Switching Theory and Logic Design

C. V. S. Rao

Pearson

8

Asynchronous Sequential Circuits

Learning Objectives

After studying this chapter, you will know about:

- Fundamental mode circuits and pulse mode circuits.
- Design steps.
- Pitfalls, remedies and essential hazards.
- Cycles, critical and non-critical races.
- Asynchronous pulse counters.

We encounter many situations when synchronising clock pulses are not available. For example, counting visitors to a library, counting cosmic particles or meteors impinging on earth's atmosphere are asynchronous phenomena. Furthermore, asynchronous circuits are generally faster than synchronous circuits as the speed of the latter is limited by the clock. At times, a large synchronous system may allow some of its subsystems to operate asynchronously in order to maintain the overall speed.

8.1 INTRODUCTORY CONCEPTS

Asynchronous sequential circuits are normally classified into **fundamental mode circuits** and **pulse mode circuits.** In **fundamental mode,** it is assumed that only one input can change at a time and all multiple input changes give rise to don't care entries. Further, it is assumed that the input changes occur only when the machine is in a stable state. (The concept of stable state and transitory states will be explained later.) The inputs and outputs assume logic levels in fundamental mode. In pulse mode, the inputs are pulses while the outputs may be pulses or levels. The model of an asynchronous sequential circuit is given in Fig. 8.1. Compare this with that of the synchronous machine of Fig. 7.8. There are essentially two differences. Firstly, there is no clock in asynchronous circuits. Secondly, the blocks marked 'Delay D' in the feedback path are essentially

some kind of flip-flops, either J-K-M-S or T or D or others, in the case of synchronous circuits, while in asynchronous circuits they are mere path delays. Fundamental mode circuits will be presented first and pulse mode circuits later.

In synchronous sequential circuits, you have learnt that the present state (denoted by lower case y's) refers to the state of the machine before the clock pulse and the next state (denoted by capital Ys) is the state attained or reached by the machine after the clock pulse. Thus the clock pulses separate in time the present state and next state of synchronous sequential machines. The present state (indicated by row label) and the inputs (indicated by column label) together decide the output of the machine as well as the next state (row). A change of state involves change of row after the clock pulse.

In asynchronous sequential circuits, a slightly different orientation of thinking is required as there are no timing pulses. As usual, Ys are produced by a combinational circuit as functions of primary inputs and y's, which are called secondary inputs or secondary (internal) state variables. Ys are simply fed back to replace y's after a small delay in the path. For clarity in illustration, delays are shown in a feedback path. Actually, the normal path delay itself would serve the purpose of delays. After the delay, ys will assume the values of Ys. As a result of the closed loop, ys and Ys keep flowing (changing) until all $y_i \equiv Y_i$, which represents a **stable state.** It is assumed that any change of inputs occurs only after the ys and Ys settle down in a stable state.

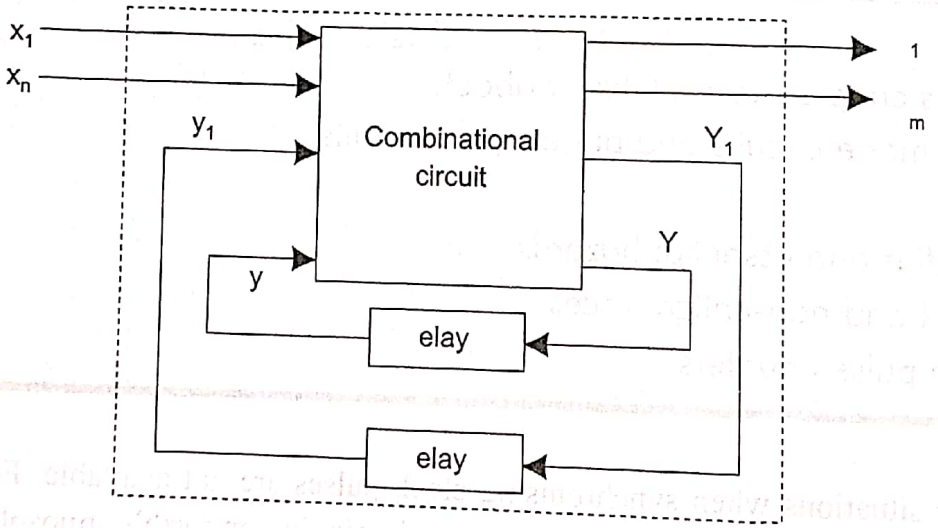


Fig. 8.1 Structure of an asynchronous sequential machine

In this case, we do not talk of a state diagram as in the case of synchronous sequential machines, but we start with a **flow table,** and as inputs change, the machine keeps flowing from one stable state to another stable state. There may be some transitory unstable states in this process.

8.2 FUNDAMENTAL MODE CIRCUITS—DESIGN STEPS

One goes through the following steps to design an asynchronous fundamental mode sequential machine.

Step I **Word statement** A problem's word statement has to be clearly understood. Important input sequences which cause changes in the output are listed for reference in the design process.

Step II **Primitive flow Table** Normally, we assume an initial stable state 1 of the machine with all inputs = 0. Then we make input changes following the sequences mentioned in the word statement, leading to changes in outputs. For every change in the inputs, introduce a new stable state by opening a new row and

...ering a circled number indicating a new stable state in the column corresponding to the changed inputs. ...so indicate a transitory state in the same column of the previous row. We first follow the given sequences, ...hich cause changes in outputs. Then we exhaust all possible transitions from the stable states without ...ntradicting the word statement. This process may require introducing new stable states and hence new rows. ...the input columns and the given sequences are finite, the process must terminate.

The following steps are better understood through the examples given next.

...ep III Merger Diagram and minimum-row-merged flow table

...ep IV Adjacency diagram and secondary state assignment, which does not cause **"Critical Races"**

...ep V Transition (also called Excitation) table

...ep VI Synthesise Ys and outputs as functions of inputs and ys. Draw a neat circuit.

4.3 DESIGN EXAMPLE 1

...certain asynchronous fundamental mode sequential circuit has two inputs x_1, and x_2 and one output Z. The ...tput Z has to change from 0 to 1 only when x_2 changes from 0 to 1 while x_1 is already 1. The output Z has to ...ange from 1 to 0 only when x_1 changes from 1 to 0 while x_2 is1.

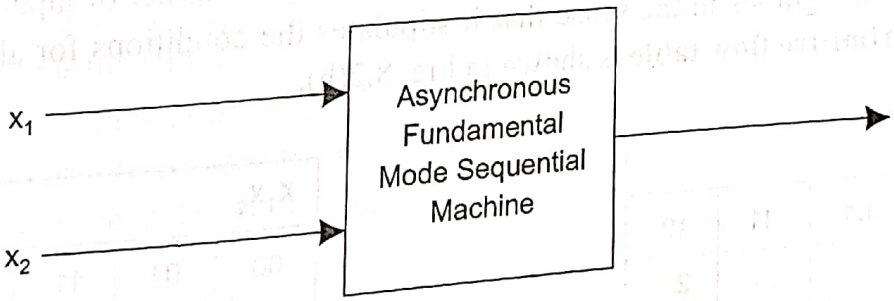
(a) Find a minimum row merged flow table.

(b) Find a valid secondary variable assignment.

(c) Make the output fast and flicker-free.

(d) Obtain hazard-free excitation and output functions.

...ep I **Word statement** The problem's word statement has to be interpreted in unambignous terms. ...portant sequences, which cause changes in outputs, are listed for reference purpose.



