaves like an *SR* flip-flop but made from a *D* flip-flop and basic logic gates. Method as shown in Section 8.11 also gives same solution.

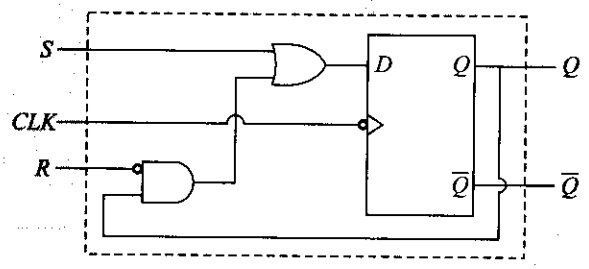


Fig. 8.44 **Solution for Example 8.11. *D* flip-flop converted to *SR* flip-flop**

8.13 HDL IMPLEMENTATION OF FLIP-FLOP

We continue our discussions on HDL from earlier chapters and in this section we look at how to represent a flip-flop using Verilog HDL. As discussed before, behavioral model is preferred for sequential circuit and **always** keyword is used in all these circuits. Since, sequential logic design also includes combinatorial design at some places we may use dataflow model for that. To start with let us see how a *D* latch (Fig. 8.15) and SR latch (Fig. 8.11) are expressed in HDL. We have used characteristic equation corresponding flip-flops given in Section 8.9. The explanation of the codes are simple. If EN = 1, output changes according to equation and if EN = 0, output does not change, i.e. remains latched to previous value.

```
module DLatch(D,EN,Q);
input D,EN;
output Q;
reg Q;
always @ (EN or D)
   if (EN) Q=D;

  //from characteristic equation
endmodule
```

```
module SRLatch(S,R,EN,Q);
input S,R,EN;
output Q;
reg Q;
always @ (EN or S or R)
   if (EN) Q=S|(~R&Q);
//from characteristic equation

endmodule
```

Next we discuss how to describe a clocked flip-flop. The following Verilog code describes a *D* flip-flop with positive edge trigger, negative edge trigger and positive edge trigger with reset (CLR) given in Figs. 8.23 (a), (b) and (c) respectively. Here, the CLR input is active low, i.e. it clears the output ($Q = 0$) when CLR is 0. We use keywords **posedge** and **negedge** for this. With keyword **always** it ensures execution of always block once every clock cycle at corresponding edge. For asynchronous CLR we use a particular nomenclature of Verilog HDL. The **always** sensitivity list (after @) contains any number of edge statements including clock

and asynchronous inputs. The **always** block puts all asynchronous conditions in the beginning through else or **else if** and the *last* **else** statement responds to clock transition.

```
module DFFpos(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always@(posedge C)
  Q=D;
endmodule
```

```
module DFFneg(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always@(negedge C)
  Q=D;
endmodule
```

```
module DFFpos_clr(D,C,CLR,Q);
input D,C,CLR; //C is clock
output Q;
reg Q;
always@(posedge C or negedge CLR)
   if (~CLR) Q=1'b0;
   //Q stores 1 binary bit 0
  else Q=D;
endmodule
```

Example 8.14 Write a Verilog code that converts an *D* flip-flop to an *SR* flip-flop following Fig. 8.43 of Section 8.11.

Solution The code is given as follows. See how combinatorial logic part of the circuit is expressed by **assign** statement.

```
module SRFFneg(S,R,C,Q);
input S,R,C; //C is clock
output Q;
wire DSR;
assign DSR = S|(~R&Q); //combinatorial logic shown in fig.8.45
DFFneg  D1(DSR,C,Q);  //instantiates negative edge triggered D FF
endmodule
```

```
module DFFneg(D,C,Q);
input D,C; //C is clock
output Q;
reg Q;
always @ (negedge C)
   Q=D;
endmodule
```

Example 8.15 Explain the use of following Verilog code in test bench preparation of sequential logic circuit.

```
Initial
    begin
        clk = 1'b0;
        repeat (20)
        #50 clk = ~clk;
    end
```

Solution The keyword **initial** says following code is run for once. The variable 'clk' is of 1 binary digit and is initialized with 0 at time = 0. Keyword **repeat** ensures repetition of following statement 20 times. In that statement,

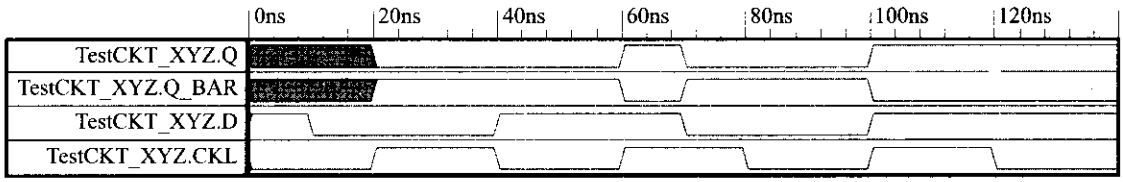
variable clk is complemented after a delay of 50 ns. Thus, clk toggles between 1 and 0 every 50 ns and for 20 times generating 10 cycles of 50 + 50 = 100 ns duration each. In a test bench, clk can be fed as clock input to simulate a sequential circuit for a finite duration. The number of clock pulse generated can be changed by changing number after **repeat** and clock period can be changed by changing delay after # sign.

Example 8.16 The Verilog code given in first column generates output given in second column and corresponding timing waveform is given next. Draw the digital circuit diagram from Verilog code and explain the output.

```
module CKT_XYZ(Q,Q_BAR,D,CLK);
output Q,Q_BAR;
input D,CLK;
wire X,Y;
nand U1 (X,D,CLK) ;
nand U2 (Y,X,CLK) ;
nand U3 (Q,Q_BAR,X);
nand U4 (Q_BAR,Q,Y);
endmodule

module TestCKT_XYZ;
wire Q, Q_BAR;
reg D, CLK;
CKT_XYZ  xyz(Q, Q_BAR, D, CLK);
initial
begin
 $monitor($time, "CLK = %b, D= %b,
  Q= %b\n",CLK,D,Q);
  D=1;CLK=0;
  #10 D = 0;   #30 D = 1;  #30 D = 0;
  #30 D = 1;
 #40 $finish; /* the module will terminate after
  140ns*/
end
always
 #20 CLK = ~CLK;
endmodule
```

```
  0   CLK = 0, D = 1, Q = x
 10   CLK = 0, D = 0, Q = x
 20   CLK = 1, D = 0, Q = 0
 40   CLK = 0, D = 1, Q = 0
 60   CLK = 1, D = 1, Q = 1
 70   CLK = 1, D = 0, Q = 0
 80   CLK = 0, D = 0, Q = 0
100   CLK = 1, D = 1, Q = 1
120   CLK = 0, D = 1, Q = 1
```



Solution The circuit diagram from the structural model given in the code is shown in Fig. 8.45. The test bench displays in the monitor time elapsed and CLK, D, Q (in binary) through first statement after **begin**. D initially 1 toggles after a delay of 10 ns, 30 ns, 30 ns, 30 ns. Simulation stops after further 40 ns taking a total time of 10 + 30 + 30 +30 + 40 = 140 ns. Clock toggles at every 20 ns starting with a value 0.

The circuit shows that if CLK = 0, U1 and U2 outputs are 1 irrespective of other inputs and Q, Q_BAR remains latched to previous value through cross coupled U3 and U4. When CLK = 1, D can change U1 output such that $X = D'$ also final output $Q = D$. The timing diagram shows Q_BAR = Q'. Thus the circuit behaves like a high level triggered D Flip-Flop.

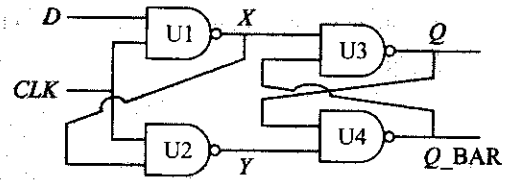


Fig. 8.45 **Circuit diagram of Verilog code given in Example 8.16**

PROBLEM SOLVING WITH MULTIPLE METHODS

Problem Analyze the circuit shown in Fig. 8.46 and find the output Y. Consider that the flip-flops are initially reset.

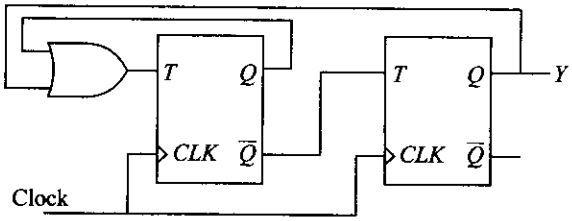


Fig. 8.46 **A T flip-flop based circuit for analysis purpose**

Solution We follow three different methods to analyze the circuit and identify the performance of Y.

In Method-1, we use state table approach. We make use of the fact that a T flip-flop does not change its state if $T = 0$ but it toggles when $T = 1$ at the clock trigger.

Let us name the first flip-flop as X and its input and output as T_X and X respectively. Similarly, let the second flip-flop be named Y and its input is T_Y while its output is already assigned as Y. Then, the state table is shown in Fig. 8.47. We find that the circuits move from states 00, 01, 10, 00, ... repetitively and the output Y goes HIGH once in three cycles and remains HIGH for one clock period.

Clock cycle	X_n	Y_n	$T_X = X_n + Y_n$	$T_Y = X_n'$	X_{n+1}	Y_{n+1}
0	0	0	0	1	0	1
1	0	1	1	1	1	0
2	1	0	1	0	0	0
3	0	0	... repeats ...			

Fig. 8.47 **State Table to analyze circuit diagram of Fig. 8.47**

In Method-2, we make use of the characteristic equation of T flip-flop.

From Section 8.9, we know that $Q_{n+1} = TQ_n' + T'Q_n$

By following similar X and Y naming of two flip-flops as in Method-1, we find that

For X flip-flop, input: $T_X = X_n + Y_n$

output: $X_{n+1} = (X_n + Y_n)X_n' + (X_n + Y_n)'X_n$

$= Y_n X_n' + X_n' Y_n' X_n$

$= Y_n X_n'$

For Y flip-flop, input: $T_Y = X_n'$

output: $Y_{n+1} = X_n' Y_n' + (X_n')' Y_n$

$= X_n' Y_n' + X_n Y_n$

The final solution is shown in Fig. 8.48.

X_n	Y_n	$X_{n+1} = Y_n X_n'$	$Y_{n+1} = X_n' Y_n' + X_n Y_n$
0	0	0	1
0	1	1	0
1	0	0	0
0	0	... repeats	...

Fig. 8.48 Solution using Method-2

In Method-3, we make use of the timing diagram as shown in Fig. 8.49. We note that the flip-flops are positive edge triggered. The T input just before the positive edge decides output of the flip-flop in next cycle.

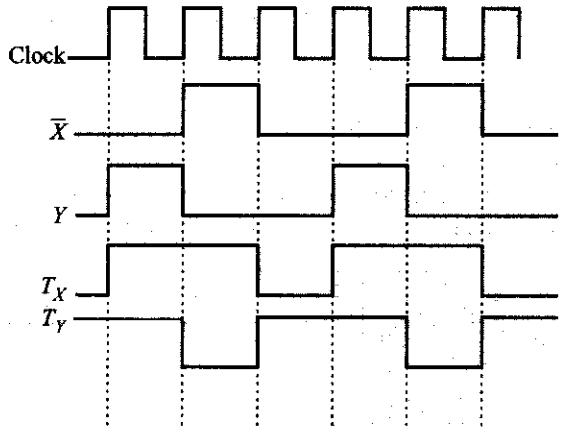


Fig. 8.49 Solution using Method-3

We start with initial $XY = 00$. Then we draw T_X by ORing X and Y waveforms and T_Y by inverting Y waveform. T_X and T_Y before positive edge decide value of X and Y respectively in next clock cycle (from T flip-flop truth table).

SUMMARY

A flip-flop is an electronic circuit that has two stable states. It is said to be bistable. A basic *RS* flip-flop, or latch can be constructed by connecting two NAND gates or two NOR gates in series with a feedback connection. A signal at the set input of an *RS* flip-flop will force the *Q* output to become a 1, while a signal at the reset input will force *Q* to become a 0.

A simple *RS* flip-flop or latch is said to be transparent—that is, its output changes state whenever a signal appears at the *R* or *S* inputs. An *RS* flip-flop can be modified to form a clocked *RS* flip-flop whose output can change states only in synchronism with the applied clock.

An *RS* flip-flop can also be modified to form a *D* flip-flop. In a *D* latch, the stored data may be changed while the clock is high. The last value of *D* before the clock returns low is the data that is stored. With edge-triggered *D* flip-flops, the data is sampled and stored on either the positive or negative clock edge.

The values of *J* and *K* determine what a *JK* flip-flop does on the next clock edge. When both are low, the flip-flop retains its last state. When *J* is low and *K* is high, the flip-flop resets. When *J* is high and *K* is low, the flip-flop sets. When both are high, the flip-flop toggles. In this last mode, the *JK* flip-flop can be used as a frequency divider.

There are various ways to represent a flip-flop like truth table, characteristic equation, state transition diagram or excitation table. Flip-flop treated as a finite state machine highlights its functional aspect. Analysis of a sequential circuit helps to understand performance of a given circuit in a systematic manner and through synthesis we develop circuit diagram for a specified problem.

GLOSSARY

- ***asynchronous*** Independent of clocking. The output can change without having to wait for a clock pulse.
- ***bistable*** Having two stable states.
- ***bistable multivibrator*** Another term for an *RS* flip-flop.
- ***buffer register*** A group of memory elements, often flip-flops, that can store a binary word.
- ***characteristic equation*** logic expression describing a flip-flop.
- ***edge triggering*** A circuit responds only when the clock is in transition between its two voltage states.
- ***finite state machine*** functional description of sequential circuit.
- ***flip-flop*** An electronic circuit that has two stable states.
- ***hold time*** The minimum amount of time that data must be present after the clock trigger arrives.
- ***latch*** Another term for an *RS* flip-flop.
- ***Mealy model*** output is dependent both on current state and input to the circuit.
- ***Moore model*** output is dependent only on current state of the circuit.
- ***propagation delay*** The amount of time it takes for the output to change states after an input trigger.
- ***setup time*** The minimum amount of time required for data inputs to be present before the clock arrives.
- ***state*** the set of memory values at any given time for a sequential logic circuit.
- ***synchronous*** When outputs change states in time with a clock. A clock signal must be present in order for the outputs to change states.
- ***transparent*** The condition that exists when the flip-flop output changes immediately after its inputs (*R*, *S*, *J*, *K*, *D*) change state.

PROBLEMS

Section 8.1

8.1 List as many bistable devices as you can think of—either electrical or mechanical. (***Hint:*** Magnets, lamps, relays, etc.)

8.2 Redraw the NOR-gate flip-flop in Fig. 8.3b and label the logic level on each pin for $R = S = 0$. Repeat for $R = S = 1$, for $R = 0$ and $S = 1$, and for $R = 1$ and $S = 0$.

8.3 Redraw the NAND-gate flip-flop in Fig. 8.7a and label the logic level on each pin for $\overline{R} = \overline{S} = 0$. Repeat for $\overline{R} = \overline{S} = 1$, for $\overline{R} = 1$, and $\overline{S} = 0$, and for $\overline{R} = 0$ and $\overline{S} = 1$.

8.4 Redraw the NAND-gate flip-flop in Fig. 8.8a and label the logic level on each pin for $R = S = 0$. Repeat for $R = S = 1$, for $R = 0$ and $S = 1$, and for $R = 1$ and $S = 0$.

Section 8.2

8.5 The waveforms in Fig. 8.50 drive the clocked *RS* flip-flop in Fig. 8.11. The clock signal goes from low to high at points *A*, *C*, *E*, and *G*. If *Q* is low before point *A* in time:

a. At what point does *Q* become a 1?

b. When does *Q* reset to 0?

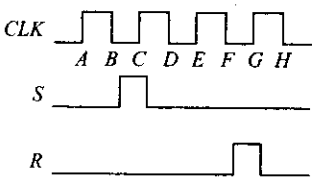


Fig. 8.50

8.6 Use the information in the preceding problem and draw the waveform at *Q*.

8.7 Prove that the flip-flop realizations in Fig. 8.12 are equivalent by writing the logic level present on every pin when $R = S = 0$ and the clock is high. Repeat for $R = S = 1$, for $R = 1$ and $S = 0$, and for $R = 0$ and $S = 1$. Describe what happens when the clock is low.

8.8 The waveforms in Fig. 8.51 drive a *D* latch as shown in Fig. 8.15. What is the value of *D* stored in the flip-flop after the clock pulse is over?

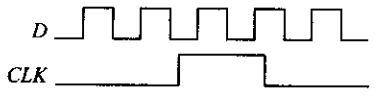


Fig. 8.51

Section 8.3

8.9 What is the advantage offered by an edge-triggered *RS* flip-flop over a clocked or gated *RS* flip-flop?

8.10 The waveforms in Fig. 8.18d illustrate the typical operation of an edge-triggered *RS* flip-flop. This circuit was connected in the laboratory, but the *R* and *S* inputs were mistakenly reversed. Draw the resulting waveform for *Q*.

8.11 An edge-triggered *RS* flip-flop will be used to produce the waveform *Q* with respect to the clock as shown in Fig. 8.52a. First, would you use a positive-edge- or a negative-edge-triggered flip-flop? Why? Draw the waveforms necessary at *R* and *S* to produce *Q*.

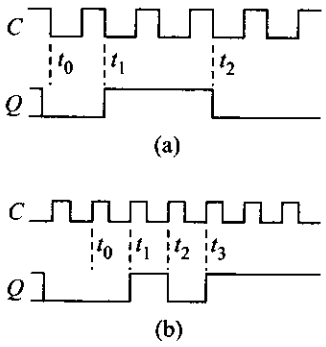


Fig. 8.52

8.12 An edge-triggered *RS* flip-flop will be used to produce the waveform *Q* with respect to the

clock as shown in Fig. 8.52b. First, would you use a positive-edge- or a negative-edge-triggered flip-flop? Why? Draw the waveforms necessary at R and S to produce Q.

Section 8.4

8.13 A positive-edge-triggered D flip-flop has the input waveforms shown in Fig. 8.51. What is the value of Q after the clock pulse?

8.14 A negative-edge-triggered D flip-flop is driven by the waveforms shown in Fig. 8.51. What is the value of D stored in the flip-flop?

8.15 A D flip-flop has the following data sheet information: setup time = 5 ns; hold time = 10 ns; propagation time = 15 ns.

a. How far ahead of the triggering clock edge must the data be applied?
b. How long after the clock edge must the data be present to ensure correct storage?
c. How long after the clock edge before the output changes?

Section 8.5 and Section 8.6

8.16 Redraw the JK flip-flop in Fig. 8.23a. Connect $J = K = 1$. (This can be done by connecting the J and K inputs to $+V_{CC}$) Now, begin with $Q = 1$, and show what logic level results on each pin after one positive clock pulse. Allow one more positive clock pulse and show the resulting logic level on every pin.

8.17 In the JK flip-flop in Fig. 8.25a, $J = K = 1$. A 1-MHz square wave is applied to its C input. It has a propagation delay of 50 ns. Draw the input square wave and the output waveform expected at Q. Be sure to show the propagation delay time.

8.18 Repeat Prob. 8.17, but use the flip-flop in Fig. 8.25c.

8.19 In Prob. 8.17, what is the period of the clock? What are the period and frequency of the output waveform at Q?

8.20 Repeat Prob. 8.17, assuming that the C input has a frequency of 10 MHz.

Section 8.7

8.21 Draw two flip-flops like the one shown in Fig. 8.25b and show how to connect them such that a 500-kHz square wave applied to pin 1 will result in a 125-kHz square wave at pin 11. Give a complete wiring diagram (show each pin connection).

8.22 Explain the meaning of the symbol ⌉ in Fig. 8.29a.

8.23 What is the significance of the symbol ⎍ in the truth table of Fig. 8.29?

8.24 Why do most modern designs incorporate edge-triggered JK flip-flops rather than pulse-triggered JK flip-flops?

Section 8.8

8.25 Show how to use a simple $\overline{R}\overline{S}$ latch to eliminate switch contact bounce (see Fig. 8.32a).

8.26 There is contact bounce present with the SPDT switch in Fig. 8.53 just as with the SPST switch discussed in Fig. 8.31. However, the RS latch used in Fig. 8.53 will remove all contact bounce, and V_o will be *high* with the switch in position 1 and *low* with the switch in position 2. Explain exactly how this debounce circuit works. You might use waveforms as an aid. Incidentally, the 54/74279 can be used to construct four of these circuits.

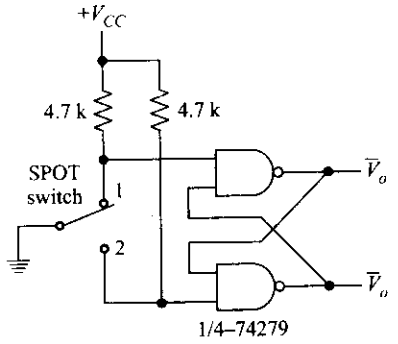


Fig. 8.53 **Debounce circuit**

Section 8.9 and Section 8.10

8.27 (a) Derive the characteristic equation and (b) draw state transition diagram of the fictitious flip-flop described in Example 8.10.

8.28 Explain the difference between Mealy and Moore model of sequential circuit.

8.29 Analyze the following circuit and explain what it does.

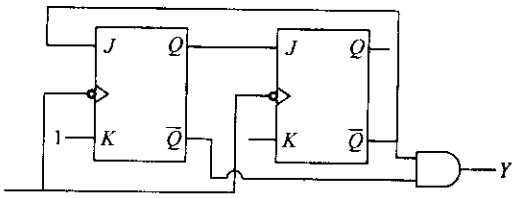


Fig. 8.54

Section 8.11

8.30 Show how to convert *D* flip-flop to *JK* flip-flop.

8.31 Convert *T* flip-flop to *D* flip-flop.

8.32 Convert *SR* flip-flop to *T* flip-flop.

LABORATORY EXPERIMENT

AIM: The aim of this experiment is to study *D* flip-flop and *JK* flip-flop and use them for analysis of sequential logic circuits.

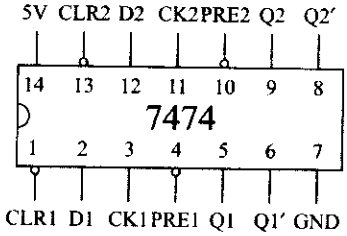
Theory: The truth table of *D* flip-flop and *JK* flip-flop are as follows.

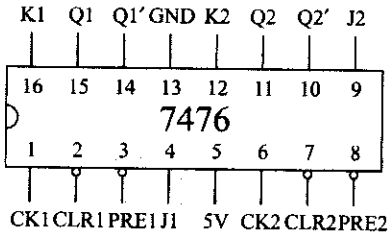
C	D	Q_{n+1}
0	X	Q_n (Last state)
↑	0	0
↑	1	1

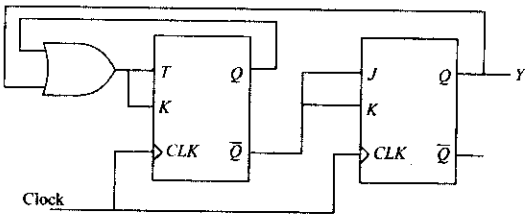
C	J	K	Q_{n+1}	Action
↑	0	0	Q_n (Last state)	No change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	$\overline{Q}_n$ (toggle)	Toggle

Their characteristic equations are:

D flip-flop: $Q_{n+1} = D_n$







JK flip-flop: $Q_{n+1} = JQ_n' + K'Q_n$

Apparatus: 5 VDC Power supply, Multimeter, Bread Board, Clock Generator, and Oscilloscope

Work element: IC 7474 is a dual, edge clocked, *D* flip-flop with both PRESET and CLEAR input while 7476 is a dual, edge clocked, *JK* flip-flop that too, has both PRESET and CLEAR input. Verify the truth table of IC 7474 and 7476. Find if it is positive or

negative edge triggered. Appreciate the function of PRESET and CLEAR if it is asynchronous or synchronous with clock. The clock may be available from clock generator or you may use 555 based pulse generator developed in laboratory experiment of previous chapter.

Connect 7476 and 7432 (OR gate) as shown, so that the analysis circuit is realized. Use CLEAR to initialize both the flip-flops to 00. Then apply clock, and see the clock and Y in a dual trace oscilloscope. Use 7474 to prepare an SR flip-flop as shown in Fig. 8.43 and find its truth table.

Answers to Self-tests

1. R stands for RESET ($Q = L$). S stands for SET ($Q = H$).
2. Quad means "four." There are four flip-flops in this IC.
3. A NAND-gate latch is considered active-low because a low input signal is required to change Q.
4. X means don't care—this input at this time has no effect.
5. Simply hold the EN input low (at 0 Vdc).
6. The D flip-flop is easier to use because it requires only one input (D).
7. It means the output responds immediately to input signals.
8. A circuit is activated by the leading edge of the clock.
9. The latch is transparent. The edge-triggered flip-flop only changes state in synchronism with the clock.
10. None. The flip-flop is disabled with C held low.
11. PRESET is active high. A high level at PRESET will set Q high.
12. The JK flip-flop has an additional input condition—$J = K = H$. This causes the flip-flop to toggle with the clock. The $R = S = H$ input condition is not allowed with an RS flip-flop.
13. Cross-couple the outputs back to the input AND gates.
14. The J and K inputs are transparent in a pulse-triggered flip-flop. Thus, J and K must be static while the clock is high.
15. While C is high, the master is SET-RESET according to the J and K inputs. When C goes low, the contents of the master shift into the slave, and Q is SET-RESET accordingly.
16. Switch contact bounce is the bouncing that occurs when a mechanical, spring-actuated device is operated.
17. The bouncing action produces multiple PTs and NTs, which may introduce unintentional signals!
18. Logic relation showing next state as a function of current state and current inputs.
19. That explains the functional behavior of a sequential circuit through finite number states and its transition from one state to another.
20. It is truth table written in a reverse way such that inputs are shown dependent on a particular state transition.
21. Finding what a given circuit does.
22. Truth table.
23. By this one need not redesign the whole circuit if flip-flop one kind is not available.
24. In analysis, problem begins with a circuit diagram and ends in state transition diagram or performance description. It uses flip-flop truth table or characteristic equation in this process. In synthesis, the path is reverse and we use excitation table instead of truth table.
25. In state analysis table, input of the flip-flops used in the circuit is written first followed by state transition whereas in state synthesis table it is other way.

