UNIT - 4

ASYNCHRONOUS SEQUENTIAL CIRCUITS AND PROGRAMMABILITY LOGIC DEVICES

4.1 ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

In synchronous sequential circuits, the change of state occurs in response to the synchronized clock signal. But in asynchronous sequential circuits the change of state occurs when there is a change in input variable.

The memory elements of synchronous sequential circuits are clocked flip-flops. The memory elements of asynchronous sequential circuits are either unclocked flip-flops or time delay elements. The asynchronous sequential circuit consists of a combinational circuit and delay elements connected to form feedback loops. There are 'n' input variables, 'm' output variables, 'k' present states and 'k' next states.

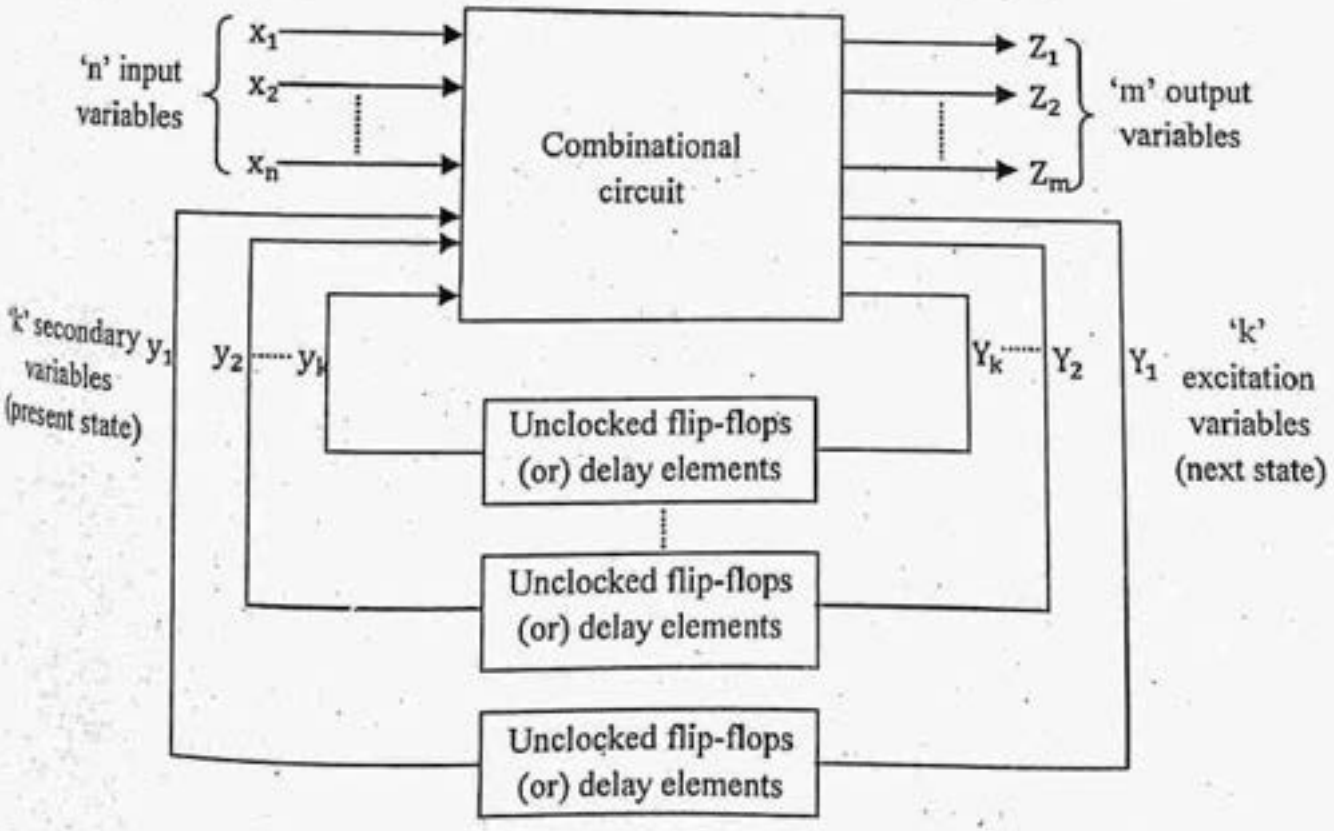


Figure 4.1 Asynchronous sequential circuit

The present states are called secondary variables. The next states are cal excitation variables. When an input variable 'x' changes, the secondary variable do not change instantaneously. It takes a certain amount of time because the sig has to propagate through the combinational circuit and delay elements. Hence time between two input changes is kept longer so that the circuit reaches a sta state.

After reaching a steady state condition both secondary variables (y) excitation variable (Y) are same. This is known to be a stable state. So asynchronous sequential circuits, because of unequal delays in the wires combinational circuits, it is impossible to have two or more input variable change exactly the same instant of time. Therefore simultaneous changes of two or m input variables are usually avoided. Only one input variable is allowed to change time.

Based on the input signals, the asynchronous sequential circuit is classified into,

a. Fundamental mode sequential circuit.
b. Pulse mode sequential circuit.

Assumptions that must be made for the fundamental mode circuit are,

1. The input variable changes only when the circuit is stable.
2. Only one input variable can change at a given time.
3. Inputs are levels and not pulses.

Assumptions that must be made for the pulse mode circuit are,

1. The input variables are pulses instead of levels.
2. The width of the pulses is long enough for the circuit to respond to the inpu
3. The pulse width must not be so long that it is still present after the new s is reached.

4.2 ANALYSIS OF ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS.

4.2.1 Analysis of Fundamental mode sequential circuits

The behavior of an asynchronous sequential circuit can be found out from inputs, outputs and its states. The steps to analyze the asynchronous sequen circuit are,

Step 1: Determine the next state equations and output equations. If there are any flip-flops, use the respective characteristic equation to find the next state equations and output equations.

Step 2: Plot the truth table.

The truth table should contain,

a. Present state
b. Input
c. Next state
d. Output

Step 3: From the truth table, plot the transition table.

Step 4: Assign states for the binary values and plot the state table.

Step 5: Plot the output map.

Step 6: Plot the state diagram.

Consider the example shown in figure 4.2.

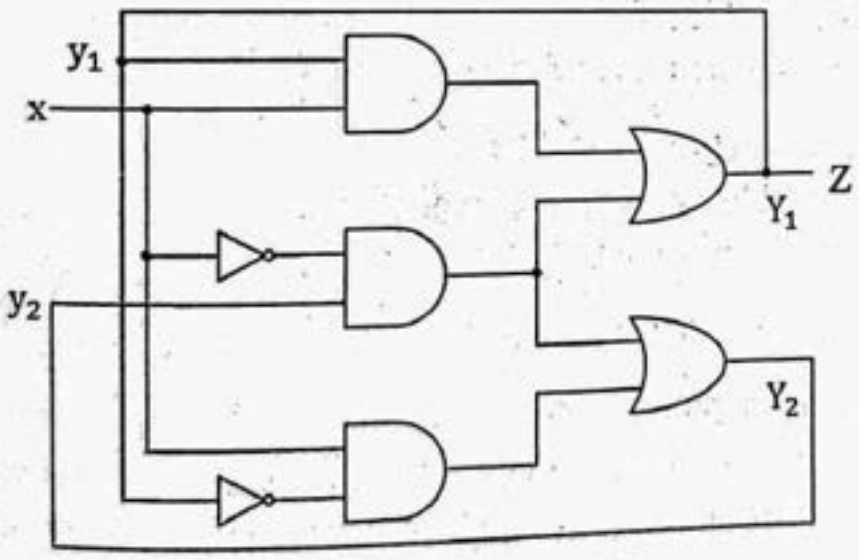


Figure 4.2

Step 1: Determine the next state equations and output equations.

Here the input is x, Present states are y_1 and y_2, Next states are Y_1 and Y_2 and Output is Z

Next state equations:

$$Y_1 = y_1x + \bar{x}y_2$$

$$Y_2 = \bar{x}y_2 + x\bar{y}_1$$

Output equation: $Z = Y_1$ or $Z = y_1x + \bar{x}y_2$

Step 2: Plot the truth table.

The truth table should contain, Present states (y_1,y_2), input (x), next states (Y_1, Y_2) and output (Z). Here the number of inputs is less than the number of present states. So write the input term first in the truth table which as shown in table 4.1.

First write the possible binary combination for present states and inputs. Then by substituting the value of x, y_2 and y_1 in the next state and output equations, find next states Y_2, Y_1 and output Z. For the given input, if the next state Y_2Y_1 is same as that of y_2y_1 then the state is said to be stable.

Input	Present states		Next states		Output	States
x	y_2	y_1	Y_2	Y_1	Z	
0	0	0	0	0	0	Stable
0	0	1	0	0	0	Unstable
0	1	0	1	1	1	Unstable
0	1	1	1	1	1	Stable
1	0	0	1	0	0	Unstable
1	0	1	0	1	1	Stable
1	1	0	1	0	0	Stable
1	1	1	0	1	1	Unstable

Table 4.1 Truth table

Step 3: Plot the transition table

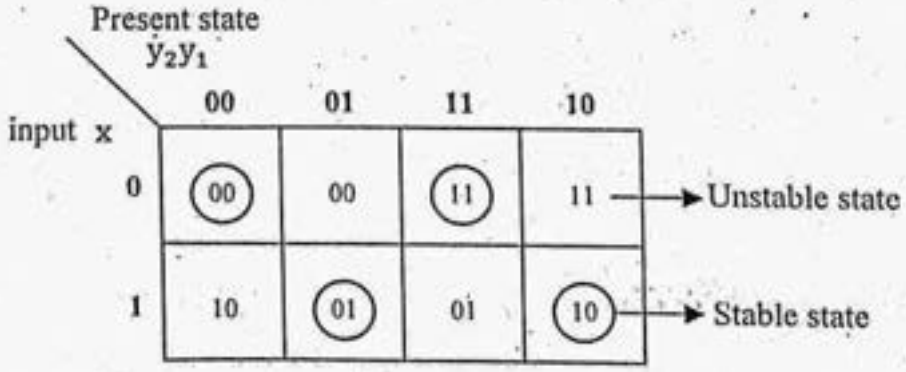


Figure 4.3 Transition table

The rows of transition table represent input 'x' and columns of transition table represent present state y_2y_1. The number written in the table represents next state Y_2Y_1. The circle around Y_2Y_1 indicate that the state is stable (present state and next state are same).

Step 4: Assign states for the binary values and plot the state table

Here present states and next states are 00, 01, 10 and 11. So assign a=00, b=01, c=10 and d=11. The state table can be drawn as shown in Figure 4.4.

	Present state			
input x	a	b	d	c
0	(a)	a	(d)	d
1	c	(b)	b	(c)

Figure 4.4 State table

Step 5: Plot the output map

	a	b	d	c
x				
0	0	-	1	-
1	-	1	-	0

Figure 4.5 Output map

The output is mapped for all stable states which are indicated by circles as shown in figure 4.5. For unstable state, the output is mapped unspecified. By using the state table and output map, draw the next state transition for input x=0 and x=1 as shown in figure 4.6. Similarly mark the output Z from the output map for each stable state.