$x_1 = 0\ 1\ 1\ 0$

$x_2 = 0\ 0\ 1\ 1$

$Z = 0\ 0\ 1\ 0$

Important sequences

$x_1x_2 = 10 \rightarrow 11 \Longrightarrow Z = 0 \rightarrow 1$

$x_1x_2 = 11 \rightarrow 01 \Longrightarrow Z = 1 \rightarrow 0$

...ep II **Primitive flow table** For the benefit of the reader, various stages in the development of the ...imitive flow table are discussed below. We first assume an initial stable state with $x_1 = x_2 = 0$ indicated by a ...ircled one–①–associated with output $z = 0$ marked adjacent to the circled number. By way of notation, all ...ircled numbers indicate **stable states** and uncircled numbers indicate **transitory (unstable) states.** Let x_1 ...nd x_2 be at 0 level in the initial state. Then we follow the input sequence $x_1x_2 = 00 \rightarrow 10 \rightarrow 11$ which changes ...Z from $0 \rightarrow 1$. We have to send the machine to a stable state ② on the change of inputs from 00 to 10. For this ...rpose, we enter an unstable 2 in 10 column and open a new row and have the stable state $②_o$ in 10 column

in the new row. We fill a dash in the 1st row of column 11 since the double-input change from 00 to 11 does not occur as assumed in the very beginning. Proceeding further with the input sequence, x_1x_2 changes from 10 to 11 and, in accordance with the word statement, Z should change from 0 to 1. Hence we send the machine to another stable state via the transitory state 3 by opening a new row and have the stable state ③$_1$ under the input 11 column, as shown. We also indicate output Z = 1 by entering 1 by the side of the state.

Having become 1, Z goes to 0 only when the input sequence 11 → 01 occurs. Therefore, send the machine to another state ④$_0$ in a fresh row under column 01 via the transitory 4, as shown. With this, we have strictly followed the word statement. From every stable state, double-input changes are marked by dashes. See the partially filled primitive flow table shown in Fig. 8.2(a). For all other possible changes in inputs x_1x_2, we have to let the machine know what it should do.

Starting from ① if $x_1x_2 = 01$ occurs, there is no need of changing the output Z which may continue to be at 0 level. We, therefore, send the machine to ④$_0$ via 4 marked in the 01 column. Now look at the 2nd row. From ②$_0$, if x_1x_2 becomes 00, we may send the machine to ①$_0$ without contradicting the word statement and hence 1 is entered.

Now, look at the 3rd row. From ③$_1$, if x_1x_2 becomes 10, what should the machine do? Z should continue to be at level 1and there is no stable state with Z = 1, in column 10. There is, thus, a need to introduce a new stable state with Z = 1 in column 10. Hence, open a new row and have the entry ⑤ as shown and direct the transition accordingly. From ④ if the inputs x_1x_2 become 00, the machine has to be sent to a state with Z=0 which is fulfilled by the stable state ①.

Following the same procedure, for the transition from ④ on $x_1x_2 = 11$, we need to introduce a new state ⑥$_0$. Similarly, for the transition from ⑤$_1$ on $x_1x_2 = 00$, a new state ⑦ has been introduced. Finally, from ⑦$_1$, on x_1x_2 becoming 01, the machine has to be sent to a new state ⑧$_1$ with Z = 1 indicated in a new row. Notice that the procedure has to terminate as the machine has a finite number of inputs and outputs and the word statement has no ambiguities in the sense that it stipulates the conditions for all possible changes of output. The complete **primitive flow table** is shown in Fig. 8.2(b).

X_1X_2			
00	01	11	10
①$_0$		--	2
	--	3	②$_0$
--	4	③$_1$	
	④$_0$		--

(a) Partially filled Table

X_1X_2			
00	01	11	10
①$_0$	4	--	2
1	--	3	②$_0$
--	4	③$_1$	5
1	④$_0$	6	--
7	--	3	⑤$_1$
--	4	⑥$_0$	2
⑦$_1$	8	--	5
7	⑧$_1$	3	--

Step III **Merger diagram** At this stage, we would like to explore the possibility of reducing the number of rows in the primitive flow table for the simple reason that we have to assign secondary state (also called internal) variables for each row. In each column, we introduced only two stable states, one with output 0 and the other with output 1. Clearly they cannot be equivalent. We now introduce another concept of **"merger of rows"**, which is analogous to the term **compatibility** used in minimising **incompletely specified sequential machines (ISSM'S)** discussed in a subsequent chapter. Consider, for instance, rows ① and ②. Using the unspecified entries and noting that the transitory unstable entries do not conflict, we might **"merge"** these two rows into one row shown below.

X_1X_2			
00	01	11	10
①₀	4	--	2
			②₀
1	--	3	

Merges into

X_1X_2			
00	01	11	10
①₀	4	3	②₀

In such a situation, stable states ① and ② will have the same internal secondary state assignment. Notice that the secondary states are identified by the rows, whereas the stable states require specification of both the row and input column (also called **input state**). Hence, the need to have the concept of **"total state"** in asynchronous machines. In synchronous machines, a change of state involves a change of row. There is no question of unstable transitory states in synchronous machines as it is the clock pulse which controls the transition. In the absence of the clock pulse, the circuit is always stable and even the inputs are not available to the machine. In asynchronous machines, a change in the input columns (horizontal movement) may result in a change of stable state with the internal state (row) unaltered. Both the row and column together identify the total state in asynchronous machines.

Notice in the primitive flow table of Fig. 8.2(b) that there is one row for each stable state. The rows 1 and 3 cannot merge as the entries in column 10 are different which direct the machine to go to ② in one case and ⑤ in the other case. Now, consider rows 1 and 4 which merge into one row, as shown below.

X_1X_2			
①₀	01	11	10
	④₀	--	2
1		6	--

Merges into

X_1X_2			
00	01	11	10
①₀	④₀	6	2

It is now clear that the following rules hold in effecting a merger of rows,

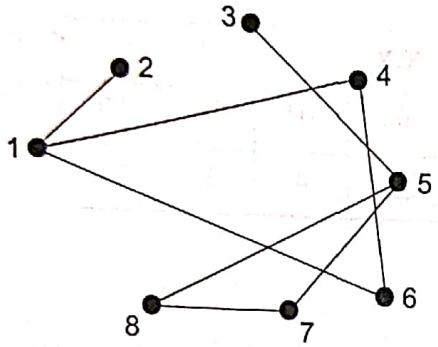
1. A stable or transitory (unstable) state and a dash (unspecified) merge into the corresponding state.
2. A stable state and a transitory state merge into the stable state.

Following the above rules, the rows are denoted by nodes and the possible mergers are indicated by edges connecting the nodes. See Fig. 8.3(a). Notice that we have to choose complete polygons for effecting mergers. A triangle formed by the nodes 1, 4, 6 is a complete polygon as every node is connected to every other node in

the set. Likewise, nodes, 5, 7, 8 form another triangle. Hence the chosen mergers are given below. A rectangle (not encountered in this example) with both the diagonals is a complete polygon.

Mergers : (1, 4, 6) (2) (3) (5, 7, 8)

Mergers (1, 2), (3, 5) are not needed but rows 2 and 3 have to be left as singleton stable states in order to cover all the stable states of the machine. Thus, we get a 4-row flow table shown in Fig. 8.3(b). Notice further that the outputs are indicated only for the stable states. Outputs of unstable entries will be considered later. The rows in the minimum row-merged flow table are labelled a, b, c and d for further discussion next.