Registers

9

OBJECTIVES

- **Understand serial in–serial out shift registers and be familiar with the basic features of the 74LS91 register**
- **Understand serial in–parallel out shift registers and be familiar with the basic features of the 74164 register**
- **Understand parallel in–serial out shift registers and be familiar with the basic features of the 74166 register**
- **Understand parallel in–parallel out shift registers and be familiar with the basic features of the 74174 and 7495A registers**
- **Understand working of Universal shift register with the basic features of the 74194 register.**
- **State various uses of shift registers**

A register is a very important digital building block. A data register is often used to momentarily store binary information appearing at the output of an encoding matrix. A register might be used to accept input data from an alphanumeric keyboard and then present this data at the input of a microprocessor chip. Similarly, registers are often used to momentarily store binary data at the output of a decoder. For instance, a register could be used to accept output data from a microprocessor chip and then present this data to the circuitry used to drive the display on a CRT screen. Thus registers form a very important link between the main digital system and the input-output channels. A universal asynchronous receiver transmitter (UART) is a chip used to exchange data in a microprocessor system. The UART is constructed using registers and some control logic.

A binary register also forms the basis for some very important arithmetic operations. For example, the operations of complementation, multiplication, and division are frequently implemented by means of a register. A shift register can also be connected to form a number of different types of counters. Shift registers

as sequence generator and sequence detector and also as parallel to serial converters offers very distinct advantages.

The many different applications of registers, along with the myriad of techniques for using them, are simply too numerous to be discussed here. Our intent is to study the detailed operation of the four basic types of shift registers. With this knowledge, you will have the ability to study and understand exactly how a shift register is used in any specific application encountered.

9.1 TYPES OF REGISTERS

A register is simply a group of flip-flops that can be used to store a binary number. "There must be one flip-flop for each bit in the binary number. For instance, a register used to store an 8-bit binary number must have eight flip-flops. Naturally the flip-flops must be connected such that the binary number can be entered (shifted) into the register and possibly shifted out. A group of flip-flops connected to provide either or both of these functions is called a *shift register*.

The bits in a binary number (let's call them the data) can be moved from one place to another in either of two ways. The first method involves shifting the data 1 bit at a time in a serial fashion, beginning with either the most significant bit (MSB) or the least significant bit (LSB). This technique is referred to as *serial shifting*. The second method involves shifting all the data bits simultaneously and is referred to as *parallel shifting*.

There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift the data out of the register. This leads to the construction of four basic register types as shown in Fig. 9.1—serial in–serial out, serial in–parallel out, parallel in–serial out, and parallel in–parallel out. All of these configurations are commercially available as TTL MSI/LSI circuits. For instance:

Serial in–serial out—54/74LS91, 8 bits

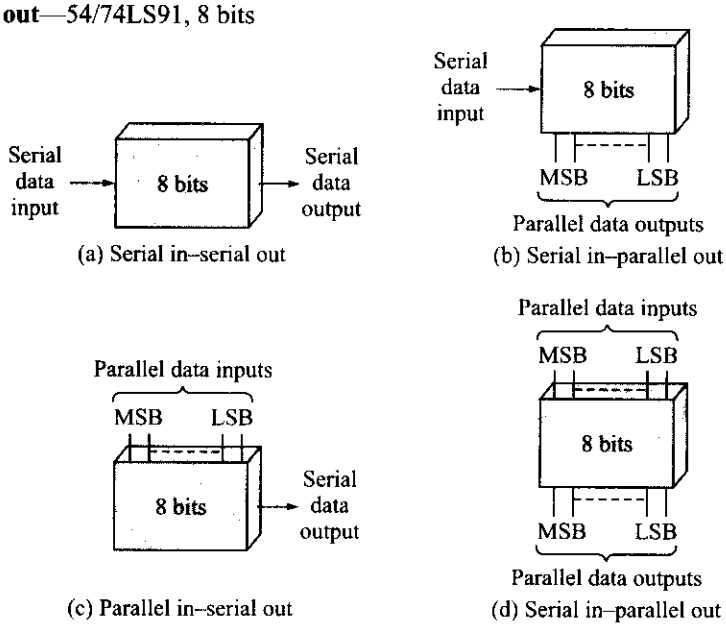


Fig. 9.1 Shift register types

Serial in–parallel out—54/74164, 8 bits

Parallel in–serial out—54/74165, 8 bits

Parallel in–parallel out—54/74198, 8 bits

We now need to consider the methods for shifting data in either a serial or parallel fashion. Data shifting techniques and methods for constructing the four different types of registers are discussed in the following sections.

9.2 SERIAL IN–SERIAL OUT

In this section we discuss how data is serially entered or exited from a shift register. The flip-flops used to construct registers are usually edge-triggered *JK*, *SR* or *D* types. We begin our discussion with shift registers made from *D* type flip-flops and then extend the idea to other types.

Consider four *D* flip-flops connected as shown in Fig. 9.2a forming 4-bit shift register. A common clock provides trigger at its negative edge to all the flip-flops. As output of one *D* flip-flop is connected to input of the next at every clock trigger data stored in one flip-flop is transferred to the next. For this circuit transfer takes place like this $Q \rightarrow R$, $R \rightarrow S$, $S \rightarrow T$ and *serial data input* is transferred to Q. Let us see how actual data transfer takes place by an example.

Assume, all the flip-flops are initially cleared. Let a binary waveform, as shown along *D* of Fig. 9.2b be fed to *serial data input* of the shift register. Corresponding *Q*, *R*, *S*, *T* are also shown in the figure.

At clock edge *A*, flip-flop *Q* has input 0 from serial data in *D*, flip-flop *R* has input 0 from output of *Q*, flip-flop *S* has input 0 from output of *R* and flip-flop *T* has input 0 from output of *S*. When clock triggers, these inputs get transferred to corresponding flip-flop outputs simultaneously so that *QRST* = 0000. Thus at clock trigger, values at *DQRS* is transferred to *QRST*.

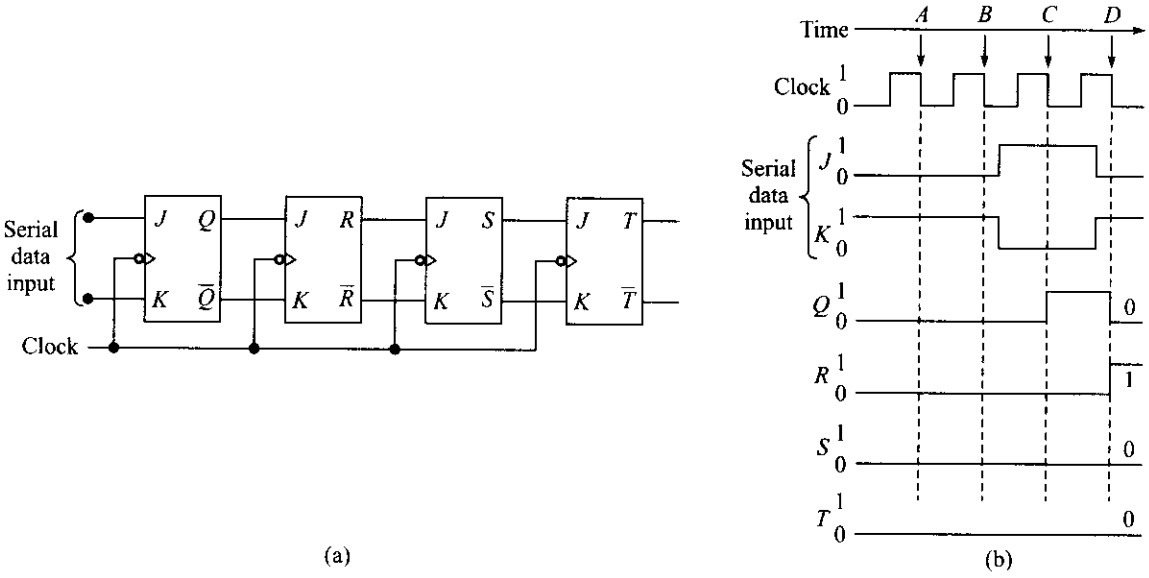


Fig. 9.2 4-bit serial input shift register

At clock edge *B*, serial data in = 0, i.e. $DQRS = 0000$. So after NT at *B*, $QRST = 0000$. Serial data becomes 1 in next clock cycle.

At clock edge *C*, $DQRS = 1000$ and after NT $QRST = 1000$. Serial data goes to 0 in next clock cycle such that **at clock edge *D***, $DQRS = 0100$ and after NT $QRST = 0100$. Example 9.1 will give another illustration of such data transfer.

A shift register made up of *JK* or *SR* flip-flops has non-inverting output *Q* of one flip-flop connected to *J* or *S* input of next flip-flop and inverting output *Q'* connected to *K* or *R* input respectively. For the first flip-flop, between *J* and *K* (or *S* and *R*) an inverter is connected and *J* (or *S*) input is treated as serial data in. Note that, in this configuration both *JK* and *SR* flip-flops effectively act like a *D* flip-flop.

Example 9.1 Show how a number 0100 is entered serially in a shift register shown in Fig. 9.2a using state table.

Solution Figure 9.3 presents the state table. The timing diagram corresponding to this is discussed in this section. Note how the data flow across the flip-flops is highlighted by arrow direction.

Clock	Serial input	*Q*	*R*	*S*	*T*
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4		0	0	1	0

Fig. 9.3 Data transfer through serial input in a shift register

Example 9.2 Draw the waveforms to shift the number 0100 into the shift register shown in Fig. 9.3 on the next page.

Solution The waveforms for this register will appear exactly as in Fig. 9.2 provided the waveform labeled *K* is eliminated and waveform *J* is labeled *D*.

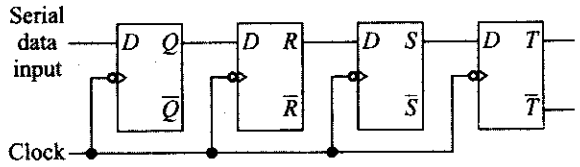


Fig. 9.3a 4-bit serial input shift register

At this point, we have developed the ideas for shifting data into a register in serial form; the serial data input can be classified as either *JK* or *D*, depending on the flip-flop type used to construct the register. Now, how about shifting data out of the register?

Let's take another look at the register in Fig. 9.3a, and suppose that it has the 4-bit number $QRST = 1010$ stored in it. If a clock signal is applied, the waveforms shown in Fig. 9.4 will be generated. Here's what happens:

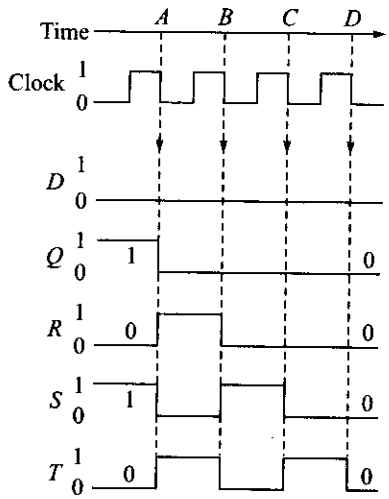


Fig. 9.4

Before Time *A* The register stores the number $QRST = 1010$. The LSB (a 0) appears at T.

At Time *A* The entire number is shifted one flip-flop to the right. A 0 is shifted into Q and the LSB is shifted out the right end and lost. The register holds the bits $QRST = 0101$, and the second LSB (a 1) appears at T.

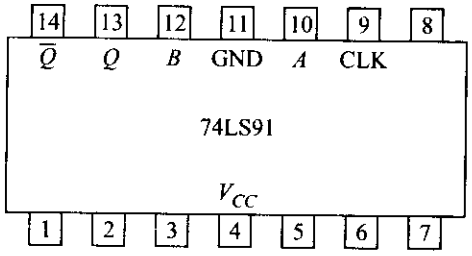
At Time *B* The bits are all shifted one flip-flop to the right, a 0 shifts into Q, and the third LSB (a 0) appears at T. The register holds $QRST = 0010$.

At Time *C* The bits are all shifted one flip-flop to the right, a 0 shifts into Q, and the MSB (a 1) appears at T. The register holds $QRST = 0001$.

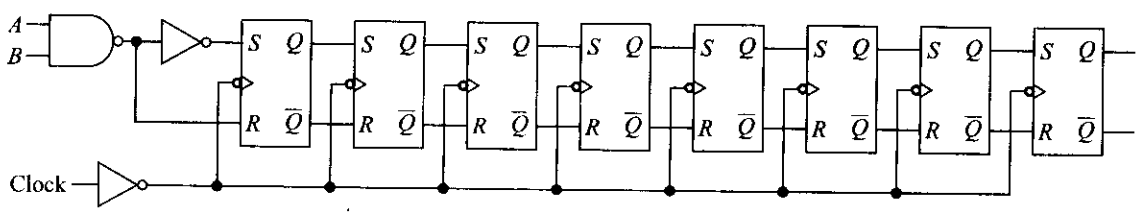
At Time *D* The MSB is shifted out the right end and lost, a 0 shifts into Q, and the register holds $QRST = 0000$.

To summarize, we have caused the number stored in the register to appear at T (this is the register output) 1 bit at a time, beginning with the LSB, in a serial fashion, over a time period of four clock cycles. In other words, the data stored was shifted out of the register at flip-flop T in a serial fashion. Thus, not only is this a serial-input shift register, it is also a serial-output shift register. It is important to realize that the stored number is shifted out of the right end of the register and lost after four clock times. Notice that the complement of the output data stream is also available at $\overline{T}$.

The pinout and logic diagram for a 74LS91 shift register are shown in Fig. 9.5. This is an 8-bit TTL MSI chip. There are eight *RS* flip-flops connected to provide a serial input as well as a serial output.



(a) DIP pinout



(b) Logic diagram

Fig. 9.5 **74LS91 8-bit shift register**

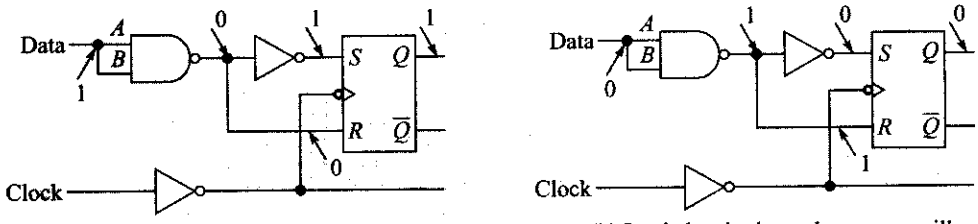
The clock input at each flip-flop is negative-edge-trigger-sensitive. However, since the applied clock signal is passed through an inverter, data will be shifted on the positive edges of the input clock pulses.

The inverter connected between R and S on the first flip-flop means that this circuit functions as a D-type flip-flop. So, the input to the register is a single line on which the data to be shifted into the register appears serially. The data input is applied at either A (pin 10) or B (pin 12). Notice that a data level at A (or B) is complemented by the NAND gate and then applied to the R input of the first flip-flop. The same data level is complemented by the NAND gate and then complemented again by the inverter before it appears at the S input. So, a 1 at input A will set the first flip-flop (in other words, this 1 is shifted into the first flip-flop) on a positive clock transition.

The NAND gate with inputs A and B simply provides a gating function for the input data stream if desired. If gating is not desired, simply connect pins 10 and 12 together and apply the input data stream to this connection.

Example 9.3 Examine the logic levels at the input of a 74LS91 and show how a 1 and then a 0 are shifted into the register.

Solution The input logic and the first flip-flop are redrawn in Fig. 9.6a, and a 1 is applied at the data input A. The R input is 0, the S input is 1, and the flip-flop will clearly be set when the clock goes high. In other words, the 1 at the data input will shift into the flip-flop. In Fig. 9.6b, a 0 is applied at the data input A. The R input is 1, the S input is 0, and the flip-flop will be reset when the clock goes high. The input 0 is thus shifted into the flip-flop.



(a) Logic levels shown by arrows will set the flip-flop

(b) Logic levels shown by arrows will reset the flip-flop

Fig. 9.6 Example 9.2

SELF-TEST

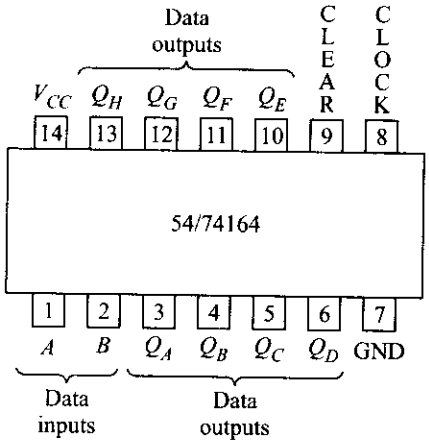
1. What is the largest decimal number that can be stored (in binary form) in a 74LS91 register?
2. Is a 74LS91 register sensitive to PTs or to NTs?

9.3 SERIAL IN–PARALLEL OUT

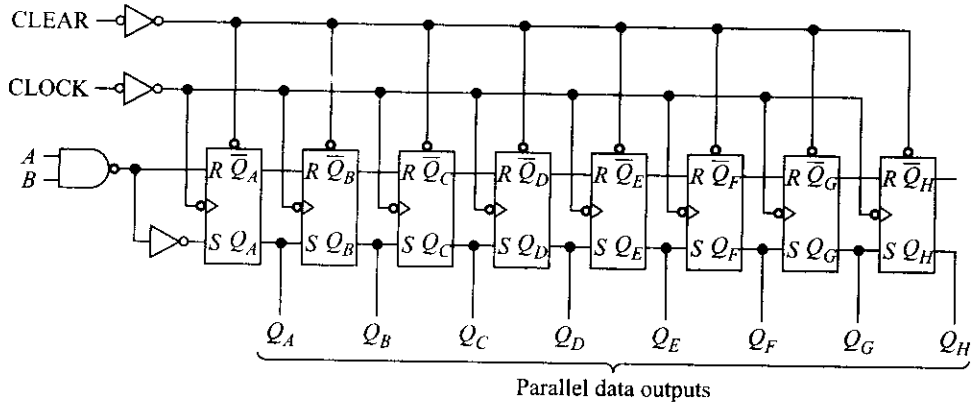
The second type of register mentioned in Sec. 9.1 is one in which data is shifted in serially, but shifted out in parallel. In order to shift the data out in parallel, it is simply necessary to have all the data bits available as outputs at the same time. This is easily accomplished by connecting the output of each flip-flop to an output pin. For instance, an 8-bit shift register would have eight output lines—one for each flip-flop in the register. The basic configuration is shown in Fig. 9.1b.

The 54/74164 is an 8-bit serial input–parallel output shift register. The pinout and logic diagram for this device are given in Fig. 9.7. It is constructed by using *RS* flip-flops having clock inputs that are sensitive to NTs. A careful examination of the logic diagram in Fig. 9.7b will reveal that this register is exactly like the 74LS91 discussed in the previous section—with two exceptions: (1) the true side of each flip-flop is available as an output—thus all 8 bits of any number stored in the register are available simultaneously as an output (this is a parallel data output); and (2) each flip-flop has an asynchronous clear input. Thus a low level at the clear input to the chip (pin 9) is applied through an amplifier and will reset (clear) every flip-flop. Notice that this is an asynchronous signal and can be applied at any time, without regard to the clock waveform and also that this signal is level sensitive. As long as the clear input to the chip is held low, the flip-flop outputs will all remain low. (The register will contain all zeros!)

Shifting data into the register in a serial fashion is exactly the same as the previously discussed 74LS91. Data at the serial inputs may be changed while the clock is either low or high, but the usual setup and hold times must be observed. The data sheet for this device gives setup time as 30 ns minimum and hold time as



(a) DIP pinout



(b) Logic diagram

Fig. 9.7 **54/74164 8-bit shift register**

0.0 ns. Since data are shifted into the register on PTs, the data input line must be stable from 30 ns before the PT until the clock transition is complete.

Let's take a look at the gated serial inputs *A* and *B*. Suppose that the serial data is connected to *A*; then *B* can be used as a control line. Here's how it works:

***B* is Held High** The NAND gate is enabled and the serial input data passes through the NAND gate inverted. The input data is shifted serially into the register.

***B* is Held Low** The NAND-gate output is forced high, the input data stream is inhibited, and the next positive clock transition will shift a 0 into the first flip-flop. Each succeeding positive clock transition will shift another 0 into the register. After eight clock pulses, the register will be full of zeros!

Example 9.4 How long will it take to shift an 8-bit number into a 54164 shift register if the clock is set at 10 MHz?

Solution A minimum of eight clock periods will be required since the data is entered serially. One clock period is 100 ns, so it will require 800 ns minimum.

Example 9.5 For the register in Example 9.4, when must the input data be stable? When can it be changed?

Solution The data must be stable from 30 ns before a positive clock transition until the positive transition occurs. This leaves 70 ns during which the data may be changing (see Fig. 9.8).

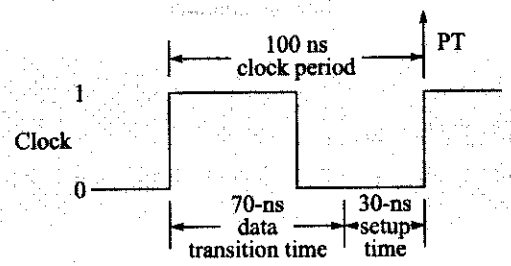


Fig. 9.8 Example 9.5

The waveforms shown in Fig. 9.9 show the typical response of a 54/74164. The serial data is input at *A* (pin 1), while a gating control signal is applied at *B* (pin 2). The first clear pulse occurs at time *A* and simply resets all flip-flops to 0.

The clock begins at time *B*, but the first PT does nothing since the control line is low. At time *C* the control line goes high, and the first data bit (a 0) is shifted into the register at time *D*.

The next 7 data bits are shifted in, in order, at times *E*, *F*, *G*, *H*, *I*, *J* and *K*. The clock remains high after time *K*, and the 8-bit number 0010 1100 now resides in the register and is available on the eight output lines. This assumes that the LSB was shifted in first and appears at Q_H. Notice that the clock *must be stopped* after its positive transition at time *K*, otherwise shifting will continue and the data bits will be lost.

Finally, another clear pulse occurs at time *L*, the flip-flops are all reset to zero, and another shift sequence may begin. Incidentally, the register can be cleared by holding the control line at *B* low and allowing the clock to run for eight PTs. This simply shifts eight 0s into the register.

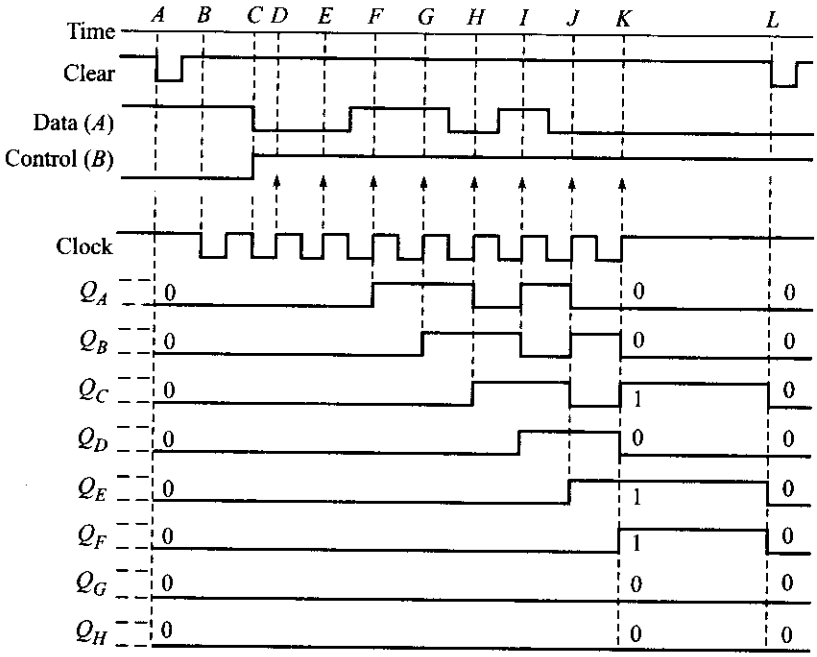


Fig. 9.9

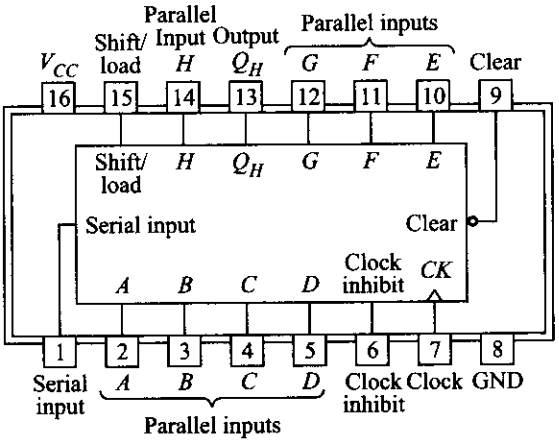
SELF-TEST

3. What is the setup time for a 74164 shift register?
4. How many clock periods are required to shift an 8-bit number into a 74164 register? To extract an 8-bit number?

9.4 PARALLEL IN–SERIAL OUT

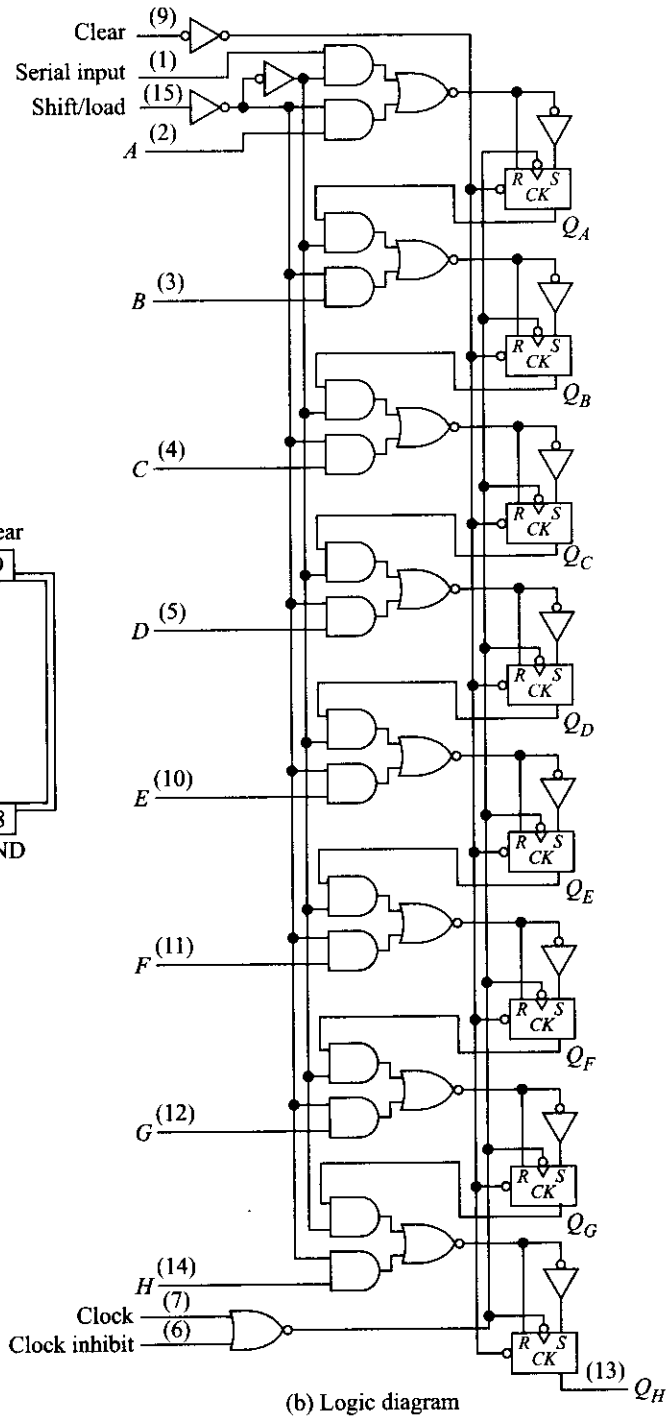
In prior sections, the ideas necessary for shifting data into and out of a register in serial have been developed. We can now use these same ideas to develop methods for the parallel entry of data into a register. There are a number of different techniques for the parallel entry of data, but we shall concentrate our efforts on commercially available TTL. At first glance, the logic diagrams for some of the shift registers seem rather formidable (see, for instance, the block diagram for the 54/74166); but they aren't really. The 54/74166, for instance, is an 8-bit shift register, and the same circuit is repeated eight times. So, it's necessary to study only one of the eight circuits, and that's what we'll do here.

