UNIT 4

ASYNCHRONOUS SEQUENTIAL CIRCUITS

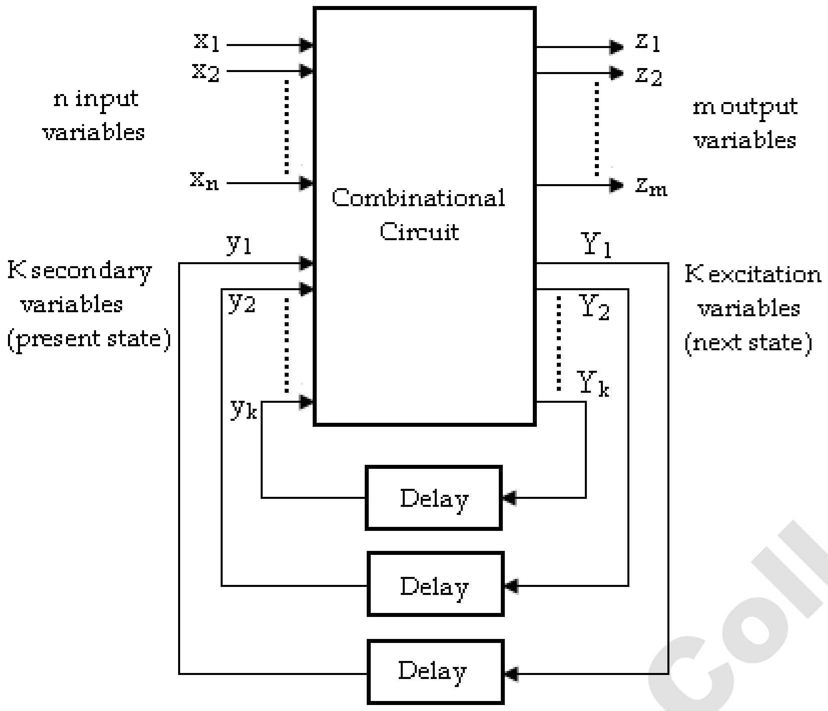
Analysis of asynchronous sequential machines, state assignment, asynchronous design problem

4.1 INTRODUCTION

A sequential circuit is specified by a time sequence of inputs, outputs and internal states. In synchronous sequential circuits, the output changes whenever a clock pulse is applied. The memory elements are clocked flip-flops.

Asynchronous sequential circuits do not use clock pulses. The memory elements in asynchronous sequential circuits are either unclocked flip-flops (Latches) or time-delay elements.

S.No	Synchronous sequential circuits	Asynchronous sequential circuits
1	Memory elements are clocked flip-flops	Memory elements are either unclocked flip-flops or time delay elements.
2	The change in input signals can affect memory element upon activation of clock signal.	The change in input signals can affect memory element at any instant of time.
3	The maximum operating speed of clock depends on time delays involved. Therefore synchronous circuits can operate slower than asynchronous.	Because of the absence of clock, it can operate faster than synchronous circuits.
4	Easier to design	More difficult to design



Block diagram of Asynchronous sequential circuits

The block diagram of asynchronous sequential circuit is shown above. It consists of a combinational circuit and delay elements connected to form feedback loops. There are ‘n’ input variables, ‘m’ output variables, and ‘k’ internal states.

The delay elements provide short term memory for the sequential circuit. The present-state and next-state variables in asynchronous sequential circuits are called secondary variables and excitation variables, respectively.

When an input variable changes in value, the ‘y’ secondary variable does not change instantaneously. It takes a certain amount of time for the signal to propagate from the input terminals through the combinational circuit to the ‘Y’ excitation variables where the new values are generated for the next state. These values propagate through the delay elements and become the new present state for the secondary variables.

In steady-state condition, excitation and secondary variables are same, but during transition they are different.

To ensure proper operation, it is necessary for asynchronous sequential circuits to attain a stable state before the input is changed to a new value. Because of unequal delays in wires and combinational circuits, it is impossible to have two or more input variable change at exactly same instant. Therefore, simultaneous changes of two or more input variables are avoided.

Only one input variable is allowed to change at any one time and the time between input changes is kept longer than the time it takes the circuit to reach stable state.

Types:

According to how input variables are to be considered, there are two types

- Fundamental mode circuit
- Pulse mode circuit.

Fundamental mode circuit assumes that:

- The input variables change only when the circuit is stable.
- Only one input variable can change at a given time.
- Inputs are levels (0, 1) and not pulses.

Pulse mode circuit assumes that:

- The input variables are pulses (True, False) instead of levels.
- The width of the pulses is long enough for the circuit to respond to the input.
- The pulse width must not be so long that it is still present after the new state is reached.

4.2 Analysis of Fundamental Mode Circuits

The analysis of asynchronous sequential circuits consists of obtaining a table or a diagram that describes the sequence of internal states and outputs as a function of changes in the input variables.

4.2.1 Analysis procedure

The procedure for obtaining a transition table from the given circuit diagram is as follows.

1. Determine all feedback loops in the circuit.
2. Designate the output of each feedback loop with variable Y1 and its corresponding inputs $y_1, y_2, \ldots y_k$, where k is the number of feedback loops in the circuit.
3. Derive the Boolean functions of all Y's as a function of the external inputs and the y's.
4. Plot each Y function in a map, using y variables for the rows and the external inputs for the columns.
5. Combine all the maps into one table showing the value of $Y= Y_1, Y_2, \ldots Y_k$ inside each square.
6. Circle all stable states where Y=y. The resulting map is the transition table.

4.2.2 Problems

1. An asynchronous sequential circuit is described by the following excitation and output function,

   $$Y= x_1x_2+ (x_1+x_2)\, y$$

   $$Z= Y$$

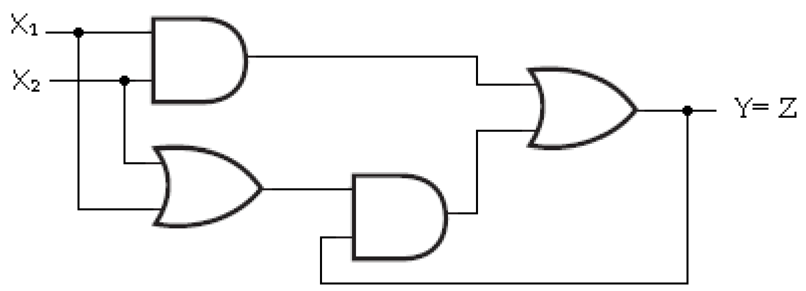
   a) Draw the logic diagram of the circuit.

   b) Derive the transition table, flow table and output map.

   c) Describe the behavior of the circuit.

Soln:

i) The logic diagram is shown as,

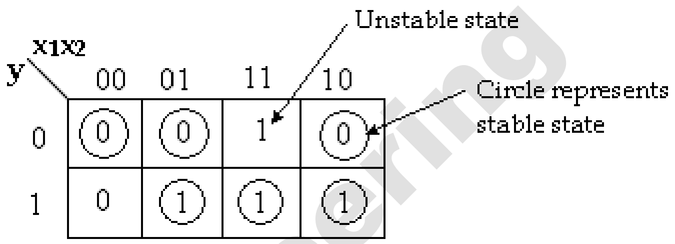


Logic diagram

ii)

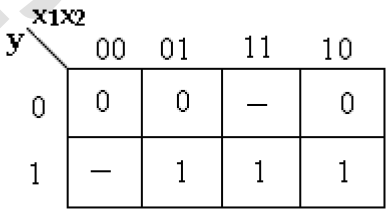
y	x_1	x_2	x_1x_2	$(x_1+x_2)y$	$Y= x_1x_2+ (x_1+x_2)y$	$Z= Y$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	0	1	1	1
1	1	1	1	1	1	1

Transition table:



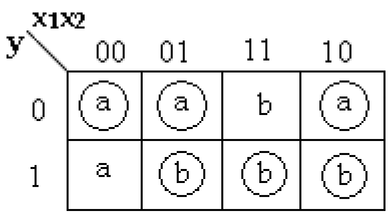
Output map:

Output is mapped for all stable states. For unstable states output is mapped unspecified.



Flow table:

Assign a= 0; b= 1



iii)

The circuit gives carry output of the full adder circuit.

2. Design an asynchronous sequential circuit that has two internal states and one output. The excitation and output function describing the circuit are as follows:

   $Y_1 = x_1x_2 + x_1y_2 + x_2y_1$

   $Y_2 = x_2 + x_1y_1y_2 + x_1y_1$

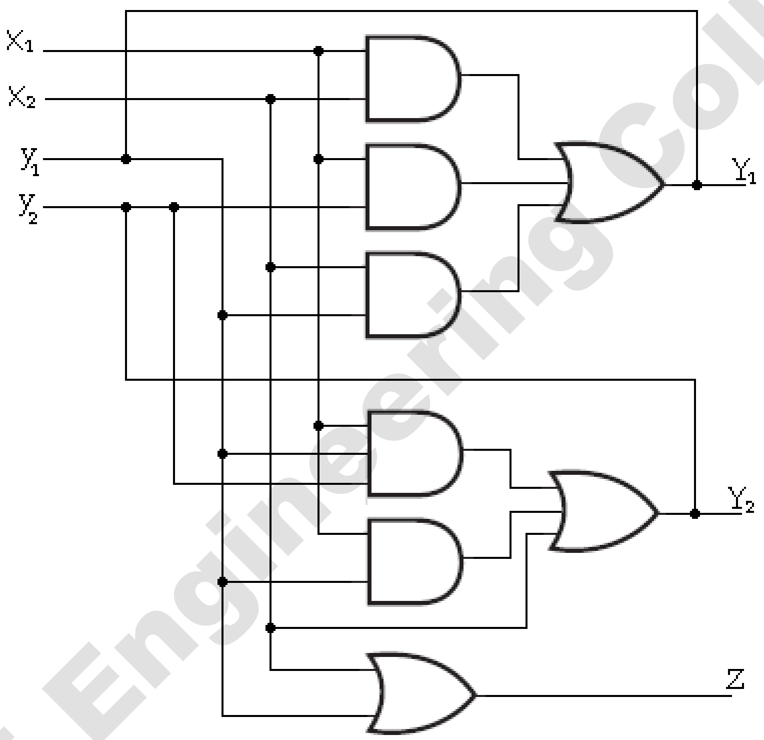
   $Z = x_2 + y_1$.

   a) Draw the logic diagram of the circuit.

   b) Derive the transition table, output map and flow table.

Soln:

i) The logic diagram is shown as,



Logic Diagram

ii)

y_1	y_2	x_1	x_2	x_1x_2	x_1y_2	x_2y_1	$x_1y_1y_2$	x_1y_1	Y_1	Y_2	$Z = x_2 + y_1$
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	1	1	1

0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	1	1
0	1	1	0	0	1	0	0	0	1	0	0
0	1	1	1	1	1	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0	0	1	1	1
1	0	1	0	0	0	0	0	1	0	1	1
1	0	1	1	1	0	1	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	0	1
1	1	0	1	0	0	1	0	0	1	1	1
1	1	1	0	0	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Map for Y_1

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	0	1	1	1
10	0	1	1	0

Map for Y_2

y_1y_2 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	1
10	0	1	1	1

Transition table and Output map

Map for Y_1Y_2

y_1y_2 \ x_1x_2	00	01	11	10
00	$\textcircled{00}$	01	11	$\textcircled{00}$
01	00	$\textcircled{01}$	11	10
11	00	$\textcircled{11}$	$\textcircled{11}$	$\textcircled{11}$
10	00	11	11	01

Transition table

y_1y_2 \ x_1x_2	00	01	11	10
00	0	–	–	0
01	–	1	–	–
11	–	1	1	1
10	–	–	–	–

Output map

Primitive Flow table

y1y2 \ x1x2	00	01	11	10
a	$\textcircled{a}$	b	c	$\textcircled{a}$
b	a	$\textcircled{b}$	c	d
c	a	$\textcircled{c}$	$\textcircled{c}$	$\textcircled{c}$
d	a	c	c	b

3. An asynchronous sequential circuit is described by the excitation and output functions,

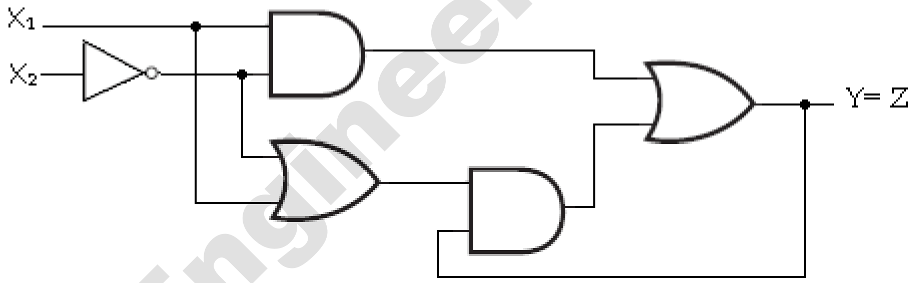
   $Y = x_1x_2' + (x_1+x_2')\,y$

   $Z = Y$

   a) Draw the logic diagram of the circuit.

   b) Derive the transition table, output map and flow table.