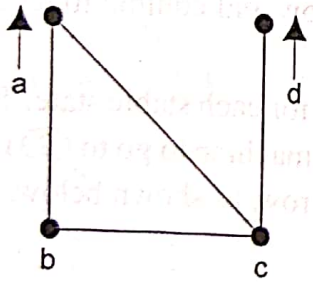
Mergers: (1, 4, 6) (2) (3) (5, 7, 8)

(a) Merger diagram

Rows	X_1X_2			
	00	01	11	10
a	①$_0$	④$_0$	⑥$_0$	2
b	1	--	3	②$_0$
c	--	4	③$_1$	5
d	⑦$_1$	⑧$_1$	3	⑤$_1$

(b) Merged flow table

Fig. 8.3 Merger diagram and minimum row-mergerd flow table



(a) Adjacency diagram

$PS(y_1y_2)$	X_1X_2	$NS(Y_1Y_2)$, Z		
	00	01	11	10
a = 00	(00), 0	(00), 0	(00), 0	01
b = 01	00	--	11	(01), 0
c = 11	--	00	(11), 0	10
d = 10	(10), 1	(10), 1	11	(10), 1

(b) Invalid assignment

$PS(y_1y_2)$	X_1X_2	$NS(Y_1Y_2)$, Z		
	00	01	11	10
a = 00	(00), 0	(00), 0	(00), 0	01, 0^f
b = 01	00, 0^f	00, 0^f	11, 0^f	(01), 0
c = 11	--	01, 0^f	(11), 0	10, 1^f
d = 10	(10), 1	(10), 1	11, 0^f	(10), 1

(c) Valid assignment and transition table

The superscript 'f' in Fig. 8.4 (c) indicates assignment of output same as that of the corresponding stable state.

Fig. 8.4 Secondary state variable assignment

Step IV **Adjacency diagram and secondary variable assignment**

Consider each transition from one stable state to another stable state through the unstable states. It is desirable that it involves change of a single secondary variable; Otherwise, we may encounter "races", discussed next. Hence the need to examine adjacencies of rows indicated by an edge connecting the rows—see the adjacency diagram shown in Fig. 8.4(a). Row c has to be adjacent to three rows—a, b, d. This is not possible using two secondary variables (also called internal state variables). A node designated or assigned by n variables can have at the most n adjacent nodes. The unstable entry 4 in Fig. 8.3(b) required adjacency between rows a and c. If this adjacency condition could be relaxed, we would be able to meet the remaining adjacencies with two secondary (internal) state variables only. Otherwise, we may need three secondary variables, in which case there will be eight rows and many unused (optional or don't care) entries. The importance of adjacency assignment would become clear when we discuss **"races"** next. The adjacency aspect will be discussed again after learning about races.

Suppose we assign two secondary state variables $y_1 y_2$ arbitrarily to the rows of 8.4(b). Remember that the stable states are identified by $y_i = Y_i$. Therefore, in the cells representing stable states, we have to fill the same entries for $Y_1 Y_2$ as $y_1 y_2$ of that row. For the transitory states, we have to fill the assignment of the corresponding stable state to which the machine has to be directed. Now, observe the entry in row 11, column 01. The next state entry 00 therein causes a **race.** The machine is directed to change its secondary state variable $y_1 y_2$ from 11 to 00, involving a double change which is referred to as a **race.** If we are lucky, y_1 will be faster than y_2. Then $y_1 y_2$ becomes 01 first where there is a dash which is preferably filled by 00 in order to direct the machine to the 00 row. Even otherwise, a moment later, y_2 also would change and the state would become 00, which is the correct state intended by design. However, if y_1 is sluggish and y_2 is faster, then $y_1 y_2$ momentarily becomes 10, changing the row in the same input column 01. Clearly, the machine lands in a wrong stable state ⑩ and gets stuck there. Such a hazardous situation is called a **critical race.** The assignment involving a critical **race** shown in Fig. 8.4(b) is not valid. Note that the **critical race** is caused because of two stable states in the same input column 01 and that the assignment for a transitory state involved a double change of secondary variables.

Observe that by making use of the dash in column 01, it is possible to avoid the critical race by directing the transition [see arrows marked in Fig. 8.4(c)] by assigning $Y_1 Y_2$ as shown. This is called a **cycle.** This does not involve double changes in secondary variables but needs one or more extra transitions and hence the corresponding delay in settling in a stable state. The unspecified entry came in handy to avoid the **critical race,** resulting in a valid assignment. Had there been only one stable state in the input column 01, it would not be a **critical race.** In that case, it would be a **non-critical race,** which is allowed. Note that each row represents only one internal (secondary) state but several stable states of the machine.

Having succeeded in obtaining a valid secondary state assignment, we now proceed to assign the outputs for the transitory states. If the output is to be fast, clearly, the output of the transitory state should be equal to the corresponding stable state. If the output has to be flicker-free, we have to examine all possible transitions between every pair of stable states with the same output and ensure that the intervening transitory states also will have the same output. Such a situation is not present in this example.

Step V **Transition table** The valid **flow table** is given below. Under the inputs $x_1 x_2$, the first column indicates Y_1, the second column denotes Y_2, and the entry after comma gives the output Z.

(y_1y_2)	$(Y_1Y_2), Z$ X_1X_2			
	00	01	11	10
00	00, 0	00, 0	00, 0	01, 0
01	00, 0	00, 0	11, 0	01, 0
11	--	01, 0	11, 0	10, 1
10	10, 1	10, 1	11, 0	10, 1

Step VI Synthesis Map the functions and obtain a static hazard-free realisation of Y_1, Y_2 and Z. Draw a neat circuit.