The pinout and logic block diagram for a 54/74166 are given in Fig. 9.10. The functional description given on the TTL data sheet says that this is an 8-bit shift register, capable of either serial or parallel data entry, and serial data output. Notice that there are eight *RS* flip-flops, each with some attached logic circuitry. Let's analyze one of these circuits by starting with the *RS* flip-flops and then adding logic blocks to accomplish our needs.



Positive logic: see description

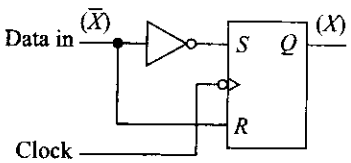
(a) Pinout



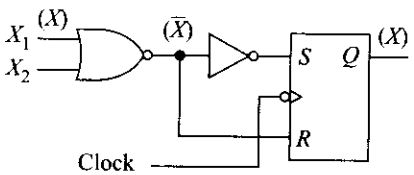
(b) Logic diagram

Fig. 9.10 54/74166

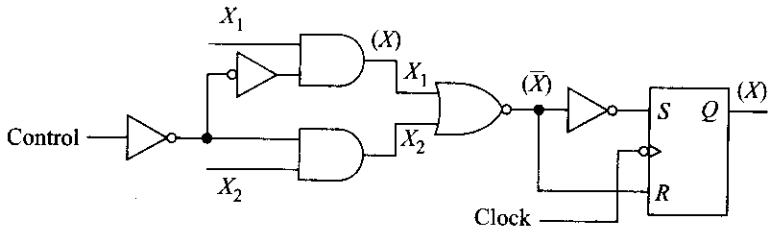
First recognize that the clocked *RS* flip-flop and the attached inverter given in Fig. 9.11a form a type *D* flip-flop. If a data bit X is to be clocked into the flip-flop, the complement of X must be present at the input. For instance, if $\overline{X} = 0$, then $R = 0$ and $S = 1$, and a 1 will be clocked into the flip-flop when the clock transitions.



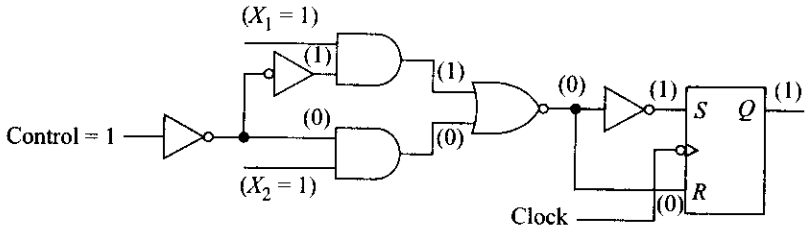
(a) Type *D* flip-flop



(b) NOR-gate added



(c) Control logic added



(d) Example 9.5

Fig. 9.11

Now, add a NOR gate as shown in Fig. 9.11b. If one leg of this NOR gate is at ground level, a data bit X at the other leg is simply inverted by the NOR gate. For instance, if $X = 1$, then at the output of the NOR gate $\overline{X} = 0$, allowing a 1 to be clocked into the flip-flop. This NOR gate offers the option of entering data from two different sources, either X_1 or X_2. Holding X_2 at ground will allow the data at X_1 to be shifted into the flip-flop; conversely, holding X_1 at ground will allow data at X_2 to be shifted in.

The addition of two AND gates and two inverters as shown in Fig. 9.11c will allow the selection of data X_1 or data X_2. If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus X_1 will appear at the upper leg of the NOR gate while the lower leg of the NOR gate will be at ground level. On the other hand, if the control line is low, the upper AND gate is disabled while the lower AND gate is enabled. This allows X_2 to appear at the lower leg of the NOR gate while the upper leg of the NOR gate is at ground level. You should now study this circuit until your understanding is crystal clear! Consider writing 0 or 1 at each gate leg in response to various inputs. To summarize:

CONTROL is High Data bit at X_1 will be shifted into the flip-flop at the next clock transition.

CONTROL is Low Data bit at X_2 will be shifted into the flip-flop at the next clock transition.

Example 9.6 For the circuit in Fig. 9.11c, write the logic levels present on each gate leg if CONTROL = 1, $X_1 = 1$, and $X_2 = 1$.

Solution The correct levels are given in parentheses in Fig. 9.11d. The data value 1 at X_1 is shifted into the flip-flop when the clock transitions.

A careful examination will reveal that exactly eight of the circuits given in Fig. 9.11c are connected together to form the 54/74166 shift register shown in Fig. 9.10. The only question is: how are they connected? The answer is: they are connected to allow two different operations: (1) the parallel entry of data and (2) the operation of shifting data serially through the register from the first flip-flop Q_A toward the last flip-flop Q_H.

If the data input labeled X_2 in Fig. 9.11c is brought out individually for each flip-flop, these eight inputs will serve as the parallel data entry inputs for an 8-bit number *ABCD EFGH*. These eight inputs are labeled *A, B, C, D, E, F, G,* and *H* in Fig. 9.10. The control line is labeled shift/load. Holding this shift/load control line low will enable the lower AND gate for each flip-flop, and the 8-bit number will be LOADED into the flip-flops with a single clock transition—PARALLEL input.

Holding the shift/load control line high will enable the upper AND gate for each flip-flop. If the input from this upper AND gate receives its data from the prior flip-flop in the register, each clock transition will shift a data bit from one flip-flop into the following flip-flop—proceeding in a direction from Q_A toward Q_H. In other words, data will be shifted through the register serially! In the first flip-flop in the register, the upper AND-gate input is labeled serial input. Thus data can also be entered into this register in a serial fashion. To summarize:

Shift/Load is Low A single clock transition loads 8 bits of data (*ABCD EFGH*) into the register in parallel.

Shift/Load is High Clock transitions will shift data through the register serially, with entering data applied at the SERIAL INPUT.

Notice that the clock is applied through a two-input NOR gate. When clock inhibit is held low, the clock signal passes through the NOR gate inverted. Since the register flip-flops respond to NTs, data will shift into the register on the PTs of the clock. When clock inhibit is high, the NOR-gate output is held low, and the clock is prevented from reaching the flip-flops. In this mode, the register can be made to stop and hold its contents.

A low level at the clear input can be applied at any time without regard to the clock, and it will immediately reset all flip-flops to 0. When not in use, it should always be held high.

The truth table in Fig. 9.12 summarizes the operation of the 54/74166

Inputs						Internal Levels		Outputs
Clear	Shift/ load	Clock inhibit	Clock	Serial	Parallel $A \ldots H$	Q_A and Q_B		Q_H
L	X	X	X	X	X	L	L	L
H	X	L	L	X	X	Q_{AO}	Q_{BO}	Q_{HO}
H	L	L	↑	X	$a \ldots h$	a	b	h
H	H	L	↑	H	X	H	Q_{An}	Q_{Gn}
H	H	L	↑	L	X	L	Q_{An}	Q_{Gn}
H	X	H	↑	X	X	Q_{AO}	Q_{BO}	Q_{HO}

X = Irrelevant, H = High level, L = Low level
↑ = Positive transition
$a \ldots h$ = Steady state input level at $A \ldots H$ respectively
Q_{AO}, Q_{BO} = Level at Q_A, $Q_B \ldots$ before steady state
Q_{An}, Q_{Gn} = Level of Q_A or Q_B before most recent transition ( ) ↑

Fig. 9.12 **54/74166 truth table**

8-bit shift register. You should study this table in conjunction with the logic diagram to understand clearly how the register can be used.

Example 9.7 Which entry in the truth table in Fig. 9.12 accounts for the parallel entry of data?

Solution The third entry from the top; clear is high, shift/load is low, clock inhibit is low, a positive clock transition occurs, and the serial data input is irrelevant.

SELF-TEST

5. What is the purpose of the shift/load line on the 74166?
6. Is data shifted into and out of a 74166 register on clock PTs or on clock NTs?

9.5 PARALLEL IN–PARALLEL OUT

The fourth type of register discussed in the introductory section of this chapter is designed such that data can be shifted either into or out of the register in parallel. In fact, simply adding an output line from each flip-flop in the 54/74166 discussed in the previous section would meet the parallel in–parallel out requirements. [It would, of course, require a larger dual in-line package (DIP)—say, a 24-pin package.]

The 54/74174

The 74174 in Fig. 9.13 is an example of a parallel in–parallel out register. The Texas Instruments data sheet refers to it as a hex *D*-type flip-flop with clear. It is simply a parallel arrangement of six *D*-type flip-flops. Each flip-flop is negative-edge-triggered, and thus a PT will shift data into the register. The six data bits, D_1 through D_6 are all shifted into the register in parallel. The stored data is immediately available, in parallel, at the outputs, Q_1 through Q_6. This type of register is simply used to store data, and is sometimes called a *data register,* or *data latch.* Notice that it is not possible to shift stored data either to the right or to the left. A low level at the clear input will immediately reset all flip-flops low. The clear input is asynchronous—that is, it can be done at any time and it takes precedence over all other inputs.

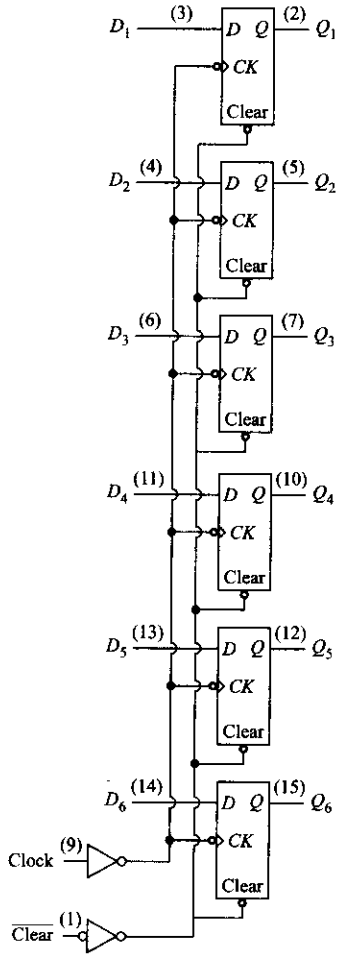


Fig. 9.13 54/74174

Example 9.8 The 74LS174 data sheet gives a setup time of 20 ns and a hold

time of 5 ns. What is the minimum required width of the data input levels (D_1 ... D_6) for the 74LS174 in Fig. 9.13?

Solution The data inputs must be steady at least 20 ns before the PT of the clock, and they must be held for a minimum of 5 ns after the PT. Thus, the data input levels must be held steady for a minimum of 25 ns (see Fig. 8.24 for comparison).

The 54/74198

The 54/74198 is an 8-bit TTL MSI having both parallel input and parallel output capability. The DIP pinout for this device is given in Fig. 9.14 on the next page. It uses positive edge-triggered flip-flops, as indicated by the small triangle at pin 11. Notice that a 24-pin package is required since 16 pins are needed just for the input and output data lines. Not only does this chip satisfy the parallel input-output requirements; it can also be used to shift data through the register in either direction—referred to as *shift right* and *shift left*. All the registers previously discussed have the ability to shift right, that is, to shift data serially from the data input flip-flop toward the right, or from a flip-flop Q_A toward flip-flop Q_B. We now need to consider how to shift left.

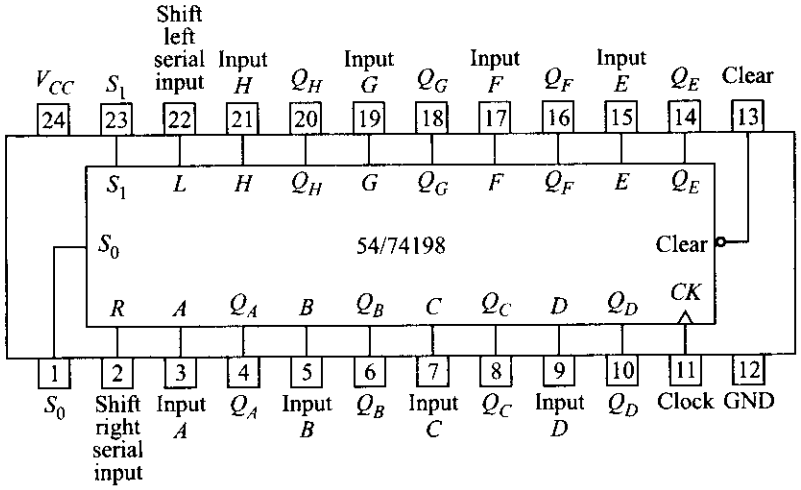
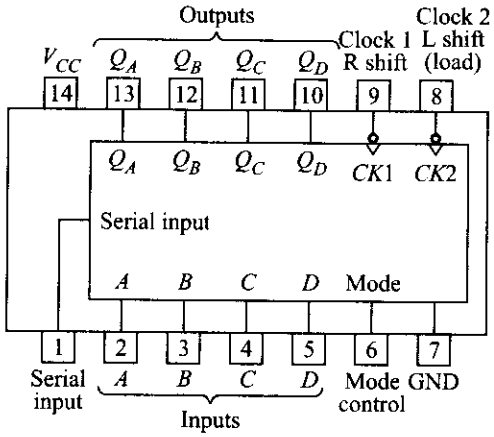


Fig. 9.14 **54/74198, 8-bit shift register. Parallel input–parallel output**

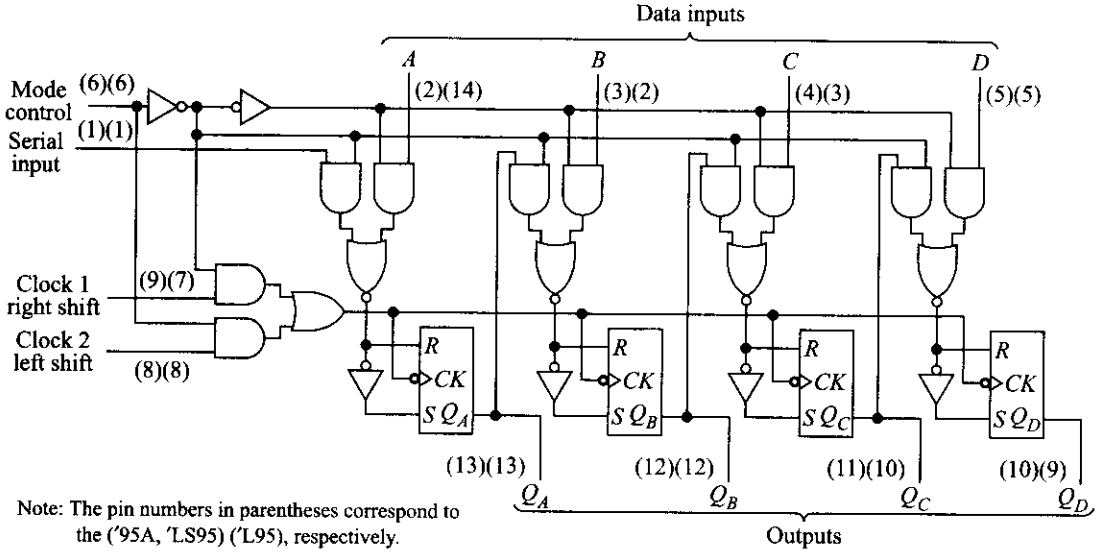
There are a number of 4-bit parallel in–parallel out shift registers available since they can be conveniently packaged in a 16-pin DIP. An 8-bit register can be created by either connecting two 4-bit registers in series or by manufacturing the two 4-bit registers on a single chip and placing the chip in a 24-pin package (such as the 54/74198). Let's analyze a typical 4-bit register, say, a 54/7495A.

The data sheet for the 54/7495A describes it as a 4-bit parallel-access shift register. It also has serial data input and can be used to shift data to the right (from Q_A toward Q_B) and in the opposite direction—to the left. The DIP pinout and logic diagram are given in Fig. 9.15. The basic flip-flop and control logic used here are exactly the same as used in the 54/74164 as shown in Fig. 9.11c.

The parallel data outputs are simply the Q sides of each of the four flip-flops in the register. In fact, note that the output Q_D could be used as a serial output when data is shifted from left to right through the register (right shift).



(a) Pinout



Note: The pin numbers in parentheses correspond to the ('95A, 'LS95) ('L95), respectively.

(b) Logic diagram

Fig. 9.15 54/7495A

When the mode control line is held high, the AND gate on the right input to each NOR gate is enabled while the left AND gate is disabled. The data at inputs, A, B, C and D will then be loaded into the register on a negative transition of the clock—this is parallel data input.

When the mode control line is low, the AND gate on the right input to each NOR gate is disabled while the left AND gate is enabled. The data input to flip-flop Q_A is now at serial input; the data input to Q_B is Q_A and so on down the line. On each clock NT, a data bit is entered serially into the register at the first flip-flop Q_A, and each stored data bit is shifted one flip-flop to the right (toward the last flip-flop Q_D). This is the serial input of data (at serial input), and also the right-shift operation.

In order to effect a shift-left operation, the input data must be connected to the D data input as shown in Fig. 9.16 below. It is also necessary to connect Q_D to C, Q_C to B, and Q_B to A as shown in Fig. 9.16. Now, when the mode control line is held high, data bit will be entered into flip-flop Q_D, and each stored data bit will be shifted one flip-flop to the left on each clock NT. This is also serial input of data (but at input D) and is the left-shift operation. Notice that the connections described here can either be hard wired or can be made by means of logic gates.

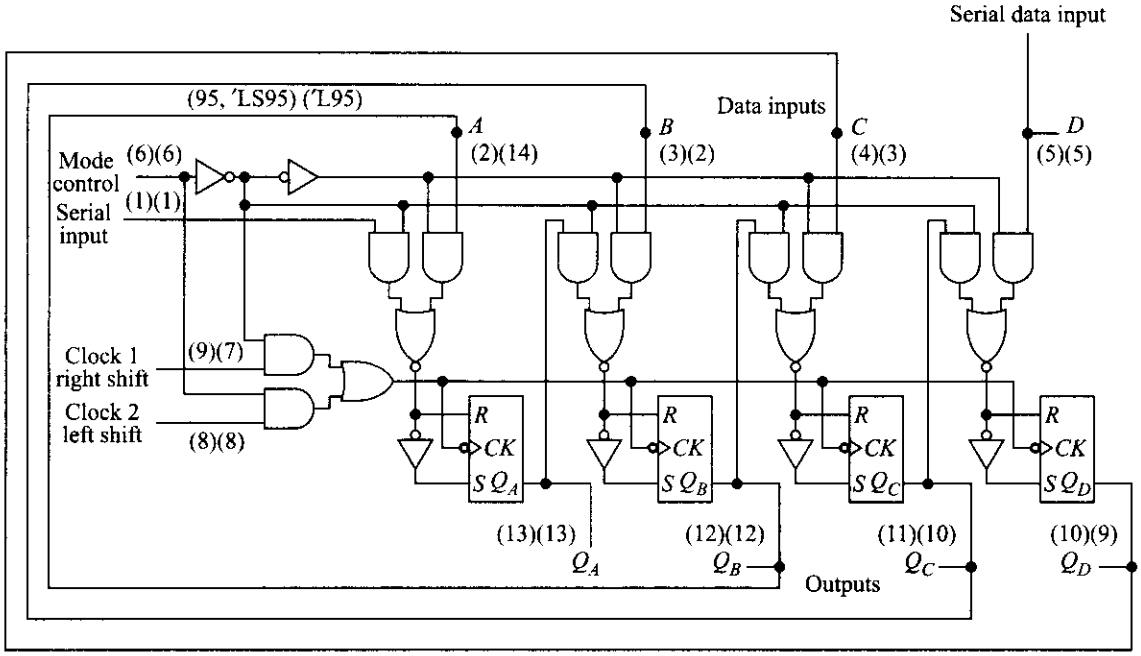


Fig. 9.16 **54/7495A wired for shift left**

There are two clock inputs—clock 1 and clock 2. This is to accommodate requirements where the clock used to shift data to the right is separate from the clock used to shift data to the left. If such a requirement is unnecessary, simply connect clock 1 and clock 2 together. The clock signal will then pass through the AND-OR gate combination noninverted, and the flip-flops will respond to clock NTs.

Example 9.9 Draw the waveforms you would expect if the 4-bit binary number 1010 were shifted into a 54/7495A in parallel.

Solution The mode control line must be high. The data input lines must be stable for more than 10 ns prior to the clock NTs (setup time for the data sheet information). A single clock NT will enter the data. (The waveforms are given in Fig. 9.17.) If the clock is stopped after the transition time T, the levels on the input data lines may be changed. However, if the clock is not stopped, the input data line levels must be maintained.

At this point, it simply cannot be overemphasized that the input control lines to any shift register *must be controlled at all times*! Remember, the register will do something every time there is a clock transition. What it does is entirely dependent on the levels applied at the control inputs. If you do not account for input control levels, you simply cannot account for the behavior of the register!

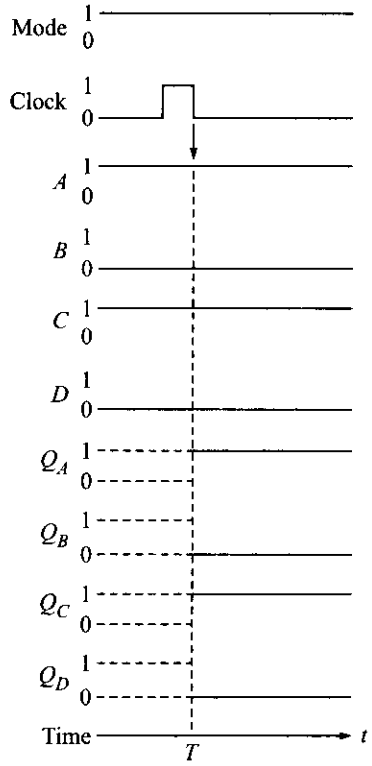


Fig. 9.17 Example 9.8

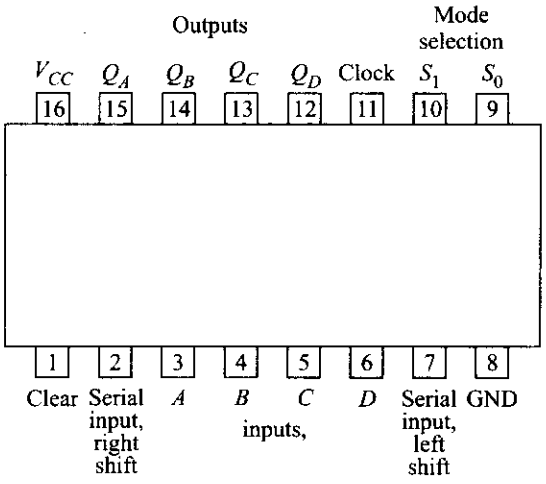


Fig. 9.18 74194 pinout

SELF-TEST

7. How can the 7495A, a 4-bit register, be used to store 8-bit numbers?
8. Why does the 7495A have two separate clock inputs?

9.6 UNIVERSAL SHIFT REGISTER

In Section 9.1, we have seen that for basic types of shift register, the following operations are possible—serial in–serial out, serial in–parallel out, parallel in–serial out, and parallel in–parallel out. Serial in or serial out again can be made possible by shifting data in any of the two directions, left shift ($Q_A \leftarrow Q_B \leftarrow Q_C \leftarrow Q_D \leftarrow$ Data in) and right shift (Data in $\rightarrow Q_A \rightarrow Q_B \rightarrow Q_C \rightarrow Q_D$). A universal shift register can perform all the four operations and is also bidirectional in nature. 7495A, described in previous section, is quite versatile except for the fact that it is in-built for right shift; the left shift is achieved through parallel loading (Fig. 9.16) and thus requires external wiring.

The 74194 is a 4-bit universal shift register in 16 pin package with pinout diagram as shown in Fig. 9.18. A, B, C and D are four parallel inputs, and Q_A, Q_B, Q_C and Q_D are corresponding parallel outputs. There are two separate inputs for serial data for left and right shift. In addition, there are two mode control inputs which

select the mode of operation for the universal shift register according to Table 9.1. The subscript n and $n + 1$ represent two consecutive states and in between them, there is a clock trigger. In the function table, next state $Q_{A,\,n+1}$ takes the value $Q_{B,n}$ at clock-trigger which means whatever was the value of Q_B at n-th state becomes the value of Q_A at $(n + 1)$-th state.

To understand how this universal shift register is implemented, refer to logic circuit diagram of 74194 in Fig. 9.19. You may identify four 4 to 1 multiplexer blocks in the circuit (one is shown with dotted lines). Two selection inputs of each of these four multiplexers, understandably, are mode selection inputs S_1S_0. For $S_1S_0 = 00$, the second AND gate output which is nothing but the previous value of the corresponding flip-flop is transferred to the output. Thus, the flip-flop output does not change and this is the 'Hold' mode. For $S_1S_0 = 01$, the fourth AND gate output is transferred which corresponds to 'Shift right'. For $S_1S_0 = 10$, the first AND gate output is transferred which corresponds to 'Shift left'. Finally, for $S_1S_0 = 11$, the third AND gate output is selected which effects parallel 'Load' synchronized with clock. The input 'Clear' is active low and resets all the flip-flops asynchronously when activated. Note that the 'Clock' is positive edge-triggered due to two inversions (bubble) in the circuit diagram.

The 74299 is an 8-bit universal shift register in 20 pin package with a similar function table as the 74194. To save number of pins, the input and output pins are made common here. This is achieved by tristating and using additional control input that make these pins bidirectional.

Table 9.1 Function Table of 74194

Mode	*Control*	*Function*	*Next State (n+1-th state)*			
S_1	S_0		$Q_{A,n+1}$	$Q_{B,n+1}$	$Q_{C,n+1}$	$Q_{D,n+1}$
0	0	Hold	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$
0	1	Shift right	Data in (Pin 2)	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$
1	0	Shift left	$Q_{B,n}$ (Pin 7)	$Q_{C,n}$	$Q_{D,n}$	Data in
1	1	Load	A	B	C	D

9.7 APPLICATIONS OF SHIFT REGISTERS

Shift registers are used in almost every sphere of a digital logic system. In this section we discuss few such applications. Shift register can be used to count number of pulses entering into a system as ring counter or switched-tail counter. As ring counter it can generate various control signals in a sequential manner. Shift register can also generate a prescribed sequence repetitively or detect a particular sequence from data input. It can also help in reduction of hardware by converting parallel data feed to serial one. Serial adder is one such application discussed in this section.

