COURSE CODE: SCS1203
COURSE NAME: FUNDAMENTALS OF DIGITAL SYSTEMS
CHAPTER NAME: BOOLEAN ALGEBRA AND LOGIC GATES

UNIT V

Circuits with Latches - Analysis procedure and Design Procedure - Reduction of state and Flow tables - Race - Free State Assignment

1. Circuits with Latches

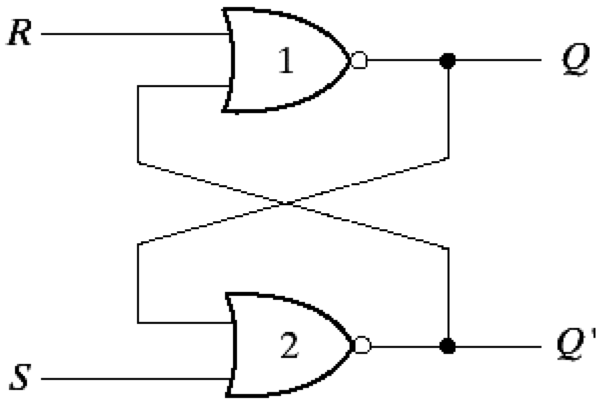
1.1 SR Latch:

The SR latch is a digital circuit with two inputs s and R and two cross-coupled NOR gates or two cross-coupled NAND gates.

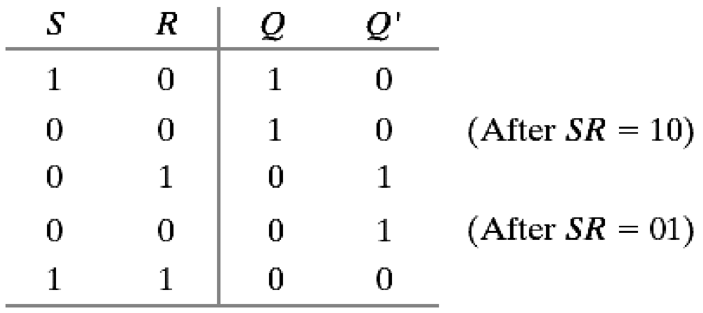
1.2 SR Latch with NOR:

The cross-co upled NOR gate circuit is shown in Fig. (a) This circuit and its trut h table are taken from Fig(b) In order to analyze the circ uit by the transition-table method, it is redrawn as fig(c). to see the feedback: path from the output of gate I to the input of gate 2. The output Q is equi valent to the excitation variable Yand the secondary variable y, The Boolean function for the output is

$Y = [(s + y)' + RI'] (S + y)R' = SR' + R'y$

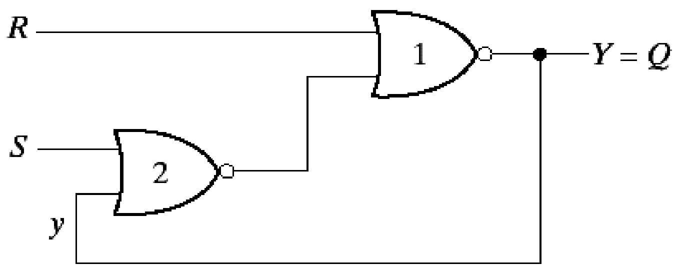


(a) Crossed-coupled circuit

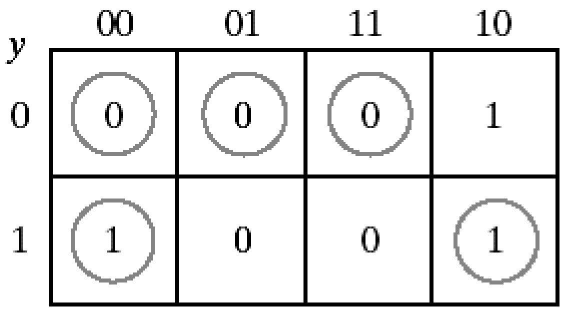


S	*R*	*Q*	*Q'*	
1	0	1	0	
0	0	1	0	(After *SR* = 10)
0	1	0	1	
0	0	0	1	(After *SR* = 01)
1	1	0	0	

(b) Truth table



(c) Circuit showing feedback



y	00	01	11	10
0	0	0	0	1
1	1	0	0	1

(d) Transition table

Plotting Y in the Fig. (d). we obtain the transitio n table for table circuit

The state with SR = 10 is a stable stale because Y = y = I; likewise. the state with SR = 0 1 is a sta - ble state. because Y = y = 0. With SR = 10. the output Q = Y = I and the latch is said to be set. Changing S to 0 leaves the circuit in the set state . With SR =01 . the output Q = Y = 0 and the latch is said to be reset. A change of R back to 0 leaves the circuit in the reset state. These conditions are also listed in the truth table. The circuit exhibits some difficulty when both S and R are equal to 1. From the truth table, we see that both Q and Q' are equal to 0. a condition that violates the requirement that these two outputs be the complement of each other. Moreover, from the transition table. we note that going from SR = 1 1 and SR = 00 produces an unpredictable result. If S goes to 0 first. the output remains at 0. but if R goes to 0 first. the output goes to 1. This condition can be expressed by the Boolean function SR = O. which states that the ANDing of S and R must always result in a 0.

Coming back to the excitation function. we note that when we OR the Boolean expression , SR' with SR. the result is the single variable S:

SR' + SR = S(R' + R) = S

we infer that SR' = S when SR = 0

Y = SR' + R'y

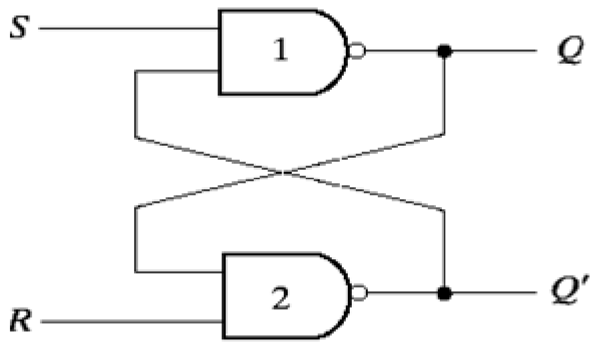
reduced excitation function Y=S + R'y when SR =0

1.3 SR Latch with NAND:

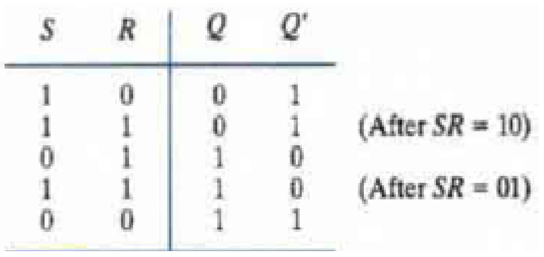
The NAND latch operates wi th both inputs normally at 1. unless the state of the latch has to be changed . The application of 0 to R causes the out put Q to goto 0, thus putting the latch in the reset state. After the R input returns to 1, a change of S to 0 causes a change to the set state. The condition to be avoided here is that both S and R not be 0 simultaneously. This condition is satisfied when S 'R' =0. The excitation function for the circuit in Fig.(c1) is

Y - [S(Ry)']' = S' + Ry

Comparing this with the excitation function of the NOR latch. we note that S has been replaced with S' and R' with R.Hence, the input variables for the NAND latch require the complemented values of those used in the NOR latch. For this reason. the NAND latch is sometimes referred to as an S'R' latch (or S- R latch).

