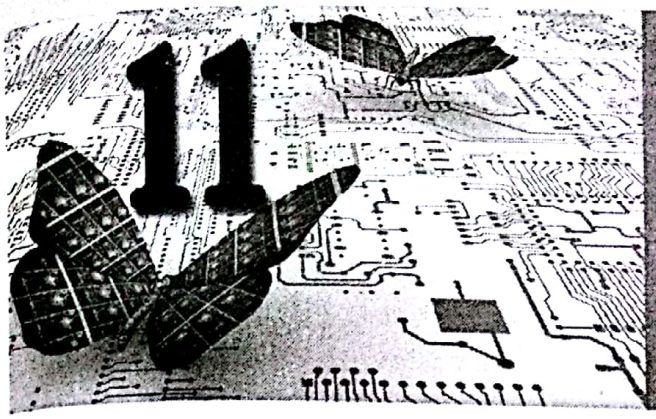


11 INTRODUCTION TO SEQUENTIAL CIRCUITS

11.1. INTRODUCTION

In combinational circuit the outputs at any instant of time are entirely dependent upon the inputs present at that time. There are many applications in which digital 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 such digital system is called sequential circuits. A block diagram of a sequential circuit is shown in Fig. 11.1. Sequential circuit is a combination of a combinational circuit and a memory elements connected in feedback path. The memory elements are devices capable of storing binary information within them. The binary information stored in the memory elements at any given time defines the state of the sequential circuit. The sequential circuit receives binary information from external inputs. Thus, a sequential circuit is specified by a time sequence of inputs, outputs and internal states.

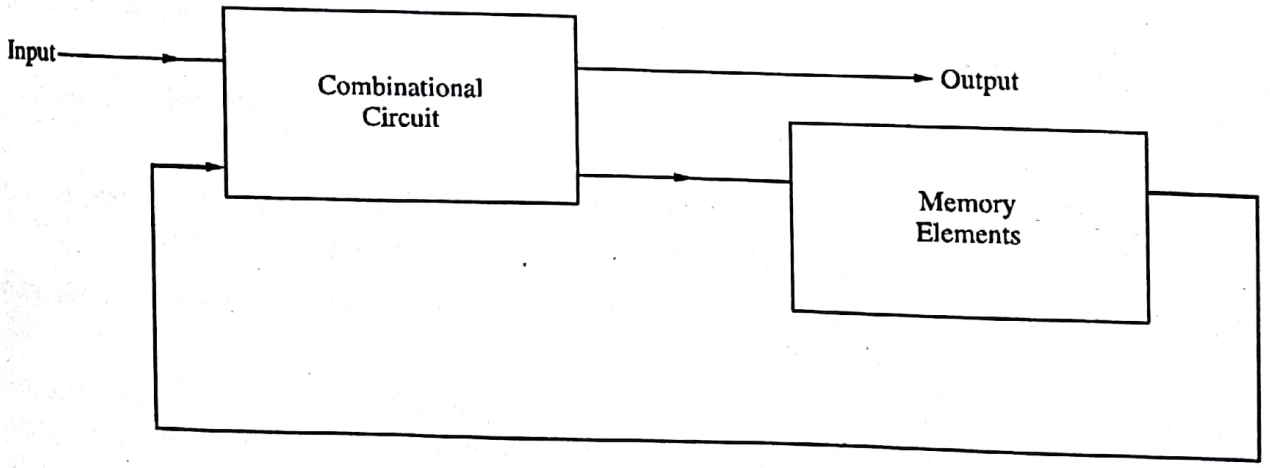


Fig. 11.1 Block diagram of a sequential circuit

There are two main types of sequential circuits.

1. **Asynchronous sequential circuit.** The sequential circuit whose output depends on the sequence in which the input changes is called asynchronous sequential circuits.

2. **Synchronous sequential circuit.** Sequential circuit whose output behavior depends on the input at discrete time is called synchronous sequential circuits. Synchronous sequential circuits that use clock pulses in the inputs of memory elements are called clocked sequential circuits.