Soln:



Logic diagram

ii)

y	x_1	x_2	x_2'	x_1x_2'	$(x_1+x_2')y$	$Y = x_1x_2' + (x_1+x_2')y$	$Z = Y$
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	1	0	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	1

Transition table:

y \ X1X2	00	01	11	10
0	$\textcircled{0}$	$\textcircled{0}$	$\textcircled{0}$	1
1	$\textcircled{1}$	0	$\textcircled{1}$	$\textcircled{1}$

Transition Table

Output map:

Output is mapped for all stable states. For unstable states output is mapped unspecified.

y \ X1X2	00	01	11	10
0	0	0	0	–
1	1	–	1	1

Output map

Flow table:

Assign a= 0; b= 1

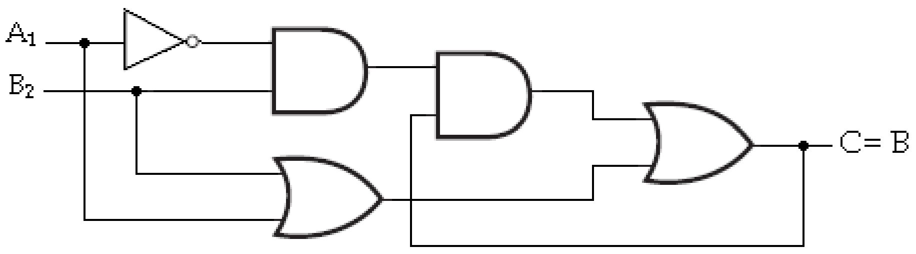
y \ X1X2	00	01	11	10
0	$\textcircled{a}$	$\textcircled{a}$	$\textcircled{a}$	b
1	$\textcircled{b}$	a	$\textcircled{b}$	$\textcircled{b}$

4. An asynchronous sequential circuit is described by the excitation and output functions, **$B = (A_1'B_2)\,b + (A_1+B_2)$ C= B**

   a) Draw the logic diagram of the circuit.

   b) Derive the transition table, output map and flow table.

Soln:

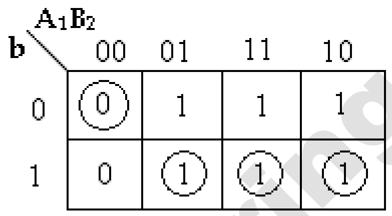


Logic Diagram

ii)

b	A_1	B_2	A_1'	$(A_1'B_2)b$	A_1+B_2	$B= (A_1'B_2)\ b+ (A_1+B_2)$	C= B
0	0	0	1	0	0	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	1	1	1
0	1	1	0	0	1	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	1	1	1
1	1	0	0	0	1	1	1
1	1	1	0	0	1	1	1

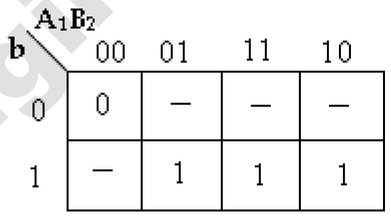
Transition table



b \ A_1B_2	00	01	11	10
0	$\textcircled{0}$	1	1	1
1	0	$\textcircled{1}$	$\textcircled{1}$	$\textcircled{1}$

Output map

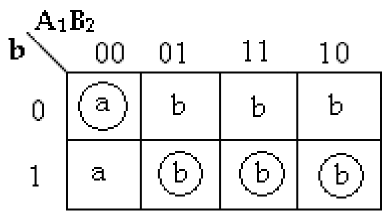
Output is mapped for all stable states.



b \ A_1B_2	00	01	11	10
0	0	–	–	–
1	–	1	1	1

Flow table

Assign a= 0; b= 1



b \ A_1B_2	00	01	11	10
0	$\textcircled{a}$	b	b	b
1	a	$\textcircled{b}$	$\textcircled{b}$	$\textcircled{b}$

5. An asynchronous sequential circuit is described by the excitation and output functions,

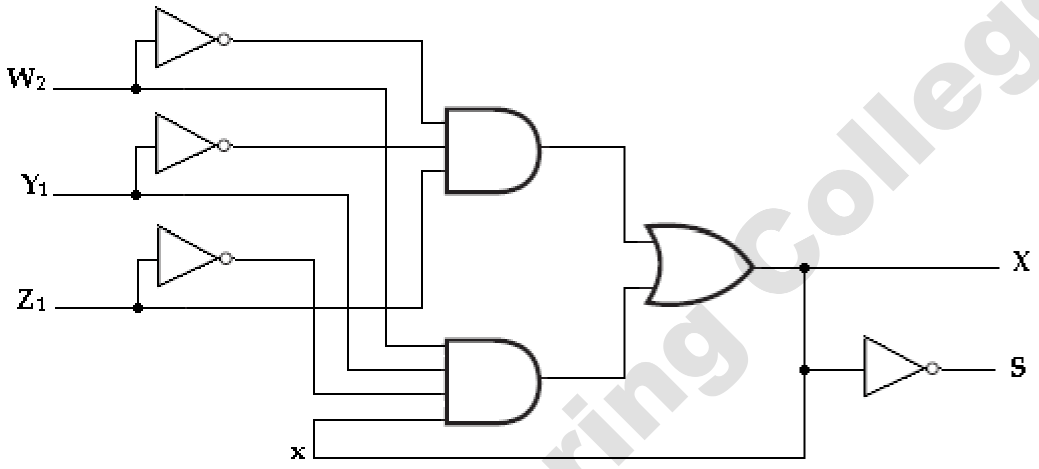
$X = (Y_1Z_1'W_2)\,x + (Y_1'Z_1W_2')$

$S = X'$

a) Draw the logic diagram of the circuit

b) Derive the translation table and output map

Soln:



x	W_2	W_2'	Y_1	Y_1'	Z_1	Z_1'	$(Y_1Z_1'W_2)\,x$	$Y_1'Z_1W_2'$	X	$S = X'$
0	0	1	0	1	0	1	0	0	0	1
0	0	1	0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1
0	1	0	0	1	0	1	0	0	0	1
0	1	0	0	1	1	0	0	0	0	1
0	1	0	1	0	0	1	0	0	0	1
0	1	0	1	0	1	0	0	0	0	1
1	0	1	0	1	0	1	0	0	0	1
1	0	1	0	1	1	0	0	1	1	0
1	0	1	1	0	0	1	0	0	0	1
1	0	1	1	0	1	0	0	0	0	1

1	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	1	0	0	0	0	1
1	1	0	1	0	0	1	1	0	1	0
1	1	0	1	0	1	0	0	0	0	1

Map for X

$xW_2 \backslash Y_1Z_1$	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	0	0	1
10	0	1	0	0

Map for S

$xW_2 \backslash Y_1Z_1$	00	01	11	10
00	1	0	1	1
01	1	1	1	1
11	1	1	1	0
10	1	0	1	1

Transition table and Output map:

Map for XS

$xW_2 \backslash Y_1Z_1$	00	01	11	10
00	01	10	01	01
01	(01)	(01)	(01)	(01)
11	01	01	01	10
10	01	(10)	01	01

Transition table

$xW_2 \backslash Y_1Z_1$	00	01	11	10
00	–	–	–	–
01	1	1	1	1
11	–	–	–	–
10	–	0	–	–

Output map

4.3 Analysis of Pulse Mode Circuits

Pulse mode asynchronous sequential circuits rely on the input pulses rather than levels. They allow only one input variable to change at a time. They can be implemented by employing a SR latch.

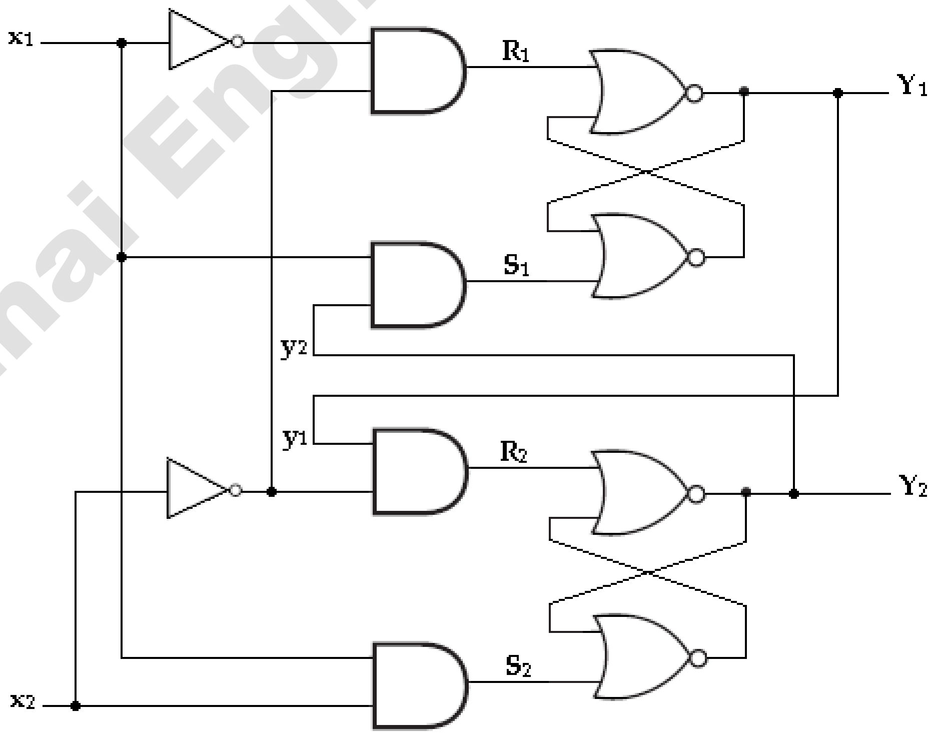
The procedure for analyzing an asynchronous sequential circuit with SR latches can be summarized as follows:

1. Label each latch output with Y_i and its external feedback path (if any) with y_i for

   $i = 1,2 ,..,, k$.
2. Derive the Boolean functions for the S_i and R_i inputs in each latch.
3. Check whether ***SR* = 0** for each NOR latch or whether **S'R' = 0** for each NAND latch. If either of these condition is not satisfied, there is a possibility that the circuit may not operate properly.
4. Evaluate **$Y = S + R'y$** for each NOR latch or **$Y = S' + Ry$** for each NAND latch.
5. Construct a map with the y's representing the rows and the x inputs representing the columns.
6. Plot the value of $Y= Y_1Y_2 \ldots\ldots Y_k$ in the map.
7. Circle all stable states such that ***Y* = y**. The resulting map is the transition table.

The analysis of a circuit with latches will be demonstrated by means of the below example.

1. Derive the transition table for the pulse mode asynchronous sequential circuit shown below.



<u>**Example of a circuit with SR latches**</u>

Soln:

There are two inputs x_1 and x_2 and two external feedback loops giving rise to the secondary variables y_1 and y_2.

Step 1:

The Boolean functions for the S and R inputs in each latch are:

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

Step 2:

Check whether the conditions SR= 0 is satisfied to ensure proper operation of the circuit.

$$S_1R_1 = x_1y_2\,x_1'x_2' = 0$$
$$S_2R_2 = x_1x_2\,x_2'y_1 = 0$$

The result is 0 because $x_1x_1' = x_2x_2' = 0$

Step 3:

Evaluate Y_1 and Y_2. The excitation functions are derived from the relation **$Y = S + R'y$.**

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

y_1	y_2	x_1	x_2	x_1y_2	x_1y_1	x_2y_1	x_1x_2	x_2y_2	$y_1'y_2$	Y_1	Y_2
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0	1
0	1	0	1	0	0	0	0	1	1	0	1
0	1	1	0	1	0	0	0	0	1	1	1
0	1	1	1	1	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	1	0

1 1	0 0	1 1	0 1	0 0	1 1	0 1	0 1	0 0	0 0	1 1	0 1
1 1 1 1	1 1 1 1	0 0 1 1	0 1 0 1	0 0 1 1	0 0 1 1	0 1 0 1	0 0 0 1	0 1 0 1	0 0 0 0	0 1 1 1	0 1 0 1

Step 4:

Maps for Y_1 and Y_2.

Map for Y_1

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	1	1
10	0	1	1	1

Map for Y_2

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	0	1	1	0
10	0	0	1	0

Step 5:

Transition table

Map for Y_1Y_2

y_1y_2 \ x_1x_2	00	01	11	10
00	$\textcircled{00}$	$\textcircled{00}$	01	$\textcircled{00}$
01	$\textcircled{01}$	$\textcircled{01}$	11	11
11	00	$\textcircled{11}$	$\textcircled{11}$	10
10	00	$\textcircled{10}$	11	$\textcircled{10}$

4.4 RACES:

A race condition is said to exist in an asynchronous sequential circuit when two or more binary state variables change value in response to a change in an input variable.