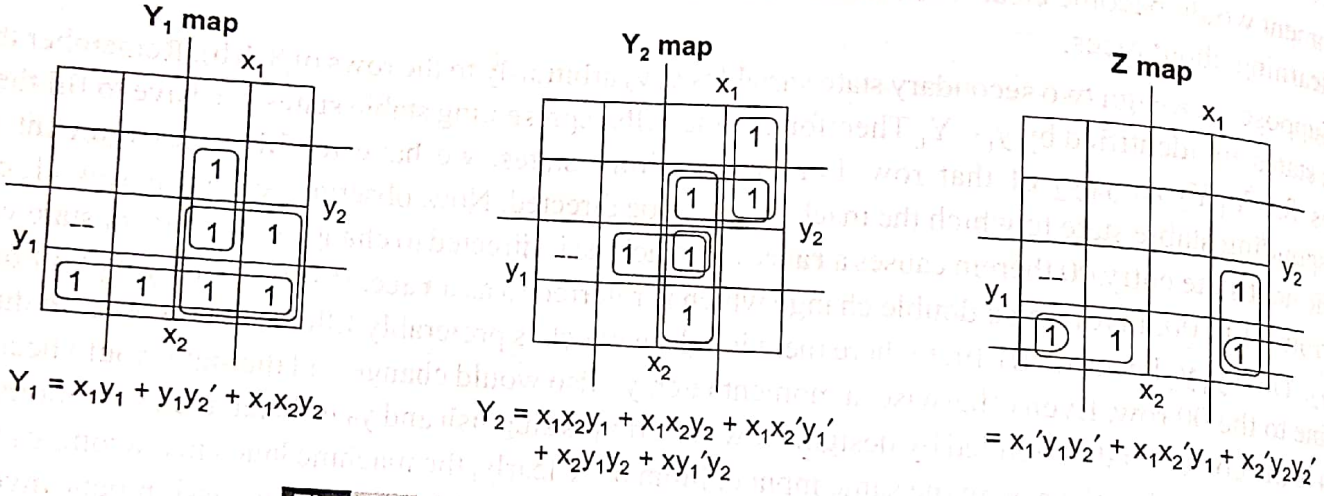


Fig. 8.5 Maps for excitations Y1, Y2 and output Z

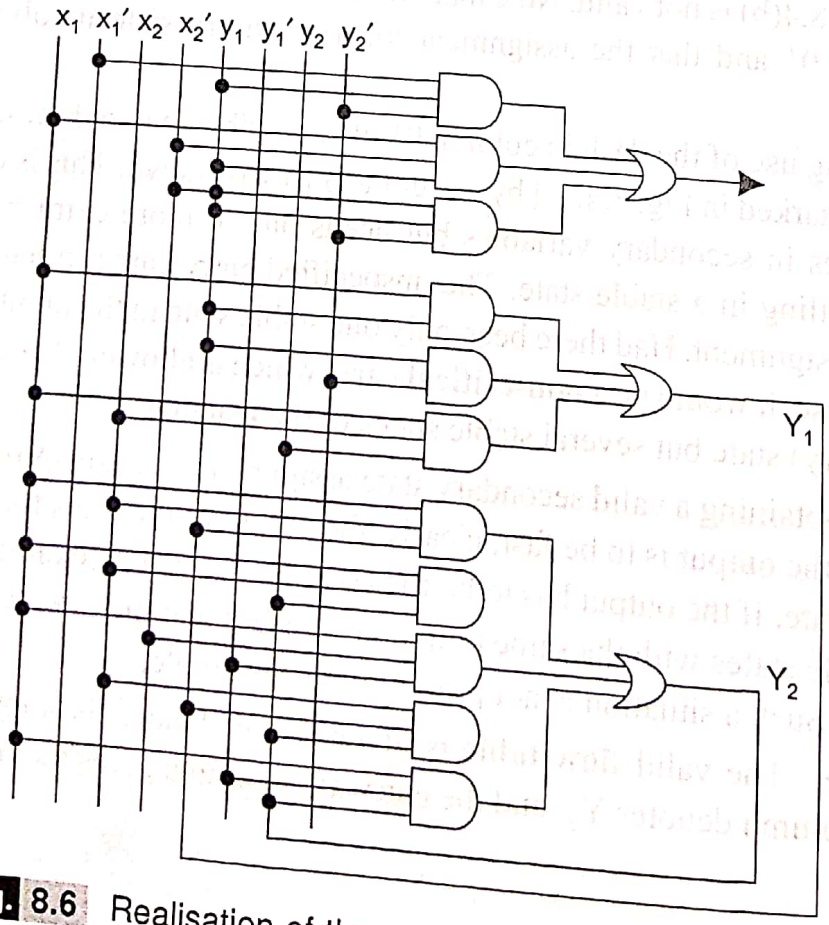


Fig. 8.6 Realisation of the asynchronous sequential circuit

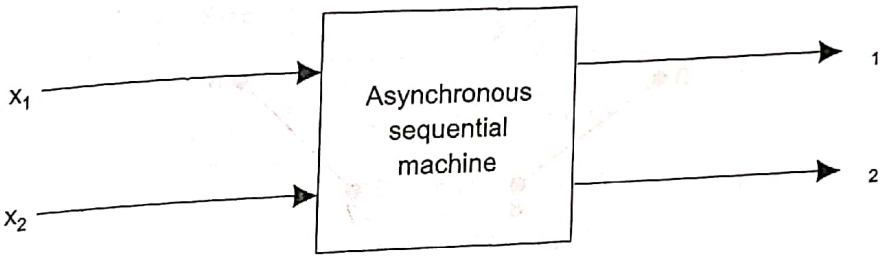
8.4 DESIGN EXAMPLE 2

In order to consolidate the concepts learnt, another example is presented in this section with all the various stages in the design.

A certain fundamental mode asynchronous sequential machine has two inputs x_1, x_2 and two outputs Z_1, Z_2. The output Z_i for i = 1 or 2 has to assume logic 1 level whenever the input that changed last was x_i and 0 at all other times.

1. Find a minimum row-reduced flow table.
2. Select a valid assignment and synthesise the circuit.
3. Does the machine have a power-on state with all $x_i = 0$ and all $Z_i = 0$?

Step I Word statement



This machine does not have a power-on state specifically defined.

$x_1x_2 = 00$ can be reached either from 01 or from 10. Hence, both the outputs being 00 will never occur except in the power-on state, which is only transitory. In steady state, the machine should have two stable states under each input column, one with $Z_1Z_2 = 01$ and the other with $Z_1Z_2 = 10$. If the power-on state is provided with $x_1x_2 = 00$, $Z_1Z_2 = 00$, the machine would never reach that state after leaving it.

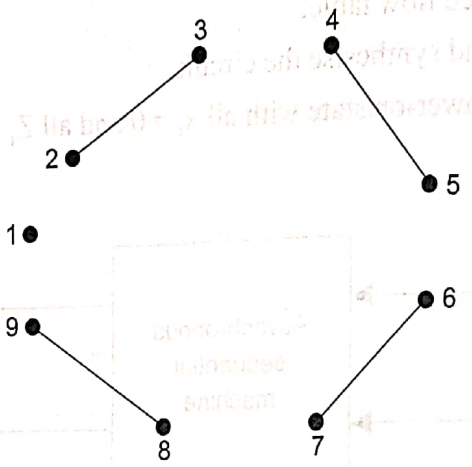
Step II Primitive flow table

x_1x_2			
00	01	11	10
①00	2	--	4
3	②01	6	--
③01	2	--	4
5	--	9	④10
⑤10	2	--	4
--	7	⑥10	8
3	⑦10	6	--
5	--	9	⑧01
--	7	⑨01	8

Fig. 8.7 Primitive flow table

Using only single input changes, 00 can be reached from 01 or 10. Let us first exhaust these possibilities by introducing the stable states ①, ②, ③, ④, ⑤. Now, follow the transition starting from ② with input 01. Stable states ⑥, ⑦ exhaust the possibility of reaching inputs 01 from 11. Stable state ⑧ exhausts the transition to 10 and ⑨ covers the last possibility of reaching 11 from 10. All double-input changes in each row are marked by dashes and the remaining cells are filled with some transitory or stable states.

Step III **Merger diagram and merged flow table**



Notice that ① is merely a power-on state. Also note that the primitive flow table does not contain any transitory state labelled 1. Once the machine leaves stable state ①, it will never return to that total state. Hence, we consider this as a transient behaviour and ignore it in forming the minimum row-merged flow table.

Mergers: (2, 3), (4, 5), (6, 7), (8, 9)

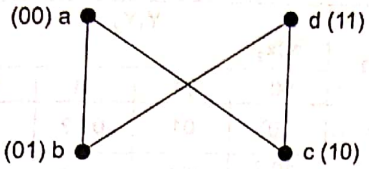
There will be four secondary internal states denoted by the rows a, b, c, d. The machine requires two binary state variables y_1, y_2.

Reduced (Merged) Flow Table

	X_1X_2			
Rows	00	01	11	10
a	③ 01	② 01	6	4
b	⑤ 10	2	9	④ 10
c	3	⑦ 10	⑥ 10	8
d	5	7	⑨ 01	⑧ 01

The row labels a, b, c, d facilitate examining adjacencies next. Outputs Z_1, Z_2 are marked for the stable states.

Step IV **Adjacency diagram and assignment** Noting the transitory states in each row of the merged flow table, adjacency requirements are indicated by connecting nodes by edges as shown below. Notice that it is possible to comply with all the adjacency requirements using only two secondary variables y_1, y_2. Adjacency assignment are marked on the vertices denoting rows.