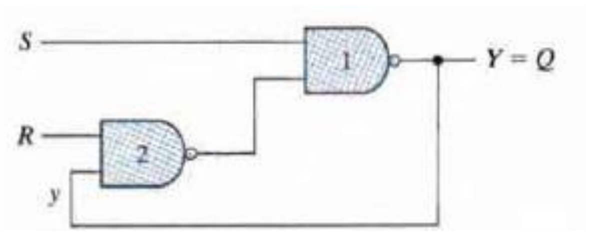


(a1) Cross coupled circuit

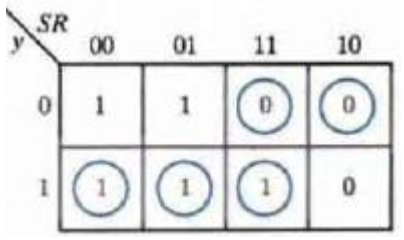


S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(After $SR = 10$)
0	1	1	0	
1	1	1	0	(After $SR = 01$)
0	0	1	1	

(b1)Truth Table



(c1)Circuit with showing feedback



y \ SR	00	01	11	10
0	1	1	$\textcircled{0}$	$\textcircled{0}$
1	$\textcircled{1}$	$\textcircled{1}$	$\textcircled{1}$	0

(d1)Transition table

2. Analysis procedure and Design Procedure

2.1 Analysis Procedure :

Procedure to analyze an asynchronous sequential circuits with SR latches:

1. Label each latch output with Yi and its external feedback path (if any) with yi
2. Derive the Boolean functions for each Si and Ri
3. Check whether SR=0 (NOR latch) or S’R’=0 (NAND latch) is satisfied
4. Evaluate Y=S+R’y (NOR latch) or Y=S’+Ry (NAND latch)
5. Construct the transition table for Y=Y1Y2…Yk
6. Circle all stable states where Y=y

Analysis Example:

Asynchronous sequential circuits can be constructed with the use of SR latches with or without external feedback paths . There is always a feedb ack loop within the latch itself.Th e analysis of a circ uit with latches will be demonstrated by means of a specific example from which it will be possible to generalize the proced ural steps necessary to analyze other similar circuits.

The following circuit figure 2 shown here has two SR latches with outputs Y1 and Y2.

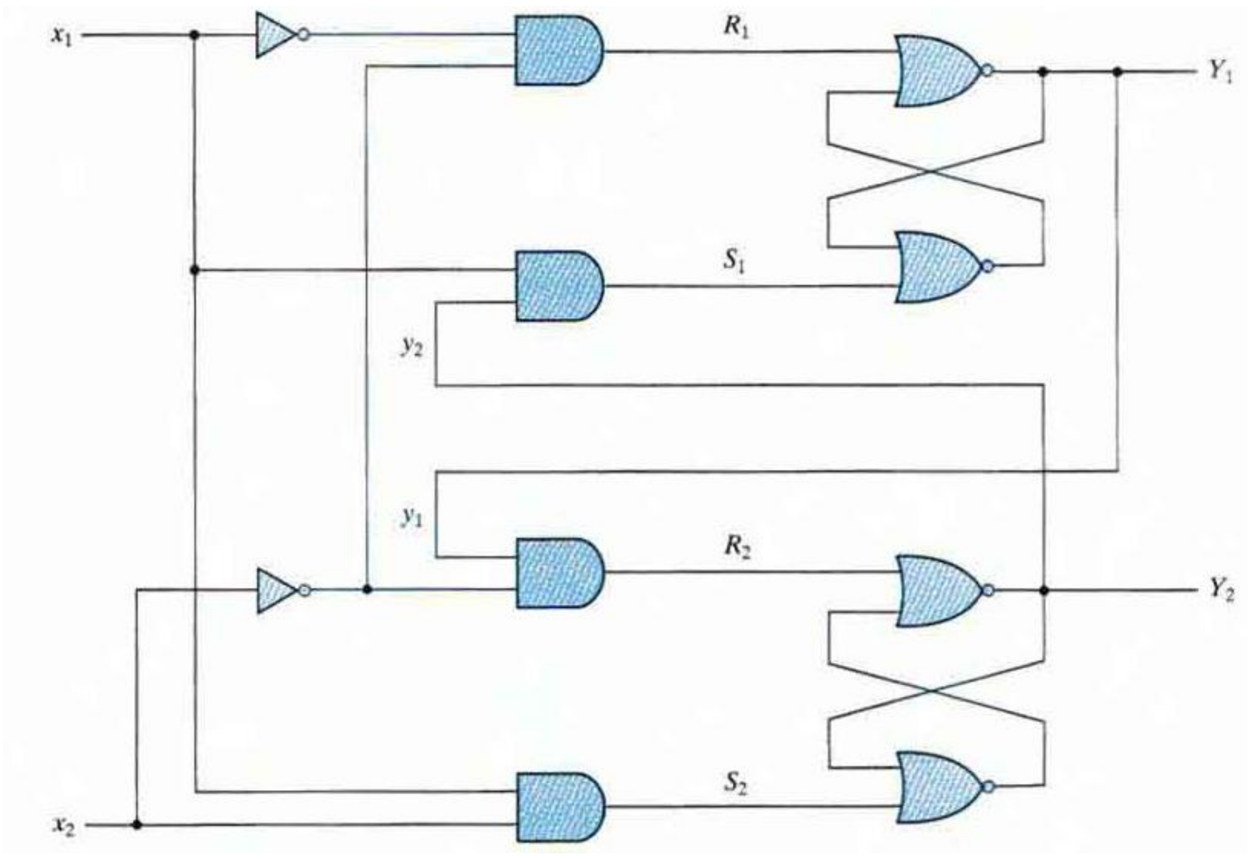


Fig 2. Example of a circuit with Srlatches

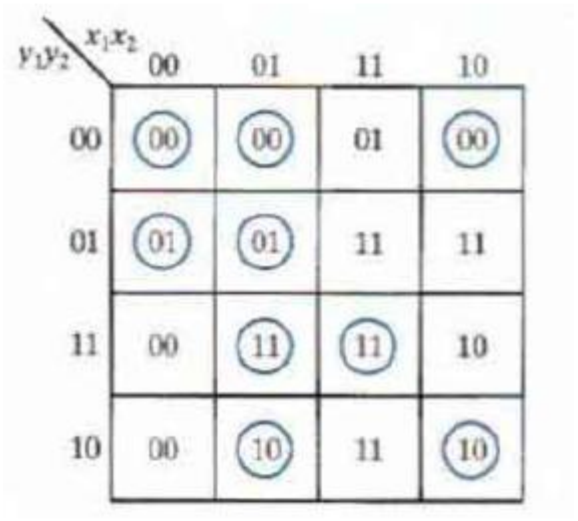
There are two input x1 and x2. and two external feedback loops giving rise to the secondary varia bles. y1 and y2,then firs t obtain the Boolean function for the S and R inputs in each latch

$$S_1 = x_1y_2 \qquad S_2 = x_1x_2$$
$$R_1 = x_1'x_2' \qquad R_2 = x_2'y_1$$

We then check whether the condition SR = 0 is satisfied to ensure proper operation of the circuit.