Races are classified as:

i. Non-critical races

ii. Critical races.

Non-critical races:

If the final stable state that the circuit reaches does not depend on the order in which the state variables change, the race is called a non-critical race.

If a circuit, whose transition table (a) starts with the total stable state y_1y_2x= *000* and then change the input from 0 to 1. The state variables must then change from 00 to 11, which define a race condition.

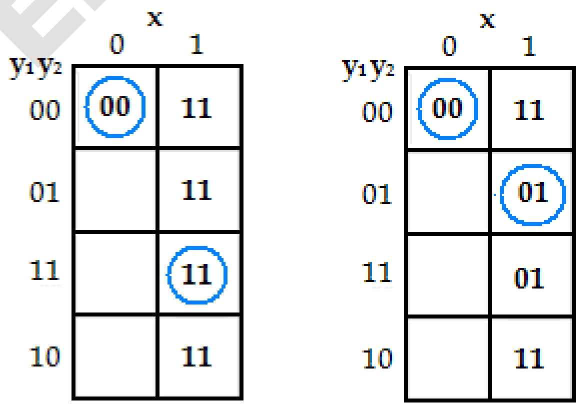
The possible transitions are:

$00 \longrightarrow 11$

$00 \longrightarrow 01 \longrightarrow 11$

$00 \longrightarrow 10 \longrightarrow 11$

In all cases, the final state is the same, which results in a non-critical condition. In (a), the final state is (y_1y_2x= 111), and in (b), it is (y_1y_2x= 011).



(a) Possible transitions:

$00 \rightarrow 11$

$00 \rightarrow 01 \rightarrow 11$

$00 \rightarrow 10 \rightarrow 11$

(b) Possible transitions:

$00 \rightarrow 11 \rightarrow 01$

$00 \rightarrow 01$

$00 \rightarrow 10 \rightarrow 11 \rightarrow 01$

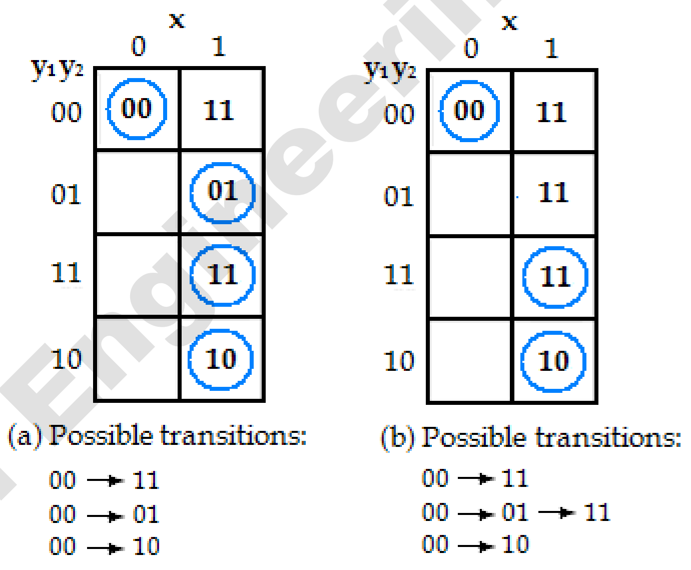
<u>**Examples of Non-critical Races**</u>

Critical races:

A race becomes critical if the correct next state is not reached during a state transition. If it is possible to end up in two or more different stable states, depending on the order in which the state variables change, then it is a critical race. For proper operation, critical races must be avoided.

The below transition table illustrates critical race condition. The transition table (a) starts in stable state (y_1y_2x= 000), and then change the input from 0 to 1. The state variables must then change from 00 to 11. If they change simultaneously, the final total stable state is 111. In the transition table (a), if, because of unequal propagation delay, Y_2 changes to 1 before Y_1 does, then the circuit goes to the total stable state 011 and remains there. If, however, Y_1 changes first, the internal state becomes 10 and the circuit will remain in the stable total state 101.

Hence, the race is critical because the circuit goes to different stable states, depending on the order in which the state variables change.

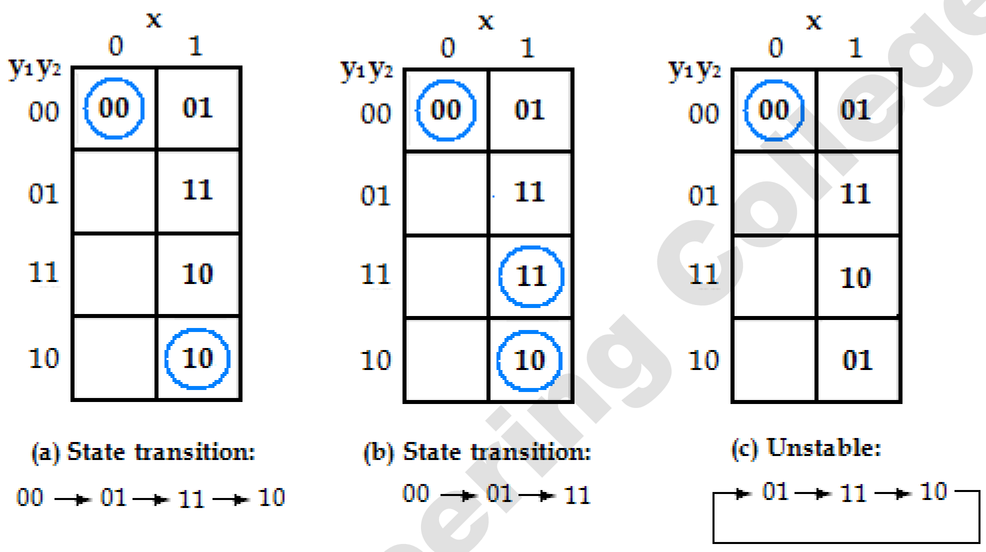


<u>**Examples of Critical Races**</u>

4.5 CYCLES

Races can be avoided by directing the circuit through intermediate unstable states with a unique state-variable change. When a circuit goes through a unique sequence of unstable states, it is said to have a ***cycle***.

Again, we start with y_1y_2 = 00 and change the input from *0* to 1. The transition table (a) gives a *unique* sequence that terminates in a total stable state 101. The table in (b) shows that even though the state variables change from 00 to 11, the cycle provides a unique transition from *00* to 01 and then to 11, Care must be taken when using a cycle that terminates with a stable state. If a cycle does not terminate with a stable state, the circuit will keep going from one unstable state to another, making the entire circuit unstable. This is demonstrated in the transition table (c).

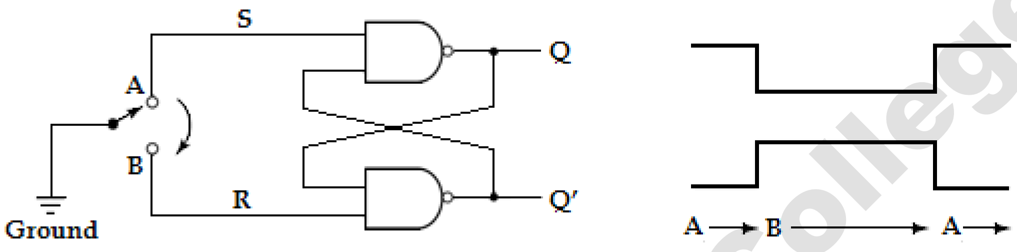


Examples of Cycles

Debounce Circuit:

Input binary information in binary information can be generated manually be means of mechanical switches. One position of the switch provides a voltage equivalent to logic 1, and the other position provides a second voltage equivalent to logic 0. Mechanical switches are also used to start, stop, or reset the digital system. A common characteristic of a mechanical switch is that when the arm is thrown from one position to the other the switch contact vibrates or bounces several times before coming to a final rest. In a typical switch, the contact bounce may take several milliseconds to die out, causing the signal to oscillate between 1 and 0 because the switch contact is vibrating.

A debounce circuit is a circuit which removes the series of pulses that result from a contact bounce and produces a single smooth transition of the binary signal from *0* to 1 or from 1 to 0. One such circuit consists of a single-pole, double-throw switch connected to an *SR* latch, as shown below. The center contact is connected to ground that provides a signal equivalent to logic 0. When one of the two contacts, A or *B,* is not connected to ground through the switch, it behaves like a logic-1 signal. When the switch is thrown from position *A* to position *B* and back, the outputs of the latch produce a single pulse as shown, negative for *Q* and positive for *Q*'. The switch is usually a push button whose contact rests in position A. When the pushbutton is depressed, it goes to position B and when released, it returns to position A.



Debounce Circuit

The operation of the debounce circuit is as follows: When the switch resets in position A, we have the condition S = 0, *R* = 1 and *Q* = 1, *Q*' = 0. When the switch is moved to position *B,* the ground connection causes R to go to *0,* while *S* becomes a 1 because contact A is open. This condition in turn causes output Q to go to 0 and Q' to go to 1. After the switch makes an initial contact with *B,* it bounces several times. The output of the latch will be unaffected by the contact bounce because Q' remains 1 (and Q remains *0)* whether *R* is equal to *0* (*contact with ground*) or equal to 1 (*no contact with ground*). When the switch returns to position A, *S* becomes *0* and *Q* returns to 1. The output again will exhibit a smooth transition, even if there is a contact bounce in position A.

4.6 DESIGN OF FUNDAMENTAL MODE SEQUENTIAL CIRCUITS

The design of an asynchronous sequential circuit starts from the statement of the problem and concludes in a logic diagram. There are a number of design steps that must be carried out in order to minimize the circuit complexity and to produce a stable circuit without critical races.

The design steps are as follows:

1. State the design specifications.
2. Obtain a primitive flow table from the given design specifications.
3. Reduce the flow table by merging rows in the primitive flow table.
4. Assign binary state variables to each row of the reduced flow table to obtain the transition table. The procedure of state assignment eliminates any possible critical races.
5. Assign output values to the dashes associated with the unstable states to obtain the output maps.
6. Simplify the Boolean functions of the excitation and output variables and draw the logic diagram.

1. Design a gated latch circuit with inputs, G (gate) and D (data), and one output, Q. Binary information present at the D input is transferred to the Q output when G is equal to 1. 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 of transition occurred is retained at the Q output. The gated latch is a memory element that accepts the value of D when G= 1 and retains this value after G goes to 0, a change in D does not change the value of the output Q.

Soln:

Step 1:

From the design specifications, we know that Q= 0 if DG= 01

and Q= 1 if DG= 11

because D must be equal to Q when G= 1.

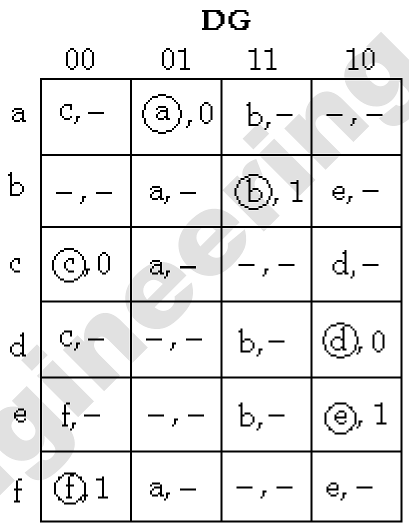
When G goes to 0, the output depends on the last value of D. Thus, if the transition is from 01 to 00 to 10, then Q must remain 0 because D is 0 at the time of the transition from 1 to 0 in G.

If the transition of DG is from 11 to 10 to 00, then Q must remain 1. This information results in six different total states, as shown in the table.

State	Inputs		Output	Comments
	D	G	Q	
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

Step 2: A primitive flow is a flow table with only one stable total state in each row. It has one row for each state and one column for each input combination.

Step 3:



	DG 00	01	11	10
a	c, –	$\textcircled{a}$, 0	b, –	–, –
b	–, –	a, –	$\textcircled{b}$, 1	e, –
c	$\textcircled{c}$, 0	a, –	–, –	d, –
d	c, –	–, –	b, –	$\textcircled{d}$, 0
e	f, –	–, –	b, –	$\textcircled{e}$, 1
f	$\textcircled{f}$, 1	a, –	–, –	e, –

Primitive flow table

The primitive flow table has only stable state in each row. The table can be reduced to a smaller number of rows if two or more stable states are placed in the same row of the flow table. The grouping of stable states from separate rows into one common row is called *merging*.

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

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

States that are candidates for merging

Thus, the three rows a, c, and d can be merged into one row. The second row of the reduced table results from the merging of rows b, e, and f of the primitive flow table.

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

Reduced table- 1

The states c & d are replaced by state a, and states e & f are replaced by state b

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

Reduced table- 2

Step 4:

Assign distinct binary value to each state. This assignment converts the flow table into a transition table. A binary state assignment must be made to ensure that the circuit will be free of critical races.

Assign 0 to state a, and 1 to state b in the reduced state table.