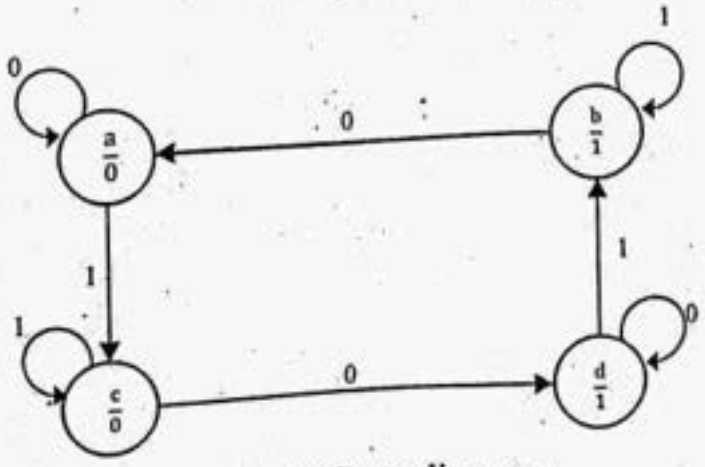


Figure 4.6 State diagram

4.2.2 Analysis of Pulse mode sequential circuits

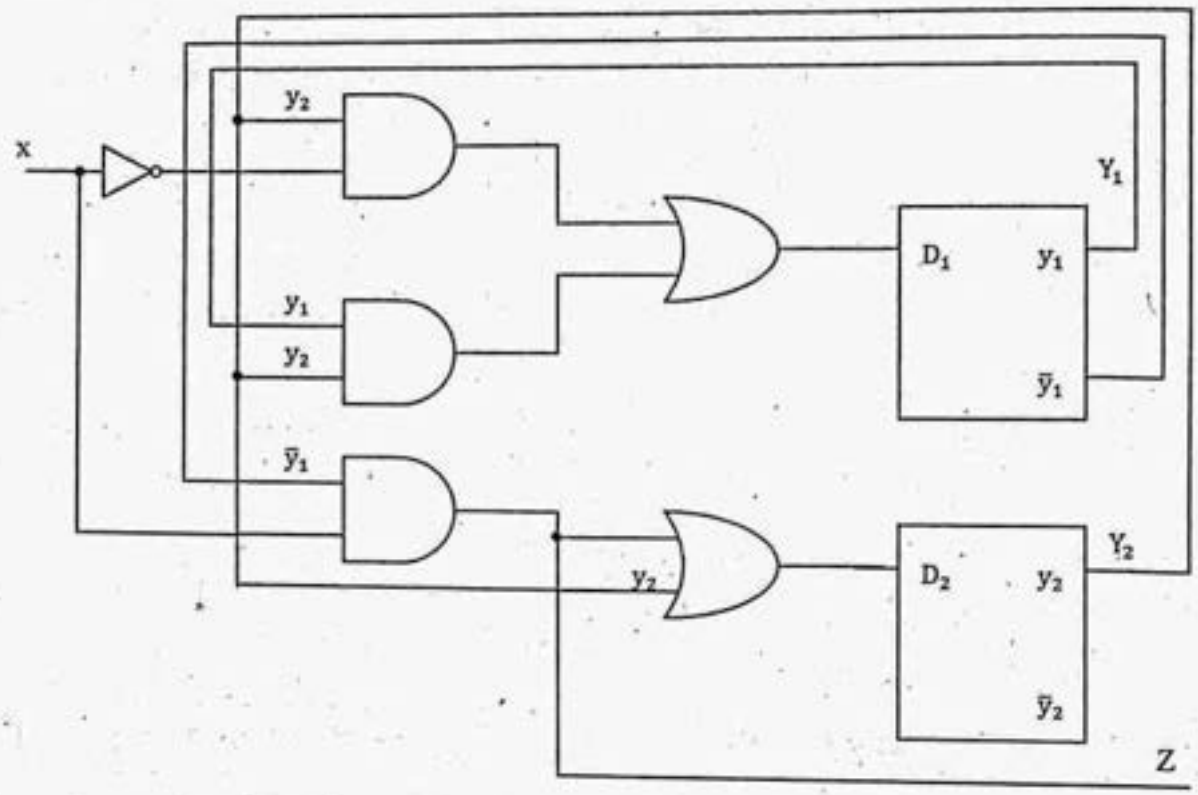


Figure 4.7

The analysis of pulse mode sequential circuit is same as that of fundamental mode sequential circuit. Consider the circuit shown in figure 4.7.

Step 1: Determine the next state equations and output equations.

Here the input is 'x', Present states are y_1 and y_2, Next states are Y_1 and Y_2 and the Output is Z.

Next state equations:

$$Y_1 = D_1 \text{ (Characteristic equation of D flip-flop)}$$

$$Y_2 = D_2$$

Here $D_1 = y_2\bar{x} + y_1y_2$ and $D_2 = x\bar{y}_1 + y_2$

Therefore, $Y_1 = y_2\bar{x} + y_1y_2$

$$Y_2 = x\bar{y}_1 + y_2$$

Output equation:

$$Z = x\bar{y}_1$$

Step 2: Plot the truth table.

The truth table should contain, Present states (y_1, y_2), input(x), Next states (Y_1, Y_2) and output (Z). Here the number of inputs is less than the number of present states. So write the input term first in the truth table as shown in table 4.2.

First write the possible binary combination for present state and inputs. Then by substituting the value of x, y_2 and y_1, find the next states Y_2, Y_1 and output Z. For the given input, if the next state Y_2Y_1 is same as that of present state y_2y_1, then the state is said to be stable.

Input	Present states		Next states		Output	States
x	y_2	y_1	Y_2	Y_1	Z	
0	0	0	0	0	0	Stable
0	0	1	0	0	0	Unstable
0	1	0	1	1	0	Unstable
0	1	1	1	1	0	Stable
1	0	0	1	0	1	Unstable
1	0	1	0	0	0	Unstable
1	1	0	1	0	1	Stable
1	1	1	1	1	0	Stable

Table 4.2 Truth table

Step 3: Plot the transition table

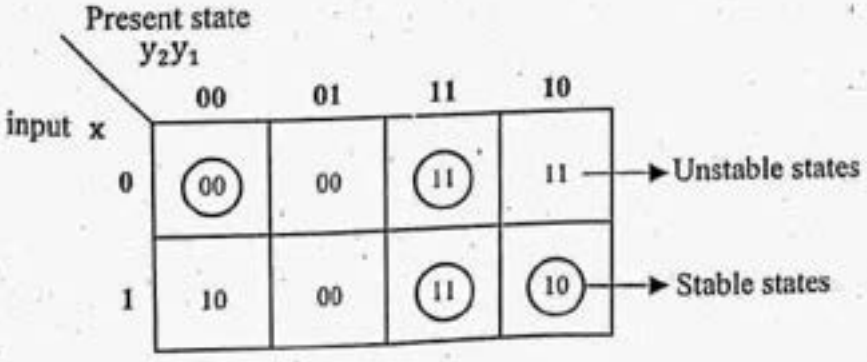


Figure 4.8 Transition table

The rows of transition table shown in figure 4.8 represent input 'x' and columns of transition table represent present state y_2y_1. The number written in the transition table represent next state Y_2Y_1. The circle around Y_2Y_1 indicate that the state is stable.

Step 4: Assign states for the binary values and plot the state table

Here present states and next states are 00, 01, 10 and 11.So assign a=00, b=01, c=10 and d=11. The state table can be drawn as shown in figure 4.9.

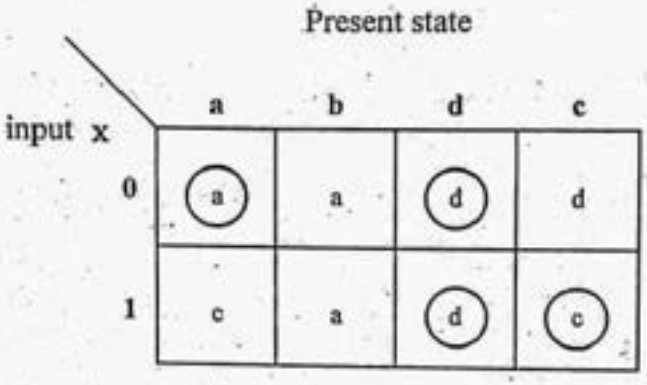


Figure 4.9 State table

Step 5 : Plot the output map

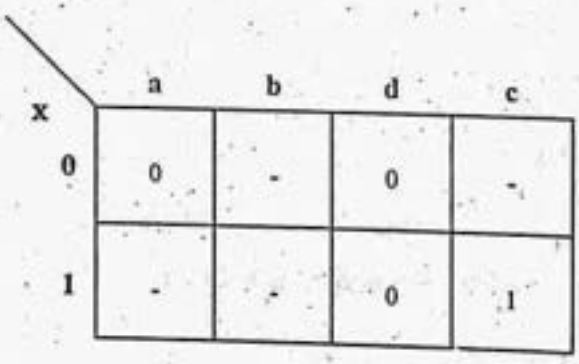


Figure 4.10 Output map

The output is mapped for all stable states which are indicated by circles as shown in figure 4.10. For the unstable states, the output is mapped unspecified.

Step 6: Plot the state diagram

By using the state table and output map, draw the next state transition for input x=0 and x=1 as shown in figure 4.11. Similarly mark the output Z from the output map for each stable state.

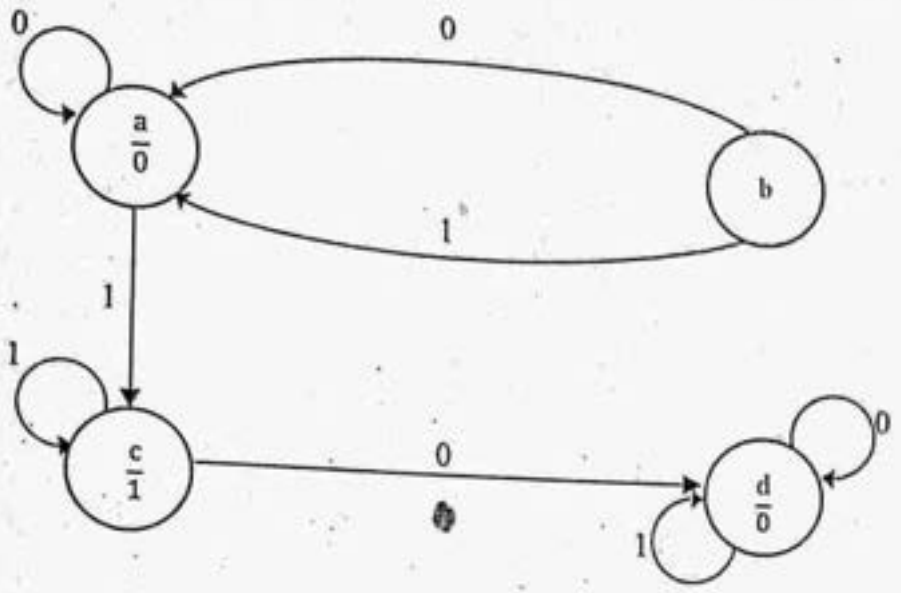


Figure 4.11 State diagram

Example 4.1: An asynchronous sequential circuit has two internal states and one output. The excitation and output function describing the circuit are as follows.

$$Y_1 = x_1x_2 + x_1y_2 + x_2y_1$$
$$Y_2 = x_2 + x_1y_1y_2 + x_1y_1$$
$$Z = x_2 + y_1$$

Solution:

Step 1: Since the equations are directly given proceed to step 2.

Step 2: Plot the truth table

Here y_2 and y_1 are present states, Y_2 and Y_1 are next states, x_2 and x_1 are inputs and Z is the output. Here the number of inputs is equal to number of present states, so we can write either inputs or present state first in the truth table as shown in table 4.3.

First write the possible binary combination for present states and inputs. Then by substituting the value of x_2, x_1, y_2, y_1 and Z in the next state and output equations, find the next states Y_2, Y_1 and output Z. For the given input, if the next states $Y_2\ Y_1$ is same as that of present states $y_2\ y_1$, then the state is said to be stable.

Input		Present state		Next state		Output	States
x_2	x_1	y_2	y_1	Y_2	Y_1	Z	
0	0	0	0	0	0	0	Stable
0	0	0	1	0	0	1	Unstable
0	0	1	0	0	0	0	Unstable
0	0	1	1	0	0	1	Unstable
0	1	0	0	0	0	0	Stable
0	1	0	1	1	0	1	Unstable
0	1	1	0	0	1	0	Unstable
0	1	1	1	1	1	1	Stable
1	0	0	0	1	0	1	Unstable
1	0	0	1	1	1	1	Unstable
1	0	1	0	1	0	1	Stable
1	0	1	1	1	1	1	Stable
1	1	0	0	1	1	1	Unstable
1	1	0	1	1	1	1	Unstable
1	1	1	0	1	1	1	Unstable
1	1	1	1	1	1	1	Stable

Table 4. 3 Truth table

Step 3: Plot the transition table.

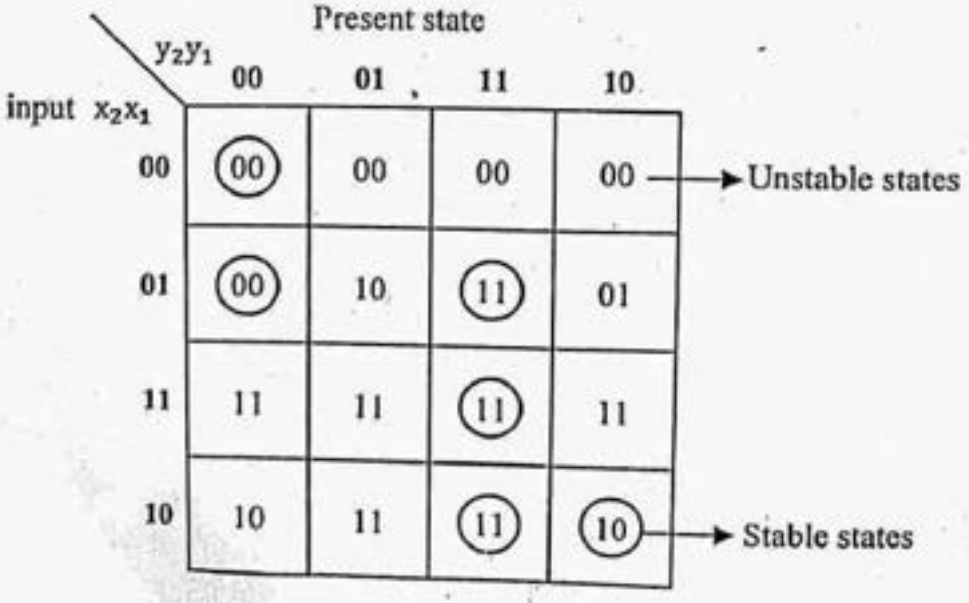


Figure 4.12 Transition table

Here the rows of transition table represent inputs x_2x_1 and columns of transition table represent present states y_2y_1. The number written in the table represent next state Y_2Y_1. The circle around Y_2Y_1 indicates that the state is stable which is shown in figure 4.12.

