CHAPTER 7

FLIP-FLOPs

7.1 INTRODUCTION

So far we have directed our studies towards the analysis and design of combinational digital circuits. Though very important, it constitutes only a part of digital systems. The other major aspect of digital systems is analysis and design of sequential circuits. However, sequential circuit design depends, to a large extent, on the combinational circuit design discussed earlier.

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement can not be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

A block diagram of a sequential circuit is shown in Fig. 7.1. It consists of combinational circuits which accept digital signals from external inputs and from outputs of memory elements and generates signals for external outputs and for inputs to memory elements referred to as excitation.

A memory element is some medium in which one bit of information (1 or 0) can be stored or retained until necessary, and thereafter its contents can be replaced by a new value. The contents of memory elements in Fig. 7.1 can be changed by the outputs of the combinational circuit which are connected to its input.

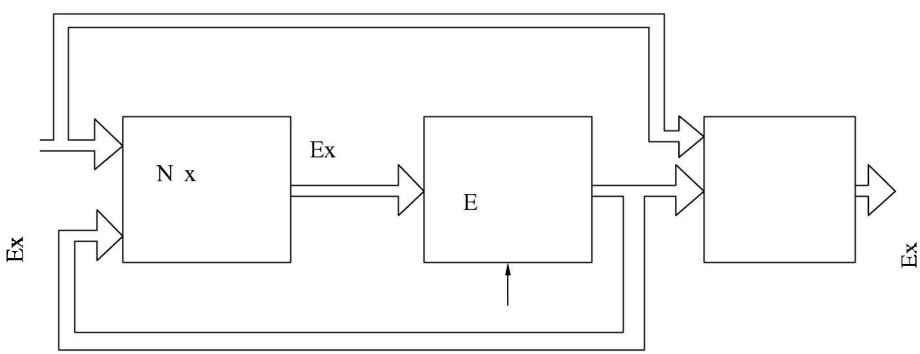


Fig. 7.1 ***Block Diagram of a Sequential Circuit***

The combinational circuit performs certain operations, some of which are used to determine the digital signals to be stored in memory elements. The other operations are performed on external inputs and memory outputs to generate the external outputs.

The above process demonstrates the dependence of the external outputs of a sequential circuit on the external inputs and the present contents of the memory elements (referred to as the *present state* of memory elements). The new contents of the memory elements, referred to as the *next state*; depend on the external inputs and the present state. Hence, the output of a sequential circuit is a function of the time sequence of inputs and the *internal states*.

Sequential circuits are classified in two main categories, known as *asynchronous* and *synchronous* sequential circuits depending on timing of their signals.

A sequential circuit whose behaviour depends upon the sequence in which the input signals change is referred to as an asynchronous sequential circuit. The outputs will be affected whenever the inputs change. The commonly used memory elements in these circuits are time delay devices. These can be regarded as combinational circuits with feedback.

A sequential circuit whose behaviour can be defined from the knowledge of its signal at discrete instants of time is referred to as a synchronous sequential circuit. In these systems, the memory elements are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a *system clock* which generates a periodic train of clock pulses as shown in Fig. 7.2. The outputs are affected only with the application of a clock pulse.

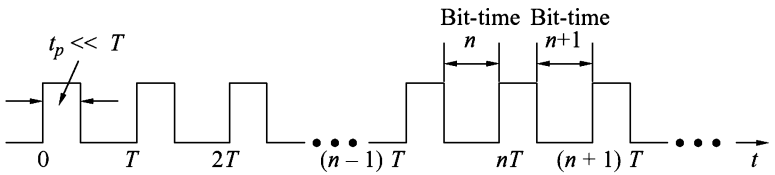


Fig. 7.2 ***A Train of Pulses***

Since the design of asynchronous circuits is more-tedious and difficult, therefore their uses are rather limited.

Synchronous circuits have gained considerable domination and wide popularity and are also known as *clocked-sequential circuits*. The memory elements used are FLIP-FLOPs which are capable of storing binary information.

7.2 A 1-BIT MEMORY CELL

The basic digital memory circuit is known as FLIP-FLOP. It has two stable states which are known as the *1 state* and the *0 state*. It can be obtained by using NAND or NOR gates. We shall be systematically developing a FLIP-FLOP circuit starting from the fundamental circuit shown in Fig. 7.3. It consists of two inverters G_1 and G_2 (NAND gates used as inverters). The output of G_1 is connected to the input of G_2 (A_2) and the output of G_2 is connected to the input of G_1 (A_1).

Let us assume the output of G_1 to be $Q = 1$, which is also the input of $G_2(A_2 = 1)$. Therefore, the output of G_2 will be $\overline{Q} = 0$, which makes $A_1 = 0$ and consequently $Q = 1$ which confirms our assumption.

In a similar manner, it can be demonstrated that if $Q = 0$, then $\overline{Q} = 1$ and this is also consistent with the circuit connections.

From the above discussion we note the following:

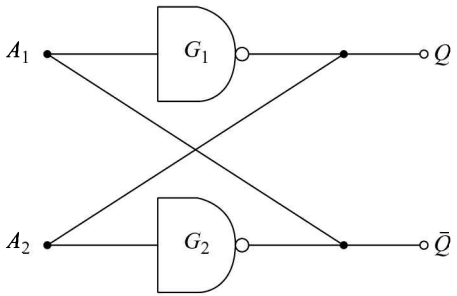


Fig. 7.3 ***Cross-coupled Inverters as a Memory Element***

1. The outputs Q and $\bar{Q}$ are always complementary.
2. The circuit has two stable states; in one of the stable state $Q = 1$ which is referred to as the 1 state (*or set state*) whereas in the other stable state $Q = 0$ which is referred to as the 0 state (*or reset state*).
3. If the circuit is in 1 state, it continues to remain in this state and similarly if it is in 0 state, it continues to remain in this state. This property of the circuit is referred to as *memory*, i.e. it can store 1-bit of digital information.

Since this information is locked or latched in this circuit, therefore, this circuit is also referred to as a *latch*.

In the latch of Fig. 7.3, there is no way of entering the desired digital information to be stored in it. In fact, when the power is switched on, the circuit switches to one of the stable states ($Q = 1$ or 0) and it is not possible to predict the state. If we replace the inverters G_1 and G_2 with 2-input NAND gates, the other input terminals of the NAND gates can be used to enter the desired digital information. The modified circuit is shown in Fig. 7.4. Two additional inverters G_3 and G_4 have been added for reasons which will become clear from the following discussion.

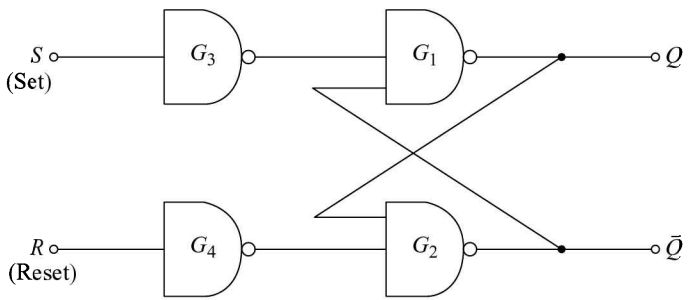


Fig. 7.4 ***The Memory Cell with Provision for Entering Data***

If $S = R = 0$, the circuit is exactly the same as that of Fig. 7.3 (Prob. 7.1). If $S = 1$ and $R = 0$, the output of G_3 will be 0 and the output of G_4 will be 1. Since one of the inputs of G_1 is 0, its output will certainly be 1. Consequently, both the inputs of G_2 will be 1 giving an output $\bar{Q} = 0$. Hence, for this input condition, $Q = 1$ and $\bar{Q} = 0$. Similarly, if $S = 0$ and $R = 1$ then the outputs will be $Q = 0$ and $\bar{Q} = 1$. The first of these two input conditions ($S = 1, R = 0$) makes $Q = 1$ which is referred to as the *set state*, whereas the second input condition ($S = 0, R = 1$) makes $Q = 0$ which is referred to as the *reset state* or *clear state*. This gives us the means for entering the desired bit in the latch.

Now we see what happens if the input conditions are changed from $S = 1, R = 0$ to $S = R = 0$ or from $S = 0, R = 1$ to $S = R = 0$. The output remains unaltered (Prob. 7.2). This shows the basic difference between a combinational circuit and a sequential circuit, even though the sequential circuit is made up of combinational circuits.

The two input terminals are designated as set (S) and reset (R) because $S = 1$ brings the circuit in set state and $R = 1$ brings it to reset or clear state.

If $S = R = 1$, both the outputs Q and $\bar{Q}$ will try to become 1 which is not allowed and therefore, this input condition is prohibited.

7.3 CLOCKED *S–R* FLIP-FLOP

It is often required to set or reset the memory cell (Fig. 7.4) in synchronism with a train of pulses (Fig. 7.2) known as clock (abbreviated as *CK*). Such a circuit is shown in Fig. 7.5, and is referred to as a *clocked set-reset* (*S–R*) FLIP-FLOP.

In this circuit, if a clock pulse is present ($CK = 1$), its operation is exactly the same as that of Fig. 7.4. On the other hand, when the clock pulse is not present ($CK = 0$), the gates G_3 and G_4 are inhibited, i.e. their outputs are 1 irrespective of the values of S or R. In other words, the circuit responds to the inputs S and R only when the clock is present.

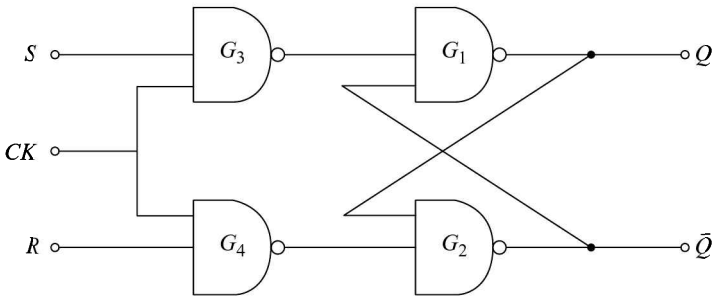


Fig. 7.5 ***A Clocked S–R*** FLIP-FLOP

Assuming that the inputs do not change during the presence of the clock pulse, we can express the operation of a FLIP-FLOP in the form of the truth table in Table 7.1 for the *S–R* FLIP-FLOP. Here S_n and R_n denote the inputs and Q_n the output during the bit time n (Fig. 7.2). Q_{n+1} denotes the output Q after the pulse passes, i.e. in the bit time $n + 1$.

Table 7.1 ***Truth Table of*** *S–R* FLIP-FLOP

Inputs		Output
S_n	R_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	?

If $S_n = R_n = 0$, and the clock pulse is applied, the output at the end of the clock pulse is same as the output before the clock pulse, i.e. $Q_{n+1} = Q_n$. This is indicated in the first row of the truth table.

If $S_n = 1$ and $R_n = 0$, the output at the end of the clock pulse will be 1, whereas if $S_n = 0$ and $R_n = 1$, then $Q_{n+1} = 0$. These are indicated in the second and third rows of the truth table respectively.

In the circuit of Fig. 7.4, it was mentioned that $S = R = 1$ is not allowed. Let us see what happens in the *S–R* FLIP-FLOP of Fig. 7.5 if $S_n = R_n = 1$. When the clock is present the outputs of gates G_3 and G_4 are both 0, making one of the inputs of G_1 and G_2 NAND gates 0. Consequently, Q and $\bar{Q}$ both will attain logic 1 which is inconsistent with our assumption of complementary outputs. Now, when the clock pulse has passed away ($CK = 0$), the outputs of G_3 and G_4 will rise from 0 to 1. Depending upon the propagation delays of the gates, either the stable state $Q_{n+1} = 1$ ($\bar{Q}_{n+1} = 0$) or $Q_{n+1} = 0$ ($\bar{Q}_{n+1} = 1$) will result. That means the state of the circuit is undefined, indeterminate or ambiguous and therefore is indicated by a question mark (fourth row of the truth table).

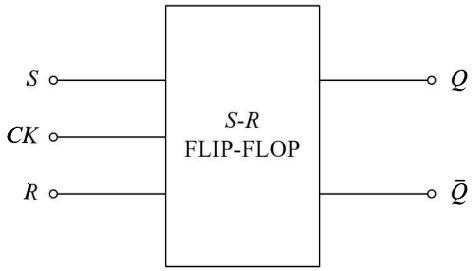


Fig. 7.6 ***Logic Symbol of Clocked*** *S–R* FLIP-FLOP

The condition $S_n = R_n = 1$ is forbidden and it must not be allowed to occur.

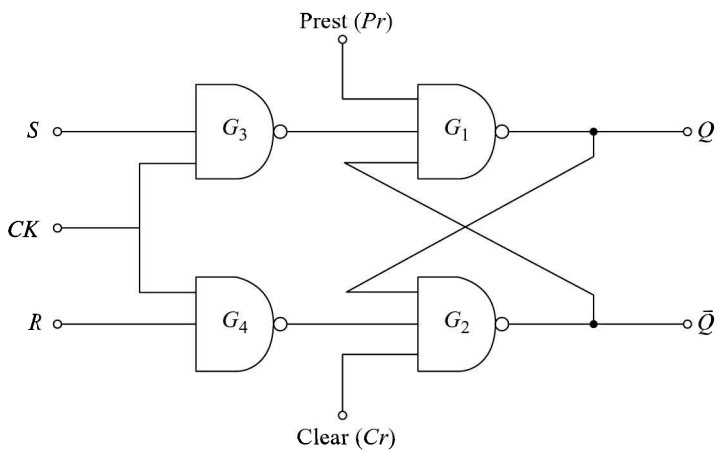
The logic symbol of clocked *S–R* FLIP-FLOP is given in Fig. 7.6.

7.3.1 Preset and Clear

In the FLIP-FLOP of Fig. 7.5, when the power is switched on, the state of the circuit is uncertain. It may come to set ($Q = 1$) or reset ($Q = 0$) state. In many applications it is desired to initially set or reset the FLIP-FLOP, i.e. the initial state of the FLIP-FLOP is to be assigned. This is accomplished by using the direct, or *asynchronous inputs*, referred to as *preset* (*Pr*) and *clear* (*Cr*) inputs. These inputs may be applied at any time between clock pulses and are not in synchronism with the clock. An *S–R* FLIP-FLOP with preset and clear is shown in Fig. 7.7. If $Pr = Cr = 1$ the circuit operates in accordance with the truth table of *S–R* FLIP-FLOP given in Table 7.1.

If $Pr = 0$ and $Cr = 1$, the output of $G_1(Q)$ will certainly be 1. Consequently, all the three inputs to G_2 will be 1 which will make $\bar{Q} = 0$. Hence, making $Pr = 0$ sets the FLIP-FLOP.

Similarly, if $Pr = 1$ and $Cr = 0$, the FLIP-FLOP is reset. Once the state of the FLIP-FLOP is established asynchronously, the asynchronous inputs *Pr* and *Cr* must be connected to logic 1 before the next clock is applied.



(a)

Fig. 7.7 ***(a) An*** *S–R* FLIP-FLOP ***with Preset and Clear, (b) Its Logic Symbol***

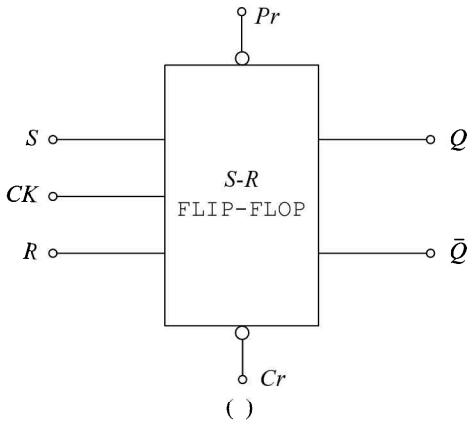


Fig. 7.7 ***(Continued)***

The condition $Pr = Cr = 0$ must not be used, since this leads to an uncertain state.

In the logic symbol of Fig. 7.7*b*, bubbles are used for *Pr* and *Cr* inputs, which means these are active-low, i.e. the intended function is performed when the signal applied to *Pr* or *Cr* is LOW The operation of Fig. 7.7 is summarised in Table 7.2.

The circuit can be designed such that the asynchronous inputs override the clock, i.e. the circuit can be set or reset even in the presence of the clock pulse (Prob. 7.4).

Table 7.2 ***Summary of Operation of S–R*** `FLIP-FLOP`

Inputs			Output	Operation performed
CK	***Cr***	***Pr***	***Q***	
1	1	1	Q_{n+1}(Table 7.1)	Normal `FLIP-FLOP`
0	0	1	0	Clear
0	1	0	1	Preset

7.4 *J–K* `FLIP-FLOP`

The uncertainty in the state of an *S–R* `FLIP-FLOP` when $S_n = R_n = 1$ (fourth row of the truth table) can be eliminated by converting it into a *J–K* `FLIP-FLOP`. The data inputs are *J* and *K* which are `AND`ed with $\bar{Q}$ and *Q*, respectively, to obtain *S* and *R* inputs, i.e.

$$S = J \cdot \bar{Q} \tag{7.1a}$$

$$R = K \cdot Q \tag{7.1b}$$

A *J–K* `FLIP-FLOP` thus obtained is shown in Fig. 7.8. Its truth table is given in Table 7.3*a* which is reduced to Table 7.3*b* for convenience. Table 7.3*a* has been prepared for all the possible combinations of *J*

and K inputs, and for each combination both the states of the output have been considered. The reader can verify this (Prob. 7.5).

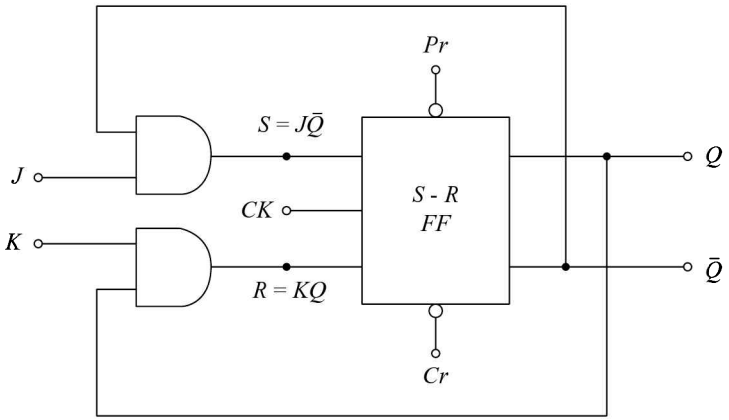


Fig. 7.8 ***An S–R*** `FLIP-FLOP` ***Converted into J–K*** `FLIP-FLOP`

Table 7.3a ***Truth Table for Fig.* 7.8**

Data inputs		Outputs		Inputs to S–R FF		Output
J_n	K_n	Q_n	$\bar{Q}_n$	S_n	R_n	Q_{n+1}
0	0	0	1	0	0	$\left.\begin{matrix}0\\1\end{matrix}\right] = Q_n$
0	0	1	0	0	0	
1	0	0	1	1	0	$\left.\begin{matrix}1\\1\end{matrix}\right] = 1$
1	0	1	0	0	0	
0	1	0	1	0	0	$\left.\begin{matrix}0\\0\end{matrix}\right] = 0$
0	1	1	0	0	1	
1	1	0	1	1	0	$\left.\begin{matrix}1\\0\end{matrix}\right] = \bar{Q}_n$
1	1	1	0	0	1	

Table 7.3b ***Truth Table of J–K*** FLIP-FLOP

Inputs		Output
J_n	K_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	$\bar{Q}_n$

It is not necessary to use the AND gates of Fig. 7.8, since the same function can be performed by adding an extra input terminal to each NAND gate G_3 and G_4 of Fig. 7.7 (Prob. 7.6). With this modification incorporated in Fig. 7.7, we obtain the *J–K* FLIP-FLOP using NAND gates as shown in Fig. 7.9. The logic symbol of *J–K* FLIP-FLOP is given in Fig. 7.10.

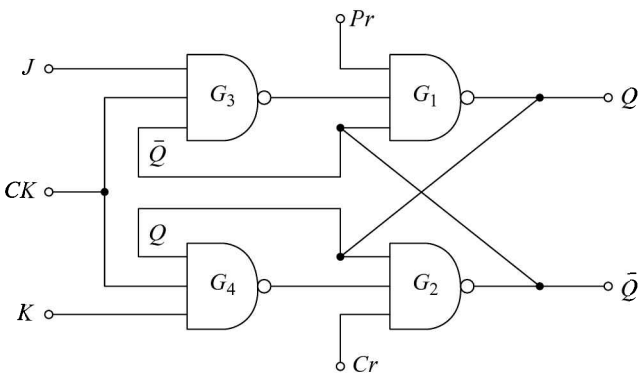


Fig. 7.9 ***A J–K*** FLIP-FLOP ***Using*** NAND ***Gates***

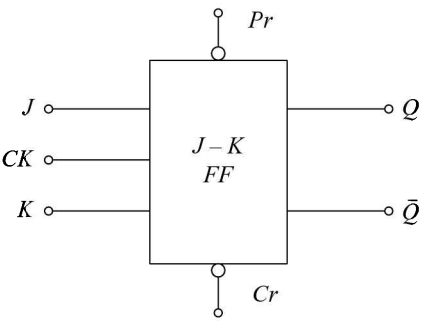


Fig. 7.10 ***Logic Symbols of J–K*** FLIP-FLOP

7.4.1 The Race-Around Condition

The difficulty of both inputs 1($S = R = 1$) being not allowed in an *S–R* FLIP-FLOP is eliminated in a *J–K* FLIP-FLOP by using the feedback connection from outputs to the inputs of the gates G_3 and G_4 (Fig. 7.9). Table 7.3 assumes that the inputs do not change during the clock pulse ($CK = 1$), which is not true because of the feedback connections. Consider, for example, that the inputs are $J = K = 1$ and $Q = 0$, and a pulse as shown in Fig. 7.11 is applied at the clock input. After a time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to $Q = 1$ (see fourth row of Table 7.3b). Now we have $J = K = 1$ and $Q = 1$ and after another time interval of Δt the output will change back to $Q = 0$. Hence, we conclude that for the duration t_p of the clock pulse, the output will oscillate back and forth between 0 and 1. At the end of the clock pulse, the value of Q is uncertain. This situation is referred to as the *race-around condition*.

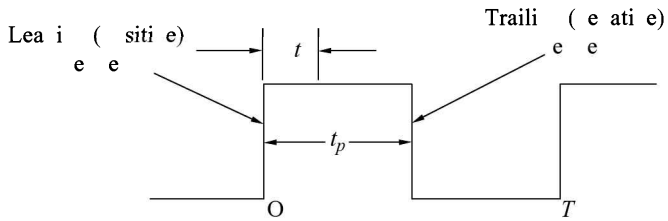


Fig. 7.11 ***A Clock Pulse***

The race-around condition can be avoided if $t_p < \Delta t < T$. However, it may be difficult to satisfy this inequality because of very small propagation delays in ICs. A more practical method for overcoming this difficulty is the use of the master–slave (*M–S*) configuration discussed below.

7.4.2 The Master–Slave *J–K* FLIP-FLOP

A master–slave *J–K* FLIP-FLOP is a cascade of two *S–R* FLIP-FLOPs, with feedback from the outputs of the second to the inputs of the first as illustrated in Fig. 7.12. Positive clock pulses are applied to the first FLIP-FLOP and the clock pulses are inverted before these are applied to the second FLIP-FLOP.

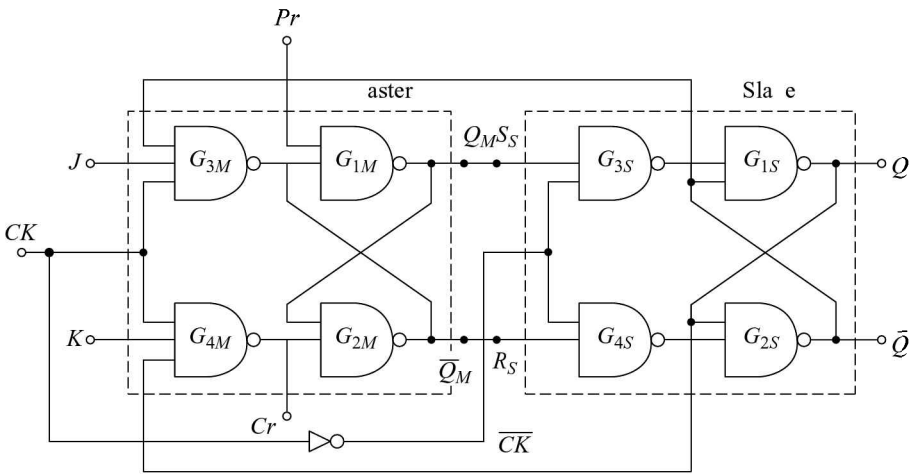


Fig. 7.12 ***A Master–Slave*** *J–K* FLIP-FLOP

When $CK = 1$, the first FLIP-FLOP is enabled and the outputs Q_M and $\overline{Q}_M$ respond to the inputs J and K according to Table 7.3. At this time, the second FLIP-FLOP is inhibited because its clock is LOW ($\overline{CK} = 0$). When CK goes LOW ($\overline{CK} = 1$), the first FLIP-FLOP is inhibited and the second FLIP-FLOP is enabled, because now its clock is HIGH ($\overline{CK} = 1$). Therefore, the outputs Q and $\overline{Q}$ follow the outputs Q_M and $\overline{Q}_M$, respectively (second and third rows of Table 7.3b). Since the second FLIP-FLOP simply follows the first one, it is referred to as the *slave* and the first one as the *master*. Hence, this configuration is referred to as *master—slave* (*M–S*) FLIP-FLOP.

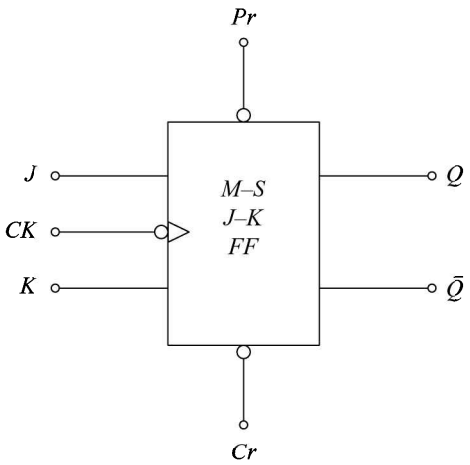


Fig. 7.13 ***A Master–Slave J–K*** `FLIP-FLOP` ***Logic Symbol***

In this circuit, the inputs to the gates G_{3M} and G_{4M} do not change during the clock pulse, therefore the race-around condition does not exist. The state of the master–slave `FLIP-FLOP` changes at the negative transition (trailing edge) of the clock pulse. The logic symbol of a *M–S* `FLIP-FLOP` is given in Fig. 7.13. At the clock input terminal, the symbol > is used to illustrate that the output changes when the clock makes a transition and the accompanying bubble signifies negative transition (change in *CK* from 1 to 0).

7.5 *D*-TYPE `FLIP-FLOP`

If we use only the middle two rows of the truth table of the *S–R* (Table 7.1) or *J–K* (Table 7.3b) `FLIP-FLOP`, we obtain a *D*-type `FLIP-FLOP` as shown in Fig. 7.14. It has only one input referred to as *D*-input or data input. Its truth table is given in Table 7.4 from which it is clear that the output Q_{n+1} at the end of the clock pulse equals the input D_n before the clock pulse.

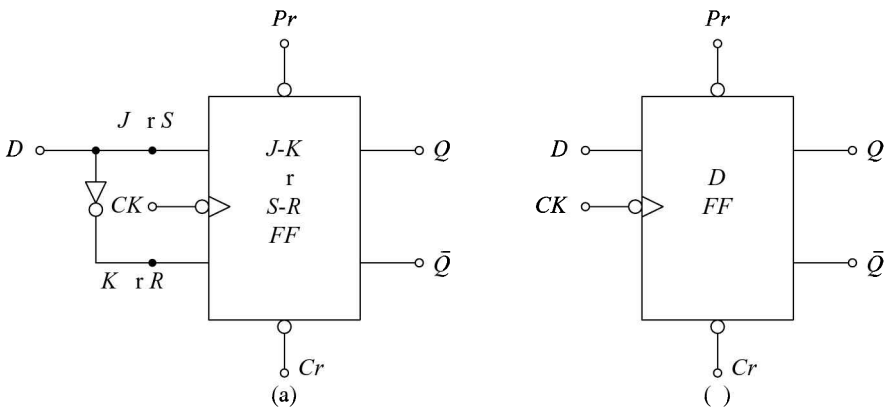


Fig. 7.14 ***(a) A J–K or S–R*** `FLIP-FLOP` ***Converted into a D-type*** `FLIP-FLOP` ***(b) its Logic Symbol***

Table 7.4 ***Truth Table of a D-type*** `FLIP-FLOP`

Input D_n	Output Q_{n+1}
0	0
1	1

This is equivalent to saying that the input data appears at the output at the end of the clock pulse. Thus, the transfer of data from the input to the output is delayed and hence the name *delay* (*D*) FLIP-FLOP. The *D*-type FLIP-FLOP is either used as a delay device or as a latch to store 1-bit of binary information.

7.6 *T*-TYPE FLIP-FLOP

In a *J–K* FLIP-FLOP, if $J = K$, the resulting FLIP-FLOP is referred to as a *T*-type FLIP-FLOP and is shown in Fig. 7.15. It has only one input, referred to as *T*-input. Its truth table is given in Table 7.5 from which it is clear that if $T = 1$ it acts as a toggle switch. For every clock pulse, the output Q changes.

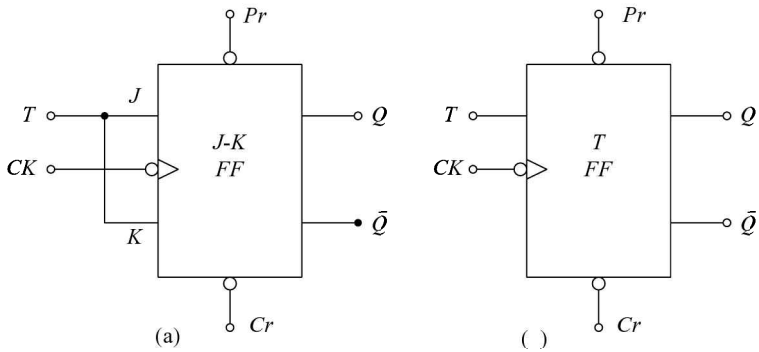


Fig. 7.15 ***(a) A J–K*** FLIP-FLOP ***Converted into a T-type*** FLIP-FLOP ***(b) its Logic Symbol***

Table 7.5 ***Truth Table of T-type*** FLIP-FLOP

Input	Output
T_n	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

An *S–R* FLIP-FLOP cannot be converted into a *T*-type FLIP-FLOP since $S = R = 1$ is not allowed. However, the circuit of Fig. 7.16 acts as a toggle switch, i.e. the output Q changes with every clock pulse (Prob. 7.11).

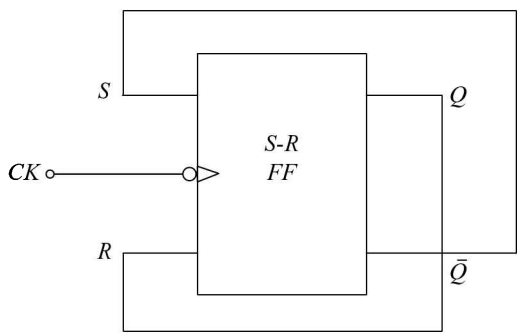


Fig. 7.16 ***An S–R*** FLIP-FLOP ***as a Toggle Switch***

7.7 EXCITATION TABLE OF FLIP-FLOP

The truth table of a FLIP-FLOP is also referred to as the *characteristic table* and specifies the operational characteristic of the FLIP-FLOP.

In the design of sequential circuits, we usually come across situations in which the *present state* and the *next state* of the circuit are specified, and we have to find the input conditions that must prevail to cause the desired transition of the state. By the present state and the next state we mean the state of the circuit prior to and after the clock pulse respectively. For example, the output of an *S–R* FLIP-FLOP before the clock pulse is $Q_n = 0$ and it is desired that the output does not change when the clock pulse is applied. What input conditions (S_n and R_n values) must exist to achieve this?

From the truth table (or the characteristic table) of an *S–R* FLIP-FLOP (Table 7.1) we obtain the following conditions:

1. $S_n = R_n = 0$ (first row)
2. $S_n = 0, R_n = 1$ (third-row)

We conclude from the above conditions that the S_n input must be 0, whereas the R_n input may be either 0 or 1 (don't-care). Similarly, input conditions can be found for all possible situations. A tabulation of these conditions is known as the *excitation table*. It is a very important and useful design aid for sequential circuits. Table 7.6 gives the excitation tables of *S–R*, *J–K*, *T*, and *D* FLIP-FLOPs. This is derived from the characteristic table of the FLIP-FLOP.

Table 7.6 ***Excitation Table of* FLIP-FLOPs**

Present State	Next State	*S–R* FF		*J–K* FF		*T*-FF	*D*-FF
		S_n	R_n	J_n	K_n	T_n	D_n
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	1	0
1	1	×	0	×	0	0	1

7.8 CLOCKED FLIP-FLOP DESIGN

In earlier sections, we defined or specified the operation of different FLIP-FLOPs assuming a circuit without regard to where the circuit came from or how it was designed. In this section, the design of a FLIP-FLOP is given. The design philosophy illustrated is, in fact, a general approach for the design of sequential circuits and systems.

Consider the general model of the FLIP-FLOP shown in Fig. 7.17. Basically, a clocked FLIP-FLOP is a sequential circuit which stores the bits 0 and 1. This operation is accomplished by using a binary cell

coupled with some combinational set/reset decoding logic to allow some input control over the set and reset operations of the cell. The steps for the design of FLIP-FLOP are given below.

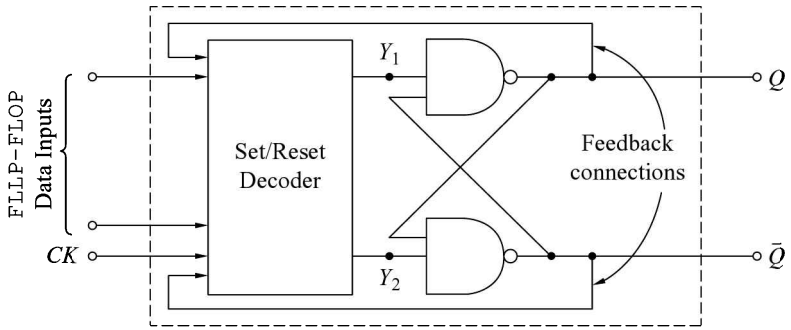


Fig. 7.17 ***The General Model of the*** FLIP-FLOP

Step 1. Examine each row of the given characteristic table, specifying the desired inputs and outputs, and answer the following questions and make a truth table with Y_1 and Y_2 as output variables.

1. Does the cell need to be set ($Q_{n+1} = 1$) for this condition?
2. Does the cell need to be reset ($Q_{n+1} = 0$) for this condition?
3. Does the cell need to be left as it is?

Step 2. Prepare the *K*-map for Y_1 and Y_2 output variables, minimize it and determine the logic for Y_1 and Y_2, respectively. Draw the complete circuit using gates.

The above design steps are illustrated in Example 7.1.

Example 7.1

Using the technique described above, design a clocked *S–R* FLIP-FLOP whose characteristic table is given in Table 7.7.

Solution

Step 1. Determine the values of Y_1 and Y_2 for each row. For example, for the first row $Q_n = 0$ and $Q_{n+1} = 0$. To obtain $Q_{n+1} = 0$, Y_1 must be equal to 1, since $\bar{Q}_n = 1$, Y_2 can be 0 or 1 since $Q_n = 0$. In a similar manner, complete the truth table (Table 7.7).

Table 7.7 ***Truth Table***

Characteristic table					Truth table for decoder	
CK	***S***	***R***	Q_n	Q_{n+1}	Y_1	Y_2
0	0	0	0	0	1	×
0	0	0	1	1	×	1
0	0	1	0	0	1	×
0	0	1	1	1	×	1
0	1	0	0	0	1	×
0	1	0	1	1	×	1
0	1	1	0	0	1	× } *
0	1	1	1	1	×	1 } *
1	0	0	0	0	1	×
1	0	0	1	1	×	1
1	0	1	0	0	1	×
1	0	1	1	0	1	0
1	1	0	0	1	0	1
1	1	0	1	1	×	1
1	1	1	0	×	×	× } **
1	1	1	1	×	×	× } **

*$S = R = 1$ can happen with no clock.
**$S = R = 1$ must not happen.

Step 2. The *K*-maps for Y_1 and Y_2 are given in Fig. 7.18 which give

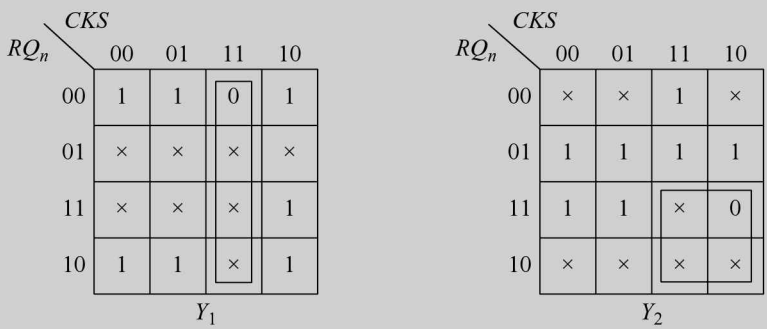


Fig. 7.18 ***K-maps for Ex. 7.1***

$$Y_1 = \overline{CK} + \overline{S} = \overline{CK \cdot S}$$

$$Y_2 = \overline{R} + \overline{CK} = \overline{CK \cdot R}$$

Thus, we see that the circuit resulting from this design is the same as that shown in Fig. 7.5.

7.8.1 Conversion from One Type of FLIP-FLOP to Another Type

In earlier sections, we have discussed conversion from *S–R* to *J–K*, *S–R* (or *J–K*) to *D*-type, and *J–K* to *T*-type FLIP-FLOPs. Now, we shall effect the conversion from one type of FLIP-FLOP to another type by using a formal technique which is similar to the one used above and will be useful in the design of clocked sequential circuits.

Consider the general model for conversion from one type of FLIP-FLOP to another type (Fig. 7.19). In this, we are required to design the combinational logic decoder (conversion logic) for converting new input definitions into input codes which will cause the given FLIP-FLOP to perform as desired.

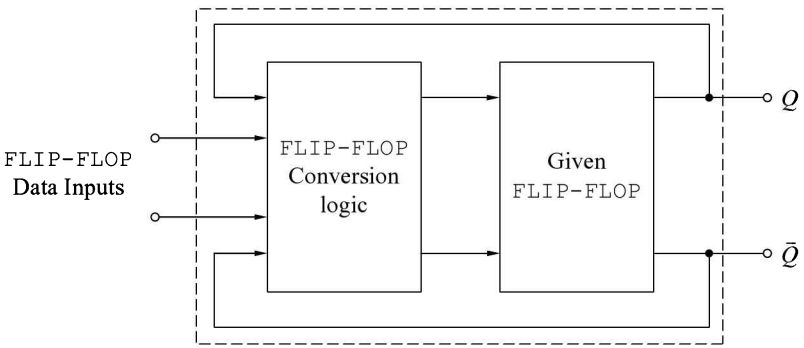


Fig. 7.19 ***The General Model Used to Convert One Type of*** FLIP-FLOP ***to Another Type***

To design the conversion logic we need to combine the excitation tables for both FLIP-FLOPs and make a truth table with data input(s) and Q as the inputs and the input(s) of the given FLIP-FLOP as the output(s). The conventional method of combinational logic design then follows as usual. The conversion is illustrated in Example 7.2.