Ring Counter

Let's begin with a simple serial shift register such as the 54/74164. One of the most logical applications of feedback might be to connect the output of the last flip-flop Q_H back to the D input of the first flip-flop A (Fig. 9.20a). Notice that the A and B data inputs are connected together. Now, suppose that all flip-flops are reset and the clock is allowed to run. What will happen? The answer is, nothing will happen since the D input to the first flip-flop is low (the input at A and B). Therefore, every time the clock goes high, the zero

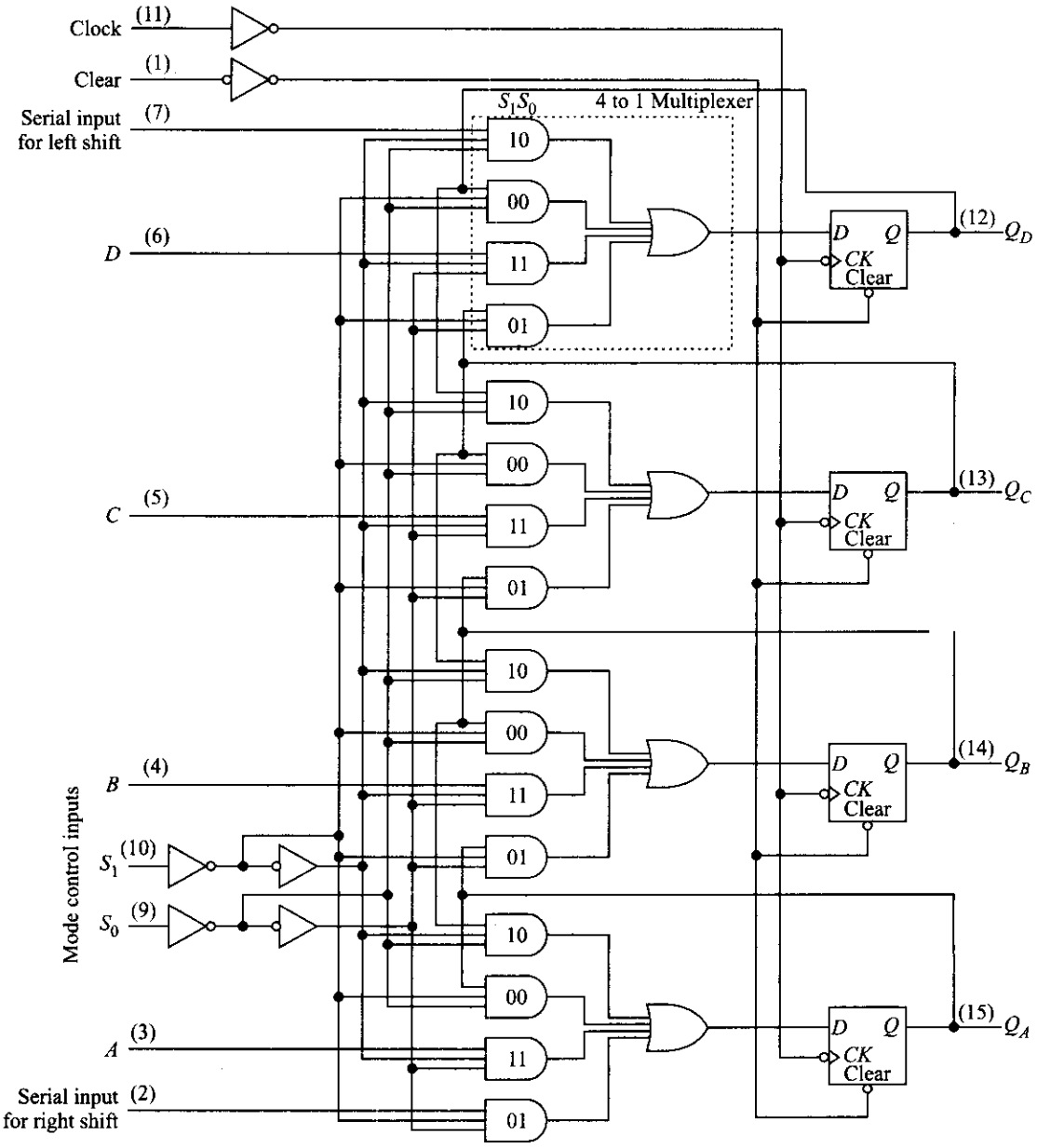
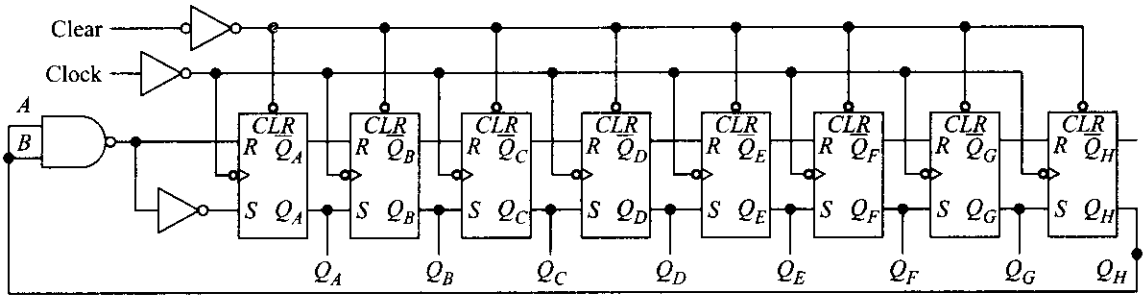
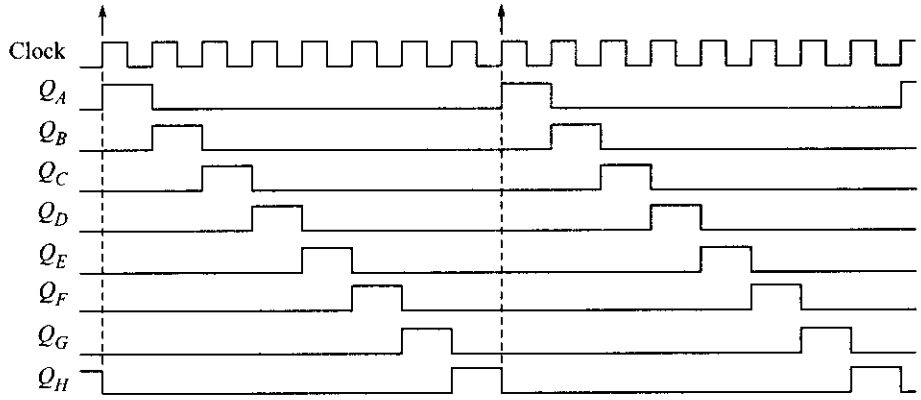


Fig. 9.19 **74194, 4-bit universal shift register**

in each flip-flop will be shifted into the next flip-flop, while the zero in the last flip-flop *H* will travel around the feedback loop and shift into the first flip-flop *A*. In other words, all the flip-flops are in a reset state, each clock PT resets them again, and each flip-flop output simply remains low. Consider the register as a tube full of zeros (ping-pong balls) that shift round and round the register, moving ahead one flip-flop with each clock PT.



(a) 54/74164 8-bit shift register with feedback line from Q_H to A-B



(b) Waveforms when register has a single one, and seven zeros

Fig. 9.20 **Ring counter**

In an effort to obtain some action, suppose that Q_A is high and all other flip-flops are low, and then allow the clock to run. On the very first clock PT, the 1 in A will shift into B and A will be reset, since the 0 in H will shift into A. All other flip-flops will still contain 0s. The second clock pulse will shift the 1 from B to C, while B resets. The third clock PT will shift the 1 from C to D, and so on. Thus this single 1 will shift down the register, traveling from one flip-flop to the next flip-flop each time the clock goes high. When it reaches flip-flop H, the next clock PT will shift it into flip-flop A by means of the feedback connection. Again, consider the register as a tube full of ping-pong balls, seven "white" ones (0s) and one "black" one (a 1). The ping-pong balls simply circulate around the register in a clockwise direction, moving ahead one flip-flop with each clock PT. This configuration is frequently referred to as a *circulating register* or a *ring counter*. The waveforms present in this ring counter are given in Fig. 9.20b.

Waveforms of this type are frequently used in the control section of a digital system. They are ideal for controlling events that must occur in a strict time sequence—that is, event A, then event B, then C, and so on. For instance, the logic diagram in Fig. 9.21 shows how to generate RESET, READ, COMPLEMENT, and WRITE (a fictitious set of control signals) as a set of control pulses that occur one after the other sequentially. The control signals are simply the outputs of flip-flips A, B, D, and E as shown in Fig. 9.20.

There is, however, a problem with such ring counters. In order to produce the waveforms shown in Fig. 9.20, the counter should have one, and only one, 1 in it. The chances of this occurring naturally when power is first applied are very remote indeed. If the flip-flops should all happen to be in the reset state when power

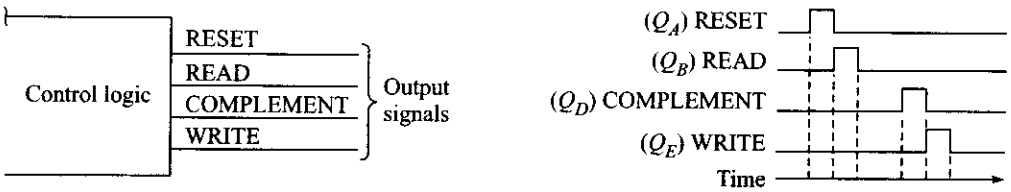
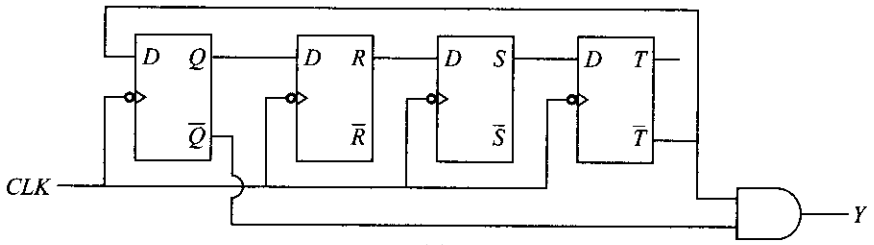


Fig. 9.21

is first applied, it will not work at all, as we saw previously. On the other hand, if some of the flip-flops come up in the set state while the remainder come up in the reset state, a series of complex waveforms of some kind will be the result. Therefore, it is necessary to preset the counter to the desired state before it can be used. Example 9.10 shows one scheme how to do presetting when power is first applied.

Switched-Tail Counter or Johnson Counter

We have seen in ring counter what happens if non-inverting output of the first flip-flop is fed back to first flip-flop of the shift register. If we instead feed inverting output back (or switch the tail) as shown in Fig. 9.22a for a 4-bit shift register we get *switched tail counter*, also known as *twisted tail counter* or *Johnson counter*. The



(a)

Clock	Serial in = T	Q	R	S	T	$Y = Q'T'$
0	1	0	0	0	0	1
1	1	1	0	0	0	0
2	1	1	1	0	0	0
3	1	1	1	1	0	0
4	0	1	1	1	1	0
5	0	0	1	1	1	0
6	0	0	0	1	1	0
7	0	0	0	0	1	0
8	1	0	0	0	0	1
9	1	1	0	0	0	0 repeats

(b)

Fig. 9.22 **(a) 4-bit switched tail counter, (b) Its state table**

circuit is explained through state table similar to Fig. 9.3 of Section 9.2. Assume all the flip-flops are cleared in the beginning. Then all the flip-flop inputs have 0 except the first one, *serial data* in which is complement of the last flip-flop, i.e. 1. When clock trigger occurs flip-flop stores *QRST* as 1000. This makes 1100 at the input of *QRST* when the next clock trigger comes and that gets transferred to output at NT. Proceeding this we complete state table of Fig. 9.22b. Note that output $Y = Q'T'$ and state of the circuit repeats every eighth clock cycle. Thus this 4-bit shift register circuit can count 8 clock pulses or called modulo-8 counter.

Following above logic and preparing state table for any N-bit shift register we see switched-tail configuration can count up to $2N$ number of clock pulse and gives modulo-$2N$ counter. The output Y, derived similarly by AND operation of first and last flip-flop inverting outputs gives a logic high at every $2N$-th clock cycle. This two-input AND gate which decodes states repeating in the memory units to generate output that signals counting of a given number of clock pulses is called decoding gate. For switched-tail counter of any modulo number we need only a 2-input AND gate. Observing the state sequences in Fig. 9.22b we find logic relation like $Y = QR'$ or $Y = RS'$ or $Y = ST'$, etc. can also be used for decoding purpose as they generate $Y = 1$ only once during $2N$ clock cycles. Note that for ring counter we don't need any decoding gate and clock pulse count can directly be obtained from any one flip-flop output. We shall discuss other counter design techniques in Chapter 10, which require less number of flip-flops for a particular modulo number. But, there decoding complexity increases with increasing number of flip-flops. For example, a modulo-8 counter is possible to design with $\log_2 8 = 3$ number of flip-flops but we need a 3 input AND gate to decode the counter. Similarly, modulo-16 counter requires 4 flip-flops and 4 input AND gate for decoding.

There is another important issue related with ring counter and switched tail counter. An n-bit register has 2^n different combination of states. But, the counter is to be initialized with one of the valid state of the counting sequence on which the design is based. Otherwise, the counter will follow a completely different state sequence (mutually exclusive) and decoding will not be proper. Solve Problem 9.25 to get an idea on what happens if circuit in Fig. 9.22a is initialized with a word outside the state sequence appearing in Fig. 9.22b.

Sequence Generator and Sequence Detector

Sequence generator is useful in generating a sequence pattern repetitively. It may be the synchronizing bit pattern sent by a digital data transmitter or it may be a control word directing repetitive control task. Sequence detector checks binary data stream and generates a signal when a particular sequence is detected.

Figure 9.23a gives the basic block diagram of a sequence generator where shift register is presented as pipe full of data and each flip-flop represents one compartment of it. The leftmost flip-flop is connected to serial data in and rightmost provides serial data out. The clock is implied and data transfer takes place only when a clock trigger arrives. Note that the shift register is connected like a ring counter and with triggering of clock the binary word stored in the clock comes out sequentially from serial out but does not get lost as it is fed back as serial in to fill the register all over again. Sequence generated for binary word 1011 is shown in the figure and for any n-bit long sequence to be generated for this configuration we need to store the sequence in an n-bit shift register.

The circuit that can detect a 4-bit binary sequence is shown in Fig. 9.23b. It has one register to store the binary word we want to detect from the data stream. Input data stream enters a shift register as serial data in and leaves as serial out. At every clocking instant, bit-wise comparisons of these two registers are done through Ex-NOR gate as shown in the figure. Two input Ex-NOR gives logic high when both inputs are low or both of them are high, i.e. when both are equal. The final output is taken from a four input AND gate, which becomes 1 only when all its inputs are 1, i.e. all the bits are matched. Figure 9.23b shows a situation

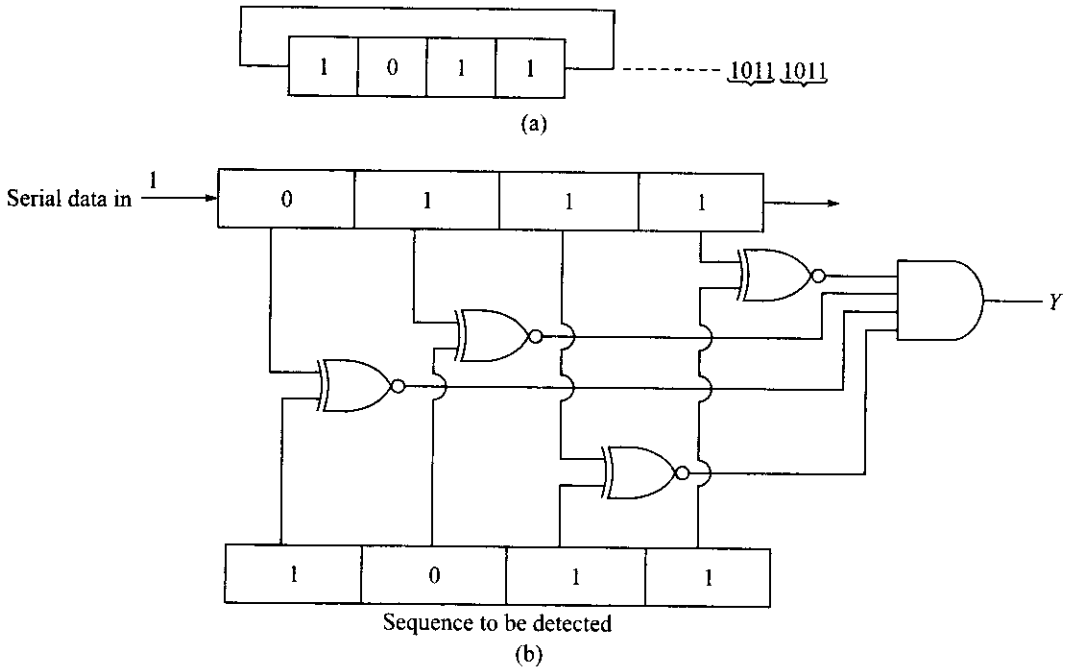


Fig. 9.23 **(a) 4-bit sequence generator, (b) 4-bit programmable sequence detector**

when data received so far is 0111 and word to be matched is 1011. The first two bits are mismatched and corresponding Ex-NOR outputs are low, so also final output Y. Now, as the next bit in the serial data stream is 1 when a clock trigger comes the first flip-flop of the shift-register stores 1 and 011 gets shifted to 2nd to 3rd flip-flops. With this both registers store 1011 and the first flip-flop of the shift-register stores 1 and 011 gets shifted to 2nd to 3rd flip-flops and $Y = 1$ completing sequence detection.

Note that Fig. 9.23b can be used as a *programmable sequence detector*, i.e. if we want to change the binary word to be detected we simply load that in the bottom register. For a fixed sequence detector, we can reduce hardware cost by removing bottom register and directly connect Ex-NOR input to $+V_{CC}$ or GND depending on whether we need a 1 or a 0 to be detected in a particular position.

Serial Adder

The addition operation and full adder (FA) circuit is discussed in detail in Chapter 6. We have seen for 8-bit addition we need 8 FA units (Fig. 6.6). There the addition is done in parallel. Using shift register we can convert this parallel addition to serial one and reduce number of FA units to only one. The benefit of this technique is more pronounced if the hardware unit that's needed to be used in parallel is very costly. Figure 9.24 shows how serial addition takes place in a time-multiplexed manner and also provides a snapshot of the register values at 3rd clock cycle.

Two 8-bit numbers, to be added ($A_7A_6 \ldots A_1A_0$ and $B_7B_6 \ldots B_1B_0$) are loaded in two 8-bit shift registers A and B. The LSB of each number appears in the rightmost position in two registers. Serial data out of A and B are fed to data inputs of full adder. The carry-in is fed from its own carry output delayed by one clock period

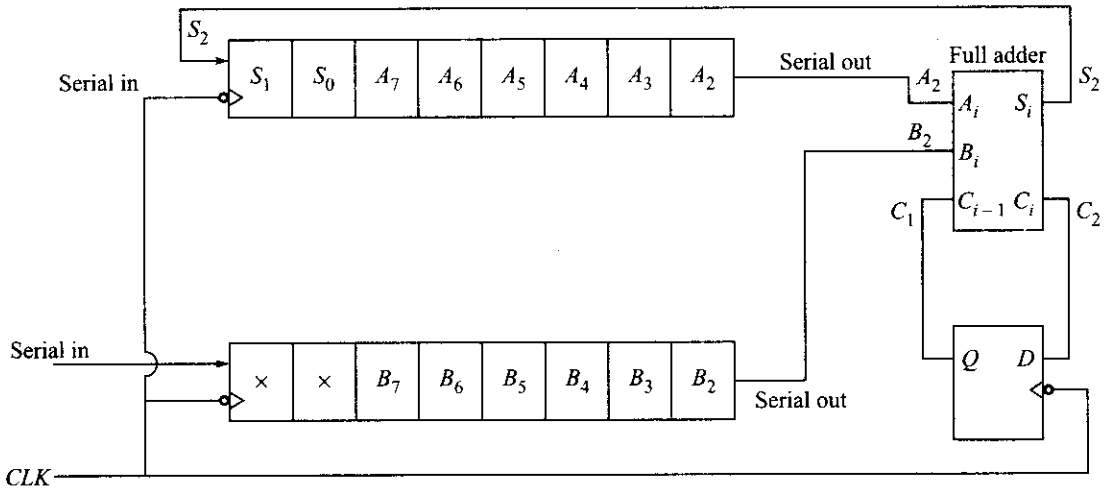


Fig. 9.24 **Serial addition of two 8-bit numbers (Register values shown are at 3rd clock cycle)**

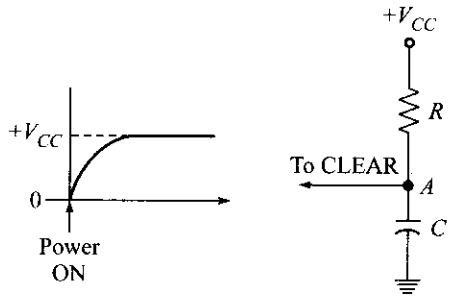
by a *D* flip-flop, which is initially cleared. Both registers and *D* flip-flop are triggered by same clock. The sum (*S*) output of FA is fed to serial data in of Shift Register *A*.

The serial addition takes place like this. The LSBs of two numbers (A_0 and B_0) appearing at serial out of respective registers are added by FA during 1st clock cycle and generate sum (S_0) and carry (C_0). S_0 is available at serial data input of register *A* and C_0 at input of *D* flip-flop. At NT of clock shift registers shift its content to right by one unit. S_0 becomes MSB of *A* and C_0 appears at *D* flip-flop output. Therefore in the second clock cycle FA is fed by second bit (A_1 and B_1) of two numbers and previous carry (C_0). In second clock cycle, S_1 and C_1 are generated and made available at serial data in of *A* register and input of *D* flip-flop respectively. At NT of clock S_1 becomes MSB of *A* and S_0 occupies next position. A_2 and B_2 now appear at FA data input and carry input is C_1. In 3rd clock cycle, S_2 and C_2 are generated and they get transferred similarly to register and flip-flop. This process goes on and is stopped by inhibiting the clock after 8 clock cycles. At that time shift register *A* stores the sum bits, S_7 in leftmost (MSB) position and S_0 in rightmost (LSB) position. The final carry is available at *D* flip-flop output.

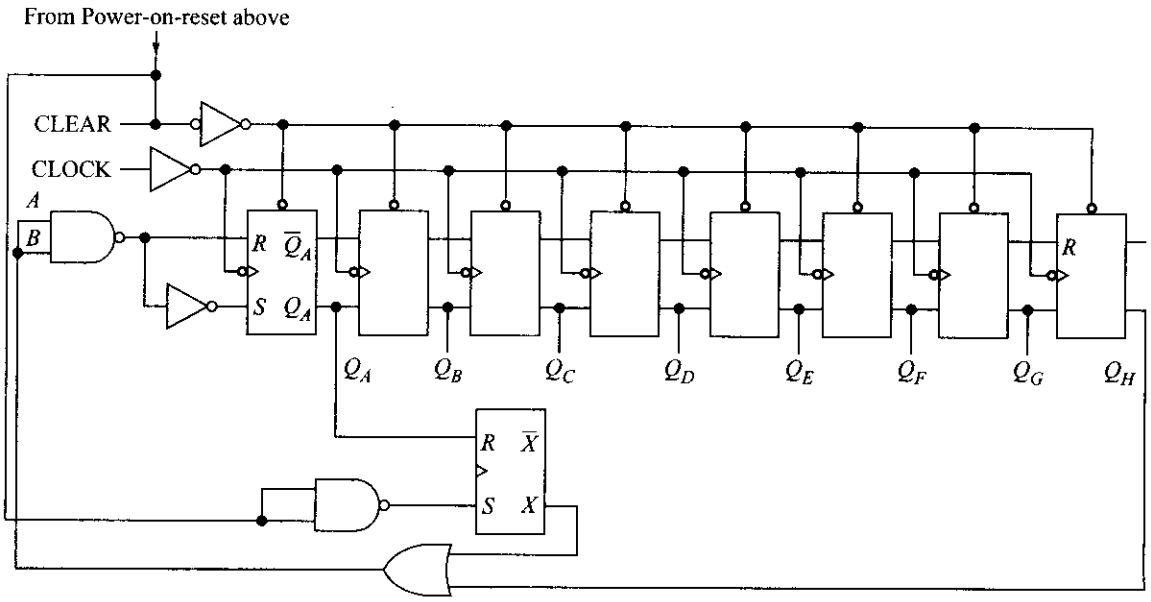
The limitation of this scheme is that the final addition result is delayed by eight clock cycles. In parallel adder the result is obtained almost instantaneously, after nanosecond order propagation delay of combinatorial circuit. However, using a high frequency clock the delay factor can be reduced considerably.

Example 9.10 The register in Fig. 9.20 can easily be cleared to all 0s by using the clear input. Show one method for setting a single 1 and the remaining 0s in the register.

Solution The simple power-on-reset circuit in Fig. 9.25a on the next page is widely used to generate the equivalent of a narrow negative pulse that occurs when power ($+V_{CC}$) is first applied to the system. Before the application of power, the voltage across the capacitor is zero. When $+V_{CC}$ is applied, the capacitor voltage charges toward $+V_{CC}$ with an *RC* time constant, and then remains at $+V_{CC}$ as long as the system power remains, as seen by the waveform in the figure. If point *A* is then connected to the clear input of the 54/74164, all flip-flops will automatically be reset to 0s when $+V_{CC}$ is first applied.



(a) Power-on-reset circuit



(b)

Fig. 9.25 **54/74164 with logic to preset a single 1 and seven 0s**

The logic added in the feedback path in Fig. 9.25b will now cause a single I to be set into the register. Here's how it works:

1. The power-on-reset pulse is inverted and used to initially set flip-flop X. This causes the output of the OR gate to be a 1, and the first clock PT will shift this 1 into Q_A.
2. When Q_A goes high, this will reset flip-flop X. At this point, the register contains a 1 in Q_A, and 0's in all other flip-flops. X will remain low as long as power is applied, and the data from Q_H will pass through the OR gate directly to the data input AB. The single 1 and the seven 0s will now shift around the register, advancing one position with each clock transition as desired.