Comparison between Synchronous & Asynchronous sequential circuit

	Synchronous sequential circuit	Asynchronous sequential circuit
1.	Output behavior depends on the input at discrete time.	Output depends on the sequence in which the input changes.
2.	Inputs and outputs are considered at discrete time instants.	Input and output signals are defined at every value of time.
3.	The time variable is discrete.	The time variable is continuous
4.	Memory element used in this are clocked flip-flops.	Memory element is unclocked flip-flop or time delay element.
5.	Easier to describe, analyze and design.	Asynchronous system are more difficult to describe, analyze and design than synchronous system.
6.	Low speed, operating speed depends on the clock pulse.	Because of absence of clock pulse, can operate faster than synchronous circuit.
7.	Change in state occurs in response to clock pulse.	State change occurs whenever input variable changes
8.	No timing problem in feedback path.	Timing problem involved in feedback path.
9.	More expensive.	Economical
10.	Complex circuitry.	It has a few components.
11.	Any number of inputs can change simultaneously (during the absence of clock).	Only one input is allowed to change at a time in the case of level inputs and only one pulse input is allowed to be present in the case of the pulse input.

11.2. BASIC DEFINITION

State. The binary information stored in the memory elements at any given time defines the state of the sequential circuits.

Present state. It shows the states of the sequential circuits before the occurrence of a clock pulse. The present state shows the state of sequential circuits at any given time t.

Next state. The next state designates the states of the sequential circuits after the application of a clock pulse *i.e.* the state at time '$t + 1$'.

Initial state. The state of sequential circuits can change only during a clock pulse transition. The behavior of a clocked sequential circuit is determined from inputs, outputs, and the state of its flip-flops. Initial state shows the state of the sequential circuit prior to the application of input sequence.

Final state. The state of the sequential circuit after the application of the input sequence is called final state.

Terminal state. A state is called a terminal state if either of the following is true :

(*i*) The corresponding vertex in the state diagram is a sink vertex, *i.e.* no outgoing arcs which emanate from it terminate in other vertices.

(*ii*) The corresponding vertex is a source *i.e.* no arcs which emanate from other vertices terminate in it. For example consider the state diagram shows in Fig. 11.2 (*a*).

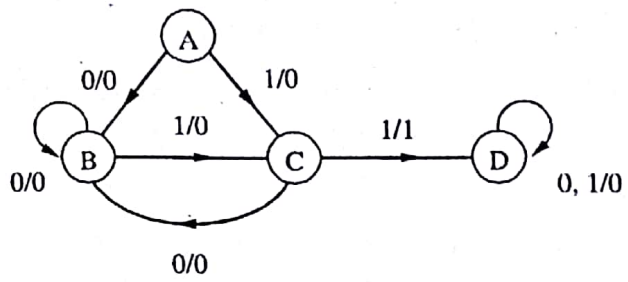


Fig. 11.2 (a)

State D is said to be terminal state because it is a sink vertex. Similarly state A is a source vertex.

State equation. It is also called a *next state equation* or *transition equation* or *characteristic equation.*

It is an algebraic expression that specifies the value of the next state as a function of the present state and inputs. The left side of equation has next state of flip-flop after one clock. The right side of the equation is a Boolean expression that specifies the present state and input condition that make the next state equal to 1 for *e.g.* the characteristic or state equation for the JK flip-flop is

$$Q(t+1) = J\,\overline{Q(t)} + \overline{K}\,Q(t)$$

where $Q(t+1)$ denotes the next state of the JK flip-flop one clock edge later.

State table. It denotes what will be the next state and output if the present state and inputs are known. It consists of three sections labeled present state, next state and output. Number of rows in state table shows the number of possible states of the circuits while, number of columns both in the next state and output sections, are equal to the number of possible input combinations.

Table 11.1 : State Table

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
A	A	B	0	0
B	E	D	0	1
C	A	D	0	0
D	C	B	0	1
E	C	A	0	1

State reduction. The reduction of the number of flip-flops in a sequential circuit while keeping the external input-output requirements unchanged is referred to as the state reduction of sequential circuit.

State assignment. State assignment procedures are concerned with methods for assigning coded binary values to the states in such a way as to reduce the cost of the combinational circuit that drives the flip-flops. Unused states are treated as don't care conditions during the design.

Table 11.2 shows the two possible binary state assignments for Table 11.1.

Table 11.2. State Table

State	Assignment 1 Binary	Assignment 2 gray code
A	000	000
B	001	001
C	010	011
D	011	010
E	100	110

Transition table. If all the states of state table are assigned by binary value then state table becomes transition table. Table 11.3 is the transition table with binary assignement 1 substituted for the letter symbols of the states in state table 11.1.

Table 11.3.