Step V **Transition table with a valid assignment** We rewrite the flow table with the assigned values. As each transition requires only a single change in secondary variables, there will be no races unlike the previous example. Remember that the cells containing stable states are identified by $Y_i = y_i$, which are circled. Where there are transitory states, the row assigned to the corresponding stable state has to be entered. Special care should be taken in making the entries. Finally, the outputs are entered after a separating comma.

y_1y_2	Y_1Y_2, Z_1Z_2 / X_1X_2			
	00	01	11	10
a = 00	(00), 01	(00), 01	10, 10	01, 10
b = 01	(01), 10	00, 01	11, 01	(01), 10
d = 11	01, 10	10, 10	(11), 01	(11), 01
c = 10	00, 01	(10), 10	(10), 10	11, 01

Step VI **Synthesis of the circuit** Notice that the columns of the transition table (assigned flow table) correspond to Y_1, Y_2, Z_1, Z_2 as functions of x_1, x_2, y_1, y_2. The maps are given below.

Y_1 map

y_1 \ x_1x_2	00	01	11	10
y_2	0	0	1	0
	0	0	1	0
y_2'	0	1	1	1
	0	1	1	1

$Y_1 = x_1x_2 + x_1y_1 + x_2y_1$

Y_2 map

y_1 \ x_1x_2	00	01	11	10
y_2	0	0	0	1
	1	0	1	1
y_2'	1	0	1	1
	0	0	0	1

$Y_2 = x_1x_2' + x_1y_2 + x_2'y_2$

Z_1 map

y_1 \ x_1x_2	00	01	11	10
y_2	0	0	1	1
	1	0	0	1
y_2'	1	1	0	0
	0	1	1	0

$Z_1 = x_1y_1'y_2' + x_2'y_1'y_2' + x_1'y_1y_2 + x_2y_1y_2 +$ terms to be added for hazard-free output $x_1x_2'y_1' + x_1'x_2'y_2 + x_1'x_2y_1 + x_1x_2y_2$

Z_2 map

y_1 \ x_1x_2	00	01	11	10
y_2	1	1	0	0
	0	1	1	0
y_2'	0	0	1	1
	1	0	0	1

$Z_2 = Z_1'$

Notice that all the terms are selective prime implicants. Drawing the logic circuit is left to the reader.

8.5 RACES, CYCLES AND VALID ASSIGNMENTS

Wherever a change of more than one secondary variable is required for the machine to reach a stable state, a race exists between the secondary variables. Consider the flow table of Fig 8.8.

y_1y_2	Y_1Y_2			
	x_1x_2 00	01	11	10
00	(00)	01	01 ?	11
01	10 ? 11 √	(01)	00 ?	(01)
11	10	01	(11)	(11)
10	(10)	01	11	01

Fig. 8.8 Illustration of races and cycles

Look at the entry in the row 01 column 00. It is an unstable 10 which means that both the variables y_1y_2 have to change simultaneously from 01 to 10 to reach the stable state (10). It is most unlikely that two variables will change exactly at the same time. This is called a **race.** If y_2 is a little faster than y_1, then y_1y_2 follows the transition $01 \rightarrow 00$ and then to 10 but on y_1y_2 becoming 00, the machine would reach a stable state and does not move out until inputs change. Thus, there is a possibility of the machine landing in a wrong state; such a situation is called a **critical race.** If y_1 is faster than y_2, the transition would be $01 \rightarrow 11$ and then to 10. The intermediary entry is an unstable 10, which directs the machine to reach the desired state and there is no hazardous behaviour. In order to avoid the critical race, we change the entry in row 01 column 00, from 10 to 11 which directs the transition indicated by arrows. This is called a **"Cycle".** It is common to introduce cycles to avoid **critical races.**

Now look at the column 01 which has only one stable state in row 01. Notice that the 01 unstable entry in row 10 causes a **race** between y_1, y_2 but whichever is faster than the other, the machine has to finally land in the correct state and hence it is referred to as a **"non-critical race".** Such non-critical races are usually permitted. It is safe to direct the transition by proper assignment of the intervening cells instead of filling them with dashes.

Now focus on the entries in column 11. The two unstable entries 01 and 00 result in a never-ending cycle. The entry in row 00 directs the machine to go to row 01 and vice versa. Care must be taken that such cycles must not be contained in any secondary variable assignment.

Finally look at the column 10, which contains two stable states in rows 01 and 11. The unstable entries in that column cause critical races which cannot be avoided unless the assignment is changed or augmented with an extra secondary variable. With three secondary variables, it is possible to assign two combinations to each of the four rows of the flow table so that any transition involving any two rows can be directed using a cycle of adjacent rows. Fig. 8.9 shows such assignments. Take a look at Fig. 8.9(a). Every row is adjacent, in one of its assigned combinations, to every other row. This is equivalent to having two equivalent secondary states for every row. As such, it is possible to have **cycles** directing the transitions and avoid **races,** more particularly **critical races.** It has been shown in literature that at most $(2n - 1)$ secondary variables will be required to assign 2^n rows. Fig. 8.9(b) shows an assignment for 8-row tables with five secondary variables.

y_3 \ y_1y_2	00	01	11	10
0	a	c	c	a
1	b	b	d	d

(a) Assignment in 4-row flow tables

$y_3y_4y_5$ \ y_1y_2	00	01	11	10
000	a	b	c	c
001	b	b	d	d
011	a	a	c	d
010	b	b	c	d
110	e	f	e	e
111	e	f	f	f
101	e	g	g	h
100	h	h	g	h

(b) Assignment in 8-row flow tables

Fig. 8.9 Race-free assignments

8.6 ESSENTIAL HAZARDS IN ASYNCHRONOUS CIRCUITS

A type of hazard called **essential hazard** occurs in asynchronous sequential circuits when a change in input variable (columns) is slower than the consequential changes in secondary state variables (rows). Consider a typical flow table given in Fig. 8.10(a). Initially, let the input x be 0 and the secondary state variables $y_1 y_2 = 00$. Clearly the machine is in stable state (00). Let the input x change from $0 \rightarrow 1$. Notice that x′ appears in the transition functions Y_1, Y_2. Due to some delay in inversion or anywhere in the path, it is possible that both the nodes denoting x and x′ may be for a short time, at 1 level, a situation similar to **static hazards** in combinational circuits. Such **static hazards** caused spurious spikes called **glitches** in combinational circuits, which are easily avoided by including some redundant terms [not shown in the expressions of Fig. 8.10(b) and (c)].

In asynchronous sequential machines, such a situation might cause the machine to land in a 'total state', different from the intended stable state. Let us see how.

Look at Fig. 8.10(d) wherein the inputs x, x′, secondary state variables y_1, y_2 and the excitations Y_1, Y_2 (same as next state variables) are tabulated. Focus on the situation that x is changing from $0 \rightarrow 1$ at $t = 0$. Look at the hazardous situation obtaining for a short moment between 0 and τ. Input x has changed to 1 but x′, after the inverter, did not change yet. The nodes x and x′ are to be considered as distinct variables. Both x and x′ are 1 momentarily. Look at the expressions for Y_1 and Y_2 in (b) and (c). For this hazardous moment $Y_1 Y_2$ will be 01. After a little while, $y_1 y_2$ will also become 01. See the dotted line indicating the flow.

For $t < \tau$, the new values of $Y_1 Y_2$ become 11 because x′ is still at 1. It follows that $y_1 y_2$ becomes 11 as shown by the dots again. In the meantime, x′ assumes its correct value 0. Notice that the state variables have changed twice already from 00 to 01 and then to 11.