Since the ring counter in Fig. 9.20 can function with more than a single 1 in it, it might be desirable to operate in this fashion at some time or other. It can, for example, be used to generate a more complex control waveform. Suppose, for instance, that the waveform shown in Fig. 9.26 were needed. This waveform

could easily be generated by simply presetting the counter in Fig. 9.20 with a 1 in A, a 1 in C, and all the other flip-flops reset. Notice that it is really immaterial where the two 1s are set initially. It is necessary only to ensure that they are spaced one flip-flop apart.

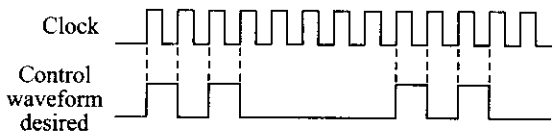


Fig. 9.26

Example 9.11 How would you preset the ring counter in Fig. 9.20 to obtain a square-wave output which is one-half the frequency of the clock? How about one-fourth the clock frequency?

Solution It is necessary only to preset a 1 in every other flip-flop, while the remaining flip-flops are all reset. This will generate a waveform at each flip flop output that is high for one clock period and then low for one clock period. The period of the output waveform is then two clock periods; therefore, the frequency is one-half the clock frequency. An output signal at one-fourth the clock frequency can be generated by presetting the shift register with two 1s, then two 0s, then two 1s, and then two 0s.

SELF-TEST

9. What is a ring counter?
10. What is a power-on-reset circuit used for?
11. What is a switched tail counter?
12. How does a serial adder work?

9.8 REGISTER IMPLEMENTATION IN HDL

In this section, we see how to describe a register using HDL. The parallel in parallel out register, primarily used for storage purpose is described for IC 741174 (Fig. 9.15) in Verilog code in the first column. We use vector notation for convenience. When Clear is activated (active LOW) all 6 outputs of Q are reset.

In second column, we show code for shift right register shown in Fig. 9.5 where T is the final output and Q, R, S are internal outputs. Since they are outputs of always block they have to be defined as **reg** and not **wire**. Note that, we use a new assignment operator <= within **always** block which unlike = operator executes all associated statements concurrently. If we had used = instead of <=, the D input through sequential execution would have reached final output in one clock cycle (unlike 4 clock cycles required in 4-bit shift register), also all the flip-flops within the register will have same value that of serial data input. Often, use of = operator is called blocking mode operation and use of <= is called non blocking mode. In column 3, we show a 4-bit serial in parallel out right shift register where all the flip-flop outputs are available externally. We use vector notation for convenience wherever possible.

```
module Reg74174(D,Clock,
   Clear,Q);
input Clock, Clear;
input [5:0] D;
output [5:0] Q;
reg [5:0] Q;
```

```
module SR1(D,Clock,T);
input Clock,D;  // Use
output T; // Clear as in
reg  T; // LHS to
   initialize
reg Q,R,S; //internal
```

```
module SR2(D,Clock,Q);
input Clock,D; //Clear as
output [3:0] Q; //in 74174
reg  [3:0] Q;
    //to initialize
```

```
always @ (negedge Clock
   or negedge Clear)
if (~Clear) Q=6'b0;
  //Q stores 6 binary 0
else Q=D;
endmodule
```

```
always @ (negedge Clock)
  begin
    Q <= D;
    R <= Q;
    S <= R;
    T <= S;
  end
endmodule
```

```
always @ (negedge
    Clock)
  begin
    Q[0] <= D;
    Q[1] <= Q[0];
    Q[2] <= Q[1];
    Q[3] <= Q[2];
  end
endmodule
```

Example 9.12 Write Verilog code for switched tail counter shown in Fig. 9.24.

Solution The code is similar to Shift Register description given above in second column. The serial data input here is taken from inverse of final flip-flop output. Output is generated from decoding logic $Y = Q'T'$.

```
module STC(Clock,Clear,Y); //Switched Tail Counter
input Clock, Clear;
output  Y;
reg Q,R,S,T; //internal outputs of flip-flops
assign Y= (~Q)&(~T);
always @ (negedge Clock)
begin
  if (~Clear) Q=6'b0; //Q stores 6 binary 0
  else
    begin
    Q <= ~T;  //Tail is switched and connected to input
    R <= Q;
    S <= R;
    T <= S;
  end
endmodule
```

PROBLEM SOLVING WITH MULTIPLE METHODS

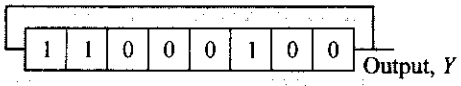
Problem Design an 8-bit sequence generator that generates the sequence 11000100 repetitively using shift register.

Solution We use ring counter and switched-tail counter derived from shift registers for this purpose.

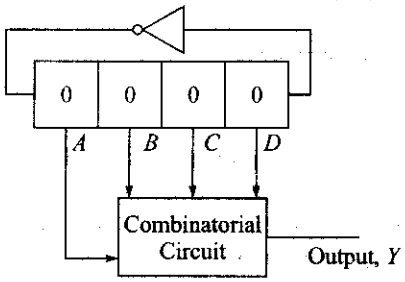
In Method-1, we load an 8-bit ring counter as shown in Fig. 9.27a with the given sequence and at the output, the sequence will be repetitively generated.

In Method-2, we consider a modulo-8 switched-tail counter developed from 4-bit shift register. Let it be initially loaded with 0000. Then the 8 repetitive states of the counter will be as shown in Fig.

9.27b and is reproduced in Fig. 9.27c. We then design a combinatorial circuit which for each of the state generates one bit of the sequence. The Karnaugh Map for this is shown in Fig. 9.27d. Note that the unused states can be considered as 'don't care'. The logic equation of the combinatorial circuit realized as Fig. 9.27b can be written as $Y = A'D' + A'B + AB'$



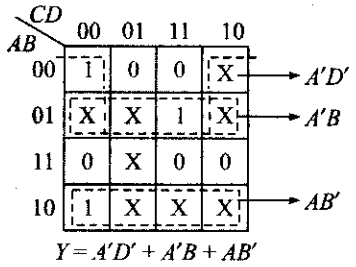
(a)



(b)

Counter A B C D	Sequence Generator Output, Y
0 0 0 0	1
1 0 0 0	1
1 1 0 0	0
1 1 1 0	0
1 1 1 1	0
0 1 1 1	1
0 0 1 1	0
0 0 0 1	0
0 0 0 0	...repeats...

(c)



(d)

Fig. 9.27 (a) Solution with Method-1, (b)-(d) Solution with Method-2, (b) Targeted realization, (c) Counter sequence vs. sequence generator output, (d) Karnaugh Map to generate logic equation

Note that Method-2 can be used with any other types of counter and is not restricted to shift register based counter. This is shown with Example 10.15 in next Chapter.

SUMMARY

Shift registers are important digital building blocks that can be used to store binary data. They can accept data bits in either a serial or a parallel format and can, likewise, deliver data in either serial or parallel. There are thus four basic register types: serial input–serial output, serial input–parallel output, parallel input–serial output, and parallel input–parallel output.

In one application, a register can be used to change data from a serial format into a parallel format, or vice versa. As such, shift registers can be regarded as *data format changers*. The UART is a good example of a data changer. There are a great many other shift register applications—arithmetic operations, logic operations,

and counters, to name only a few. Our intent has not been to discuss all the possible applications of shift registers, but rather to consider in detail how each type of register functions. With this knowledge, one can then discover the many and varied practical applications in existing digital designs.

GLOSSARY

- ***Johnson counter*** Refer to switched-tail counter.
- ***parallel shift*** Data bits are shifted simultaneously with a single clock transition.
- ***register capacity*** Determined by the number of flip-flops in the register. There must be one flip-flop for each binary bit; the register capacity is 2^n, where n is the number of flip-flops.
- ***ring counter*** A basic shift register with direct feedback such that the contents of the register simply circulate around the register when the clock is running.
- ***serial shift*** Data bits are shifted one after the other in a serial fashion with one bit shifted at each clock transition. Therefore, n clock transitions are needed to shift an n-bit binary number.
- ***sequence detector*** Detects a binary word from input data stream.
- ***sequence generator*** Generates a binary data sequence.
- ***serial adder*** Converts parallel data to serial and use adder block sequentially.
- ***switched tail counter*** Shift register with inverting output of last flip-flop fed to first flip-flop input. For n-bit shift register can give modulo $2N$ counter.
- ***shift register*** A group of flip-flops connected in such a way that a binary number can be shifted into or out of the flip-flops.
- ***UART*** Universal asynchronous receiver-transmitter.

PROBLEMS

Section 9.1

9.1 Determine the number of flip-flops needed to construct a shift register capable of storing:
   a. A 6-bit binary number
   b. Decimal numbers up to 32
   c. Hexadecimal numbers up to F

9.2 A shift register has eight flip-flops. What is the largest binary number that can be stored in it? Decimal number? Hexadecimal number?

9.3 Name the four basic types of shift registers, and draw a block diagram for each.

Section 9.2

9.4 Draw the waveforms to shift the binary number 1010 into the register in Fig. 9.2.

9.5 Draw the waveforms to shift the binary number 1001 into the register in Fig. 9.3.

9.6 The register in Fig. 9.2 has 0100 stored in it. Draw the waveforms for four clock transitions, assuming that both J and K are low.

9.7 Draw the waveforms showing how the decimal number 68 is shifted into the 54/74LS91 in Fig. 9.5. Show eight clock periods.

9.8 The hexadecimal number AB is stored in the 54/74LS91 in Fig. 9.5. Show the

waveforms at the output, assuming that the clock is allowed to run for eight cycles and that $A = B = 0$.

Section 9.3

9.9 How long will it take to shift an 8-bit binary number into the 54/74164 in Fig. 9.7 if the clock is:

a. 1 MHz b. 5 MHz

9.10 For the 54/74164 in Fig. 9.7, B is high, clear is high, a 1-MHz clock is used to shift the decimal number 200 into the register at A. Draw all the waveforms (such as in Fig. 9.9).

9.11 On the basis of information in Example 9.5, what is the maximum frequency of the clock if the minimum data transition time is 30 ns?

9.12 In Fig. 9.9, if control is taken low at time K, will the data stored in the register remain even if the clock is allowed to run? Explain.

Section 9.4

9.13 For the circuit in Fig. 9.11c, write the logic levels on each gate leg, given:

a. Control = 1, $X_1 = 0$, $X_2 = 1$

b. Control = 0, $X_1 = 0$, $X_2 = 1$

9.14 Redraw the 54/74166 in Fig. 9.10 showing only those gates used to shift data into the register in parallel. If a gate is disabled, don't draw it.

9.15 Redraw the 54/74166 in Fig. 9.10 showing only those gates used to shift data into the register in serial. If a gate is disabled, don't draw it.

9.16 Explain the operation of the 54/74166 for each of the six truth table entries in Fig. 9.12.

9.17 Draw all the input and output waveforms for the 54/74166 in Fig. 9.10, assuming that the decimal number 190 is shifted into the register in:

a. Parallel b. Serial

Section 9.5

9.18 Redraw the 54/7495A shift register in Fig. 9.15 showing only those gates used to shift data into the register in parallel. If a gate is disabled, don't draw it.

9.19 Repeat Prob. 9.18, assuming that the data is shifted in serially.

9.20 Draw the waveforms necessary to enter, and shift to the right a single 1 through the shift register in Fig. 9.15.

9.21 Repeat Prob. 9.20, but do a left shift. (See Fig. 9.16.)

Section 9.6

9.22 Draw the waveforms that would result if the circulating register (ring counter) in Fig. 9.20 had alternate 1s and 0s stored in it and a 1-MHz clock were applied.

9.23 The register in Fig. 9.20 can easily be cleared to all 0s by using the clear input. See if you can design logic circuitry to set the register with alternating 1s and 0s.

9.24 Explain the operation of the 54/74165 shift register. Redraw one of the eight flip flops along with its two NAND gates, and analyze:

a. Parallel data entry

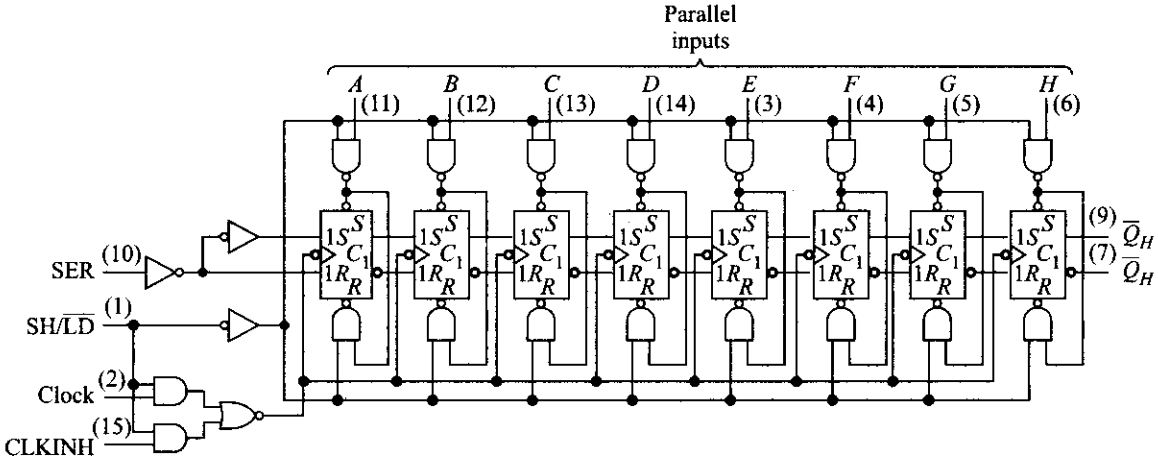
b. Shift right

c. Serial data entry

The logic diagram is given in Fig. 9.28 on the next page.

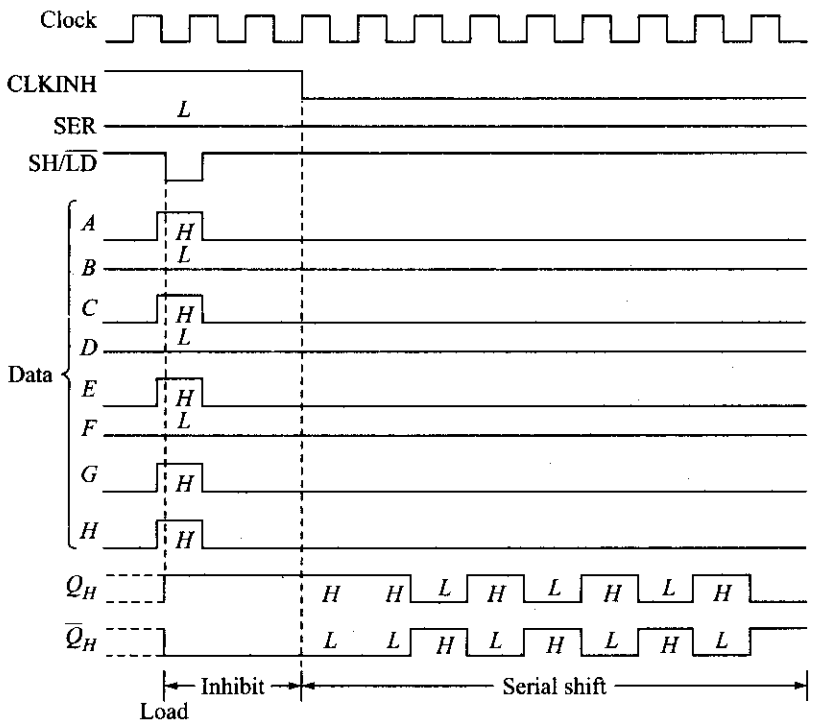
9.25 Show how modulo-8 switched tail counter works if is initialized with '1001'. How to decode this counter?

9.26 Show the circuit diagram for an 8-bit sequence detector which has to detect a fixed pattern '10011110' from incoming binary data stream.



Pin numbers shown are for *J* and *N* packages.

(a) Logic diagram (positive logic)



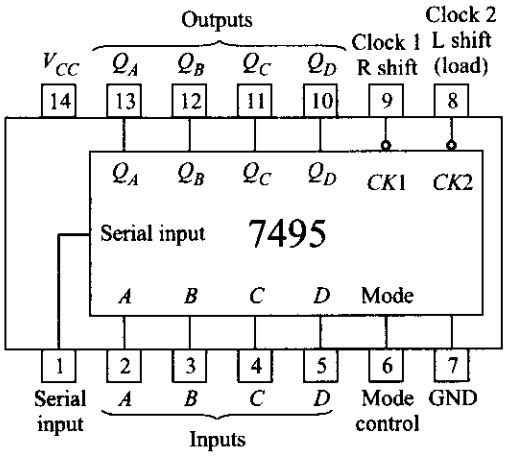
(b) Typical shift, load, and inhibit sequences

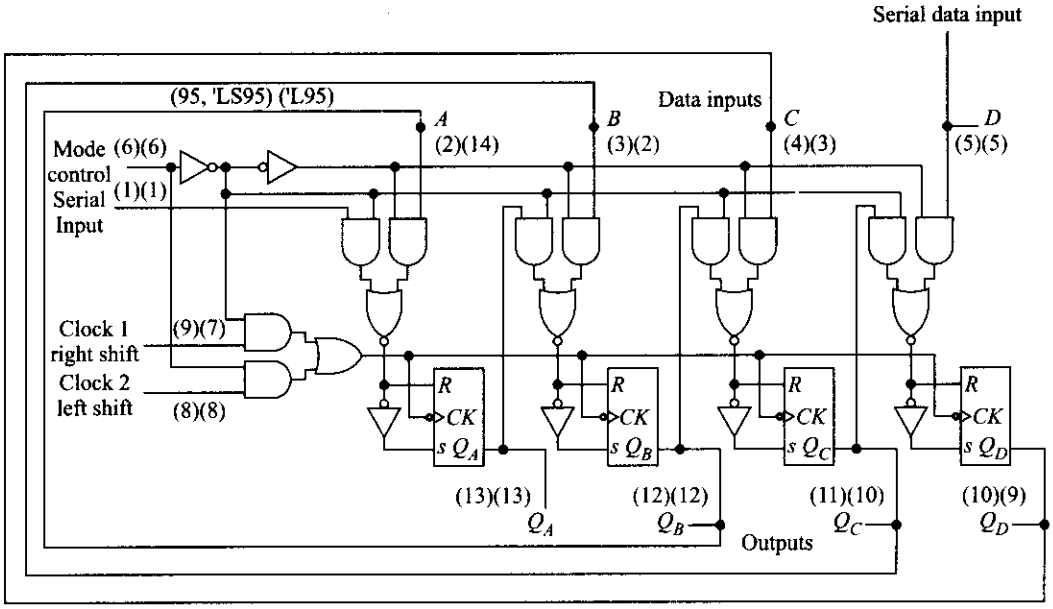
Fig. 9.28 **54/74165, 8-bit shift register**

LABORATORY EXPERIMENT

AIM: The aim of this experiment is to study Shift Register and use it to get Ring Counter and Johnson Counter.

Theory: The shift register is a special kind of register, i.e. group of memory units where binary data can be shifted from one unit to another in a sequential manner. The loading of shift register may be done serially or parallely. In serial loading as many number of clock cycles are required as the size of the register to load it fully. In parallel loading, all the memory units are loaded simultaneously in one clock cycle. The data within the register can be





made to shift in left, right or both directions. Feedback from uncomplemented final output to the serial input can make the shift register work as Ring Counter while feedback from complemented output allows it to work as Johnson Counter.

Apparatus: 5 VDC Power supply, Multimeter, Bread Board, Clock Generator, and Oscilloscope

Work element: IC 7495 is a 4-bit shift register with a mode control that allows parallel loading or right shift. Verify the truth table of this. The register can be made to work like a left shift register using parallel loading as shown in the figure. Verify the left shift operation. Use feedback to make it work like (i) Ring Counter and then (ii) Johnson Counter. Initialize appropriately and verify the counter output w.r.t. clock in a dual trace oscilloscope. Find how counting sequence varies on different initializations and comment on the modulo number of the counter.

Answers to Self-tests

1. 255
2. PTs
3. 30 ns
4. Eight; one
5. The shift/load line on the 74166 allows either serial or parallel data entry.
6. PTs
7. Two 7495As connected in series will store 8 bit numbers.
8. The 7495A has separate clock inputs to accommodate separate shift-right and shift-left signals.
9. A direct-feedback shift register—the contents of the register circulate around the register when the clock is running.
10. A power-on-reset circuit is used to preset flip-flops to any desired states.
11. Switched tail counter is a shift register with inverting output of last flip-flop fed to first flip-flop input. For n-bit shift register this configuration can give modulo $2N$ counter.
12. Serial adder converts parallel data to serial using shift register and performs addition sequentially using an adder block.

Counters

10

OBJECTIVES

- **Describe the basic construction and operation of an asynchronous counter**
- **Determine the logic circuit needed to decode a given state from the output of a given counter**
- **Describe the synchronous counter and its advantages**
- **See how the modulus of a counter can be reduced by skipping one or more of its natural counts**
- **Understand how to design counter as a finite state machine**

A counter is probably one of the most useful and versatile subsystems in a digital system. A counter driven by a clock can be used to count the number of clock cycles. Since the clock pulses occur at known intervals, the counter can be used as an instrument for measuring time and therefore period or frequency. There are basically two different types of counters—synchronous and asynchronous.

The ripple counter is simple and straightforward in operation and its construction usually requires a minimum of hardware. It does, however, have a speed limitation. Each flip-flop is triggered by the previous flip-flop, and thus the counter has a cumulative settling time. Counters such as these are called serial, or asynchronous.

An increase in speed of operation can be achieved by use of a parallel or synchronous counter. Here, every flip-flop is triggered by the clock (in synchronism), and thus settling time is simply equal to the delay time of a single flip-flop. The increase in speed is usually obtained at the price of increased hardware.

Serial and parallel counters are used in combination to compromise between speed of operation and hardware count. Serial, parallel, or combination counters can be designed such that each clock transition advances the contents of the counter by one; it is then operating in a count-up mode. The opposite is also

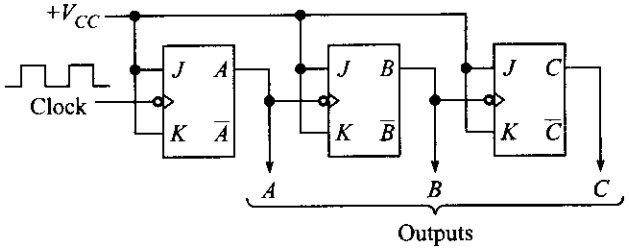
possible; the counter then operates in the count-down mode. Furthermore, many counters can be either "cleared" so that every flip-flop contains a zero, or preset such that the contents of the flip-flops represent any desired binary number.

Now, let's take a look at some of the techniques used to construct counters.

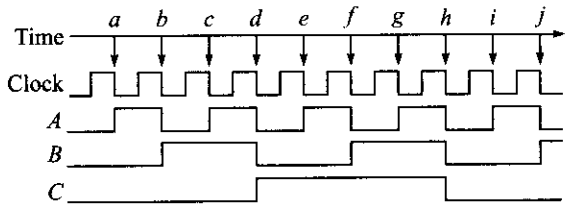
10.1 ASYNCHRONOUS COUNTERS

Ripple Counters

A binary ripple counter can be constructed using clocked *JK* flip-flops. Figure 10.1 shows three negative-edge-triggered, *JK* flip-flops connected in cascade. The system clock, a square wave, drives flip-flop *A*. The output of *A* drives *B*, and the output of *B* drives flip-flop *C*. All the *J* and *K* inputs are tied to $+V_{CC}$. This means that each flip-flop will change state (toggle) with a negative transition at its clock input.



(a) Three-bit binary ripple counter



(b) Waveforms

Negative clock transitions	C	B	A	State or count
---	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0

(c) Truth table

Fig. 10.1

When the output of a flip-flop is used as the clock input for the next flip-flop, we call the counter a *ripple counter*, or *asynchronous counter*. The *A* flip-flop must change state before it can trigger the *B* flip-flop, and the *B* flip-flop has to change state before it can trigger the *C* flip-flop. The triggers move through the flip-flops like a ripple in water. Because of this, the overall propagation delay time is the sum of the individual delays. For instance, if each flip-flop in this three-flip-flop counter has a propagation delay time of 10 ns, the overall propagation delay time for the counter is 30 ns.

The waveforms given in Fig. 10.1b show the action of the counter as the clock runs. Let's assume that the flip-flops are all initially reset to produce 0 outputs. If we consider *A* to be the least-significant bit (LSB) and *C* the most-significant bit (MSB), we can say the contents of the counter is *CBA* = 000.

Every time there is a clock NT, flip-flop *A* will change state. This is indicated by the small arrows (↓) on the time line. Thus at point *a* on the time line, *A* goes high, at point *b* it goes back low, at *c* it goes back high, and so on. Notice that the waveform at the output of flip-flop *A* is one-half the clock frequency.

Since A acts as the clock for B, each time the waveform at A goes low, flip-flop B will toggle. Thus at point b on the time line, B goes high; it then goes low at point d and toggles back high again at point f. Notice that the waveform at the output of flip-flop B is one-half the frequency of A and one-fourth the clock frequency.

Since B acts as the clock for C, each time the waveform at B goes low, flip-flop C will toggle. Thus C goes high at point d on the time line and goes back low again at point h. The frequency of the waveform at C is one-half that at B, but it is only one-eighth the clock frequency.

Example 10.1 What is the clock frequency in Fig. 10.1 if the period of the waveform at C is 24 μs?

Solution Since there are eight clock cycles in one cycle of C, the period of the clock must be 24/8 = 3 μs. The clock frequency must then be $1/(3 \times 10^{-6}) = 333$ kHz.

Notice that the output condition of the flip-flops is a binary number equivalent to the number of clock NTs that have occurred. Prior to point a on the time line the output condition is $CBA = 000$. At point a on the time line the output condition changes to $CBA = 001$, at point b it changes to $CBA = 010$, and so on. In fact, a careful examination of the waveforms will reveal that the counter content advances one count with each clock NT in a "straight binary progression" that is summarized in the truth table in Fig. 10.1c.

Because each output condition shown in the truth table is the binary equivalent of the number of clock NTs, the three cascaded flip-flops in Fig. 10.1 comprise a 3-bit binary ripple counter. This counter can be used to count the number of clock transitions up to a maximum of seven. The counter begins at count 000 and advances one count for each clock transition until it reaches count 111. At this point it resets back to 000 and begins the count cycle all over again. We can say that this ripple counter is operating in a count-up mode.

Since a binary ripple counter counts in a straight binary sequence, it is easy to see that a counter having n flip-flops will have 2^n output conditions. For instance, the three-flip-flop counter just discussed has $2^3 = 8$ output conditions (000 through 111). Five flip-flops would have $2^5 = 32$ output conditions (00000 through 11111), and so on. The largest binary number that can be represented by n cascaded flip-flops has a decimal equivalent of $2^n - 1$. For example, the three-flip-flop counter reaches a maximum decimal number of $2^3 - 1$. The maximum decimal number for five flip-flops is $2^5 - 1 = 31$, while six flip-flops have a maximum count of 63.