Step 4: Assign states for the binary values and plot the state table

Here present states and next states are 00, 01, 10 and 11. So assign a=00, b=01, c=10 and d=11. The state table can be drawn as shown in figure 4.13.

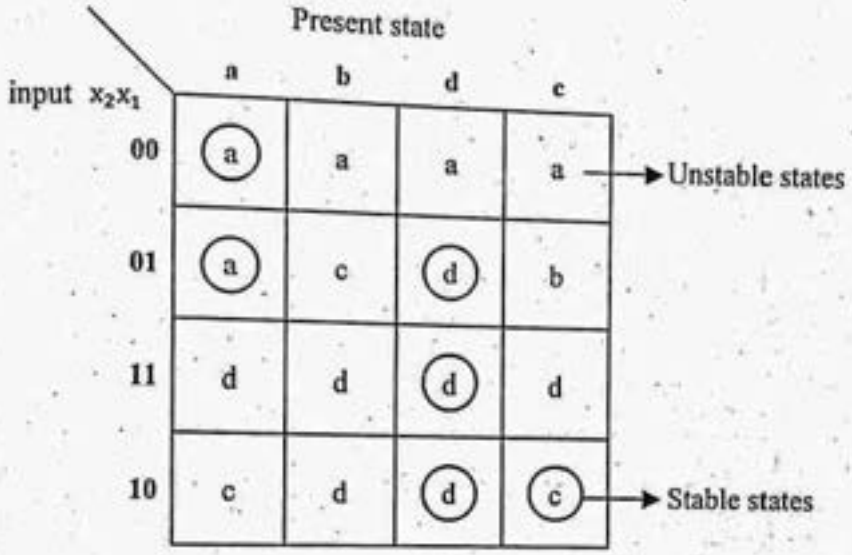


Figure 4.13 State table

Step 5: Plot the output map

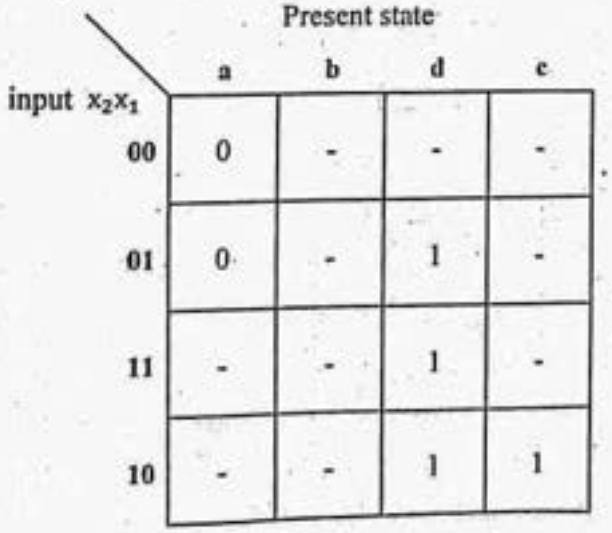


Figure 4.14 Output map

The output is mapped for all stable states which are indicated by circles as shown in figure 4.14. For unstable states, the output is mapped unspecified.

Step 6: Plot the state diagram

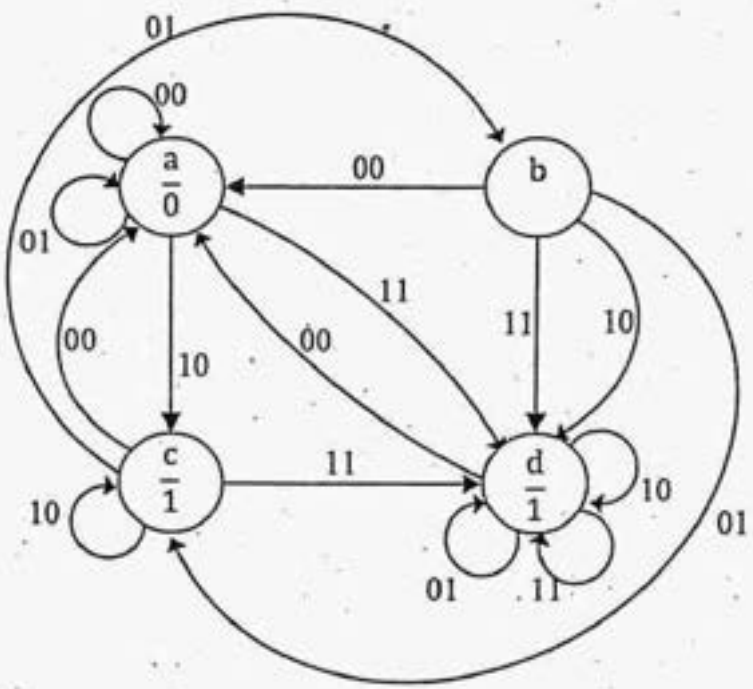


Figure 4.15 State Diagram

By using the state table, draw the next state transition for input $x_2x_1 = 00$, $x_2x_1 = 01$, $x_2x_1 = 10$ and $x_2x_1 = 11$ as shown in figure 4.15. Similarly mark the output Z from the output map for each stable state.

Example 4.2: ***An asynchronous sequential circuit is described by the following excitation and output function.***

$$Y = X_1X_2 + (X_1 + X_2)Y$$

$$Z = Y$$

(i) ***Draw the logic diagram.***

(ii) ***Derive the transition table and output map.***

(iii) ***Describe the behavior of the circuit***

Solution:

Here Y is found on both sides of the equations.

Let the next state be $Y^+ = X_1X_2 + (X_1 + X_2)Y$

Output $Z = Y^+$

Here Y is the present state, Y^+ is the next state, Z is the output and X_1 and X_2 are the inputs.

i. **Logic diagram**

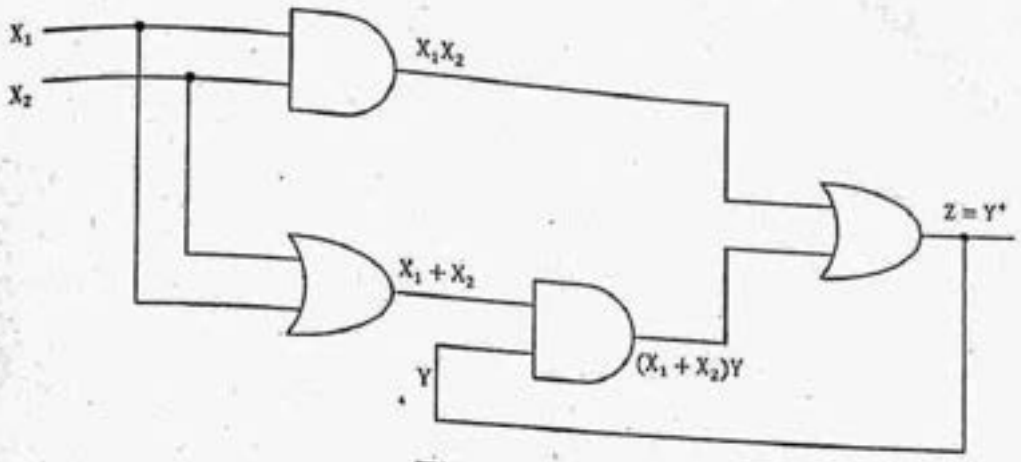


Figure 4.16

Present state	Input		Next state	Output	State
Y	X_2	X_1	Y^+	Z	
0	0	0	0	0	Stable
0	0	1	0	0	Stable
0	1	0	0	0	Stable
0	1	1	1	1	Unstable
1	0	0	0	0	Unstable
1	0	1	1	1	Stable
1	1	0	1	1	Stable
1	1	1	1	1	Stable

Table 4.4 Truth table

The truth table should contain the present state (Y), inputs (X_1, X_2), next state (Y^+) and output 'Z'. Here the number of present states is less than the number of inputs, so write the present state first in the truth table as shown in table 4.4.

First write the possible binary combination for present states and inputs. Then by substituting the value of Y, X_1 and X_2 in the next state and output equation, find Y^+ and Z. For the given input, if the next state Y_2Y_1 is same as that of present states y_2y_1, then the state is said to be stable.

ii. Transition table and output map

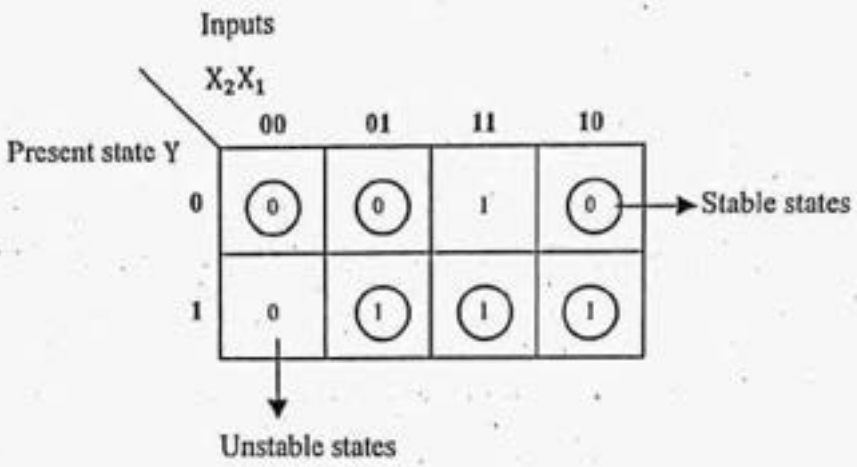


Figure 4.17 Transition table

The rows of transition table represents the present state 'Y' and columns of transition table represents the inputs 'X_2X_1'. The number written in the table represents next state Y^+. The circle around Y^+ indicate that the state is stable which is shown in figure 4.17. Then assign states for the binary values and plot the state table. Here present states and next states are 0 and 1. So assign a=0 and b=1. The state table can be drawn as shown in figure 4.18.

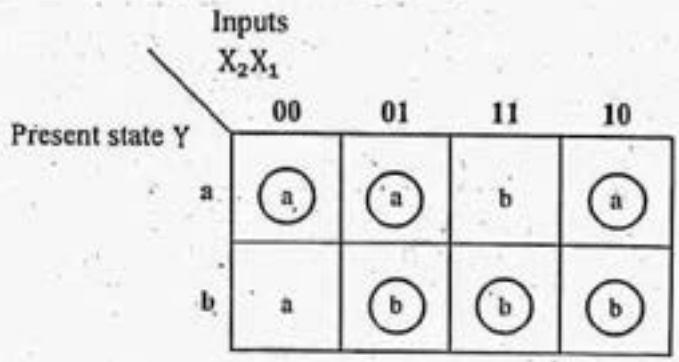


Figure 4.18 State table.

The output is mapped for all stable states which is indicated by circles. For unstable states, the output is mapped unspecified.

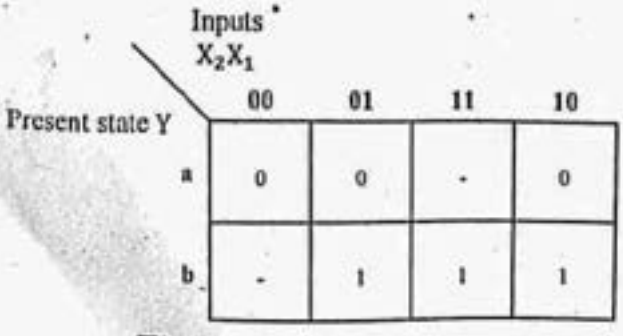


Figure 4.19 Output map

Plot the state diagram by using the state table and output map as shown in figure 4.20.

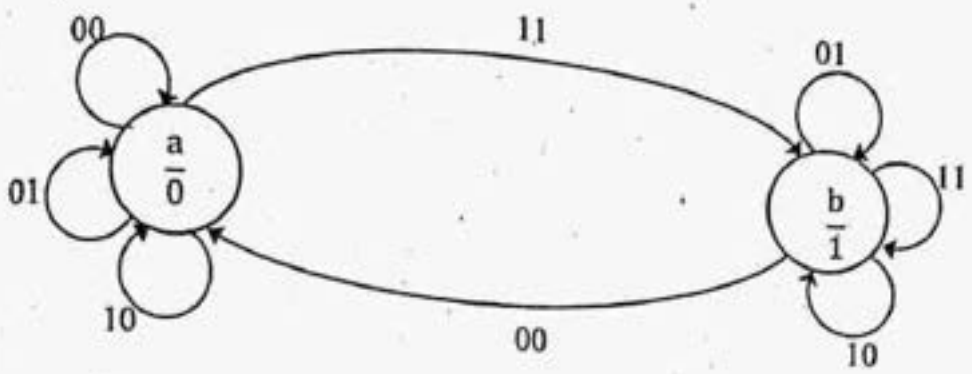


Figure 4.20 State diagram

iii. Behavior of the circuit

The given circuit gives the carry output of the full adder circuit.

4.3 REDUCTION OF STATE AND FLOW TABLES

Present state	Next state, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
a	(a), 0	b, -	-, -	c, -
b	a, -	(b), 0	d, -	-, -
c	a, -	-, -	e, -	(c), 0
d	-, -	f, -	(d), 1	c, -
e	-, -	f, -	(e), 0	c, -
f	a, -	(f), 1	d, -	-, -

Table 4.5 Primitive flow table.