Look at the row for $t = \tau$. Compute $Y_1 Y_2$ which would become 10, which would flow to the next row as $y_1 y_2 = 10$. Now look at the row for $t > \tau$ and satisfy yourself that $Y_i \equiv y_i \Rightarrow$ stable state. Now trace the transition from the initial stable state to the next stable state.

y_1y_2 \ x	0	1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

(a) Flow table

Y_1 map

y_1y_2 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

$Y_1 = xy_1 + x'y_2$

(b) Next state function Y_1

Y_2 map

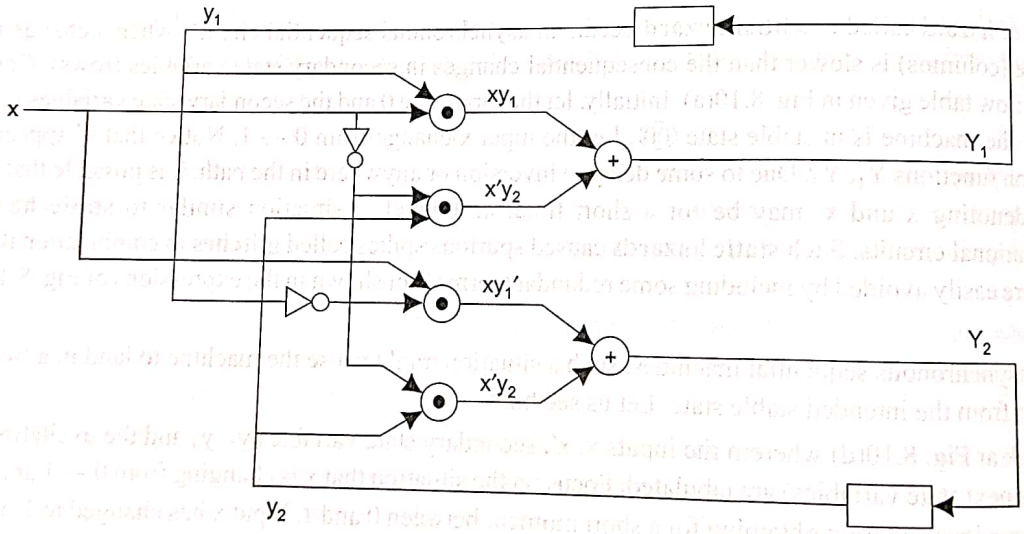
y_1y_2 \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

$Y_2 = xy_1' + x'y_2$

(c) Y_2

Timing	x x′	y_1 y_2	Y_1 Y_2	
t < 0	0 1	0 0	0 0	Stable
t ≥ 0	1 1?	0 0	0 1	
t < τ	1 1?	0 1	1 1	
t = τ	1 0	1 1	1 0	
t > τ	1 0	1 0	1 0	Stable

(d) Essential hazard situation



(e) Circuit realisation of the flow table

Fig. 8.10 Illustration of essential hazard

$$Y_1Y_2 = (00) \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow (10)$$

Notice that the intended stable state was (01) but the machine landed in a different state (10) after going through three transitory (unstable) secondary states. This is referred to as **essential hazard.**

In the literature on this subject, it has been proved that wherever a single change and three consecutive changes of an input lead the machine to different stable states, there will be an essential hazard. A typical flow table containing essential hazard is given below.

x = 0	x = 1
(1)	2
3	(2)
(3)	4
1	(4)

For the sake of completeness, notice that the above table corresponds to the flow chart of Fig. 8.10(a) which does not include output logic.

One final word. It is possible to avoid essential hazards by providing sufficiently long delays in the feedback paths at appropriately chosen points in the circuit. The idea is to allow adequate time for the input changes to settle down before the secondary state variables change.

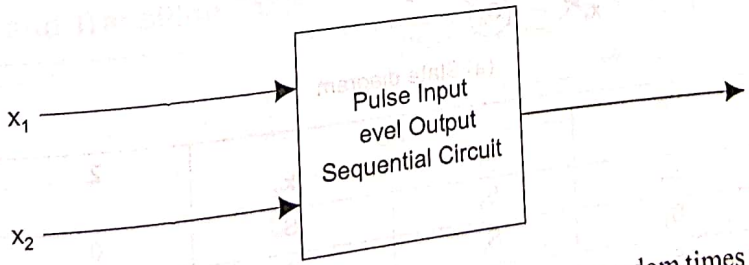
8.7 PULSE MODE ASYNCHRONOUS CIRCUITS

Pulse mode sequential circuits are useful for counting articles, events, or persons. Generally, no clock pulse generator is available to control the timing of events. Some kind of mechanism for producing pulses representing events has to be provided at the front end of the system. The outputs may be levels or pulses depending on the designer's choice. The methods used for designing synchronous circuits are found to be suitable for **asynchronous pulse mode circuits** too. In this section, the objective is to present design techniques through an example.

8.8 DESIGN EXAMPLE 3

An office room is partitioned into two cabins to accommodate one officer in each cabin. In order to conserve electricity, the management decides to have an automatic system to switch off the power supply to the room when not in use. The room has two doors on either side used by the officers independently. Using photocells and relays, design a circuit with two pulse inputs and one level output to switch the power on or off.

Step I **Word statement** The circuit has two pulse inputs x_1, x_2 and one level output z shown in the block diagram below.



All the possibilities of the officers entering or leaving at their own random times are to be considered and the appropriate output is to be provided.

Let 0 indicate no pulse and 1 indicate a pulse either on x_1 lead or on x_2 lead, but not both at the same time. Let us use the same letters x_1, x_2 for indicating the entry and exit of officers for preparing the following table which leads to the next step.

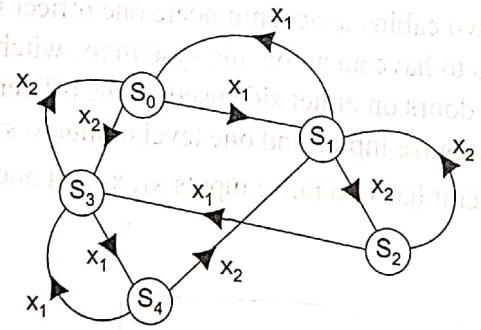
Event	x_1	x_2	Z	State
Initial	0	0	0	S_0
x_1 enters	1	0	1	S_1
x_1 leaves	1	0	0	S_0
x_1 enters	1	0	1	S_1
x_2 enters	0	1	1	S_2
x_2 leaves	0	1	1	S_1
x_1 leaves	1	0	0	S_0

Repeat, starting with S_2. It is more convenient to draw a state diagram, shown next.

Step II **State diagram** Note the following points which help in drawing the state diagram in Fig. 8.11(a) below.

- S_0 is the initial state when the room is vacant.
- S_1 is the state of the machine when only one officer x_1 is in the room.
- S_2 is the state of the machine when both x_1 and x_2 are in.
- S_3 represents the state when only x_2 is in.
- S_4 represents the state when x_1 enters the room and x_2 is already in.

Step III **State table** The information contained in the state diagram is posted in a tabular form yielding the state table shown in Fig. 8.11(b).



(a) State diagram

PS	NS		Z
	x_1	x_2	
S_0	S_1	S_3	0
S_1	S_0	S_2	1
S_2	S_3	S_1	1
S_3	S_4	S_0	1
S_4	S_3	S_1	1

(b) State table

Fig. 8.11 Design example of pulse input asynchronous sequential machine

Step IV Reduction of state table and standard form We reduce the state table using the partition technique. Associate the outputs to the states in the columns under NS. There are three different output configurations 11, 01, 10 which results in partition P_1.

$$P_0 = (S_0\ S_1\ S_2\ S_3\ S_4)$$

$$P_1 = (S_0\ S_2\ S_4)(S_1)(S_3)$$

$$P_2 = (S_0)(S_2\ S_4)(S_1)(S_3)$$

$$P_3 \equiv P_2$$

Hence, $S_2 \equiv S_4$. This result could have been observed at the beginning itself as these two states represent the state of the machine when both x_1 and x_2 are in the room.