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

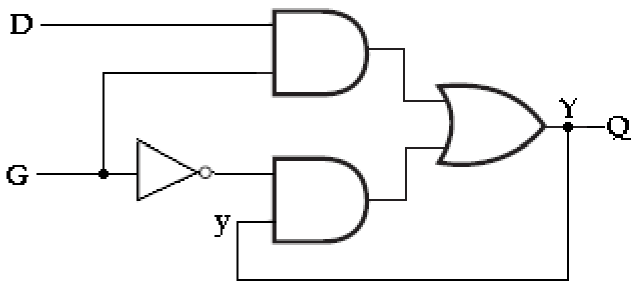
$Y = DG + \overline{G}y$

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

$Q = Y$

Transition table and output map

Step 5:



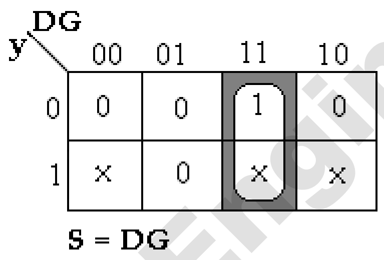
Gated-Latch Logic diagram

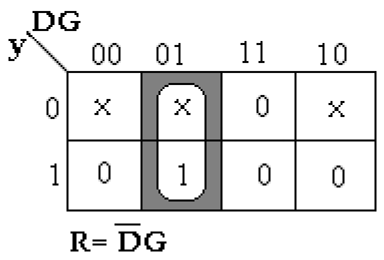
The diagram can be implemented also by means of an SR latch. Obtain the Boolean function for S and R inputs.

y	Y	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

SR Latch excitation table

From the information given in the transition table and from the latch excitation table conditions, we can obtain the maps for the S and R inputs of the latch.

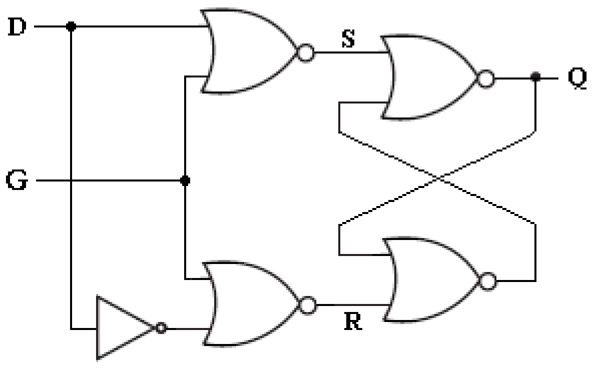




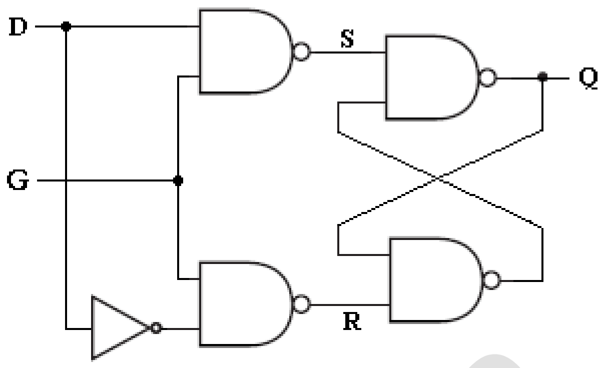
Maps for S and R

The logic diagram consists of an SR latch using NOR latch and the gates required to implement the S and R Boolean functions. With a NAND latch, we must use the complemented values for S and R.

S′ = (DG)′ and **R′ = (D′G)′**



Logic diagram with NOR latch



Logic diagram with NAND latch

2. Design a negative-edge triggered T flip-flop. The circuit has two inputs, T (toggle) and G (clock), and one output, Q. the output state is complemented if T= 1 and the clock changes from 1 to 0 (negative-edge triggering). Otherwise, under any other input condition, the output Q remains unchanged.

Step 1:

Starting with the input condition TC= 11 and assign it to a. The circuit goes to state b and output Q complements from 0 to 1 when C changes from 1 to 0 while T remains a 1.

Another change in the output occurs when the circuit changes from state c to state d. In this case, T=1, C changes from 1 to 0, and the output Q complements from 1 to 0. The other four states in the table do not change the output, because T is equal to 0. If Q is initially 0, it stays at 0, and if initially at 1, it stays at 1 even though the clock input changes.

State	Inputs		Output	Comments
	T	G	Q	
a	1	1	0	Initial output is 0
b	1	0	1	After state a
c	1	1	1	Initial output is 1
d	1	0	0	After state c
e	0	0	0	After state d or f
f	0	1	0	After state e or a
g	0	0	1	After state b or h
h	0	1	1	After state g or c

Specifications of total states

Step 2: **Merging of the flow table**

The information for the primitive flow table can be obtained directly from the condition listed in the above table. We first fill in one square in each row belonging to stable state in that row as listed in the table.

Then we enter dashes in those squares whose input differs by two variables from the input corresponding to the stable state.

The unstable conditions are then determined by utilizing the information listed under the comments in the above table.

Step 3: Compatible pairs

	TC			
	00	01	11	10
a	−,−	f,−	ⓐ,0	b,−
b	g,−	−,−	c,−	ⓑ,1
c	−,−	h,−	ⓒ,1	d,−
d	e,−	−,−	a,−	ⓓ,0
e	ⓔ,0	f,−	−,−	d,−
f	e,−	ⓕ,0	a,−	−,−
g	ⓖ,1	h,−	−,−	b,−
h	g,−	ⓗ,1	c,−	−,−

Primitive flow table

The rows in the primitive flow table are merged by first obtaining all compatible pairs of states. This is done by means of the implication table.

b	a,c x						
c	X	b,d x					
d	b,d x	X	a,c x				
e	b,d x	e,g x b,d x	f,h x	✓			
f	✓	e,g x a,c x	f,h x a,c x	✓	✓		
g	f,h x	✓	b,d x	e,g x b,d x	X	e,g x f,h x	
h	f,h x a,c x	✓	✓	d,e x c,f x	e,g x f,h x	X	✓
	a	b	c	d	e	f	g

Implication table

The implication table is used to find the compatible states. The only difference is that when comparing rows, we are at liberty to adjust the dashes to fit any desired condition. The two states are compatible if in every column of the corresponding rows in the primitive flow table, there are identical or compatible pairs and if there is no conflict in the output values.

A check mark (✓) designates a square whose pair of states is compatible. Those states that are not compatible are marked with a cross (x). The remaining squares are recorded with the implied pairs that need further investigation.

The squares that contain the check marks define the compatible pairs:

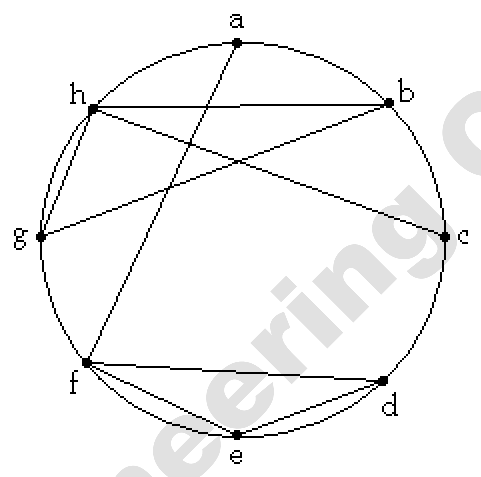
(a, f) (b, g) (b, h) (c, h) (d, e) (d, f) (e, f) (g, h)

Step 4: Maximal compatibles

Having found all the compatible pairs, the next step is to find larger set of states that are compatible. The ***maximal compatible*** is a group of compatibles that contain all the possible combinations of compatible states. The maximal compatible can be obtained from a merger diagram.

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.

- A line represents a compatible pair
- A triangle constitutes a compatible with three states
- An n-state compatible is represented in the merger diagram by an n-sided polygon with all its diagonals connected.



Merger Diagram

The merger diagram is obtained from the list of compatible pairs derived from the implication table. There are eight straight lines connecting the dots, one for each compatible pair. The lines form a geometrical pattern consisting of two triangles connecting (b, g, h) & (d, e, f) and two lines (a, f) & (c, h). The maximal compatibles are:

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

	TC 00	01	11	10
a, f	e, –	ⓕ, 0	ⓐ, 0	b, –
b, g, h	ⓖ, 1	ⓗ, 1	c, –	ⓑ, 1
c, h	g, –	ⓗ, 1	ⓒ, 1	d, –
d, e, f	ⓔ, 0	ⓕ, 0	a, –	ⓓ, 0

Reduced Flow table

The reduced flow table is drawn. The compatible states are merged into one row that retains the original letter symbols of the states. The four compatible set of states are used to merge the flow table into four rows.

	TC 00	01	11	10
a	d, –	ⓐ, 0	ⓐ, 0	b, –
b	ⓑ, 1	ⓑ, 1	c, –	ⓑ, 1
c	b, –	ⓒ, 1	ⓒ, 1	d, –
d	ⓓ, 0	ⓓ, 0	a, –	ⓓ, 0

Final Reduced Flow table

Here we assign a common letter symbol to all the stable states in each merged row. Thus, the symbol f is replaced by a; g & h are replaced by b, and similarly for the other two rows.

Step 5: State Assignment and Transition table

Find the race-free binary assignment for the four stable states in the reduced flow table. Assign a= 00, b= 01, c= 11 and d= 10.

Substituting the binary assignment into the reduced flow table, the transition table is obtained. The output map is obtained from the reduced flow table.

Transition Table and Output Map

y_1y_2 \ TC	00	01	11	10
00	10	(00)	(00)	01
01	(01)	(01)	11	(01)
11	01	(11)	(11)	10
10	(10)	(10)	00	(10)

Transition table

y_1y_2 \ TC	00	01	11	10
00	0	0	0	X
01	1	1	1	1
11	1	1	1	X
10	0	0	0	0

Output map Q= y_2

Logic Diagram:

y1y2 \ TC	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	X	X	X
10	X	X	0	X

$S_1 = y_2TC + y'_2T'C'$

y1y2 \ TC	00	01	11	10
00	0	X	X	X
01	X	X	0	X
11	1	0	0	0
10	0	0	1	0

$R_1 = y_2T'C' + y_2'TC$

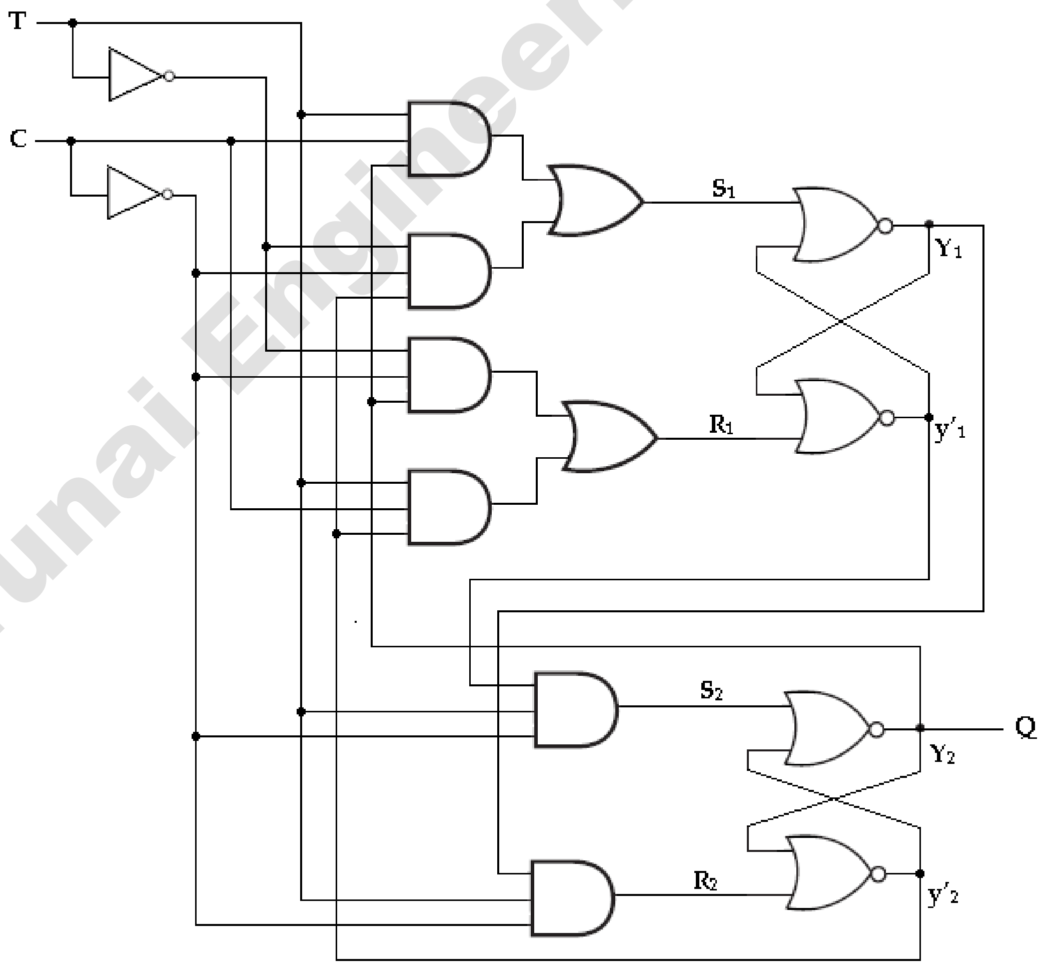
y1y2 \ TC	00	01	11	10
00	0	0	0	1
01	X	X	X	X
11	X	X	X	0
10	0	0	0	0

$S_2 = y'_1TC'$

y1y2 \ TC	00	01	11	10
00	X	X	X	0
01	0	0	0	0
11	0	0	0	1
10	X	X	X	X

$R_2 = y_1TC'$

Maps for Latch Inputs



3. Develop a state diagram and primitive flow table for a logic system that has two inputs, X and Y, and a single output X, which is to behave in the following manner. Initially, both inputs and output are equal to 0. Whenever X= 1 and Y= 0, the Z becomes 1 and whenever X= 0 and Y= 1, the Z becomes 0. When inputs are zero, i.e. X= Y= 0 or inputs are one, i.e. X= Y= 1, the output Z does not change; it remains in the previous state. The logic system has edge triggered inputs without having a clock. The logic system changes state on the rising edges of the two inputs. Static input values are not to have any effect in changing the Z output.