Example 7.2

Convert an *S–R* FLIP-FLOP to a *J–K* FLIP-FLOP.

Solution

The excitation tables of *S–R* and *J–K* FLIP-FLOPs are given in Table 7.6 from which we make the truth table given in Table 7.8.

Table 7.8 ***Truth Table of Conversion Logic***

Row	*FF* data inputs		Output	*S–R FF* inputs	
	J	***K***	***Q***	***S***	***R***
1	0	0	0	0	×
2	0	1	0	0	×
3	1	0	0	1	0
4	1	1	0	1	0
5	0	1	1	0	1
6	1	1	1	0	1
7	0	0	1	×	0
8	1	0	1	×	0

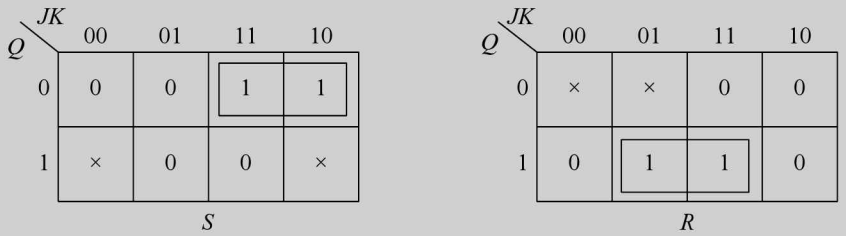


Fig. 7.20 ***K-maps for Ex. 7.2***

The *K*-maps are given in Fig. 7.20, which give

$$S = J \cdot \bar{Q} \qquad \text{and} \qquad R = K \cdot Q$$

Thus, we see that the circuit resulting from this design is the same as that shown in Fig. 7.8.

7.9 EDGE-TRIGGERED FLIP-FLOPs

All FLIP-FLOPs other than the master–slave type discussed in earlier sections are level triggered, i.e. the outputs respond to the inputs (according to the truth table) as long as the clock is present. The only ICs available in this category are latches, for example 7475 and 74100 are transparent latches.

The master–slave FLIP-FLOPs are also referred to as the *pulse-triggered* FLIP-FLOPs, i.e. the outputs respond to the inputs when a pulse is applied at the clock input. The master FLIP-FLOP responds when the

clock is present ($CK = 1$) and the output of the slave will be available at the falling edge of the clock pulse ($CK = 0$). As discussed in Section 7.4, this eliminates the problem of race-around condition. In this the data is *locked-out* at the falling edge of the clock pulse, i.e. the changes occurring at the inputs once CK goes to 0 will not affect the operation of the FLIP-FLOP.

The inputs to the FLIP-FLOP may change during the presence of the clock pulse due to certain operations in the system. This causes uncertainty in the outputs of the FLIP-FLOP which is eliminated by using edge-triggered FLIP-FLOPs.

In the case of an edge-triggered FLIP-FLOP, the transfer of information from data input(s) to the output of the FLIP-FLOP occurs at the positive (or negative) edge of the clock pulse. The only time the outputs can change state is during the brief interval of time when the clock signal is making a transition from the 0 to 1 (↑), or in some circuits, from the 1 to 0 (↓) states. A FLIP-FLOP which responds only to rising (or falling) edge is referred to as *positive-edge-triggered* (or *negative-edge-triggered*). The data lock-out occurs at the end of the edge.

The logic symbol used for an edge-triggered FLIP-FLOP is the same as that of a master–slave FLIP-FLOP. These are shown in Figs. 7.13, 7.14*b*, and 7.15*b*.

The timing specifications of an edge-triggered FLIP-FLOP are illustrated in Fig. 7.21 and are explained below.

Set-up Time (t_s) It is the time required for the input data to settle in before the triggering edge of the clock. Its minimum time is usually specified by the manufacturers.

Hold Time (t_h) It is the time for which the data must remain stable after the triggering edge of the clock. Minimum value of hold time is specified by the manufacturers.

The t_s and t_h timings are shown in Fig. 7.21*a*.

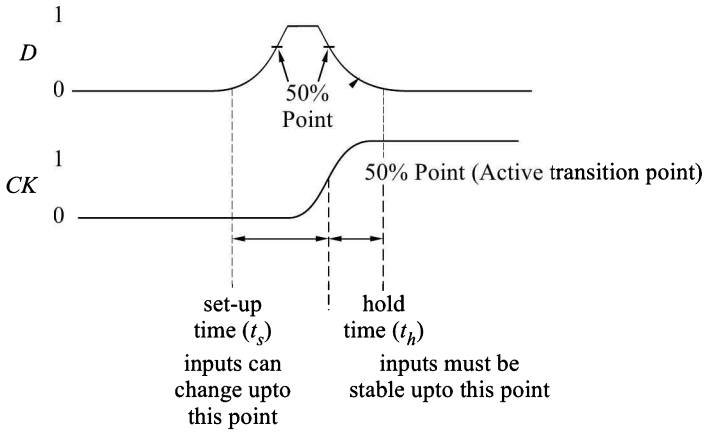
Propagation Delays Similar to propagation delay in combinational circuits, there is delay in the FLIP-FLOP output Q, making a change from HIGH-to-LOW or LOW-to-HIGH when a clock pulse is applied. These delays are specified between the 50% points on the clock and data input waveforms. The propagation delay, when the output Q changes from LOW-to-HIGH and HIGH-to-LOW, are specified as t_{pLH} and t_{pHL} respectively. These are shown in Fig. 7.21*b*.

Clock Pulse Width The minimum time duration for which the clock pulse must remain HIGH (t_{CH}) and LOW (t_{CL}) are specified by the manufacturers. Failure to meet these requirements may result in unreliable triggering. These timings are specified between the 50% points on the clock transitions and are shown in Fig 7.21*c*.

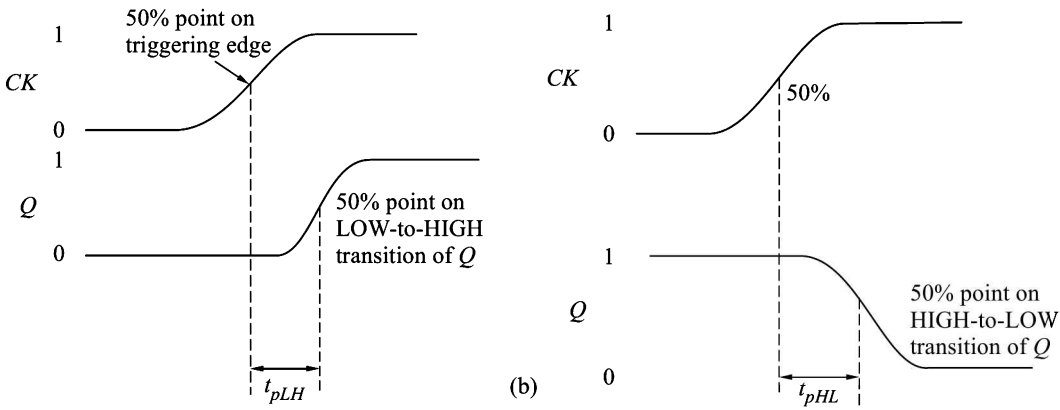
Preset and Clear Pulse Width The manufacturers also specify the minimum time duration for a preset input (t_{pLH}) and clear input (t_{pHL}).

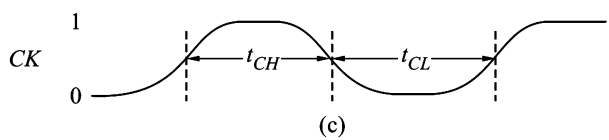
Maximum Clock Frequency The maximum clock frequency (f_{max}) is the highest rate at which a FLIP-FLOP can be reliably triggered. If the clock frequency is higher than this, the FF will be enable to trigger reliably.

Various timing values for some of the commonly used TTL and CMOS FLIP-FLOPs are given in Table 7.9.



(a)





(c)

Fig. 7.21 FLIP-FLOP ***Timings (a) Set-up and Hold Timings (b)Propagation Delays (c) Clock*** LOW ***and*** HIGH ***Timings***

7.9 *Timing Parameters of* TTL *and* CMOS FLIP-FLOP*s*

	TTL			CMOS			
arameter	**74LS74A**	**74LS112**	**74F74**	**74HC74A**	**74HC112**	**74AHC74**	**UN**
et up time)	20	20	2	14	25	5	n
hold time)	5	0	1	3	0	0.5	n
(*CK* to *Q*)	40	24	6.8	17	31	4.6	n
(*CK* to *Q*)	25	16	8	17	31	4.6	n
(*Cr* to *Q*)	40	24	9	18	41	4.8	n
(*Pr* to *Q*)	25	16	6.1	18	41	4.8	n
clock HIGH time)	25	20	4	10	25	5	n
clock LOW time)	25	15	5	10	25	5	n
(*Pr* or *Cr* OW time)	25	15	4	10	25	5	n
$_{AX}$ (Max. equency)	25	30	100	35	20	170	M

Example 7.3

The clock (Fig. 7.22*a*) and input (Fig. 7.22*b*) waveforms are applied to *D* or *J* input of each of the following type of FLIP–FLOPs. Sketch the output waveform in each case.

(a) Positive-edge-triggered *D*-type FLIP–FLOP 74 AHC74.
(b) Positive-level-triggered *D*-type FLIP–FLOP 7475 (transparent latch).
(c) Negative-edge-triggered *J–K* FLIP–FLOP (*K* = 1) 74 HC112.
(d) Master–slave *J–K* FLIP–FLOP (*K* = 1) 7476.

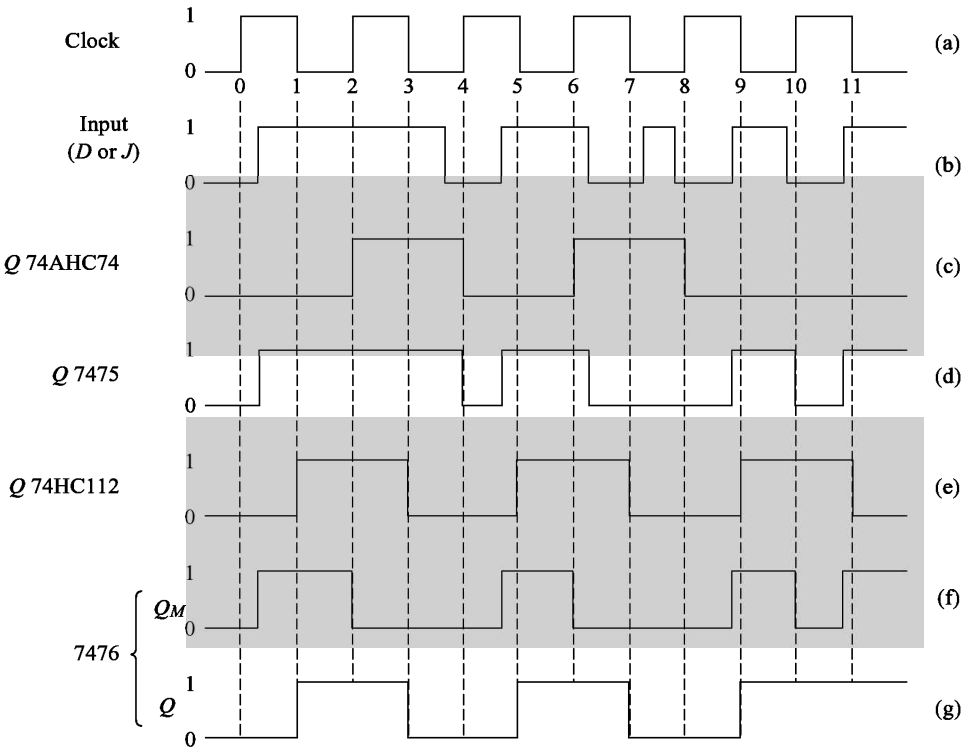


Fig. 7.22 ***Waveforms of Ex. 7.3***

Solution

(a) In the case of positive-edge-triggered *D*-type FLIP–FLOP the output *Q* is same as the input *D* at the positive edge of the clock pulse. The output does not change till the next positive edge arrives. The output (*Q*) waveform is shown in Fig. 7.22c.
(b) 7475 is a transparent latch, i.e. the output (*Q*) follows the input (*D*) as long as *CK* = 1. The output does not change when *CK* = 0. The output (*Q*) waveform is shown in Fig. 7.22*d*.
(c) In the case of negative-edge-triggered *J–K* FLIP–FLOP the output (*Q*) responds to the *J* and *K* inputs present at

the negative edge of the clock pulse (according to *J–K* truth table). The output does not change till the arrival of the next negative edge. The output (Q) waveform is shown in Fig. 7.22*e*.

(d) In the case of the master–slave *J–K* FLIP-FLOP, the output of the master responds to the *J* and *K* inputs present when $CK = 1$ (according to *J–K* truth table). The output of master (Q_M) is shown in Fig. 7.22*f*.

The output of the slave (Q) follows Q_M at the negative edge of the clock-pulse. The output (Q) waveform is shown in Fig. 7.22*g*.

7.10 APPLICATIONS OF FLIP-FLOPs

Some of the common uses of FLIP-FLOPs are as:

1. Bounce elimination switch,
2. Latch,
3. Registers,
4. Counters,
5. Memory, etc.

Some examples of the uses of FLIP-FLOPs are given below.

7.10.1 Bounce-Elimination Switch

Mechanical switches are employed in digital systems as input devices by which digital information (0 or 1) is entered into the system. There is a very serious problem associated with these switches, viz. *switch-bouncing* (or *chattering*). When the arm of the switch is thrown from one position to another, it chatters or bounces several times before finally coming to rest in the position of contact. The bounce is the result of the spring-loaded impact of the switch throw contact and the pole contacts.

In a sequential circuit, if a 1 is to be entered through a switch, then the switch is thrown to the corresponding position. As soon as it is thrown to this position, the output is 1 but the output oscillates between 0 and 1 for some time due to make and break (bouncing) of the switch at the point of contact before coming to rest. This changes the output of the sequential circuit and creates difficulties in the operation of the system. This problem is eliminated by using bounce-elimination switches.

Example 7.4

Show that the circuit of Fig. 7.23*a* acts as a bounce-elimination (chatterless) switch.

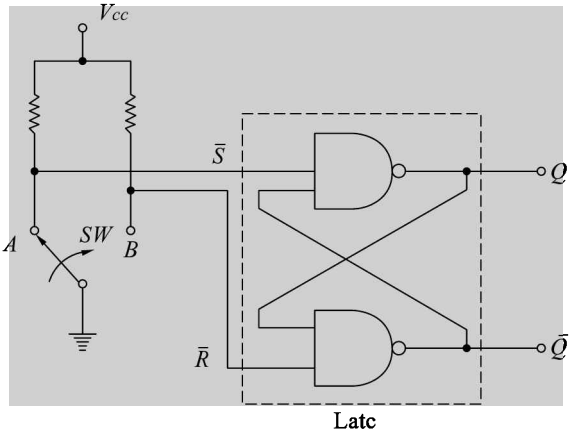
Solution

The waveforms at $\overline{S}$, $\overline{R}$, Q, and $\overline{Q}$ are illustrated in Fig. 7.23*b*. The switch (SW) is thrown from position *A* to *B* at $t = 0$. Therefore, at $t = 0^+$ the voltage at $\overline{S}$ will be V_{CC} (logic 1) and will continue to remain so as long as the switch is not thrown to position *A* again.

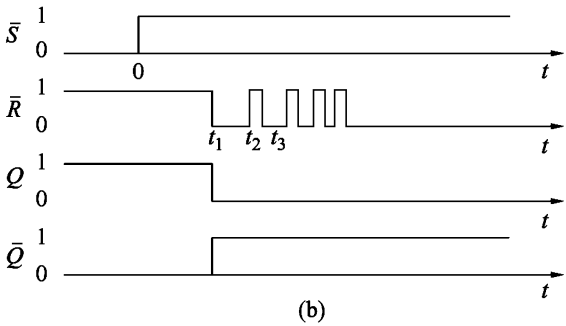
At $\overline{R}$ (B), the voltage at $t = 0^-$ is V_{CC} (logic 1) and goes to 0 V (logic 0) at t_1 (t_1 being the time delay of the switch). The switch arm makes contact at B at $t = t_1$, and then bounces off. Therefore, the level at $\overline{R}$ changes from 0 to 1 and vice-versa. This is illustrated in Fig. 7.23*b*.

Between $t = 0$ and $t = t_1$, both the inputs $\overline{S}$ and $\overline{R}$ are at logic 1 and therefore, Q does not change. The output Q changes at t_1 and becomes 0. Now, even when $\overline{R}$ is changing at t_2, t_3, etc. Q does not change. This shows that it is a chatterless switch.

The latch used in Fig. 7.23*a* can be replaced by the IC 74279 which is a quad $\overline{S} - \overline{R}$ latch.



(a)



(b)

Fig. 7.23 ***(a) A Bounce-Elimination Switch (b) Waveforms of $\overline{S}$, $\overline{R}$, Q, and $\overline{Q}$***

7.10.2 Registers

A register is composed of a group of FLIP-FLOPs to store a group of bits (word). For storing an N-bit word, the number of FLIP-FLOPs required is N (one FLIP-FLOP for each bit). A 3-bit register using 7474 positive-edge-triggered FLIP-FLOPs is shown in Fig. 7.24. The bits to be stored are applied at the D-inputs which are clocked in at the leading-edge of the clock pulse. In this register, the data to be entered must be available in parallel form. Other types of registers will be discussed in Chapter 8.

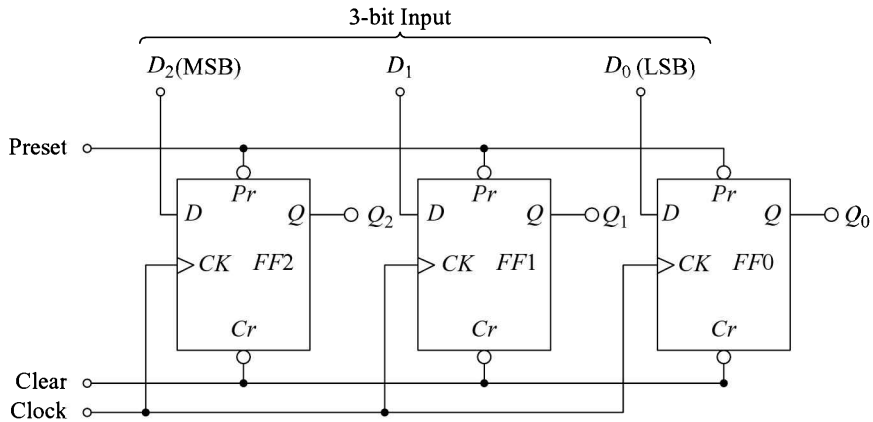


Fig. 7.24 ***A 3-bit Register Using*** `FLIP-FLOPs`

7.10.3 Counters

Digital counters are often needed to count events. For example, counting the number of tablets filled in a vial. Electrical pulses corresponding to the event are produced using a transducer and these pulses are counted using a counter.

The counters are composed of `FLIP-FLOPs`. A 3-bit counter consisting of three `FLIP-FLOPs` is shown in Fig. 7.25. A circuit with *n*- `FLIP-FLOPs` has 2^n possible states. Therefore, the 3-bit counter can count from decimal 0 to 7.

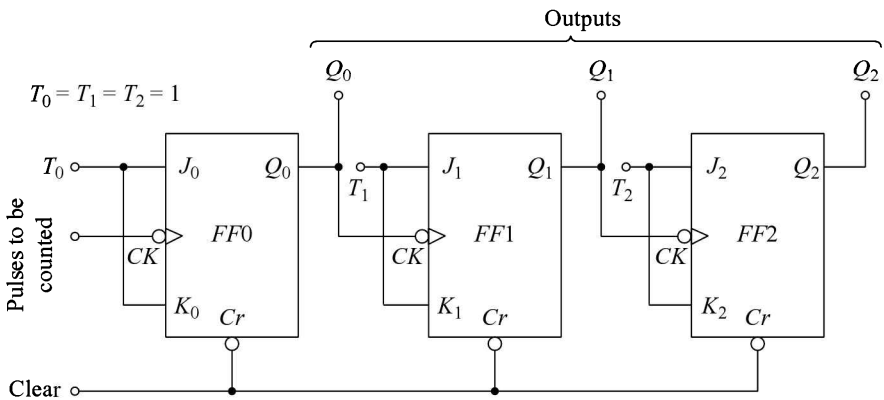


Fig. 7.25 ***A 3-bit Counter Using*** `FLIP-FLOPs`

The `FLIP-FLOPs` used are 74107 *J–K* master–slave `FLIP-FLOPs`, used as *T*-type. The pulses to be counted are connected at the clock input of FF0. The Q_0 output of FF0 is connected to the clock input of FF1 and similarly Q_1 is connected to the clock input of FF2.

The FLIP-FLOPs are cleared by applying logic 0 at the clear input terminal momentarily. For normal counting operation, it is to be maintained at logic 1. The pulses and the output waveforms are illustrated in Fig. 7.26.

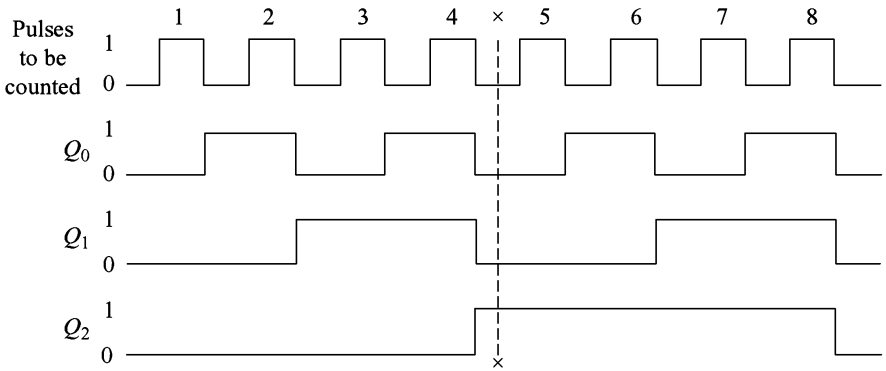


Fig. 7.26 ***Waveforms of Counter of Fig. 7.25***

The output Q_0 of the least-significant stage changes at the negative edge of each pulse (since $T_0 = 1$). The output Q_1 changes at the negative edge of each Q_0 pulse (since Q_0 acts as *CK* for FF1 and $T_1 = 1$) and the output Q_2 changes at the negative edge of each Q_1 pulse (since Q_1 acts as *CK* for FF2 and $T_2 = 1$).

At any time, the decimal equivalent of the binary number $Q_2 Q_1 Q_0$ is the number of pulses counted till that time. For example, at × the count is 100 (decimal 4). The circuit resets after counting eight pulses.

There are various types of counters, some of which will be discussed in the next chapter.

7.10.4 Random-Access Memory

In computers, digital control systems, information processing systems, etc. it is necessary to *store* digital data and *retrieve* the data as desired. For this purpose, earlier only magnetic memory devices were possible, whereas these days it has become possible to make memory devices using semiconductor devices. Semiconductor memories have become very popular because of their small size (available in ICs) and convenience to use. Chapter 11 deals with various semiconductor memories.

FLIP-FLOPs can be used for making memories in which data can be stored for any desired length of time and then read out whenever required. In such a memory, data can be put into (*writing* into the memory) or retrieved from (*reading* from the memory) the memory in a random fashion and is known as *random-access memory*.

A 1-bit read/write memory is shown in Fig. 7.27 which is the basic memory element and memory ICs are built around a system of basic 1-bit cell.

In this memory cell, a level *D* FLIP-FLOP is used which has *Q* output that follows the *D* input as long as *CK* terminal is at logic 1. The moment the *CK* input changes to logic 0, the *Q* output does not change and it retains the *D* input level that existed just before the transition from 1 to 0 at input *CK*. This input is used to select the memory cell. In the 1-bit cell shown there are three inputs — D_i (data input), A_n (address select) and $R/\overline{W}$ (read/write control) and one output D_0 (data output). $A_n = 1$ enables the cell for reading or writing

operation, $R/\overline{W}$ at logic 1 is for reading from the cell and logic 0 for writing into the cell. As long as $A_n = 0$, all input and output activities are blocked, and the cell is in the hold mode where its stored output is protected.

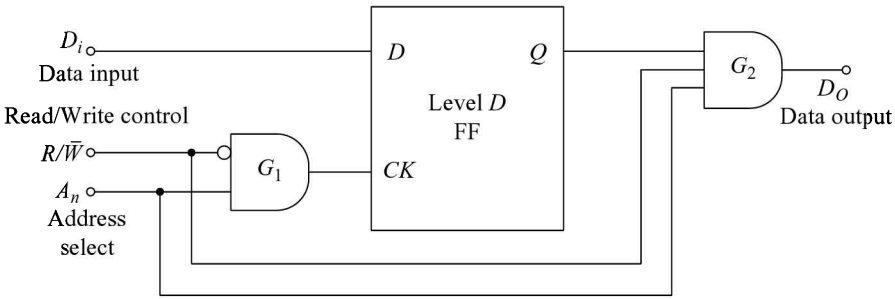


Fig. 7.27 ***A 1-bit Read/Write Memory Cell***

The complete function of this cell can be understood from the function table of Table 7.10. The read operation is nondestructive, that is, the stored bit can be read out any number of times without disturbing it. The stored bit will be protected as long as power is on. Therefore, this type of memory is known as *volatile memory*.

Table 7.10 ***Function Table of 1-bit Memory Cell***

Inputs			Mode
A_n	$R/\overline{W}$	D_i	
0	×	×	Hold, $D_0 = 0$
1	0	0	Write 0 into memory, $D_0 = 0$
1	0	1	Write 1 into memory, $D_0 = 0$
1	1	×	Read, D_0 = stored D_i bit.

As far as writing into the cell is concerned, it is not required to be cleared before entering the new bit. Whenever a new bit is entered the earlier one gets destroyed automatically.

SUMMARY

The basic element of sequential circuits, FLIP-FLOP has been introduced here. This is a basic memory element which is used to store 1-bit of digital information. The four types of commonly used FLIP-FLOPs *S–R*, *J–K*, *T*-type, and *D*-type have been covered in detail, including their design using gates.

The various triggering systems have been discussed which will be very helpful in understanding the detailed operation of the FLIP-FLOP and other circuits containing these FLIP-FLOPs.

Simple examples of the uses of FLIP-FLOPs in registers, counters, memory elements, etc. have been included. These topics will be covered in greater detail in later chapters.

GLOSSARY

Asynchronous sequential circuit A sequential circuit whose behaviour depends upon the sequence in which the input signals change. It is event driven and does not require clock pulses.

Bit time Amount of time to transmit a single bit.

Bouncing Moving back and forth between two states before reaching a final state. Same as chattering.

Bounce-elimination circuit A circuit that eliminates the effect of switch bouncing.

Characteristic table A table describing operation of a FLIP-FLOP.

Chatterless switch A switch in which the bouncing effect has been eliminated.

Clear Setting the contents of a FLIP-FLOP or a circuit containing FLIP-FLOPs or a memory to zero.

Clear input The input used for clearing a digital circuit.

Clock A train of pulses of usually constant frequency that synchronize the operation of any synchronous sequential circuit including a microprocessor based system.

Clock cycle The interval between successive positive or negative transitions in a clock.

Clocked FLIP-FLOP A FLIP-FLOP that responds to the data inputs only on the occurrence of the appropriate clock signal.

Clock frequency The number of clock cycles per second.

Clocked sequential circuit The sequential circuits whose operation is synchronized with the application of clock pulses, between which no changes of state occur.

Counter A digital circuit that can count the number of pulses.

Data Information in digital (binary) form.

Debouncing switch Same as chatterless switch.

D-type FLIP-FLOP A FLIP-FLOP whose output follows the input when triggered.

Edge-triggered FLIP-FLOP A FLIP-FLOP whose state changes on the rising (positive) or falling (negative) edge of a clock pulse.

Excitation table A tabular representation of the operation of a FLIP-FLOP.

Falling edge A transition from high to low voltage.

Frequeucy The repetition rate of a cyclic signal. It is expressed in Hertz (Hz).

Hold time In FLIP-FLOPs and memories, a minimum amount of time after the application of a clock (or Read/write) signal, when data(s) or address inputs must remain stable.

J–K FLIP-FLOP A FLIP-FLOP with inputs J and K. The state of the FLIP-FLOP does not change when $J = K = 0$, whereas it changes with every clock pulse when $J = K = 1$. The FLIP-FLOP is set when $J = 1$ and $K = 0$ and reset when $J = 0$ and $K = 1$. All the above operations are performed in synchronism with the clock.

Latch A temporary storage device consisting of D-type FLIP-FLOPs. Its contents are fixed at their current values by a transition of the clock and remain fixed until the next clock transition occurs.

Leading edge A transition from low to high voltage.

Level-triggereed FLIP-FLOP A FLIP-FLOP that is triggered (i.e. the outputs respond to the inputs) when the level of the clock signal is appropriate.

Master-slave FLIP-FLOP A FLIP-FLOP consisting of the cascade of two FLIP-FLOPs; the first FF is the master and the second FF is the slave. The master FF is triggered when the clock is HIGH and the slave FF follows the master FF when the clock goes LOW.

Preset The state of a FLIP-FLOP when $Q = 1$ and $\bar{Q} = 0$

Preset input The input used for setting a FLIP-FLOP.

Pulse-triggered FLIP-FLOP A FLIP-FLOP that requires a clock pulse for triggering. It is same as the master-slave FF.

Random-access memory (RAM) The ability to address a semiconductor memory for read-and-write operation in which any memory location can be accessed at random. It is same as Read and Write memory.

Read (memory) Process of getting (retrieving) a word from a memory.

Register An array of FLIP-FLOPs used to store binary information.

Reset Same as clear.

Reset input Same as the clear input

Rising edge Same as the leading edge.

Set $Q = 1$ state of a FLIP-FLOP.

Setup time The time required for the input data to settle in before the triggering edge of the clock pulse.

S–R FLIP-FLOP A FLIP-FLOP with inputs S and R. The state of the FF does not change when $S = R = 0$. It is set when $S = 1$ and $R = 0$; reset when $S = 0$ and $R = 1$. $S = R = 1$ is not allowed.

Stable state A state in which a digital circuit remains forever unless changed by a triggering signal.

State The value of FLIP-FLOP(s) output in a digital circuit.

Switch bouncing Fluctuations in the switch positions between ON and OFF when the switch position is changed.

Synchronous sequential circuit A sequential circuit whose operation is synchronized with the application of clock pulses, between which no change of state can occur.

Toggling Causing as FF to change its state.

Trailing edge Same as the falling edge.

Trigger To cause change of state.

T-type *FLIP-FLOP* A FLIP-FLOP with one data input (T) which changes state for every clock pulse if $T = 1$ and does not change the state if $T = 0$.

Unstable state The state which is not a stable state. The circuit comes out of this state without applying any triggering signal.

Write (memory) Process of placing (storing) a word into a specific memory location.

REVIEW QUESTIONS

7.1 FLIP-FLOP is a _____ element.

7.2 Number of FLIP-FLOPs required for storing n-bit of information is _____.

7.3 In an S–R FLIP-FLOP $S = R = 1$ _____ permitted.

7.4 'Preset' and 'Clear' inputs are used in a FLIP-FLOP for making Q = _____ and _____ respectively.
7.5 In a J–K FLIP-FLOP if $J = K = 1$, its Q output will be _____ when a clock pulse is applied.
7.6 Master-slave configuration is used in a J–K FLIP-FLOP to eliminate _____.
7.7 In a T FLIP-FLOP, the Q output _____ when $T = O$ and clock pulse is applied.
7.8 A FLIP-FLOP has _____ states.
7.9 A latch is used to store 1 _____ of data.
7.10 A negative edge-triggered FLIP-FLOP changes state at the _____ of the clock pulse.
7.11 An active low 'Clear' input clears the FLIP-FLOP when it is _____.
7.12 A FLIP-FLOP with active-low 'preset' input will have $\overline{Q}$ = _____ when preset is connected to LOW.
7.13 A chatterless switch can be implemented using a _____.
7.14 A tabulation specifying inputs required for a FLIP-FLOP to change from a present state to a specified next state is known as _____.
7.15 Registers and Counters can be designed using _____.

PROBLEMS

7.1 Show that the circuit of Fig. 7.4 with $S = R = 0$ is the same as that of Fig. 7.3.
7.2 In a circuit of Fig. 7.4 if the inputs change from
(a) $S = 1$, $R = 0$ to $S = R = 0$, and
(b) $S = 0$, $R = 1$ to $S = R = 0$,
show that the outputs do not change.
7.3 Design an S–R latch using two 2-input NOR gates
7.4 In the FLIP-FLOP circuit of Fig. 7.28 show that if
(a) $Pr = 0$ and $Cr = 1$, then $Q = 1$ (independent of S, R, and CK).
(b) $Pr = 1$ and $Cr = 0$, then $Q = 0$ (independent of S, R, and CK).

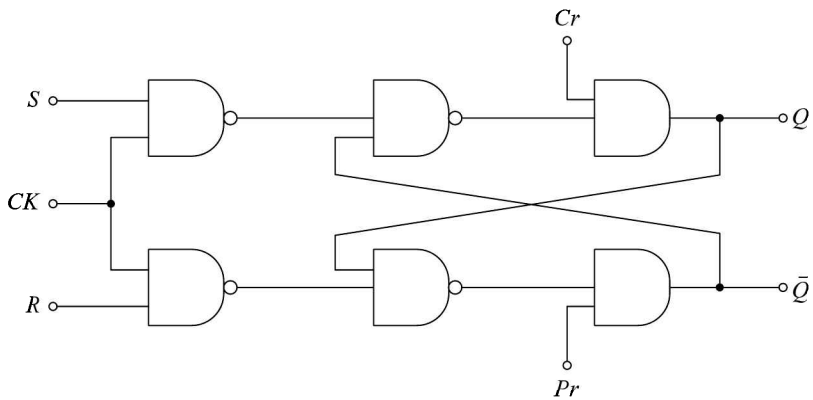


Fig. 7.28 FLIP-FLOP ***Circuit for Problem 7.4***

(c) $Pr = Cr = 1$, then it functions as a clocked *S–R* FLIP-FLOP.

7.5 Verify Table 7.3a.

7.6 Determine the output Y_1 of Fig. 7.29*a* and Y_2 of Fig. 7.29*b* and show that $Y_1 = Y_2$.

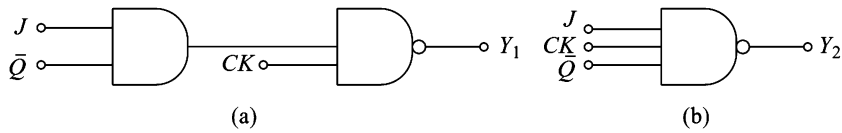


Fig. 7.29 ***Figures for Problem 7.6***

7.7 Identify Q and $\bar{Q}$ outputs of the clocked *J–K* FLIP-FLOP shown in Fig. 7.30.

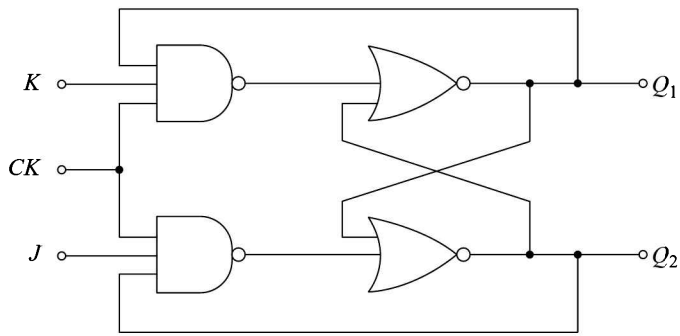


Fig. 7.30 ***Figure for Problem 7.7***

7.8 For the circuit shown in Fig. 7.31 the clock and input waveforms shown in Fig. 7.32 are applied. Sketch the waveforms of Q and $\bar{Q}$ if the FLIP-FLOP is edge-triggered.

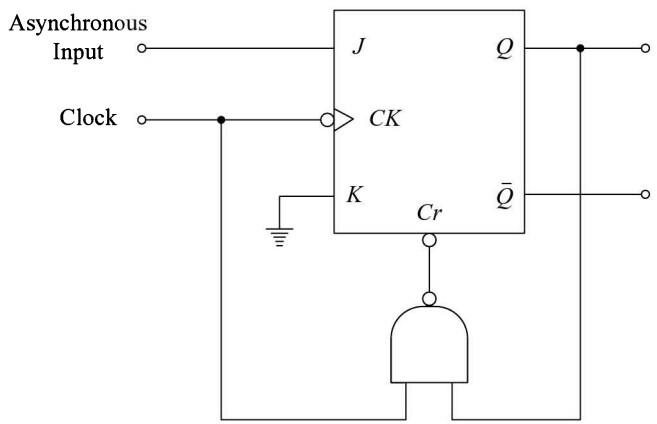


Fig. 7.31 ***Figure for Problem 7.8***

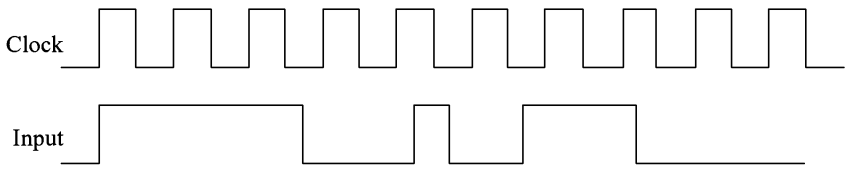


Fig. 7.32 ***Waveforms for Problem 7.8***

7.9 Repeat Problem 7.8 if the FLIP-FLOP is master–slave.

7.10 If the waveforms shown in Fig. 7.33 are applied to the circuit of Fig. 7.31, sketch the output (Q) waveform. Assume *M–S* FLIP-FLOP.