Consider a primitive flow table shown in table 4.5

The number of states in the primitive flow table can be reduced by using state reduction technique. This is similar to state reduction in synchronous sequential circuits. State reduction is nothing but reducing the equivalent states to a single state. Two states are said to be equivalent if the two states produces the same next state and same output for each input. Here a, b, c, d, e, and f are present states.

Two rows of primitive flow table can be merged into a single row, if there is no output conflict and no state conflicts in any column.

A stable state and unstable state can be merged to a stable state. The specified output is filled when specified output is merged with unspecified output.

In the primitive flow table shown in table 4.6 states a and b have same next states and outputs so this states can be merged as shown below.

Present state	Next state, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
a	(a), 0	b, -	-, -	c, -
b	a, -	(b), 0	d, -	-, -
Reduced state				
S_0	(a), 0	(b), 0	d, -	c, -

Table 4.6

Here states a and b are reduced to a single state S_0.

$(a,b) \rightarrow S_0$

Here the next state of 'a' when xy = 00 is 'a' and it is stable but the next state of 'b' when xy = 00 is 'a' and it is unstable. So the reduced state is 'a' and it is stable.

The next state of 'a' when xy = 11 is unspecified, but the next state of 'b' when xy = 11 is 'd' and it is unstable. So the reduced state is 'd' and it is unstable.

The output of state 'a' when xy=00 is 0 and the output of state 'b' when xy=00 is unspecified. So the reduced output is 0.

The output of state 'a' and 'b' when xy=11 is unspecified so the reduced output is also unspecified.

Present state	Next state, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
c	a , -	- , -	e , -	(c), 0
e	- , -	f , -	(e), 0	c , -
Reduced state				
S_1	a , -	f , -	(e), 0	(c), 0

Table 4.7

In the primitive flow table state 'c' and 'e' have same next states and outputs. So these states can be merged to S_1.

$$(c, e) \rightarrow S_1$$

Also the states 'd' and 'f' have same next states and outputs. So 'd' and 'f' can be merged to S_2

$$(d, f) \rightarrow S_2$$

Present state	Next state, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
d	- , -	f , -	(d), 1	c , -
f	a , -	(f), 1	d , -	- , -
Reduced state				
S_2	a , -	(f), 1	(d), 1	c , -

Table 4.8

Present state	Next state, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
S_0	(a), 0	(b), 0	d, -	c, -
S_1	a, -	f, -	(e), 0	(c), 0
S_2	a, -	(f), 1	(d), 1	c, -

Table 4.9 Reduced flow table

Two states can be reduced to a single state as given in table 4.10.

Equivalent state	Reduced state
Stable state and unstable state	Stable state
Stable state and stable state	Stable state
Stable state and unspecified state	Stable state
Unstable state and unstable state	Unstable state
Unstable state and unspecified state	Unstable state
Specified output and unspecified output	Specified output
Specified output and specified output	Specified output
Unspecified output and unspecified output	Unspecified output

Table 4.10

4.4 RACE FREE STATE ASSIGNMENT

To avoid critical races, it is necessary that present state and next state should be given adjacent assignments. If the present state and next state are said to be adjacent, if the binary value differ in only one bit. For example 010 and 011 are adjacent because they differ only in the third bit. The binary values 010 and 111 are not adjacent because the first and third bit differs. Consider the reduced state table shown in table 4.11.

Present state	Next state, output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
S_0	S_0 , 0	S_0 , 0	S_2 , -	S_1 , -
S_1	S_0 , -	S_2 , -	S_1 , 0	S_1 , 0
S_2	S_0 , -	S_2 , 1	S_2 , 1	S_1 , -

Table 4.11

Now if we assign $S_0 = 00$, $S_1 = 01$ and $S_2 = 10$

Present state	Next state A^+B^+, Output Z			
	xy = 00	xy = 01	xy = 11	xy = 10
$S_0(00)$	00 , 0	00 , 0	10 , -	01 , -
$S_1(01)$	00 , -	10 , -	01 , 0	01 , 0
$S_2(10)$	00 , -	10 , 1	10 , 1	01 , -

Table 4.12 State assignment without race free

Here if the present state is AB=01 and input is xy=01, the next state is $A^+B^+ = 10$. Also if the present state is AB=10 and input is 10, the next state $A^+B^+ = 01$. In both cases present state and next state are not adjacent. A race-free assignment can be obtained if we add an extra row to the flow table. The first three rows represent the same conditions as the original three- row table. The fourth row is assigned with 11. Now the transition of 01 to 10 for input xy=01 must go through 11. Also the transition of 10 to 01 for input xy=10 must go through 11.

Present state	Next state A^+B^+, output			
	xy = 00	xy = 01	xy = 11	xy = 10
$S_0(00)$	00 , 0	00 , 0	10 , -	01 , -
$S_1(01)$	00 , -	11 , -	01 , 0	01 , 0
$S_2(10)$	00 , -	10 , 1	10 , 1	11 , -
$S_3(11)$	- , -	10 , -	- , -	01 , -

Table 4.13 Race Free State assignment

Since the present states and next states differ by single bit, the circuit for this flow table will be free from races.

4.5 INCOMPLETELY SPECIFIED STATE MACHINES

Sequential circuits in which some of the states are left unspecified are called Incompletely specified state machines. In sequential circuits, not all combinations of states and inputs are possible. For example, consider the state table shown in table 4.14. Here the state 'b' will never receive a '0' input and hence the next state and outputs are left unspecified by a dash (-). In some situation, the state transitions are completely defined but for some combinations of states and inputs, the output values may be left unspecified.

Present state	Next state, Output Z	
	X = 0	X = 1
a	d,1	b,0
b	- , -	c,0
c	a, 1	b,-
d	a, 0	d,1

Table 4.14 State table

In the state table shown in table 4.14, the next state of 'c' is specified as 'b' for the input '1' but the output is unspecified as dash.When a state transition is unspecified, the future behavior of the sequential machine may become unpredictable.

4.6 DESIGN OF ASYNCHRONOUS SEQUENTIAL CIRCUITS

The procedure for designing asynchronous sequential circuit is given below.

1. From the given description, draw the state diagram.
2. Construct a primitive flow table from the state diagram.
3. Reduce the primitive flow table by eliminating the redundant states by using state reduction.
4. Assign binary values to states based on race free state assignment.
5. Find output table by using a flip-flop excitation table.
6. Find the flip-flop input equations and output equations using K-map.
7. Draw the logic diagram.

Example 4.3: Design an asynchronous sequential circuit with two inputs X and Y with one output Z. Whenever Y is 1, input X is transferred to Z. When Y is 0, the output does not change for any change in X. Use D flip-flop.

Solution : It is given that X and Y are inputs and Z is the output.

If Y=1; Z=X

If Y=0, there will be no change in output

Step 1: Draw the state diagram.

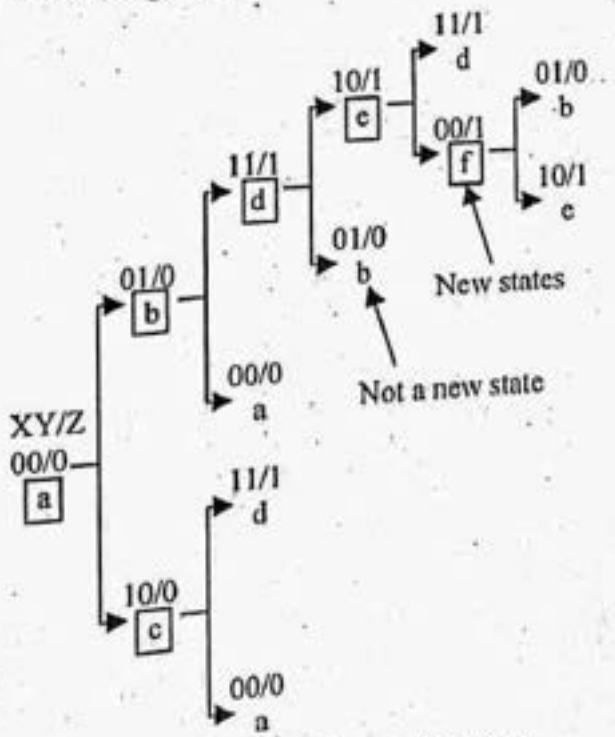


Figure 4.21 (a)

The state diagram for the given problem can be easily drawn using the diagram shown in figure 4.21(a). Initially assume XY=00 and Z=0. The input XY=00 can change to XY=01 or XY=10. Given that if Y=1, Z=X. If Y=0, the output does not change.

Therefore XY=01 has the output Z=0 and XY=10 has the output Z=0. Assign 01/0 as 'b', 10/0 as 'c'. Highlight a, b and c because a, b and c are new states. The input XY=01 can change to XY=11 or XY=00. The output when XY=11 is 1 and XY=00 is 0. Assign 11/1 as 'd' and highlight it because it is a new state.

Assign 00/0 as 'a' because we have already assigned 00/0 as 'a'. No need to highlight it because it is not a new state. The input XY=10 can change to XY=11 or XY=00. The output when XY=11 is 1 and XY=00 is 0. Assign 11/1 as'd' and 00/0 as 'a'. No need to highlight 'a' and 'd' because 'a' and 'd' are not a new states. Proceed this process for all new states.

Now the state diagram can be easily drawn using figure 4.21(a). The different states are a, b, c, d, e and f.

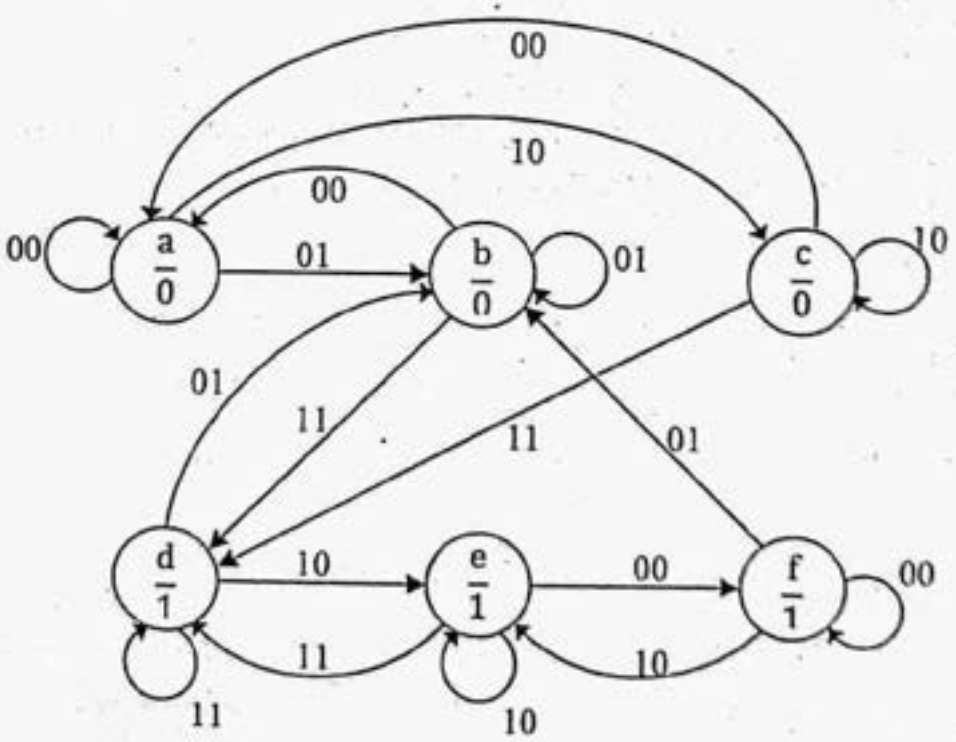


Figure 4.21 (b) State diagram

Step 2: Construct the primitive flow table

Present state	Next state, Output Z			
	XY=00	XY=01	XY=11	XY=10
a	(a), 0	b, -	-,-	c, -
b	a, -	(b), 0	d, -	-,-
c	a, -	-,-	d, -	(c), 0
d	-,-	b, -	(d), 1	e, -
e	f, -	-,-	d, -	(e), 1
f	(f), 1	b, -	-,-	e, -

Table 4.15 Primitive flow table

Here the circle represents that the state is stable.

Step 3: Reduce the primitive flow table

The primitive flow table can be minimized by merging the states a, b, c to a single state 'S_0' because the next state and the output for the states a, b, c are same.

$(a, b, c) \rightarrow S_0$

Also we can merge the states d, e, f to a single state 'S_1' because the next state and the output for the states d, e, f are same.

$(d, e, f) \rightarrow S_1$

Present state	Next state, Output Z			
	XY=00	XY=01	XY=11	XY=10
S_0	(S_0), 0	(S_0), 0	S_1, -	(S_0), 0
S_1	(S_1), 1	S_0, -	(S_1), 1	(S_1), 1

Table 4.16 Reduced primitive flow table

Step 4: Assign binary values to states based on race free state assignment.