Present state	Next state		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
000	000	001	0	0
001	100	011	0	1
010	000	011	0	0
011	010	001	0	1
100	010	000	0	1

State diagram. The information available in a state table can also be represented in the form of graph which is called *state diagram* or *state graph*. In this diagram a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles. The directed lines are labeled with two binary numbers separated by a /. The input value that causes the state transition is labeled first and the number after the symbol / gives the value of the output during the present state. The state diagram of the state Table 11.1 is shown in Fig. 11.2 (*b*).

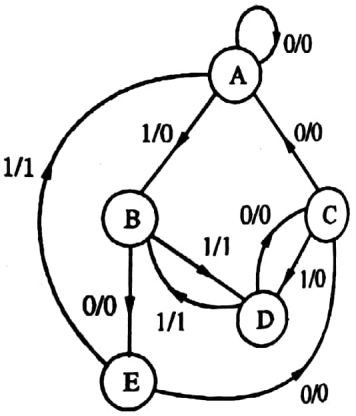


Fig. 11.2 (b).

11.3. ANALYSIS OF SEQUENTIAL CIRCUITS

To analyse the sequential circuits from logic diagram we go through following steps.

1. Determine the flip-flops inputs equations and output equations from the given sequential circuit.
2. Obtain the next state equation for each flip-flop from its input equations.
3. Plot a next-state and output k-maps for each flip-flop.
4. Combine these maps to form the state table.
5. After eliminating the redundant states, develop a simplified state table.
6. Make a state assignment and document the same in a state diagram.

11.4. FINITE STATE MACHINE

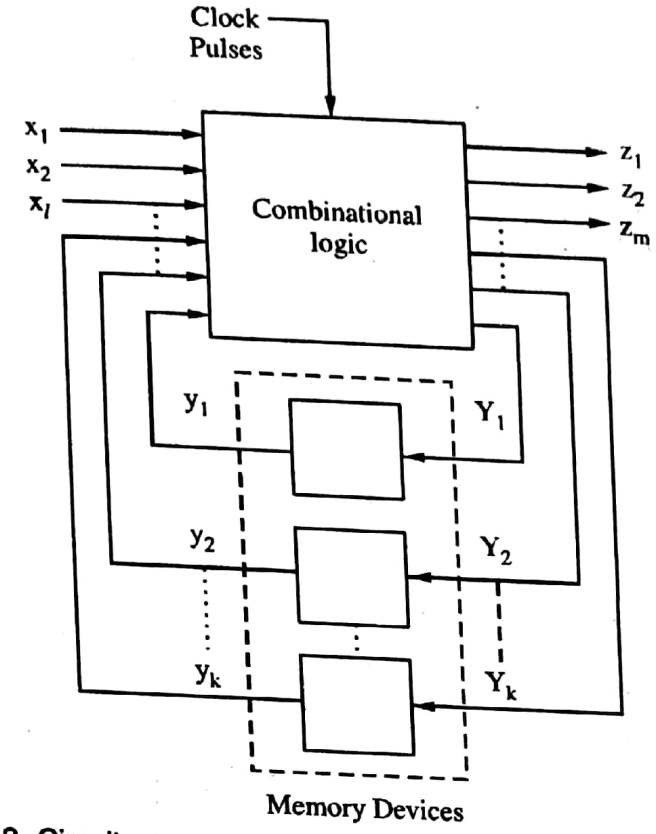


Fig. 11.3. Circuit representation of a synchronous sequential machine.

The behavior of a finite-state machine is described as a sequence of events that occur at discrete instants, designated $t = 1, 2, 3$, etc. Suppose that a machine M has input variable set $x = \{x_1, x_2, \ldots x_l\}$ and output variable set $z = \{z_1, z_2, \ldots, z_m\}$ each element of set z_i may have value 0 or 1. Output of each memory element is called state variable.

Set of state variable $y = \{y_1, y_2, \ldots y_k\}$.

Every finite-state machine contains a finite number of memory devices, which store the information regarding the past input history. At any time, value of all state variable is called present state then set of present state

$$Y = \{Y_1, Y_2, \ldots Y_k\}$$

11.5. MEALY MACHINE AND MOORE MACHINE

In a sequential circuit the output and the next state depends on the present state of the circuit and the external input. Two machine or models are used to represent these circuits. These are called Moore model or Moore machine and Mealy model or Mealy machine.

Mealy machine. In the Mealy machine, the output is a function of both the present state and input.

The block diagram of Mealy machine is shown in Fig. 11.4 (a).