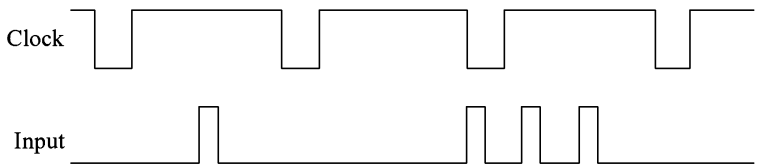


Fig. 7.33 ***Waveforms for Problem 7.10***

7.11 Verify that the circuit of Fig. 7.16 acts as a toggle switch.

7.12 Prepare the truth table for the circuit of Fig. 7.34 and show that it acts as a *T*-type FLIP-FLOP.

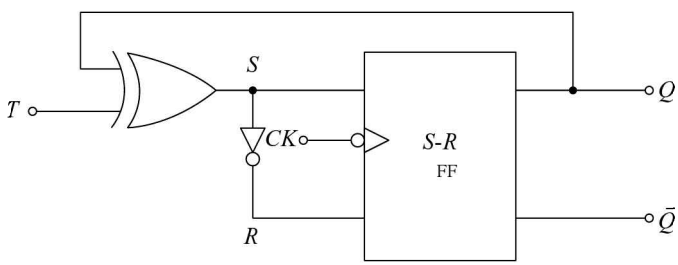


Fig. 7.34 ***Circuit for Problem 7.12***

7.13 If $\bar{Q}$ output of a *D*-type FLIP-FLOP is connected to D input, it acts as a toggle switch. Verify.

7.14 Using the method outlined in Section 7.8, design the following FLIP-FLOPs:
(a) *J–K*
(b) *D*-type
(c) *T*-type

7.15 Using the conversion method outlined in Section 7.8, carry out the following conversions:

(a) *S–R* to *D*
(b) *J–K* to *D*
(c) *D* to *J–K*
(d) *S–R* to *T*
(e) *J–K* to *T*
(f) *T* to *J–K*
(g) *T* to *D*
(h) *D* to *S–R*
(i) *D* to *T*
(j) *T* to *S–R*
(k) *J–K* to *S–R*

7.16 Figure 7.35 shows a positive-edge-triggered *D* `FLIP-FLOP`. Verify its operation.

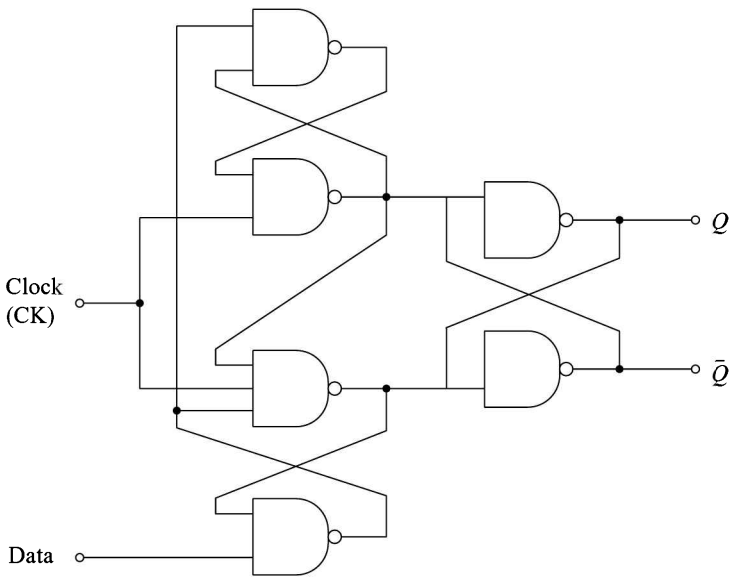


Fig. 7.35 ***Positive-edge-triggered*** *D* `FLIP-FLOP`.

7.17 IC 7411I is a master–slave *J–K* `FLIP-FLOP` with data lock out at the positive edge of the clock (the state changes at the negative edge of the clock).

If the waveform shown in Fig. 7.22*a* is applied at the clock input and the waveform of 7.22*b* is applied at the *J* input, sketch the output (*Q*) waveform when
(a) *K* is connected to logic 0.
(b) *K* is connected to logic 1.

7.18 Verify the operation of the debounce switches shown in Fig. 7.36.

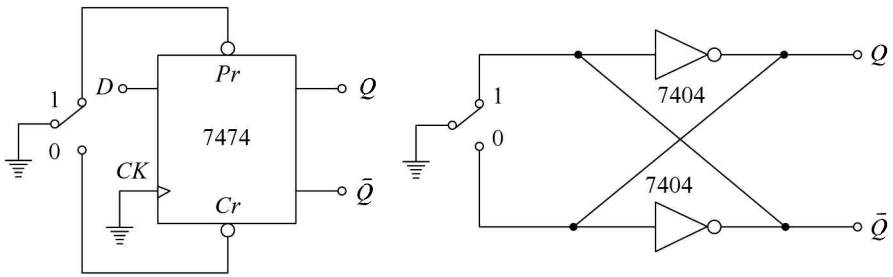


Fig. 7.36 ***Debounce Switches for Problem 7.18***

7.19 Consider the circuit of Fig. 7.37 consisting of positive-edge-triggered FLIP-FLOPs. Δt_1 and Δt_2 are the time delays introduced in the clock (CLOCK SKEW) by buffer devices and the propagation delay of wires. At the rising edge of the clock, new data enters the source FF and the content of source FF enters the destination FF. Show the effect of clock skew ($\Delta t_2 > \Delta t_1$) on the operation of the circuit. Discuss the problems created by clock skew for data transmission.

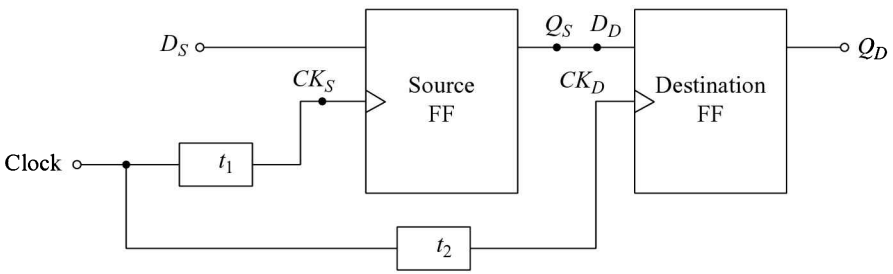


Fig. 7.37 ***Circuit for Problem 7.19***

7.20 A mod-3 counter (resets after every three pulses) is shown in Fig. 7.38. The FLIP-FLOPs used are master–slave *J–K*. Sketch the waveforms of Q_0 and Q_1 when clock pulses are applied and verify its operation. Assume $Q_0 = Q_1 = 0$ initially.

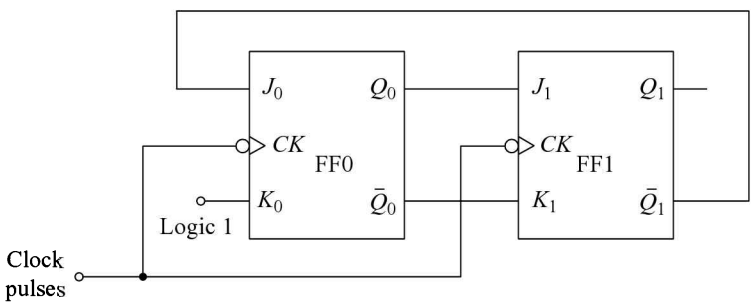


Fig. 7.38 ***A Mod-3 Counter***

7.21 In the circuit shown in Fig. 7.39, the time constant (RC) is very small. Explain the operation of this circuit.

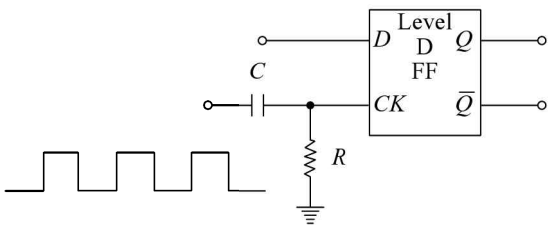


Fig. 7.39 ***Circuit for Problem 7.21***

CHAPTER 8

SEQUENTIAL LOGIC DESIGN

8.1 INTRODUCTION

The FLIP-FLOP is a basic element of sequential logic system. Using FLIP-FLOPs and combinational logic circuits, any sequential logic circuit can be designed. The most important sequential circuits that are widely used in digital systems are registers and counters. These are discussed in detail in the following sections.

The design of registers and counters using FLIP-FLOPs has been introduced followed by standard MSI devices. Some of the common applications of registers and counters have also been discussed.

The design methods for general clocked sequential circuits have also been discussed. Basically, the system may be specified in terms of input–output relationship or/and the sequence of states to be followed when clock pulses are applied. The usual design steps are: reduction of states, state assignment and next-state decoder design.

The synchronous counter is a special case of the general clocked sequential circuit and can be designed using the design methods developed for clocked sequential circuits.

Asynchronous sequential circuits do not use clock pulses and their response depends upon the sequence in which the input signal changes. These circuits use delay circuits as memory elements. The delay is inherent in logic circuits and the feedback introduced in combinational circuits provides memory capability to these circuits. The S-R FLIP-FLOPs fall under this category of delay circuits which are extensively used in asynchronous sequential circuits. The analysis and design of asynchronous sequential circuits have been discussed in detail.

8.2 REGISTERS

As discussed in Chapter 7, a FLIP-FLOP can store (or remember) 1-bit of digital information (1 or 0). It is also referred to as a 1-bit *register*. An array of FLIP-FLOPs is required to store binary information, the number of FLIP-FLOPs required being equal to the number of bits in the binary word (one FLIP-FLOP for each bit) and is referred to as a register. Registers find application in a variety of digital systems including microprocessors. For example, Intel's 8085 microprocessor chip contains seven 8-bit registers, referred to as *general purpose registers* and five 1-bit registers, referred to as *flags*.

The data can be entered in *serial* (one-bit at a time) or in *parallel* form (all the bits simultaneously) and can be retrieved in the serial or parallel form. Data in serial form is also referred to as *temporal code* (a time arrangement of bits) and in parallel form as *spacial code*. A 4-bit data 1010 in serial form is shown in Fig. 8.1*a* and in parallel form in Fig. 8.1*b*. For serial input/output, only one line is required for data input and one line for data output, whereas the number of lines required is equal to the number of bits for parallel input/output.

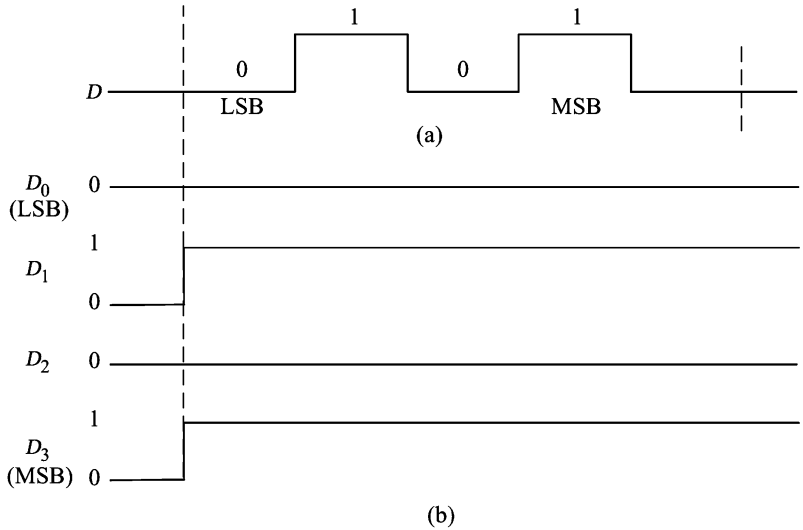


Fig. 8.1 ***Data Representation in (a) Serial Form (b) Parallel Form***

Registers are classified depending upon the way in which data are entered and retrieved. There are four possible modes of operation:

1. Serial-in, serial-out (SISO),
2. Serial-in, parallel-out (SIPO),
3. Parallel-in, serial-out (PISO), and
4. Parallel-in, parallel-out (PIPO).

Registers can be designed using discrete FLIP-FLOPs (*S-R* or *J-K* as *D*-type) and are also available as MSI devices. The registers available in 54/74 TTL and CMOS logic families are given in Table 8.1. The complete IC No. is different for different series. For example, 74AC164, 74ACT164, 74HC164, 74HCT164, 74ALS164A etc.

Registers in which data are entered or/and taken out in serial form are referred to as *shift registers*, since bits are shifted in the FLIP-FLOPs with the occurrence of clock pulses either in the right direction (*right-shift register*) or in the left direction (*left-shift register*). In the *bi-directional shift register*, data can be shifted from left to right as well as in the reverse direction, using the mode control. For example, IC 74295A is a bi-directional shift register.

A register is referred to as a *universal register* if it can be operated in all the four possible modes and also as a bi-directional register. For example, 74194 is a universal register.

Table 8.1 ***Shift Registers Available in* 54/74 TTL *and* CMOS *Families***

IC No.	Description
7491, 7491A	8-bit serial-in, serial-out
7494	4-bit parallel-in, serial-out
7495	4-bit serial/parallel-in, parallel-out (right-shift, left-shift)
7496	5-bit parallel-in/parallel-out, serial-in/serial-out
7499	4-bit bi-directional (universal)
74164	8-bit serial-in, parallel-out
74165	8-bit serial/parallel-in, serial-out
74166	8-bit serial/parallel-in, serial-out
74178, 74179	4-bit bi-directional (universal)
74194	4-bit bi-directional (universal)
74195	4-bit serial/parallel-in, parallel-out
74198	8-bit bi-directional (universal)
74199	8-bit serial/parallel-in, parallel-out
74295A	4-bit TRI-STATE serial/parallel-in, parallel-out bi-directional
74395	4-bit TRI-STATE cascadable serial/parallel-in, serial/parallel-out.

8.2.1 Shift Register

A 5-bit shift register using five master-slave *S-R* (or *J-K*) `FLIP-FLOPs` is shown in Fig. 8.2. This circuit can be used in any of the four modes. The operation of this circuit is explained by assuming the 5-bit data 10110. For any other 5-bit data, the operation will be similar to the one explained.

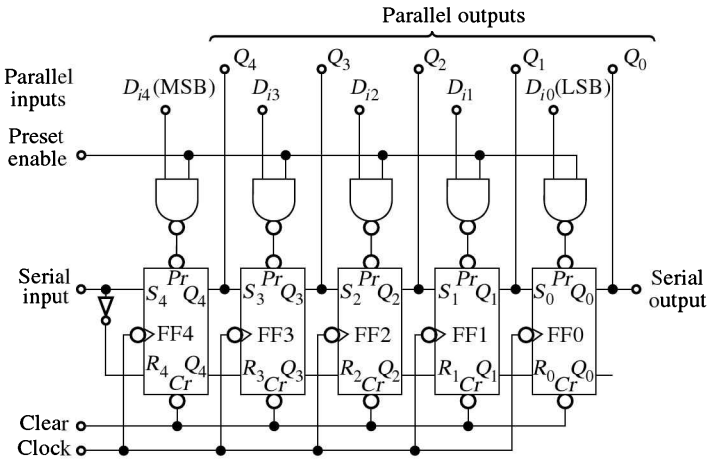


Fig. 8.2 ***A 5-bit Shift Register* (7496)**

Serial Input

The data word in the serial form (Fig. 8.1*a*) is applied at the serial input after clearing the `FLIP-FLOPs` using the clear line. The preset enable is to be held at 0 so that *Pr* for every `FLIP-FLOP` is 1. The input and output waveforms are illustrated in Fig. 8.3.

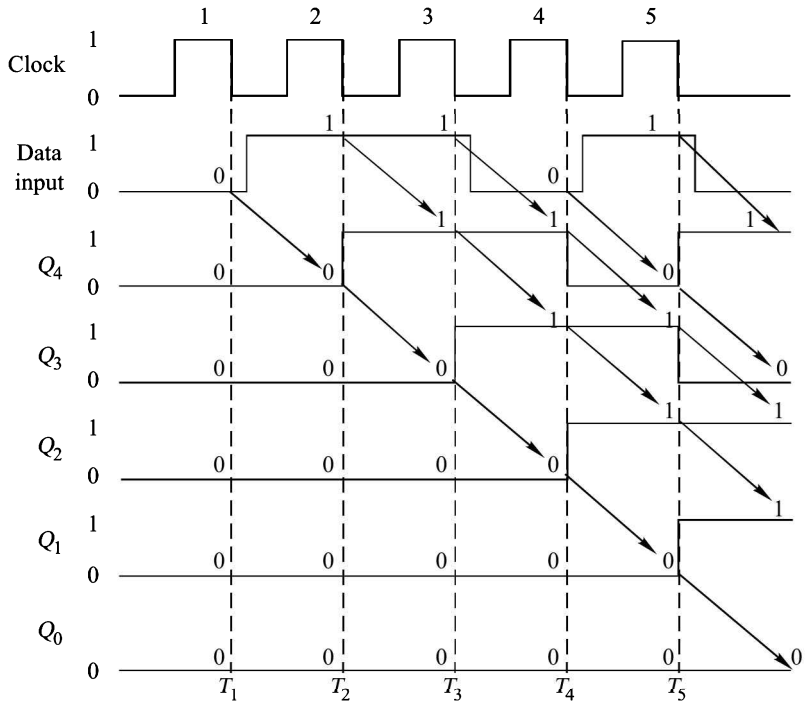


Fig. 8.3 ***Waveforms of Shift Register for Serial Input***

The process of entering the digital word starts with the data input corresponding to the least-significant bit (0) at the serial input and first clock pulse. At the falling edge (T_1) of the first clock pulse the output of FF4 (Q_4) will be 0 and the outputs of all other FLIP-FLOPs are 0 since their inputs are 0. Next, the input corresponding to next bit is applied and, at the falling edge (T_2) of the second clock pulse, the FLIP-FLOP outputs will be

$$Q_4 = 1$$
$$Q_3 = Q_2 = Q_1 = Q_0 = 0$$

Similarly, the input corresponding to each bit is applied till the MSB and the bits go on shifting from left to right at the falling edge of each clock pulse as illustrated in Fig. 8.3. At the end of fifth clock pulse, the outputs of FLIP-FLOPs are

$$Q_4 = 1$$
$$Q_3 = 0$$
$$Q_2 = 1$$
$$Q_1 = 1$$
$$Q_0 = 0$$

which is the same as the number to be stored. The number of clock pulses required for entering the data, is the same as the number of bits. The process of entering the data is also referred to as *writing* into the register.

The data stored can be retrieved (also referred to as *reading*) in two ways: serial-out and parallel-out. The data in the serial form is obtained at Q_0 when clock pulses are applied. The number of clock pulses required will be same as the number of bits (five in this case).

In the parallel form, the data is available at $Q_4\,Q_3\,Q_2\,Q_1\,Q_0$ and clock is not required for reading. In the case of serial output, after the nth clock pulse, for an n-bit word, each FLIP-FLOP output is 0. This means that once the data is retrieved the register is empty. On the other hand, in the case of parallel output, the contents of the register can be *read* any number of times until new data is stored in the register.

The clock rate may be different for the input data and the output data in case of a serial-in, serial-out shift register. Hence, this method can be used for changing the spacing in time of a binary code which is referred to as *buffering*.

Parallel Input

Data can be entered in the parallel form making use of the preset inputs. After clearing the FLIP-FLOPs, if the data lines are connected to the parallel inputs (D_{i4}, D_{i3}, D_{i2}, D_{i1}, and D_{i0}) and a 1 is applied at the preset input, the data are written into the register. This is referred to as *asynchronous loading*.

As explained above, the stored word may be read in the serial form at Q_0 by applying five clock pulses or in the parallel form at Q outputs.

The data can also be entered in parallel form by using D-type FLIP-FLOPs connected as shown in Fig. 7.24. In this, the data is loaded when a clock pulse is applied and hence, it is referred to as *synchronous loading*.

Bi-directional Register

There are applications in which shifting data to the right and/or to the left is required. For example, a binary number can be divided by two by shifting it one stage to the right. In this process the least-significant bit is lost (unless additional circuitry is used to preserve it) causing an error of 0.5 if the number is odd. Similarly, a number stored in a shift register can be multiplied by two by shifting it one stage to the left, provided a 1 is not shifted out of the most-significant stage. A 4-bit bi-directional shift register is shown in Fig. 8.4.

When the mode control $M = 1$, all the A AND gates are enabled and the data at D_R is shifted to the right when clock pulses are applied. On the other hand, when $M = 0$, the A gates are inhibited and B gates are enabled allowing the data at D_L to be shifted to the left. M should be changed only when $CK = 0$, otherwise the data stored in the register may be altered.

8.3 APPLICATIONS OF SHIFT REGISTERS

The primary uses of shift registers are temporary data storage and bit manipulations. Some of the common applications of shift registers are discussed below:

8.3.1 Delay Line

A SISO shift register may be used to introduce time delay Δt in digital signals given by

$$\Delta t = N \times \frac{1}{f_C} \tag{8.1}$$

where N is the number of stages and f_C is the clock frequency.

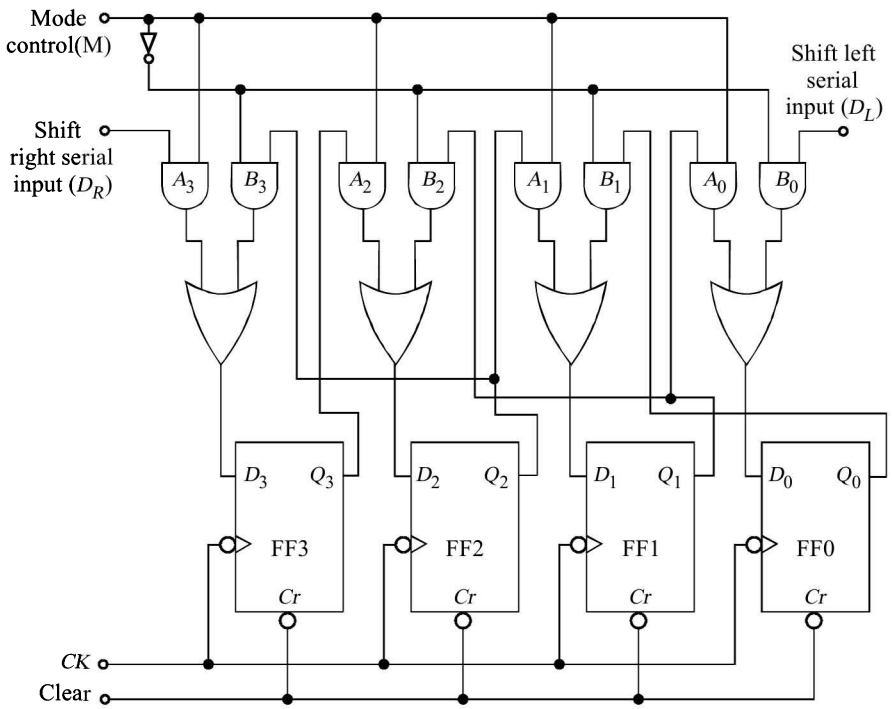


Fig. 8.4 ***A 4-bit Bi-directional Shift Register***

Thus, an input pulse train appears at the output delayed by Δt. The amount of delay can be controlled by the clock frequency or the number of FLIP-FLOPs in the shift register.

8.3.2 Serial-to-Parallel Converter

Data in the serial form can be converted into parallel form by using a SIPO shift register.

8.3.3 Parallel-to-Serial Converter

Data in the parallel form can be converted into serial form by using the PISO shift register.

8.3.4 Ring Counter

If the serial output Q_0 of the shift register of Fig. 8.2 is connected back to the serial input, then an injected pulse will keep circulating. This circuit is referred to as a *ring counter*. The pulse is injected by entering 00001 in the parallel form after clearing the FLIP-FLOPs. When the clock pulses are applied, this 1 circulates around the circuit. The waveforms at the Q outputs are shown in Fig. 8.5. The outputs are sequential non-overlapping pulses which are useful for control-state counters, for stepper motor (which rotates in steps) which require sequential pulses to rotate it from one position to the next, etc.

This circuit can also be used for counting the number of pulses. The number of pulses counted is read by noting which FLIP-FLOP is in 1 state. No decoding circuitry is required. Since there is one pulse at the output for each of the N clock pulses, this circuit is referred to as a *divide-by-N counter* or an *N : 1 scalar*.

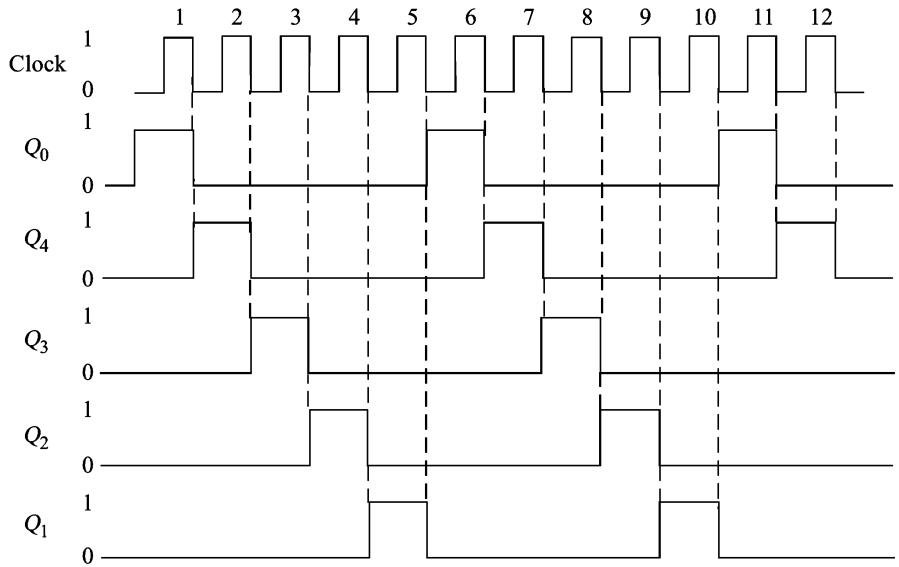


Fig. 8.5 ***Output Waveforms of Ring Counter***

8.3.5 Twisted-Ring Counter

In the shift register of Fig. 8.2, if $\overline{Q}_0$ is connected to the serial input, the resulting circuit is referred to as a *twisted-ring, Johnson*, or *moebius* counter. If the clock pulses are applied after clearing the FLIP-FLOPs, square waveform is obtained at the Q outputs as shown in Fig. 8.6.

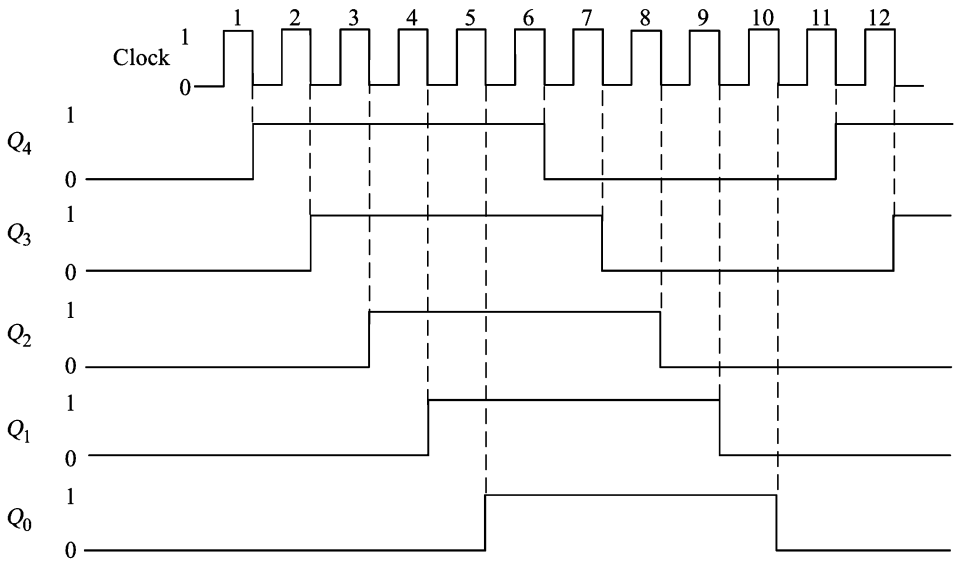


Fig. 8.6 ***Output Waveforms of Twisted-Ring Counter***

Similar to ring-counter sequence, the moebius sequence is also useful for control-state counters. It is also useful for the generation of multiphase clock.

The moebius counter is a divide-by-2N counter. For decoding the count, two- input AND gates are required. The decoder circuit for a five-stage counter is shown in Fig. 8.7.

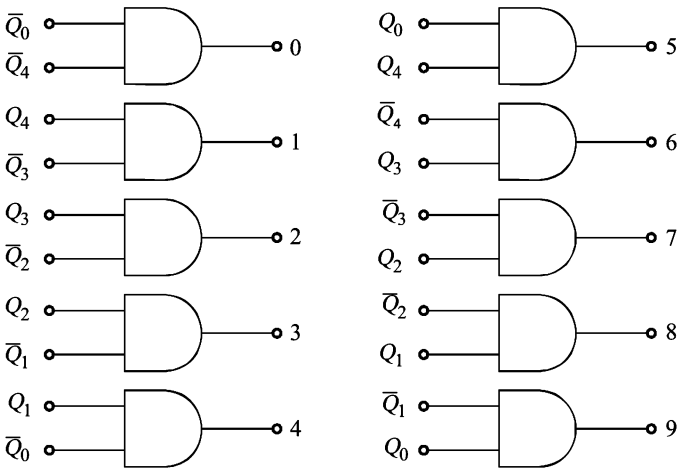


Fig. 8.7 ***The Decoding Logic for a 5-Stage Twisted-Ring Counter***

8.3.6 Sequence Generator

A circuit which generates a prescribed sequence of bits, in synchronism with a clock, is referred to as a sequence generator. Such generators can be used as

1. Counters,
2. Random bit generators,
3. Prescribed period and sequence generators, and
4. Code generators.

The basic structure of a sequence generator is shown in Fig. 8.8.

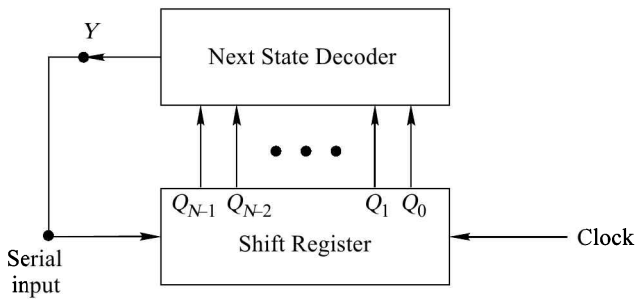


Fig. 8.8 ***Basic Structure of a Sequence Generator***

The output Y of the next state decoder is a function of $Q_{N-1}\,Q_{N-2} \ldots Q_1\,Q_0$. This system is similar to a ring counter ($Y = Q_0$) or a twisted-ring counter ($Y = \overline{Q}_0$) which are special cases of sequence generators. The design of the decoder will be clear from Example 8.1.

Example 8.1

Design a sequence generator to generate the sequence … 1101011…

Solution

The minimum number of FLIP-FLOPs, N, required to generate a sequence of length S is given by

$$S \leq 2^N - 1 \tag{8.2}$$

In this case $S = 7$, therefore, the minimum value of N, which may generate this sequence is 3. However, it is not guaranteed to lead to a solution. If the given sequence leads to seven distinct states, then only three FLIP-FLOPs are sufficient otherwise we have to increase the number of FLIP-FLOPs. We write the states of the circuit as given in Table 8.2. The prescribed sequence is listed under Q_2 and the sequence listed under Q_1 and Q_0 are the same sequence delayed by one and two clock pulses respectively. From the table we observe that all the states are not distinct, which means $N = 3$ is not sufficient. Next we assume $N = 4$ and prepare Table 8.3 in a similar manner as Table 8.2. The last column gives the required serial input for getting the desired change of state when a clock pulse is applied. This is obtained by assuming D-type FLIP-FLOPs and looking at the Q_3 output. For example, at the falling edge of the first clock pulse, $Q_3 = 1$. The second clock pulse must result in $Q_3 = 1$ which requires its D input to be 1. In the same manner, all the entries in column Y are determined. The K-map of Table 8.3 is given in Fig. 8.9 and the simplified expression is given by

$$Y = \overline{Q}_3 + \overline{Q}_1 + \overline{Q}_0 \tag{8.3}$$

Table 8.2 ***State Table of Sequence Generator* (N = 3)**

Number of clock pulses	FLIP-FLOP outputs		
	Q_2	Q_1	Q_0
1	1	1	1
2	1	1	1
3	0	1	1
4	1	0	1
5	0	1	0
6	1	0	1
7	1	1	0

Table 8.3 ***Truth Table of Sequence Generator* (N = 4)**

Number of clock pulses	FLIP-FLOP outputs				Serial input
	Q_3	Q_2	Q_1	Q_0	Y
1	1	1	1	0	1
2	1	1	1	1	0
3	0	1	1	1	1

(*Continued*)

Table 8.3 ***(Continued)***

Number of clock pulses	FLIP-FLOP outputs				Serial input
	Q_3	Q_2	Q_1	Q_0	Y
4	1	0	1	1	0
5	0	1	0	1	1
6	1	0	1	0	1
7	1	1	0	1	1
1	1	1	1	0	1
2	1	1	1	1	0
3	0	1	1	1	1
*	1	0	1	1	0
*	0	1	0	1	1
*	1	0	1	0	1

The sequence generator circuit can be designed using a 4-stage shift register using *D*-type FLIP-FLOPs and the decoder circuit given by Eq. (8.3).

8.4 RIPPLE OR ASYNCHRONOUS COUNTERS

A circuit used for counting the pulses is known as a *counter*. In Sec. 8.3, two types of counters have been discussed. The number of states in an N-stage ring counter is N, whereas it is $2N$ in the case of moebius counter. These counters are referred to as modulo N (or divide-by-N) and modulo $2N$ (or divide-by-$2N$) counters respectively, where modulo indicates the number of states in the counter. When the pulses to be counted are applied to a counter, it goes from state to state and the output of the FLIP-FLOPs in the counter is decoded to read the count. The circuit comes back to its starting state after counting N pulses in the case of modulo N counter.

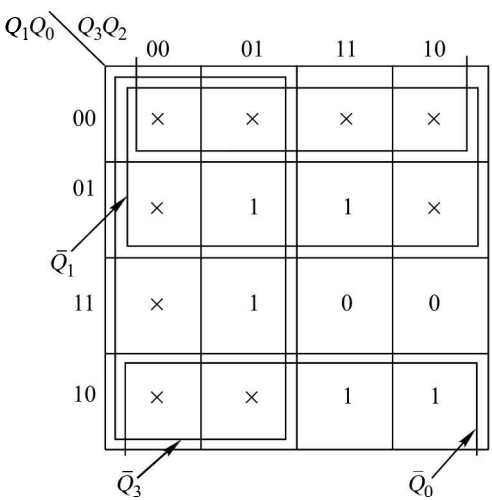


Fig. 8.9 ***K-Map of Table 8.3***

The ring counter and the twisted-ring counter do not make efficient use of FLIP-FLOPs. A FLIP-FLOP has two states, therefore a group of N FLIP-FLOPs will have 2^N states. This means it is possible to make a modulo 2^N counter using N FLIP-FLOPs. Basically there are two types of such counters:

1. Asynchronous counter (ripple counter), and
2. Synchronous counter.

In the case of an asynchronous counter, all the FLIP-FLOPs are not clocked simultaneously, whereas in a synchronous counter all the FLIP-FLOPs are clocked simultaneously. The ring- and the twisted-ring counters are examples of synchronous counters.

Consider the count sequence shown in Table 8.4. The number of states in this sequence is 8 which requires 3 FLIP-FLOPs ($2^3 = 8$) and Q_2, Q_1, and Q_0 are the outputs of these FLIP-FLOPs. Assume master-slave FLIP-FLOPs.

Table 8.4 ***Counting Sequence of a 3-bit Binary Counter***

Counter state	Count		
	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The output Q_0 of the least-significant FLIP-FLOP, changes for every clock pulse. This can be achieved by using T-type FLIP-FLOP with $T_0 = 1$. The output Q_1 makes a transition (from 0 to 1 or 1 to 0) whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to the clock input of next T-type FLIP-FLOP, FF1 with $T_1 = 1$, Q_1 will change whenever Q_0 goes from 1 to 0 (falling edge of clock pulse). Similarly, Q_2 makes a transition whenever Q_1 goes from 1 to 0 and this can be achieved by connecting Q_1 to the clock input of the most-significant FLIP-FLOP, FF2 (and $T_2 = 1$). The resulting circuit is shown in Fig. 8.10. The waveforms of the outputs of the FLIP-FLOPs are shown in Fig. 8.11.

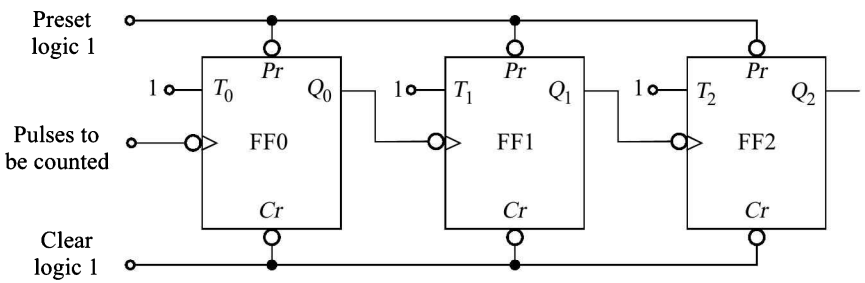


Fig. 8.10 ***A 3-bit Binary Counter***

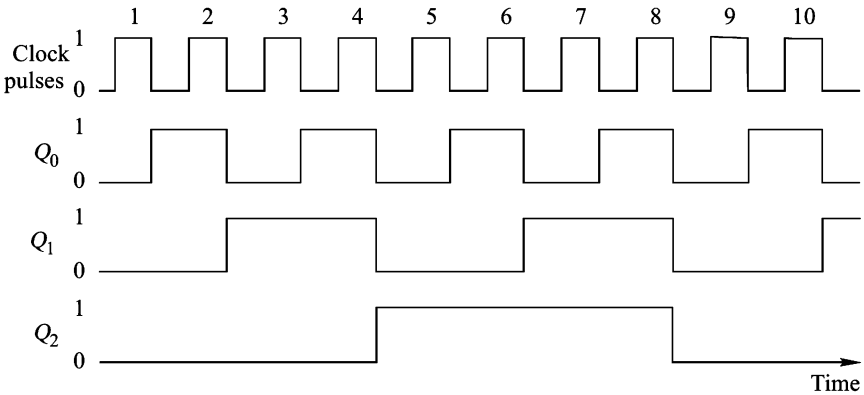


Fig. 8.11 ***Output Waveforms of Counter of Fig. 8.10***

A decoder circuit for decoding the count is shown in Fig. 8.12. In this, the output corresponding to the number counted goes low (active-low).

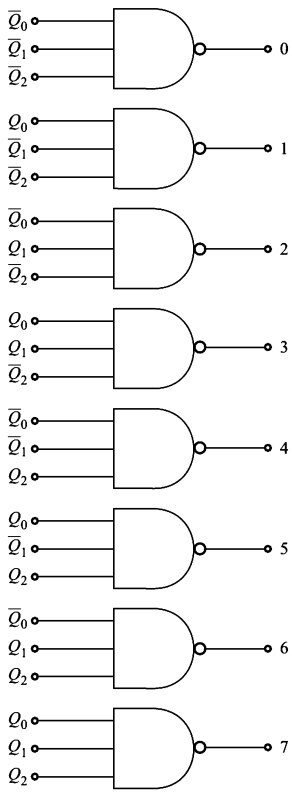


Fig. 8.12 ***A Decoder Circuit for a 3-bit Binary Counter***

At the decoder outputs, false pulses of a short duration, known as *spikes*, occur as counter FLIP-FLOPs change state. This is because of the propagation delay of the FLIP-FLOPs due to which either all the FLIP-FLOPs do not change state at exactly the same time or only one FLIP-FLOP changes state for any clock pulse.