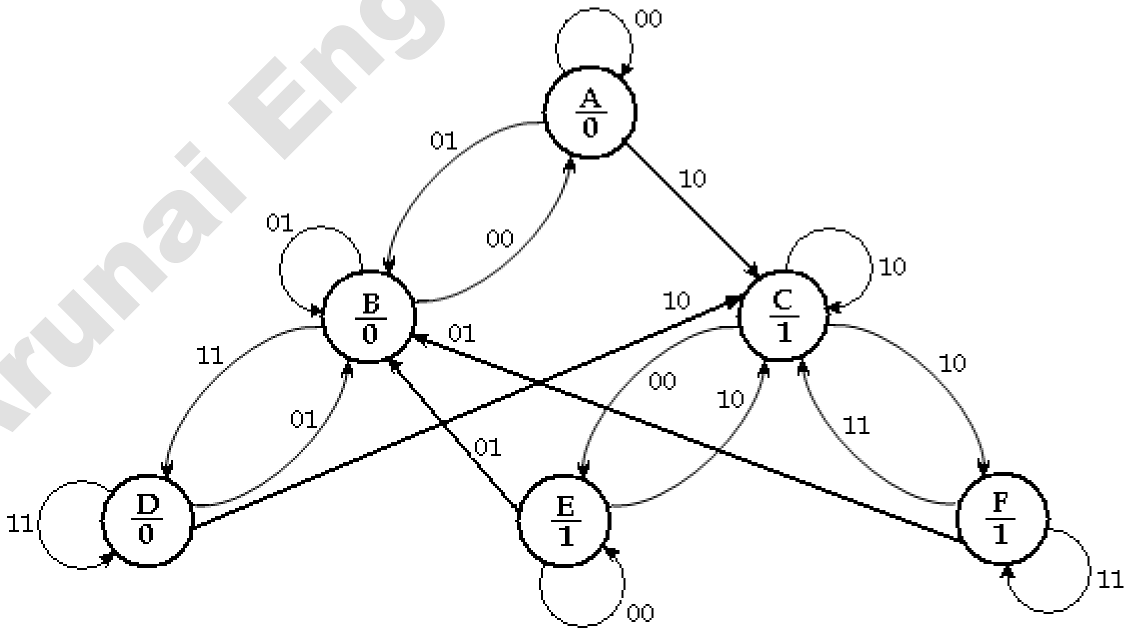
<u>**Soln:**</u>

The conditions given are,

- Initially both inputs X and Y are 0.
- When X= 1, Y= 0; Z= 1
- When X= 0, Y= 1; Z= 0
- When X= Y= 0 or X= Y= 1, then Z does not change, it remains in the previous state.

<u>Step 1:</u>

The above state transitions are represented in the state diagram as,



<u>**State diagram**</u>

Step 2:

A primitive flow table is constructed from the state diagram. The primitive flow table has one row for each state and one column for each input combination. Only one stable state exists for each row in the table. The stable state can be easily identified from the state diagram. For example, state A is stable with output 0 when inputs are 00, state C is stable with output 1 when inputs are 10 and so on.

We know that both inputs are not allowed to change simultaneously, so 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 00. Since only one input can change at any given time, it can change to 01 or 10, but not to 11. Therefore we can enter two dashes in the 11 column of row A.

The remaining places in the primitive flow table can be filled by observing state diagram. For example, state B is the next state for present state A when input combination is 01; similarly state C is the next state for present state A when input combination is 10.

	XY			
	00	01	11	10
A	Ⓐ,0	B,–	–,–	C,–
B	A,–	Ⓑ,0	D,–	–,–
C	E,–	–,–	F,–	Ⓒ,1
D	–,–	B,–	Ⓓ,0	C,–
E	Ⓔ,1	B,–	–,–	C,–
F	–,–	B,–	Ⓕ,1	C,–

Primitive flow table

Step 3:

The rows in the primitive flow table are merged by first obtaining all compatible pairs of states. This is done by means of the implication table.

B	✓				
C	A,E ✗	A,E ✗ D,F ✗			
D	✓	✓	D,F ✗		
E	A,E ✗	A,E ✗	✓	✓	
F	✓	D,F ✗	✓	D,F ✗	✓
	A	B	C	D	E

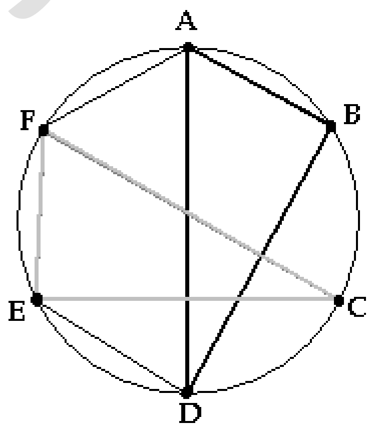
The squares that contain the check marks (✓) define the compatible pairs:

(A, B) (A, D) (A, F) (B, D) (C, E) (C, F) (D, E) (E, F)

Step 4:

The merger diagram is obtained from the list of compatible pairs derived from the implication table. There are eight straight lines connecting the dots, one for each compatible pair. The lines form a geometrical pattern consisting of two triangles connecting (A, B, D) & (C, E, F) and two lines (A, F) & (D, E). The maximal compatibles are:

(A, B, D) **(C, E, F)** **(A, F)** **(D, E)**



Merger diagram

Closed covering condition:

The condition that must be satisfied for merging rows is that the set of chosen compatibles must cover all the states and must be *closed.* The set will cover all the states if it includes all the states of the original state table. The closure condition is satisfied if there are no implied states *or* if the implied states are included within the set. A closed set of compatibles that covers all the states is called a *closed covering.*

If we remove (A, F) and (D, E), we are left with a set of two compatibles:

(A, B, D) (C, E, F)

All six states from the primitive flow table are included in this set. Thus, the set satisfies the covering condition.

The reduced flow table is drawn as below.

	XY 00	01	11	10
A,B,D	Ⓐ,0	Ⓑ,0	Ⓓ,0	C,–
C,E,F	Ⓔ,1	B,–	Ⓕ,1	Ⓒ,1

Reduced flow table

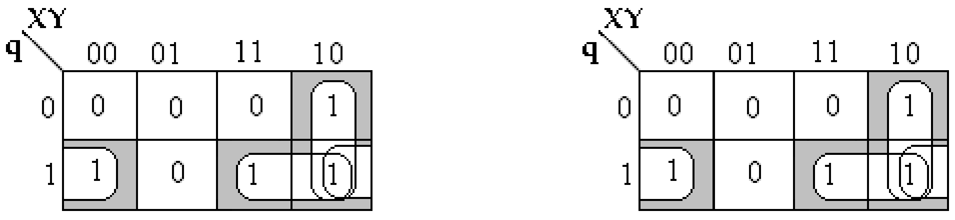
Here we assign a common letter symbol to all the stable states in each merged row. Thus, the symbol B & D is replaced by A; E & F are replaced by C.

	XY 00	01	11	10
A	Ⓐ,0	Ⓐ,0	Ⓐ,0	C,–
C	Ⓒ,1	A,–	Ⓒ,1	Ⓒ,1

Step 5:

Find the race-free binary assignment for the four stable states in the reduced flow table. Assign A= 0 and C= 1

Substituting the binary assignment into the reduced flow table, the transition table is obtained. The output map is obtained from the reduced flow table.

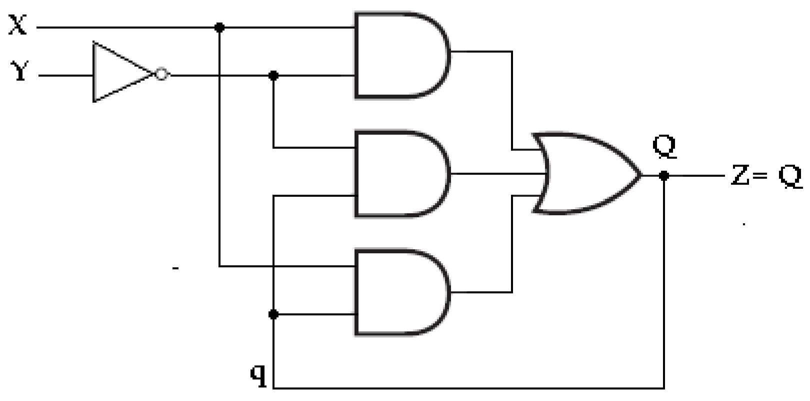


Q= qY' + XY' + qX

Z= Q

Transition table and output map

Step 6:



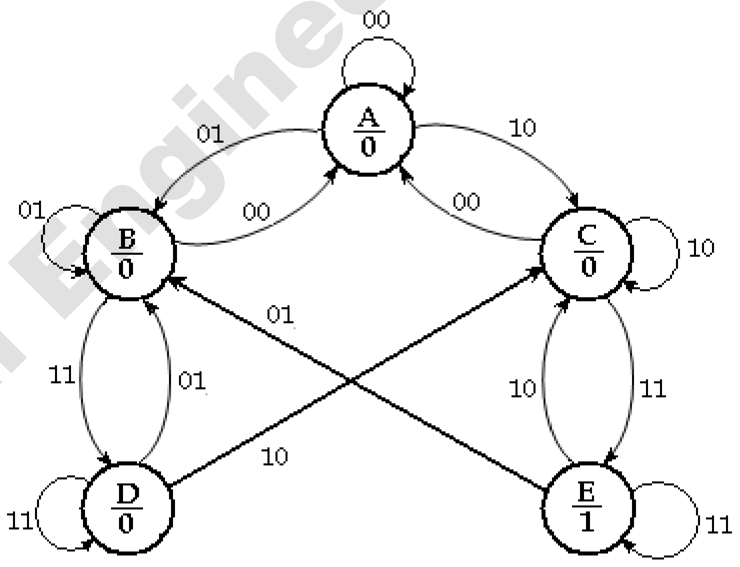
Gated-Latch Logic diagram

4. Design a circuit with inputs X and Y to give an output Z= 1 when XY= 11 but only if X becomes 1 before Y, by drawing total state diagram, primitive flow table and output map in which transient state is included.

Soln:

Step 1:

The state diagram can be drawn as,



State table

Step 2:

A primitive flow table is constructed from the state table as,

	XY 00	01	11	10
A	Ⓐ,0	B,–	–,–	C,–
B	A,–	Ⓑ,0	D,–	–,–
C	A,–	–,–	E,–	Ⓒ,0
D	–,–	B,–	Ⓓ,0	C,–
E	–,–	B,–	Ⓔ,1	C,–

Primitive flow table

Step 3:

The rows in the primitive flow table are merged by first obtaining all compatible pairs of states. This is done by means of the implication table.

B	✓			
C	✓	D,E ×		
D	✓	✓	D,E ×	
E	✓	D,E ×	✓	D,E ×
	A	B	C	D

Implication table

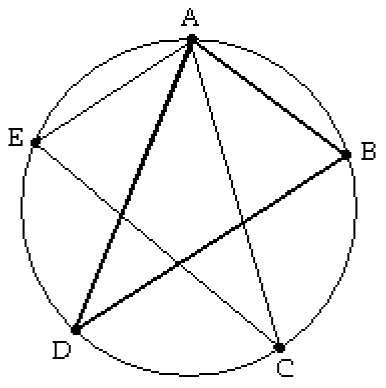
The squares that contain the check marks (✓) define the compatible pairs:

(A, B) (A, C) (A, D) (A, E) (B, D) (C, E)

Step 4:

The merger diagram is obtained from the list of compatible pairs derived from the implication table. There are six straight lines connecting the dots, one for each compatible pair. The lines form a geometrical pattern consisting of one triangle connecting (A, B, D) & a line (C, E). The maximal compatibles are:

(A, B, D) (C, E)



Merger diagram

The reduced flow table is drawn as below.