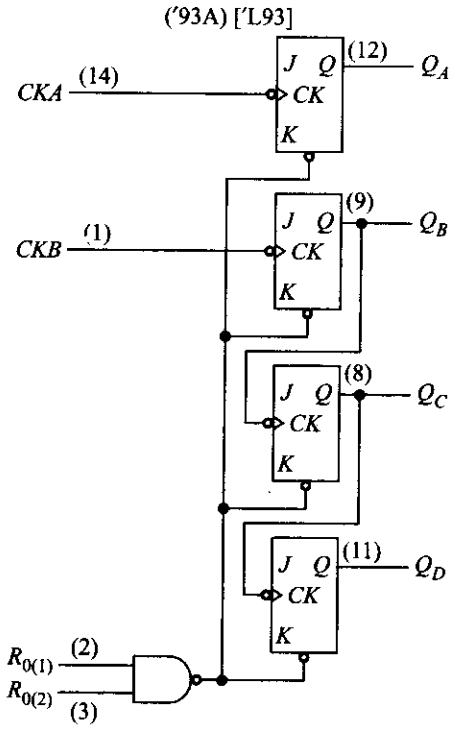
A three-flip-flop counter is often referred to as a modulus-8 (or mod-8) counter since it has eight states. Similarly, a four-flip-flop counter is a mod-16 counter, and a six-flip-flop counter is a mod-64 counter. The *modulus* of a counter is the total number of states through which the counter can progress.

Example 10.2 How many flip-flops are required to construct a mod-128 counter? A mod-32? What is the largest decimal number that can be stored in a mod-64 counter?

Solution A mod-128 counter must have seven flip-flops, since $2^7 = 128$. Five flip-flops are needed to construct a mod-32 counter. The largest decimal number that can be stored in a six-flip-flop counter (mod-64) is 111111 = 63. Note carefully the difference between the modulus (total number of states) and the maximum decimal number.

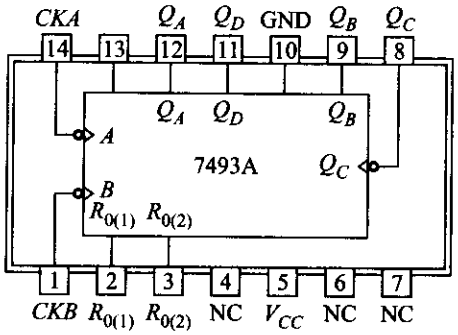
The 54/7493A

The logic diagram, DIP pinout, and truth table for a 54/7493A are given in Fig. 10.2. This TTL MSI circuit is a 4-bit binary counter that can be used in either a mod-8 or a mod-16 configuration. If the clock is applied at input CKB, the outputs will appear at Q_B, Q_C, and Q_D, and this is a mod-8 binary ripple counter exactly like that in Fig. 10.1. In this case, flip-flop Q_A is simply unused.



(a) Logic diagram

'L93 (top view)



Positive logic: see function tables
NC – No internal connection

(b) DIP pinout

'L93A, 'L93, 'LS93 Count sequence

Count	Output			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

(c) Truth table

Fig. 10.2 7493A

On the other hand, if the clock is applied at input *CKA* and flip-flop Q_A is connected to input *CKB*, we have a mod-16, 4-bit binary ripple counter. The outputs are Q_A, Q_B, Q_C, and Q_D. The proper truth table for this connection is given in Fig. 10.2c.

All the flip-flops in the 7493A have direct reset inputs that are active low. Thus a high level at both reset inputs of the NAND gate, $R_{0(1)}$ and $R_{0(2)}$, is needed to reset all flip-flops simultaneously. Notice that this reset operation will occur without regard to the clock.

Example 10.3 Draw the correct output waveforms for a 7493A connected as a mod-16 counter.

Solution The correct waveforms are shown in Fig. 10.3. The contents of the counter is 0000 at point *a* on the time line. With each negative clock transition, the counter is advanced by one until the counter contents are 1111 at point

b on the time line. At point *c*, the counter resets to 0000, and the counting sequence repeats. Clearly, this is a mod-16 counter, since there are 16 states (0000 through 1111), and the maximum decimal number that can be stored in the flip-flops is decimal 15 (1111).

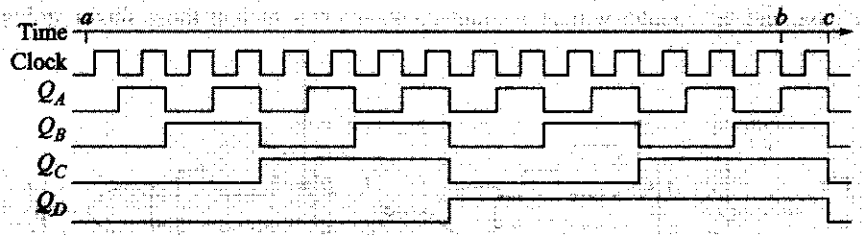
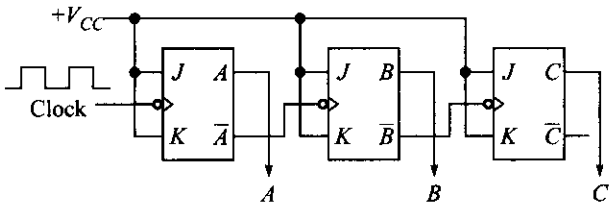
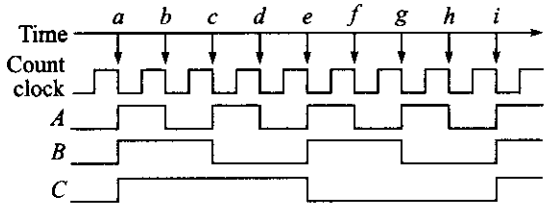


Fig. 10.3 **Waveforms for a mod-16, 7493A**

An interesting and useful variation of the 3-bit ripple counter in Fig. 10.1 is shown in Fig. 10.4. The system clock is still used at the clock input to flip-flop *A*, but the complement of *A*, $\overline{A}$, is used to drive flip-flop *B*, likewise; $\overline{B}$ is used to drive flip-flop *C*. Take a look at the resulting waveforms.



(a)



(b)

Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

(c)

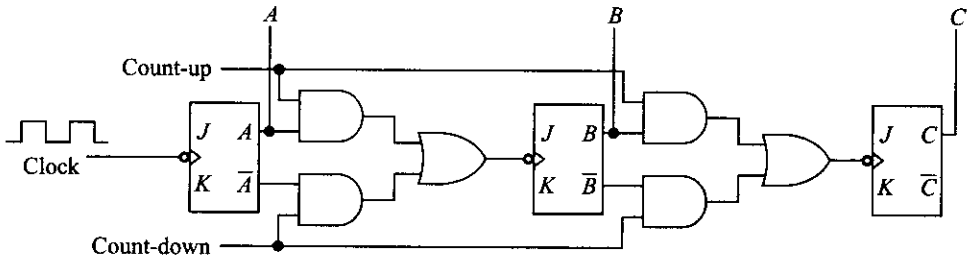
Fig. 10.4 **A down counter**

Flip-flop *A* simply toggles with each negative clock transition as before. But flip-flop *B* will toggle each time *A goes high*! Notice that each time *A* goes high, $\overline{A}$ goes low, and it is this negative transition on *A* that triggers *B*. On the time line, *B* toggles at points *a*, *c*, *e*, *g* and *i*.

Similarly, flip-flop *C* is triggered by $\overline{B}$ and so *C* will toggle each time *B* goes high. Thus *C* toggles high at point *a* on the time line, toggles back low at point *e* and goes back high again at point *i*.

The counter contents become *ABC* = 111 at point *a* on the time line, change to 110 at point *b*, and change to 101 at point *c*. Notice that the counter contents are reduced by one count with each clock transition! In other words, the counter is operating in a count-down mode. The results are summarized in the truth table in Fig. 10.4c. This is still a mod-8 counter, since it has eight discrete states, but it is connected as a *down counter*.

A 3-bit asynchronous *up-down counter* that counts in a straight binary sequence is shown in Fig. 10.5. It is simply a combination of the two counters discussed previously. For this counter to progress through a count-up sequence, it is necessary to trigger each flip-flop with the true side of the previous flip-flop (as opposed to the complement side). If the count-down control line is low and the count-up control line high, this will be the case, and the counter will have count-up waveforms such as those shown in Fig. 10.1.



Note: The J and K inputs are all tied to $+V_{CC}$.
The counter outputs are A, B, and C.

Fig. 10.5 **3-bit binary up-down counter**

On the other hand, if count-down is high and count-up is low, each flip-flop will be triggered from the complement side of the previous flip-flop. The counter will then be in a count-down mode and will progress through the waveforms as shown in Fig. 10.4.

This process can be continued to other flip-flops down the line to form an up-down counter of larger moduli. It should be noticed, however, that the gates introduce additional delays that must be taken into account when determining the maximum rate at which the counter can operate.

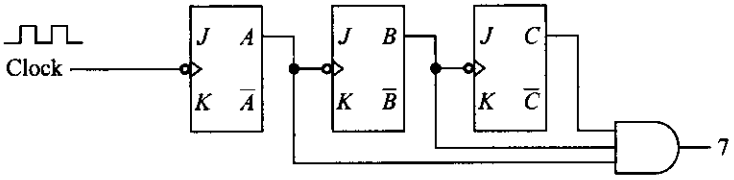
SELF-TEST

1. What is the largest binary number representable by a mod-6 ripple counter?
2. How many flip-flops are required to construct a mod-1024 ripple counter?

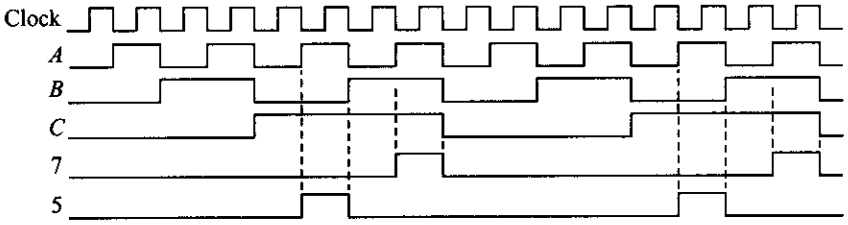
10.2 DECODING GATES

A decoding gate can be connected to the outputs of a counter in such a way that the output of the gate will be high (or low) only when the counter contents are equal to a given state. For instance, the decoding gate connected to the 3-bit ripple counter in Fig. 10.6a will decode state 7 (CBA = 111). Thus the gate output will be high only when A = 1, B = 1, and C = 1 and the waveform appearing at the output of the gate is labeled 7. The Boolean expression for this gate can be written 7 = CBA. A comparison with the truth table for this counter (in Fig. 10.1) will reveal that the condition CBA = 111 is true only for state 7.

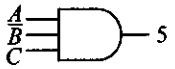
The other seven states of the counter can be decoded in a similar fashion. It is only necessary to examine the truth table for the counter and then the proper Boolean expression for each gate can be written. For instance, to decode state 5, the truth table reveals that CBA = 101 is the unique state. For the gate output to be high during this time, we must use C, $\overline{B}$, and A at the gate inputs. Notice carefully that if B = 0, then $\overline{B}$ = 1! The correct Boolean expression is then 5 = $C\overline{B}A$, and the desired gate is that given in Fig. 10.6c. The waveform is again that given in Fig. 10.6b and is labeled 5.



(a) Decoding gate for state 7



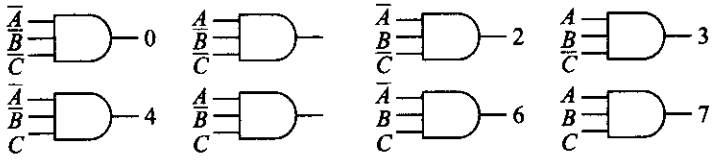
(b) Waveforms



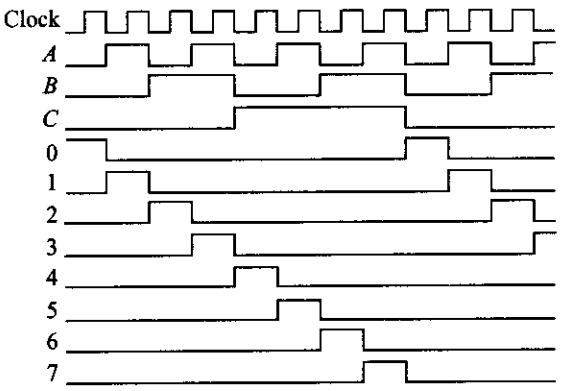
(c) Gate to decode state 5

Fig. 10.6

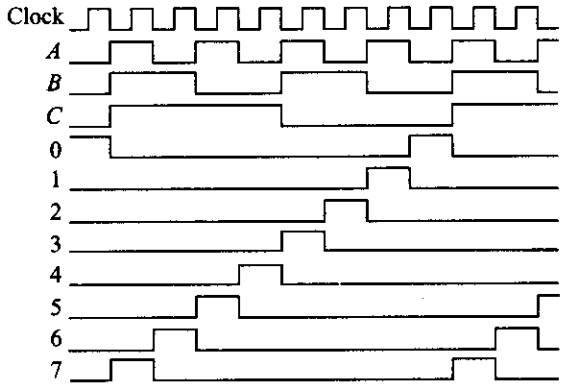
All eight gates necessary to decode the eight states of the 3-bit counter in Fig. 10.1 are shown in Fig. 10.7a. The gate outputs are shown in Fig. 10.7b. These decoded waveforms are a series of positive pulses that occur in a strict time sequence and are very useful as control signals throughout a digital system. If we consider state 0 as the first event, then state 1 will be the second, state 2 the third, and so on, up to state 7. Clearly the counter is counting upward in decimal notation from 0 to 7 and then beginning over again at 0.



(a) Gates



(b) Count-up mode



(c) Count-down mode

Fig. 10.7 Decoding gates for a 3-bit binary ripple counter

If these eight gates are connected to the up-down counter shown in Fig. 10.5, the decoded waveforms will appear exactly as shown in Fig. 10.7b, provided the counter is operating in the count-up mode. If the counter is operated in the count-down mode, the decoded waveforms will appear as in Fig. 10.7c. In this case, if state 0 is considered the first event, then state 7 is the second event, then state 6, and so on, down to state 1. Clearly the counter is counting downward in decimal notation from 7 to 0 and then beginning again at 7.

Example 10.4 Show how to use a 54LS11, triple 3-input AND gate to decode states 1, 4, and 6 of the counter in Fig. 10.5.

Solution The logic diagram and pinout for a 54LS11 is given in Fig. 10.8. The correct Boolean expressions for the desired states are $1 = \overline{C}\,\overline{B}A$, $4 = C\overline{B}\,\overline{A}$, and $6 = CB\overline{A}$. Wiring from the counter flip-flop outputs to the chip is given in Fig. 10.8.

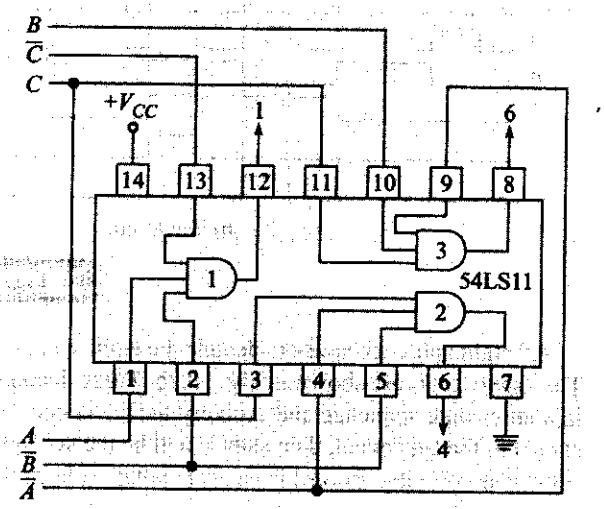


Fig. 10.8

Let's take a more careful look at the waveforms generated by the counter in Fig. 10.5 as it operates in the count-up mode. The clock and each flip-flop output are redrawn in Fig. 10.9, and the propagation delay time of each flip-flop is taken into account. Notice carefully that the clock is the trigger for flip-flop A, and the A waveform is thus delayed by t_p from the negative clock transition. For reference purposes, the complement of A, $\overline{A}$, is also shown. Naturally it is the exact mirror image of A.

Since A acts as the trigger for B, the B waveform is delayed by one flip-flop delay time from the negative transition of A. Similarly, the C waveform is delayed by t_p from each negative transition of B.

At first glance, these delay times would seem to offer no more serious problem than a speed limitation for the counter, but a closer examination reveals a much more serious problem. When the decoding gates in Fig. 10.7 are connected to this counter (or, indeed, when decoding gates are connected to any ripple counter), glitches may appear at the outputs of one or more of the gates. Consider, for instance, the gate used to decode state 6. The proper Boolean expression is $6 = CB\overline{A}$. So, in Fig. 10.9 the correct output waveform for this gate is high only when $C = 1$, $B = 1$, and $\overline{A} = 1$.

But look at the glitch that occurs when the counter progresses from state 7 to state 0. On the time line, A goes low ($\overline{A}$ goes high) at point a. Because of flip-flop delay time, however, B does not go low until point b on the time line! Thus between points a and b on the time line we have the condition $C = 1$, $B = 1$, and $A = 1$—therefore, the gate output is high, and we have a glitch! Look at the waveform $6 = CB\overline{A}$.

Depending on how the decoder gate outputs are used, the glitches (or unwanted pulses) may or may not be a problem. Admittedly the glitches are only a few nanoseconds wide and may even be very difficult to observe on an oscilloscope. But TTL is *very fast*, and TTL circuits will respond to even the smallest glitches—usually when you least expect it, and always at unwanted times! Therefore, you must beware to avoid this condition. There are at least two solutions to the glitch problem. One method involves strobing the gates; we discuss that technique here. A second method is to use synchronous counters; we consider that topic in the next section.

Consider using a 4-input AND gate to decode state 6 as shown in Fig. 10.9b, where the clock is now used as a strobe. An examination of the waveforms in this figure clearly reveals that the clock is low between points a and b on the time line. Since the clock must be high for the gate output to be high, the glitch cannot possibly occur! On the other

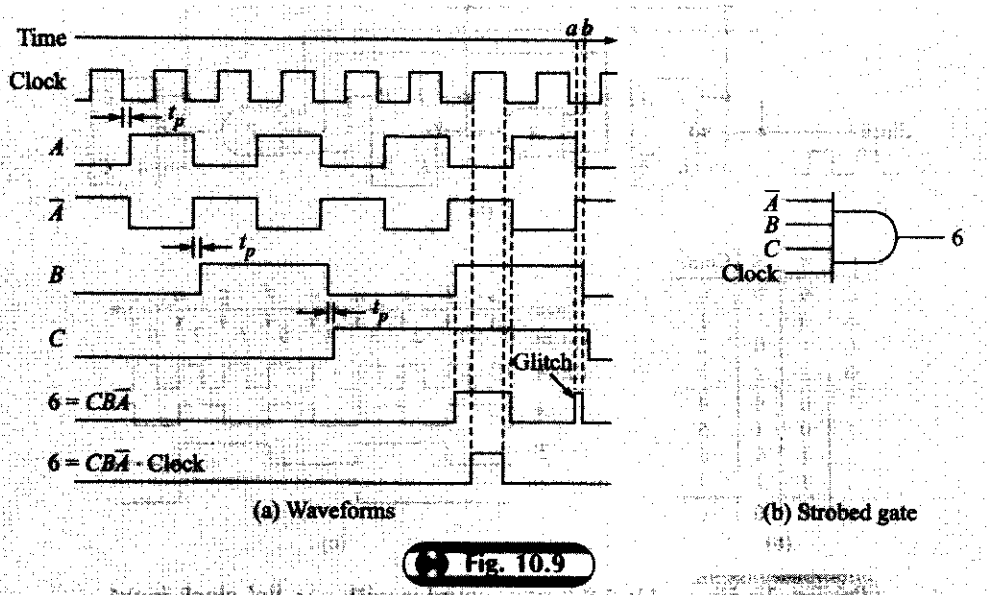


Fig. 10.9

hand, the clock is high when $C = 1$, $B = 1$, and $\overline{A} = 1$, and the waveform for 6 appears exactly as it should. Notice that the width of the positive pulse at 6 is exactly the width of the positive portion of the clock. Look at the waveform $6 = CB\overline{A}$ clock. This technique can be applied to the other seven decoding gates for this counter (or for any other counter), and the decoded output waveforms will be glitch-free.

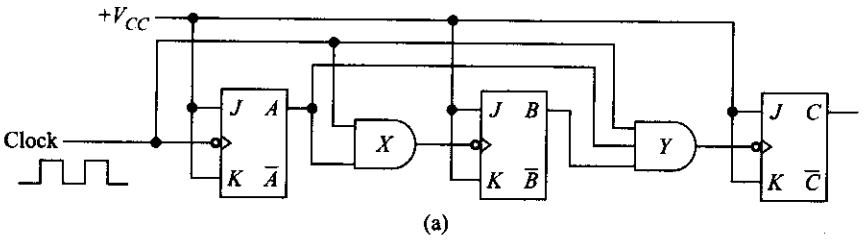
SELF-TEST

3. What is the Boolean expression for an AND gate needed to decode state 3 of the counter in Fig. 10.6?
4. What is the primary cause of glitches that sometimes occur at the output of a decoding gate used with a ripple counter? What is one method to eliminate these glitches?

10.3 SYNCHRONOUS COUNTERS

The ripple counter is the simplest to build, but there is a limit to its highest operating frequency. As previously discussed, each flip-flop has a delay time. In a ripple counter these delay times are additive, and the total "settling" time for the counter is approximately the delay time times the total number of flip-flops. Furthermore, there is the possibility of glitches occurring at the output of decoding gates used with a ripple counter. The first problem fully and the second problem, to some extent can be overcome by the use of a synchronous parallel counter. The main difference here is that every flip-flop is triggered in synchronism with the clock. Note that *strobing* as the solution to glitches has been discussed before in a separate subsection of Section 7.7 of Chapter 7.

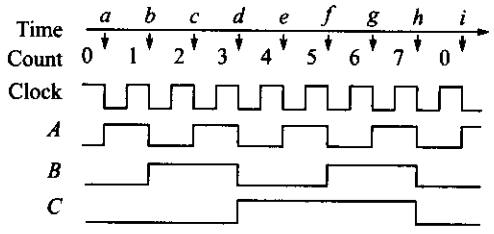
The construction of one type of parallel binary counter is shown in Fig. 10.10, along with the truth table and the waveforms for the natural count sequence. Since each state corresponds to an equivalent binary number (or count), we refer to each state as a count from now on. The basic idea here is to keep the J and K



(a)

C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0

(b)



(c)

Fig. 10.10 Mod-8 binary counter with parallel clock input

inputs of each flip-flop high, such that the flip-flop will toggle with any clock NT at its clock input. We then use AND gates to gate every second clock to flip-flop *B*, every fourth clock to flip-flop *C*, and so on. This logic configuration is often referred to as "steering logic" since the clock pulses are gated or steered to each individual flip-flop.

The clock is applied directly to flip-flop *A*. Since the *JK* flip-flop used responds to a negative transition at the clock input and toggles when both the *J* and *K* inputs are high, flip-flop *A* will change state with each clock NT.

Whenever *A* is high, AND gate *X* is enabled and a clock pulse is passed through the gate to the clock input of flip-flop *B*. Thus *B* changes state with every other clock NT at points *b*, *d*, *f*, and *h* on the time line. Since, there is an additional AND gate delay for the clock at *B* flip-flop in comparison to *A* flip-flop, it is not a parallel counter in a strict sense of the term.

Since AND gate *Y* is enabled and will transmit the clock to flip-flop *C* only when both *A* and *B* are high, flip-flop *C* changes state with every fourth clock NT at points *d* and *h* on the time line.

Examination of the waveforms and the truth table reveals that this counter progresses upward in a natural binary sequence from count 000 up to count 111, advancing one count with each clock NT. This is a mod-8 parallel or synchronous binary counter operating in the count-up mode.

Let's see if this counter configuration has cured the glitch problem discussed previously. The waveforms for this counter are expanded and redrawn in Fig. 10.11, and we have accounted for the individual flip-flop propagation times. Study these waveforms carefully and note the following:

1. The clock NT is the mechanism that toggles each flip-flop.
2. Therefore, whenever a flip-flop changes state, it toggles at exactly the same time as all the other flip-flops—in other words, all the flip-flops change states in synchronism!
3. As a result of the synchronous changes of state, it is not possible to produce a glitch at the output of a decoding gate, such as the gate for 6 shown in Fig. 10.11. Therefore, the decoding gates need not be strobed. All the decoding gates in Fig. 10.7 can be used with this counter without fear of glitches!

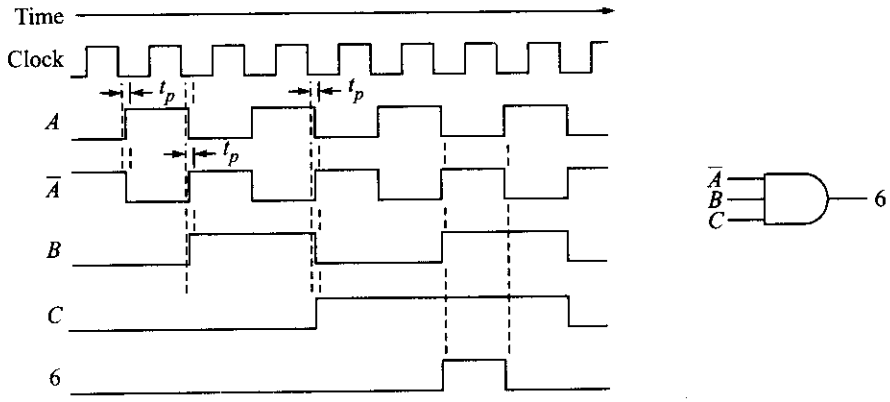
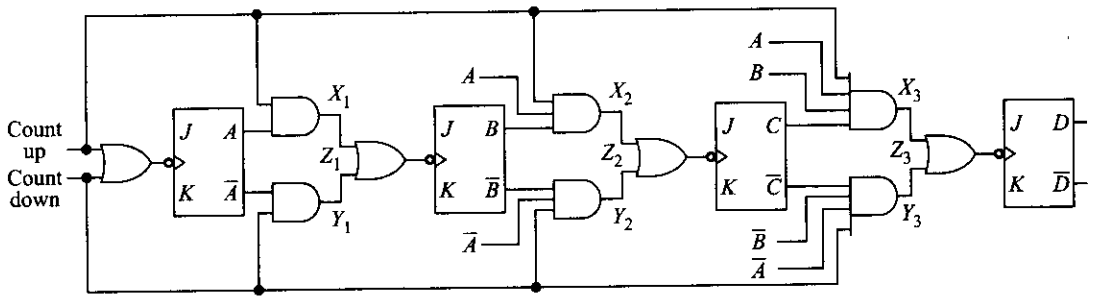


Fig. 10.11

You should take time to compare these waveforms with those generated by the ripple counter as shown in Fig. 10.9.