The problems of spikes at the decoder outputs is eliminated by using a *strobe* pulse input. With this, the decoding will occur only when all the FLIP-FLOPs have come to steady state.

The frequency, f, of clock pulses for reliable operation of the counter is given by

$$\frac{1}{f} \geq N \cdot (t_d) + T_s \tag{8.4}$$

where N = number of FLIP-FLOPs
t_d = propagation delay of one FLIP-FLOP
T_s = strobe pulse width.

Example 8.2

In a 4-stage ripple counter, the propagation delay of a FLIP-FLOP is 50 ns. If the pulse width of the strobe is 30 ns, find the maximum frequency at which the counter operates reliably.

Solution

The maximum frequency is

$$f_{\max} = \frac{10^3}{4 \times 50 + 30} \text{ MHz}$$
$$= 4.35 \text{ MHz}$$

8.4.1 UP/DOWN Counters

The counter of Fig. 8.10 counts in the UP direction, i.e. the decimal equivalent of the counter output increases with successive clock pulses. It is also possible to make a counter in which the decimal equivalent of the counter output decreases with the application of successive clock pulses, i.e. the counting proceeds in the DOWN direction (Prob. 8.5). The former is referred to as an *UP counter* and the latter as a *DOWN counter*.

An UP/DOWN counter can also be designed which can count in any direction depending upon the direction control input (Prob. 8.6).

8.4.2 Modulus of the Counter

The counter discussed above is referred to as a *ripple counter*, since the pulses applied ripple from stage to stage. It is a modulo n counter where $n = 2^N$.

If it is desired to have a modulo m counter, the number of FLIP-FLOPs required is determined using Eq. (8.5) as the minimum value of N which satisfies the equation

$$m \leq 2^N \tag{8.5}$$

For example, the value of N is 4 for any value of m from 9 to 16. If $m = 16$, then the circuit can be designed as discussed above but if it is less than 16, say 10, then out of 16 states only 10 states are used and the remaining six states are unused. The counter is required to be reset (i.e. the normal counting is to be terminated) at the

end of the tenth clock pulse. This can be achieved by generating a logic 0 signal immediately after the tenth pulse and applying it to the clear input of all the FLIP-FLOPs.

For a modulo 10 counter (also referred to as a *decade counter*) the circuit for resetting the counter after the tenth pulse is shown in Fig. 8.13.

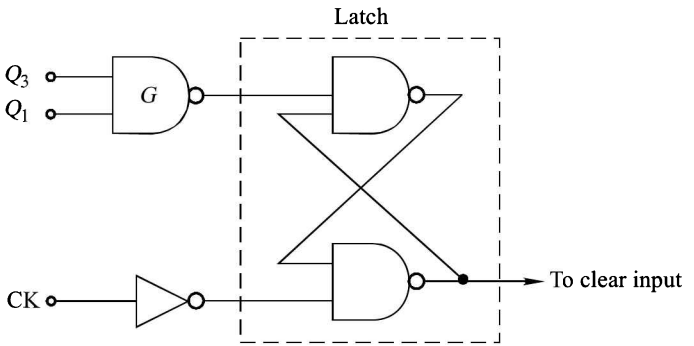


Fig. 8.13 ***A Circuit to be Used to Obtain a Decade Counter from Modulo-16 Counter***

At the end of the tenth pulse, $Q_3 = Q_1 = 1$, therefore, the output of the NAND gate G will be 0, making the output of the latch 0. This will reset the counter. The latch is used in order to keep the clear line at 0 till both the FLIP-FLOPs are cleared.

A counter in which the starting state is not 0 can be designed by making use of the preset inputs of the FLIP-FLOPs, similar to Fig. 8.2. This is referred to as *loading* the counter asynchronously (not synchronous with the clock pulse). This is referred to as a *presettable counter.*

8.4.3 54/74 Series Asynchronous Counter ICs

The design of asynchronous counters using FLIP-FLOPs has been discussed above. Some asynchronous counters are available in MSI and are given in Table 8.5 along with some of their features. Depending upon these features, these ICs are divided into three groups A, B, and C. The group to which a particular IC belongs is also indicated in the table. All these ICs consist of four master-slave positive edge-triggered FLIP-FLOPs. The load, set, and reset (clear) operations are asynchronous, i.e. independent of the clock pulse.

Table 8.5 ***Available Asynchronous Counter* ICs *in* TTL *and* CMOS *Families***

IC No.	Description	Features	Group
7490, 74290	BCD counter	Set, reset	A
7492	Divide-by-12 counter	Reset	B
7493, 74293	4-bit binary counter	Reset	B
74176, 74196	Presettable BCD counter	Reset, load	C
74177, 74197	Presettable 4-bit binary counter	Reset, load	C
74390	Dual decade counters	Reset	B
74393	Dual 4-bit binary counters	Reset	B
74490	Dual BCD counters	Set, reset	A

Group A Asynchronous Counter ICs

Figure 8.14 shows the basic internal structure of 7490. It consists of four FLIP-FLOPs internally connected to provide a mod-2 counter and a mod-5 counter. The mod-2 and mod-5 counters can be used independently or in combination. FLIP-FLOP FFA operates as a mod-2 counter whereas the combination of FLIP-FLOPs FFB, FFC, and FFD form a mod-5 counter. There are two reset inputs R_1 and R_2 both of which are to be connected to logic 1 level for clearing all the FLIP-FLOPs. The two set inputs S_1 and S_2 when connected to logic 1 level, are used for setting the counter to 1001.

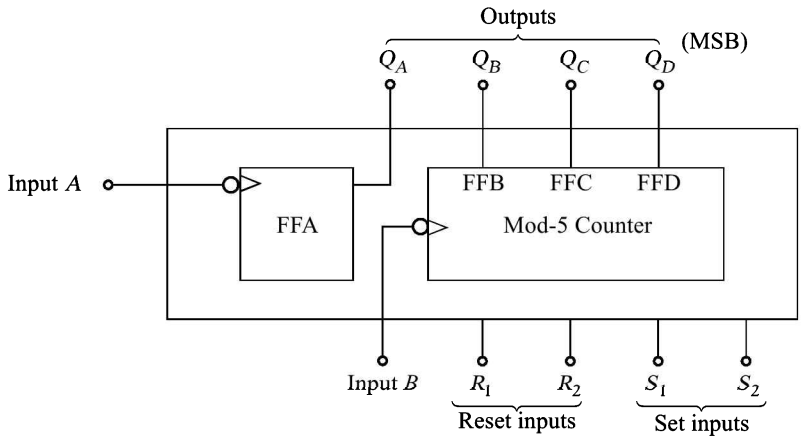


Fig. 8.14 ***Basic Internal Structure of 7490 Ripple Counter IC***

IC 74490 is a dual BCD counter consisting of two independent BCD counters. Each section consists of four FLIP-FLOPs, all connected internally to form a decade counter. For each section there is a reset (R) and a set (S) input which are active-high.

Example 8.3

In a 7490 if Q_A output is connected to B input and the pulses are applied at A input, find the count sequence and the waveforms at Q outputs.

Solution

When Q_A output is connected to B input, we have the mod-2 counter followed by the mod-5 counter. The count sequence obtained is given in Table 8.6.

Table 8.6

Counter state	FLIP-FLOP outputs			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1

(*Continued*)

Table 8.6 (*Continued*)

Counter state	FLIP-FLOP outputs			
	Q_D	Q_C	Q_B	Q_A
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

It may be noted that when Q_A changes from 0 to 1, the state of the mod-5 counter does not change, whereas when Q_A changes from 1 to 0, the mod-5 counter goes to the next state.

The waveforms at Q outputs are illustrated in Fig. 8.15.

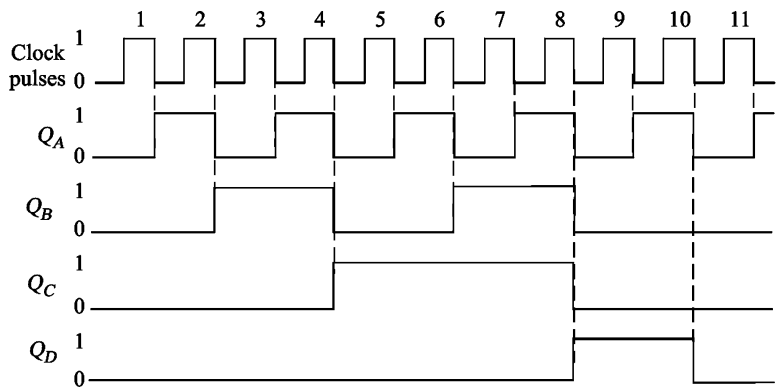


Fig. 8.15 ***Waveforms at Q Outputs for Ex. 8.3***

Example 8.4

In a 7490, if Q_D output is connected to A input and the pulses are applied at B input, find the count sequence and the waveform of output Q_A. Compare its count sequence with that of Table 8.6.

Solution

When Q_D output is connected to A input and the pulses are applied at B input, we have the mod-5 counter followed by the mod-2 counter. The count sequence obtained is given in Table 8.7. Here the states of the mod-5 counter change in a normal binary sequence and Q_A changes whenever Q_D goes from 1 to 0.

Table 8.7

Counter State	FLIP-FLOP outputs			
	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	0	0	0	1
6	0	0	1	1
7	0	1	0	1
8	0	1	1	1
9	1	0	0	1
10	0	0	0	0

The waveforms of Q_A output is illustrated in Fig. 8.16 which is a square wave.

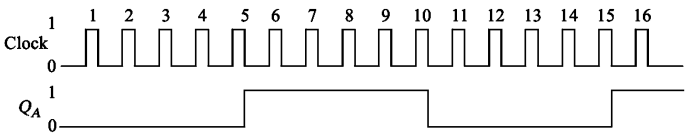


Fig. 8.16 ***Waveform of Output Q_A***

The count sequence of this counter is different from that of Table 8.6 although both are divide-by-10 counters. The waveform of Q_A output is square wave. Hence this circuit can be used for the generation of square waves.

Example 8.5

Design a divide-by-6 counter using 7490.

Solution

Connect the counter as divide-by-10 for normal binary sequence (Ex. 8.3). Outputs Q_B and Q_C are connected to the reset inputs. Therefore as soon as Q_B and Q_C both become 1, the counter is reset to 0000. Figure 8.17 shows the divide-by-6 ripple counter.

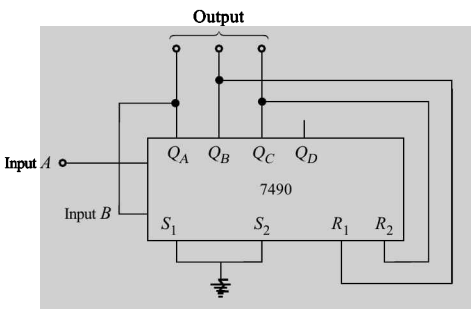


Fig. 8.17 ***A Divide-by-6 Ripple Counter Using 7490***

Group B Asynchronous Counter ICs

The basic internal structure of 7492, 7493, and 74293 asynchronous counter ICs is shown in Fig. 8.18. The operation of these ICs is identical to the operation of IC 7490 except that the set inputs are not present and mod-6 counter does not count in straight binary sequence. The sequence of mod-6 counter is given in Table 8.8. These ICs are not used as counters but are used for frequency division.

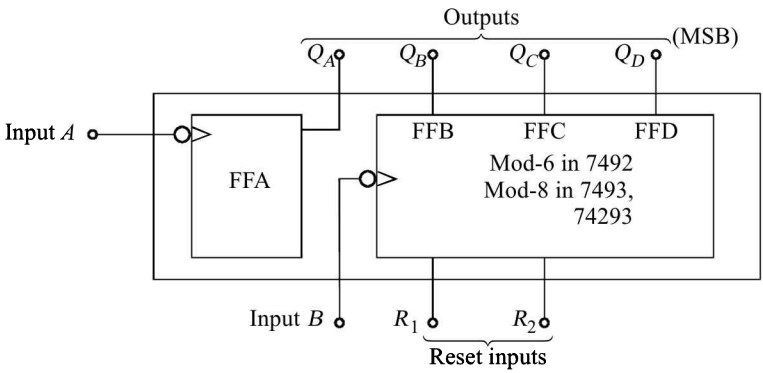


Fig. 8.18 ***Basic Internal Structure of 7492, 7493 and 74293 Counter ICs***

Table 8.8 ***Count Sequence of* Mod-6 *Counter***

Q_D	Q_C	Q_B
0	0	0
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0

74390 IC is a dual BCD counter consisting of two independent BCD counters similar to 7490. There is one reset (R) input for each section. 74393 is a dual 4-bit binary counter with one reset (R) input for each section which is active-high.

Example 8.6

If output Q_A of a divide-by-12 ripple counter 7492 is connected to the B input and the pulses are applied at the A input, find the count sequence.

Solution

The count sequence is given in Table 8.9. It may be noted from it that simultaneous divisions of 2, 6, and 12 are performed at the Q_A, Q_C, and Q_D outputs, respectively.

Table 8.9

Q_D	Q_C	Q_B	Q_A
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1

Group C Asynchronous Counter ICs

The basic internal structure of group C counter ICs is shown in Fig. 8.19. 74176 and 74196 are both BCD counters with a difference in only maximum clock frequency specification. Similarly, 74177 and 74197 are both 4-bit binary counters with the same difference.

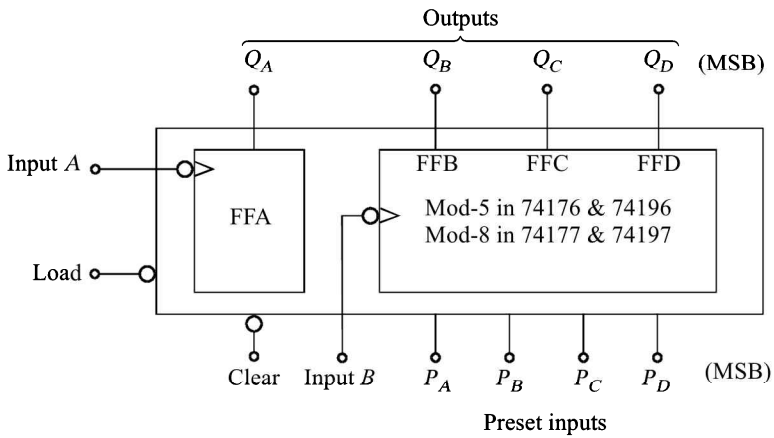


Fig. 8.19 ***Basic Internal Structure of Group C Asynchronous Counter ICs***

These counters are presettable versions of 7490 and 7493 counters respectively. The counter is cleared by connecting logic 0 to clear input (active-low). Setting load input to logic 0 (while clear is at logic 1) stops the count and loads any binary number present at the preset inputs into the counter. For normal UP counting operation, both the load and clear inputs should be connected to logic 1.

The presettable 4-bit binary counters can be used as variable mod-n counter in which the counter modulus is equal to $15 - P$, where P is the binary number connected at the preset inputs. In other words, for designing

a mod-n counter, the value of P is $15 - n$. When the counter output reaches the count 1111, the counter must be loaded again with P. This is made possible by using a 4-input NAND gate between the Q outputs of the counter and the load input.

Example 8.7

Design a divide-by-12 counter using 74177.

Solution

The circuit of a divide-by-12 counter is shown in Fig. 8.20. The counter is loaded with $P = 1111 - 1100 = 0011$ as soon as the output becomes 1111.

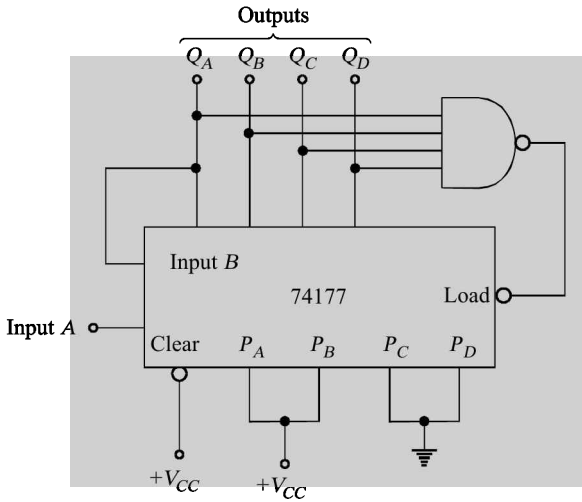


Fig. 8.20 ***A Divide-by-12 Counter Using 74177***

Cascading of Ripple Counter ICs

Ripple counters of any cycle length can be obtained by cascading the ICs discussed above. The desired cycle length is decoded and used to reset all the counters to 0. The strobe should be used to eliminate false data.

The cascading arrangement for all the asynchronous counter ICs is same where Q_D of preceding stage goes to the clock input terminal of the succeeding stage. The load and clear inputs of all ICs are to be connected together.

Example 8.8

Design a 2-decade BCD counter using the IC 74390.

Solution

The 74390 IC is a dual BCD counter, therefore only one IC is required to design a 2-decade BCD counter. The 2-decade counter is shown in Fig. 8.21.

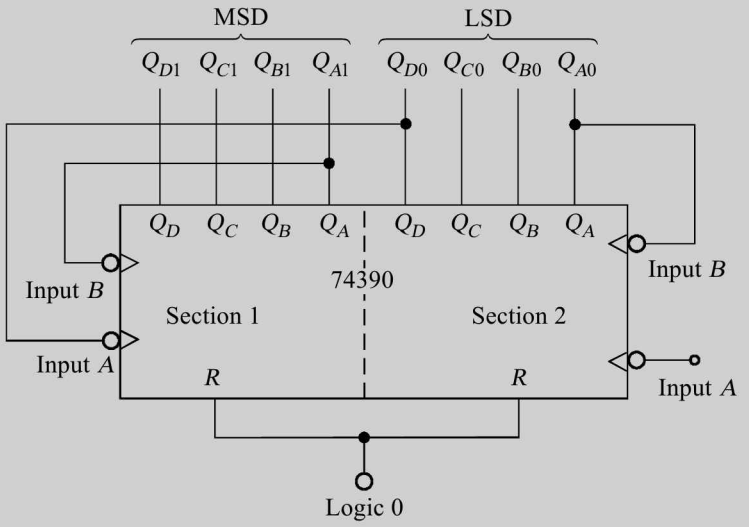


Fig. 8.21 *A 2-Decade BCD Counter Using 74390 IC*

8.5 SYNCHRONOUS COUNTERS

The ripple counters have the advantage of simplicity (only FLIP-FLOPs are required) but their speed is low because of ripple action. The maximum time is required when the output changes from 111 ... 1 to 000 ... 0 and this limits the frequency of operation of ripple counters.

The speed of operation improves significantly if all the FLIP-FLOPs are clocked simultaneously. The resulting circuit is known as a *synchronous counter*. Synchronous counters can be designed for any count sequence (need not be straight binary). These counters can be designed following a systematic approach. Before we discuss the formal method of design for such counters, we shall consider an intuitive method.

Consider the count sequence of Table 8.4. The output Q_0 of the least-significant FLIP-FLOP changes for every clock pulse. This can be achieved by using a *T*-type FLIP-FLOP with $T_0 = 1$. The output Q_1 changes whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to *T* input (T_1) of the next FLIP-FLOP, Q_1 will change from 1 to 0 (or 0 to 1) when $Q_0 = 1$ ($T_1 = 1$) and will remain unaffected when $Q_0 = T_1 = 0$. Similarly, we observe from Table 8.4 that Q_2 changes whenever Q_1 and Q_0 are both 1. This can be achieved by making the *T*-input (T_2) of the most-significant FLIP-FLOP equal to $Q_1 \cdot Q_0$. The circuit thus obtained is shown in Fig. 8.22.

In addition to FFs, synchronous counters require some gates also. *J-K* FLIP-FLOPs are the most commonly used FLIP-FLOPs for the design of synchronous counters. In this, each FLIP-FLOP has two control inputs (*J* and *K*) and circuit is required to be designed for each control input. Many programmable logic devices (PLDs) used for the design of digital systems utilise *D* FLIP-FLOPs for their memory elements, therefore, counter design using *D* FLIP-FLOPs will be useful for programming inside a PLD. It has only one control input which makes its design simpler than the design using *J-K* FLIP-FLOPs.

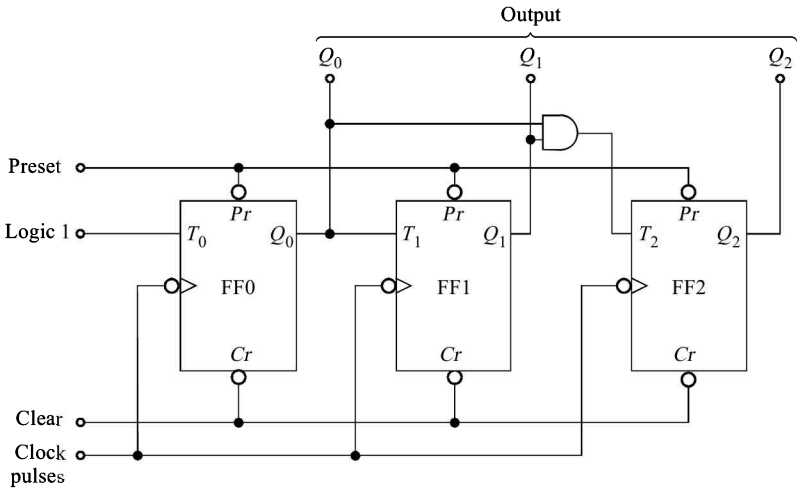


Fig. 8.22 ***A 3-bit Synchronous Counter***

8.5.1 Synchronous Counter Design

Synchronous counters for any given count sequence and modulus can be designed in the following way:

1. Find the number of FLIP-FLOPs required using Eq. (8.5).
2. Write the count sequence in the tabular form similar to Table 8.4.
3. Determine the FLIP-FLOP inputs which must be present for the desired next state from the present state using the excitation table of the FLIP-FLOPs (Table 7.6).
4. Prepare *K*-map for each FLIP-FLOP input in terms of FLIP-FLOP outputs as the input variables. Simplify the *K*-maps and obtain the minimized expressions.
5. Connect the circuit using FLIP-FLOPs and other gates corresponding to the minimized expressions.

The above design steps can be clearly understood from the following examples.

Example 8.9

Design a 3-bit synchronous counter using *J-K* FLIP-FLOPs.

Solution

The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1, and FF2 and their inputs and outputs are given below:

FLIP-FLOP	Inputs	Output
FF0	J_0, K_0	Q_0
FF1	J_1, K_1	Q_1
FF2	J_2, K_2	Q_2

Table 8.10

Counter state			FLIP-FLOP inputs					
			FF0		FF1		FF2	
Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	1	×	0	×	0	×
0	0	1	×	1	1	×	0	×
0	1	0	1	×	×	0	0	×
0	1	1	×	1	×	1	1	×
1	0	0	1	×	0	×	×	0
1	0	1	×	1	1	×	×	0
1	1	0	1	×	×	0	×	0
1	1	1	×	1	×	1	×	1
0	0	0						

The count sequence and the required inputs of FLIP-FLOPs are given in Table 8.10. The inputs to the FLIP-FLOPs are determined in the following manner:

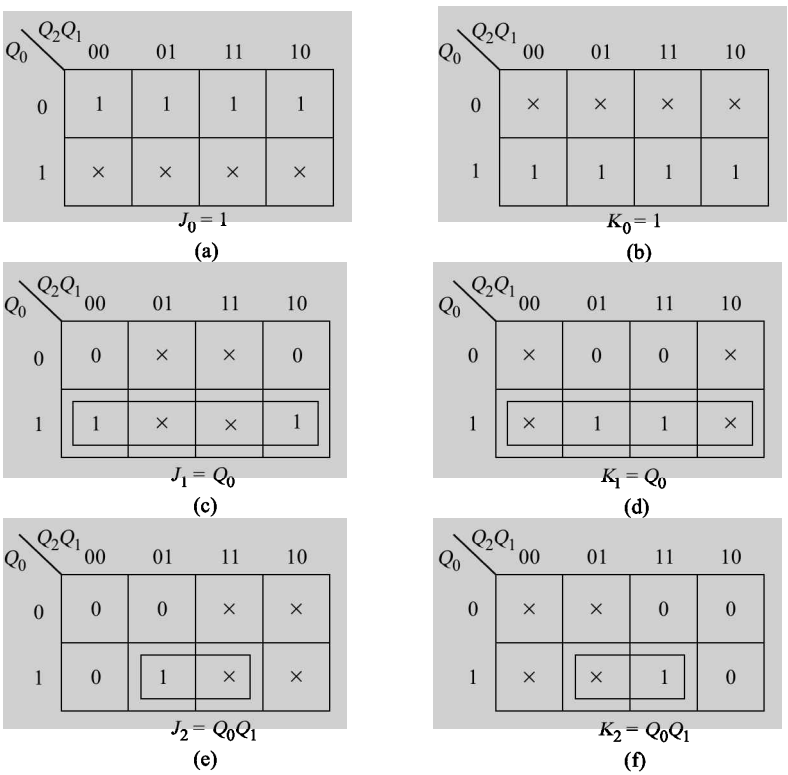


Fig. 8.23 ***K-Maps of Ex. 8.9***

Consider one column of the counter state at a time and start from the first row, for example, consider Q_0. Before the first pulse is applied, $Q_0 = 0$ and it is required to be 1 at the end of the first clock pulse. Therefore, to achieve this condition, the values of J_0 and K_0 are 1 and × respectively (from the excitation Table 7.6). These are entered in the table in the row corresponding to 0 pulse. When the second clock pulse is applied Q_0 is to change from 1 to 0, therefore, the required inputs are

$$J_0 = \times, K_0 = 1$$

In a similar manner inputs of each FLIP-FLOP are determined.

Now, we prepare the *K*-maps (Fig. 8.23) with Q_2, Q_1, and Q_0 as input variables and FLIP-FLOP inputs as output variables. We then minimize the *K*-maps and the resulting minimized expressions are:

$$J_0 = 1, \quad K_0 = 1$$
$$J_1 = Q_0, \quad K_1 = Q_0$$
$$J_2 = Q_0 Q_1, \quad K_2 = Q_0 Q_1$$

The resulting counter circuit is same as the circuit of Fig. 8.22.

Example 8.10

Design a 3-bit binary UP/DOWN counter with a direction control *M*. Use *J-K* FLIP-FLOPs.

Solution

The count sequence is given in Table 8.11. For $M = 0$, it acts as an UP counter and for $M = 1$ as a DOWN counter. The number of FLIP-FLOPs required is 3. The inputs of the FLIP-FLOPs are determined in a manner similar to the one employed in Ex. 8.9.

Table 8.11

Direction control	Counter state			FLIP-FLOP inputs					
M	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	×	0	×	0	×
0	0	0	1	×	1	1	×	0	×
0	0	1	0	1	×	×	0	0	×
0	0	1	1	×	1	×	1	1	×
0	1	0	0	1	×	0	×	×	0
0	1	0	1	×	1	1	×	×	0
0	1	1	0	1	×	×	0	×	0
0	1	1	1	×	1	×	1	×	1
1	0	0	0	1	×	1	×	1	×
1	1	1	1	×	1	×	0	×	0
1	1	1	0	1	×	×	1	×	0
1	1	0	1	×	1	0	×	×	0
1	1	0	0	1	×	1	×	×	1
1	0	1	1	×	1	×	0	0	×
1	0	1	0	1	×	×	1	0	×
1	0	0	1	×	1	0	×	0	×
	0	0	0						

From Table 8.11, we obtain

$$J_0 = K_0 = 1$$

The K-maps for J_1, K_1, J_2, and K_2 are shown in Fig. 8.24. From the K-maps, the minimized expressions are obtained as

$$J_1 = K_1 = Q_0\overline{M} + \overline{Q}_0 M$$
$$J_2 = K_2 = \overline{M}\,Q_1 Q_0 + M\,\overline{Q}_1\,\overline{Q}_0$$

The counter circuit can be drawn using the above expressions

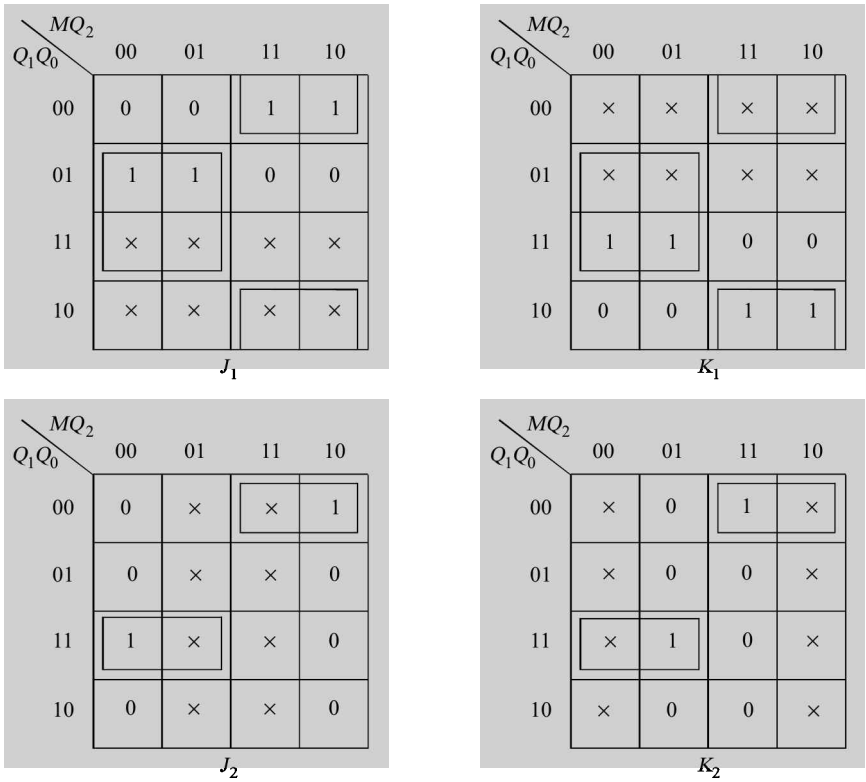


Fig. 8.24 ***K-Maps for Ex. 8.10***

Example 8.11

Design a decade UP counter. Use *J-K* FLIP-FLOPs.

Solution

There are ten states in a decade counter, which requires four FLIP-FLOPs. The remaining six states are unused states. The count sequence and the FLIP-FLOP inputs are given in Table 8.12.

Table 8.12

Counter state				FLIP-FLOP inputs							
Q_3	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2	J_3	K_3
0	0	0	0	1	×	0	×	0	×	0	×
0	0	0	1	×	1	1	×	0	×	0	×
0	0	1	0	1	×	×	0	0	×	0	×
0	0	1	1	×	1	×	1	1	×	0	×
0	1	0	0	1	×	0	×	×	0	0	×
0	1	0	1	×	1	1	×	×	0	0	×
0	1	1	0	1	×	×	0	×	0	0	×
0	1	1	1	×	1	×	1	×	1	1	×
1	0	0	0	1	×	0	×	0	×	×	0
1	0	0	1	×	1	0	×	0	×	×	1
0	0	0	0								

The *K*-maps are shown in Fig. 8.25 from which the minimized expressions are obtained as

$$J_0 = 1, \qquad K_0 = 1$$
$$J_1 = Q_0\overline{Q}_3, \qquad K_1 = Q_0$$
$$J_2 = Q_0Q_1, \qquad K_2 = Q_0Q_1$$
$$J_3 = Q_0Q_1Q_2, \qquad K_3 = Q_0$$

The counter circuit can be drawn using the above expressions.

Example 8.12

Design a natural binary sequence mod-8 synchronous counter using *D* FLIP-FLOPs.

Solution

The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1 and FF2 with inputs D_0, D_1 and D_2 respectively. Their outputs are Q_0, Q_1, and Q_2 respective. The count sequence and the corresponding FLIP-FLOPs input required are given in Table 8.13. Using the excitation Table 7.6, the FLIP-FLOPs inputs are determined in the same way as determined for the *J-K* FLIP-FLOPs.

Table 8.13 ***Counter States and D* FLIP-FLOPs *Input***

Counter state			FLIP-FLOP inputs		
Q_2	Q_1	Q_0	D_0	D_1	D_2
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
0	0	0			

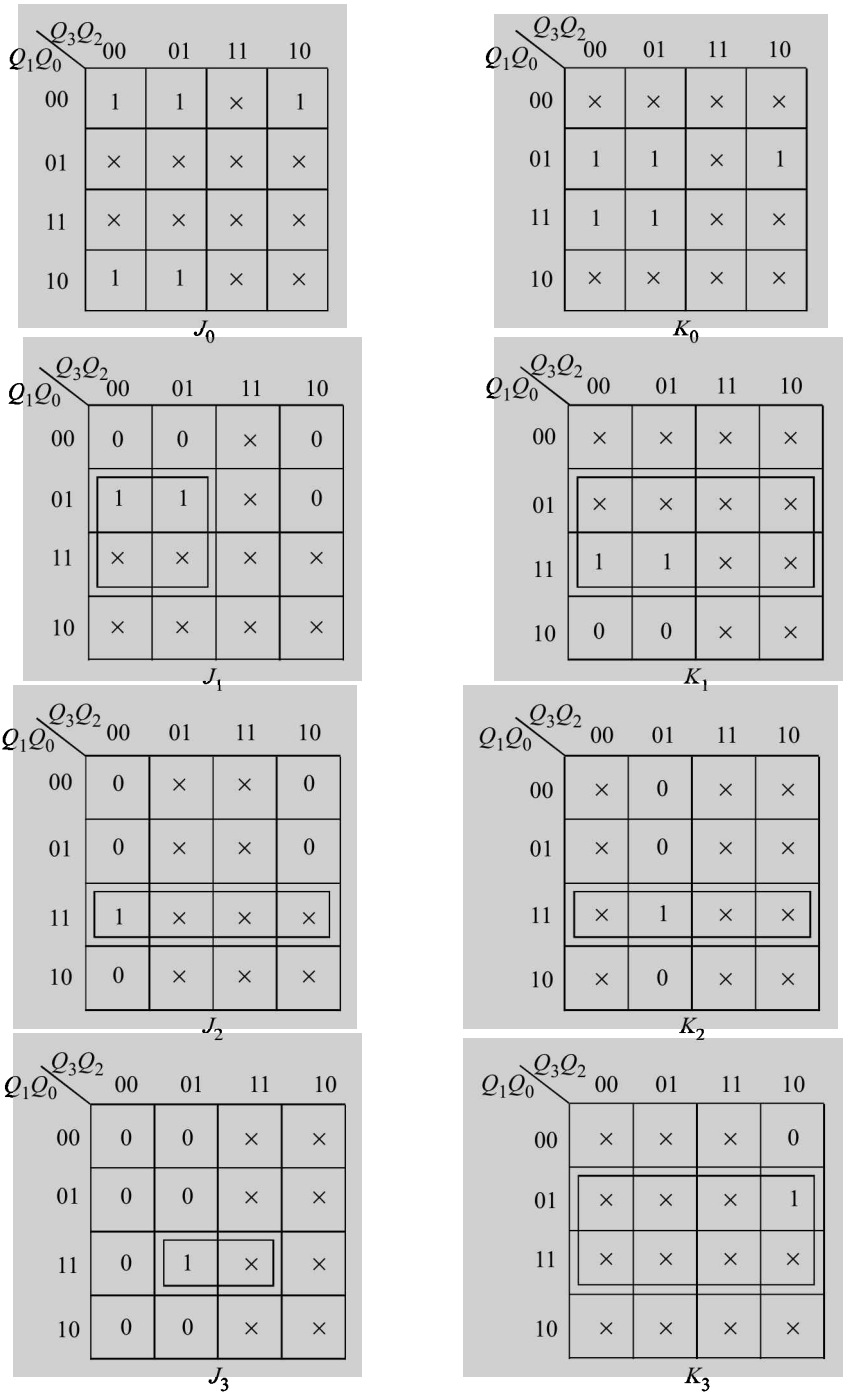


Fig. 8.25 ***K-Maps for Ex. 8.11***

The *K*-maps for D_0, D_1, and D_2 are given in Fig. 8.26.

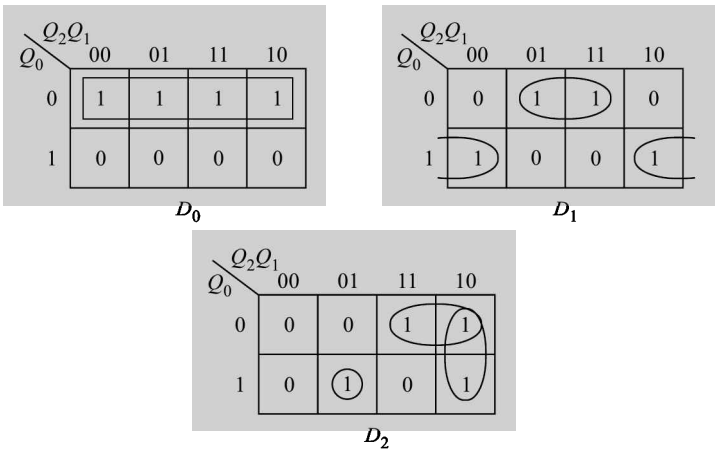


Fig. 8.26 ***K-Maps of Ex. 8.12***

The minimised expressions for D_0, D_1, and D_2 are:

$$D_0 = \overline{Q}_0 \qquad D_1 = Q_1\overline{Q}_0 + \overline{Q}_1 Q_0$$

$$\begin{aligned} D_2 &= Q_2\overline{Q}_0 + Q_2\overline{Q}_1 + \overline{Q}_2 Q_1 Q_0 \\ &= Q_2(\overline{Q}_0 + \overline{Q}_1) + \overline{Q}_2 Q_1 Q_0 \\ &= Q_2(\overline{Q_0 \cdot Q_1}) + \overline{Q}_2(Q_1 Q_0) \\ &= Q_2 \oplus Q_1 \cdot Q_0 \end{aligned}$$

The complete circuit of the synchronous counter using positive edge triggered *D* FLIP-FLOPs is shown in Fig. 8.27.