Here, there are two states S_0 and S_1. So assign S_0=0, S_1=1 also, let the present state be A and the Next state be A^+. Let the output be Z. Since the present states and next states differ by single bit, the circuit for this flow table will be free from races.

Present state A	Next state A^+, Output Z			
	XY=00	XY=01	XY=11	XY=10
0	0 , 0	0 , 0	1 , -	0 , 0
1	1 , 1	0 , -	1 , 1	1 , 1

Table 4.17 State assignment

Step 5: Find the output table.

By using the excitation table of D flip-flop shown in table 4.18, draw the output table.

A	A^+	D_A
0	0	0
0	1	1
1	0	0
1	1	1

Table 4.18 Excitation table of D flip-flop

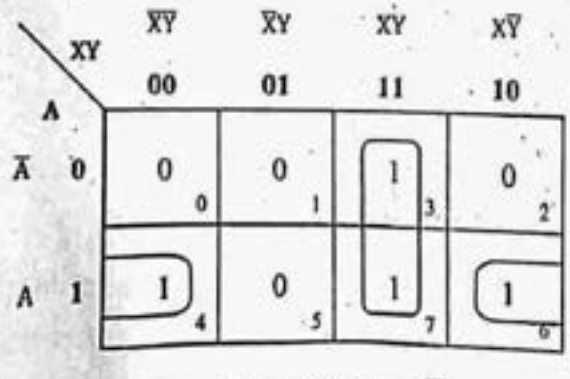
Present state	Flip-flop input D_A, Output Z			
	XY=00	XY=01	XY=11	XY=10
0	0 , 0	0 , 0	1 , -	0 , 0
1	1 , 1	0 , -	1 , 1	1 , 1

Table 4.19 Output table

Step 6: Find the flip-flop input equations and output equations using K-map.

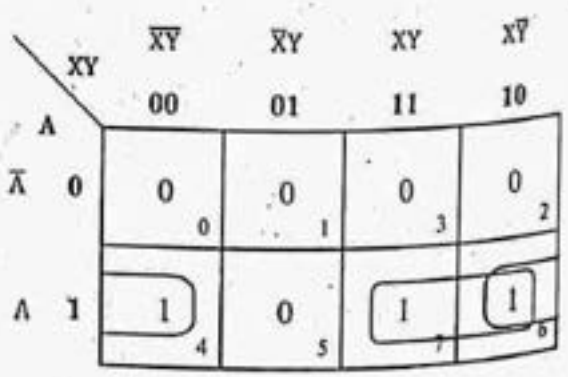
Assume '0' for unspecified outputs.

K-map for D_A



$D_A = XY + A\bar{Y}$

K-map for Z



$Z = AX + A\bar{Y}$

Step 7: Draw the logic diagram

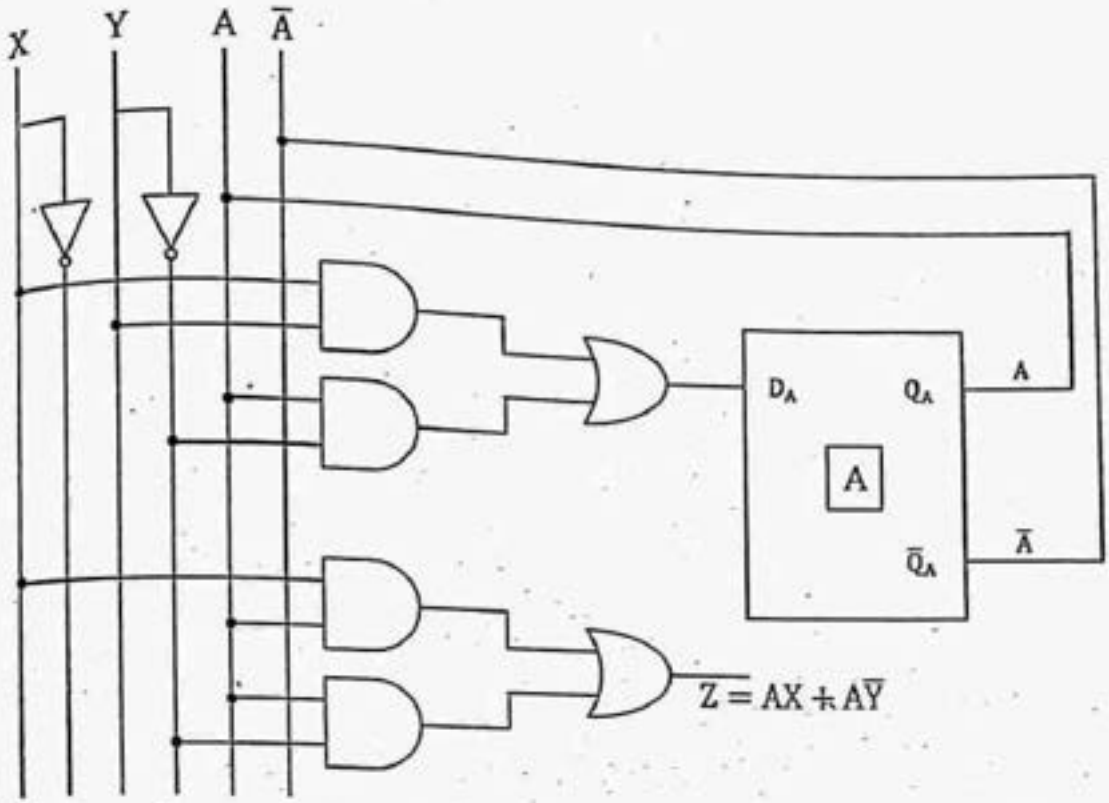


Figure 4.22 Logic diagram

Example 4.4: *Design an asynchronous sequential circuit that has two inputs x_1 and x_2 and one output Z. The output Z=1, if x_1 changes from 0 to 1. Z=0, if x_2 changes from 0 to 1 and Z=0 otherwise. Realize the circuit using JK flip-flop.*

Solution:

It is given that x_1 and x_2 are inputs and Z is the output.

If x_1 changes from 0 to 1, Z=1

If x_2 changes from 0 to 1, Z=0

Otherwise, Z=0

Step 1: Draw the state diagram

The state diagram for the given problem can be easily drawn using the diagram shown in figure 4.23.

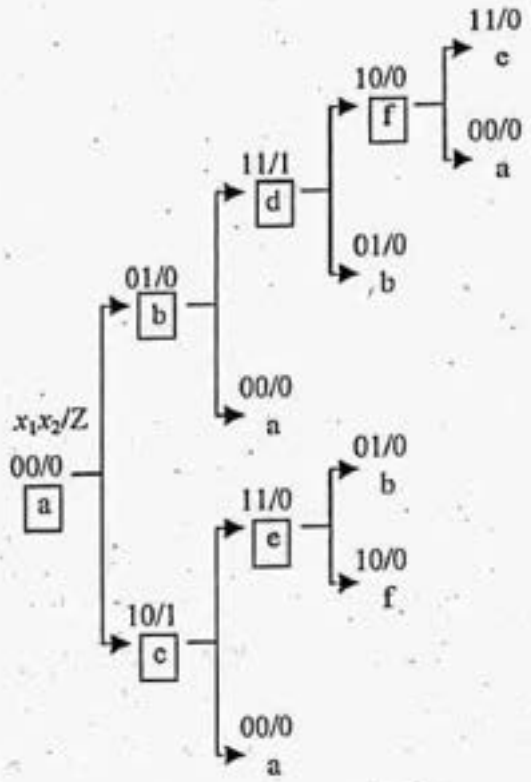


Figure 4.23

Initially assume x_1x_2=00 and output Z=0. The input x_1x_2=00 can change to x_1x_2=01 or x_1x_2=10. If x_2 changes from 0 to1, Z=0. If x_1 changes from 0 to 1, Z=1. So the next states of 00/0 are 01/0 and 10/1. Assign 00/0 as 'a', 01/0 as 'b' and 10/1 as 'c'. Highlight a, b, and c because a, b and c are new states. Proceed this process for all new states.

Now the state diagram can be easily drawn from the figure 4.23.

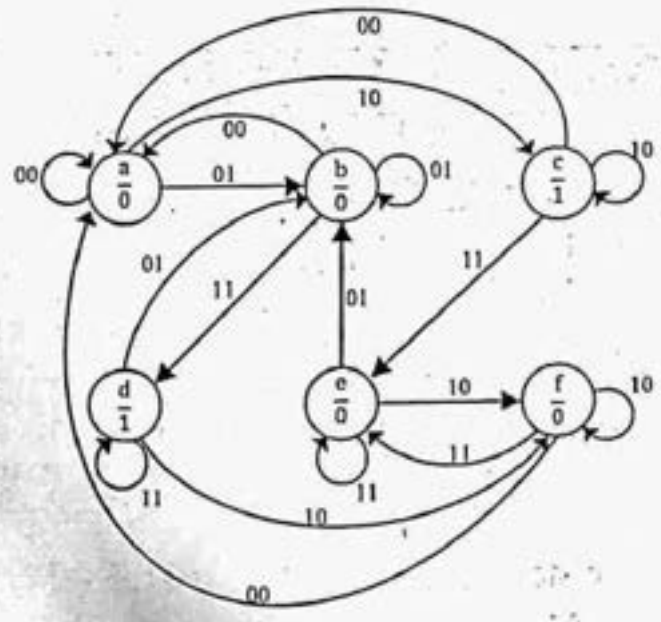


Figure 4.24 State diagram

Step 2: Construct the primitive flow table.

Present state	Next state, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
a	(a), 0	b, -	-,-	c, -
b	a, -	(b), 0	d, -	-,-
c	a, -	-,-	e, -	(c), 1
d	-,-	b, -	(d), 1	f, -
e	-,-	b, -	(e), 0	f, -
f	a, -	-,-	e, -	(f), 0

Table 4.20 Primitive flow table

Step 3: Reduce the primitive flow table

The primitive flow table can be minimized by merging the states 'a' and 'c' to a single state S_0

$$(a, c) \rightarrow S_0$$

Also we can merge the states 'b' and 'd' to a single state 'S_1'

$$(b, d) \rightarrow S_1$$

Also we can merge the states 'e' and 'f' to a single state 'S_2'

$$(e, f) \rightarrow S_2$$

Present state	Next state, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
S_0	(S_0), 0	S_1, -	S_2, -	(S_0), 1
S_1	S_0, -	(S_1), 0	(S_1), 1	S_2, -
S_2	S_0, -	S_1, -	(S_2), 0	(S_2), 0

Table 4.21 Reduced primitive flow table

Step 4: Assign binary values to the states based on race free state assignment.

Since there are three states assign S_0=00, S_1=01, S_2=10. Let the present states be AB and the next state be A^+B^+.

Present state AB	Next state A^+B^+, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
00	00 , 0	01 , -	10 , -	00 , 1
01	00 , -	01 , 0	01 , 1	[10] , -
10	00 , -	[01] , -	10 , 0	10 , 0

Table 4.22 State assignment

Here the present state AB=10 and the next state $A^+B^+ = 01$ differs by two bits.

Also the present state AB=01 and the next state $A^+B^+ = 10$ differs by two bits. Since the present state and next states are not adjacent, races will be there for the circuit of the above flow table. So by race free state assignment the above flow table can be modified by a new state 11.

Present state AB	Next state A^+B^+, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
00	00 , 0	01 , -	10 , -	00 , 1
01	00 , -	01 , 0	01 , 1	[11] , -
10	00 , -	[11] , -	10 , 0	10 , 0
11	- , -	[01] , -	- , -	[10] , -

Table 4.23 Race free state assignment

Step 5: Find the output table.

By using the excitation table of JK flip-flop shown in table 4.24, draw the output table.

A	A^+	J_A	K_A
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

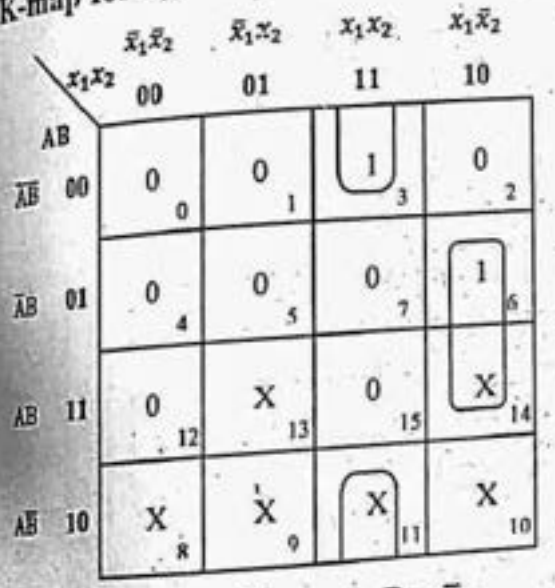
Table 4.24 Excitation table of JK flip-flop

Present state AB	Flip-flop inputs $J_A K_A$, $J_B K_B$, Output Z			
	x_1x_2=00	x_1x_2=01	x_1x_2=11	x_1x_2=10
00	0X,0X,0	0X,1X,-	1X,0X,-	0X,0X,1
01	0X,X1,-	0X,X0,0	0X,X0,1	1X,X0,-
10	X1,0X,-	X0,1X,-	X0,0X,0	X0,0X,0
11	-,-,-	X1,X0,-	-,-,-	X0,X1,-

Table 4.25 Output table

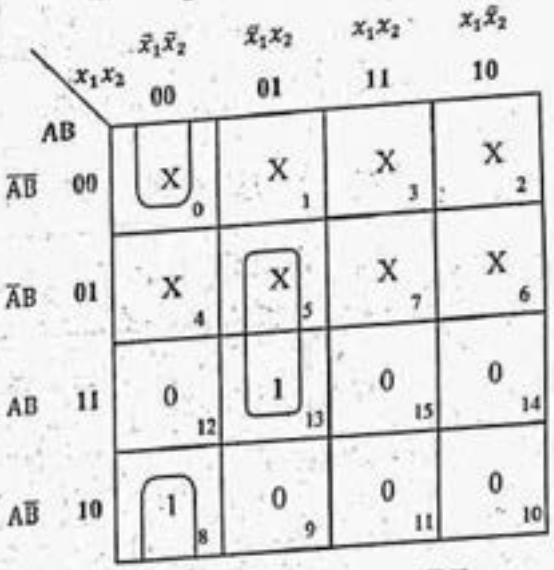
Step 6: Find the flip-flop input equations and output equations using K-map.