	XY 00	01	11	10
A,B,D	Ⓐ,0	Ⓑ,0	Ⓓ,0	C,–
C,E	A,–	B,–	Ⓔ,1	Ⓒ,0

Reduced flow table

Here we assign a common letter symbol to all the stable states in each merged row. Thus, the symbol B & D is replaced by A; E is replaced by C.

	XY 00	01	11	10
A	Ⓐ,0	Ⓐ,0	Ⓐ,0	C,–
C	A,–	A,–	Ⓒ,1	Ⓒ,0

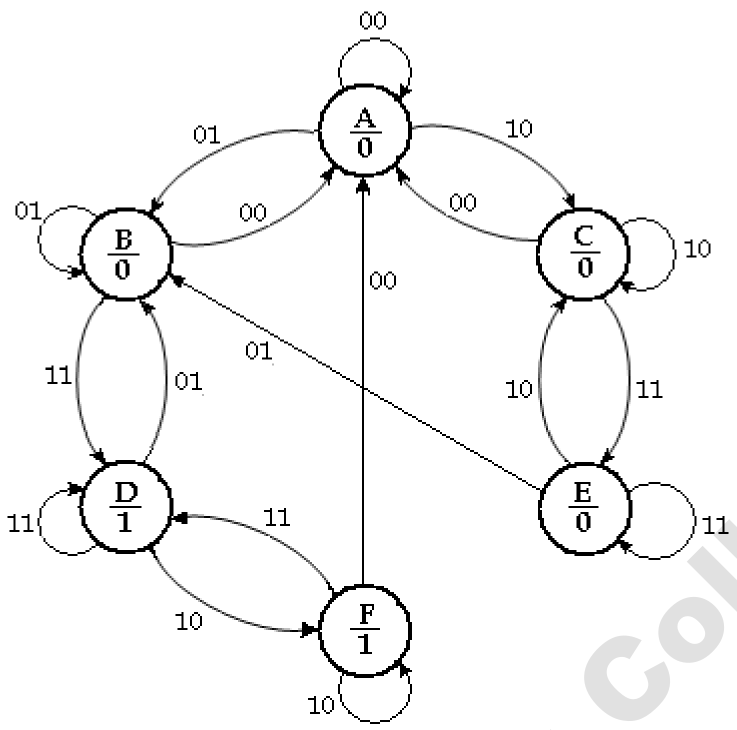
Transition table

5. Design a circuit with primary inputs A and B to give an output Z equal to 1 when A becomes 1 if B is already 1. Once Z= 1 it will remain so until A goes to 0. Draw the total state diagram, primitive flow table for designing this circuit.

Soln:

Step 1:

The state diagram can be drawn as,



State diagram

Step 2:

A primitive flow table is constructed from the state table as,

	AB 00	01	11	10
A	Ⓐ, 0	B, –	–, –	C, –
B	A, –	Ⓑ, 0	D, –	–, –
C	A, –	–, –	E, –	Ⓒ, 0
D	–, –	B, –	Ⓓ, 1	F, –
E	–, –	B, –	Ⓔ, 0	C, –
F	A, –	–, –	D, –	Ⓕ, 1

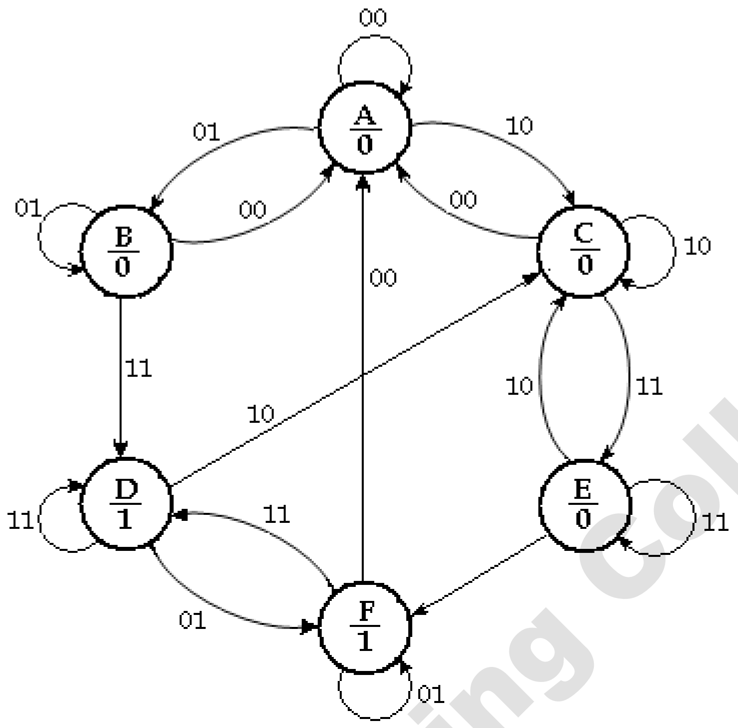
Primitive flow table

6. Design an asynchronous sequential circuit that has two inputs X_2 and X_1 and one output Z. When $X_1 = 0$, the output Z is 0. The first change in X_2 that occurs while X_1 is 1 will cause output Z to be 1. The output Z will remain 1 until X_1 returns to 0.

Soln:

Step 1:

The state diagram can be drawn as,



State diagram

Step 2:

A primitive flow table is constructed from the state table as,

	X_2X_1			
	00	01	11	10
A	Ⓐ,0	B,–	–,–	C,–
B	A,–	Ⓑ,0	D,–	–,–
C	A,–	–,–	E,–	Ⓒ,0
D	–,–	F,–	Ⓓ,1	C,–
E	–,–	F,–	Ⓔ,0	C,–
F	A,–	Ⓕ,1	D,–	–,–

Primitive flow table

Step 3:

The rows in the primitive flow table are merged by obtaining all compatible pairs of states. This is done by means of the implication table.

B	✓				
C	✓	D,E ×			
D	B,F ×	B,F ×	D,F ×		
E	B,F ×	B,F × D,E ×	✓	D,E ×	
F	B,F ×	B,F ×	D,E ×	✓	D,E ×
	A	B	C	D	E

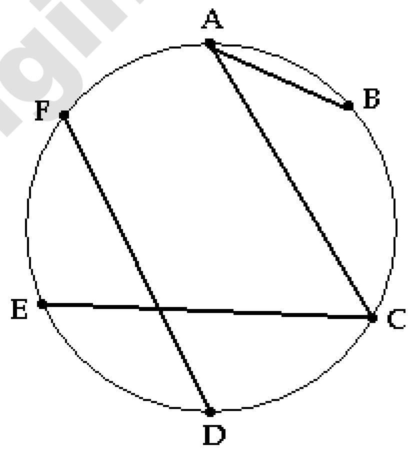
Implication table

The squares that contain the check marks (✓) define the compatible pairs:

(A, B) (A, C) (C, E) (D, F)

Step 4:

The merger diagram is obtained from the list of compatible pairs derived from the implication table. There are four straight lines connecting the dots, one for each compatible pair. It consists of four lines (A, B), (A, C), (C, E) and (D, F).



Merger diagram

The maximal compatibles are:

(A, B) **(C, E)** **(D, F)**

This set of maximal compatible covers all the original states resulting in the reduced flow table.

The reduced flow table is drawn as below.

	X_2X_1			
	00	01	11	10
A, B	Ⓐ,0	Ⓑ, 0	D, –	C, –
C, E	A, –	F, –	Ⓔ, 0	Ⓒ, 0
D, F	A, –	Ⓕ, 1	Ⓓ, 1	C, –

Flow table

Here we assign a common letter symbol to all the stable states in each merged row. Thus, the symbol B is replaced by A; E is replaced by C and F is replaced by D.

Step 5:

	X_2X_1			
	00	01	11	10
A	Ⓐ,0	Ⓐ,0	D, –	C, –
C	A, –	D, –	Ⓒ, 0	Ⓒ, 0
D	A, –	Ⓓ, 1	Ⓓ, 1	C, –

Reduced Flow table

Find the race-free binary assignment for the four stable states in the reduced flow table. Assign A= S_0, C= S_1 and D= S_2.

	X_2X_1			
	00	01	11	10
S_0	$\textcircled{S_0}$,0	$\textcircled{S_0}$,0	S_2, –	S_1, –
S_1	S_0, –	S_2, –	$\textcircled{S_1}$, 0	$\textcircled{S_1}$, 0
S_2	S_0, –	$\textcircled{S_2}$, 1	$\textcircled{S_2}$, 1	S_1, –

Now, if we assign S_0= 00, S_1 = 01 and S_2 = 10, then we need one more state S_3= 11 to prevent critical race during transition of $S_0 \rightarrow S_1$ or $S_2 \rightarrow S_1$. By introducing S_3 the transitions $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$ are routed through S_4.

Thus after state assignment the flow table can be given as,

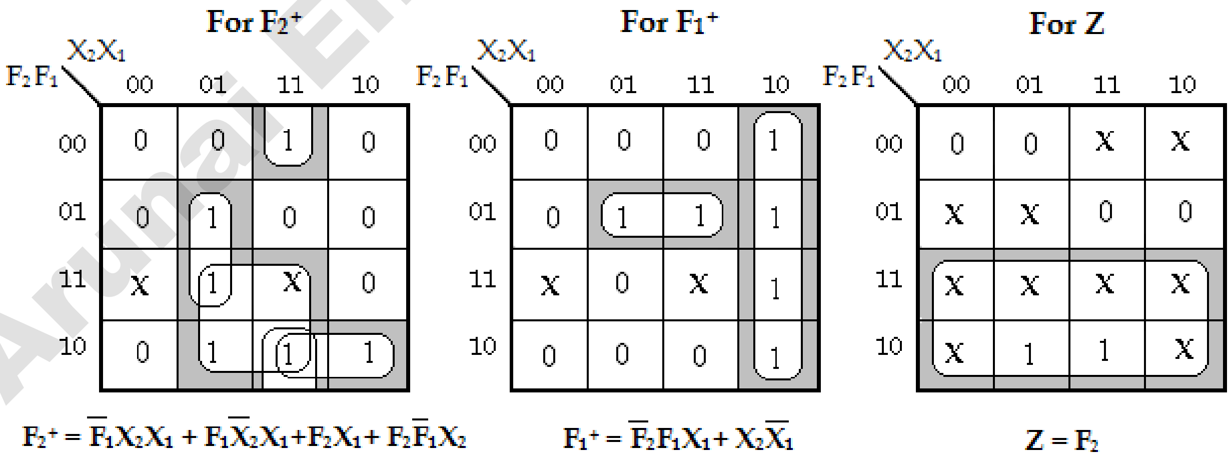
Present State F_2F_1	Next state for Inputs X_2X_1, Output 00	01	11	10
$S_0 \rightarrow 00$	$\textcircled{S_0}$, 0	$\textcircled{S_0}$, 0	S_2, –	S_1, –
$S_1 \rightarrow 01$	S_0, –	S_3, –	$\textcircled{S_1}$, 0	$\textcircled{S_1}$, 0
$S_2 \rightarrow 10$	S_0, –	$\textcircled{S_2}$, 1	$\textcircled{S_2}$, 1	S_3, –
$S_3 \rightarrow 11$	–, –	S_2, –	–, –	S_1, –

Flow table with state assignment

Substituting the binary assignment into the reduced flow table, the transition table is obtained. The output map is obtained from the reduced flow table.