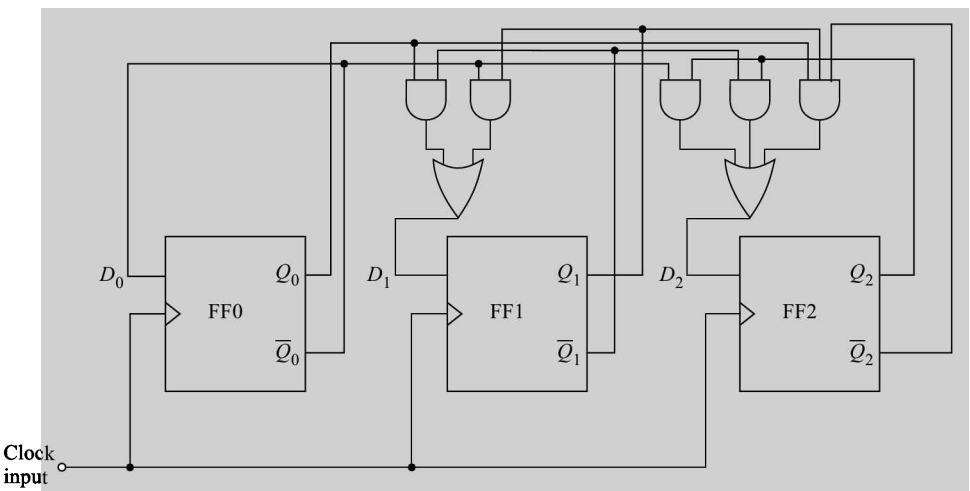


Fig. 8.27 ***Synchronous Counter Circuit of Ex. 8.12***

8.5.2 Lock Out

In the counter specified by Table 8.12, logic states, $Q_3Q_2Q_1Q_0$ = 1010, 1011, 1100, 1101, 1110, and 1111 are not used. If, by chance, the counter happens to find itself in any one of the unused states, its next state would not be known. It may just be possible that the counter might go from one unused state to another and never arrive at a used state. Of course, such a situation makes the counter useless for its intended purpose. A counter whose unused states have this feature is said to suffer from *lock out*. To make sure that at the starting point the counter is in its initial state or it comes to its initial state within a few clock cycles (count error due to noise), external logic circuitry is to be provided.

To ensure that lock out does not occur, we design the counter assuming the next state to be the initial state, from each of the unused states. Beyond this, the design procedure is the same as discussed earlier.

8.5.3 54/74 Series Synchronous Counter ICs

The design of synchronous counters using FLIP-FLOPs has been discussed above. Counters for any count sequence and modulus can be designed using these methods. Some synchronous counters are available in MSI and are given in Table 8.14 along with some of their features. All theses ICs are positive-edge-triggered, i.e. the change of state, synchronous loading, and clearing take place on the positive going edge of the input clock pulse. Basically these ICs can be divided into four groups—A, B, C, and D. A brief description of each group is given below:

Table 8.14 ***Available Synchronous Counter* ICs *in* TTL *and* CMOS *Families***

IC No.	Description	Features	Group
74160	Decade UP counter	Synchronous preset and asynchronous clear	A
74161	4-bit binary UP counter	—do—	A
74162	Decade UP counter	Synchronous preset and clear	A
74163	4-bit binary UP counter	—do—	A
74168	Decade UP/DOWN counter	Synchronous preset and no clear	B
74169	4-bit binary UP/DOWN counter	—do—	B
74190	Decade UP/DOWN counter	Asynchronous preset and no clear	C
74191	4-bit binary UP/DOWN counter	—do—	C
74192	Decade UP/DOWN counter	Asynchronous preset and clear	D
74193	4-bit binary UP/DOWN counter	—do—	D

Group A Synchronous Counter ICs

The block diagram and the function table of these ICs are given in Fig. 8.28. In these ICs there are two separate enable inputs, *ENT* and *ENP*. Setting either of these inputs to logic 0 stops counting asynchronously. Ripple carry (RC) output is normally at logic 0 and goes to logic 1 whenever the counter reaches its highest count (binary 9 for BCD counters and binary 15 for 4-bit binary counters). Setting *ENT* to logic 0 also inhibits *RC* changing from logic 0 to logic 1.

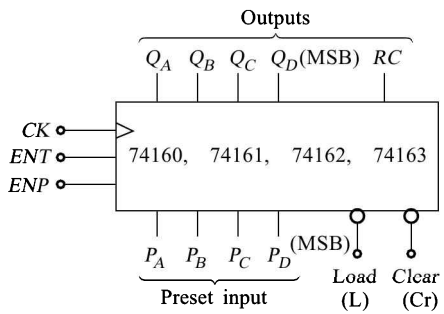


Fig. 8.28 ***Group A Synchronous Counter ICs (a) Block Diagram***

Load L	*ENP*	*ENT*	*Cr*	*CK*	*Mode*
0	×	×	1	↑	Preset
1	0	1	1	×	Stop count
1	×	0	1	×	Stop count, disable RC
×	×	×	0	*	Reset to zero
1	1	1	1	↑	UP count

* × for 74160 and 74161
↑ for 74162 and 74163

Fig. 8.28 ***(b) Function Table***

Example 8.13

Design a normal mod-12 counter using 74161.

Solution

The circuit is designed for the normal UP counting (last row of Fig. 8.28*b*). The Q_D and Q_C outputs through a NAND gate are connected to the *Cr* terminal which clears the counter as soon as the output is 1100. The states of the counter are from 0000 through 1011. The mod-12 counter is shown in Fig. 8.29.

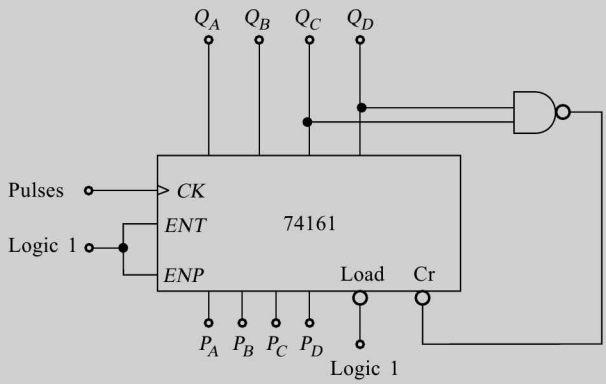


Fig. 8.29 ***Figure for Ex. 8.13***

Using the approach followed in Ex. 8.13, the count can be terminated at any desired value and a counter with any modulus (less than 16 for binary and less than 10 for decade counter) can be obtained.

Example 8.14

Design a divide-by-11 counter using 74163. Make use of the *RC* output and preset inputs.

Solution

For obtaining a divide-by-11 counter, the counter is preset at binary 0101 (decimal 5). When the count reaches 1111, *RC* output goes to 1, which is used to load the data present at the preset inputs into the counter. The circuit of the counter is shown in Fig. 8.30.

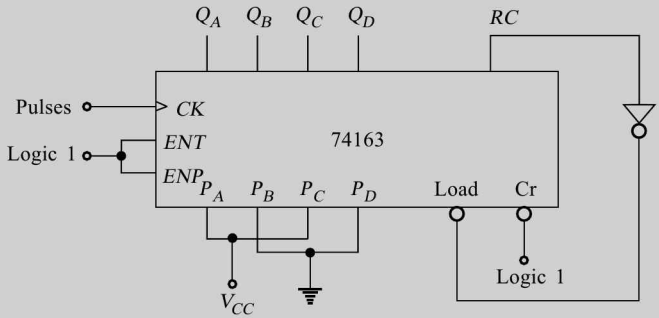


Fig. 8.30 ***Counter for Ex. 8.14***

In general, for obtaining a divide-by-*m* counter, the preset input, *P*, is given by

$P = 16 - m$ for 4-bit binary counter
$= 10 - m$ for decade counter

Cascading of group A counters in fully synchronous mode is shown in Fig. 8.31.

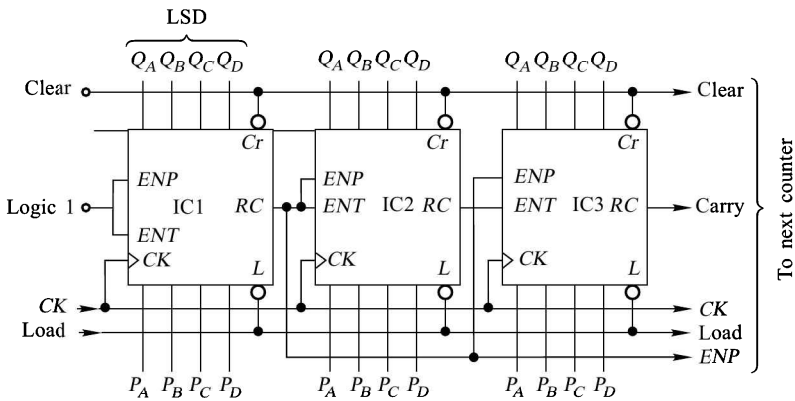


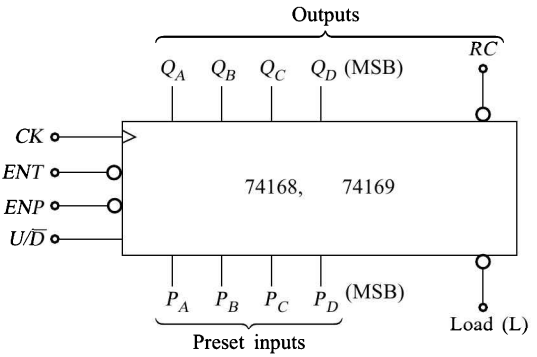
Fig. 8.31 ***Cascading Arrangement of Group A Synchronous Counter ICs***

Group B Synchronous Counter ICs

The block diagram and the function table of these ICs are given in Fig. 8.32.

The functions of *ENT* and *ENP* are same as in group A ICs except that these are active-low. Ripple carry (*RC*) output is normally held at logic 1 and goes to logic 0 (i) when the count reaches maximum during UP counting, and (ii) when the count reaches minimum during DOWN counting. Signal at $U/\overline{D}$ terminal decides the direction of counting, $U/\overline{D} = 1$ for UP counting and $U/\overline{D} = 0$ for DOWN counting.

In this group of ICs, the clear terminal is not available. Therefore, if it is desired to terminate the count before it reaches the maximum value, a NAND gate is used to detect the count corresponding to the required number and its output is connected to the load input terminal. The preset inputs can be given corresponding to the required starting state of the counter.



(a)

Load L	*ENP*	*ENT*	$U/\overline{D}$	*CK*	Mode
0	×	×	×	↑	Preset
1	1	0	×	×	Stop count
1	×	1	×	×	Stop count, disable RC
1	0	0	1	↑	UP count
1	0	0	0	↑	DOWN count

(b)

Fig. 8.32 ***Group B Synchronous Counter ICs (a) Block Diagram (b) Function Table***

Example 8.15

Design a counter with states 0011 through 1100 using 74169 counter.

Solution

The preset input is 0011 and as soon as the output reaches 1100, on the next pulse it should come back to its original state. Therefore, the number corresponding to the highest required state is to be detected for loading the counter. The counter is shown in Fig. 8.33.

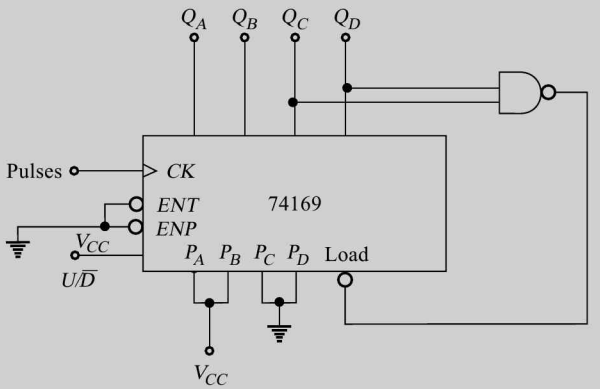


Fig. 8.33 ***Counter for Ex. 8.15***

If the counter is required to count up to the maximum/minimum value then the *RC* output is to be connected to the load input, for loading the initial count at the next pulse after maximum/minimum count has been reached. The frequency of the output waveform at *RC* (f_{out}) is related to the input clock frequency (f_{in}) as follows:

Binary Counter 74169

$$f_{out} = \frac{f_{in}}{N+1}, \quad 1 \le N \le 15 \quad \text{(for DOWN counting)}$$

$$= \frac{f_{in}}{16-N}, \quad 0 \le N \le 14 \quad \text{(for UP counting)}$$

Decade Counter 74168

$$f_{out} = \frac{f_{in}}{N+1}, \quad 1 \le N \le 9 \quad \text{(for DOWN counting)}$$

$$= \frac{f_{in}}{10-N}, \quad 0 \le N \le 8 \quad \text{(for UP counting)}$$

where N is the decimal equivalent of the preset input.

The cascading of group B counter ICs is similar to that of group A counter ICs.

Group C Synchronous Counter ICs

These ICs have only one enable input terminal *ENAB* which is active-low. MAX/MIN output is used to detect the counter's maximum or minimum count. It is normally at logic 0 and goes to logic 1 when the count is maximum (1001 for 74190 and 1111 for 74191) for the UP counting and minimum (0000) for the DOWN counting. It serves the purpose of an overflow detector while UP counting and an underflow detector while DOWN counting. The *RC* output is normally at logic 1, and goes to logic 0 when the counter reaches a MAX/MIN point and the *CK* input is low.

The block diagram and the function table of these ICs are given in Fig. 8.34.

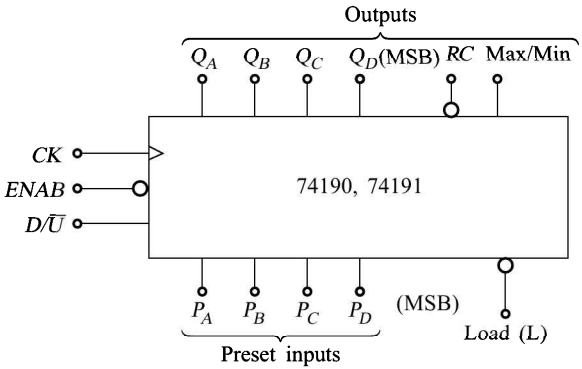
Frequency Dividers

These counters can also be used as programmable frequency dividers. By presetting any number into the counter and counting to the maximum (UP counting) or minimum (DOWN counting) count, division is achieved. The *RC* output is connected to the load input and the required output waveform is obtained at MAX/MIN output. The frequency of the output waveform f_{out} and the input clock frequency (f_{in}) are related as follows:

Binary Counter 74191

$$f_{out} = \frac{f_{in}}{N}, \quad 1 \le N \le 15 \quad \text{(for DOWN counting)}$$

$$= \frac{f_{in}}{15-N}, \quad 0 \le N \le 14 \quad \text{(for UP counting)}$$



(a)

Load L	*ENAB*	*D/$\overline{U}$*	*CK*	*Mode*
×	1	×	×	Stop count
0	0	×	×	Preset
1	0	0	↑	UP count
1	0	1	↑	DOWN count

(b)

Fig. 8.34 ***Group C Synchronous Counter ICs (a) Block Diagram (b) Function Table***

Decade Counter 74190

$$f_{out} = \frac{f_{in}}{N} \quad \text{for} \quad 1 \le N \le 9 \text{ (for DOWN counting)}$$

$$= \frac{f_{in}}{9 - N} \quad \text{for} \quad 1 \le N \le 8 \text{ (for UP counting)}$$

where N is the decimal equivalent of preset input. For example, in case of 74191, if the preset input is 1000(8), and the clock frequency is 560 Hz, the frequency of output waveform will be 80 Hz for *UP* counting and 70 Hz for DOWN counting.

Cascading of Group C Counters

These counters can be cascaded in three different ways:

1. *Synchronous counter ICs cascaded as an asynchronous counter:* The *RC* output of each stage is connected to the *CK* input of the succeeding stage and the clock pulses are applied at the *CK* input of the first stage. In this, each IC is synchronous within itself, but between stages the overall system is a ripple counter.

2. *Synchronous counter ICs cascaded with ripple carry between stages:* The *RC* output of each stage is connected to the *ENAB* input of the succeeding stage. All *CK* inputs are connected together and the clock pulses are applied at this common clock terminal.

3. *Synchronous counter ICs cascaded with parallel carry:* Figure 8.35 shows a 3-decade synchronous counter with parallel carry. The speed of operation is maximum in this type of cascading. The number of stages that can be cascaded in this manner may be restricted due to loading of MAX/MIN output by the external gating.

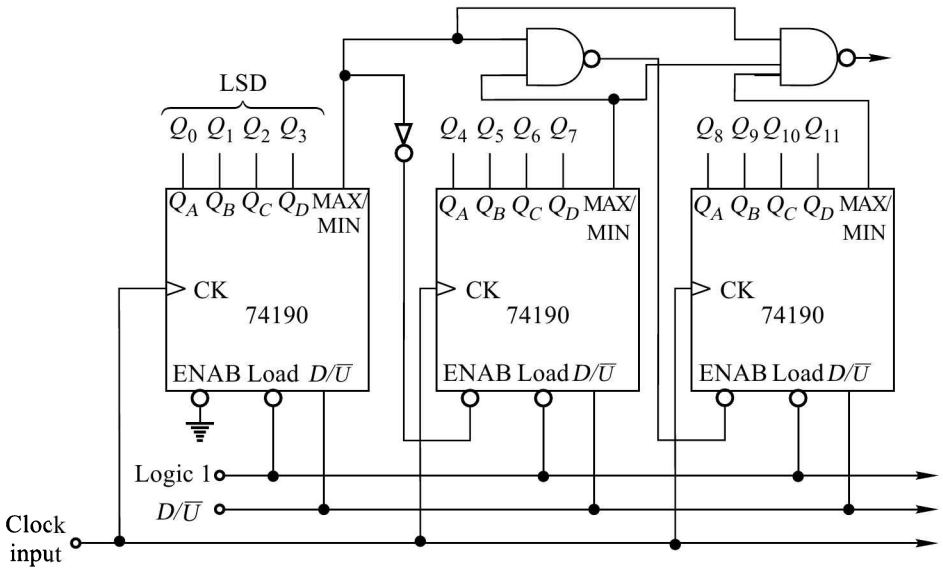


Fig. 8.35 ***A 3-Decade Synchronous Counter Using 74190 Counter ICs With Parallel Carry***

Group D Synchronous Counter ICs

In these counters, for UP counting the clock is applied at *CK*-UP terminal with *CK*-DOWN connected to logic 1 and for DOWN counting at the *CK*-DOWN terminal with *CK*-UP connected to logic 1.

The carry and borrow outputs are normally at logic 1. The carry output drops to logic 0 when the counter shows its maximum count while UP counting and the *CK*-UP input is at logic 0. The borrow output remains at logic 1 as long as the circuit is operating from the *CK*-UP input.

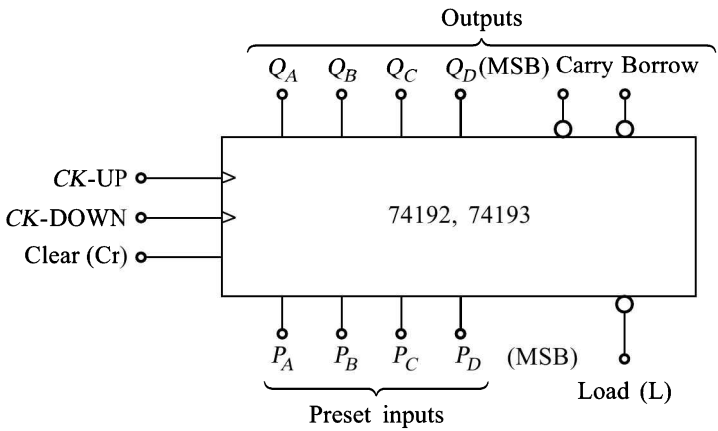
The function of borrow in DOWN counting, is the same as the function of carry in UP counting.

The block diagram and function table of these ICs are given in Fig. 8.36.

Frequency Dividers

These counters can be used as programmable frequency dividers in a manner similar to the one used for group C counters except that carry (or borrow) output is to be connected to the load input for UP (or DOWN) counting.

For example, in the DOWN count mode, if the binary equivalent of decimal number 11 is applied to the preset inputs of a 74193 4-bit binary counter, the frequency of the pulses at borrow output will be one-eleventh of the clock frequency.



(a)

Load *L*	*Clear* *Cr*	*CK-UP*	*CK-DOWN*	*Mode*
×	1	×	×	Reset to Zero
1	0	↑	1	UP count
1	0	1	↑	DOWN count
0	0	×	×	Preset
1	0	1	1	Stop count

Fig. 8.36 ***Group D Synchronous Counter ICs (a) Block Diagram (b) Function Table***

Cascading of Group D Counters

For cascading these counter ICs, the carry and borrow outputs of each stage are to be connected to the *CK*-UP and *CK*-DOWN inputs of the succeeding stage respectively. For steering the clock to *CK*-UP or *CK*-DOWN input the circuit shown in Fig. 8.37 can be used.

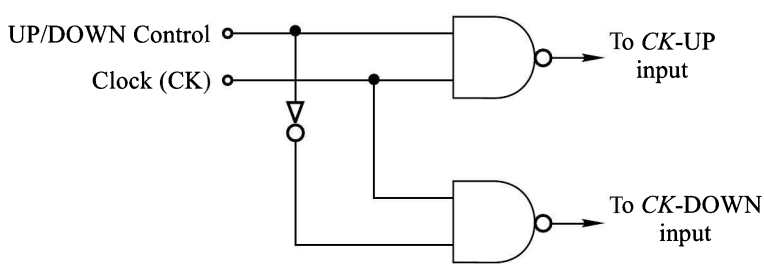


Fig. 8.37 ***Circuit for Steering Clock Pulses to CK-UP (for UP Count) or CK-DOWN (for DOWN Count)***

8.6 SYNCHRONOUS SEQUENTIAL CIRCUITS DESIGN

Sequential logic has been discussed above. The analysis and design of sequential circuits have been discussed using various types of FLIP-FLOPs. The use of clock signal for the synchronisation of the operation of one type of sequential circuits, i.e., synchronous sequential circuits require FLIP-FLOP inputs to become stable before the active edge of the clock pulse arrives. The analysis and design of clocked sequential circuits require good understanding of a logic designer regarding the operation of FLIP-FLOPs and combinational circuits.

8.6.1 Synchronous Sequential Circuits Models

A general block diagram of a clocked sequential circuit is shown in Fig. 7.1. This is also known as a finite state machine (FSM). Depending upon the way the external outputs are obtained from the circuit, there are two different models of sequential circuits. These are Mealy model and Moore model.

Mealy Model

In the case of a Mealy model, the next state is a function of the present state and the present inputs. Its output is also a function of the present state and the present inputs. Figure 7.1 represents Mealy model.

In general, the next state and the output of a Mealy model are uniquely defined by

$$\text{Next State} = F_1 \text{ (Present state, inputs)}$$
$$\text{Outputs} = F_2 \text{ (Present state, inputs)}$$

Moore Model

The block diagram of a Moore model of circuit is shown in Fig. 8.38. Similar to the Mealy model, the next state in a Moore model is also a function of the present state and the inputs but the outputs of Moore model are functions of only the present state and are independent of the inputs. Therefore, the Moore model is defined as:

$$\text{Next state} = F_1 \text{ (Present state, inputs)}$$
$$\text{Outputs} = F_2 \text{ (Present state)}$$

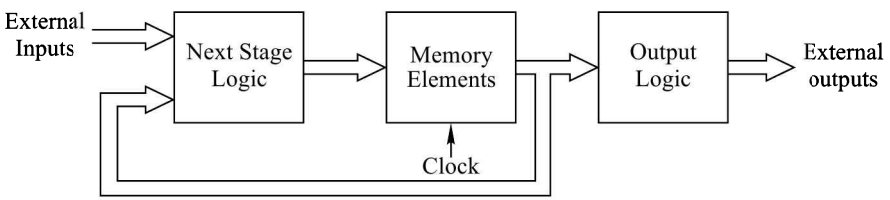


Fig. 8.38 ***Block Diagram of a Moore Model***

8.6.2 Basic Concepts

The general block diagram of a clocked sequential circuit is shown in Fig. 7.1, which also represents Mealy model. The block diagram of Moore model is shown in Fig. 8.38. Both the models use clock signal

input for the memory elements which are FLIP-FLOPs. All the FLIP-FLOPs in the circuit are clocked simultaneously. The outputs and the next state of a clocked sequential circuit depend upon the external inputs and the present state of the circuit. Systematic procedure for the design of clocked sequential circuits is based on the concept of 'state'.

For these circuits, the sequence of inputs, present and next states, and output can be represented by a *state table* or a *state diagram.*

The behaviour of a combinational circuit can be described in the form of Boolean expression(s), truth table, or *K*-map. Similarly, the behaviour of a synchronous sequential circuit can be described in a number of ways. Since the operation of a synchronous sequential circuit is always in synchronism with the clock pulses, therefore, the occurrence of a clock pulse is vital to describe its operation. Before occurrence of any given clock pulse, the following items are required to be known.

1. Present state, i.e., the output of FLIP-FLOP(s) in the circuit.
2. Signal(s) present at the external input(s), known as the input(s).

The combination of present state and external input results in a transition to the next state (when clock is applied) and an output.

Similar process is repeated, when the next clock pulse occurs, except that the present state is now what the next state was after the completion of the preceding clock pulse. This process can be represented by the sequence as shown in Fig. 8.39.

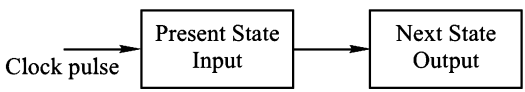


Fig. 8.39

The above operation of a clocked sequential circuit can be described in the form of a diagram, known as *state diagram*; a table, known as *state table*; or in the form of a flow chart, known as *algorithmic state machine* (*ASM*) chart.

8.6.3 State Diagram

It is a directed graph, consisting of *vertices* (or *nodes*) and directed arcs between the nodes. Every state of the circuit is represented by a node in the graph. A node is represented by a circle with the name of the state written inside the circle. The directed arcs represent the *state transitions.* With the circuit in any one state, at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, both in accordance with the requirements of the circuit. This state transition is represented by a directed line emanating from the node corresponding to the present state and terminating on the node corresponding to the next state. The labels are put on the directed arcs specifying the inputs and outputs separated by a slash (/). Consider a portion of a state diagram shown in Fig. 8.40*a.* In this, when the circuit is in state *A*, an input 1 causes the circuit to make a transition to the next state *B* and gives an output 0. *A* and *B* represent the present state and next state respectively, connected by an arc from *A* to *B* (labelled 1/0). For a circuit with single input, when the circuit is in any state, the input can be 0 or 1. For each possibility of the input, there will be a directed arc. Thus two arcs emanate from each node, one each for a 0 and for a 1 input. In general, for an *n*-input machine, 2^n arcs will emanate from each node. This is illustrated in Fig. 8.40*b.* In some sequential

circuits, the outputs are taken from the outputs of the FLIP-FLOPs directly, i.e., the output logic circuit is not there, such as FLIP-FLOPs and counters. In such cases, the directed arcs will have only inputs written adjacent to the arcs.

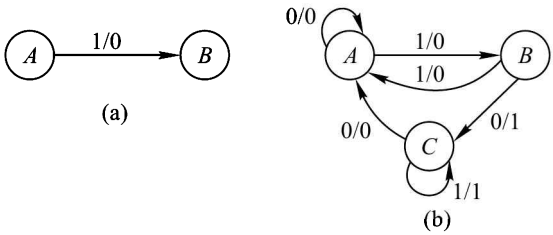


Fig. 8.40 ***Illustrations of Directed Graph***

In some cases a state may be a *terminal state*, i.e., the corresponding vertex in the state diagram may be a *sink* vertex or a *source* vertex. A vertex is a sink vertex if there are no outgoing arcs which emanate from it and terminate in other vertices. For example in Fig. 8.41*a*, on vertex *D*, whatever may be the input (0 or 1), the arc emanating from it terminates on itself and there is no arc emanating from *D* that terminates in any other vertex. Therefore, vertex *D* is a sink vertex. It means no state is accessible from a sink state. Similarly, a vertex is known as a source vertex if there are no arcs which emanate from other vertices terminating in it. For example, vertex *A* in Fig. 8.41*b*, is a source vertex.

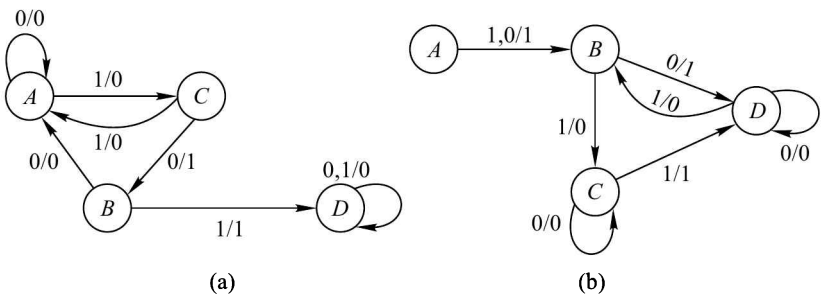


Fig. 8.41 ***Example of (a) Sink Vertex (b) Source Vertex***

Example 8.16

Draw the state diagram of a *D* type FLIP-FLOP shown in Fig. 8.42.

Solution

This circuit has one input (*D*), two states ($Q = 0$ and $Q = 1$), and a positive edge-triggered clock (*CK*) terminal.

The two states 0 and 1 are represented by two nodes as shown in Fig. 8.43. Let us assume the circuit to be in state 0 and $D = 0$, when a clock pulse occurs. At the end of the clock pulse, the circuit remains in the same state. This is indicated by a directed arc emanating from the state

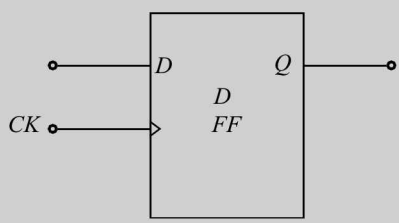


Fig. 8.42 ***D-Type*** FLIP-FLOP ***for Ex. 8.16***

0 and terminating on the same state. If $D = 1$, while the circuit's present state is 0, state transition takes place taking the circuit to another state 1 when a clock pulse occurs. Similarly, if the circuit is in state 1, for $D = 0$ a clock pulse applied will cause state to change to 0, whereas for $D = 1$ it remains in the same state. All these operations are clearly indicated in the state diagram shown in Fig. 8.43.

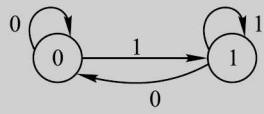


Fig. 8.43 ***State Diagram of a D*** `FLIP-FLOP`

Example 8.17

Draw the state diagram of a *J-K* `FLIP-FLOP`.

Solution

A *J-K* `FLIP-FLOP` has two inputs (J and K) and one clock input (CK). There are two states ($Q = 0$ and $Q = 1$). Its state diagram is shown in Fig. 8.44.

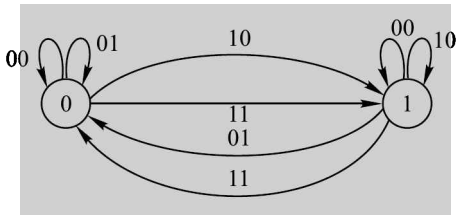


Fig. 8.44 ***State Diagram of a J-K*** `FLIP-FLOP`

Since, there are two inputs (J and K), therefore, there are $2^2 = 4$ possible input conditions and consequently, four directed arcs emanate from each state. The two inputs are indicated in JK order, i.e., the first input is represented by the first bit and the second input is represented by the second bit.

8.6.4 State Table

A state table is a tabular form of describing the operation of a synchronous circuit. Each row of the state table corresponds to a state of the circuit and each column corresponds to a combination of external inputs. The entries of the table denote the state transitions (next states) and the outputs associated with these transitions. The number of rows in the state table will be equal to the number of states of the circuit, and the number of columns will be same as the number of combinations of the inputs (2 for 1 input, 4 for 2 inputs, and so on).

A state table can be easily constructed from a state diagram. In fact, whatever information is available in a state diagram is also available in its state table and one can be obtained from the other.

Example 8.18

Construct state table for the state diagram of Fig. 8.40*b*.

Solution

There are three states in this circuit A, B, and C, therefore, its state table will contain three rows. There is one input (X) and one output (Y) variable. There is one column for each value of X, i.e., $X = 0$ and $X = 1$. Each entry of the table contains two pieces of information: next state and output separated by a comma. The first entry is for the next state and after it is output.

Let us start making entries in the state table starting from the state A. When the circuit is in state A, the external input present is 0, in response to a clock pulse the state does not change, i.e., the next state is also A and the output is 0. This is written in the first row and first column as A, 0. If the present input is 1 while the circuit is in state A, a clock pulse will cause next state to be B and output = 0. The corresponding entry in the first row second column will be B, 0.

Similarly, take the state B and find out the next state and the resulting output when $X = 0$ and when $X = 1$ and the corresponding entries are made in the second row under the appropriate columns. Same procedure will give entries for the third row corresponding to the state C.

The complete table is given in Table 8.15.

Table 8.15 ***State Table for the State Diagram of Fig.* 8.40*b***

Present state	**Next state, Output** *NS, Y*	
PS	$X = 0$	$X = 1$
A	A, 0	B, 0
B	C, 1	A, 0
C	A, 0	C, 1

8.6.5 State Assignment

For designing a clocked sequential circuit, the requirements (or specifications) may be specified as a set of statements. State diagram is constructed from the set of statements and state table can be prepared from the state diagram. For constructing the state diagram, normally the states are not specified in the binary form. Therefore, we make use of some letter symbols such as A, B, C, … etc. for the states. The exact number of states is also usually not known, but using the set of statements, arbitrary number of states may be chosen to satisfy the given requirements of the circuit. The states chosen in this way may contain more than the minimum number of states required. The inclusion of redundant states will help in the proper representation of the circuit.

For the design of any system, it is always desirable to design a minimal-cost system i.e. a system costing minimum money. There are two issues involved in the design of clocked sequential circuits. These are given below.

- The number of states must be minimum possible so as to be able to design a system with the minimum number of FLIP-FLOPs. Since one FLIP-FLOP has two states, therefore, the number of FLIP-FLOPs required can be determined from the number of states in the circuit. The redundant states get eliminated by *state reduction* method which will be discussed later.
- Combinational circuits are required to generate the excitation functions and the output. The combinational circuits required should also be designed for minimal cost. For this purpose, the states which have been labeled as A, B, C, … etc. have to be assigned binary values suitably. This process is known as the *state-assignment.* There may be a number of different possibilities for assigning binary values to the states. Each option will lead to different logic expressions for the FLIP-FLOP excitations and output, but will produce the required sequence of outputs for any given sequence of inputs. In a case in which it is required to have a specific output sequence for a given input sequence, the binary values of the

individual states may be of no consequence. However, the requirements of states for circuits whose external outputs are taken directly from the FLIP-FLOPs with binary sequence fully specified need only the specified binary values assigned to the states and therefore, no alternatives are available for such circuits. Therefore, for circuits requiring the specified input-output relationship, the binary values of the states are of no consequence and the state assignment should be made which produces a minimal-cost combinational circuit.

Rules for State Assignment

In the design of a sequential circuit, the complexity of the combinational circuit obtained depends on the chosen state assignment. There is no general procedure for the state assignment which produces the most cost effective design of the resulting combinational circuit, however, trial and error attempts at making state assignments are not practically feasible. The following thumb rules will help in generating simple combinational circuits.

Rule 1

Adjacent codes should be assigned to the states having the same next state for:

(a) each input combination
(b) different input combinations, if the next state can also be assigned adjacent codes
(c) some of the input combinations, not all

Rule 2

Adjacent codes should be assigned to the next state (s) of every present state.

Rule 3

Adjacent codes should be assigned to states that have the same outputs.

For a given state table, it may not be possible to achieve all the adjacencies of the above rules. Application of some of these rules may lead to conflicting state assignments, in such cases, the higher-priority rules should be given precedence. Even if the rules can be fully implemented, they do not guarantee an optimal assignment, i.e., these rules do not constitute an optimal algorithm.

Example 8.19

Partial state diagram of a sequential machine is shown in Fig. 8.45. Make suitable state assignment for obtaining minimal logic expression. Assume the machine with a single input and the complete state diagram contains seven states.

Solution

The portion of the state diagram shown has state transitions from the state A to C and from B to C for both the values of the input X and Y is the output. Since, there are 7 states, therefore, the number of FLIP-FLOPs required will be $\log_2^7 = 3$ (next higher integer). Its state

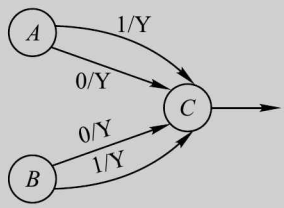


Fig. 8.45 ***State Diagram for Ex. 8.19***

assignment map can be prepared giving the state transitions from the present state to the next state. It is similar to a *K*-map. It is shown in Fig. 8.46. Q_3, Q_2 and Q_1 are the FLIP-FLOP outputs and the states will be assigned binary numbers in the order $Q_3 Q_2 Q_1$.

The two states, *A* and *B* can be assigned two adjacent cells according to Rule 1. One possible assignment is shown in the Fig. 8.46, for which $A = 100$ and $B = 110$. When the logic function is simplified using *K*-map or Quine-McCluskey method, combination of two adjacent cells will give a term with only two literals. Whereas if the states are assigned as 010 and 101, the expression will contain two minterms which cannot be combined.

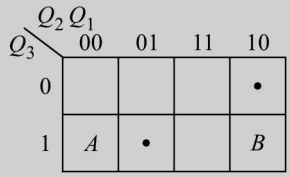


Fig. 8.46 ***State Assignment Map of Ex. 8.19***

Example 8.20

For the partial state diagram of a sequential circuit shown in Fig. 8.47 make suitable state assignment for obtaining minimal logic expression. Assume one single input and a total of 6 states in the state diagram.

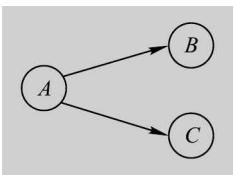


Fig. 8.47 ***State Diagram for Ex. 8.20***

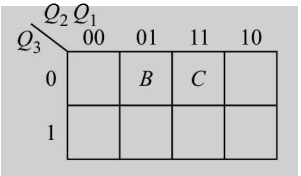


Fig. 8.48 ***State Assignment Map of Ex. 8.20***

Solution

State assignment map can be prepared similar to the Ex. 8.19. It is shown in Fig. 8.48.

One possible state assignment shown in the map gives $B = 001$ and $C = 011$.

Example 8.21

Table 8.16 gives state table of a sequential circuit. Give a good state assignment scheme.

Table 8.16 ***State Table for Ex. 8.21***

Present state *PS*	Next state, Output *NS, Y*	
	X = 0	*X* = 1
A	*B*, 1	*B*, 0
B	*C*, 0	*D*, 1
C	*E*, 1	*F*, 0
D	*F*, 0	*E*, 1
E	*G*, 0	*A*, 0
F	*A*, 0	*G*, 0
G	*B*, 0	*B*, 0

Solution

Since there are seven states in this circuit, therefore, it will have three FLIP-FLOPs FF3, FF2, and FF1 with state variables Q_3, Q_2, and Q_1 respectively. Application of the rules of state assignment is given below.

Rule 1

(a) Next state from the present states *A* and *G* is same. It is *B*. Therefore, the states *A* and *G* should be assigned adjacent codes.

(b) From the present state *C*, the next state is *E* for $X = 0$ and *F* for $X = 1$, whereas from the present state *D*, the next states are *F* and *E* for $X = 0$ and $X = 1$ respectively. Therefore, *C* and *D* may be assigned adjacent codes. Similarly, *E* and *F* may be assigned adjacent codes.

Rule 2

From the state *B*, the next states are *C* and *D*, therefore, *C* and *D* may be assigned adjacent states.
Similarly, *E* and *F*; *G* and *A* may be assigned adjacent states.

Rule 3

The outputs are same for the present states *A* and *C*; *B* and *D*, therefore, *A* and *C*; *B* and *D*, may be assigned adjacent codes.