$$S_1R_1 = x_1y_2x_1'x_2' = 0$$
$$S_2R_2 = x_1x_2x_2'y_1 = 0$$

The transition table is given below in figure 3



y_1y_2 \ x_1x_2	00	01	11	10
00	00	00	01	00
01	01	01	11	11
11	00	11	11	10
10	00	10	11	10

Fig.3 Transition table

The result is 0 because x1x1’=x2x2’=0

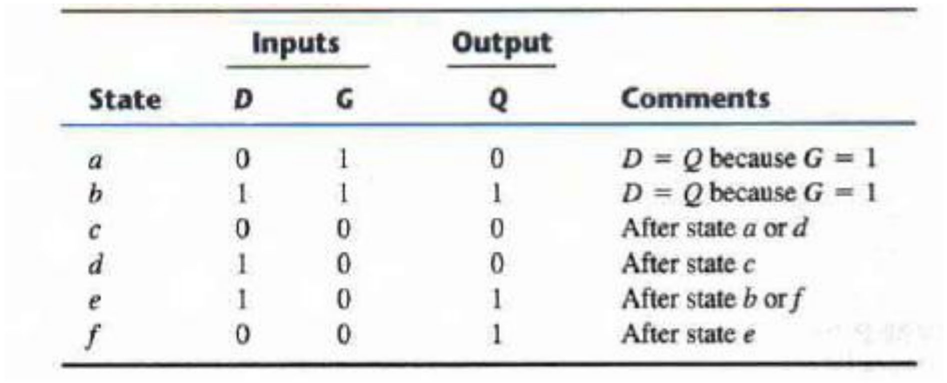
The next step is to derive the transition tab le of the circuit. Remember that thetransition table specifies the value of Y as a funct ion of y and x. The excitation functions are derived from the relation Y = S + R'y.

$$Y_1 = S_1 + R_1'y_1 = x_1y_2 + (x_1 + x_2)y_1 = x_1y_2 + x_1y_1 + x_2y_1$$
$$Y_2 = S_2 + R_2'y_2 = x_1x_2 + (x_2 + y_1')y_2 = x_1x_2 + x_2y_2 + y_1'y_2$$

2.2 Design Procedure:

1.Obtain a primitive flow table from the given de sign specifications. This is the most difficult pan of the design , because it is necessary to use intuition and experience to arrive at the correct interpretation of the problem specifications.
2.Reduce the flow table by merging rows in the primitive table.
3.Assign binary state variables to eac h row of the reduced flow table to obtain the transition table.
4.Assign outp ut value s to the dashes associ ated with the unstable states to obtain the output maps.
5.Simplify the Boolean functions of the excitation and output variables and draw the logic diagram.
Design Example:
Primitive table:
It is necessary to design a gated latch circuit with two inputs G (gate) and D (data) and one out-put Q. Binary information present at the D input is transferred to the Q output when G is equal to I. The Q output will follow the D input as long as G = 1. When G goes to 0, the information that was present at the D input at the time the transition occurred is retained at the Qoutput. The gated latch is a memory element that accepts the value of D when G = I and retains this value after G goes to 0. Once G = 0, a change in D does not change the value of the output Q. The Gated-latch total states is given in Table 1.

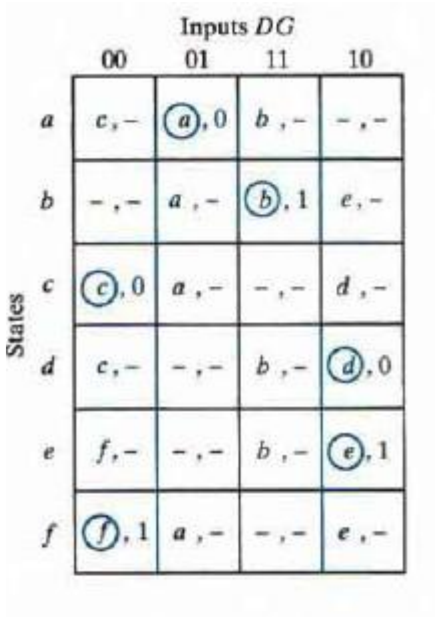
2.3 Derive transition table and flow table:
no simultaneous transitions of two variables
state a: after inputs DG=01
state b: after inputs DG=11
only one stable state in each row



State	Inputs D	Inputs G	Output Q	Comments
a	0	1	0	*D* = *Q* because *G* = 1
b	1	1	1	*D* = *Q* because *G* = 1
c	0	0	0	After state *a* or *d*
d	1	0	0	After state *c*
e	1	0	1	After state *b* or *f*
f	0	0	1	After state *e*

Table1 . Gated-latch total states

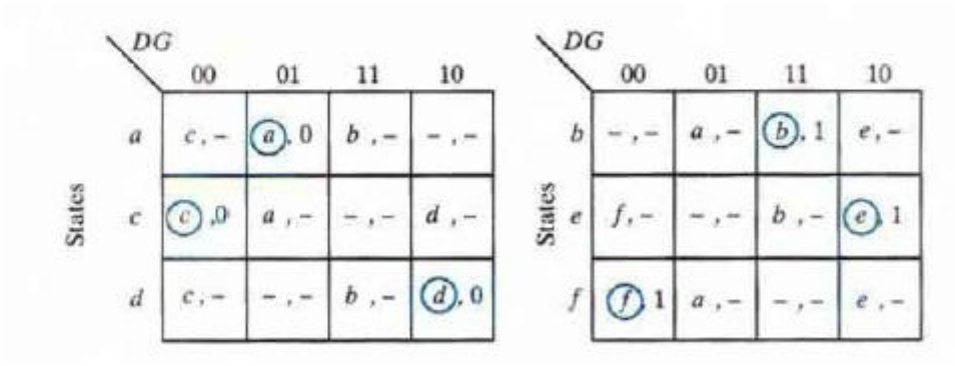
The primitive flow table for the gated latch is shown in Table 2 . It has one row for each state and one column for each input combination. First. we fill in one square in each row belonging to the stable state in that row. These entries are determined from Table 1. For example, State a is stable and the output is 0 when the input is 0 1. This infomation is entered into the flow table in the first row and second column . Similarly. the other five stable states together with their output are entered into the corresponding input columns.
we can enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state.For example, the first row in the flow table shows a stable state with an input of 0 1. Since only one input can change at any given time. it can change to 00 or 11. but not to 10 . Therefore. we enter two dashes in the 10 column of row a.. This will eventually result in a don' t-care condition for the next state and output in this square . Following the same procedure, we fill in a second square in each row of the primitive flow table.



States	Inputs *DG* 00	01	11	10
a	*c*, –	ⓐ, 0	*b* , –	– , –
b	– , –	*a* , –	ⓑ, 1	*e*, –
c	ⓒ, 0	*a* , –	– , –	*d* , –
d	*c*, –	– , –	*b* , –	ⓓ, 0
e	*f*, –	– , –	*b* , –	ⓔ, 1
f	ⓕ, 1	*a* , –	– , –	*e* , –

Table 2.Primitive Flow table

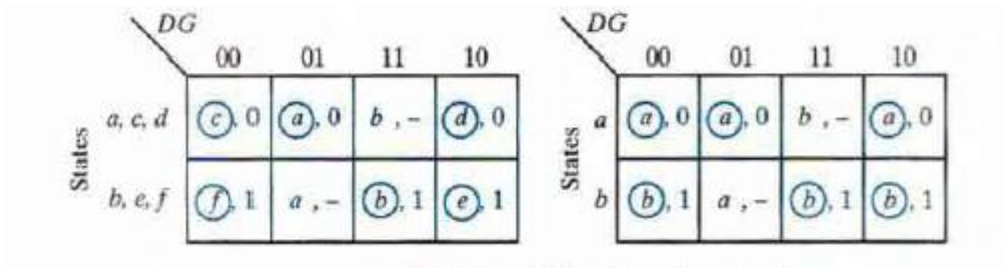
2.4 Reduction of the Primitive Flow Table:
 Two or more rows in the primitive flow table can be merged if there are nonconflicting states and outputs in each of columns.
– Primitive flow table is separated into two parts of three rows each in shown in fig 5.