K-map for J_A



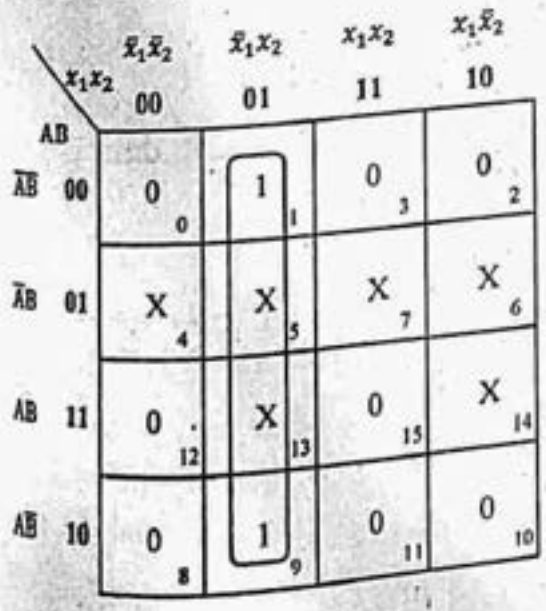
$$J_A = \bar{B}x_1x_2 + Bx_1\bar{x}_2$$

K-map for K_A



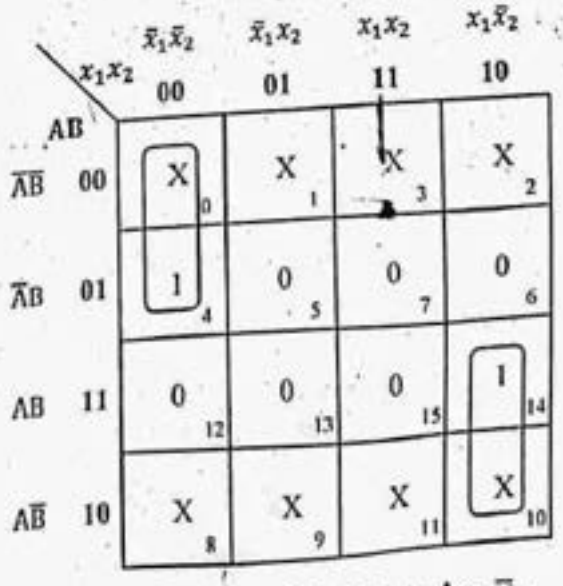
$$K_A = \bar{B}\bar{x}_1\bar{x}_2 + B\bar{x}_1x_2$$

K-map for J_B



$$J_B = \bar{x}_1x_2$$

K-map for K_B



$$K_B = \bar{A}\bar{x}_1\bar{x}_2 + Ax_1\bar{x}_2$$

K-map for Z

AB \ x_1x_2	$\bar{x}_1\bar{x}_2$ 00	$\bar{x}_1x_2$ 01	x_1x_2 11	$x_1\bar{x}_2$ 10
$\bar{A}\bar{B}$ 00	0 (0)	0 (1)	0 (3)	1 (2)
$\bar{A}B$ 01	0 (4)	0 (5)	1 (7)	0 (6)
AB 11	0 (12)	0 (13)	0 (15)	0 (14)
$A\bar{B}$ 10	0 (8)	0 (9)	0 (11)	0 (10)

$$Z = \bar{A}\bar{B}x_1\bar{x}_2 + \bar{A}Bx_1x_2$$

Step 7: Draw the logic diagram

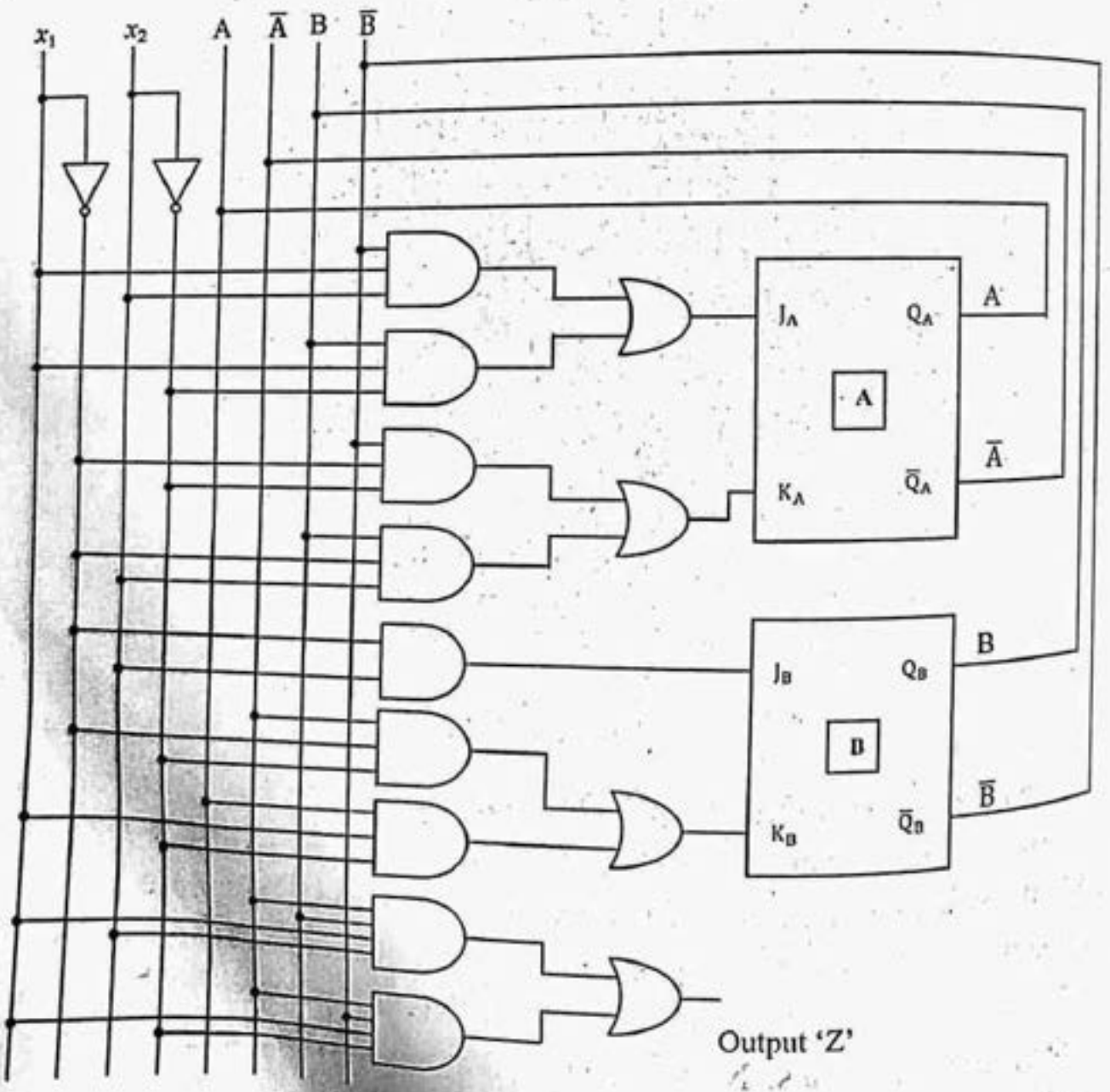


Figure 4.25 Logic di...

Example 4.5: ***Design an asynchronous sequential circuit that has two inputs X_2 and X_1 and output Z. When $X_1 = 0$, the output $Z = 0$, the first change in X_2 that occur while X_1 is 1 will cause output Z to be 1. The output Z will remain 1 until X_1 returns to 0. Use SR flip-flop.***

Solution: It is given that X_2, X_1 are the inputs and Z is the output

If X_1=0, Z=0

If there is change in X_2 while X_1=1, Z=1

Otherwise, Z=0

Step 1: Draw the state diagram

The state diagram can be easily drawn by using the diagram shown in figure 4.26.

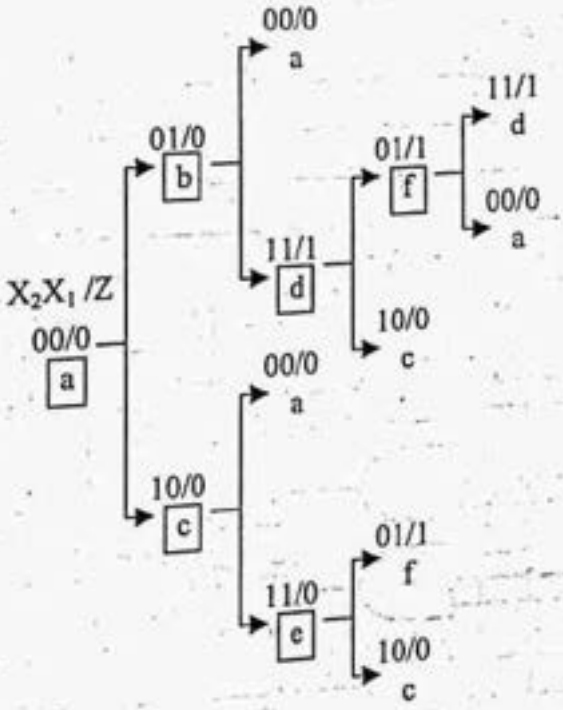


Figure 4.26

Initially assume X_2X_1=00 and Z=0. The input X_2X_1 can change to X_2X_1=01 or X_2X_1=10. It is given that if X_1=0 the output is 0. If X_1=1 and X_2 changes then Z=1 and Z=0 otherwise. So the next states of 00/0 are 01/0 and 10/0. Also assign 00/0 as 'a', 01/0 as 'b' and 10/0 as 'c'. Highlight a, b and c because a, b and c are new states. Then find the next state of 'b' and 'c'. Proceed the same process for all new states by highlighting the new states. Now the state diagram can be easily drawn from the figure 4.26.

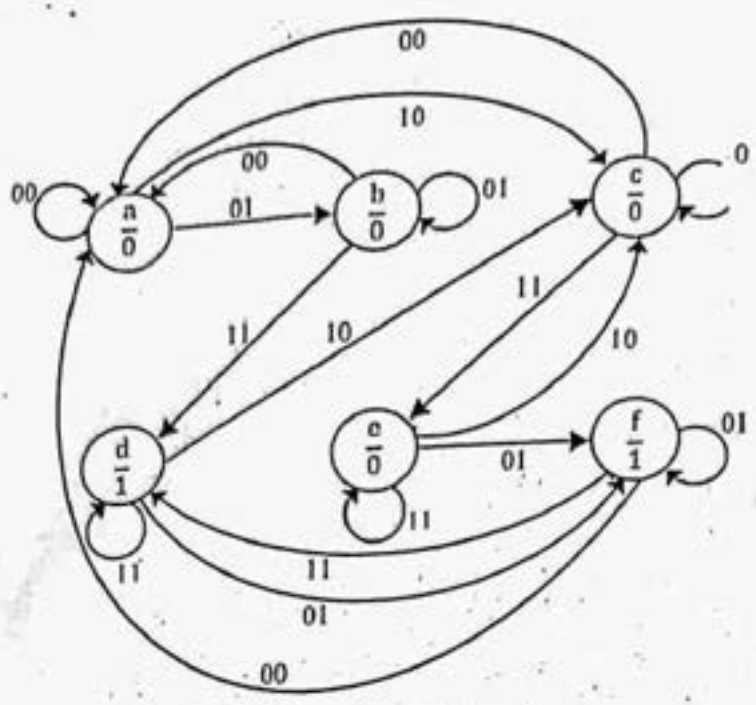


Figure 4.27 State diagram

Step 2: Construct the primitive flow table.