Reduced State Table

PS	NS		Z
	x_1	x_2	
S_0	S_1	S_2	0
S_1	S_0	S_2	1
S_2	S_3	S_1	1
S_3	S_2	S_0	1

Step V List the states of rows for checking the standard form

$S_0\ S_1\ S_2$

$S_1\ S_0\ S_2$

$S_2\ S_3\ S_1$

$S_3\ S_2\ S_0$

The first occurrence is in order. Hence, it is in RSFST form.

Step VI There is no need to examine adjacencies in pulse mode circuits as the state transitions are triggered by pulses.

State Assignment and Transition Table

PS	NS ($Y_1 Y_2$)		Z
y_1y_2	x_1	x_2	
00	01	11	0
01	00	11	1
11	10	01	1
10	11	00	1

Step VII Choose D flip-flops Write the excitation functions by inspection.

$$D_1 = x_1 y_1 + x_2 y_1'$$

$$D_2 = x_1 y_2' + x_2 (y_1' + y_2)$$

$$Z = (y_1 + y_2)$$

Step VIII **Drawing the circuit is left to the reader.**

Note: 1. As we are dealing only with pulses, complement input variables do not arise. Even so, state variables may appear in both the forms.

2. The outputs are associated with the states. This is a Moore model.

8.9 ASYNCHRONOUS PULSE INPUT COUNTERS

In **asynchronous counters,** the count pulses drive only the first flip-flop in a cascade chain. The output of the first flip-flop drives the second flip-flop and the output of the second flip-flop drives the third flip-flop, and so on. For this reason, they are commonly called **ripple counters,** which cannot be described by Boolean equations used for clocked sequential circuits.

Contrast this with the synchronous counter wherein all the flip-flops are simultaneously driven by a clock pulse generator, in which case all the flip-flops ideally change state in parallel. In both types of counters, it is common to use J-K master-slave flip-flops connected as per design requirements. In the asynchronous case, the external pulses replace the clock for the first (LSB) flip-flop and the output of each flip-flop is connected to the clock input of the succeeding flip-flop. Remember that if both J and K are kept at 1 level (tied to the voltage V_{cc} in practice), the flip-flop changes state on every negative going transition at the output of the preceeding flip-flop.

Counting Sequence for a decade counter

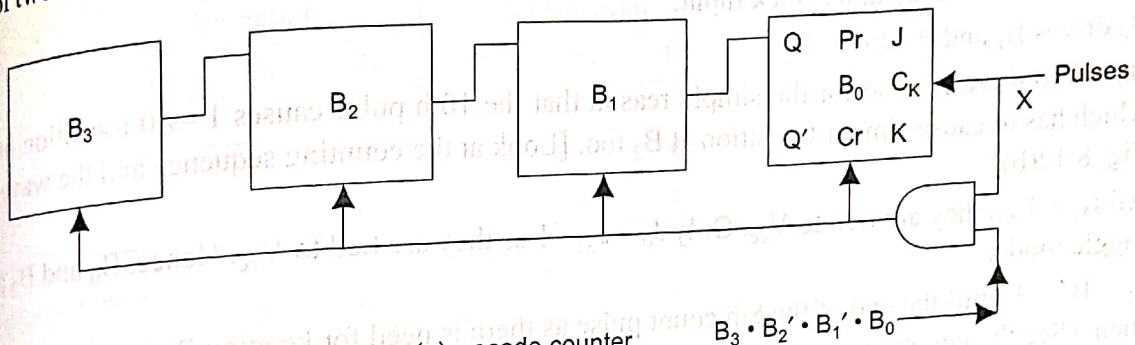
B_3	B_2	B_1	B_0	No. of Pulses on Input x
0	0	0	0	0 (zero)
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
0	0	0	0	10

A **decade counter** constructed with four flip-flops is shown in Fig. 8.12. Note that the binaries are arranged in the natural BCD order, with MSB to the left and LSB to the right. With four flip-flops it is possible to count from 0 to 15 but for the 10th input pulse, which ensures that the counter goes to the initial state. This is achieved in a number of ways. One simple method is to clear all the flip-flops at the prescribed time defined by the logic $X \cdot B_3 \cdot B_2' \cdot B_1' \cdot B_0$ where X is the input lead on which count pulses occur and B_i represents the corresponding outputs of the flip-flops.

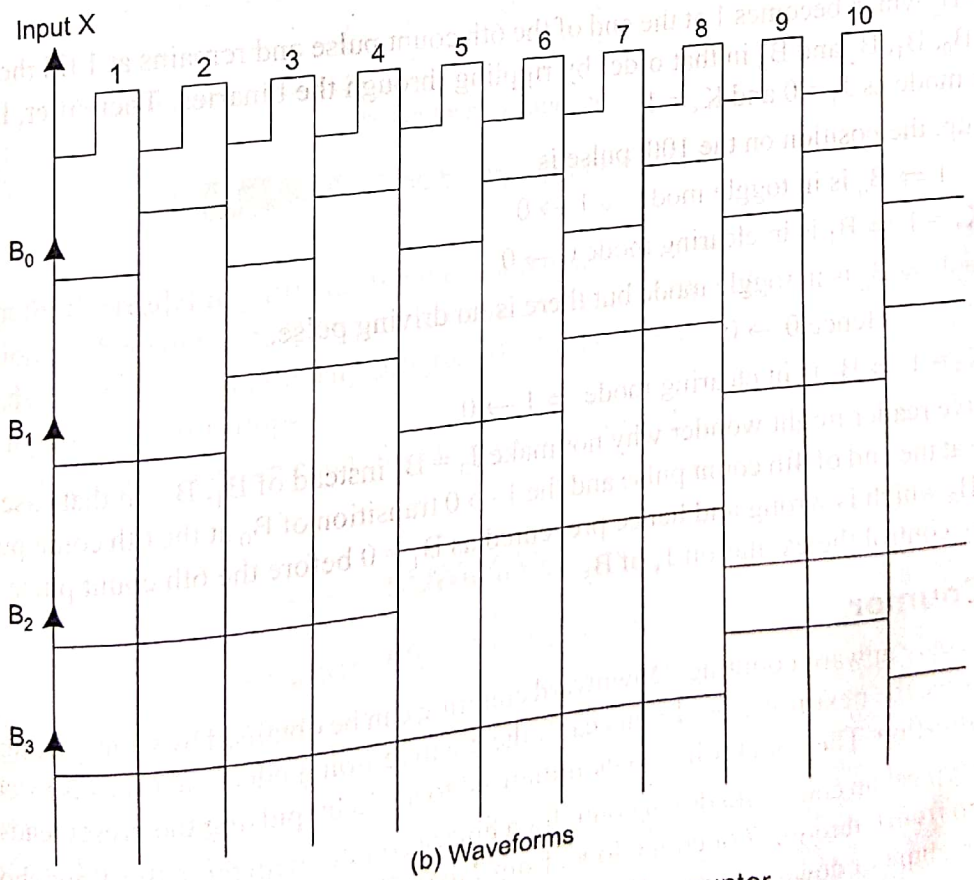
The counting sequence for the **decade counter** is given above. Notice that B_0 changes state for every incoming pulse on the negative-going edge, B_1 for every two pulses, B_2 for four pulses and B_3 at the end of eight pulses. The waveforms at the outputs of flip-flops are shown in Fig. 8.12(b). All the four flip-flops are

cleared to 0 on the 10th pulse which enables the AND gate to which other inputs are B_3, B_2', B_1', B_0 whose states represent the count $1001 = 9$. Note that the negative-going transition of B_3 as the counter proceeds from 9 to 0, can be used to drive another decade counter block. Any method which does not involve abrupt clearing of all the four flip-flops is not simple. It requires a great deal of imagination and ingenuity on the part of the designer to build a decade counter which counts in a straight binary sequence from 0000 upto 1001 and back to 0000 using a four flip-flop chain. One such circuit is shown in Fig. 8.13. The counting sequence is the same as above.