States \ DG	00	01	11	10
a	c , –	ⓐ, 0	b , –	– , –
c	ⓒ, 0	a , –	– , –	d , –
d	c , –	– , –	b , –	ⓓ, 0

States \ DG	00	01	11	10
b	– , –	a , –	ⓑ, 1	e , –
e	f , –	– , –	b , –	ⓔ, 1
f	ⓕ, 1	a , –	– , –	e , –

Fig 5. State that are canditates for merging



States \ DG	00	01	11	10
a, c, d	ⓒ, 0	ⓐ, 0	b , –	ⓓ, 0
b, e, f	ⓕ, 1	a , –	ⓑ, 1	ⓔ, 1

States \ DG	00	01	11	10
a	ⓐ, 0	ⓐ, 0	b , –	ⓐ, 0
b	ⓑ, 1	a , –	ⓑ, 1	ⓑ, 1

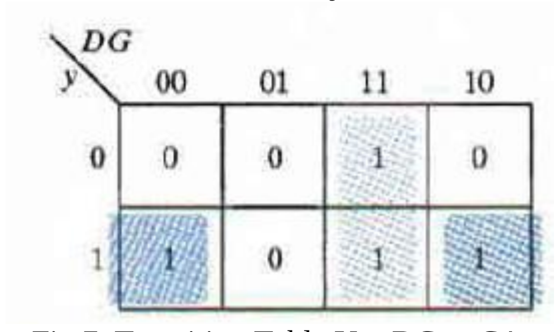
Fig 6 Reduced table(two alternatives)

Each part shows three stable states that can be merged because there are no conflicting entries in each o f the four columns. The first column shows state *c* in all the rows and 0 or a dash for the output. Since a dash represents a don' t-care condition, it can be associated with any state or output. The two dashes in the first column can be taken to be 0 output to make all three rows identical to a stable state *c* with a 0 output. The second column shows that the dashes can be assigned to correspond to a stable state *a* with a 0 output. Note that if a state is circled in one of the rows, it is also circled in the merged row. Similarly. the third column can be merged into an unstable state *b* with a don't-care output, and the fourth column can be merged into stable state *d* and a 0 output. Thus, the three rows *a, c* and *d* can be merged into one row with three stable states and one unstable stale, as shown in the first row of Fig.6.

2.5 Transition Table and Logic Diagram:

In order to obtain the circuit described by the reduced flow table, it is necessary to assign a distinct binary value to each state. This assignment converts the flow table into a transi-tion table. In the general case, a binary state assignment must be made to ensure that the circuit will be free of critical races. Assigning 0 to sta te *a* and 1 to state *b* in the reduced flow table of Fig. 6, we obtain the transition table of Fig.7 . The transition table is, in effect, a map for the excitation variable Y. The simplified Boolean function for *Y* is then ob-tai ned from the map as

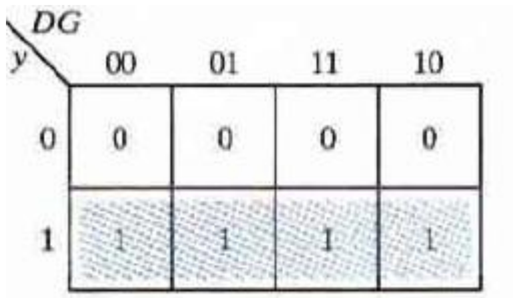
$$Y = DC \ + C\,'y$$



y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Fig 7. Transition Table Y = DC + C 'y

There are two don' t-care outputs in the final reduced flow table. If we assign values to the output as shown in Fig. 8, it is possible to make output Q identical to the map of the excitation function Y.Tthe logic diagram of the gated latch is as shown in Fig. 9



y \ DG	00	01	11	10
0	0	0	0	0
1	1	1	1	1

Fig 8. Output map for gated latch Q=Y

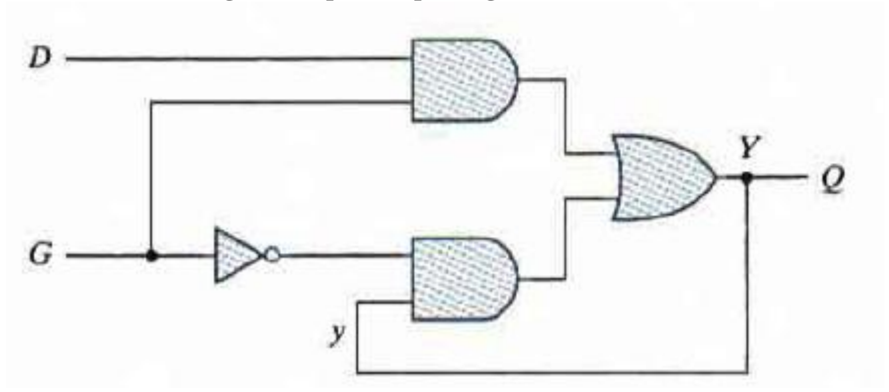
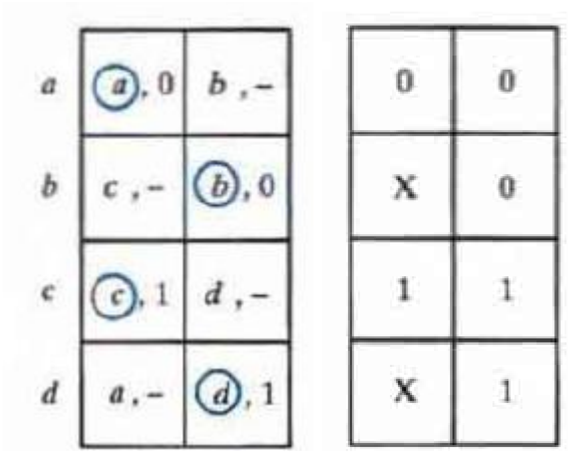


Fig 9. Logic diagram of the gated latch

2.6 Assign Outputs to Unstable States:

The stable states in a flow table have specific output values associated with them. The unstable states have unspecified output entries designated by a dash. The output values for the unstable states must be chosen so that no momentary false outputs occur when the circuit switches between stable states. The Flow table and
Output Assignment are shown in fig 10(a) and (b) respectively.
•the unstable states have unspecified output values
•no momentary false outputs occur when circuit switches between stable states
0→0 ==0 : assign 0 if the transient state between two 0 stable states
1→1 = 1 : assign 1 if the transient state between two 1 stable states
0→1, 1→0 = don't care: assign don't care if the transient state between two different stable states



a	ⓐ, 0	b , –
b	c , –	ⓑ, 0
c	ⓒ, 1	d , –
d	a , –	ⓓ, 1

0	0
X	0
1	1
X	1

Fig 10(a). Flow table (b)Output Assignment

3. Reduction of state and Flow tables

Sequential circuits.