Now, the state assignment is to be made so that as many as possible of these adjacencies can be accommodated. For this assignment map is prepared. Let us assign the state 000 to *A*, then *G* must be assigned an adjacent state. There are three possible adjacent states. Any one of them can be assigned to *G*, since *G* is not required to be adjacent to any other state. Similarly, all the other assignments are made and the resulting assignment map is shown in Fig. 8.49. The binary states are given below.

A-000
B-111
C-100
D-101
E-001
F-011
G-010

Q_3 \ Q_2Q_1	00	01	11	10
0	*A*	*E*	*F*	*G*
1	*C*	*D*	*B*	

Fig. 8.49 ***State Assignment Map of Ex. 8.21***

8.6.6 Design Procedure

For the design of any clocked sequential circuit, the following general procedure is used.

1. The design specifications may be specified in the form of a set of statements, state table, or state diagram. In case a set of statements is given, a state diagram can be constructed and from the state diagram, a state table can be constructed. In general, a state table is needed for the design.
2. When a state diagram is constructed, it is a normal practice to use some letter symbols for the states because of the non-availability of binary values for the states. Also, a number of redundant states may be included to avoid any confusion while constructing the state diagram. The state diagram is not unique. The number of states may be reduced in case there are equivalent states (discussed below) to obtain a reduced state table which will contain the minimum number of states. Since, the number of

FLIP-FLOPs required depends upon the number of states, therefore, this step will ensure minimum number of FLIP-FLOPs.

3. Assign binary values to the letter symbols for each state, i.e., the state assignment is to be done.
4. Choose the type of FLIP-FLOP to be used. *J-K* FLIP-FLOP is the most general type of FLIP-FLOP. It can be *T*-type or *D*-type also depending upon the circuit requirements.
5. Construct state transition table and output table and from this obtain *K*-maps for FLIP-FLOP excitations and output. Minimise the *K*-maps and obtain minimal logic expressions for excitations and output.
6. Construct logic circuit incorporating the FLIP-FLOPs and the combinational circuits based on the expressions obtained in step 5.

8.6.7 State Equivalence and Minimisation

Two states are said to be equivalent if, for each input condition, they give exactly the same output and go to the same next state. In a state table, when two states are found to be equivalent, one of them is eliminated without altering the input-output relations. This process is referred to as *state-reduction.* Using the concept of equivalent states and state reduction, all possible equivalent states are determined and the reduced state table is obtained which will contain the minimum number of states. This process minimises the number of states.

Example 8.22

For the state table given in Table 8.17, obtain reduced state table with minimum number of states.

Table 8.17 ***State Table for Ex.* 8.22**

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	c	b	0	0
b	d	c	0	0
c	g	d	1	1
d	e	f	1	0
e	f	a	0	1
f	g	f	1	0
g	f	a	0	1

Solution

(i) By observing Table 8.17, we see that from the initial state e, the next state is f for $X = 0$ and it is a for $X = 1$. The output is 0 and 1 for $X = 0$ and $X = 1$ respectively. Similarly, from the initial state g also, the next state is f and a for $X = 0$ and $X = 1$ respectively; and the output is 0 and 1 for $X = 0$ and $X = 1$ respectively. From this, we conclude that the two states e and g are equivalent, since for each input condition, they go to the same next state and give same output. We can eliminate one of them, say g. Thus g is replaced by e wherever it occurs.

(ii) After replacing g by e in the state table, we notice that the states d and f are equivalent. Thus, one of them, say d, can be eliminated.

The effect of eliminating equivalent states is illustrated in Table 8.18 and the reduced state table is given in Table 8.19. The reduced state table contains 5 states which is the minimum number of states required to implement the circuit represented by the given state table.

Table 8.18 ***Effect of Eliminating Equivalent States***

Present state	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	c	b	0	0
b	$(d)f$	c	0	0
c	$(g)e$	$(d)f$	1	1
d	e	f	1	0
e	f	a	0	1
f	$(g)e$	f	1	0
g	f	a	0	1

Table 8.19 ***Reduced State Table***

Present state	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	c	b	0	0
b	f	c	0	0
c	e	f	1	1
e	f	a	0	1
f	e	f	1	0

8.6.8 Design Examples

Using the various concepts discussed for the design of clocked sequential circuits, any synchronous sequential circuit can be designed. The following designs will illustrate clearly the application of various concepts.

Example 8.23

Design a minimal clocked sequential circuit for the reduced state table given in Table 8.19.

Solution

The reduced state table was obtained, using state equivalence and minimisation concepts, from Table 8.17. The next step in the design is to assign binary values to the states. Since, there are 5 states in this machine, therefore, the number of FLIP-FLOPs required is three. Let the three FLIP-FLOPs be designated as FF2, FFl, and FF0; and their outputs are Q_2, Q_1, and Q_0 respectively. Therefore, every state will be a 3-bit number with Q_2 value as the MSB and Q_0 as LSB. Assume D FLIP-FLOPs.

From application of the rules of state assignment, we find that

(i) the next states are same for the states c and f for $X = 0$ and $X = 1$, therefore, the states c and f should be assigned adjacent codes (Rule 1 (a)).

(ii) from the states b and e, the next state is f for $X = 0$, therefore, the states b and e should be assigned adjacent codes (Rule 1 (c)).

State assignment map can then be constructed. Let us assume $a = 000$ as the initial state. The state assignment map is shown in Fig. 8.50.

Therefore, the states assigned are:

$a = 000$
$b = 011$
$c = 010$
$e = 111$
$f = 110$

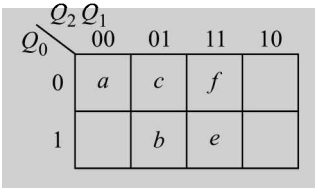


Fig. 8.50 ***State Assignment Map***

The state transition and output table is constructed next. From the state a (000) the next state is $c = 010$. When $X = 0$. The output corresponding to this is $Y = 0$, these entries are made in the first row. Similarly all the entries are made in this table.

Since D-type of FLIP-FLOPs have been chosen here, therefore, it is not necessary to construct excitation table separately. For a D-type FLIP-FLOP for next state to be 0, the D input must be 0 just before the clock pulse. Similarly, for the next state to be 1, its D input must be 1. Therefore Q_2^*, Q_1^* and Q_0^* values are the same as the excitation values of D_2, D_1, and D_0 respectively. The state transition and output table is given in Table 8.20.

Table 8.20 ***State Transition and Output Table***

Present state			Input	Next state			Output
Q_2	Q_1	Q_0	X	Q_2^*	Q_1^*	Q_0^*	Y
0	0	0	0	0	1	0	0
0	1	1	0	1	1	0	0
0	1	0	0	1	1	1	1
1	1	1	0	1	1	0	0
1	1	0	0	1	1	1	1
0	0	0	1	0	1	1	0
0	1	1	1	0	1	0	0
0	1	0	1	1	1	0	1
1	1	1	1	0	0	0	1
1	1	0	1	1	1	0	0

The next step involves finding out the minimised logic expressions for D_2, D_1, D_0, and Y in terms of Q_2, Q_1, Q_0, and X. This is a combinational logic design problem. For this K-maps are constructed for D_2, D_1, D_0, and Y and are minimised. The K-maps are shown in Fig. 8.51.

Since, there are eight states possible using three bits, out of which only five states are required for this circuit. The other three states do not exist and are therefore, taken as don't cares (X's).

The minimised expressions are:

$$D_2 = Q_1\overline{Q}_0 + Q_1\overline{X}$$
$$D_1 = \overline{Q}_2 + \overline{Q}_0 + \overline{X}$$
$$D_0 = \overline{Q}_1 X + Q_1\overline{Q}_0\overline{X}$$
$$Y = Q_1\overline{Q}_0\overline{X} + \overline{Q}_2 Q_1\overline{Q}_0 + Q_2 Q_0 X$$

The complete circuit is given in Fig. 8.52.

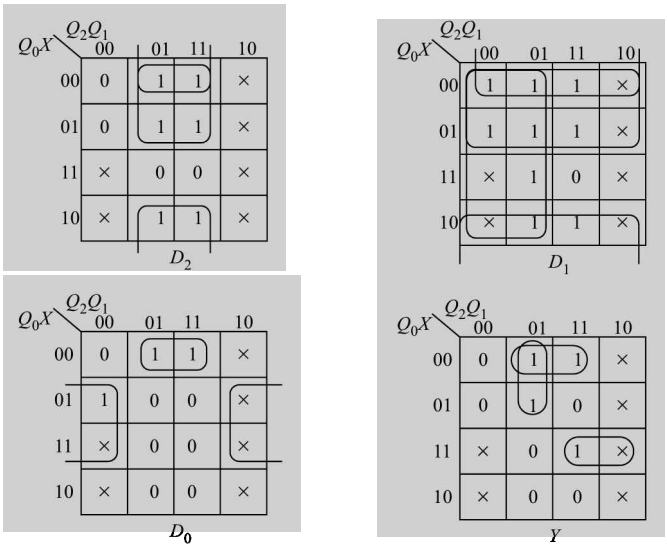


Fig. 8.51 ***K-Maps for D_2, D_1, D_0, and Y***

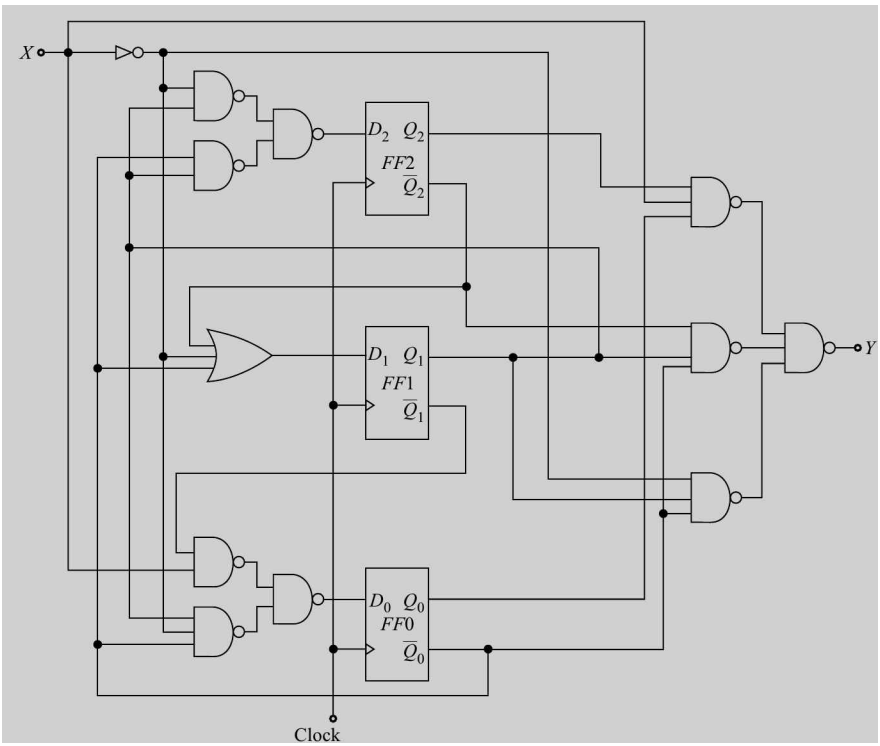


Fig. 8.52 ***Sequential Circuit for Ex. 8.23***

Example 8.24

For the reduced state table given in Table 8.19, design the circuit assuming the state assignment given below. Compare the hardware requirements of this state assignment with the state assignment done in Example 8.23.

$$a = 000,\ b = 001,\ c = 010,\ e = 011, \quad \text{and} \quad f = 100.$$

Solution

The state transition and output table is given in Table 8.21, and the *K*-maps are given in Fig. 8.53. The unused states have been taken as don't care conditions.

Table 8.21 ***State Transition and Output Table for Ex.* 8.24**

Present state			Input	Next state			Output
Q_2	Q_1	Q_0	X	Q_2^*	Q_1^*	Q_0^*	Y
0	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0
0	0	1	0	1	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	1
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0

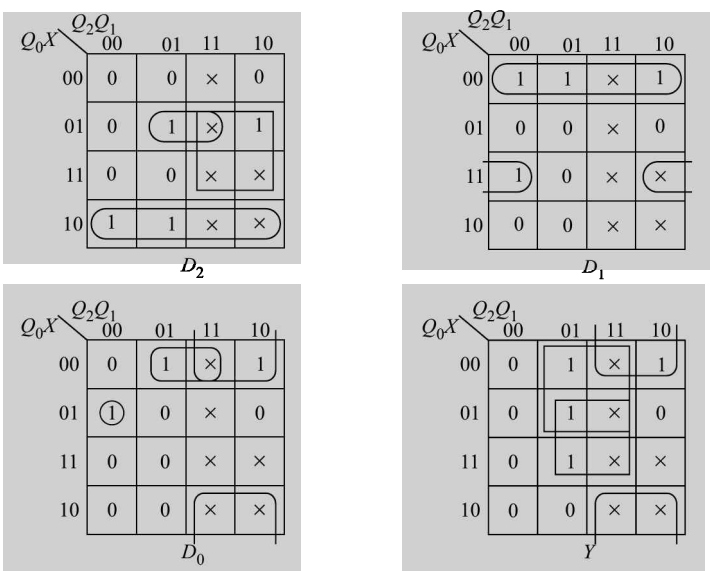


Fig. 8.53 ***K-Maps for Table 8.21***

The minimised expressions are:

$$D_2 = Q_0 \cdot \overline{X} + Q_2 \cdot X + Q_1 \cdot \overline{Q}_0 \cdot X$$
$$D_1 = \overline{Q}_0 \cdot \overline{X} + \overline{Q}_1 \cdot Q_0 \cdot X$$
$$D_0 = Q_2 \cdot \overline{X} + Q_1 \cdot \overline{Q}_0 \cdot \overline{X} + \overline{Q}_2 \cdot \overline{Q}_1 \cdot \overline{Q}_0 \cdot X$$
$$Y = Q_1 \overline{Q}_0 + Q_1 \cdot X + Q_2 \cdot \overline{X}$$

The complete circuit is given in Fig. 8.54.

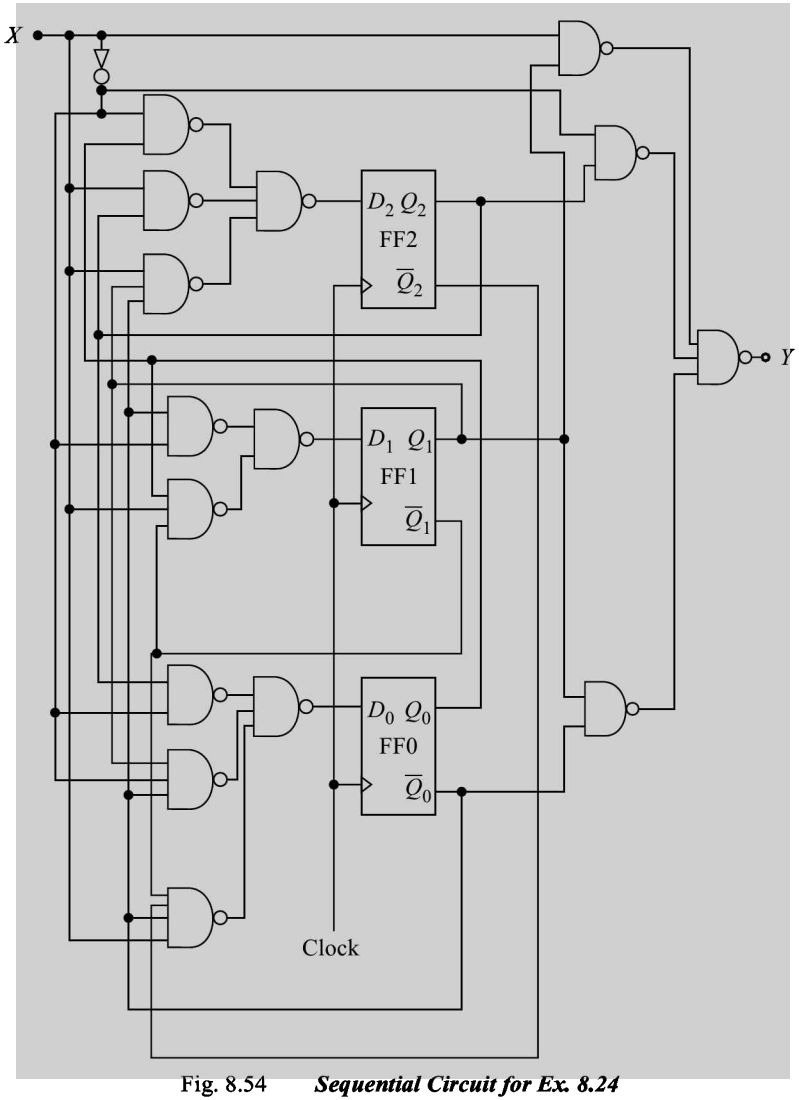


Fig. 8.54 ***Sequential Circuit for Ex. 8.24***

The circuit can be designed using *J-K* FFs also (Prob. 8.30).
The requirement of gates for the circuits of Figs. 8.52 and 8.54 are given below.

For Fig. 8.52

3-input NAND gates-4 (one NAND gate can be shared between D_0 and Y)
2-input NAND gates-5
3-input OR gate-1

For Fig. 8.54

4-input NAND gate-1
3-input NAND gates-6
2-input NAND gates-7 (two NAND gates can be shared between D_0 and Y)

From the requirement of gates by two different state assignments, we can conclude that a carefully done state assignment will lead to a circuit requiring lesser number of gates.

Example 8.25

A synchronous sequential circuit is to be designed having a single input X and a single output Y to detect single change of level (from 0 to 1 or from 1 to 0) in a 3-bit word and produce an output $Y = 1$, otherwise $Y = 0$. When a new 3-bit word is to come, the circuit must be at its initial (reset) state and there should be a time delay of one clock cycle between the words.

Solution

There are eight possible words of three bits. These are: 000, 001, 010, 011, 100, 101, 110, and 111. The number of level change is one in 001, 011, 100, and 110. When any one of these words is applied bit by bit at the X input, Y must be 1 after three clock cycles, i.e., when the third bit is applied. A state diagram is to be constructed first for this. The state diagram is given in Fig. 8.55.

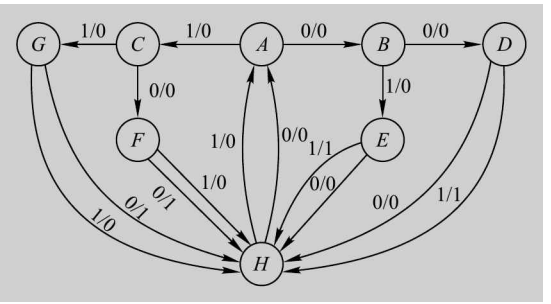


Fig. 8.55 ***State Diagram of Ex. 8.25***

A is the initial state. For each 3-bit word, find the next state and output bit by bit. Let us take the input word 000. When first 0 is applied, the circuit goes to state B and output is 0, next 0 will take the circuit to D with output 0, the third 0 will take the circuit to H with output 0. Therefore, for this 3-bit word, the output is 0 at the end of the third bit. Hence, we conclude that the number of level changes from the first to the third bit is not 1. When the fourth clock pulse appears, the circuit is reset, i.e., it reaches its initial state A irrespective of the value of X (0 or 1).

Now consider input word 110. From the reset state the circuit goes to $C \rightarrow G \rightarrow H$ and produces an output of 1 at the third clock pulse indicating 1 level change. On fourth clock pulse, the circuit resets. Similarly, it can be verified for each input word.

Next a state table (Table 8.22) is constructed from the state diagram.

Table 8.22 ***State Table of Ex. 8.25***

Present State	Next state, Output	
	$X = 0$	$X = 1$
A	B, 0	C, 0
B	D, 0	E, 0
C	F, 0	G, 0
D	H, 0	H, 1
E	H, 0	H, 1
F	H, 1	H, 0
G	H, 1	H, 0
H	A, 0	A, 0

Now the state reduction table is to be constructed. From the state table, we observe the following.

(i) From the present states D and E, the next state and output are same for $X = 0$ and $X = 1$, therefore, these two states are equivalent and state D is eliminated and replaced by state E.

(ii) From the present states F and G, the next state and output are same for $X = 0$ and $X = 1$, therefore, these two states are equivalent and state G is eliminated and replaced by state F.

The reduced state table is given in Table 8.23 and Fig. 8.56 gives reduced state diagram.

Table 8.23 ***Reduced State Table of Ex. 8.25***

Present state	Next state, Output	
	$X = 0$	$X = 1$
A	B, 0	C, 0
B	E, 0	E, 0
C	F, 0	F, 0
E	H, 0	H, 1
F	H, 1	H, 0
H	A, 0	A, 0

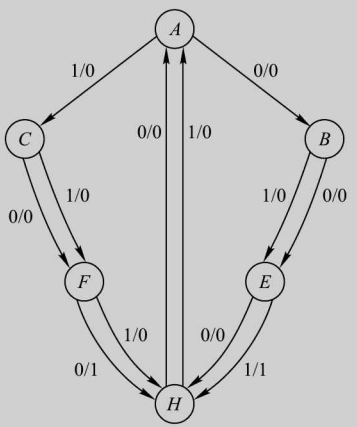


Fig. 8.56 ***Reduced State Diagram of Ex. 8.25***

Since, there are six states, therefore, the number of FLIP-FLOPs required is 3. Let us assume D FLIP-FLOPs.

From the states E and F, the next state is H for both the values of X, therefore, E and F should be assigned adjacent codes.

States having same outputs should be assigned adjacent states. These are:

$$AB, AC, BC, AH, BH, CH$$

The state assignment map is shown in Fig. 8.57.

The states assigned are:

$A = 000$ $\quad$ $E = 100$

$B = 010$ $\quad$ $F = 101$

$C = 001$ $\quad$ $H = 011$

Q_0 \ Q_2Q_1	00	01	11	10
0	A	B		E
1	C	H		F

Fig. 8.57 ***State Assignment Map of Ex. 8.25***

State transition and output table is constructed on the basis of state assignment. It is given in Table 8.24 and K-maps are given in Fig. 8.58.

The minimised logic expressions are:

$$D_2 = Q_1\overline{Q}_0 + \overline{Q}_2\overline{Q}_1Q_0$$

$$D_1 = Q_2 + \overline{Q}_1\overline{Q}_0\overline{X}$$

$$D_0 = Q_2 + \overline{Q}_1X + \overline{Q}_1Q_0$$

$$Y = Q_2\overline{Q}_0X + Q_2Q_0\overline{X}$$

Table 8.24 ***State Transition and Output Table of Ex.* 8.25**

Present state			Input	Next state			Output
Q_2	Q_1	Q_0	X	Q_2^*	Q_1^*	Q_0^*	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	0	0	0
0	0	1	0	1	0	1	0
1	0	0	0	0	1	1	0
1	0	1	0	0	1	1	1
0	1	1	0	0	0	0	0
0	0	0	1	0	0	1	0
0	1	0	1	1	0	0	0
0	0	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	0	1	1	0	1	1	0
0	1	1	1	0	0	0	0

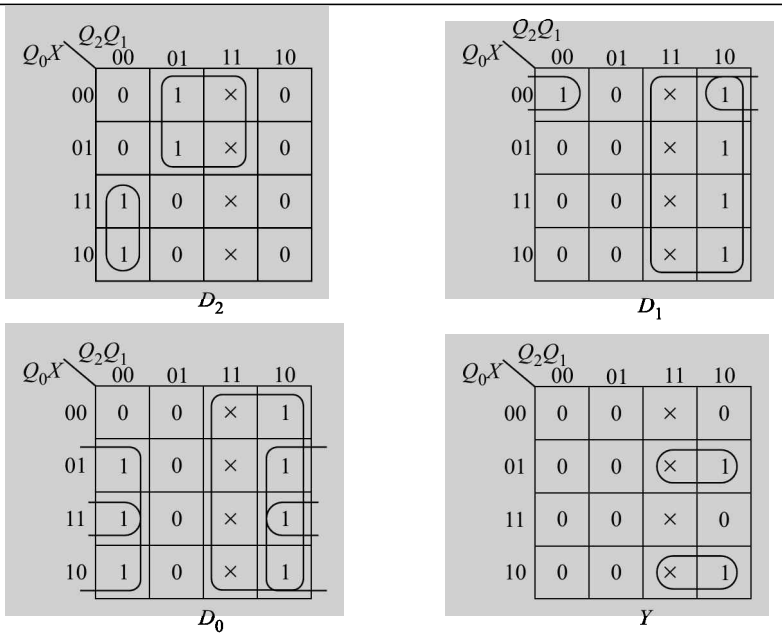


Fig. 8.58 ***K-Maps of Ex. 8.25***

The complete circuit can be drawn using three *D* FLIP-FLOPs and NAND gates.

Example 8.26

Design a serial adder to add two binary numbers.

Solution

We are familiar with the process of decimal addition using paper and pencil. In this, digits in the same significant position are added starting from the LSD alongwith a carry generated from the addition of digits from the previous significant position. A similar process is used for the serial addition of binary numbers.

Let the two binary inputs be X_1 and X_2. At X_1 and X_2 inputs are applied sequentially. Assume

$$\begin{aligned} X_1 &= 10010110 \\ X_2 &= 11011110 \\ \hline Y &= 101110100 \end{aligned}$$

The sum output Y does not depend on the present inputs alone, it also depends on the carry from the previous position. In the LSB (b_0) position both the input bits are 0 and the output is also 0; in the next position b_1, the sum of 1 and 1 is 0 and a carry c_1 is generated; in the next position b_2, the two 1's alongwith the carry c_1 are to be added resulting in sum bit $Y = 1$ and so on. Therefore, a memory device is required to keep track of the carry input. The memory should have two states 0 and 1 corresponding to carry = 0 or 1. We can construct a state transition and output table from the above discussion.

Table 8.25 gives the state transition and output table and Fig. 8.59 gives *K*-maps for the *D* input of *D* FLIP-FLOP and output *Y*.

Table 8.25 ***State Transition and Output Table for Serial Adder***

Present state	Inputs		Next state	Output
$PS(Q)$	X_1	X_2	$NS(Q^*)$	Y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

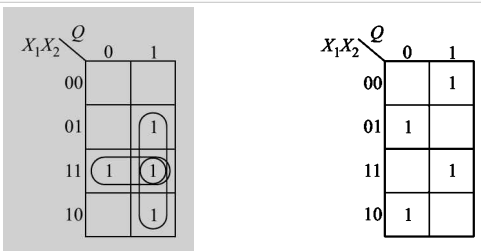


Fig. 8.59 ***K-Maps for*** `FLIP-FLOP` ***Input D and Output Y***

The logic expressions for D and Y are:

$$D = X_1X_2 + X_1Q + X_2Q$$

$$Y = \overline{X}_1\,\overline{X}_2Q + \overline{X}_1X_2\,\overline{Q} + X_1X_2Q + X_1\,\overline{X}_2\,\overline{Q}$$

Its circuit is shown in Fig. 8.60.

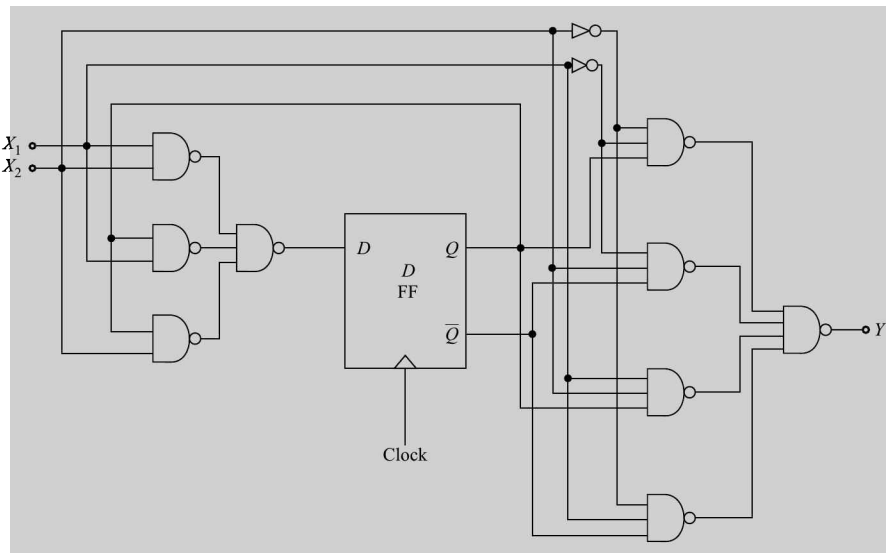


Fig. 8.60 ***Serial Adder Circuit***

Example 8.27

Design a sequence detector circuit to detect a serial input sequence of 1010. It should produce an output 1 when the input pattern has been detected.

Solution

Let the input string be 10101010, the desired output string can be determined, which is given below

Input X	1	0	1	0	1	0	1	0
Output Y	0	0	0	1	0	1	0	1

A state diagram can be constructed for the required input-output relationship.

Let us start with the initial state A. If the input is 0, the detection cycle must not start and therefore, the state remains the same and output is 0. When it is 1, it should go to the next state B with output as 0. In the present state B, if the input is 0 the next state will be C and output 0, while for an input 1, it is to be counted as the first correct bit in the string (since it is the second 1 bit from the start) and the state of the circuit should remain unchanged. In the present state C, if an input bit 0 occurs, the circuit should go back to the initial state (since it is second consecutive 0), while an input 1 causes state transition to next state D. When the circuit is in the state D, a 0 input will detect the correct sequence and will produce an output of 1. If the input is 1, it is a second consecutive 1 which must take the circuit to the state B. The complete state diagram is shown in Fig. 8.61. Its state table is constructed as given in Table 8.26.

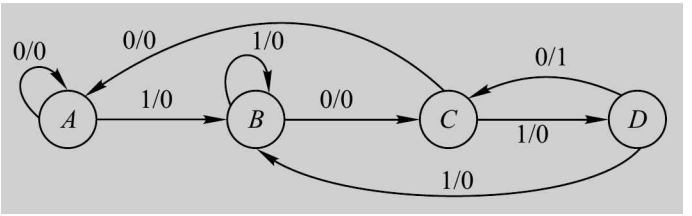


Fig. 8.61 ***State Diagram of Sequence Detector Circuit***

There are four states in this circuit and no equivalent states are present. Therefore, two D-type `FLIP-FLOPs` can be used for the implementation of this circuit.

Table 8.26 ***State Table of Sequence Detector***

Present state	Next state, Output	
	$X = 0$	$X = 1$
A	A, 0	B, 0
B	C, 0	B, 0
C	A, 0	D, 0
D	C, 1	B, 0

The two states B and D must be assigned adjacent codes, since the next state is same from these states for $X = 0$ and $X = 1$. Therefore, the following state assignment is made.

$$A \rightarrow 00$$
$$B \rightarrow 01$$
$$C \rightarrow 10$$
$$D \rightarrow 11$$

Table 8.27 gives state transition and output table. From this K-maps are constructed for the FLIP-FLOP inputs D_1 and D_0; and output Y. These are given in Fig. 8.62.

Table 8.27 ***State Transition and Output Table***

Present state		Input	Next state		Output
Q_1	Q_0	X	Q_1^*	Q_0^*	Y
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	0	0	0
1	1	0	1	0	1
0	0	1	0	1	0
0	1	1	0	1	0
1	0	1	1	1	0
1	1	1	0	1	0

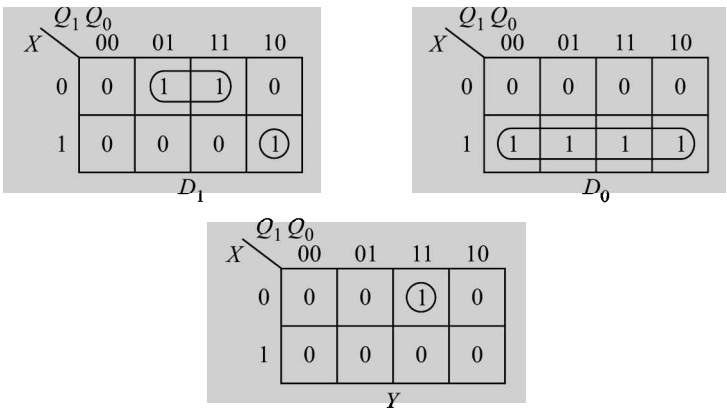


Fig. 8.62 ***K-Maps of Ex. 8.27***

The minimised logic expressions are:

$$D_1 = Q_0 \overline{X} + Q_1 \overline{Q}_0 X$$
$$D_0 = X$$
$$Y = Q_1 Q_0 \overline{X}$$

Its circuit diagram is given in Fig. 8.63.

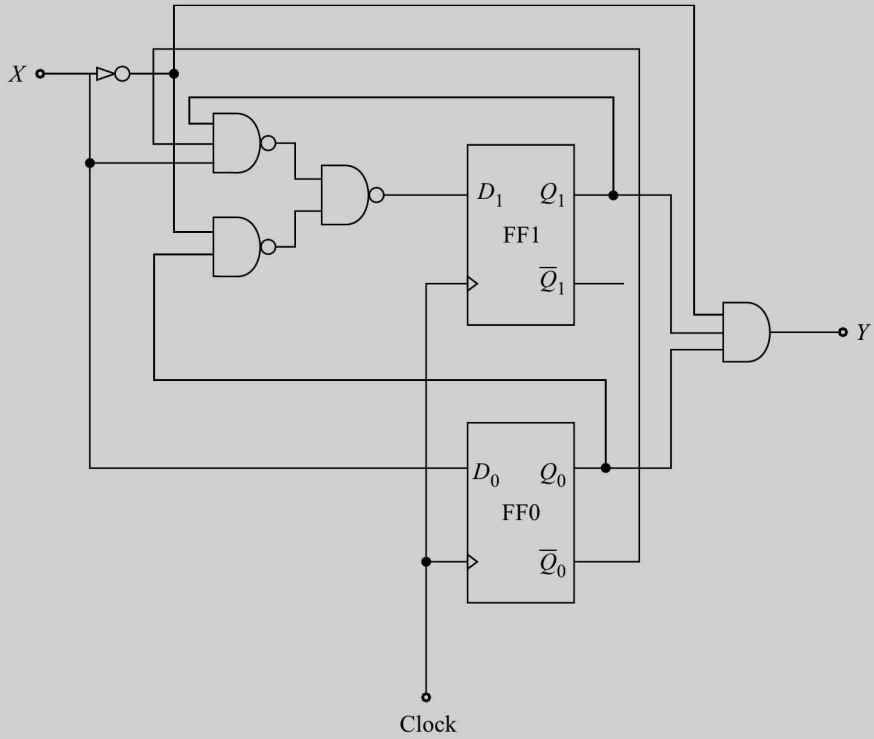


Fig. 8.63 ***Circuit of Sequences Detector***

8.7 ASYNCHRONOUS SEQUENTIAL CIRCUITS

The two types of sequential circuits were introduced in Section 7.1. The design of clocked sequential circuits have been discussed in section 8.6. Another important class of sequential circuits, i.e. asynchronous sequential circuits have been discussed here.

8.7.1 Asynchronous versus Synchronous Sequential Circuits

- In a clocked sequential circuit a change of state occurs only in response to a synchronizing clock pulse. All the FLIP-FLOPs are clocked simultaneously by a common clock pulse. In an asynchronous sequential circuit, the state of the circuit can change immediately when an input change occurs. It does not use a clock.
- In clocked sequential circuits input changes are assumed to occur between clock pulses. The circuit must be in the stable state before next clock pulse arrives.

  In asynchronous sequential circuits input changes should occur only when the circuit is in a stable state.

- In clocked sequential circuits, the speed of operation depends on the maximum allowed clock frequency.
  Asynchronous sequential circuits do not require clock pulses and they can change state with the input change. Therefore, in general the asynchronous sequential circuits are faster than the synchrous sequential circuits.
- In clocked sequential circuits, the memory elements are clocked FLIP-FLOPs.
  In asynchronous sequential circuits, the memory elements are either unclocked FLIP-FLOPs (latches) or gate circuits with feedback producing the effect of latch operation.
- In clocked sequential circuits, any number of inputs can change simultaneously (during the absence of the clock).

  In asynchronous sequential circuits only one input is allowed to change at a time in the case of the level inputs and only one pulse input is allowed to be present in the case of the pulse inputs. If more than one level inputs change simultaneously or more that one pulse input is present, the circuit makes erroneous state transitions due to different delay paths for each input variable.

8.7.2 Applications of Asynchronous Sequential Circuits

- An asynchronous circuit is preferred over synchronous circuit when high speed of operation is required since asynchronous sequential circuits respond immediately whenever there is a change in any input variable without having to wait for a clock pulse.
- Asynchronous sequential circuits are useful in applications in which input signals may change at any time.
- Asynchronous sequential circuits cost less than the sequential circuits, therefore, for economical reasons, they find useful applications.

8.7.3 Asynchronous Sequential Machine Modes

There are two modes of operations of asynchronous sequential machines depending upon the type of input signals. These are:

- Fundamental Mode
- Pulse Mode

Fundamental Mode

In a fundamental mode circuit, all of the input signals are considered to be levels. Fundamental mode operation assumes that the input signals will be changed only when the circuit is in a stable state and that only one variable can change at a given time.

Pulse Mode

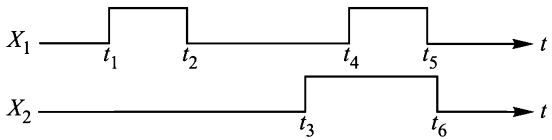
In pulse mode circuits, the inputs are pulses rather than levels. In this mode of operation the width of the input pulses is critical to the circuit operation. The input pulse must be long enough for the circuit to respond to the input but it must not be so long as to be present even after new state is reached. In such a situation the state of the circuit may make another transition.

The minimum pulse width requirement is based on the propagation delay through the next-state logic (Fig 7.1). The maximum pulse width is determined by the total propagation delay through the next state logic and the memory elements.

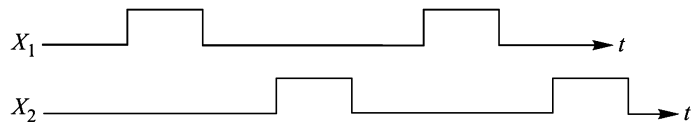
Both fundamental and pulse mode asynchronous sequential circuits use unclocked S-R FLIP-FLOPs or latches.

In pulse-mode operation, only one input is allowed to have pulse present at any time. This means that when pulse occurs on any one input, while the circuit is in stable state, pulse must not arrive at any other input. Figure 8.40 illustrates unacceptable and acceptable input pulse change. X_1 and X_2 are the two inputs to a pulse mode circuit. In Fig. 8.64*a* at time t_3 pulse at input X_2 arrives.

While this pulse is still present, another pulse at X_1 input arrives at t_4. Therefore, this kind of the presence of pulse inputs is not allowed.



(a) Unacceptable Pulse Mode Input Changes



(b) Acceptable Pulse Mode Input Changes

Fig. 8.64

8.7.4 Analysis of Asynchronous Sequential Machines

Analysis of asynchronous sequential circuits operation in fundamental mode and pulse mode will help in clearly understanding the asynchronous sequential circuits.