Let us refer to the four binaries by their outputs—B_0, B_1, B_2 and B_3. Notice carefully the following in respect of two aspects namely, "driving the flip-flops" and "mode setting".



(a) ecade counter



(b) Waveforms

Fig. 8.12 Asynchronous decade counter

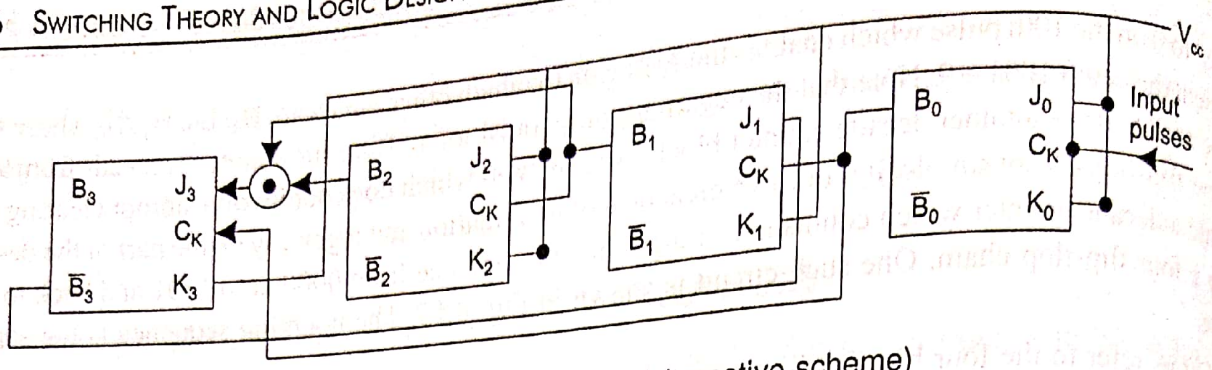


Fig. 8.13 Decade counter (alternative scheme)

1. Input pulses drive B_0 at its clock input.
2. B_0 drives B_1 and B_1 drives B_2
3. B_0 has to drive B_3 also for the simple reason that the 10th pulse causes $1 \rightarrow 0$ transition only at B_0 which has to cause similar transition at B_3 too. [Look at the counting sequence and the waveforms of Fig. 8.12(b)].
4. All $K_i = 1$ as they are tied to V_{cc}. Only $J_0 = J_2 = 1$ as they are tied to V_{cc}. Hence, B_0 and B_2 are set in toggle mode.

   $J_1 = B_3' = 1$ until the end of the 8th count pulse as there is need for keeping B_1 in toggling mode until then. (See the counting sequence.) After the 8th count, J_1 becomes 0 while $K_1 = 1$ which sets B_1 in a clearing mode.

   $J_3 = B_1 \cdot B_2$ which becomes 1 at the end of the 6th count pulse and remains at 1 till the 8th count pulse toggles B_0, B_1, B_2 and B_3 in that order by rippling through the binaries. Thereafter, B_3 would be in a clearing mode as $J_3 = 0$ and $K_3 = 1$.

   To sum up, the position on the 10th pulse is

   $J_0 = K_0 = 1 \Rightarrow B_0$ is in toggle mode $\Rightarrow 1 \rightarrow 0$

   $J_1 = 0, K_1 = 1 \Rightarrow B_1$ is in clearing mode $0 \rightarrow 0$

   $J_2 = K_2 = 1 \Rightarrow B_2$ is in toggle mode but there is no driving pulse.

   Hence $0 \rightarrow 0$

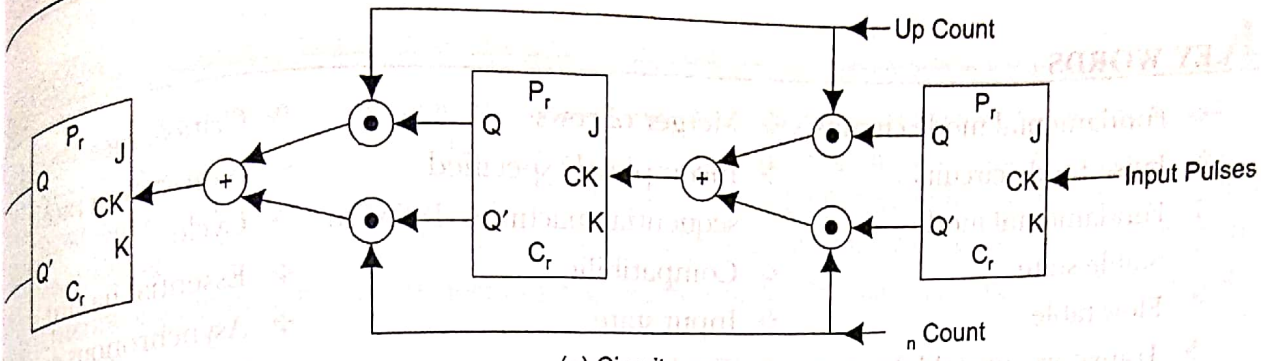
   $J_3 = 0, K_3 = 1 \Rightarrow B_3$ is in clearing mode $\Rightarrow 1 \rightarrow 0$

An inquisitive reader might wonder why not make $J_3 = B_2$ instead of $B_1 . B_2$. In that case, B_3 would be in a toggling mode at the end of 4th count pulse and the $1 \rightarrow 0$ transition of B_0 at the 6th count pulse would cause the toggling of B_3 which is wrong and hence prevented as $B_1 = 0$ before the 6th count pulse. For this reason, B_1 and B_2 jointly control the excitation J_3 of B_3.

Up-down Counter

Hitherto we used only upward counting. Downward counting can be obtained by simply using the Q′ output of the flip-flop to drive the next flip-flop. In this case, the positive-going edges (at Q) cause a change of state in the succeeding flip-flop. The counter has to be initialised to all 1s by pulsing the preset leads.

Look at the typical up count and down count for a binary counter with three flip-flops shown in Fig. 8.14. This can count up from 0 through 7 or count down from 7 to 0. The logic circuit between each pair of flip-flops enables either up count or down count.



(a) Circuit

Up-count			Down-count			Pulses
B_2	B_1	B_0	B_2	B_1	B_0	
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	2
0	1	1	1	0	0	3
1	0	0	0	1	1	4
1	0	1	0	1	0	5
1	1	0	0	0	1	6
1	1	1	0	0	0	7
0	0	0	1	1	1	8

(b) Counting sequence

Fig. 8.14 3-bit binary up-down counter

It is important for the reader to particularly note that the facility of up-down counting is simply obtained by having the option of choosing Q or Q′ output to drive the next flip-flop. This kind of arrangement is used only if the count advances in a straight binary progression. Initialisation can be done easily by using either all reset or all clear inputs of the flip-flops.

Can you think of obtaining a down count for the decade counter? Does skipping counts cause problems? Think it over

SUMMARY

The reader is exposed to common life situations requiring asynchronous sequential machines. The subject of design of asynchronous sequential machines is introduced through design examples. Design steps starting from primitive flow table ending with the hardware circuit has been illustrated. Cycles, races, critical and non-critical races have been illustrated. Procedures are given to select a valid assignment of internal states. Essential hazards and their identification and elimination are presented. Design of asynchronous pulse-input-counters has received due emphasis.