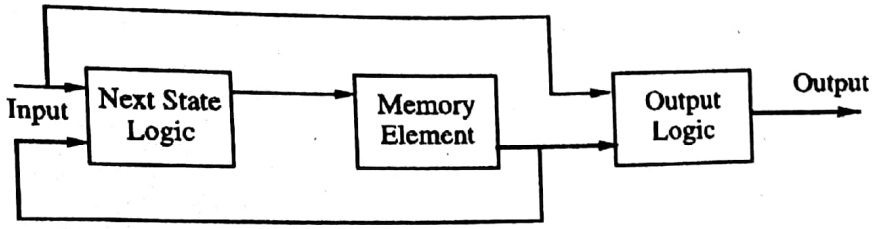


Fig. 11.4 (a) Mealy machine.

Moore machine. In the Moore machine, the output is a function of the present state only. The block diagram of Moore machine is shown in Fig. 11.4 (*b*).

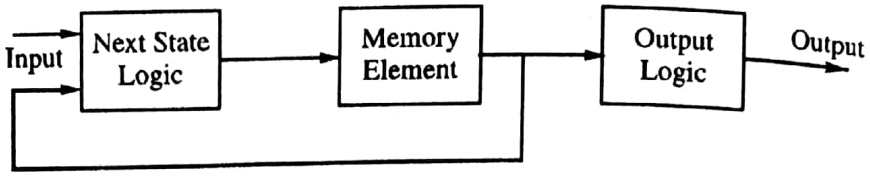
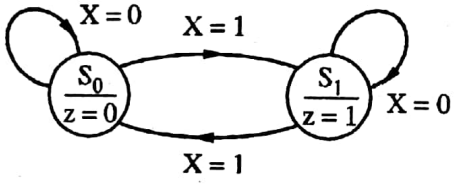
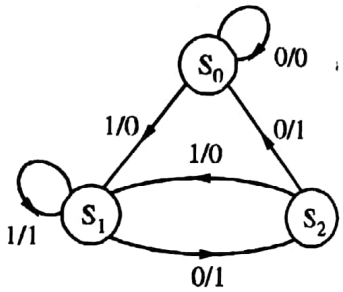
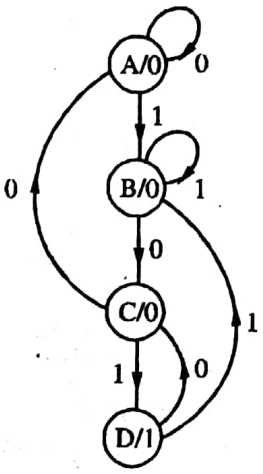
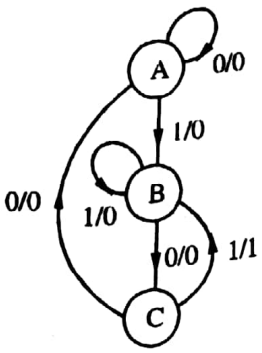


Fig. 11.4 (b) Moore machine.

Comparison between Mealy machine and Moore machine.

	Moore machine	Mealy machine
1.	The output is a function of the present state only.	The output is a function of both the present state and input.
2.	The state graph for a Moore machine has the output associated with the state. for *e.g.*	The state graph for a Mealy machine has the output associated with the arrow going between states.
	 Fig. 11.5 (a)	 Fig. 11.5 (b)
3.	In Moore machine one state is associated with only one output.	For *e.g.* where 0/1 or 1/1 or 1/0 or 0/0 represents *x*/*z*. In Mealy machine one state may have more than one output.
4.	During analysis output will only change when the state change.	Output may change either when input changes or when the flip-flop change state.
5.	Number of states require for implementation is generally more for *e.g.* state graph for sequence detector (101) is	It require less number of states to implement the same circuit.
	 Fig. 11.5 (c)	 Fig. 11.5 (d)

	Moore machine	Mealy machine
6.	Moore circuit is less flexible for design.	Designer has more flexibility.
7.	Output is observed after the entire input sequence has been applied.	Output is observed after applying each input.
8.	No false output appear because output is not a function of input. Changing the inputs at same time the state change occurs will eliminate false outputs.	The outputs may have momentary false values when the state has changed to its next value but the input has not yet changed.
9.	In state table, the output states are written in a seperate column. For *e.g.*	In this model there is no seperate column for outputs. Instead the ouptuts are written alongwith states. For *e.g.*
10.	Block diagram Refer Fig. 11.4 (*b*).	Block diagram Refer Fig. 11.4 (*a*).