The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

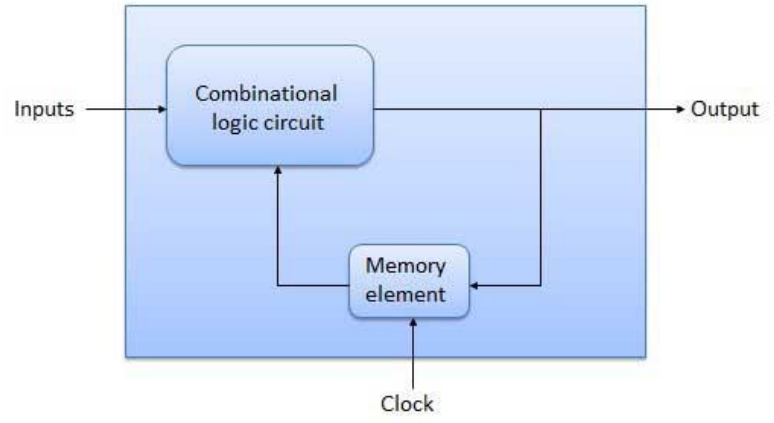


Figure 1: Representation of a Sequential circuit.

In this model the effect of all previous inputs on the outputs is represented by a state of the circuit. Thus, the output of the circuit at any time depends upon its current state and the input. These also determine the next state of the circuit. The relationship that exists among the inputs, outputs, present states and next states can be specified by either the **state table** or the **state diagram**.

3.1 State Table

The state table representation of a sequential circuit consists of three sections labelled present state, next state and output. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state shows the states of flip-flops after the clock pulse, and the output section lists the value of the output variables during the present state.

3.2 State Diagram

In addition to graphical symbols, tables or equations, flip-flops can also be represented graphically by a state diagram. In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines (or arcs) connecting the circles. An example of a state diagram is shown in Figure 2 below.

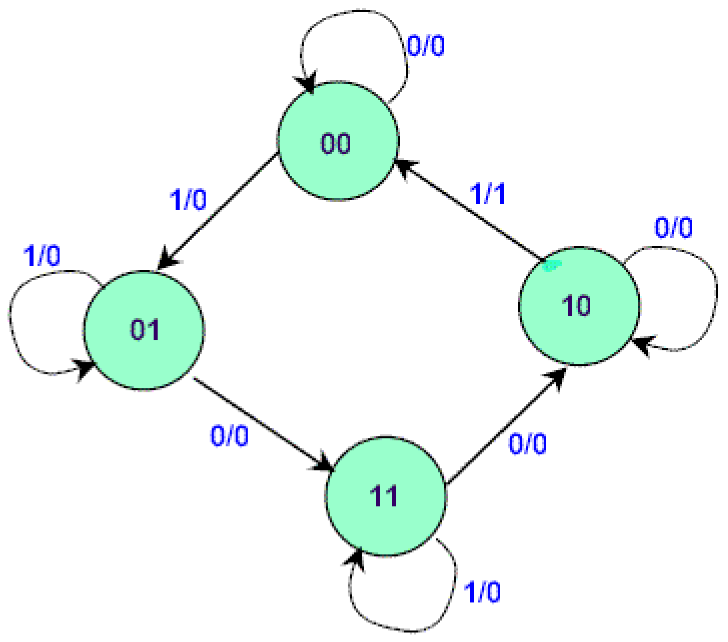


Figure 2: An example of a state diagram

The binary number inside each circle identifies the state the circle represents. The directed lines are labelled with two binary numbers separated by a slash (/). The input value that causes the state transition is labelled first. The number after the slash symbol / gives the value of the output. For example, the directed line from state 00 to 01 is labelled 1/0, meaning that, if the sequential circuit is in a present state and the input is 1, then the next state is 01 and the output is 0. If it is in a present state 00 and the input is 0, it will remain in that state. A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram provides exactly the same information as the state table and is obtained directly from the state table.

3.3 State Reduction:

The reduction of the number of flip-flops in a sequential circuit is referred to as the state reduction problem. State-reduction algorithms are concerned with procedures for reducing the number of states in a state table, while keeping the external input-output requirements unchanged. Since (N) flip-flops produce (2N) states, a reduction in the number of states may (or may not) result in a reduction in the number of flip-flops. An unpredictable effect in reducing the number of flip-flops is that sometimes the equivalent circuit (with fewer flip-flops) may require more combinational gates. We will illustrate the state reduction procedure with an example. We start with a sequential circuit whose specification is given in the state diagram shown in Figure 3. In this example, only the input-output sequences are important; the internal states are used merely to provide the required sequences. For this reason, the states marked inside the circles are denoted by letter symbols instead of their binary values. This is in constant to a binary counter, where the binary value sequence of the state themselves is taken as the outputs.

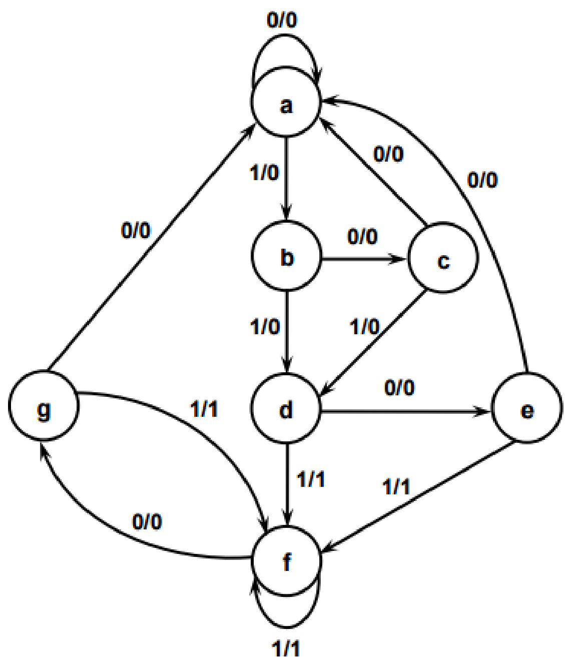


Figure 3: State diagram

There are an infinite number of input sequences that may be applied to the circuit; each results in a unique output sequence. As an example, consider the input sequence [01010110100] starting from the initial state (a). Each input of 0 or 1 produces an output of 0 or 1 and causes the circuit to go to the next state. the output and state sequence for the given input sequence as follows: With the circuit in initial state (a), an input of 0 produces an output of 0 and the circuit remains in state (a). With present state (a) and input of 1, the output is 0 and the next state is (b). With present state (b) and input of 0, the output is 0 and next state is (c). Continuing this process, we find the complete sequence to be as follows:

State	a	a	b	c	d	e	f	f	g	f	g	a
Input	0	1	0	1	0	1	1	0	1	0	0	
Output	0	0	0	0	0	1	1	0	1	0	0	

In each column, we have the present state, input value, and output value. The next state is written on top of the next column. It is important to realize that in this circuit, the states themselves are of secondary importance because we are interested only in output sequences caused by input sequences. Now let us assume that we have found a sequential circuit whose state diagram has less than seven states and we wish to compare it with the circuit whose state diagram is given by Figure 3. If identical input sequences are applied to the two circuits and identical outputs occur for all input sequences, then the two circuits are said to be equivalent (as far as the input-output is concerned) and one may be replaced by the other. The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships. We now proceed to reduce the number of states for this example. First, we need the state table; it is more convenient to apply procedures for state reduction using a table rather than a diagram. The state table of the circuit is listed in Table 1 and is obtained directly from the state diagram.