Present State F_2 F_1	Next state for Inputs X_2X_1, Output 00	01	11	10
0 0	$\textcircled{00}$, 0	$\textcircled{00}$, 0	10, –	01, –
0 1	00, –	11, –	$\textcircled{01}$, 0	$\textcircled{01}$, 0
1 0	00, –	$\textcircled{10}$, 1	$\textcircled{10}$, 1	11, –
1 1	–, –	10, –	–, –	01, –

K- Map simplification:



For F_2^+

F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	X	1	X	0
10	0	1	1	1

$F_2^+ = \overline{F_1}X_2X_1 + F_1\overline{X_2}X_1 + F_2X_1 + F_2\overline{F_1}X_2$

For F_1^+

F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	0	1
01	0	1	1	1
11	X	0	X	1
10	0	0	0	1

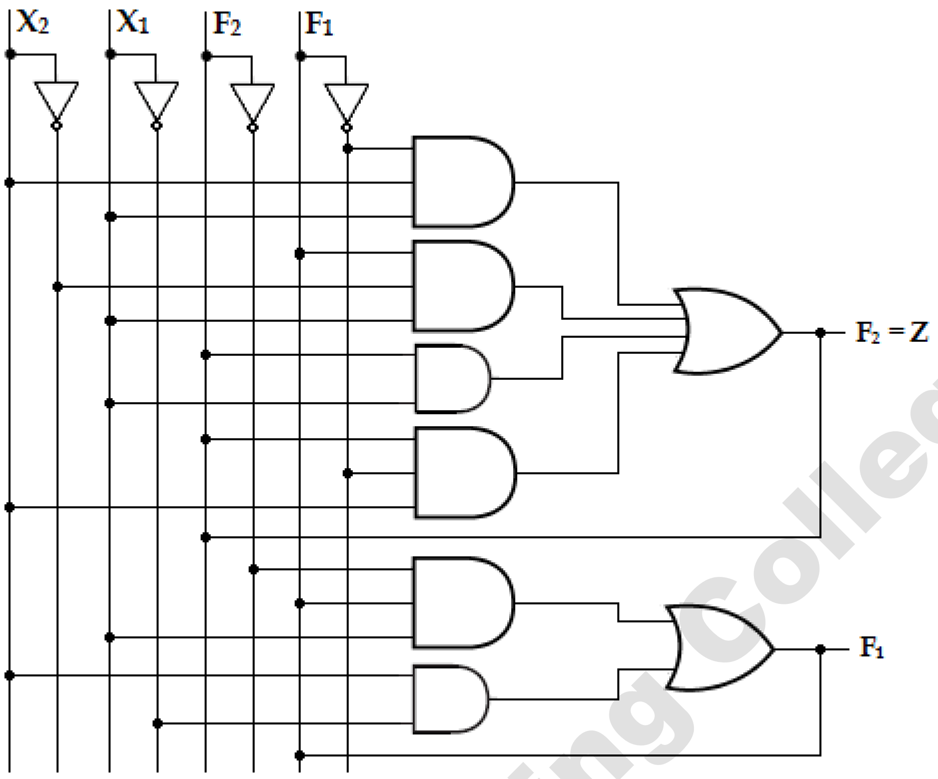
$F_1^+ = \overline{F_2}F_1X_1 + X_2\overline{X_1}$

For Z

F_2F_1 \ X_2X_1	00	01	11	10
00	0	0	X	X
01	X	X	0	0
11	X	X	X	X
10	X	1	1	X

$Z = F_2$

Logic Diagram:



7. Obtain a primitive flow table for a circuit with two inputs x_1 and x_2 and two outputs z_1 and z_2 that satisfies the following four conditions.

   i. When $x_1x_2 = 00$, output $z_1z_2 = 00$.

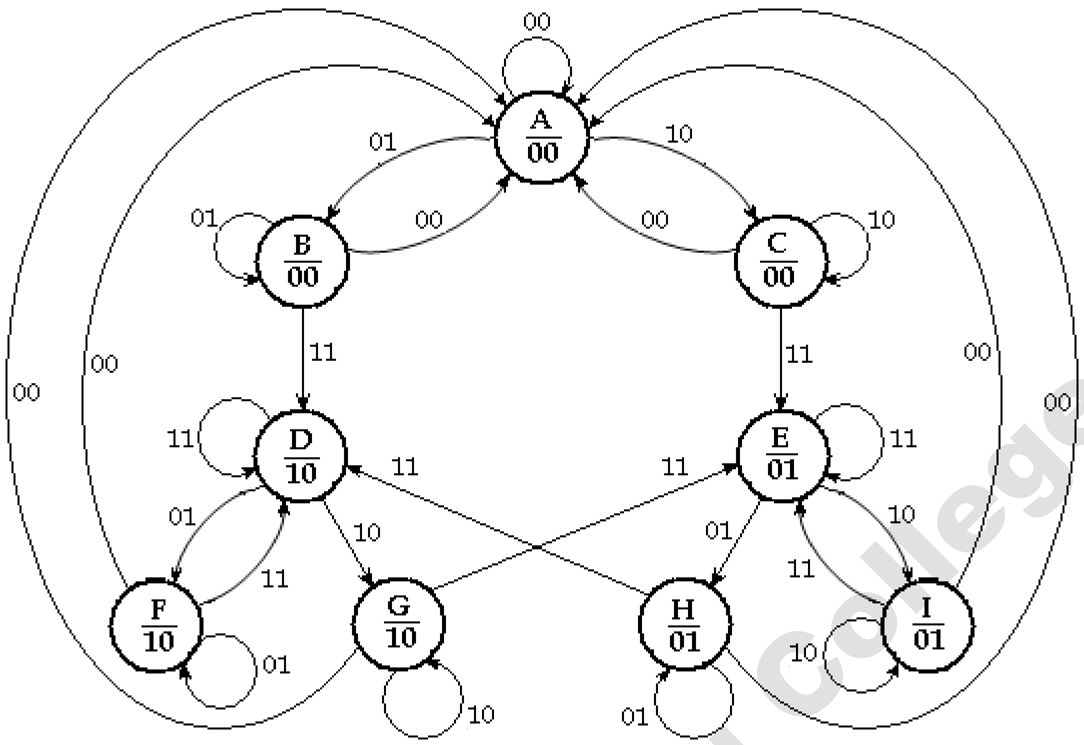
   ii. When $x_1 = 1$ and x_2 changes from 0 to 1, the output $z_1z_2 = 01$.

   iii. When $x_2 = 1$ and x_1 changes from 0 to 1, the output $z_1z_2 = 10$.

   iv. Otherwise the output does not change.

Soln:

The state diagram can be drawn as,



State diagram

Step 2: A primitive flow table is constructed from the state table as,

	X_1X_2			
	00	01	11	10
A	Ⓐ,00	B,–	–,–	C,–
B	A,–	Ⓑ,00	D,–	–,–
C	A,–	–,–	E,–	Ⓒ,00
D	–,–	F,–	Ⓓ,10	G,–
E	–,–	H,–	Ⓔ,01	I,–
F	A,–	Ⓕ,10	D,–	–,–
G	A,–	–,–	E,–	Ⓖ,10
H	A,–	Ⓗ,01	D,–	–,–
I	A,–	–,–	E,–	Ⓘ,01

Primitive flow table

4.7 HAZARDS

Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.

Hazards occur in combinational circuits, where they may cause a temporary false-output value. When this condition occurs in asynchronous sequential circuits, it may result in a transition to a wrong stable state.

Hazards in Combinational Circuits:

A hazard is a condition where a single variable change produces a momentary output change when no output change should occur.

Types of Hazards:

- Static hazard
- Dynamic hazard

4.7.1 Static Hazard

In digital systems, there are only two possible outputs, a '0' or a '1'. The hazard may produce a wrong '0' or a wrong '1'. Based on these observations, there are three types,

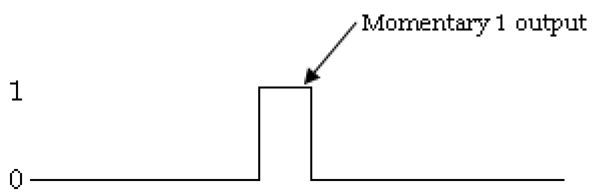
- Static- 0 hazard,
- Static- 1 hazard,

Static- 0 hazard:

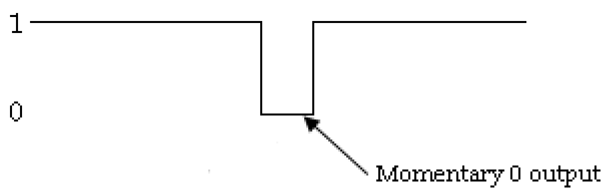
When the output of the circuit is to remain at 0, and a momentary 1 output is possible during the transmission between the two inputs, then the hazard is called a static 0-hazard.

Static- 1 hazard:

When the output of the circuit is to remain at 1, and a momentary 0 output is possible during the transmission between the two inputs, then the hazard is called a static 1-hazard.



(a) Static 0-hazard

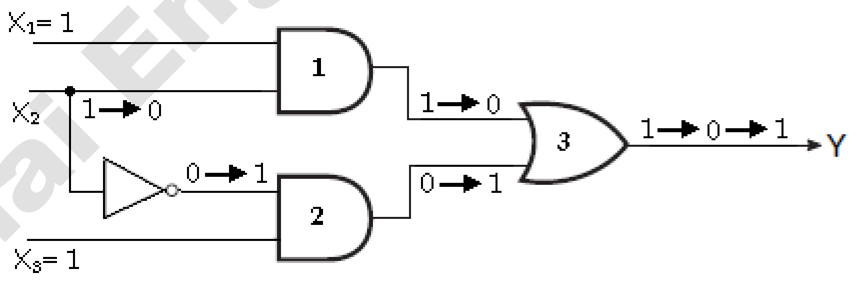


(b) Static 1-hazard

The below circuit demonstrates the occurrence of a static 1-hazard. Assume that all three inputs are initially equal to 1 i.e., $X_1X_2X_3$= 111. This causes the output of the gate 1 to be 1, that of gate 2 to be 0, and the output of the circuit to be equal to 1. Now consider a change of X_2 from 1 to 0 i.e., $X_1X_2X_3$= 101. The output of gate 1 changes to 0 and that of gate 2 changes to 1, leaving the output at 1. The output may momentarily go to 0 if the propagation delay through the inverter is taken into consideration.

The delay in the inverter may cause the output of gate 1 to change to 0 before the output of gate 2 changes to 1. In that case, both inputs of gate 3 are momentarily equal to 0, causing the output to go to 0 for the short interval of time that the input signal from X_2 is delayed while it is propagating through the inverter circuit.

Thus, a static 1-hazard exists during the transition between the input states $X_1X_2X_3$= 111 and $X_1X_2X_3$= 101.

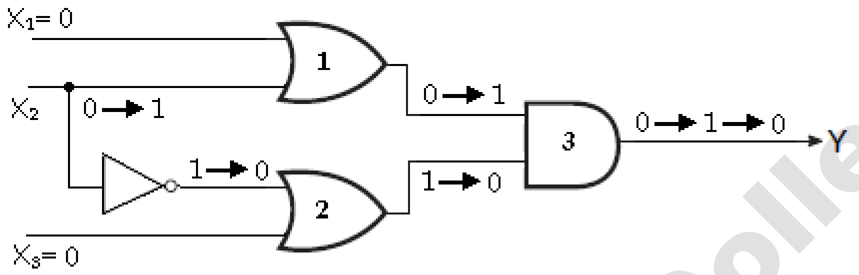


Circuit with static-1 hazard

Now consider the below network, and assume that the inverter has an appreciably greater propagation delay time than the other gates. In this case there is a static 0-hazard in the transition between the input states $X_1X_2X_3$= 000 and $X_1X_2X_3$= 010 since it is possible for a logic-1 signal to appear at both input terminals of the AND gate for a short duration.

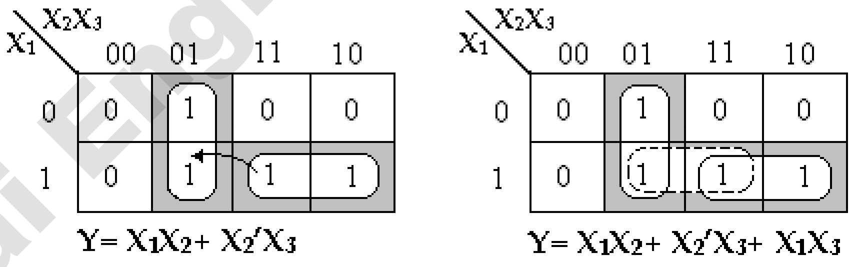
The delay in the inverter may cause the output of gate 1 to change to 1 before the output of gate 2 changes to 0. In that case, both inputs of gate 3 are momentarily equal to 0, causing the output to go to 1 for the short interval of time that the input signal from X_2 is delayed while it is propagating through the inverter circuit.

Thus, a static 0-hazard exists during the transition between the input states $X_1X_2X_3 = 000$ and $X_1X_2X_3 = 010$.



Circuit with static-0 hazard

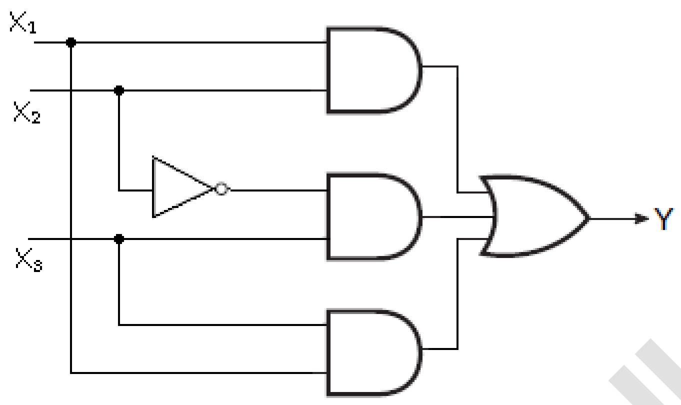
A hazard can be detected by inspection of the map of the particular circuit. To illustrate, consider the map in the circuit with static 0-hazard, which is a plot of the function implemented. The change in X_2 from *1* to 0 moves the circuit from minterm 111 to minterm 101. The hazard exists because the change in input results in a different product term covering the two minterrns.



Maps demonstrating a Hazard and its Removal

The minterm 111 is covered by the product term implemented in gate 1 and minterm 101 is covered by the product term implemented in gate 2. Whenever the circuit must move from one product term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to an undesirable 0 output.