Present state	Next state, Output Z			
	$X_2 X_1$=00	$X_2 X_1$=01	$X_2 X_1$=11	$X_2 X_1$=10
a	(a), 0	b, -	-,-	c, -
b	a, -	(b), 0	d, -	-,-
c	a, -	-,-	e, -	(c), 0
d	-,-	f, -	(d), 1	c, -
e	-, -	f, -	(e), 0	c, -
f	a, -	(f), 1	d, -	-, -

Table 4.26 Primitive flow table

Step 3: Reduce the primitive flow table

The primitive flow table can be minimized by merging the states 'a' and 'b' to a single state 'S_0'

$$(a, b) \rightarrow S_0$$

Also we can merge the states 'c' and 'e' to a single state 'S_1'

$$(c, e) \rightarrow S_1$$

Also we can merge the states 'd' and 'f' to a single state 'S_2'

$$(d, f) \rightarrow S_2$$

Present state	Next state, Output Z			
	$X_2 X_1$=00	$X_2 X_1$=01	$X_2 X_1$=11	$X_2 X_1$=10
S_0	(S0), 0	(S0), 0	S_2, -	S_1, -
S_1	S_0, -	S_2, -	(S1), 0	(S1), 0
S_2	S_0, -	(S2), 1	(S2), 1	S_1, -

Table 4.27 Reduced primitive flow table

Step 4: Assign binary values to the states based on race free state assignment.

Since there are three states assign, S_0=00, S_1=01 and S_2=10.

Present state AB	Next state A^+B^+, Output Z			
	$X_2 X_1$=00	$X_2 X_1$=01	$X_2 X_1$=11	$X_2 X_1$=10
00	00, 0	00, 0	10, -	01, -
01	00, -	[10], -	01, 0	01, 0
10	00, -	10, 1	10, 1	[01], -

Table 4.28 State assignment

Since the states S_0, S_1, S_2 are two bits. Let the present states are AB and the next states are A^+B^+.

Here the present state AB=10 and the next state $A^+B^+ = 01$ differs by two bits. Also the present state AB=01 and the next state $A^+B^+ = 10$ differs by two bits. Since the present state and next states are not adjacent races will be there for the circuit of the above flow table. So by race free state assignment the above flow table can be modified by a new state 11, which avoids the critical races.

Present state AB	Next state A^+B^+, Output Z			
	$X_2 X_1$=00	$X_2 X_1$=01	$X_2 X_1$=11	$X_2 X_1$=10
00	00, 0	00, 0	10, -	01, -
01	00, -	[11], -	01, 0	01, 0
10	00, -	10, 1	10, 1	[11], -
11	-,-	[10], -	-,-	[01], -

Table 4.29 Race free state assignment

Step 5: Find the output table.

By using the excitation table of SR flip-flop shown in table 4.30, draw the output table

A	A^+	S_A	R_A
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Table 4.30 Excitation table of SR flip-flop

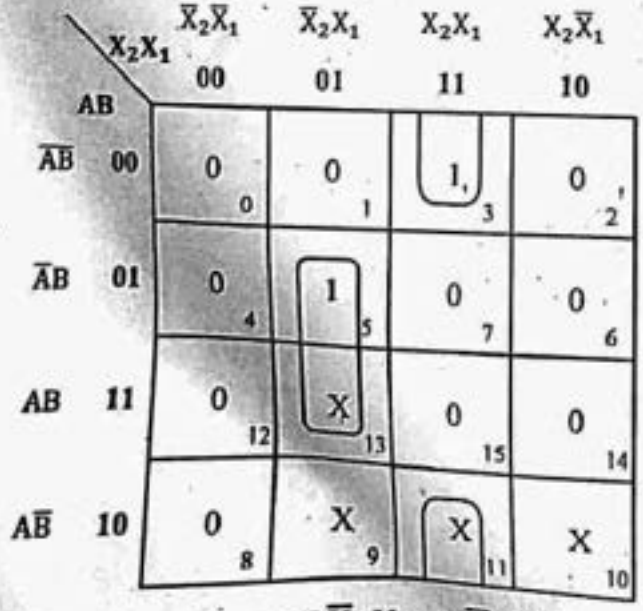
Present state AB	Flip-flop inputs S_A R_A, S_B R_B, Output Z			
	X_2 X_1=00	X_2 X_1=01	X_2 X_1=11	X_2 X_1=10
00	0X,0X,0	0X,0X,0	10,0X,-	0X,10,-
01	0X,01,-	10,X0,-	0X,X0,0	0X,X0,0
10	01,0X,-	X0,0X,1	X0,0X,1	X0,10,-
11	-, -, -	X0,01,-	-, -, -	01,X0,-

Table 4.31 Output table

Step 6: Find the flip-flop input equations and output equations using K-map.

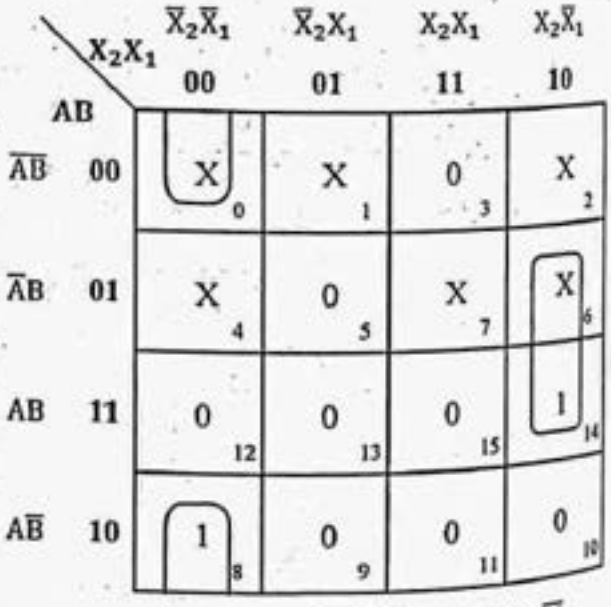
Assume '0' for unspecified flip-flop inputs and unspecified outputs.

K-map for S_A



$$S_A = B\overline{X}_2X_1 + \overline{B}X_2X_1$$

K-map for R_A



$$R_A = \overline{B}X_2\overline{X}_1 + BX_2\overline{X}_1$$

K-map for S_B

AB \ X_2X_1	$\bar{X}_2\bar{X}_1$ 00	$\bar{X}_2X_1$ 01	X_2X_1 11	$X_2\bar{X}_1$ 10
$\bar{A}\bar{B}$ 00	0 (0)	0 (1)	0 (3)	1 (2)
$\bar{A}B$ 01	0 (4)	X (5)	X (7)	X (6)
AB 11	0 (12)	0 (13)	0 (15)	X (14)
$A\bar{B}$ 10	0 (8)	0 (9)	0 (11)	1 (10)

$$S_B = X_2\bar{X}_1$$

K-map for R_B

AB \ X_2X_1	$\bar{X}_2\bar{X}_1$ 00	$\bar{X}_2X_1$ 01	X_2X_1 11	$X_2\bar{X}_1$ 10
$\bar{A}\bar{B}$ 00	X (0)	X (1)	X (3)	0 (2)
$\bar{A}B$ 01	1 (4)	0 (5)	0 (7)	0 (6)
AB 11	0 (12)	1 (13)	0 (15)	0 (14)
$A\bar{B}$ 10	X (8)	X (9)	X (11)	0 (10)

$$R_B = \bar{A}X_2\bar{X}_1 + A\bar{X}_2X_1$$

K-map for Z

AB \ X_2X_1	$\bar{X}_2\bar{X}_1$ 00	$\bar{X}_2X_1$ 01	X_2X_1 11	$X_2\bar{X}_1$ 10
$\bar{A}\bar{B}$ 00	0 (0)	0 (1)	0 (3)	0 (2)
$\bar{A}B$ 01	0 (4)	0 (5)	0 (7)	0 (6)
AB 11	0 (12)	0 (13)	0 (15)	0 (14)
$A\bar{B}$ 10	0 (8)	1 (9)	1 (11)	0 (10)

$$Z = A\bar{B}X_1$$

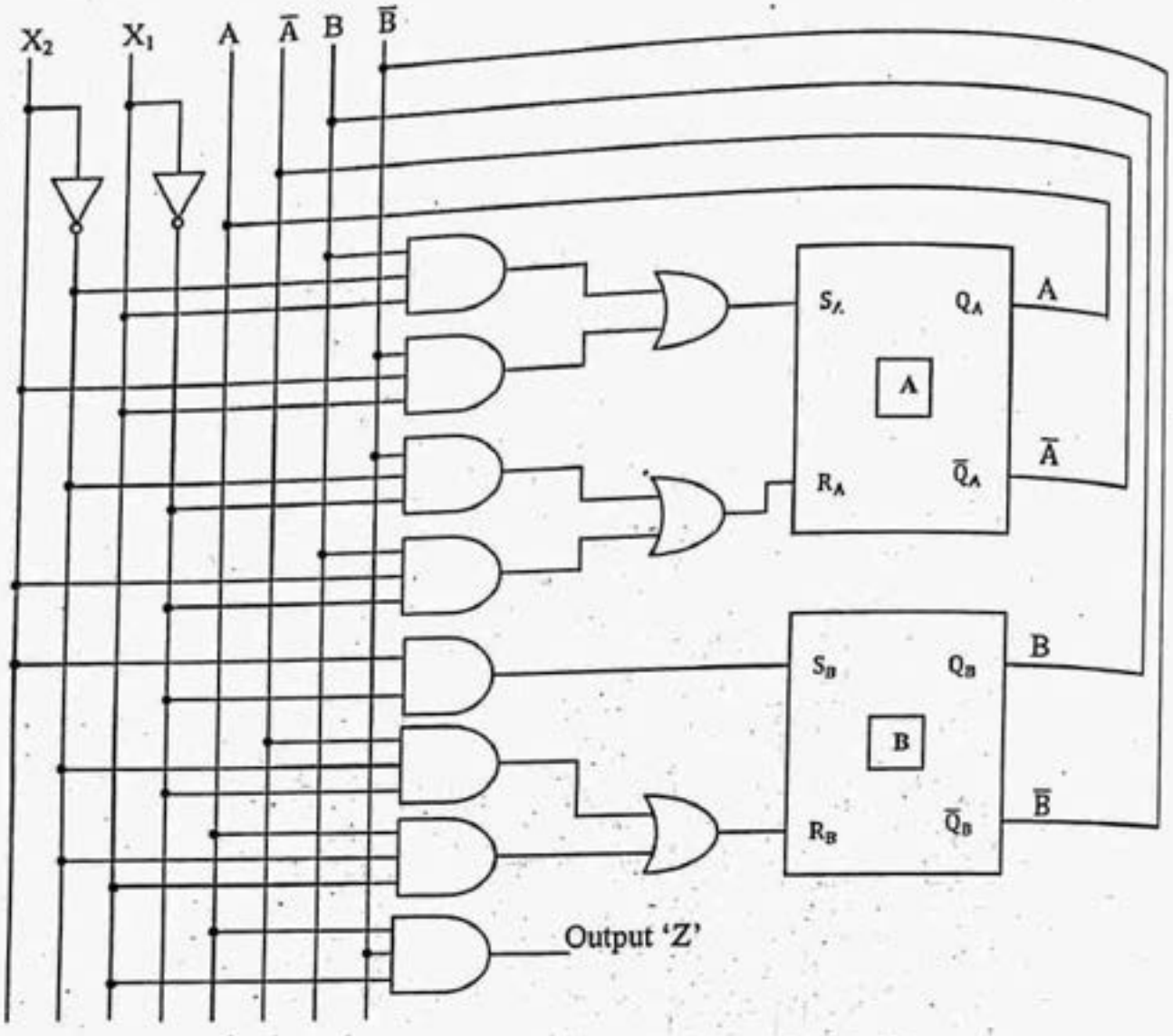


Figure 4.28 Logic diagram

Example 4.6: ***Design an asynchronous circuit that has two inputs x_1 and x_2 and one output Z. The circuit is required to give an output whenever the input sequence (0,0) (0,1) and (1,1) received but only in that order. Design it using T flip-flop.***

Solution:

Given that x_1 and x_2 are inputs and Z is the output.

If the sequence (0,0) (0,1) and (1,1) arrives Z=1

Otherwise Z=0

Step 1: Draw the state diagram

The state diagram can be easily drawn by using the following diagram shown in figure 4.29. Initially assume x_1x_2=00 and output Z=0. The input x_1x_2=00 can

change to x_1x_2=01 or x_1x_2=10. Proceed the steps and assign states. Also highlight new states. The output will be Z=1 if the sequence (0,0) (0,1) and (1,1) arrives. So state 'd' has output 1. Now the state diagram can be easily drawn by using the figure 4.29 as shown in figure 4.30.