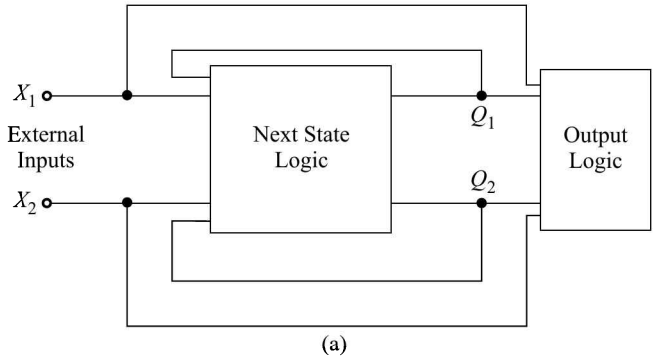
Fundamental Mode Circuits

Fundamental mode circuits are of two types:

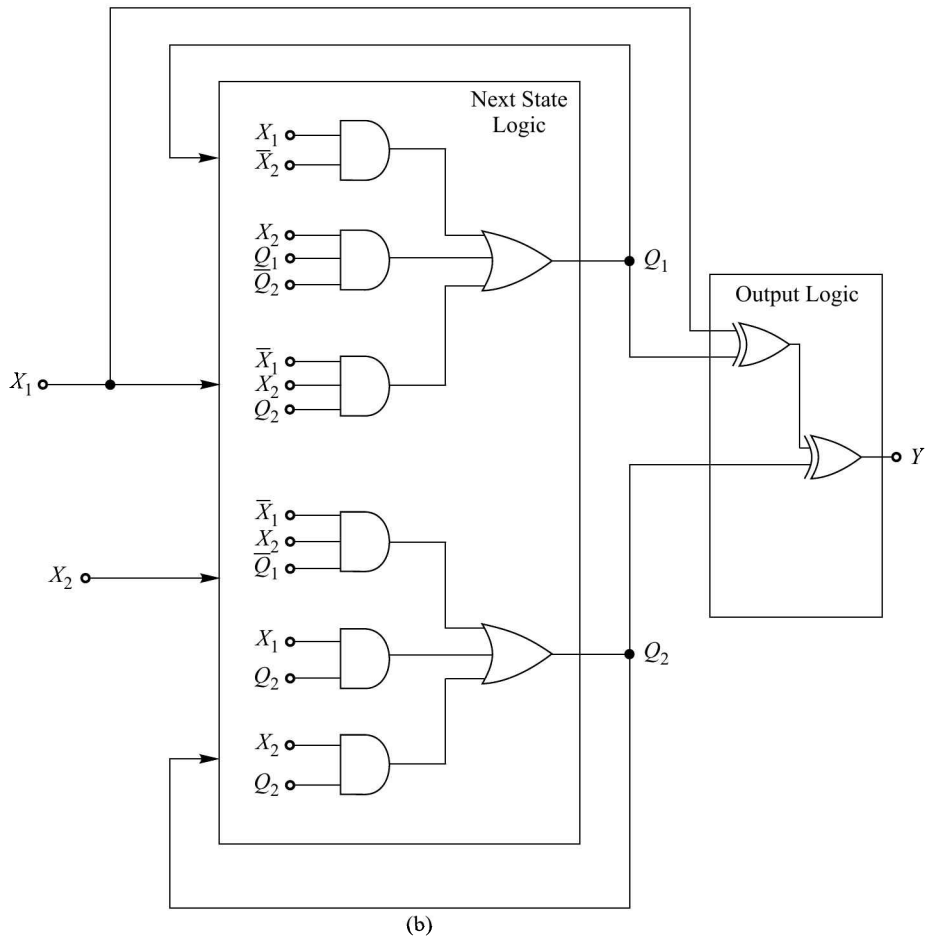
- Circuits without latches
- Circuits with latches

Circuits Without Latches Consider a fundamental mode circuit shown in Fig. 8.65. This circuit has only gates and no explicit memory elements are present. There are two feedback paths from Q_1 and Q_2 to the next-state logic circuit. This feedback creates the latching effect due to delays, necessary to produce a sequential circuit. It may be noted that a memory element latch is created due to feedback in gate circuit (Fig. 7.4).

The first step in the analysis is to identify the *states* and the *state variables.* The combination of level signals from external sources X_1, X_2 is referred to as the *input state* and X_1, X_2 are the *input state variables.*



(a)



(b)

Fig. 8.65 ***Fundamental Mode Asynchronous Sequential Circuit Without Latch (a) Block Diagram (b) Circuit Diagram***

The combination of the outputs of memory elements are known as *secondary*, or *internal states* and these variables are known as *internal* or *secondary state variables*. Here, Q_1 and Q_2 are the internal variables since no explicit elements are present. The combination of both, input state and the secondary state (Q_1, Q_2, X_1, X_2) is known as the *total state*. Y is the output variable.

The next secondary state and output logic equations are derived from the logic circuit in the next-state logic block. The next secondary state variables are denoted by Q_1^+ and Q_2^+ these are given by

$$Q_1^+ = X_1 \overline{X}_2 + \overline{X}_1 X_2 Q_2 + X_2 Q_1 \overline{Q}_2 \tag{8.6}$$

$$Q_2^+ = \overline{X}_1 X_2 \overline{Q}_1 + X_1 Q_2 + X_2 Q_2 \tag{8.7}$$

$$Y = X_1 \oplus Q_1 \oplus Q_2 \tag{8.8}$$

Here, Q_1 and Q_2 are the present secondary state variables when $X_1 X_2$ input-state variables occur, the circuit goes to next secondary state. A state table shown in Table 8.28 is constructed using these logic equations. If the resulting next secondary state is same as the present state, i.e. $Q_1^+ = Q_1$ and $Q_2^+ = Q_2$, the total state $Q_1 Q_2 X_1 X_2$ is said to be *stable*. Otherwise it is unstable. The stability of the next total state is also shown in Table 8.28.

Transition Table A state table can be represented in another form known as *transition table*. The transition table for the state table of Table 8.28 is shown in Fig. 8.66.

In a transition table, columns represent input states (one column for each input state) and rows represent secondary states (one row for each secondary state). The next secondary state values are written into the squares, each indicating a total state. The stable states are circled. For any given present secondary state ($Q_1 Q_2$), the next secondary state is located in the square corresponding to row for the present secondary state and the column for the input state ($X_1 X_2$). For example, for $Q_1 Q_2 = 11$ and $X_1 X_2 = 00$, the next secondary state is 00 (third row, first column) which is an unstable state.

Table 8.28 ***State Table***

Present total state				Next total state				Stable total state	Output
Q_1	Q_2	X_1	X_2	Q_1^+	Q_2^+	X_1	X_2	Yes/No	Y
0	0	0	0	0	0	0	0	Yes	0
0	0	0	1	0	1	0	1	No	0
0	0	1	1	0	0	1	1	Yes	1
0	0	1	0	1	0	1	0	No	1
0	1	0	0	0	0	0	0	No	1
0	1	0	1	1	1	0	1	No	1
0	1	1	1	0	1	1	1	Yes	0
0	1	1	0	1	1	1	0	No	0
1	1	0	0	0	0	0	0	No	0
1	1	0	1	1	1	0	1	Yes	0
1	1	1	1	0	1	1	1	No	1
1	1	1	0	1	1	1	0	Yes	1
1	0	0	0	0	0	0	0	No	1
1	0	0	1	1	0	0	1	Yes	1
1	0	1	1	1	0	1	1	Yes	0
1	0	1	0	1	0	1	0	Yes	0

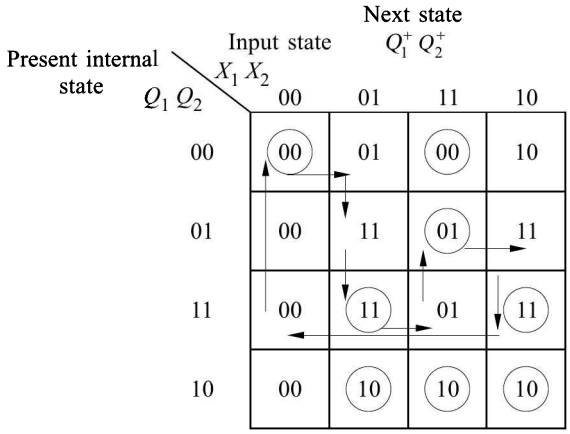


Fig. 8.66 ***Transition Table for Table 8.28***

For a given input sequence, the total state sequence can be determined from the transition table.

Example 8.28

For the transition table shown in Fig 8.66, the initial total state is $Q_1 Q_2 X_1 X_2 = 0000$. Find the total state sequence for an input sequence $X_1 X_2 = 00, 01, 11, 10, 00$.

Solution

For a given internal state of the circuit, a change in the value of the circuit input causes a *horizontal* move in the transition table to the column corresponding to the new input value. A change in the internal state of the circuit is reflected by a vertical move. Since a change in the input can occur only when the circuit is in a stable state, a horizontal move can emanate only from a circled entry.

The initial total state is 0000 (first row, first column) which is a stable state. When the input state changes from 00 to 01, the circuit makes a transition (horizontal move) from the present total state to the next total state 0101 (first row, second column) which is unstable. Next, the circuit makes another transition from 0101 to 1101 (vertical move) (second row, second column) which is also an unstable state. Finally in the next transition (vertical move) it comes to stable state 1101 (third row, second column). All these transitions are indicated by arrows. Thus we see that a single input change produces two secondary state changes before a stable total state is reached. If the input is next changed to 11 the circuit goes to total state 0111 (horizontal move) which is unstable and then to stable total state 0111 (vertical move). Similarly, the next input change to 10 will take the circuit to unstable total state 1110 (horizontal move) and finally to stable total state 1110 (vertical move). A change in input state from 10 to 00 causes a transition to unstable total state 0000 (horizontal move) and then to stable total state 0000 (vertical move), completing the state transitions for the input sequence. All the state transitions are indicated by arrows.

The total state sequence is

From the above discussions, we see that from the logic diagram of an asynchronous sequential circuit, logic equations, state table, and transition table can be determined. Similarly, from the transition table, logic equations can be written and the logic circuit can be designed.

Flow table In asynchronous sequential circuits design, it is more convenient to use *flow table* rather than transition table. A flow table is basically similar to a transition table except that the internal states are represented symbolically rather than by binary states. The column headings are the input combinations and the entries are the next states, and outputs. The state changes occur with change of inputs (one input change at a time) and next state logic propagation delay.

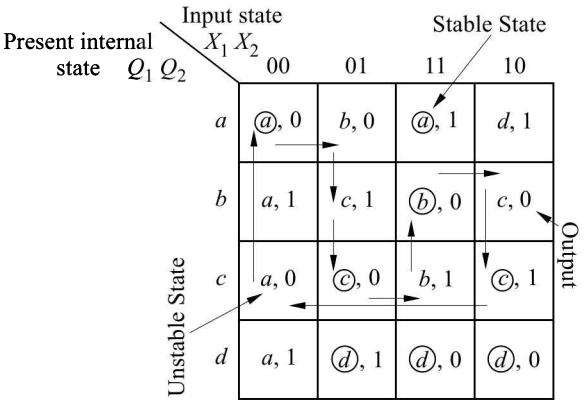


Fig. 8.67 ***Flow Table***

The flow of states from one to another is clearly understood from the flow table. The transition table of Fig. 8.66 constructed as a flow table is shown in Fig. 8.67. Here, *a*, *b*, *c*, and *d* are the states. The binary value of the output variable is indicated inside the square next to the state symbol and is separated by a comma. A stable state is circled.

From the flow table, we observe the following behaviour of the circuit.

When $X_1X_2 = 00$, the circuit is in state ⓐ. It is a stable state. If X_2 changes to 1 while $X_1 = 0$, the circuit goes to state *b* (horizontal move) which is an unstable state. Since *b* is an unstable state, the circuit goes to *c* (vertical move), which is again an unstable state. This causes another vertical move and finally the circuit reaches a stable state ⓒ. Now consider X_1 changing to 1 while $X_2 = 1$, there is a horizontal movement to the next column. Here *b* is an unstable state and therefore, there is a vertical move and the circuit comes to a stable state ⓑ. Next change in X_2 from 1 to 0 while X_1 remaining 1 will cause horizontal move to state *c* (unstable state) and finally to stable state ⓒ due to the vertical move. Similarly changing X_1 from 1 to 0 while $X_2 = 0$ will cause the circuit to go to the unstable state *a* and finally to stable state ⓐ. The flow of circuit states are shown by arrows.

In the flow table of Fig. 8.67 there are more than one stable states in rows. For example, the first row contains stable states in two columns. If every row in a flow table has only one stable state, the flow table is known as a *primitive flow table*.

From a flow table, transition table can be constructed by assigning binary values to each state and from the transition table logic circuit can be designed by constructing *K*-maps for Q_1^+ and Q_2^+.

Circuits with Latches

In Section 7.2 latch was introduced using NAND gates. Latch circuits using NAND and NOR gates are shown in Fig. 8.68.

For the circuit of Fig 8.68*a*, the next-state equation is

$$\begin{aligned} Q^+ &= \overline{\overline{S}\cdot\overline{Q}} = \overline{\overline{S}\cdot\overline{(Q\overline{R})}} \\ &= S + \overline{R}Q \end{aligned} \tag{8.9}$$

Similarly, for the circuit of Fig. 8.68*b*, the next-state equation is

$$\begin{aligned} Q^+ &= \overline{R+\overline{Q}} = \overline{R+\overline{(S+Q)}} \\ &= \overline{R}\cdot(S+Q) \\ &= S\overline{R} + \overline{R}Q \end{aligned}$$

Since, $S = R = 1$ is not allowed, which means $SR = 0$, therefore,

$$S\overline{R} = S\overline{R} + SR = S(\overline{R} + R) = S$$

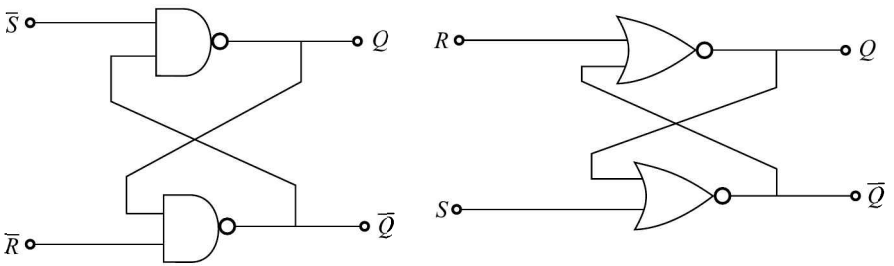


Fig. 8.68 *(a) $\overline{S}$-$\overline{R}$ Latch Using* **NAND** *Gates (b) S-R Latch Using* **NOR** *Gates*

which gives,

$$Q^+ = S + \overline{R}Q$$

It is same as Eq. (8.9)

The transition table of *S-R* latch is shown in Fig. 8.69.

Q \ SR	00	01	11	10
0	⓪	⓪	⓪	1
1	①	0	0	①

$Q^+ = S\overline{R} + \overline{R}Q$
$= S + \overline{R}Q$

Fig. 8.69 ***Transition Table of S-R Latch***

From the transition table of *S-R* FLIP-FLOP, we observe that when *SR* changes from 11 to 00 the circuit will attain either the stable state ⓪ (first row, first column) or ① (second row, first column) depending upon whether *S* goes to 0 first or *R* goes to 0 first respectively. Therefore, $S = R = 1$ must not be applied.

Consider an asynchronous sequential circuit with latches shown in Fig. 8.70.

For FF-1, $R_1 = 0$ and the excitation equation for S_1 is

$$S_1 = X_1 \overline{X}_2 \overline{Q}_2 + \overline{X}_1 Q_2 \tag{8.10}$$

The next-state equation is

$$Q_1^+ = S_1 + \overline{R}_1 Q_1 \tag{8.11}$$

Substituting the value of S_1 from Eq. (8.10) in Eq. (8.11) we obtain,

$$Q_1^+ = X_1 \overline{X}_2 \overline{Q}_2 + \overline{X}_1 Q_2 + Q_1 \tag{8.12}$$

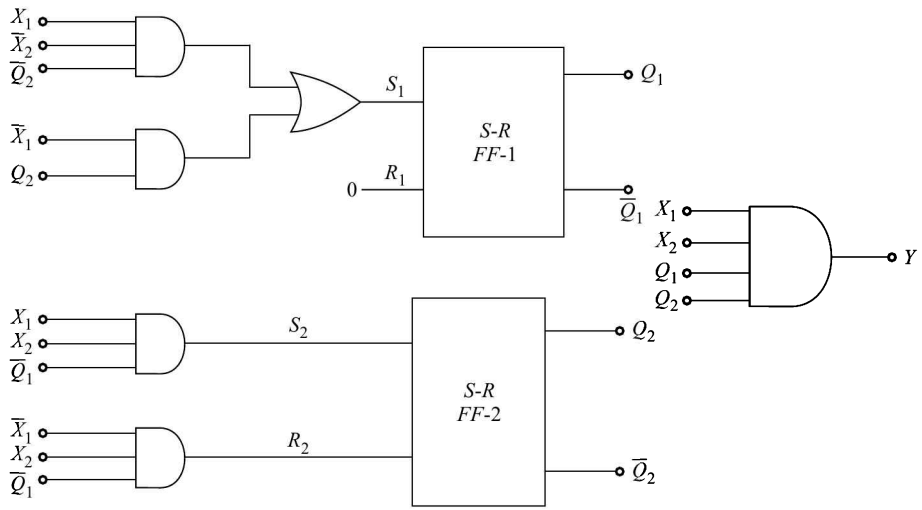


Fig. 8.70 ***Asynchronous Sequential Circuit with Latches***

Similarly, the excitation equations for FF-2 are

$$S_2 = X_1 X_2 \overline{Q}_1, R_2 = \overline{X}_1 X_2 \overline{Q}_1 \tag{8.13}$$

The next-state equation is

$$\begin{aligned} Q_2^+ &= S_2 + \overline{R}_2 Q_2 \\ &= X_1 X_2 \overline{Q}_1 + \overline{\overline{X}_1 X_2 \overline{Q}_1} \cdot Q_2 \end{aligned} \tag{8.14}$$

Using Eqs. (8.12) and (8.14), transition table is obtained as shown in Fig. 8.71.
The output function is

$$Y = X_1 X_2 Q_1 Q_2 \tag{8.15}$$

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
	$Q_1^+ Q_2^+$			
00	(00)	(00)	10	10
01	11	11	(01)	(01)
11	(11)	10	(11)	(11)
10	(10)	(10)	(10)	(10)

Fig. 8.71 ***Transition Table for the Circuit of Fig. 8.70***

Its flow table is shown in Fig. 8.72.

$Q_1\,Q_2$ \ $X_1\,X_2$	00	01	11	10
a	ⓐ, 0	ⓐ, 0	b, 0	d, 0
b	c, 0	c, 0	ⓑ, 0	ⓑ, 0
c	ⓒ, 0	d, 0	ⓒ, 1	ⓒ, 0
d	ⓓ, 0	ⓓ, 0	ⓓ, 0	ⓓ, 0

(Next state $Q_1^+\,Q_2^+$)

Fig. 8.72 ***Flow Table for the Circuit of Fig. 8.70***

From a flow table, transition table can be obtained by assigning binary values to the states. From the transition table, logic equations can be obtained by constructing *K*-maps for *S* and *R* inputs of every latch. For this, the excitation table of *S-R* latch will be used. Logic circuit can then be designed using the logic equation for *S*, *R* inputs of every latch.

Example 8.29

Design logic circuit using *S-R* latches for the transition table of Fig. 8.66.

Solution

Since, there are two internal states Q_1 and Q_2, therefore, two *S-R* latches are required for the design of logic circuit. Let the two latches be L_1 and L_2. The inputs and outputs of these latches are given below.

Latch	*Inputs*	*Outputs*
L_1	S_1, R_1	$Q_1, \overline{Q_1}$
L_2	S_2, R_2	$Q_2, \overline{Q_2}$

The excitation table of an *S-R* latch is given in Table 8.29. This is same as Table 7.1 for *S-R* `FLIP-FLOP`.

Table 8.29 ***Excitation Table of S-R Latch***

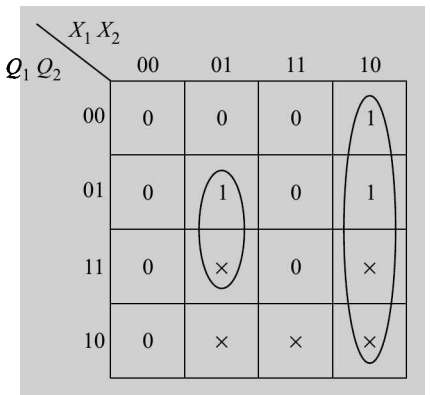
Present state	**Next state**	**Inputs**	
Q	Q^+	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

To determine S_1 and R_1 for different values of $X_1\ X_2$, we make use of Q_1 and Q_1^+ values for every square of transition table. For example, the square in the first row and first column gives $Q_1 = 0$ and $Q_1^+ = 0$. This means, for the present state 0 the circuit gives next state as 0 for Q_1. Corresponding to this we find the value of S_1 and R_1 using the Table 8.29, which are

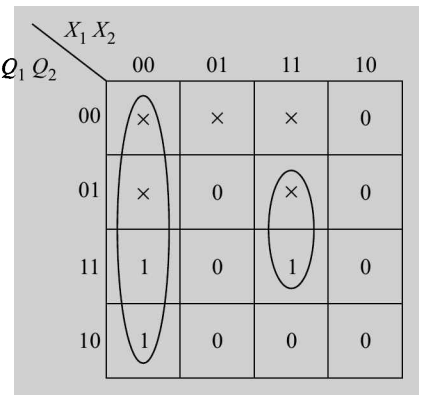
$$S_1 = 0 \quad \text{and} \quad R_1 = X$$

Thus the entry in the cell corresponding to $X_1\ X_2 = 00$ and $Q_1\ Q_2 = 00$ for K-map of S_1 will be 0 and for K-map of R_1 it will be X. Similarly, K-map entries are determined for S_1 and R_1.

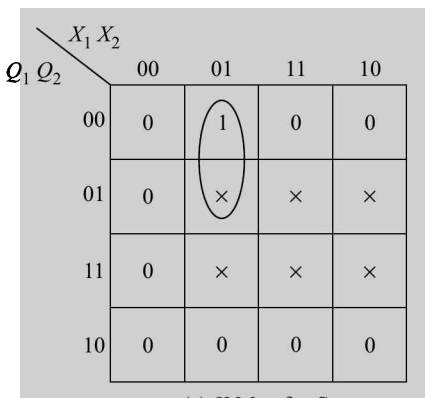
Following similar procedure, K-maps for S_2 and R_2 are constructed. The K-maps are given in Fig. 8.73.



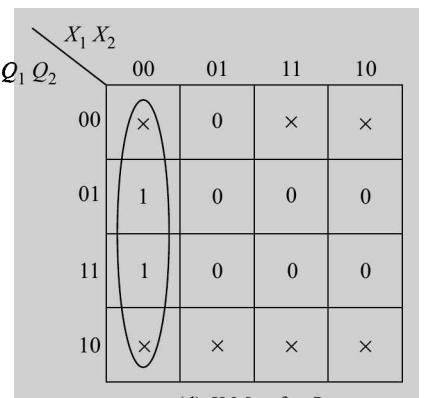
(a) K-Map for S_1



(b) K-Map for R_1



(c) K-Map for S_2



(d) K-Map for R_2

Fig. 8.73

From the K-map of Fig. 8.73, we obtain logic equations for S_1, R_1 S_2, and R_2.

$$S_1 = X_1\,\overline{X}_2 + \overline{X}_1 X_2 Q_2$$
$$R_1 = \overline{X}_1\,\overline{X}_2 + X_1 X_2 Q_2$$
$$S_2 = \overline{X}_1 X_2\,\overline{Q}_1$$
$$R_2 = \overline{X}_1\,\overline{X}_2$$

The logic circuit is shown in Fig. 8.74.

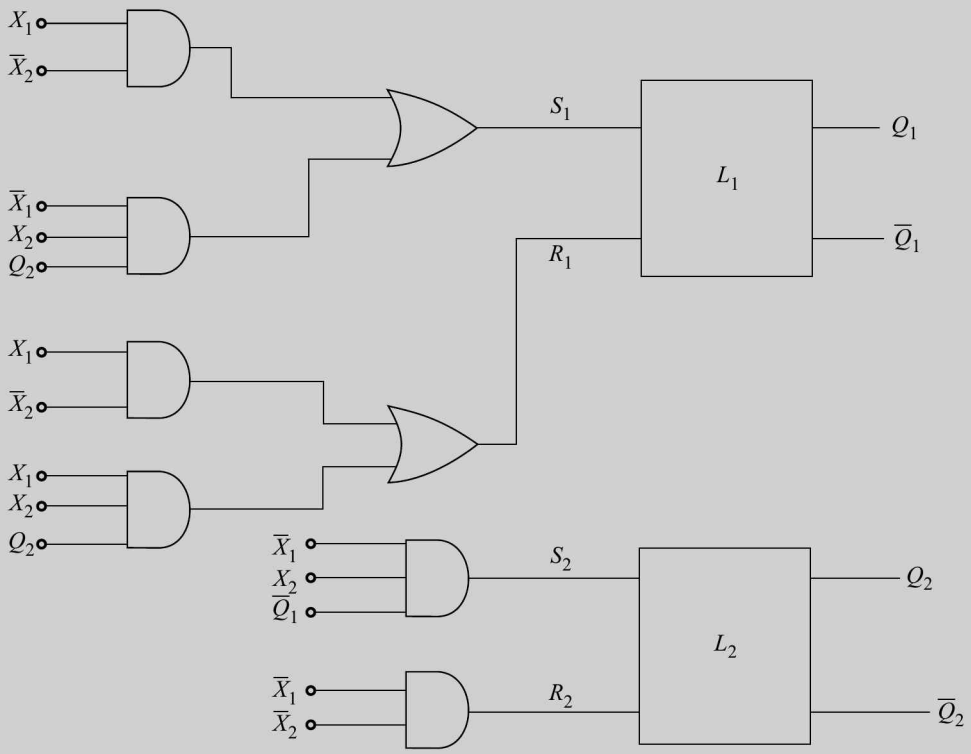


Fig. 8.74 ***Logic Circuit for Ex. 8.29***

Races and Cycles

A *race* condition exists in an asynchronous sequential circuit when more than one state variable change value in response to a change in an input variable. This is caused because of unequal propagation delays in the path of different secondary variables in any practical electronic circuit. Consider a transition table shown in Fig. 8.75. When both the inputs X_1 and X_2 are 0 and the present state is $Q_1\,Q_2 = 00$, the resulting next-state $Q_1^+\,Q_2^+$ will have $Q_1^+ = 1$ and $Q_2^+ = 1$ simultaneously if the propagation delays in the paths of Q_1 and Q_2 are equal.

Since Q_1 and Q_2 both are to change and in general the propagation delays in the paths of Q_1 and Q_2 are not same, therefore, either Q_1 or Q_2 may change first instead of both changing simultaneously. As a consequence of this the circuit will go to either state 01 or to state 10.

$Q_1\,Q_2$ \ $X_1\,X_2$	00	01	11	10
	$Q_1^+\,Q_2^+$			
00	11	(00)	10	01
01	11	00	11	(01)
11	(11)	00	10	(11)
10	11	(10)	(10)	11

Fig. 8.75

If Q_2^+ changes faster than Q_1^+, the next state will be 01, then 11 (first column, second row) and then the stable state (11) (first column, third row) will be reached. On the other hand, if Q_1^+ changes faster than Q_2^+, the next-state will be 10, then 11 (first column, fourth row) and then to the stable state (11) (first column, third row) will be reached. In both the situations, the circuit goes to the same final stable state (11). This situation, where a change of more than one secondary variable is required is known as a *race*. There are two types of races: *noncritical race* and *critical race*. In the case of noncritical race, the final stable state in which the circuit goes does not depend on the sequence in which the variables change. The race discussed above is a noncritical race. In the case of critical race, the final stable state reached by the circuit depends on the sequence in which the secondary variables change. Since the critical race results in different stable states depending on the sequence in which the secondary states change, therefore, it must be avoided.

Example 8.30

In the transition table of Fig. 8.75, consider the circuit in stable total state 1100. Will there be any race, if the input state changes to 01? If yes, find the type of race.

Solution

When the circuit is in stable total state, $X_1 X_2 = 00$. Now X_2 changes to 1 while $X_1 = 0$. From Fig. 8.75 we see that the required transition is to state 00. If Q_1^+ and Q_2^+ become 00 simultaneously, then the transition will be

(11) → 00 → (00)

These transitions are shown by solid arrows in Fig. 8.76. If Q_2^+ becomes 0 faster than Q_1^+, the circuit will go to the state 10 and then to (10), which is a stable state. The transition is

(11) → 10 → (10)

On the other hand, if Q_1^+ becomes 0 faster than Q_2^+, the transition will be

(11) → 01 → 00 → (00)

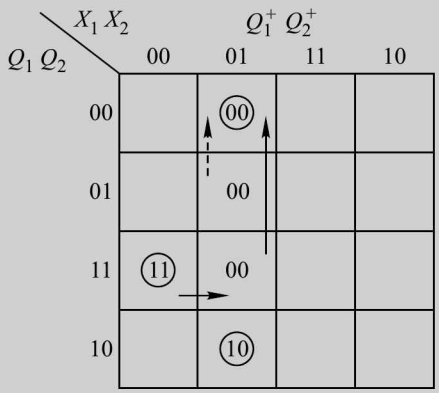


Fig. 8.76

It is shown by dotted arrow in Fig. 8.76. Thus, we see that the circuit attains different stable states (00) or (10) depending up on the sequence in which the secondary variables change. Therefore, the race condition exists in this circuit and it is critical race.

Races can be avoided by making a proper binary assignment to the state variables in a flow table. The state variables must be assigned binary numbers in such a way so that only one state variable can change at any one time when a state transition occurs in the flow table. The state transition is directed through a unique sequence of unstable state variable change. This is referred to as a *cycle*. This unique sequence must terminate in a stable state, otherwise the circuit will go from one unstable state to another unstable state making the entire circuit unstable.

Example 8.31

In the state transition table of Fig. 8.75, if $X_1 X_2 = 10$ and the circuit is in stable state ⓪①, find the cycle when X_2 is changed to 1 while X_1 remaining 1.

Solution

The circuit is in stable state ⓪① (fourth column, second row). When X_2 changes to 1, the circuit will go to the state 11 (third column, second row), then to state 10 (third column, third row) and finally to the stable state ①⓪ (third column, fourth row). Thus the cycle is

⓪① → 11 → 10 → ①⓪

Pulse-Mode Circuits

In a pulse mode asynchronous sequential circuit, an input pulse is permitted to occur only when the circuit is in stable state and there is no pulse present on any other input. When an input pulse arrives, it triggers the circuit and causes a transition from one stable state to another stable state so as to enable the circuit to receive another input pulse. In this mode of operation critical race can not occur. To keep the circuit stable between two pulses, FLIP-FLOPs whose outputs are levels must be used as memory elements.

For the analysis of pulse-mode circuits, the model used for the fundamental-mode circuits is not valid since the circuit is stable when there are no inputs and the absence of a pulse conveys no information. For this a model similar to the one used for synchronous sequential circuits will be convenient to use.

In pulse-mode asynchronous circuits the number of columns in the next-state table is equal to the number of input terminals.

Consider a pulse-mode circuit logic diagram shown in Fig. 8.77. In this circuit there are four input variables X_1, X_2, X_3, and X_4, and Y is the output variable. It has two states Q_1 and Q_2.

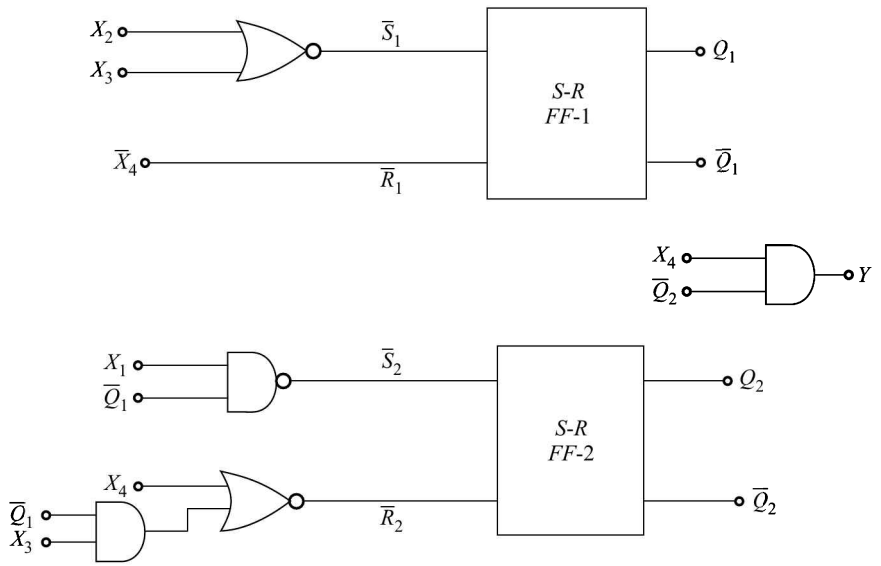


Fig. 8.77 ***A Pulse-Mode Asynchronous Circuit***

The excitation and output equations are:

$$\overline{S}_1 = \overline{(X_2 + X_3)} \quad \text{or} \quad S_1 = X_2 + X_3 \tag{8.16a}$$

$$\overline{R}_1 = \overline{X}_4 \quad \text{or} \quad R_1 = X_4 \tag{8.16b}$$

$$\overline{S}_2 = \overline{\overline{Q}_1 X_1} \quad \text{or} \quad S_2 = \overline{Q}_1 X_1 \tag{8.16c}$$

$$\overline{R}_2 = \overline{(X_4 + \overline{Q}_1 X_3)} \quad \text{or} \quad R_2 = X_4 + \overline{Q}_1 X_3 \tag{8.16d}$$

$$Y = X_4 \overline{Q}_2 \tag{8.17}$$

The next-state equations are obtained by using the excitation equations (Eqs. (8.16)) and the characteristic equation of latch (Eq. (8.9)). These are:

$$\begin{aligned} Q_1^+ &= S_1 + \overline{R}_1 Q_1 \\ &= X_2 + X_3 + \overline{X}_4 Q_1 \end{aligned} \tag{8.18}$$

and

$$\begin{aligned} Q_2^+ &= S_2 + \overline{R}_2 Q_2 \\ &= \overline{Q}_1 X_1 + \overline{(X_4 + \overline{Q}_1 X_3)} \cdot Q_2 \\ &= \overline{Q}_1 X_1 + \overline{X}_4 \cdot \overline{(\overline{Q}_1 X_3)} \cdot Q_2 \\ &= \overline{Q}_1 X_1 + \overline{X}_4 \cdot (Q_1 + \overline{X}_3) \cdot Q_2 \\ &= \overline{Q}_1 X_1 + Q_1 Q_2 \overline{X}_4 + Q_2 \overline{X}_3 \overline{X}_4 \end{aligned} \tag{8.19}$$

The transition table is constructed by evaluating the next-state and output for each present state and input value using Eqs. (8.17), (8.18), and (8.19). The transition table is shown in Fig. 8.78. It has four rows (one row for each combination of state variables) and four columns (one column for each input variable). Since in pulse-mode circuits only one input variable is permitted to be present at a time, therefore, the columns are for each input variable only and not for the combinations of input variables.

Flow table can be constructed from the transition table and is shown in Fig. 8.79. Here, S_0, S_1, S_2, and S_3 are the four state variables.

From a flow table a transition table can be constructed by assigning binary values to the states. From a transition table next-state equations can be obtained and the logic diagram can then be obtained.

Present state $Q_1\,Q_2$	Input variables X_1	X_2	X_3	X_4
00	01, 0	10, 0	10, 0	00, 1
01	01, 0	11, 0	10, 0	00, 0
11	11, 0	11, 0	11, 0	00, 0
10	10, 0	10, 0	10, 0	10, 0

Next-state value

Output value

Fig. 8.78 ***Transition Table of Fig. 8.77***

Present state $Q_1\,Q_2$	Input variables X_1	X_2	X_3	X_4
S_0	S_1, 0	S_3, 0	S_3, 0	S_0, 1
S_1	S_1, 0	S_2, 0	S_3, 0	S_0, 0
S_2	S_2, 0	S_2, 0	S_2, 0	S_0, 0
S_3	S_3, 0	S_3, 0	S_3, 0	S_0, 0

Fig. 8.79 ***Flow Table of Pulse-Mode Circuit***

8.7.5 Asynchronous Sequential Circuit Design

Design of asynchronous sequential circuits is more difficult than that of synchronous sequential circuits because of the timing problems involved in these circuits. Designing an asynchronous sequential circuit requires obtaining logic diagram for the given design specifications. Usually the design problem is specified in the form of statements of the desired circuit performance precisely specifying the circuit operation for every applicable input sequence.

Design Steps

1. Primitive flow table is obtained from the design specifications. When setting up a primitive flow table it is not necessary to be concerned about adding states which may ultimately turn out to be redundant. A sufficient number of states are to be included to completely specify the circuit performance for every allowable input sequence. Outputs are specified only for stable states.
2. Reduce the primitive flow table by eliminating the redundant states, which are likely to be present. These redundant states are eliminated by merging the states. *Merger diagram* is used for this purpose.
3. Binary numbers are assigned to the states in the reduced flow table. The binary state assignment must be made to ensure that the circuit will be free of critical races. The output values are to be chosen for the unstable states with unspecified output entries. These must be chosen in such a way so that momentary false outputs do not occur when the circuit switches from one stable state to another stable state.
4. Transition table is obtained next.
5. From the transition table logic diagram is designed by using the combinational design methods. The logic circuit may be a combinational circuit with feedback or a circuit with *S-R* latches.

The above design steps will be illustrated through an example.

Example 8.32

The output (Y) of an asynchronous sequential circuit must remain 0 as long as one of its two inputs X_1 is 0. While $X_1 = 1$, the occurrence of first change in another input X_2, should give $Y = 1$ as long as $X_1 = 1$ and becomes 0 where X_1 returns to 0. Construct a primitive flow table.

Solution

This circuit has two inputs X_1, X_2 and one output Y. For the construction of flow table, the next-state and output are required to be obtained. The flow table is shown in Fig. 8.80.

For $X_1X_2 = 00$, let us take state a. When the circuit has $X_1X_2 = 00$ the output is 0 (since $X_1 = 0$) and the circuit is in stable state @. The next-state and output are shown in the first column, first row of Fig. 8.80. Since only one input is allowed to change at a time, therefore, the next input may be $X_1X_2 = 01$ or 10.