Table 1: State table

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

An algorithm for the state reduction of a completely specified state table is given here without proof:"Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state." When two states are equivalent, one of them can be removed without altering the input-output relationships. Now apply this algorithm to Table 1. Going through the state table, we look for two present states that go to the same next state and have the same output for both input combinations. States (g) and (e) are two such states: one of these states can be removed. The procedure of removing a state and replacing it by its equivalent is demonstrated in Table 2. The row with present state (g) is removed and state (g) is replaced by state (e) each time it occurs in the next-state columns.

Table 2: Reducing the State table

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

Table 3: Reduced State table

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

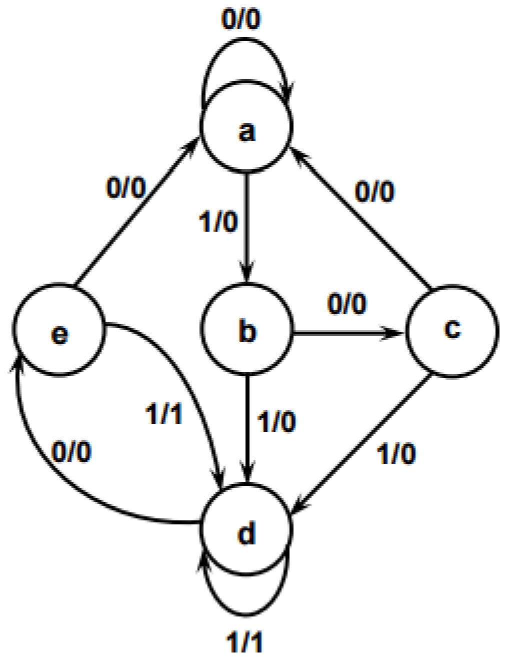


Figure 4: Reduced State diagram

Present state (f) now has next states (e and f) and outputs 0 and 1 for x=0 and x=1, respectively. The same next states and outputs appear in the row with present (d). Therefore, states (f and d) are equivalent and state (f) can be removed and replaced by (d). The final reduced table is shown in Table 3. The state diagram for the reduced table consists of only five states and is shown in Figure 4. This state diagram satisfies the original input-output specifications and will produce the required output sequence for any given input sequence. The following list derived from the state diagram of Figure 4 is for the input sequence used previously (note that the same output sequence results, although the state sequence is different):

State	a	a	b	c	d	e	d	d	e	d	e	a
Input	0	1	0	1	0	1	1	0	1	0	0	
Output	0	0	0	0	0	1	1	0	1	0	0	

Infact, this sequence is exactly the same as that obtained for Figure 3, if we replace (g by e and f by d). Checking each pair of states for possible equivalency can be done systematically by means of a procedure that employs an implication table. The implication table consists of squares, one for every suspected pair of possible equivalent states. By judicious use of the table, it is possible to determine all pairs of equivalent states in a state table. The use of the implication table for reducing the number of states in a state table is demonstrated in the next section. The sequential circuit of this example was reduced from seven to five state. In general, reducing the number of states in a state table may result in a circuit with less equipment. However, the fact that a state table has been reduced to fewer state doesn't guarantee a saving in the number of flip-flops or the number of gates.

Example 1: Consider a sequential circuit shown in Figure 5. It has one input x, one output Z and two state variables Q1Q2 (thus having four possible present states 00, 01, 10, 11).

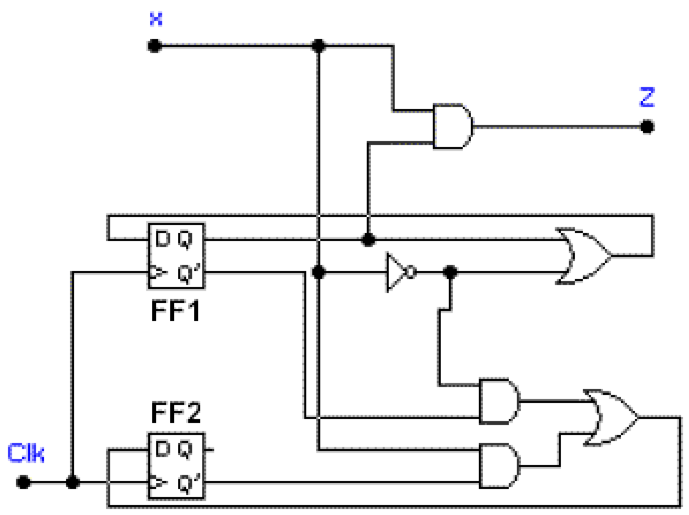


Figure 5: A sequential circuit

The behaviour of the circuit is determined by the following Boolean expressions:

Z = x*Q1

D1 = x' + Q1

D2 = x*Q2' + x'*Q1'

These equations can be used to form the state table. Suppose the present state (i.e. Q1Q2) = 00 and input x = 0. Under these conditions, we get Z = 0, D1 = 1, and D2 = 1. Thus the next state of the circuit D1D2 = 11, and this will be the present state after the clock pulse has been applied. The output of the circuit corresponding to the present state Q1Q2 = 00 and x = 1 is Z = 0. This data is entered into the state table as shown in Table 4.

Table 4: State table for the sequential circuit in Figure 5.

Present State	Next State		Output	
Q1Q2	x = 0	x = 1	x = 0	x = 1
0 0	1 1	0 1	0	0
0 1	1 1	0 0	0	0
1 0	1 0	1 1	0	1
1 1	1 0	1 0	0	1

The state diagram for the sequential circuit in Figure 5 is shown in Figure 6.

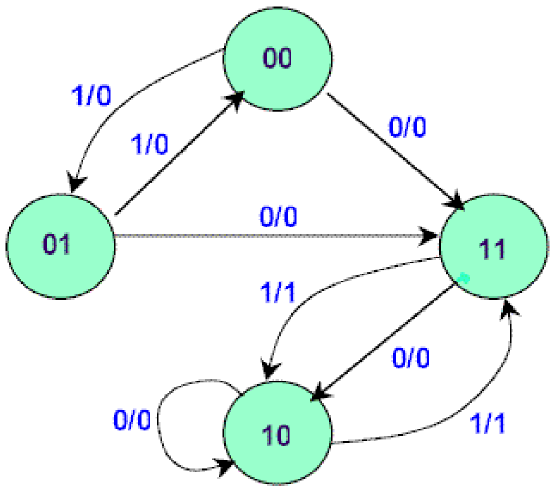


Figure 6: State Diagram of circuit in Figure 5.

3.4 Implication Table:

The state-reduction procedure for completely specified state tables is based on the algorithm that two states in a state table can be combined into one if they can be shown to be equivalent. Two states are equivalent if for each possible input, they give exactly the same output and go to the same next states or to equivalent next state. Consider for example, the state table shown in Table 5. The present states (a) and (b) have the same output for the same input. Their next states are (c and d) for x=0 and (b and a) for x=1. If we can show that the pair of states (c, d) are equivalent, then the pair of states (a, b) will also be equivalent because they will have the same or equivalent next states. When this relationship exists, we say that (a, b) imply (c, d). Similarly, from the last two rows of Table 5, we find that the pair of states (c, d) imply the pair of states (a, b). The characteristic of equivalent states is that if (a, b) imply (c, d) and (c, d) imply (a, b), then both pairs of states are equivalent; that is, (a and b) are equivalent as well as (c and d). As a consequence, the four rows of Table 5 can be reduced to two rows by combining (a and b) into one state and (c and d) into a second state.