Table 11.4 (*a*)

Present state	Next state		Output
	$x=0$	$x=1$	z
A	A	C	0
B	B	A	0
C	C	D	0
D	D	A	1

Table 11.4 (*b*)

Present state	Next state, output	
	$x=0$	$x=1$
A	A, 0	C, 1
B	B, 0	D, 1
C	C, 0	A, 0
D	D, 0	A, 0

11.6. STATE REDUCTION

The complexity of the digital logic that implement a sequential machine is directly related to the complexity of the state diagram or state table from which the function is implemented. In general, reducing the number of states in a table will reduce the amount of logic required, and the number of flip-flops may also be reduced. For example if a state table with 9 states ($S_0 \ldots S_8$) is reduced to 8 states or less than 9 states, then the number of flip-flops required is also reduced from 4 to 3, with a possible corresponding reduction in the amount of input logic for the sequential network. The number of states needed to realize a sequential machine directly affects circuit cost, operating speed and design complexity. Therefore, the reduction of state table is desirable.

It is important to find the good state assignment to minimize the logic required to realize the machine. But the problem of finding a good state assignment which leads to an economical network is a difficult one, because the best assignment for one type of flip-flop circuit is not the best for another type.

11.6.1. Basic Definition

Adjacent states. Two states are adjacent if they differ in only one variable. For example the following assignment is use for the states of flip-flops A and B. $S_0 = 00$, $S_1 = 10$, $S_2 = 11$ then (S_0, S_1) and (S_1, S_2) are adjacent states while (S_0, S_2) is non adjacent states.

Equivalent states. Two states, S_i and S_j are said to be equivalent if and only if, for every input sequence, the sequential circuit produces the same output sequence regardless of whether S_i or S_j is the starting state. The important properties of equivalent state are :

(*i*) For every possible input X sequence, the outputs are the same and the next states are equivalent.

(*ii*) If $S_i = S_j$ and $S_j = S_k$ then $S_i = S_k$

Redundant state. When two states are equivalent, one of them can be removed without altering the input-output relation and therefore redundant. The state which we have removed is known as redundant state. Consider the state table

Table 11.5.

Present state	Next state, output	
	$x = 0$	$x = 1$
S_0	S_0, 0	S_1, 0
S_1	S_2, 0	S_3, 0
S_2	S_0, 0	S_3, 1
S_3	S_4, 0	S_4, 1
S_4	S_0, 0	S_3, 1

Observe the state S_2 and S_4. If inputs are same say '1' it sends the machine to same state *i.e.* S_3 and gives same output. So, S_2 and S_4 are equivalent state. One of them can be removed from state table. S_4 is redundant state because its function can be accomplished by S_2. New state table after deleting the row of S_4 and replacing it with its equivalent state S_2 is

Table 11.6.

Present state	Next state, output	
	$x = 0$	$x = 1$
S_0	S_0, 0	S_1, 0
S_1	S_2, 0	S_3, 0
S_2	S_0, 0	S_3, 1
S_3	S_2, 0	S_2, 1

Distinguishable states. Two states, S_1 and S_2, are distinguishable if and only if there exists at least one finite input sequence which, when applied to machine, causes different output sequences, depending on whether S_1 or S_2 is the initial state. The sequence which distinguishes these states is called a distinguishing sequence of the pair (S_1, S_2). If there exists for pair (S_1, S_2) a minimum distinguishing sequence of length k, the states in (S_1, S_2) are said to be k-distinguishable. As an example consider the state Table 11.6. State S_1 and S_2 are 1-distinguishable because if the machine is in state S_1 and input 1 occurs, it gives output 0 whereas if it is in state S_2 and 1 occurs, machine gives output 1. On the other hand, the pair (S_0, S_1) is 2-distinguishable because when $x = 11$, the output sequences corresponding to initial states S_0 and S_1 are 00 and 01 respectively. States that are not k-distinguishable are called k-equivalent states. For example the pair (S_0, S_1) are 1-equivalent.

Compatible states. Two state S_1 and S_2 of incompletely specified machine are compatible if and only if, for every input sequence that effects the two states, the same output sequence occurs when the incompletely specified outputs are specified and their next states are either the same or also compatible, regardless of which state is the initial state. A compatible state is very similar to an equivalent state except for the introduction of incompletely specified outputs. Consider the state table, shown in Table 11.7.

Table 11.7.