If $X_1X_2 = 01$, let us take another state b, correspondingly the second row of the flow table corresponds to state b. when the inputs change from $X_1X_2 = 00$ to 01, the circuit is required to go to stable state ⓑ and output is 0 (since $X_1 = 0$). Therefore, the entry in the second column, first row will be b, 0 and in the second column, second row will be ⓑ, 0. The output corresponding to unstable state b is taken as 0 so that no momentary false outputs occur when the circuit switches between stable states. On the other hand if X_1X_2

Present-state	X_1X_2 = 00	01	11	10
a	ⓐ, 0	b, 0	–, –	c, 0
b	a, 0	ⓑ, 0	d, 0	–, –
c	a, 0	–, –	e, –	ⓒ, 0
d	–, –	b, 0	ⓓ, 0	t, –
e	–, –	b, –	ⓔ, 0	t, 1
f	a, –	–, –	e, 1	ⓕ, 1

Fig. 8.80 ***Flow Table of Ex. 8.32***

= 10, the circuit is required to go to another stable state ⓒ with output 0. Therefore, the entries in the fourth column, first row and fourth column, third row will be respectively *c*, 0 and ⓒ, 0.

Since both the inputs cannot change simultaneously, therefore, from stable state ⓐ, the circuit cannot go to any specific state corresponding to $X_1X_2 = 11$ and accordingly the entry in the third column, first row will be –, –. The dashes represent the unspecified state, output.

Now consider the stable state ⓑ. The inputs X_1X_2 can chage to 00 or 11.

If $X_1X_2 = 00$, the circuit will go to state *a*. Therefore, the entry in the first column, second row will be *a*, 0. From this unstable state the circuit goes to stable state ⓐ. On the other hand if $X_1X_2 = 11$, then the circuit goes to *a* new state *d*. The output corresponding to $X_1X_2 = 11$ will be 0 since, there is no change in X_2, which is already 1. Therefore, the entry in the third column, second row will be *d*, 0. The fourth row corresponds to state *d*, and the entry in the third column, fourth row, will be ⓓ, 0. From ⓑ, the circuit is not required to go to any specific state and therefore, the entry in the fourth column, second row will be –, –.

Similarly, now consider stable state ⓒ. The inputs can change to $X_1X_2 = 11$ or 00. If $X_1X_2 = 11$, the circuit goes to a new stable state ⓔ and the output will be 1, since X_2 changes from 0 to 1 while $X_1 = 1$. The entry in the third column, third row will be *e*, –. Output has to change from 0 to 1 from stable state ⓒ to stable state ⓔ, which may or may not change to 1 for unstable *e*. The entry in the third column, fifth row will be ⓔ, 1. The entry in the second column third row will be –, – and the entry in the first column, third row will be *a*, 0 (for $X_1X_2 = 00$).

In the same manner, we consider the stable ⓓ and obtain the entries *f*, – (fourth column, fourth row); ⓕ, 1 (fourth column, sixth row); *b*, 0 (second column, fourth row) and –, – (first column, fourth row).

Similar procedure applied to ⓔ and ⓕ, yields the remaining entries of the flow table (Problem. 8.36).

Since, every row in the flow table of Fig. 8.80 contains only one stable state, therefore, this flow table is a primitive flow table.

Reduction of States

The necessity of reducing the number of states has been discussed in Section 8.6 and the equivalent states have been defined. When asynchronous sequential circuits are designed, the design process starts from the construction of primitive flow table. A primitive flow table is never completely specified. Some states and outputs are not specified in it as shown in Fig. 8.80 by dashes. Therefore, the concept of equivalent states can not be used for reduction of states. However, incompletely specified states can be combined to reduce the number of states in the flow table. Two incompletely specified states can be combined if they are *compatible.*

Two states are *compatible* if and only if, for every possible input sequence both produce the same output sequence whenever both outputs are specified and their next states are compatible whenever they are specified. The unspecified outputs and states shown as dashes in the flow table have no effect for compatible states.

Example 8.33

In the primitive flow table of Fig. 8.80, find whether the states *a* and *b* are compatible or not. If compatible, find out the merged state.

Solution

The rows corresponding to the states *a* and *b* are shown in Fig. 8.81. Each column of these two states is examined.

Column–1 Both the rows have the same state *a* and the same output 0. *a* in first row is stable state and in the second row is unstable state.

Since for the same input both the states *a* and *b* have the same specified next-state *a* and the same specified output 0. Therefore, this input condition satisfies the requirements of compatibility.

Column–2 The input condition $X_1\ X_2 = 01$ satisfies the requirements of compatibility as discussed for column–1.

Column–3 The first row has unspecified next-state and output and the second row has specified state and output. The unspecified state and output may be assigned any desired state and output and therefore, for this input condition also the requirements of compatibility are satisfied.

Column–4 The requirements of compatibility are satisfied for the reasons same as applicable to column–3.

Therefore, we conclude that since the next-states and the outputs for all the input combinations are compatible for the two states *a* and *b*, the two states are compatible.

The merged state will be as shown in Fig. 8.82. When the merged state entries are determined a circled entry and an uncircled entry results in a circled entry, since the corresponding state must be stable as shown in Fig. 8.82.

Present state	X_1X_2 = 00	01	11	10
a	ⓐ, 0	*b*, 0	–, –	*c*, 0
b	*a*, 0	ⓑ, 0	*d*, 0	–, –

Fig. 8.81

	X_1X_2 = 00	01	11	10
a	ⓐ, 0	ⓑ, 0	*d*, 0	*c*, 0

Fig. 8.82

Example 8.34

In the primitive flow table of Fig. 8.80 find whether the states *a* and *e* are compatible or not. Examine their compatibility if the entries in the fourth column for the states *a* and *e* have same output.

Solution

The partial flowtable for states *a* and *e* of Fig. 8.80 is shown in Fig. 8.83.

From this we observe the following

Column-1	compatible
Column-2	compatible
Column-3	compatible
Column-4	not compatible, since the outputs are different.

Present state	X_1X_2 = 00	01	11	10
a	ⓐ, 0	*b*, 0	–, –	*c*, 0
e	–, –	*b*, –	ⓔ, 0	*t*, 1

Fig. 8.83

Therefore, the states *a* and *e* are not compatible.

In case of same output in column-4, the outputs are said to be *not conflicting* and the states *a* and *e* are compatible if and only if the states *c* and *f* are compatible. This is referred to as *c*, *f* is *implied* by *a*, *b* or *a*, *b implies c*, *f*.

Merger Diagram

A *merger diagram (or graph)* is prepared for a primitive flow table to determine all the possible compatible states (maximal compatible states) and from this a minimal collection of compatibles covering all the states.

A merger graph is constructed following the steps outlined below:

- Each state is represented by a vertex, which means it consists of *n* vertices, each of which corresponds to a state of the circuit for an *n*- state primitive flow table. Each vertex is labelled with the state name.
- For each pair of compatible states an undirected arc is drawn between the vertices of the two states. No arc is drawn for incompatible states.
- For compatible states implied by other states a broken arc is drawn between the states and the implied pairs are entered in the broken space.

The flow table is required to be examined for all the possible pairs of states. All the pairs are checked and the merger graph is obtained. Thus we see that the merger graph displays all possible pairs of compatible states and their implied pairs. Next it is necessary to check whether the incompatible pair(s) does not invalidate any other implied pair. If any implied pair is invalidated it is neglected. All the remaining valid compatible pairs form a group of *maximal* compatibles.

The maximal compatible set can be used to construct the reduced flow table by assigning one row to each member of the group. However, the maximal compatibles do not necessarily constitute the set of *minimal* compatibles. The set of minimal compatibles is a smaller collection of compatibles that will satisfy the row merging.

The conditions that must be satisfied for row merging are:

- the set of chosen compatibles must *cover* all the states, and
- the set of chosen compatibles must be *closed.*

The condition of covering requires inclusion of all the states of the primitive flow graph in the set of chosen compatibles. This condition only defines a *lower bound* on the number of states in the minimal set. However, if none of their implied pairs are contained in the set, the set is not sufficient and this is referred to as *closed* condition not being satisfied. Therefore, condition of *closed covering* is essentially required for row merging.

Example 8.35

Construct merger diagram for the primitive flow table of Fig. 8.80. Determine maximal compatibles and the minimal set of compatibles.

Solution

For construction of merger diagram, every row of the primitive flow table is checked with every other row to determine compatibility of states.

Consider row-1 (state *a*)

a, *b* are compatible
a, *c* are compatible
a, *d* are compatible if *c*, *f* are compatible
a, *e* are compatible if *c*, *f* are compatible
a, *f* are compatible if *c*, *f* are compatible

row-2 (state *b*)

b, *c* are compatible if *d*, *e* are compatible
b, *d* are compatible
b, *e* are not compatible (outputs are conflicting)
b, *f* are not compatible (outputs are conflicting)

row-3 (state *c*)

c, *d* are compatible if *e*,*d* and *c*, *f* are compatible
c, *e* are not compatible (outputs are conflicting)
c, *f* are not compatible (outputs are conflicting)

row-4 (state *d*)

d, *e* are not compatible (outputs are conflicting)
d, *f* are not compatible (outputs are conflicting)

row-5 (state *e*)

e, *f* are compatible

The primitive flow table has six states therefore, there are six vertices in the merger diagram as shown in Fig 8.84. Solid arcs are drawn between (*a*, *b*), (*a*, *c*), (*b*, *d*), and (*f*, *e*) vertices. Corresponding to these states being compatibles. Since (*c*, *f*) and (*d*, *e*) are not compatible, therefore, there are no implied pairs available.

From the merger diagram, we get the maximal compatibles:

$$(a, b), (a, c), (b, d), (e, f)$$

Since (*a*, *b*) is covered by (*a*, *c*) and (*b*, *d*), therefore, the minimal set is

$$(a, c), (b, d), (e, f)$$

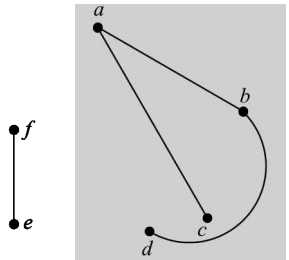


Fig. 8.84 ***Merger Diagram***

Example 8.36

Determine the reduced flow table of Fig. 8.81.

Solution

From the merger diagram, we have obtained three pairs of compatible states: These compatibles are merged and are represented by

$$a, c : S_0$$
$$b, d : S_1$$
$$e, f : S_2$$

The reduced flow table is shown in Fig. 8.85.

Present state	X_1X_2 = 00	01	11	10
S_0	(S0), 0	S_1, 0	S_2, –	(S0), 0
S_1	S_0, 0	(S1), 0	(S1), 0	S_2, –
S_2	S_0, –	S_1, –	(S2), 1	(S2), 1

Fig. 8.85 ***Reduced Flow Table***

Example 8.37

Assign binary states to the reduced flow table of Fig. 8.85. Avoid critical race.

Solution

Let us assign the following binary states to S_0, S_1, and S_2 for the reduced flow table of Fig. 8.85

$$S_0 \rightarrow 00$$
$$S_1 \rightarrow 01$$
$$S_2 \rightarrow 11$$

The transition table will be as shown in Fig. 8.86

In the transition table of Fig. 8.86, we observe that race condition occurs in the following cases:

(i) From stable state (00) to unstable state 11 when $X_1 X_2$ changes from 10 to 11.

(ii) From stable state (11) to unstable state 00 when $X_1 X_2$ changes from 10 to 00.

To avoid critical race, one unstable state 10 is added with the entries

$$00, –; –, –; 11, –; –, –$$

and the entries in third column, first row is changed from 11, – to 10, – and in first column, third row from 00, – to 10, –.

The modified transition table is given in Fig. 8.87.

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
00	(00), 0	01, 0	11, –	(00), 0
01	00, 0	(01), 0	(01), 0	11, –
11	00, –	01, –	(11), 1	(11), 1
10				

Fig. 8.86 ***Transition Table***

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
00	(00), 0	01, 0	10, –	(00), 0
01	00, 0	(01), 0	(01), 0	11, –
11	10, –	01, –	(11), 1	(11), 1
10	00, 1	–, –	11, 1	–, –

Fig. 8.87 ***Modified Transition Table***

Example 8.38

Design logic circuit with feedback for the transition table of Fig. 8.87.

Solution

The *K*-maps for Q_1^+, Q_2^+, and Y determined from the transition table are given in Fig. 8.88.

From the *K*-maps, we obtain,

$$Q_1^+ = X_1 X_2 \overline{Q}_2 + \overline{X}_2 Q_1 Q_2 + X_1 \overline{X}_2 Q_2 + X_1 Q_1$$

$$Q_2^+ = \overline{X}_1 X_2 + X_1 Q_2 + X_1 Q_1$$

$$Y = Q_1$$

Logic circuit using gates can be obtained from the above logic equations.

Thus we see that the design steps outlined above can be used to design an asynchronous sequential circuit.

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	1	0	1	1
10	0	×	1	×

Q_1^+

(a)

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	0	1	1	1
10	0	×	1	×

Q_2^+

(b)

$Q_1 Q_2$ \ $X_1 X_2$	00	01	11	10
00	0	0	×	0
01	0	0	0	×
11	×	×	1	1
10	×	×	×	×

Y

(c)

Fig. 8.88 ***K-Maps for (a) Q_1^+ (b) Q_2^+ (c) Y***

8.8 HAZARDS IN SEQUENTIAL CIRCUITS

Hazards in combinational circuits have been discussed in Section 5.12. The synchronous sequential circuits operate in synchronism with the clock pulses and the inputs must be stable before the triggering edge arrives. Therefore, even if hazard condition in the combinational part of the synchronous circuit creates erroneous signal i.e., glitch, the operation of the circuit is not affected. However, asynchronous sequential circuits may malfunction because of the occurrence of hazard condition in its combinational part and/or unequal delays along two or more feedback paths that originate from the same input variable change.

In case the hazard condition is caused due to the hazard in the combinational part of an asynchronous sequential circuit, the hazard is either static-1, static-0, or dynamic type of hazard. The hazard caused due to the effects of a single input variable change reaching one feedback path before another feedback path is known as *essential-hazard.*

8.8.1 Hazards in Asynchronous Sequential Circuits Implemented Using Latches

The momentary false signals in the combinational logic part of an asynchronous sequential circuit can cause improper operation. Therefore, the combinational logic portion of an asynchronous sequential circuit should be hazard-free.

It has been demonstrated in section 5.12.2 that the following types of hazards may occur in 2-level realizations.

- Static-1 hazard in SOP realization i.e. AND–OR or NAND–NAND realization. This type of hazard causes the output of a combinational circuit to go momentarily to 0.
- Static-0 hazard in POS realization i.e. OR–AND or NOR–NOR realization. This type of hazard causes the output of a combinational circuit to go momentarily to 1.

  Now let us examine the effect of static-1 and static-0 hazards on the operation of the *S-R* latches shown in Figs. 8.68*a* and *b*.
- In the case of NOR latch of Fig. 8.68*b*, a momentary 1 signal on either *S* or *R* inputs can cause it to set or reset, respectively. However, a momentary 0 signal on *S* or *R* inputs has no effect since, when $S = 0$ and $R = 0$, the state of the circuit does not change. Therefore, the combinational logic circuit that precedes the input terminals of an *S-R* latch is not required to be free of static-1 hazards, but it must be free of static-0 hazards.
- In the case of NAND latch of Fig. 8.68*a*, a momentary 0 signal on either $\overline{S}$ or $\overline{R}$ inputs can cause it to set or reset, respectively. However, a momentary 1 signal on $\overline{S}$ or $\overline{R}$ inputs has no effect since, when $\overline{S} = 1$ and $\overline{R} = 1$, the state of the circuit does not change. Therefore, in this case, the combinational logic circuit that precedes the input terminals of an $\overline{S}$-$\overline{R}$ latch is not required to be free of static-0 hazards, but it must be free of static-1 hazards.

The hazard-free asynchronous circuits are shown in Fig. 8.89.

8.8.2 Essential-Hazards

In some asynchronous sequential circuits, there is an inherent possibility of making an incorrect transition due to differences in delay through various paths because of change in a single input variable. Such type of hazard is known as essential hazard. It is not possible to modify the circuit to make it free of essential hazard.

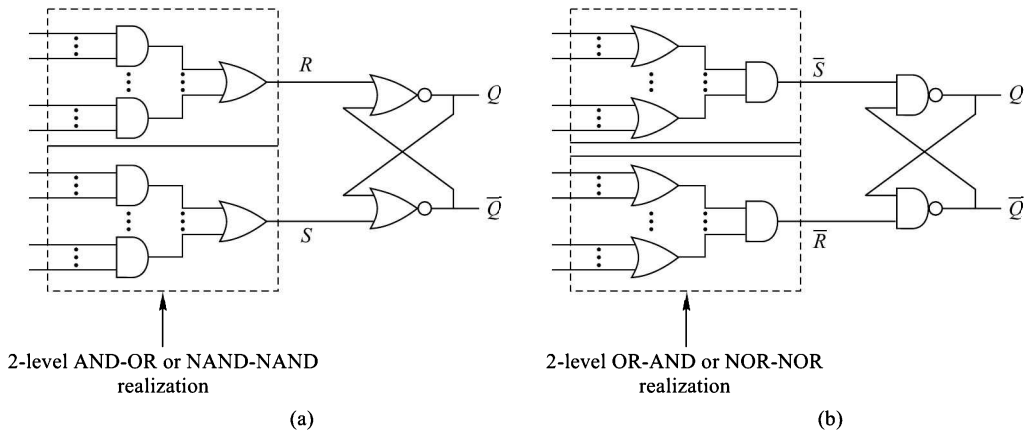


Fig. 8.89 ***Hazard Free Asynchronous Circuits Using Latches***

The essential hazard can only be eliminated by introducing additional delay in the feedback path so that incorrect transition does not take place.

Essential hazard in an asynchronous sequential circuit can be detected by observing its flow table. Consider a fundamental mode asynchronous sequential circuit in a total stable state (S_1). A change in an input variable x causes transition to another stable state (S_2) in an adjacent column, assuming equal delay through all the feedback paths. If the circuit goes to a different stable state (S_3) after three consecutive changes in x due to different delays in the feedback paths, then we say that essential hazard exists in the flow table.

Consider the flow table shown in Fig. 8.90. Let the circuit be in a stable state (A), changing x input from 0 to 1 will cause transition to another stable state (B). The state transition is indicated by dotted arrows. This operation corresponds to the situation when there is no problem of occurrence of essential hazard due to unequal delay in different feedback paths. Now let us assume that because of unequal delays in the different feedback paths, the circuit goes to state C (second row, first column), then to state (C) (third row, first column) and finally becomes stable in (C) (third row, second column). This illustrates the occurrence of essential hazard. The operation of the circuit is indicated by solid arrows.

Present state	Next State, Input x = 0	Next State, Input x = 1
A	(A)	(B)
B	C	(B)
C	(C)	(C)
D	–	–

Fig. 8.90 ***Flow Table***

Example 8.39

Determine whether essential hazard exists in the flow tables of Fig. 8.91. Assume initial stable state (A).

Solution

In Fig. 8.91*a* when x changes from 0 to 1, the circuit may go to B and then to (B) or it may go to A (second row, first column) and eventually to (B). This shows that the circuit is free of essential hazard.

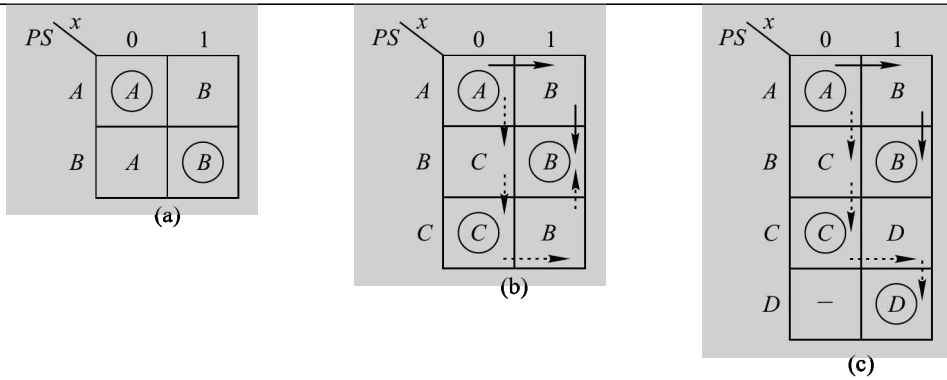


Fig. 8.91 ***Flow Tables of Ex. 8.39***

In Fig. 8.91*b*. The circuit goes to stable state (B) when x changes from 0 to 1. It may follow either the path shown by solid arrows or the path shown by the dotted arrows.

In Fig. 8.91*c*. The circuit goes to stable state (B) (see the solid arrows) or to the stable state D following the path shown by the dotted arrows. Therefore, unequal delays in the different feedback paths will cause transition to stable state (D) rather than (B) which shows the existence of essential-hazard.

Essential hazard can be eliminated by introducing additional delay in the feedback path with lesser delay.

SUMMARY

A very important and useful class of digital circuits, sequential circuits, have been discussed in detail in this chapter. Clocked sequential circuits, such as, counters, shift registers, useful in many applications of digital systems have been discussed. Their design using FLIP-FLOPs and standard MSI devices have been included. Design of general clocked sequential circuits using the two models, Mealy and Moore, will be helpful in designing any clocked sequential circuit.

Standard counter ICs are available for counting in normal binary sequence and natural BCD for different modulus. In many applications counters can be designed using these available ICs.

In case any other sequence of count or modulus is required, then the circuit can be designed using FLIP-FLOPs and gates using the methods discussed here.

The analysis and design of asynchronous sequential circuits have been discussed in detail. Although the design of asynchronous sequential circuits is much more difficult than the design of clocked (synchronous) sequential circuits, nonetheless they find widespread applications requiring high speeds, low cost systems in which input signals may change at any time.

There are various problems associated with the asynchronous sequential circuits such as races and cycles, hazards, and essential-hazards. All these cause malfunctioning of the circuit. The causes of existence these problems and the methods used to eliminate them have been discussed.

GLOSSARY

Asynchronous counter A digital counter in which all the FLIP-FLOPs are not triggered simultaneously.

BCD counter A modulo-10 counter with count sequence from 0000 to 1001, i.e 0 to 9 decimal digitis.

Bidirectional register A shift register in which data can be shifted in both the directions, i.e from left to right and right to left.

Binary counter An N-stage counter that recycles after 2^N counts. The count proceeds in specificed binary sequence.

Counter, Pressetable A counter which can be set to a desired value before the start of the counting.

Counter, Ripple See Ripple counter.

Counter, UP-DOWN A counter that can count in both up and down direction, depending upon a control input. This means the count can be in ascending or descending order.

Count-sequence The repeating sequence of binary numbers at the output of a counter.

Critical race In an asynchronous sequential circuit when two or more internal state variables have to change simultaneously due to change in input, then due to different delay paths, it is very difficult to predict the state variable which will change first. Due to this race condition is said to exist and the circuit may go to different next-states depending upon the delays involved in different paths.

Cycle A cycle occurs is an asynchronous sequential circuit when multiple state transitions occur without changing the inputs, taking the circuit from one unstable state to another unstable state and finally reaching a stable state.

Decade counter A counter that recycles after every 10 pulses. Same as a BCD counter.

Down-counter A counter in which the count decreases with every clock pulse.

Essential hazard Inherent hazard occuring in an asynchronous sequential circuit due to different delay in different feedback paths, causing incorrect transition for a change in single input variable.

First-in-first-out (`FIFO`) A register organisation in which the data which is stored first is the first data available for taking out. (See bidirectional shift register)

Flow table A tabular method for showing state transitions in asynchronous sequential circuits.

Fundamental mode An asynchronous sequential circuit with level inputs.

Frequency divider A sequential logic circuit that can divide the frequency of a pulse source by an integer.

Internal state A state in asynchronous sequential circuit. Same as secondary state.

Left shift register A shift register in which data gets shifted in the left direction in response to clock pulses.

Last-in-first-out **(LIFO)** A register organisation in which the data which is stored last is the first data available for taking out. (See bidirectional shift register).

Loading of a counter Presetting (or loading) the counter with the desired initial count.

Lock out A condition which may exist in a counter taking the counter from one unused state to another unused state and not allowing it to come to any of the used states.

Merger diagram A graph used for the possible merging of states in an asynchronous sequential circuit.

Modulus **(MOD)** The number of clock pulses after which a counter reaches its initial state.

Non-critical race A race condition in which an asynchronous sequential circuit will reach the same stable state.

Parallel-data All the data bits are available simultaneously on different data lines.

Parallel-loading Simultaneous loading of all the bits in a register or counter.

Parallel-to-serial converter A logic circuit that converts data from parallel to serial form.

Primitive flow table A flow table in which every row has only one stable state.

Pulse mode An asynchronous sequential circuit with pulse inputs.

Race A condition occuring in an asynchronous sequential circuit due to different path delays when transition takes place requiring more than one state to change simultaneously.

Right-shift register A shift register in which the data gets shifted in the right direction in response to clock pulses.

Ring counter A shift register whose output is fedback to the input, i.e. the data circulates around the FLIP-FLOPs in response to clock pulses.

Ripple counter A counter in which all the FLIP-FLOPs are not clocked simultaneously. A particular FLIP-FLOP will get triggered only if the state of the previous FLIP-FLOP changes suitably.

Sequence detector A synchronous sequential circuit which can detect a sequence of binary input.

Sequential circuit Logic circuit whose outputs are determined by the sequence in which input signals are applied.

Serial adder An adder circuit in which addition is performed serially, i.e., bit by bit.

Serial data Data arranged as one bit at a time at a regular interval starting from the LSB.

Serial-to-Parallel converter A logic circuit that converts data from serial to parallel form.

Shift register A cascade of FLIP-FLOPs in which the data gets shifted in response to clock pulses.

State assignment Assigning binary values to states in a sequential circuit.

State diagram It is a graphical representation of a sequential circuit indicating the states, input values, output values, and directed arcs for state transitions.

State reduction Process of reducing the states by eliminating the redundant states in sequential circuits.

State table (or State transition table) A tabular form of representing the behaviour of a sequential circuit.

Synchronous counter A digital counter in which all the FLIP-FLOPs are clocked simultaneously.

Timing diagram Graphical representation of various signals with reference to time.

Transition table A tabular form of representing the behaviour of a sequential circuit.

Twisted-ring counter A shift register with its complement output ($\overline{Q}$) of the last stage connected to the D-input of the first stage.

UP-counter A counter that increments on every clock pulse.

REVIEW QUESTIONS

8.1 The minimum number of FLIP-FLOPs required for a decade counter is _____ .

8.2 Race condition may exist in _____ sequential circuits.

8.3 The modulo of a 4-bit binary counter is _____ .

8.4 The count of a 4-bit binary DOWN Counter is 0000. When a clock pulse is applied its count will be _____ .

8.5 The flow table of an asynchronous sequential circuit with 6 states and 2 inputs will have _____ rows and _____ columns.

8.6 The cascade of divide-by-5 counter followed by divide-by 2 counter is in state 0000. When a clock pulse is applied its state will be _____ . Assume negative edge-triggered circuits.
8.7 The modulo of a 4-stage twisted-ring counter is _____ .
8.8 A ripple counter is _____ sequential circuit.
8.9 A 4-bit binary code 1010 will be represented as _____ in temporal code.
8.10 The speed of an asynchronous counter is _____ than that of a synchronous counter.
8.11 The number of inputs which can change simultaneously in a fundamental mode asynchronous circuit is _____ .
8.12 The number of pulse inputs which are allowed to be present simultaneously in a pulse-mode asynchronous sequential circuit is _____ .
8.13 A _____ shift register can be used for SISO, SIPO, PISO, and PIPO of data.
8.14 The minimum number of FLIP-FLOPs required to generate a sequence of 9 bits is _____ .
8.15 The maximum possible number of states in a clocked sequential circuit having 5 FLIP-FLOPs is _____ .
8.16 The modulo of a 4-stage twisted-ring counter is _____.
8.17 A 3-bit synchronous counter can be converted to a mod-8 ring-counter by using a _____.
8.18 Race condition may exist in _____ sequential circuits.
8.19 The flow table of an asynchronous sequential circuit with 6 states and 2 inputs will have _____ rows and _____ columns.
8.20 Hazard may occur in _____ sequential circuits.
8.21 An asynchronous sequential circuit consisting of NOR latch is required to be hazard free. Its inputs should be supplied from _____ combinational circuit.
8.22 An asynchronous sequential circuit consisting of a NAND latch and OR-AND combinational circuit is _____.
8.23 Essential-hazard causes _____ of an asynchronous sequential circuit.
8.24 An asynchronous sequential circuit with NOR latch is free of static- ____ hazard.
8.25 An asynchronous sequential circuit with NAND latch is free of static- ____ hazard.
8.26 Critical race must be _____ in an asynchronous sequential circuit.
8.27 An asynchronous sequential circuit reaches the same stable next state through two different paths in the flow table _____ race occurs in this circuit.
8.28 A 3-input sequential circuit will have _____ number of arcs emanating from a state.
8.29 In a state diagram if a directed arc terminates on the same node from which it has emanated, the next state will be _____.
8.30 In a state table containing four rows, the number of states is _____.
8.31 0/1 written by the side of a directed arc in a state diagram indicates 0 _____ and 1 _____.
8.32 A minimum number of _____ changes must occur in the output corresponding to dynamic hazard.
8.33 A minimum of _____ paths must exist between an input variable and output for dynamic hazard to occur.

PROBLEMS

8.1 Explain the operation of the bi-directional shift register of Fig. 8.4.
8.2 Explain the operation of a twisted-ring counter and give its state diagram.
8.3 Verify the decoder logic of Fig. 8.7.

8.4 A stepper-motor drive circuit requires four signal waveforms given in Fig. 8.92. Design a sequence generator to provide the necessary signals for this stepper-motor drive.

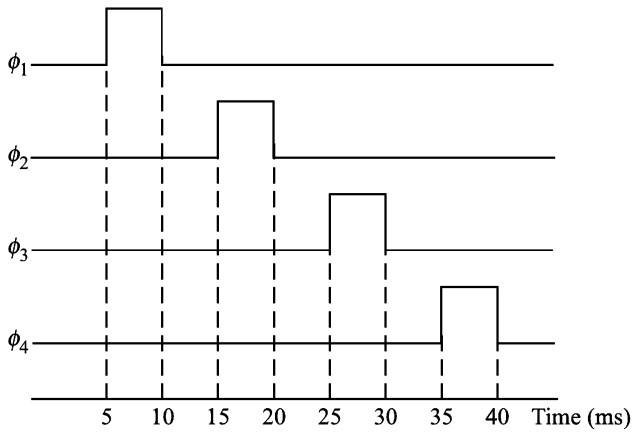


Fig. 8.92 ***Waveforms for Prob. 8.4***

8.5 Write the count sequence of a 3-bit binary DOWN counter. Design a ripple counter using FLIP-FLOPs for this sequence.

8.6 Design a 4-bit binary UP/DOWN ripple counter with a control for UP/DOWN counting.

8.7 Design a 4-bit asynchronous counter with provision for asynchronous loading.

8.8 The latch used for eliminating resetting difficulties due to unequal internal delays of FLIP-FLOPs is shown in Fig. 8.13. Explain the operation of this latch and show how the resetting difficulties are eliminated.

8.9 Design the following ripple counters using FLIP-FLOPs:
(a) Divide-by-5, (b) Divide-by-7.

8.10 In a 7492 divide-by-12 ripple counter if Q_D output is connected to A input and the pulses are applied at the B input, find the count sequence.
Show that the frequency divisions of 3, 6, and 12 are obtained at the Q_C, Q_D, and Q_A outputs respectively.

8.11 Design the following counters using 7492 used as in Problem 8.10:
(a) Divide-by-7, (b) Divide-by-9, (c) Divide-by-11.

8.12 Convert 7493 into a 4-bit DOWN counter.

8.13 Design a divide-by-128 counter using 7493 ICs.

8.14 Design a divide-by-96 counter using 7490 ICs.

8.15 Design a divide-by-78 counter using 7493 and 7492 (as divide-by-6) counter ICs.

8.16 Sketch the output waveforms of the counter circuit of Fig. 8.29.

8.17 Sketch the output waveforms of the counter circuit of Fig. 8.30.

8.18 If the counter 74161 is replaced by 74163 in the Ex. 8.13, what will be its modulus? Also draw the output waveforms.

8.19 Explain the operation of the circuit of Fig. 8.31.

8.20 Design a mod-12 UP counter using 74193 IC and compare it with the circuit of Fig. 8.29.

8.21 Design a divide-by-5 DOWN counter using 74192 IC. Make use of the borrow output. Sketch the output waveforms.

8.22 Construct the state diagram of a modulo-4 UP/DOWN counter. Design its circuit using *J-K* FLIP-FLOPs.

8.23 Design a circuit that gives the input–output relationship shown in Fig. 8.93.

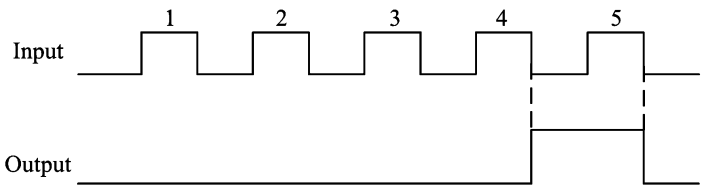


Fig. 8.93 ***Waveform for Prob. 8.23***

8.24 Design a decade counter to count in the Excess-3 code sequence. Use minimum number of *J-K* FFs.

8.25 Design a mod-12 normal binary counter for the count sequence 0000 to 1011 using 74163 IC.

8.26 A clocked synchronous sequential circuit using positive-edge-triggered D FFs has an input X and an output Y.

The excitation equations are:

$$D_1 = Q_1 \cdot \overline{X} + \overline{Q_1} \cdot Q_0 \cdot X + Q_1 \cdot \overline{Q_0} \cdot X$$
$$D_0 = Q_0 \cdot \overline{X} + \overline{Q_0} \cdot X$$

and the output equation is

$$Y = Q_1 \cdot Q_0 \cdot X$$

(a) Draw its circuit diagram,
(b) Obtain its state diagram,
(c) Redesign the circuit using *J-K* FFs.

8.27 For a clocked sequential circuit shown in Fig. 8.94, obtain
(a) Excitation and output equations,
(b) The output sequence for an input sequence of 101001 assuming initial state to be $Q_1 Q_0 = 00$.

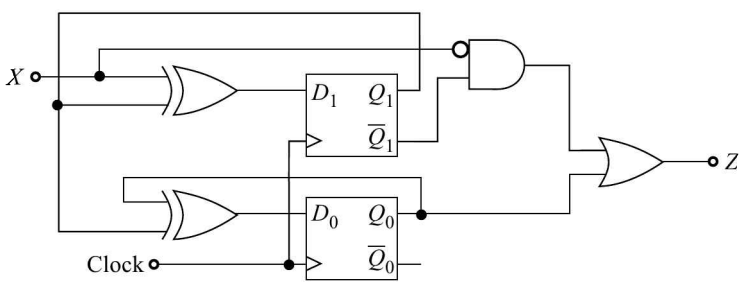


Fig. 8.94 ***Figure for Prob. 8.27***

8.28 For the state diagram shown in Fig. 8.95, obtain the state table and design the circuit using minimum number of *J-K* FFs.

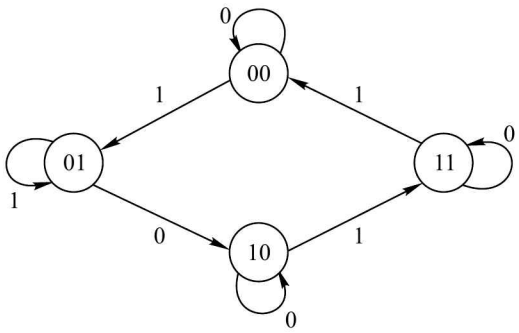


Fig. 8.95 ***State Diagram for Prob. 8.28***

8.29 For the state diagram shown in Fig. 8.96, design the circuit using
(a) *T* FFs, (b) *S-R* FFs, (c) *J-K* FFs.

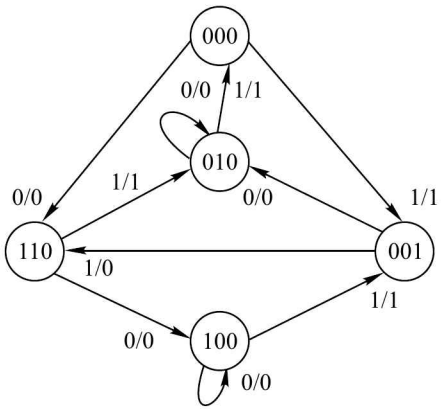


Fig. 8.96 ***State Diagram for Prob. 8.29***

8.30 Design the clocked sequential circuit for Example 8.24 using minimum number of *J-K* FFs.

8.31 For the state table given in Table 8.30, obtain the state diagram. If possible, determine the reduced state table.

8.32 Design the circuit for Prob. 8.31 using minimum number of FFs of
(a) Edge-triggered *D* type,
(b) *J-K* type.

8.33 Obtain the output sequence for Prob. 8.31 for an input sequence of 01110010011. Assume initial state $Q_2 Q_1 Q_0 = 000$.

Table 8.30

Present state			Next state						Output	
			$X=0$			$X=1$			Y	
Q_2	Q_1	Q_0	Q_2^*	Q_1^*	Q_0^*	Q_2^*	Q_1^*	Q_0^*	$X=0$	$X=1$
0	0	0	1	0	1	0	0	1	0	0
0	0	1	0	1	1	0	1	0	0	0
0	1	0	1	0	1	1	0	0	0	0
0	1	1	1	1	0	0	0	0	1	0
1	0	0	0	1	1	0	1	0	0	0
1	0	1	1	0	1	0	0	1	1	1
1	1	0	1	1	0	1	1	1	0	1
1	1	1	1	1	0	0	0	0	1	0

8.34 (a) Explain the operation of the transition table shown in Fig. 8.97 from initial total state $Q_1 Q_2 X_1 X_2$ of
(i) 0001 when $X_1 X_2$ becomes 11.
(ii) 1111 when $X_1 X_2$ becomes 01.
(b) Determine whether there are race conditions in (a)–(i) & (ii) above and whether they are critical or noncritical.

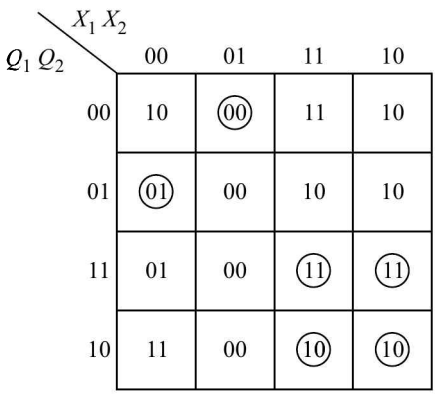


Fig. 8.97 ***Transition Table***

8.35 For an asynchronous sequential circuit, the sequence of internal states $Q_1 Q_2$ obtained is 00, 00, 01, 11, 11, 01, 00 for an input sequence of $X_1 X_2$: 00, 10, 11, 01, 11, 10, 00.
(a) Determine transition table
(b) Next-state logic equations
(c) Logic circuit

8.36 In the flow table of Fig. 8.80, verify the entries in the rows '*e*' and '*f*'.

8.37 Design the serial adder circuit of Ex. 8.26 using *J-K* FLIP-FLOPs.

8.38 Design the sequence detector circuit of Ex. 8.27 using *J-K* FLIP-FLOPs.