The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically by means of an implication table. The implication table is a chart that consists of squares, one for every possible pair of states, that provide spaces for listing any possible implied states. By judicious use of the table, it is possible to determine all pairs of equivalent states. The state table of Table 6 will be used to illustrate this procedure. The implication table is shown in Figure 7. On the left side along the vertical are listed all the states defined in the state table except the first, and across the bottom horizontally are listed all the states expect the last. The result is a display of all possible combinations of two states with a square placed in the intersection of a row and a column where the two states can be tested for equivalence. Two states that are not equivalent are marked with a cross (x) in the corresponding square, whereas their equivalence recorded with a check mark (√). Some of the squares have entries of implied states that must be further investigated to determine whether they are equivalent or not. The step-by-step procedure of filling in the squares is as follows. First, we place a cross in any square corresponding to a pair of states whose outputs are not equal for every input. In this case, state (c) has a different output than any other state, so a cross is placed in the two squares of row (c) and the four squares of column (c). There are nine other squares in this category in the implication table.

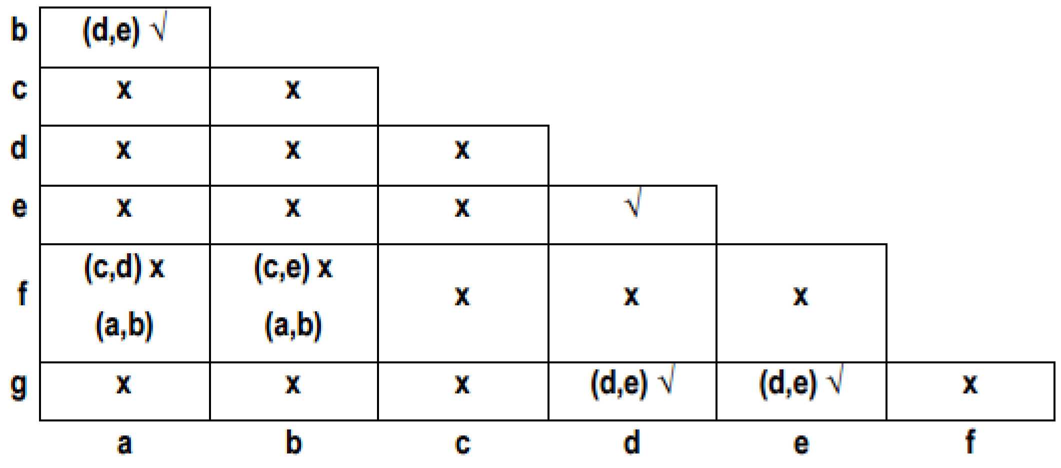
Table 5: State Table to Demonstrate Equivalent States.

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

Next, we enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. We do that starting from the top square in the left column and going down and then proceeding with the next column to the right. From the state table, we see that pair (a,b) imply (d,e), so (d,e) is recorded in the square defined by column (a and row b). We proceed in this manner until the entire table is completed. Note that states (d,e) are equivalent because they go to the same next state and have the some output. Therefore, a check mark is recorded in the square defined by column (d and row e), indicating that the two states are equivalent and independent of any implied pair. The next step is to make successive passes through the table to determine whether any additional squares should be marked with a cross. A square in the table is crossed out if it contains at least one implied pair that is not equivalent. For example, the square defined by (a) and (f) is marked with a cross next to (c,d) because the pair (c,d) defines a square that contains a cross. This procedure is repeated until no additional squares can be crossed out.
Finally, all the squares that have no crosses are recorded with check marks. These squares define pairs of equivalent states. In this example, the equivalent states are: (a,b) (d,e) (d,g) (e,g).

Table 6: State Table to be Reduced.

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	d	a	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



b	(d,e) √					
c	x	x				
d	x	x	x			
e	x	x	x	√		
f	(c,d) x (a,b)	(c,e) x (a,b)	x	x	x	
g	x	x	x	(d,e) √	(d,e) √	x
	a	b	c	d	e	f

Figure 7: Implication table.

We now combine pairs of states into larger groups of equivalent states. The last three pairs can be combined into a set of three equivalent states (d,e,g) because each one of the states in the group is equivalent to the other two. The final partition of the states consists of the equivalent states found from the implication table, together with all the remaining states in the state table that are not equivalent to any other state. (a,b) (c) (d,e,g) (f) This means that Table 6 can be reduced from seven states to four states, one for each member of the above partition. The reduced table is obtained by replacing state (b by a and states e and g by d).

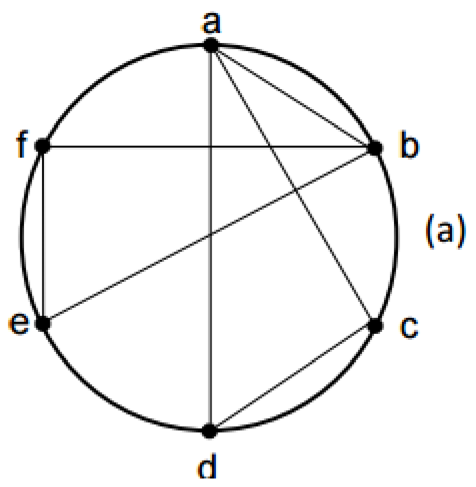
Table 7: Reduced state table

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	d	a	0	0
c	d	f	0	1
d	a	d	1	0
f	c	a	0	0

3.5 Merger Diagram:

Having found all the compatible pairs, the next step is to find larger sets of states that are compatible. The maximal compatible is a group of compatibles that contains all the possible combinations of compatible states. The maximal compatible can be obtained from a merger diagram, as shown in Figure 8. The merger diagram is a graph in which each state is represented by a dot placed along the circumference of a circle. Lines are drawn between any two corresponding dots that form a compatible pair. All possible compatibles can be obtained from the merger diagram by observing the geometrical patterns in which states are connected to each other. An isolated dot represents a state that is not compatible to any other state. A line represents a compatible pair. A triangle constitutes a compatible with three states. An nstate compatible is represented in the merger diagram by an n-state polygon with all its diagonals connected. The merger diagram of Figure 8 is obtained from the list of compatible pairs derived from the implication table. There are seven straight lines connecting the dots, one for each compatible pair. The lines from a geometrical pattern consisting of two triangles connecting (a, c, d) and (b, e, f) and a line (a, b). The maximal compatibles are: (a,b) (a,c,d) (b,e,f) Figure 8b shows the merger diagram of an 8-state. The geometrical patterns are a rectangle with its two diagonals connected to form the 4-state compatible (a, b, e, f), a triangle (b, c, h), a line (c, d), and a single state (g) that is not compatible to any other state. The maximal compatibles are:

(a,b,e,f) (b,c,h) (c,d) (g)