Present state	Next state, output	
	$x=0$	$x=1$
S_0	S_2, 1	S_4, –
S_1	S_2, –	S_4, 1
S_2	S_1, 0	S_0, 1
S_3	S_3, 0	S_4, 1
S_4	S_3, 1	S_0, 0

The following observation can be made from the Table 11.7. States S_0 and S_1 are compatible state because for each possible input, they have the same output where ever specified and their next states are same.

State Reduction. By eliminating redundant states the machine can accomplish its task without altering the input-output relations, with less logic and usually in less time. The process of eliminating redundant states is called state reduction.

Minimal or Reduced Machine. The machine which contains no equivalent states is called the minimal, or reduced machine.

Strongly Connected Machine. If for every pair of states S_i, S_j of a machine M there exists an input sequence which takes M from S_i to S_j, then machine M is said to be strongly connected. In other words, any nontrivial machine which has terminal states is not strongly connected.

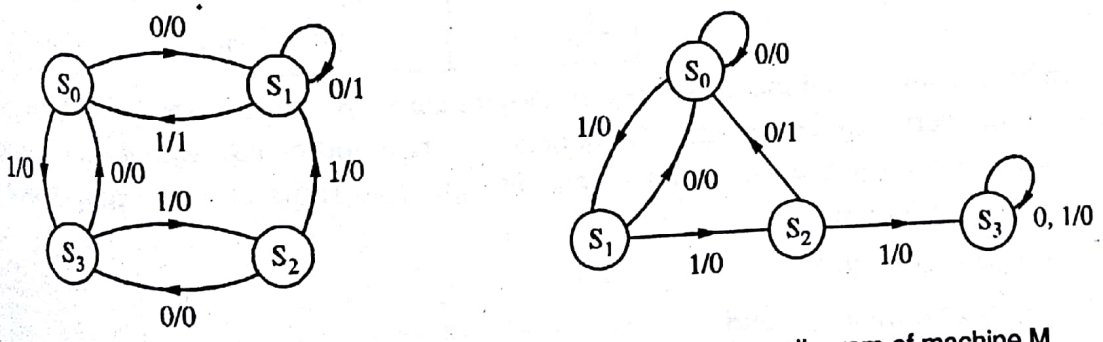


Fig. 11.6.

Fig. 11.6 shows the state diagram of strongly connected machine M_1 and the state diagram of machine M_2 which is not strongly connected. The machine M_2 is not strongly connected because no input sequence exists which can take machine M_2 out of state S_3 and thus state S_3 is said to be a terminal state. Machine M_1 is strongly connected because there exist atleast one direct or indirect path between every pair of states of a machine M_1.

Isomorphic Machine. If one machine can be obtained from the other by relabeling its states, they are said to be isomorphic to each other. The necessary and sufficient condition for two machines to be isomorphic to each other is that their state diagrams be idential, except for the labeling of their vertices. For example machines, M_1 and M_2 are isomorphic machine.

Table 11.8 (*a*) Machine M_1

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

Table 11.8 (*b*) Machine M_2

Present state	Next state	
	$x=0$	$x=1$
S_0	$S_1, 0$	$S_2, 0$
S_1	$S_3, 1$	$S_0, 0$
S_2	$S_3, 0$	$S_2, 0$
S_3	$S_1, 0$	$S_2, 1$

Incompletely Specified Machines. When a sequential machine is used as part of a larger digital system, it frequently happens that certain sequences will never occur as inputs to the sequential machine and consequently the corresponding transition and its associated output may be left unspecified. For some combinations of states and inputs the output values may not be critical and thus are left unspecified. Such restrictions lead to unspecified next states or outputs in the state table. Such machines are said to be incompletely specified. For example the state table for incompletely specified machine is shown in Table 11.9.

Table 11.9.

Present state	Next state	
	$x=0$	$x=1$
S_0	–, –	S_1, –
S_1	–, 0	S_2, 0
S_2	S_0, 1	S_1, 0
S_3	S_0, –	–, –

Equivalent Machine or Machine Equivalence. Two machines M_1 and M_2 are said to be equivalent if and only if for every state in machine M_1, there is a corresponding equivalent state in machine M_2 and vice-versa. For example machines M_1 and M_2 (given in Table 11.8) are equivalent if–

A is equivalent to S_0,

B is equivalent to S_1,

C is equivalent to S_2, and

D is equivalent to S_3

***X*-successor of S_i.** If an input sequence X takes a machine from state S_i to state S_j, then S_j is said to be the X-successor of S_i. For example, in Table 11.6, S_1 is the 1-successor of S_0 where as S_3 is the 101-successor of S_0.