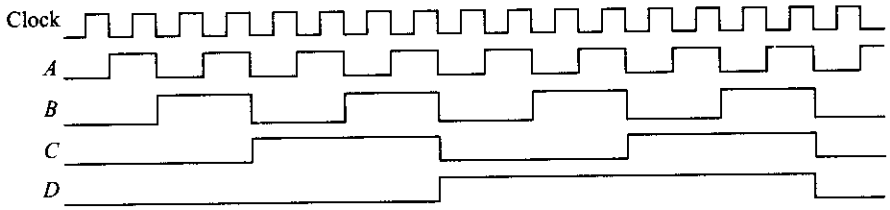
A parallel up-down counter can be constructed in a fashion similar to that shown in Fig. 10.12. In any parallel counter, the time at which any flip-flop changes state is determined by the states of all previous flip-flops in the counter. In the count-up mode, a flip-flop must toggle every time all previous flip-flops are in a 1 state, and the clock makes a transition. In the count-down mode, flip-flop toggles must occur when all prior flip-flops are in a 0 state.

The counter in Fig. 10.12 is a synchronous 4-bit up-down counter. To operate in the count-up mode, the system clock is applied at the count-up input, while the count-down input is held low. To operate in the count-down mode, the system clock is applied at the count-down input while holding the count-up input low.

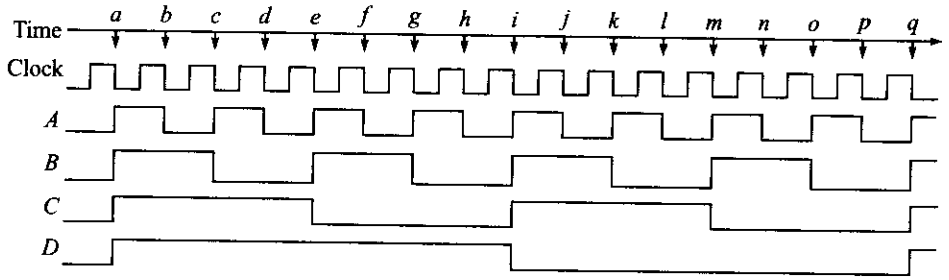


Note : All J and K inputs are tied to $+V_{CC}$.

(a) Logic diagram



(b) Count up waveforms



(c) Count down waveforms

Fig. 10.12 Synchronous, 4-bit up-down counter

Holding the count-down input low (at ground) will disable AND gates Y_1, Y_2, and Y_3. The clock applied at count-up will then go directly into flip-flop A and will be steered into the other flip-flops by AND gates X_1, X_2, and X_3. This counter will then function exactly as the previously discussed parallel counter shown in Fig. 10.10. The only difference here is that this is a mod-16 counter that advances one count with each clock NT, beginning with 0000 and ending with 1111. The correct waveforms are shown in Fig. 10.12b.

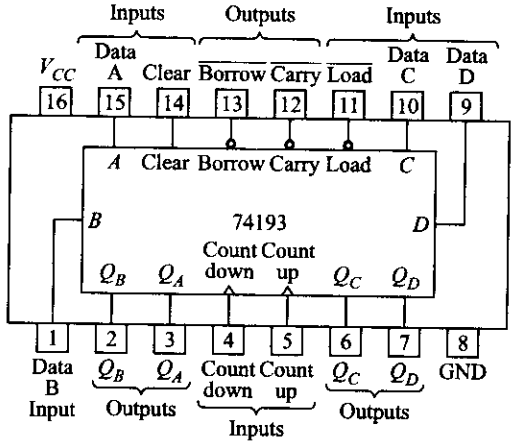
If the count-up line is held low, the upper AND gates X_1, X_2, and X_3 are disabled. The clock applied at input count-down will go directly into flip-flop A and be steered into the following flip-flops by AND gates Y_1, Y_2, and Y_3.

Flip-flop A will toggle each time there is a clock NT as shown in Fig. 10.12c. Each time $\overline{A}$ is high, AND gate Y_1 will be enabled and the clock NT will toggle flip-flop B at points a, c, e, g, and so on. Whenever both $\overline{A}$ and $\overline{B}$ are high, AND gate Y_2 is enabled, and thus a clock will be steered into flip-flop C at points a, e, i, m, and q. Similarly, AND gate Y_3 will steer a clock into flip-flop D only when $\overline{A}$, $\overline{B}$, and $\overline{C}$ are all high. Thus flip-flop D will toggle at points a and i on the time line. The waveforms in Fig. 10.12c clearly show that the counter is operating in a count-down mode, progressing one count at a time from 1111 to 0000.

If you examine the logic diagram for the 54/74193 TTL circuit shown in Fig. 10.13, you will see that it uses steering logic just like the counter in Fig. 10.12. This MSI circuit is a synchronous 4-bit up-down

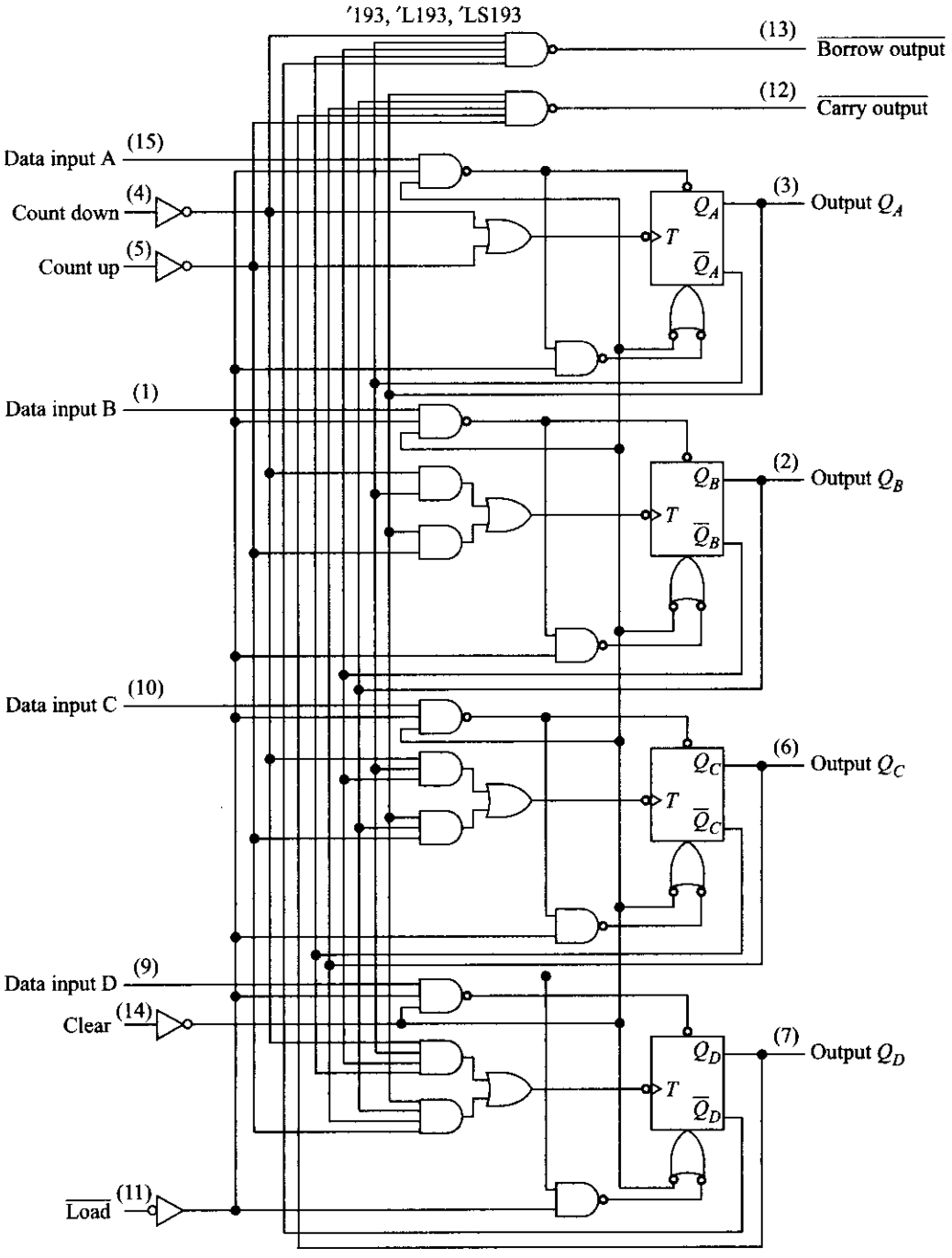
'192, '193, 'LS192, 'LS193.
(Top view)

Logic: low input to load sets Q_A = A, Q_B = B, Q_C = C, and Q_D = D



(a) Pinout

Fig. 10.13 54/74193



(b) Logic

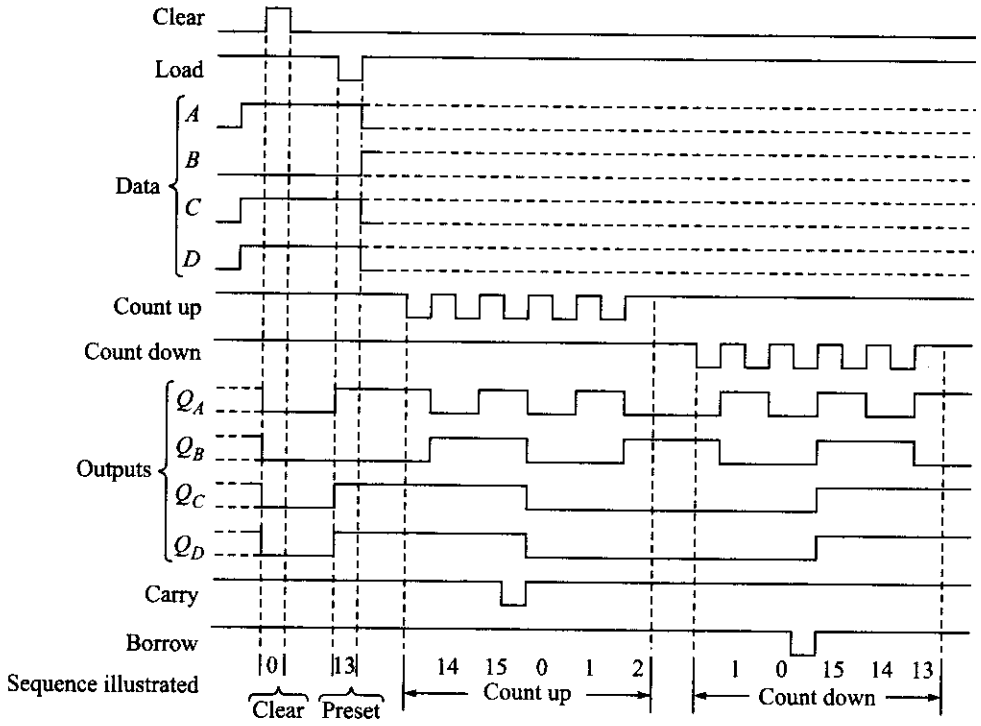
Fig. 10.13 **(Continued)**

'193, 'L193, 'LS193 Binary counters

Typical clear, load, and count sequences

Illustrated below is the following sequence.

1. Clear outputs to zero.
2. Load (preset) to binary thirteen.
3. Count up to fourteen, fifteen, carry, zero, one, and two.
4. Count down to one, zero, borrow, fifteen, fourteen, and thirteen.



Notes: A. Clear overrides load, data, and count inputs.
B. When counting up, count-down input must be high; when counting down, count-up input must be high.

(c) Waveforms for 54/74193

Fig. 10.13 (Continued)

counter that can also be cleared and preset to any desired count—attributes that we discuss later. For now, you should carefully examine the steering logic for each flip-flop and study the OR gate and the two AND gates at the input of the OR gate used to provide the clock to each flip-flop.

The waveforms for the 54/74193 are exactly the same as those shown in Fig. 10.12, except that the flip-flop outputs change states when the clock makes a low-to-high transition. Note carefully that the external clock (applied at either the count-up or the count-down input) passes through an inverter before being applied to the AND-OR-gate logic of each flip-flop clock input.

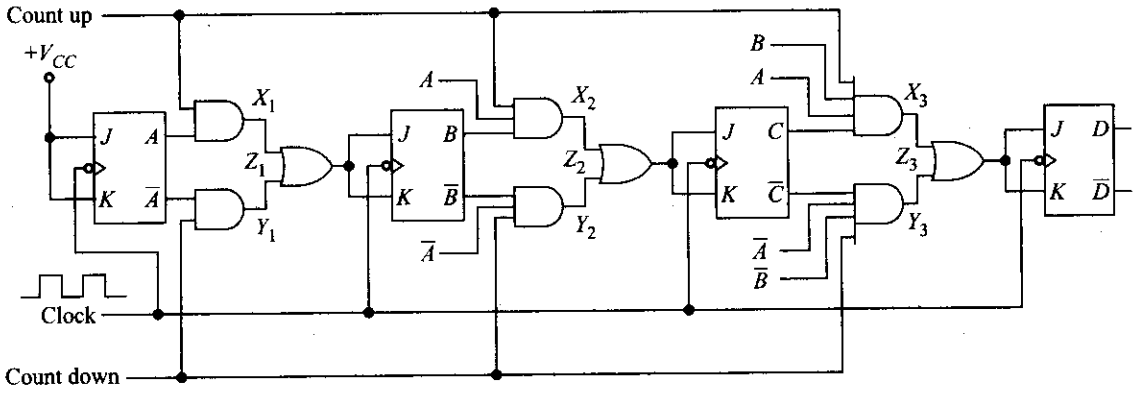
Example 10.5 Write a Boolean expression for the AND gate connected to the lower leg of the OR gate that drives the clock input to flip-flop Q_D in the 54/74193.

Solution The correct expression is

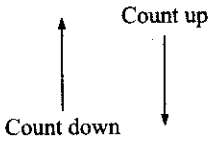
$$x = \overline{(\text{count-up clock})}\,(Q_A)(Q_B)(Q_C)$$

A parallel up-down counter can be formed by using a slightly different logic scheme, as shown in Fig. 10.14. Remember that in a parallel counter, the time at which any flip-flop changes state is determined by the states of all previous flip-flops in the counter. In the count-up mode, a flip-flop must toggle every time all previous flip-flops are in a 1 state, and the clock makes a transition. In the count-down mode, flip-flop toggles must occur when all prior flip-flops are in a 0 state.

This particular counter works in an *inhibit mode*, since each flip-flop changes state on a clock NT provided its J and K inputs are both high; a change of state will not occur when the J and K inputs are low. We



(a)



D	C	B	A	Count
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15
0	0	0	0	0

(b)

Fig. 10.14 **Parallel up-down counter**

might consider this is "look-ahead logic," since the mode of operation occurs in a strict time sequence as follows:

1. Establish a level on the J and K inputs (low or high)
2. Let the clock transition high to low
3. Look at the flip-flop output to determine whether it toggled.

To understand the logic used to implement this counter, refer to the truth table shown in Fig. 10.14b.

A is required to change state each time the clock goes low, and flip-flop A therefore has both its J and K inputs held in a high state. This is true in both the count-up and count-down modes, and thus no other logic is necessary for this flip-flop.

In the count-up mode, B is required to change state each time A is high and the clock goes low. Whenever the count-up line and A are both high, the output of gate X_1 is high. Whenever either input to Z_1 is high, the output is high. Therefore, the J and K inputs to flip-flop B are high whenever both count-up and A are high. Then, in the count-up mode, a clock NT will toggle B each time A is high, such as in going from count 1 to count 2, or 3 to 4, and so on.

In the count-down mode, B must change state each time $\overline{A}$ is high and the clock goes low. The output of gate Y_1 is high, and thus the J and K inputs to flip-flop B are high whenever $\overline{A}$ and count-down are high. Thus, in the count-down mode, B changes state every time $\overline{A}$ is high and the clock goes low—going from 0 to 15, or from 14 to 13, etc.

In the count-up mode, a clock NT must toggle C every time both A and B are high (transitions 3 to 4, 7 to 8, 11 to 12, and 15 to 0). The output of gate X_2 is high whenever both A and B are high and the count-up line is high. Thus, the J and K inputs to flip-flop C are high during these times and C changes state during the needed transitions.

In the count-down mode, C is required to change state whenever both $\overline{A}$ and $\overline{B}$ are high. The output of gate Y_2 is high any time both $\overline{A}$ and $\overline{B}$ are high, and the count-down line is high. Thus the J and K inputs to flip-flop C are high during these times, and C then changes state during the required transitions—that is, 0 to 15, 12 to 11, 8 to 7, and 4 to 3.

In the count-up mode, D must toggle every time $\overline{A}$, $\overline{B}$, and $\overline{C}$ are all high. The output of gate X_3 is high, and thus the J and K inputs to flip-flop D are high whenever A, B, and C and count-up are all high. Thus D changes state during the transitions from 7 to 8 and from 15 to 0.

In the count-down mode, a clock NT must toggle D whenever A, B, and C are all high. The output of gate Y_3 is high, and thus the J and K inputs to flip-flop D are high whenever $\overline{A}$, $\overline{B}$, and $\overline{C}$ and count-down are all high. Thus D changes state during the transitions from 0 to 15 and from 8 to 7. The count-up and count-down waveforms for this counter are exactly like those shown in Fig. 10.12.

Take a look at the logic diagram for the 54/74191 TTL MSI circuit shown in Fig. 10.15. This is a synchronous up-down counter. A careful examination of the AND-OR-gate logic used to precondition the J and K inputs to each flip-flop will reveal that this counter uses look-ahead logic exactly like the counter in Fig. 10.14. Additional logic allows one to clear or preset this counter to any desired count, and we study these functions later. For now, carefully compare the logic diagram with the counter in Fig. 10.14 to be certain you understand its operation.

Notice carefully that the clock input passes through an inverter before it is fed to the individual flip-flops. Thus the outputs of the four master-slave flip-flops will change states only on low-to-high transitions of the input clock. Typical waveforms are given in Fig. 10.15. Incidentally, these are precisely the same waveforms one would expect when using the 54/74193 discussed previously.

Example 10.6 Write a Boolean expression for the 4-input AND gate connected to the lower leg of the OR gate that conditions the J and K inputs to the Q_D flip-flop in a 54/74191.

Solution The correct logic expression is

$$x = \overline{(\text{down}-\text{up})}(Q_A)(Q_B)(Q_C)\overline{(\text{enable})}$$

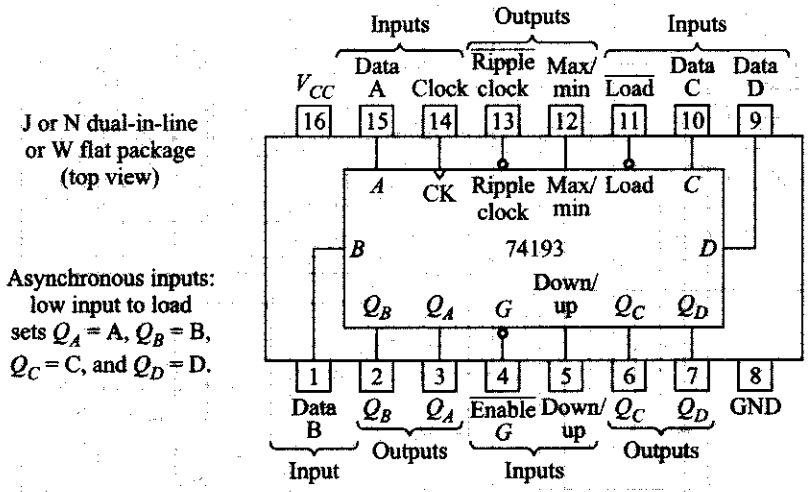


Fig. 10.15 **54/74191 (continued on next page)**

SELF-TEST

5. How does a parallel (synchronous) counter differ from a serial (asynchronous) counter?
6. Why are decoding gate glitches eliminated in a synchronous counter?
7. Does the 74193 change state with PTs or with NTs?

10.4 CHANGING THE COUNTER MODULUS

Counter Modulus

At this point, we have discussed asynchronous (ripple) counters and two different types of synchronous (parallel) counters, all of which have the ability to operate in either a count-up or count-down mode. All of these counters progress one count at a time in a strict binary progression, and they all have a modulus given by 2^n, where n indicates the number of flip-flops. Such counters are said to have a "natural count" of 2^n.

A mod-2 counter consists of a single flip-flop; a mod-4 counter requires two flip-flops, and it counts through four discrete states. Three flip-flops form a mod-8 counter, while four flip-flops form a mod-16 counter. Thus we can construct counters that have a natural count of 2, 4, 8, 16, 32, and so on by using the proper number of flip-flops.

It is often desirable to construct counters having a modulus other than 2, 4, 8, and so on. For example, a counter having a modulus of 3, or 5, would be useful. A small modulus counter can always be constructed

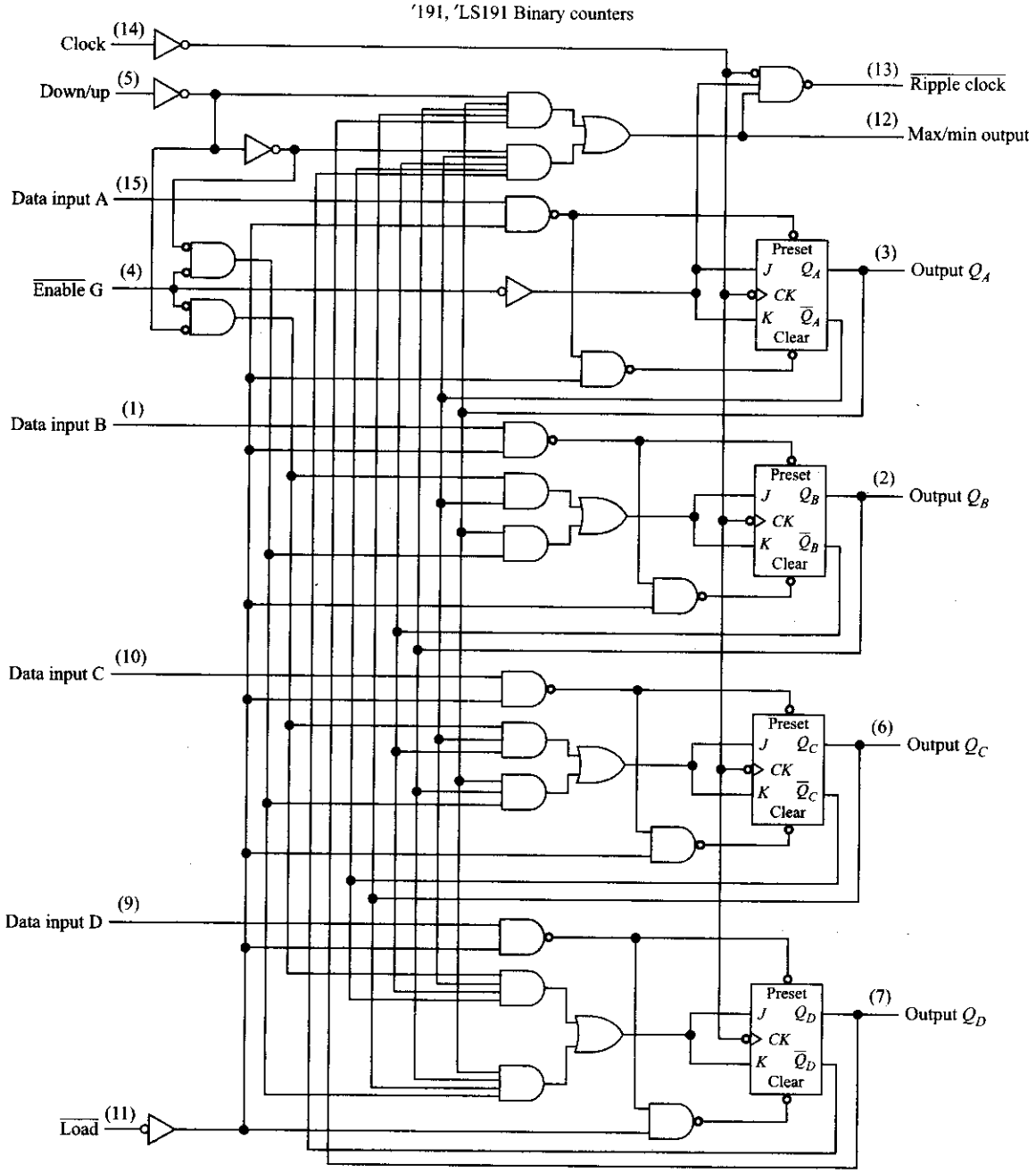


Fig. 10.15 (Continued)

'191, 'LS191 Binary counters

Typical load, count and inhibit sequences

Illustrated below is the following sequence.

1. Load (preset) to binary thirteen.
2. Count up to fourteen, fifteen (maximum), zero, one, and two.
3. Inhibit
4. Count down to one, zero (minimum), fifteen, fourteen, and thirteen.

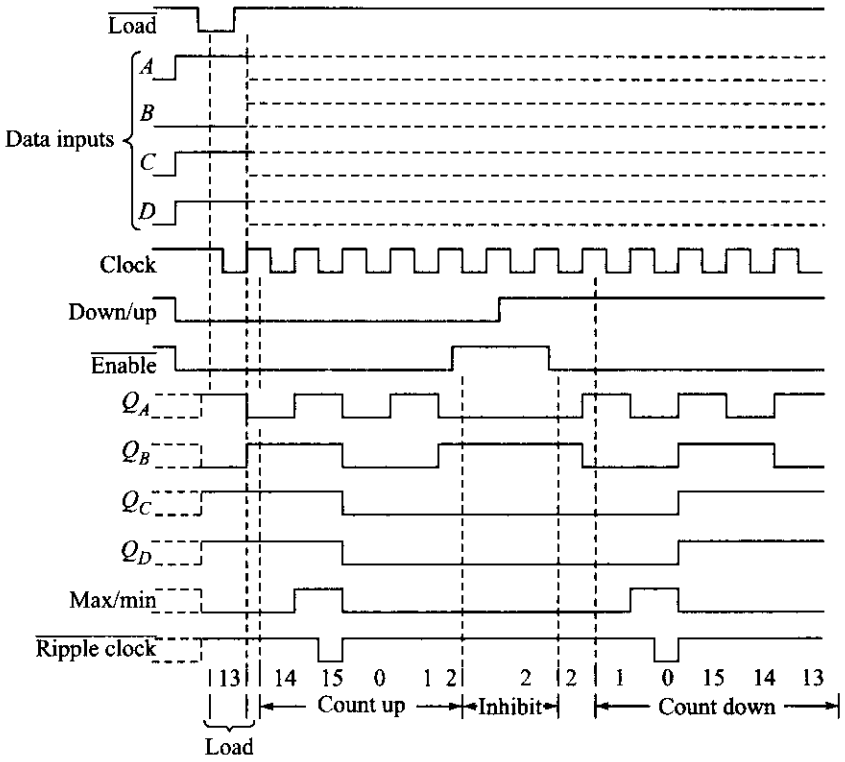


Fig. 10.15 **(Continued)**

from a larger modulus counter by skipping states. Such counters are said to have a *modified count*. It is first necessary to determine the number of flip-flops required. The correct number of flip-flops is determined choosing the lowest natural count that is greater than the desired modified count. For example, a mod-7 counter requires three flip-flops, since 8 is the lowest natural count greater than the desired modified count of 7.