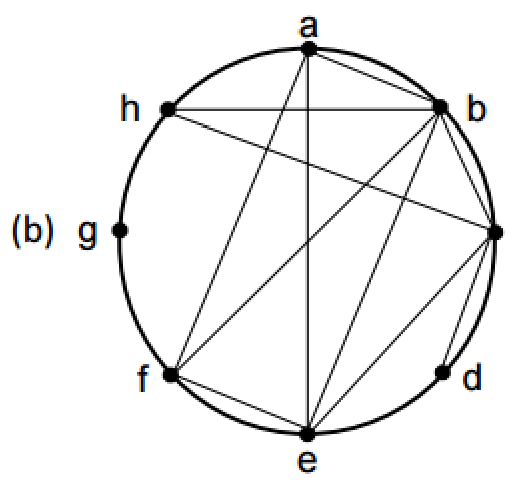


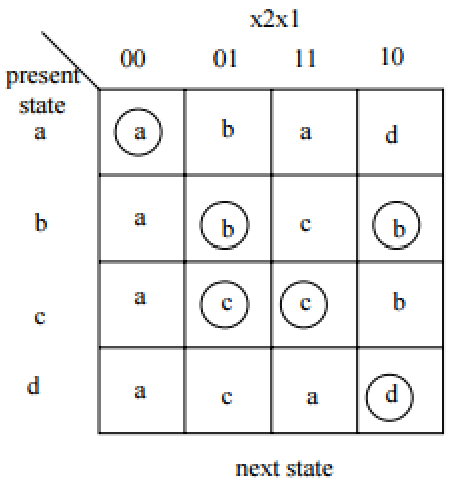
Figure 8: Merger Diagram

3.6 Reduction of flow table

Reduction of primitive flow table has two functions: Elimination of redundant stable states, Merging those stable states which are distinguishable by input states.
Let 00 = a; 01 = b; 11 = c; 01 = d

- 00 = a
- 01 = b
- 11 = c
- 01 = d

The resulting flow table is



present state \ x2x1	00	01	11	10
a	$\textcircled{a}$	b	a	d
b	a	$\textcircled{b}$	c	$\textcircled{b}$
c	a	$\textcircled{c}$	$\textcircled{c}$	b
d	a	c	a	$\textcircled{d}$

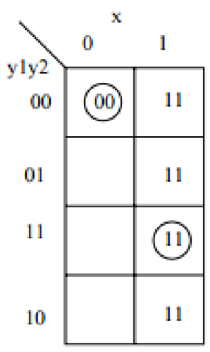
next state

4. Race-Free State Assignment

A race condition is caused when two or more binary state variables change value due to change in an input variable

- Unequal delays may cause state variables to change in unpredictable manner
- Race condition may be (i) non- critical, or (ii) critical

(i) Non critical race



y1y2 \ x	0	1
00	$\textcircled{00}$	11
01		11
11		$\textcircled{11}$
10		11

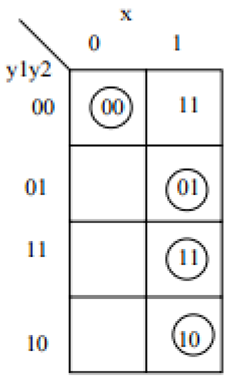
Possible transitions

00->11

00->01->11

00->10->11

(ii) Critical race



y1y2 \ x	0	1
00	00	11
01		01
11		11
10		10

Possible transitions

00->11

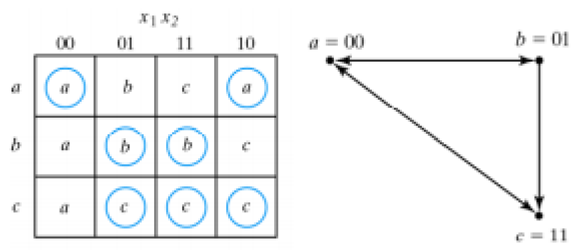
00->01

00->10

Once a reduced flow table has been derived, the next step in the design is to assign binary variables to each stable state. The main objective in choosing a proper binary state assignment is the prevention of critical races. Adjacent Binary Values: 2 binary values are said to be adjacent if they differ in only one variable (e.g. 010 and 011 are adjacent).

- 2-Row Flow-Table: The assignment of a single variable to a flow table with two rows does not impose critical race problems. [two adjacent adjacent values 0 and 1]
- 3-Row Flow-Table :

  Example: A flow table with 3 states requires an assignment of 2 variables. • We have the following transitions: a -> b, a -> c, b -> a, b -> c & c -> a (see the transition diagram in figure 9.



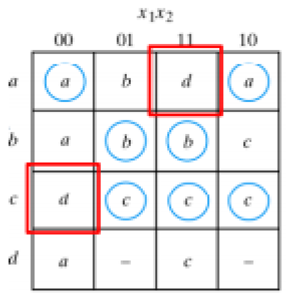
x_1x_2	00	01	11	10
a	a	b	c	a
b	a	b	b	c
c	a	c	c	c

Figure 9: (a)Flow table (b) Transition diagram

If we take the following assignment:

State	Value
a	00
b	01
c	11

This assignment will cause a critical race during the transition from a to c (2 changes in the binary state), and also from c to a.
A race-free assignment can be obtained by adding an extra row to the original flow table : The use of an extra row will not increase the number of binary state variables (2 variables), but it allows the formation of cycles between two stable states. The added row (d) is assigned the binary value (10), which is adjacent to both a & c. The transition from a to c must go through d, thus avoiding a critical race. The two squares with dashes in row d represent unspecified states (don’t care). These squares must not be assigned to 01 in order to avoid the possibility of stable state being established in the 4th row. Then the corresponding flow table and transition diagram will be shown in figure 10.



x_1x_2	00	01	11	10
a	a	b	d	a
b	a	b	b	c
c	d	c	c	c
d	a	–	c	–

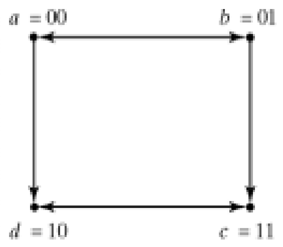
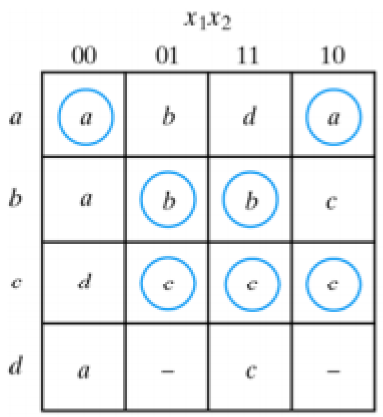
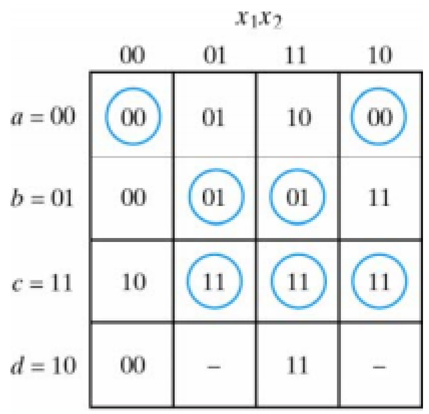


Figure 10: (a) Flow table (b) Transition diagram

The new flow table is converted to a transition table to complete the design process



x_1x_2	00	01	11	10
a	a	b	d	a
b	a	b	b	c
c	d	c	c	c
d	a	–	c	–



x_1x_2	00	01	11	10
a = 00	00	01	10	00
b = 01	00	01	01	11
c = 11	10	11	11	11
d = 10	00	–	11	–

Figure 11: (a) Flow table (b) Transition table