11.7. MINIMIZATION OF THE STATE TABLE OF COMPLETELY SPECIFIED MACHINE

When the initial state diagram is made from a verbal problem statement it is easy to introduce redundant states. Sometimes these introduce redundant states are obvious and can be eliminated with trial and error methods or row matching. Various procedure used for minimization of the state table of completely specified machine are

(*i*) Partition method or equivalence-class method
(*ii*) Implication table method or Merger table method.

If desired, row matching can be used to partially reduce the state table before going for minimization.

11.7.1. Partition Method or Moore Reduction Procedure. A procedure for state classification in order to determine n-equivalence are obtained in several steps, begining from the classes of 1-equivalent states, following with the classes of 2-equivalent states, and so on. Consequently the minimization procedure using partition method indicates how to obtain P_1, how to determine partition P_{i+1} from P_i, and when to stop. Rules for state minimization are

1. The first partition P_0 corresponding to 0-distinguishability, and it defines our initial "ignorance" regarding the response of the various states, prior to the application of any input.
2. **Obtaining P_1.** The classes of 1-equivalent states are obtained directly from the state table by placing those state having the same outputs, under all inputs, in the same block.
3. The next step is to obtain the P_{i+1} partition from P_i partition. To obtain P_{i+1} it is only possible to break down classes from P_i but not to recombine states that are in different classes of P_i. In general, the P_{i+1} partition is obtained from P_i by placing in the same block of P_{i+1} those state which are in the same block of P_i and whose X_i-successors for every possible X_i are also in a common block of P_i.
4. The procedure stops when partition P_{i+1} is the same as partition P_i.
5. Write the reduced table.

11.7.2. Implication table or Merger table. Implication table provide a graphic method of identifying redundant states. The implication table uses a grid array to list the possible state equivalencies for different inputs. This table has a square for every possible pair of states. The intersection of the horizontal and vertical state spaces indicates a possible state equivalency. Implication table for 6 states machines is illustrated in Fig. 11.7.

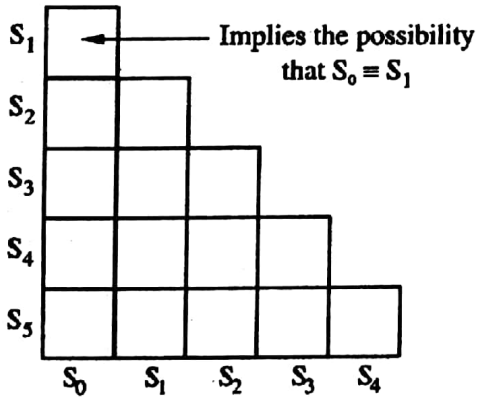


Fig. 11.7.

The implication chart squares are filled in as follows :

1. Construct a chart which contains a square for each pair of states.
2. Place an X in the squares where the outputs are different. These states can not be equivalent.
3. If the outputs and next states are the same place a check (✓) in square $i-j$ to indicate that $S_i \equiv S_j$.
4. Consider the remaining squares. We need look at only the state equivalency pair because the outputs were taken care of in step 2.
5. If the next states of S_i and S_j are S_m and S_n for some input x, then $S_i \equiv S_j$ iff (if and only if) $S_m \equiv S_n$. We write the state-equivalency pair S_m, S_n in the S_i, S_j square.

6. If square $i-j$ contains the implied pair $S_m \equiv S_n$, and square $m-n$ contains an X, then $S_i \not\equiv S_j$ and an X should be placed in square $i-j$.
7. Test the equivalencies of each square by testing the implied equivalent state pairs written in each square until no more X's are added.
8. For each square $i-j$ which does not contain an X, $S_i \equiv S_j$.

11.8. STATE ASSIGNMENT

After the number of states in a state table has been reduced, the next step in realizing the table is to assign unique binary codes to the state variables. The cost of the logic required to realize a machine is strongly dependent on the way this state assignment is made. For example, consider the state machine of Table 11.10. In Table 11.11, we will realize the circuit using two state assignments to illustrate the effect the assignments has on the final logic.

Table 11.10.