Example 10.7 Indicate how many flip-flops are required to construct each of the following counters: (a) mod-3, (b) mod-6, and (c) mod-9.

Solution

a. The lowest natural count greater than 3 is 4. Two flip-flops provide a natural count of 4. Therefore, it requires at least two flip-flops to construct a mod-3 counter.

b. Construction of a mod-6 counter requires at least three flip-flops, since 8 is the lowest natural count greater than 6.

c. A mod-9 counter requires at least four flip-flops, since 16 is the lowest natural count greater than 9.

A single flip-flop has a natural count of 2; thus we could use a single flip-flop to construct a mod-2 counter, and that's all. However, a two flip-flop counter has a natural count of 4. Skipping one count will lead to a mod-3 counter. So, two flip-flops can be used to construct either a mod-4 or mod-3 counter.

Similarly, a three-flip-flop counter has a natural count of 8, but by skipping counts we can use three flip-flops to construct a counter having a modulus of 8, 7, 6, or 5. Note that counters having a modulus of 4 or 3 could also be constructed, but these two counters can be constructed by using only two flip-flops.

Example 10.8 What modulus counters can be constructed with the use of four flip-flops?

Solution A four-flip-flop counter has a natural count of 16. We can thus construct any counter that has a modulus between 16 and 2, inclusive. We might choose to use four flip-flops only for counters having a modulus between 16 and 9, since only three flip-flops are required for a modulus of less than 8, and only two are required for a modulus of less than 4.

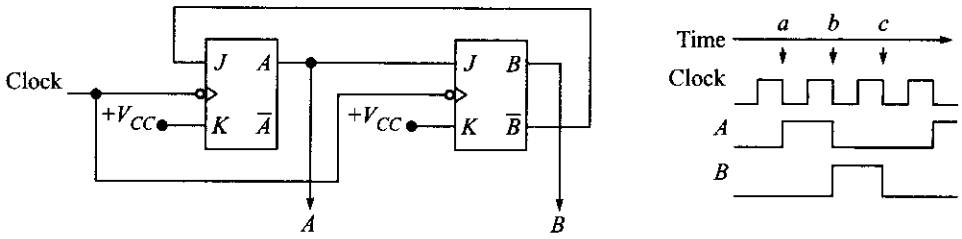
A Mod-3 Counter

There are a great many different methods for constructing a counter having a modified count. A counter can be synchronous, asynchronous, or a combination of these two types; furthermore, there is the decision of which count to skip. For instance, if a mod-6 counter using three flip-flops is to be constructed, which two of the eight discrete states should be skipped? Our purpose here is not to consider all possible counter configurations and how to design them; rather, we devote our efforts to one or two designs widely used in TTL MSI. A mod-3 counter is considered in this section and a mod-5 in the next section, and then we consider the use of presettable counters to achieve any desired modulus.

The two flip-flops in Fig. 10.16 have been connected to provide a mod-3 counter. Since two flip-flops have a natural count of 4, this counter skips one state. The waveforms and the truth table in Fig. 10.16 show that this counter progresses through the count sequence 00, 01, 10, and then back to 00. It clearly skips count 11. Here's how it works:

1. Prior to point a on the time line, $A = 0$ and $B = 0$. A negative clock transition at a will cause:
   a. A to toggle to a 1, since its J and K inputs are high
   b. B to reset to 0 (it's already a 0), since its J input is low and its K input is high
2. Prior to point b on the time line, $A = 1$, and $B = 0$. A negative clock transition at b will cause:
   a. A to toggle to a 0, since its J and K inputs are high
   b. B to toggle to a 1, since its J and K inputs are high
3. Prior to point c on the time line, $A = 0$ and $B = 1$. A negative clock transition at c will cause:
   a. A to reset to 0 (it's already 0), since its J input is low and its K input is high
   b. B to reset to 0 since its J input is low and its K input is high
4. The counter has now progressed through all three of its states, advancing one count with each negative clock transition.

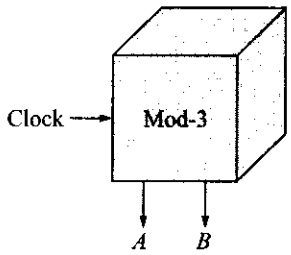
This two-flip-flop mod-3 counter can be considered as a logic building block as shown in Fig. 10.16d. It has a clock input and outputs at A and B. It can be considered as a divide-by-3 block, since the output



(a) Logic diagram (b) Waveforms

B	A	Count
0	0	0
0	1	1
1	0	2
0	0	0

(c) Truth table



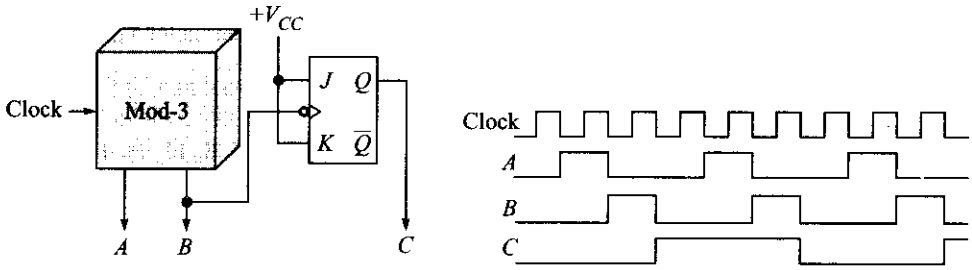
(d) Logic block

Fig. 10.16 Mod-3 counter

waveform at B (or at A) has a period equal to three times that of the clock—in other words, this counter divides the clock frequency by 3. Notice that this is a synchronous counter since both flip-flops change state in synchronism with the clock.

A Mod-6 Counter

If we consider a basic flip-flop to be a mod-2 counter, we see that a mod-4 counter (two flip-flops in series) is simply two mod-2 counters in series. Similarly, a mod-8 counter is simply a $2 \times 2 \times 2$ connection, and so on. Thus a great number of higher-modulus counters can be formed by using the product of any number of lower-modulus counters. For instance, suppose that we connect a flip-flop at the B output of the mod-3 counter in Fig. 10.16. The result is a $(3 \times 2 = 6)$ mod-6 counter as shown in Fig. 10.17. The output of the single flip-flop is labeled C. Notice that it is a symmetrical waveform, and it also has a frequency of one-sixth that of the input clock. Also, this can no longer be considered a synchronous counter since flip flop C is triggered by flip-flop B; that is, the flip-flops do not all change status in synchronism with the clock.

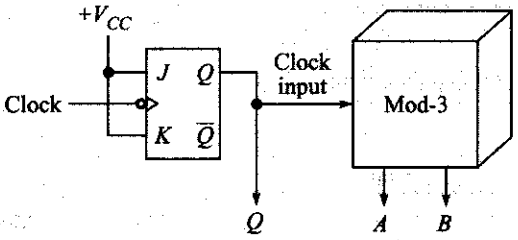


(a) 3×2 Mod-6 counter (b) Waveforms

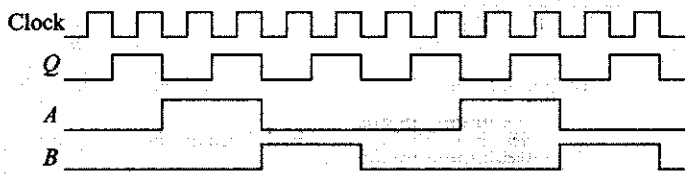
Fig. 10.17

Example 10.9 Draw the waveforms you would expect from the mod-6 counter by connecting a single flip-flop in front of the mod-3 counter in Fig. 10.16.

Solution The resulting counter is a 2 × 3 = mod-6 counter that has the waveforms shown in Fig. 10.18. Notice that *B* now has a period equal to six clock periods, but it is not symmetrical.



(a) 2 × 3 Mod-6 counter



(b) Waveforms

Fig. 10.18

The 54/7492A

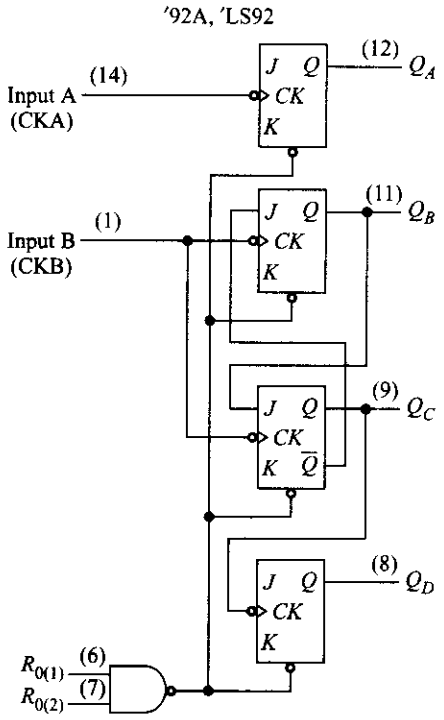
The 54/7492A ('92A) in Fig. 10.19 is a TTL divide-by-12, MSI counter. A careful examination of the logic diagram will reveal that flip-flops Q_B, Q_C, and Q_D are exactly the same as the 3 × 2 counter in Fig. 10.17. Thus if the clock is applied to input *B* of the '92A and the outputs are taken at Q_B, Q_C, and Q_D, this is a mod-6 counter.

On the other hand, if the clock is applied at input *A* and Q_A is connected to input *B*, we have a 2 × 3 × 2 mod-12 counter. The proper truth table for the mod-12 configuration is given in Fig. 10.19b. Again, this must be considered as an asynchronous counter since all flip-flops do not change states at the same time. Thus there is the possibility of glitches occurring at the outputs of any decoding gates used with the counter.

Example 10.10 Use the truth table for the '92A to write a Boolean expression for a gate to decode count 8.

Solution The correct expression is "8" = $Q_D Q_{\overline{C}} Q_B Q_{\overline{A}}$.

At this point, we can construct counters that have any natural count (2, 4, 8, 16, etc.) and, in addition, a mod-3 counter. Furthermore, we can cascade these counters in any combination, such as 2 × 2, 2 × 3, 3 × 4, and so on. So far we can construct counters having a modulus of 2, 3, 4, 6, 8, 9, 12, and so on. Therefore, let's consider next a mod-5 counter.

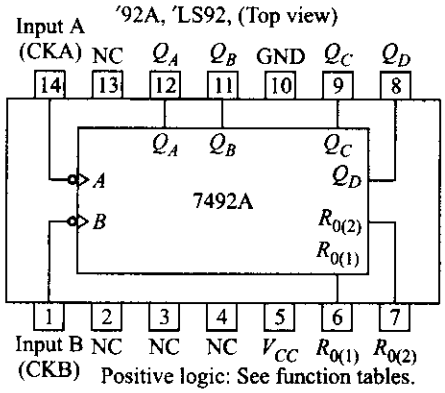


(a) Logic

'92A, 'LS92 Count sequence
(See Note C)

Count	Output Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	H	L	L	L
7	H	L	L	H
8	H	L	H	L
9	H	L	H	H
10	H	H	L	L
11	H	H	L	H

(b) Truth table



Note: Output Q_A connected to input B

(c) Pinout

Fig. 10.19 54/7492A

SELF-TEST

8. How many flip-flops are required to construct a mod-12 counter?
9. Three flip-flops are available. What modulus counters could be constructed?

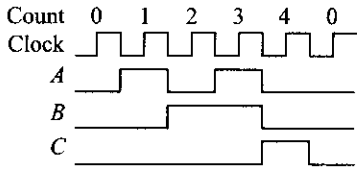
10.5 DECADE COUNTERS

A Mod-5 Counter

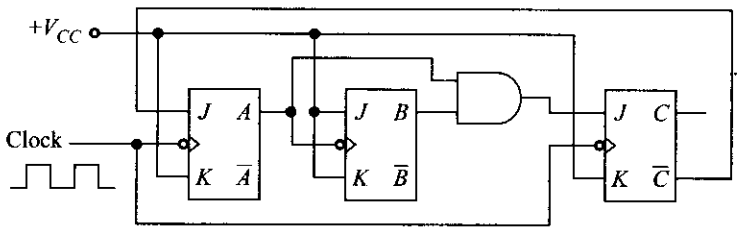
The three-flip-flop counter shown in Fig. 10.20 has a natural count of 8, but it is connected in such a way that it will skip over three counts. It will, in fact, advance one count at a time, through a strict binary sequence, beginning with 000 and ending with 100; therefore, it is a mod-5 counter. Let's see how it works.

C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
0	0	0	0

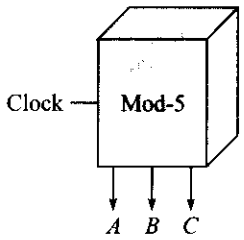
(a)



(b)



(c)



(d) Logic block

Fig. 10.20 **Mod-5 binary counter**

The waveforms show that flip-flop A changes state each time the clock goes negative, except during the transition from count 4 to count 0. Thus, flip-flop A should be triggered by the clock and must have an inhibit during count 4—that is, some signal must be provided during the transition from count 4 to count 0. Notice that $\overline{C}$ is high during all counts except count 4. If $\overline{C}$ is connected to the J input of flip-flop A, we will have the desired inhibit signal. This is true since the J and K inputs to flip-flop A are both true for all counts except count 4; thus the flip-flop triggers each time the clock goes negative. However, during count 4, the J side is low and the next time the clock goes negative the flip-flop will be prevented from being set. The connections which cause flip-flop A to progress through the desired sequence are shown in Fig. 10.20.

The desired waveforms (Fig. 10.20b) show that flip-flop B must change state each time A goes negative. Thus the clock input of flip-flop B will be driven by A (Fig. 10.20c).

If flip-flop C is triggered by the clock while the J input is held low and the K input high, every clock pulse will reset it. Now, if the J input is high only during count 3, C will be high during count 4 and low during all other counts. The necessary levels for the J input can be obtained by ANDing flip-flops A and B. Since A and B are both high only during count 3, the J input to flip-flop C is high only during count 3. Thus, when the clock goes negative during the transition from count 3 to count 4, flip-flop C will be set. At all other times, the J input to flip-flop C is low and is held in the reset state. The complete mod-5 counter is shown in Fig. 10.20.

In constructing a counter of this type, it is always necessary to examine the omitted states to make sure that the counter will not malfunction. This counter omits states 5, 6, and 7 during its normal operating sequence. There is however, a very real possibility that the counter may set up in one of these omitted (illegal) states when power is first applied to the system. It is necessary to check the operation of the counter when starting from each of the three illegal states to ensure that it progresses into the normal count sequence and does not become inoperative.

Begin by assuming that the counter is in state 5 (CBA = 101). When the next clock pulse goes low, the following events occur:

1. Since $\overline{C}$ is low, flip-flop A resets. Thus A changes from a 1 to a 0.
2. When A goes from a 1 to a 0, flip-flop B triggers and B changes from a 0 to a 1.
3. Since the J input to flip-flop C is low, flip-flop C is reset and C changes from a 1 to a 0.
4. Thus the counter progresses from the illegal state 5 to the legal state 2 (CBA = 010) after one clock.

Now, assume that the counter starts in the illegal state 6 (CBA = 110). On the next negative clock transition, the following events occur:

1. Since $\overline{C}$ is low, flip-flop A is reset. Since A is already a 0, it just remains a 0.
2. Since A does not change, flip-flop B does not change and B remains a 1.
3. Since the J input to flip-flop C is low, flip-flop C is reset and C changes from a 1 to a 0.
4. Thus the counter progresses from the illegal state 6 to the legal state 2 after one clock transition.

Finally, assume that the counter begins in the illegal state 7 (CBA = 111). On the next negative clock transition, the following events occur:

1. Since $\overline{C}$ is low, flip-flop A is reset and A changes from a 1 to a 0.
2. Since A changes from a 1 to a 0, flip-flop B triggers and B changes from a 1 to a 0.
3. The J input to flip-flop C is high; therefore, flip-flop C toggles from a 1 to a 0.
4. Thus the counter progresses from the illegal count 7 to the legal count 0 after one clock transition.

None of the three illegal states will cause the counter to malfunction, and it will automatically work itself out of any illegal state after only one clock transition.

A Mod-10 Counter

This mod-5 counter configuration can be considered as a logic block as shown in Fig. 10.20d and can be used in cascade to construct higher-modulus counters. For instance, a 2 × 5 or a 5 × 2 will form a mod-10 counter, or *decade counter*.

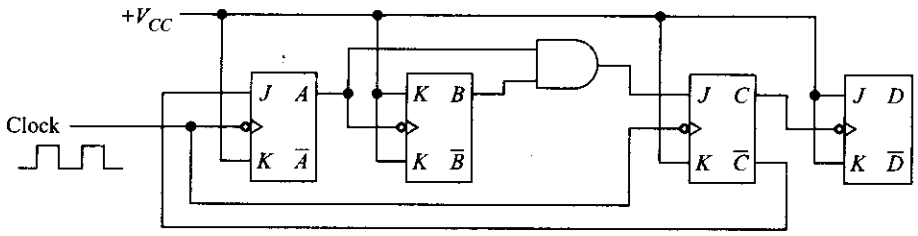
Example 10.11 Show a method for constructing a 5 × 2 (mod-10) decade counter.

Solution A decade counter can be constructed by using the mod-5 counter in Fig. 10.20 and adding an additional flip-flop, labeled D, as shown in Fig. 10.21. The appropriate waveforms and truth table are included. Notice that the counter progresses through a *biquinary* count sequence and does not count in a straight binary sequence.

A decade counter could be formed just as easily by using the mod-5 counter in Fig. 10.20 in conjunction with a flip-flop, but connected in a 2 × 5 configuration as shown in Fig. 10.22. The truth table for this configuration, and the resulting waveforms are shown. This is still a mod-10 (decade) counter since it still has 10 discrete states. Notice that this counter counts in a straight binary sequence from 0000 up to 1001, and then back to 0000.

The 7490A

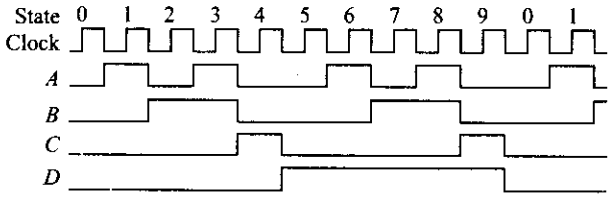
The 54/7490A is a TTL MSI decade counter. Its logic diagram, truth table, and pinout are given in Fig. 10.23. A careful examination will reveal that flip-flops Q_B, Q_C, and Q_D form a mod-5 counter exactly like the one in Fig. 10.20. Notice, however, that flip-flop Q_D in the '90A is an RS flip-flop that has a direct connection from its Q output back to its R input. The net result in this case is that Q_D behaves exactly like a JK flip-flop.



(a)

D	C	B	A	State
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
1	0	0	0	5
1	0	0	1	6
1	0	1	0	7
1	0	1	1	8
1	1	0	0	9
0	0	0	0	0

(b)

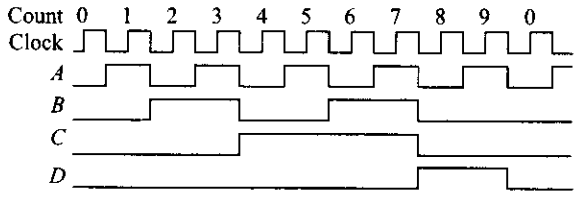


(c)

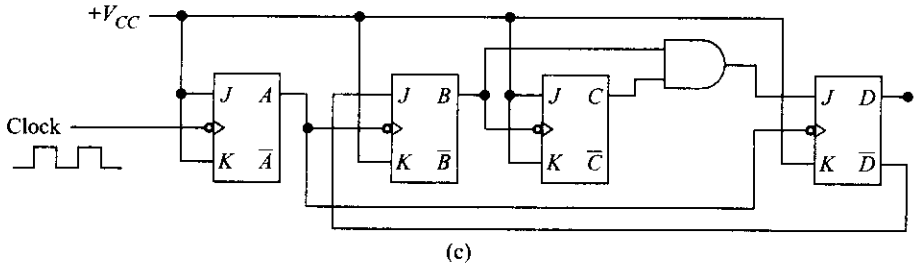
Fig. 10.21 A decade counter

D	C	B	A	Count
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
0	0	0	0	0

(a)

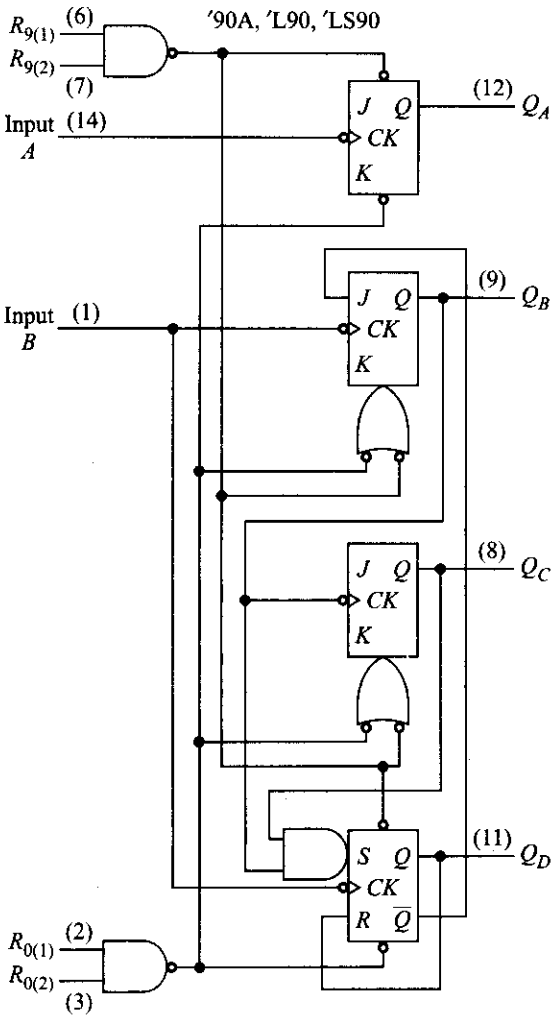


(b)



(c)

Fig. 10.22 A decade counter



(a) Logic

'90A, 'L90, 'LS90
BCD count sequence
(See note A)

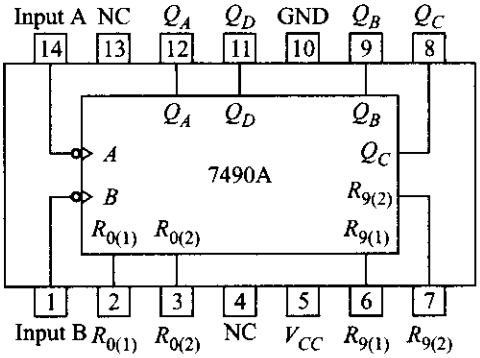
Count	Output			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

'90A, 'L90, 'LS90
Bi-quinary (5-2)
(See note B)

Count	Output			
	Q_A	Q_D	Q_C	Q_B
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

(b) Truth table

'90A, 'L90, 'LS90 (Top view)



Positive logic: See function tables

Note: A output Q_A connected to input B
B output Q_D connected to input A

(c) Pinout

Fig. 10.23 **54/7490A**

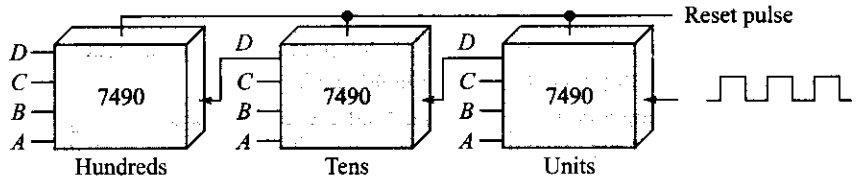


Fig. 10.24 **Cascaded 7490's can count to 999**

If the system clock is applied at input A and Q_A is connected to input B, we have a true binary decade counter exactly as in Fig. 10.22. On the other hand, if the system clock is applied at input B and Q_D is connected to input A, we have the biquinary counter as discussed in Example 10.11. Take time to study the logic diagram and the truth table for the '90A; it is widely used in industry, and the time spent will be well worth your while.

An interesting application using three 54/7490A decade counters is shown in Fig. 10.24. The three '90A counters are connected in series such that the first one (on the right) counts the number of input pulses at its clock input. We call it a *units counter*.

The middle '90A will advance one count each time the units counter counts 10 input pulses, because D from the units counter will have a single negative transition as that counter progresses from count 9 to 0. This middle block is then called *tens counter*.

The left '90A will advance one count each time the tens counter progresses from count 9 to 0. This will occur once for every 100 input pulses. Thus this block is called the *hundreds counter*.

Now the operation should be clear. This logic circuit is capable of counting input pulses from one up to 999. The procedure is to reset all the '90As and then count the number of pulses at the input to the units counter. This cascaded arrangement is widely used in digital voltmeters, frequency counters, etc., where a decimal count is needed.

It should be pointed out that the 54/7490A is only one of a number of TTL MSI decade counters. In particular, the 54/74176 is another popular asynchronous decade counter, and the 54/74160, 54/74162, 54/74190, and 54/74192 are all popular synchronous decade counters. Each has particular attributes that you should consider, and a study of their individual data sheets would be worthwhile.

SELF-TEST

10. What is a decade counter?
11. What is the difference between the 5×2 decade counter in Fig. 10.21 and the 2×5 decade counter in Fig. 10.22?

10.6 PRESETTABLE COUNTERS

Up to this point we have discussed the operation of counters that progress through a natural binary count sequence in either a count-up or count-down mode and have studied two counters that have a modified count—a mod-3 and a mod-5. With these basic configurations, and with cascaded combinations of these basic units, it is possible to construct counters having moduli of 2, 3, 4, 5, 6, 7, 8, 9, 10, and so on. The ability to quickly and easily construct a counter having any desired modulus is so important that the semiconductor industry has provided a number of TTL MSI circuits for this purpose. The presettable counter is the basic building block that can be used to implement a counter that has any modulus.

Nearly all the presettable counters available as TTL MSI are constructed by using four flip-flops, and they are generally referred to as *4-bit counters*. They may be either synchronous or asynchronous. When connected such that the count advances in a natural binary sequence from 0000 to 1111, it is simply referred to as a *binary counter*. For instance, the 54/74161 and the 54/74163 are both synchronous binary counters that operate in a count-up mode. The 54/74191 and the 54/74193 are also synchronous binary counters, but they can operate in either a count-down or count-up mode.