The remedy for eliminating a hazard is to enclose the two minterms in question with another product term that overlaps both groupings. This situation is shown in the *map* above, where the two terms that causes the hazard are combined into one product term. The hazard- free circuit obtained by this combinational is shown below.



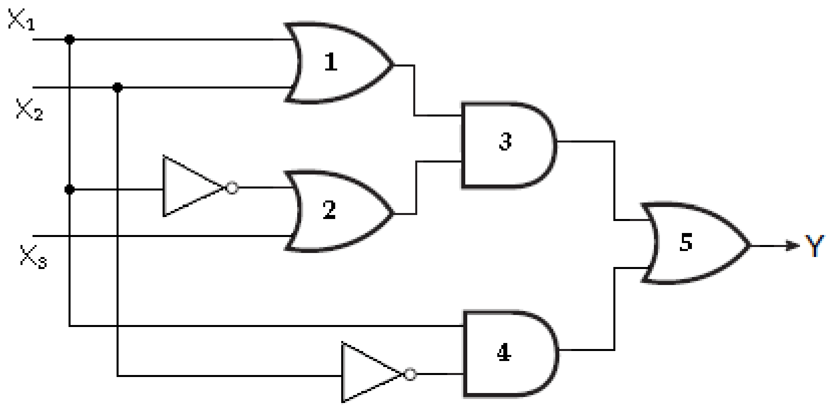
Hazard-free Circuit

The extra gate in the circuit generates the product term X_1X_4. The hazards in combinational circuits can be removed by covering any two minterms that may produce a hazard with a product term common to both. The removal of hazards requires the addition of redundant gates to the circuit.

4.7.2 Dynamic Hazard

A dynamic hazard is defined as a transient change occurring three or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two input states differing in the value of one variable.

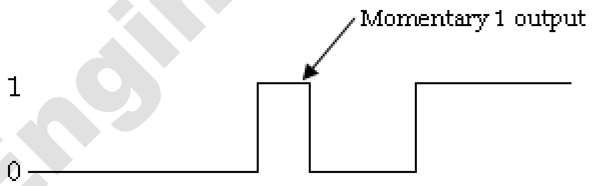
Now consider the input states $X_1X_2X_3$= 000 and $X_1X_2X_3$= 100. For the first input state, the steady state output is 0; while for the second input state, the steady state output is 1. To facilitate the discussion of the transient behavior of this network, assume there are no propagation delays through gates G_3 and G_5 and that the propagation delays of the other three gates are such that G_1 can switch faster than G_2 and G_2 can switch faster than G_4.



Circuit with Dynamic hazard

When X_1 changes from 0 to 1, the change propagates through gate G_1 before gate G_2 with the net effect that the inputs to gate G_3 are simultaneously 1 and the network output changes from 0 to 1. Then, when X_1 change propagates through gate G_2, the lower input to gate G_3 becomes 0 and the network output changes back to 0.

Finally, when the X_1= 1 signal propagates through gate G_4, the lower input to gate G_5 becomes 1 and the network output again changes to 1. It is therefore seen that during the change of X_1 variable from 0 to 1 the output undergoes the sequence, $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$, which results in three changes when it should have undergone only a single change.



Dynamic hazard

4.7.3 Essential Hazard

An essential hazard is caused by unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard.

Essential hazards elimination:

Essential hazards can be eliminated by adjusting the amount of delays in the affected path. To avoid essential hazards, each feedback loop must be handled with individual care to ensure that the delay in the feedback path is long enough compared with delays of other signals that originate from the input terminals.

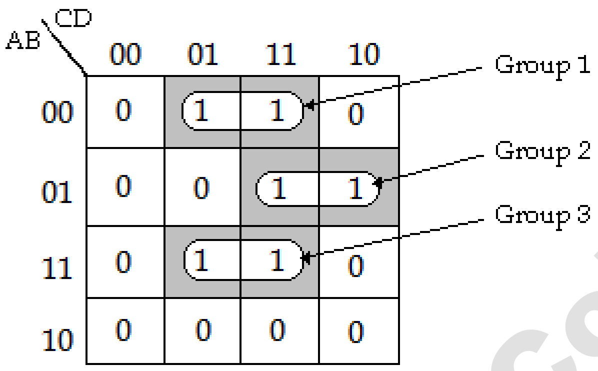
4.8 Design Of Hazard Free Circuits

1. Design a hazard-free circuit to implement the following function.

F (A, B, C, D) = ∑m (1, 3, 6, 7, 13, 15)

Soln:

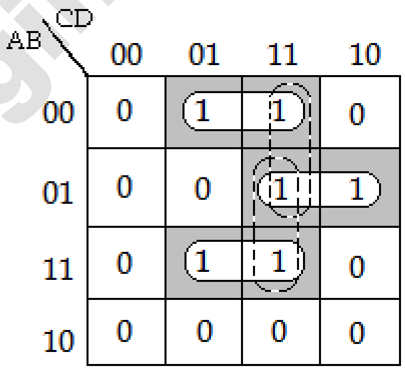
a) K-map Implementation and grouping



F=A'B'D+ A'BC+ ABD

b) Hazard- free realization

The first additional product term A'CD, overlapping two groups (group 1 & 2) and the second additional product term, BCD, overlapping the two groups (group 2 & 3).



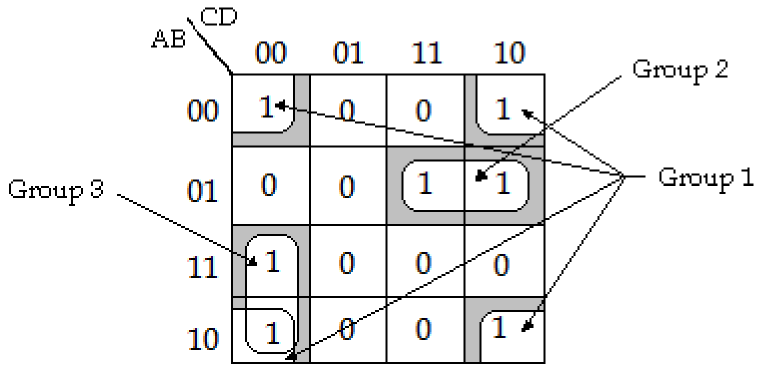
F=A'B'D+ A'BC+ ABD+ A'CD+ BCD

2. Design a hazard-free circuit to implement the following function.

F (A, B, C, D) = ∑m (0, 2, 6, 7, 8, 10, 12).

Soln:

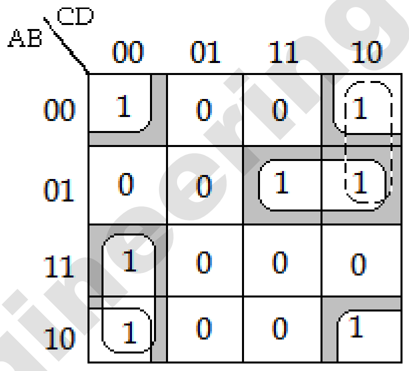
a) K-map Implementation and grouping



F= B′D′+ A′BC+ AC′D′

b) Hazard- free realization

The additional product term, A′CD′ overlapping two groups (group 1 & 2) for hazard free realization. Group 1 and 3 are already overlapped hence they do not require additional minterm for grouping.

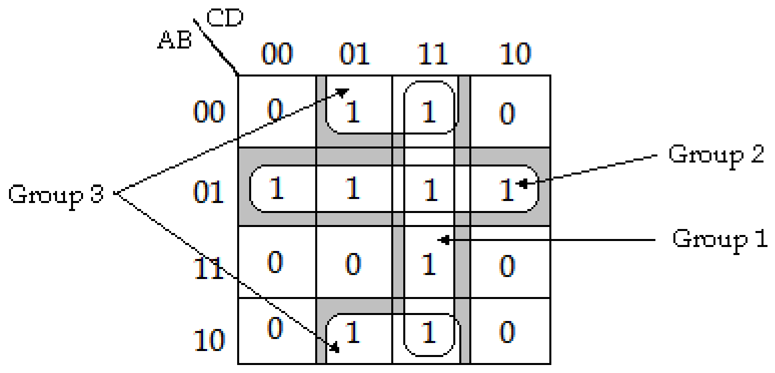


F= B′D′+ A′BC+ AC′D′+ A′CD′

3. Design a hazard-free circuit to implement the following function.

F (A, B, C, D) = ∑m (1, 3, 4, 5, 6, 7, 9, 11, 15).

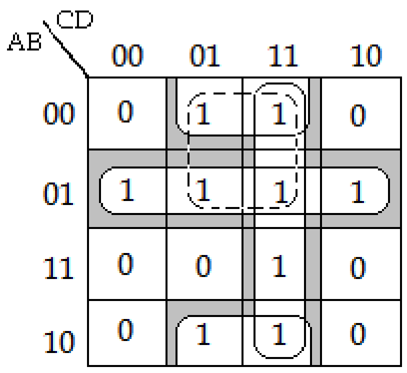
a) K-map Implementation and grouping



F= CD+ A′B+ B′D

b) <u>Hazard- free realization</u>

The additional product term, A′D overlapping two groups (group 2 & 3) for hazard free realization. Group 1 and 2 are already overlapped hence they do not require additional minterm for grouping.



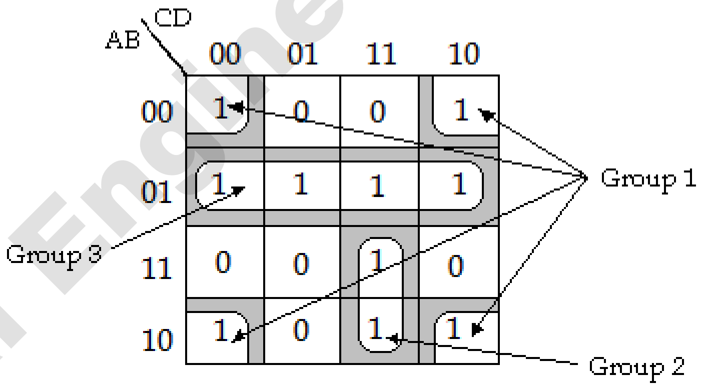
F= CD+ A′B+ B′D+ A′D

4. Design a hazard-free circuit to implement the following function.

   F (A, B, C, D) = ∑m (0, 2, 4, 5, 6, 7, 8, 10, 11, 15).

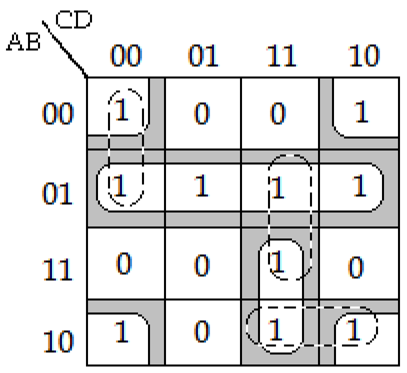
Soln:

a) <u>K-map Implementation and grouping</u>



F= B′D′+ A′B+ ACD

b) <u>Hazard- free realization</u>

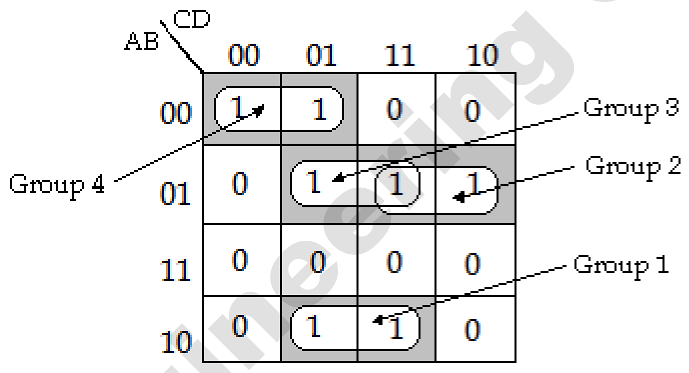


F= B′D′+ A′B+ ACD+ A′C′D′+ BCD+ AB′C

5. Design a hazard-free circuit to implement the following function.

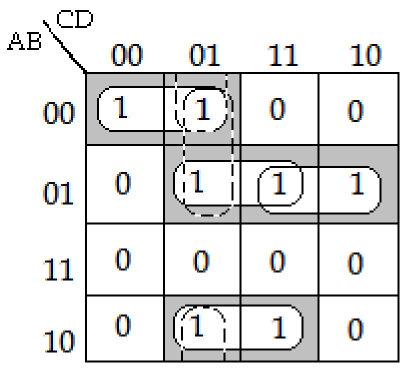
   F (A, B, C, D) = $\sum$m (0, 1, 5, 6, 7, 9, 11).

a) <u>K-map Implementation and grouping</u>



F= AB′D+ A′BC+ A′BD+ A′B′C′

b) <u>Hazard- free realization:</u>



F= AB′D+ A′BC+ A′BD+ A′B′C′+ A′C′D+ B′C′D