Present state	Next state, output	
	$x=0$	$x=1$
S_1	$S_1, 0$	$S_2, 0$
S_2	$S_1, 0$	$S_3, 0$
S_3	$S_4, 0$	$S_2, 0$
S_4	$S_1, 1$	$S_2, 0$

The excitation equations using state assignment 1 are

$$D_2 = Q_2 Q_1' x' + Q_2' Q_1 x$$
$$D_1 = Q_2 Q_1' + Q_2 x + Q_1' x$$

The excitation equations using state assignment 2 are

$$D_2 = Q_2 Q_1 x' + Q_2' Q_1 x$$
$$D_1 = x$$

We can easily see that the equations using state assignment 2 require fewer gates than assignment 1. This example illustrates the effect of state assignment for realizing a state table.

Table 11.11.

Present state		Next state		D_2		D_1		Output, z	
		$x=0$	$x=1$	$x=0$	$x=1$	$x=0$	$x=1$	$x=0$	$x=1$
State assignment 1									
	Q_2Q_1								
S_1	00	00	01	0	0	0	1	0	0
S_2	01	00	10	0	1	0	0	0	0
S_3	10	11	01	1	0	1	1	0	0
S_4	11	00	01	0	0	0	1	1	0
State assignment 2									
S_1	00	00	01	0	0	0	1	0	0
S_2	01	00	11	0	1	0	1	0	0
S_3	11	10	01	1	0	0	1	0	0
S_4	10	00	01	0	0	0	1	1	0

Two state assignments are equivalent if one can be derived from the other by permuting and complementing columns. Two state assignments which are not equivalent are said to be distinct. For example, state assignments *a*, *b*, and *c* are equivalent to each other.

Table 11.12.

Assignments			Present state
a	*b*	*c*	
00	00	11	A
01	10	10	B
10	01	01	C

11.8.1. State Assignments Permutations. The number of possible state assignments, *S*, for *N* states realized by *M* state variables is given by the relation

$$S = \frac{(2^M)!}{(2^M - N)!}$$

For example, consider a state machine with four states and two state variables. (In sequential circuit state variables are called secondary state variables to distinguish it from external primary input variables). The number of possible state assignments are

$$M = 2$$

$$N = 4$$

$$S = \frac{(2^2)!}{(2^2 - 4)!} = \frac{4!}{0!} = \frac{4!}{1} = 24$$

Thus, there are 24 possible state assignments for the four states, as shown in Table 11.13.

Table 11.13.

	Q_2Q_1																							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
S_1	00	00	00	00	00	00	01	01	01	01	01	01	10	10	10	10	10	10	11	11	11	11	11	11
S_2	01	01	10	10	11	11	10	10	11	11	00	00	11	11	00	00	01	01	00	00	01	01	10	10
S_3	10	11	01	11	01	10	11	00	10	00	10	11	00	01	11	01	11	00	10	10	00	10	00	01
S_4	11	10	11	01	10	01	00	11	00	10	11	10	01	00	01	11	00	11	01	01	10	00	01	00

If we interchange the columns of assignments 2 we get assignment 4, so assignment 2 and 4 have the same cost. Similarly, assignments 5 and 6 have the same cost. Interchanging rows, however, will generally change the cost of realization. Not all of the possible assignments need to be considered when making the state assignment.

The number of distinct state assignments are

$$S = \frac{(2^M - 1)!}{(2^M - N)!\,M!}$$

where N = The number of states, M = The number of state variables.

The number of distinct assignments increases very rapidly with the number of states as shown in Table 11.14.

Table 11.14.

No. of states	Minimum No. of state variables	No. of Distinct Assignment
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
9	4	10,810, 800
⋮		
20	5	143.14×10^{21}

To evaluate a state assignment's effectiveness in reducing the excitation logic needed for state transitions of all the states in a sequential machine would require a computer search to reach an optimum solution. Hand solution is feasible for 2, 3 or 4 states, computer solution is feasible for 5 through 8 states, but for 9 or more states it is not practical to try all assignments. For practical purpose, no manual "best" solution is likely, however, a "reasonable" solution can be found.

There are various ways to assign the binary values to the state variables.

1. **Binary assignment.** In this we can assign simple binary code to the states.
2. **Gray code assignment.** In this type of assignment we assign gray code to the states.
3. **One hot assignment.** This configuration uses as many bits as there are states in the circuit. At any given time, only one bit is equal to 1 while all others are kept at 0. This type of assignment uses one flip-flop per state.

11.8.2. Guidelines for State Assignment. The following guidelines are useful in making state assignments.

1. States having the same next state, for a given input value, should be given adjacent assignments.
2. The next states of a single present state should be given adjacent assignments.
3. Adjacent assignments should be given to states that have the same outputs.