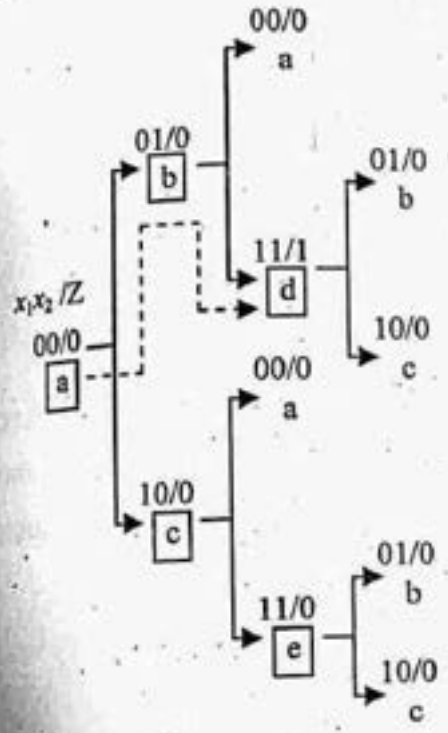


Figure 4.29

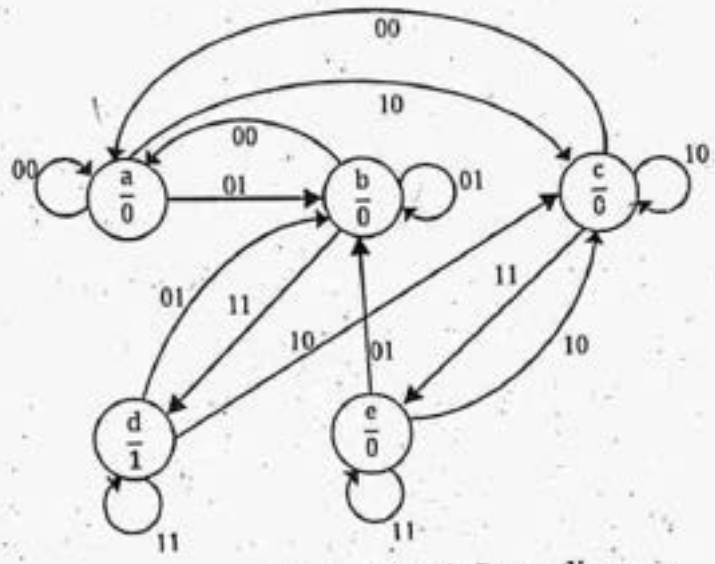


Figure 4.30 State diagram

Step 2: Construct the primitive flow table

Present state	Next state, Output Z			
	$x_1\,x_2$=00	$x_1\,x_2$=01	$x_1\,x_2$=11	$x_1\,x_2$=10
a	(a), 0	b , -	- ,-	c , -
b	a , -	(b), 0	d , -	- ,-
c	a , -	- ,-	e , -	(c),0
d	- , -	b , -	(d), 1	c , -
e	- , -	b , -	(e), 0	c , -

Table 4.32 Primitive flow table

Step 3: Reduce the primitive flow table

The primitive flow table can be minimized by merging the states 'a', 'b' and 'd' to a single state 'S_0'

$$(a, b, d) \rightarrow S_0$$

Also we can merge the states 'c' and 'e' to a single state 'S_1'

$(c, e) \rightarrow S_1$

Present state	Next state, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
S_0	(S0), 0	(S0), 0	(S0), 1	S_1, -
S_1	S_0, -	S_0, -	(S1), 0	(S1), 0

Table 4.33 Reduced primitive flow table

Step 4: Assign binary values to the states based on race free state assignment.

Since there are two states, assign S_0=0 and S_1=1. Also assign the unfilled next states as initial state '0'. Let the present states be A and the next state be A^+. Since the present states and next states differ by single bit, the circuit for the flow table shown in table 4.33 will be free from races.

Present state A	Next state A^+, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
0	0, 0	0, 0	0, 1	1, -
1	0, -	0, -	1, 0	1, 0

Table 4.34 State assignment

Step 5: Find output table

By using the T flip-flop excitation table shown below. Plot the output table.

A	A^+	T_A
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.35 Excitation table of T flip-flop

Present state A	Flip-flop inputs T_A, Output Z			
	$x_1 x_2$=00	$x_1 x_2$=01	$x_1 x_2$=11	$x_1 x_2$=10
0	0 , 0	0 , 0	0 , 1	1 , -
1	1 , -	1 , -	0 , 0	0 , 0

Table 4.36 Output table

Step 6: Find the flip-flop input equations and output equations using K-map.

Assume '0' for unspecified flip-flop inputs and unspecified outputs.

K-map for T_A

A \ x_1x_2	$\bar{x}_1\bar{x}_2$ 00	$\bar{x}_1x_2$ 01	x_1x_2 11	$x_1\bar{x}_2$ 10
$\bar{A}$ 0	0 (0)	0 (1)	0 (3)	1 (2)
A 1	1 (4)	1 (5)	0 (7)	0 (6)

$$T_A = A\bar{x}_1 + \bar{A}x_1\bar{x}_2$$

K-map for Z

A \ x_1x_2	$\bar{x}_1\bar{x}_2$ 00	$\bar{x}_1x_2$ 01	x_1x_2 11	$x_1\bar{x}_2$ 10
$\bar{A}$ 0	0 (0)	0 (1)	1 (3)	0 (2)
A 1	0 (4)	0 (5)	0 (7)	0 (6)

$$Z = \bar{A}x_1x_2$$

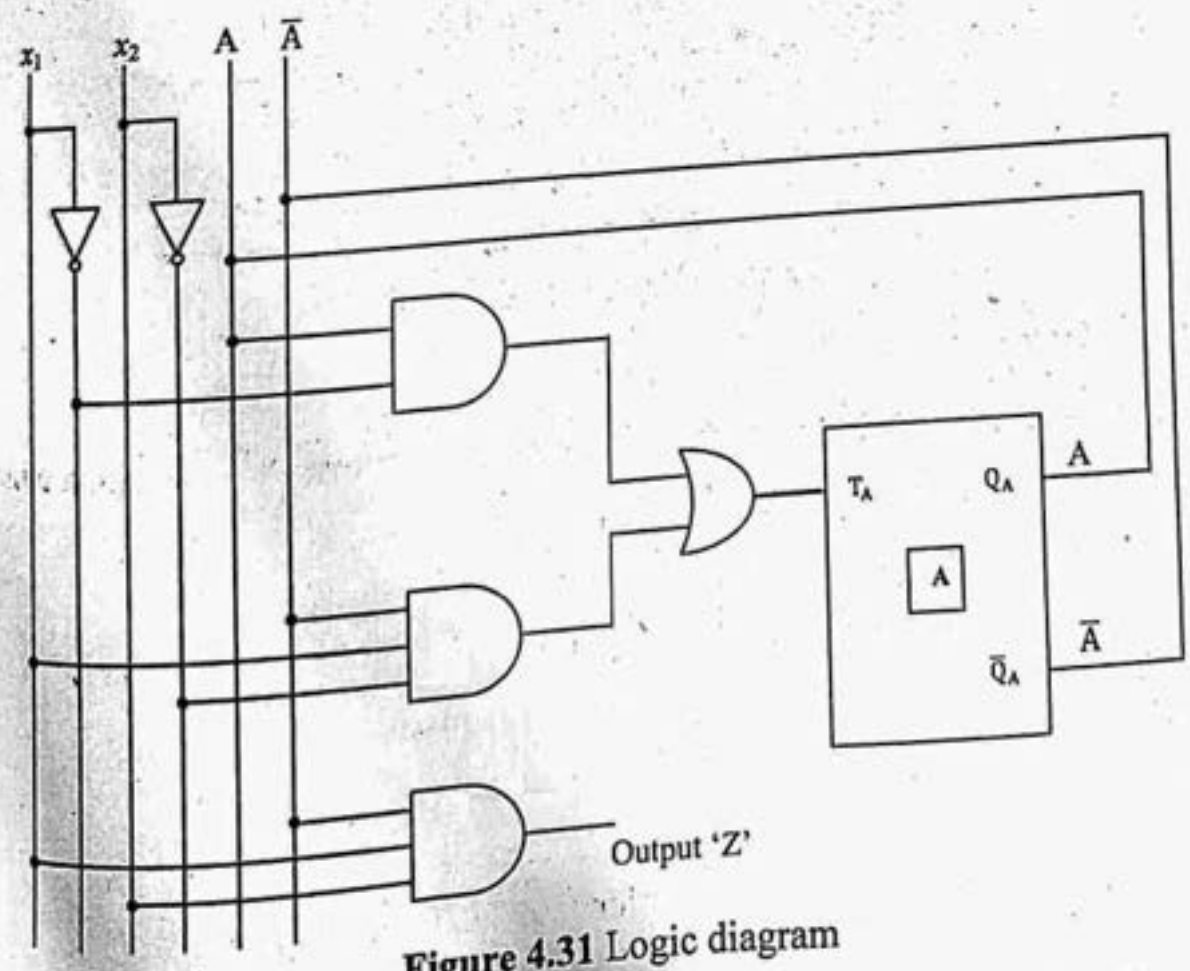


Figure 4.31 Logic diagram

Example 4.7: Design a T flip flop using logic gates.

Solution:

The T flip-flop has one excitation input and one clock input. But here we use another input P, that will function as a clock. The T flip-flop will change state if $T = 1$ and when the clock (P) changes from 1 to 0. Under all other input conditions, output Q will remain constant.

We assume that T and P do not change simultaneously. The state diagram for a T flip-flop can be drawn as shown in figure 4.32.

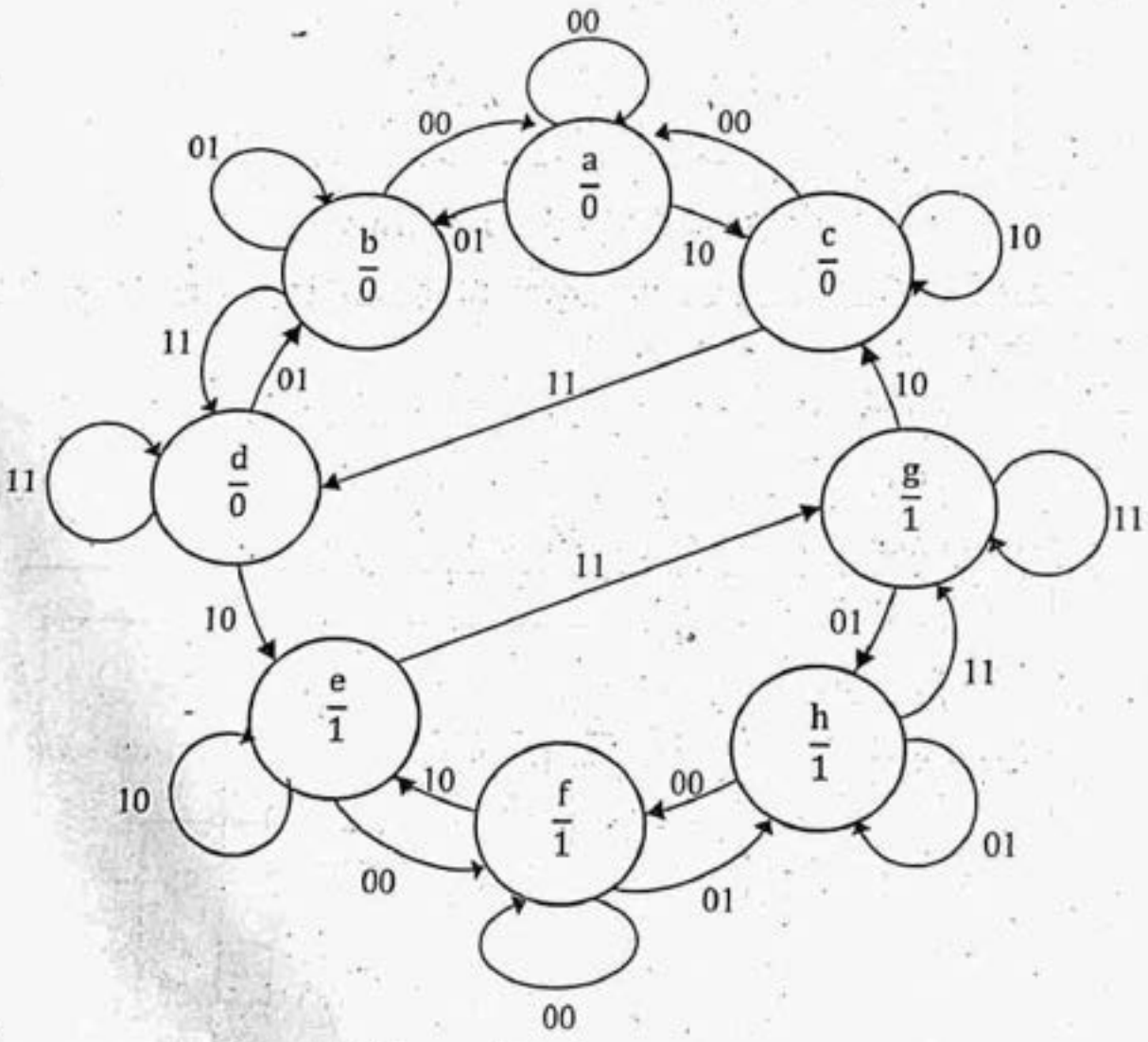


Figure 4.32 State diagram

Primitive flow table

Present state	Next state, Output Z			
	TP = 00	TP = 01	TP = 11	TP = 10
a	(a), 0	b, -	-, -	c, -
b	a, -	(b), 0	d, -	-, -
c	a, -	-, -	d, -	(c), 0
d	-, -	b, -	(d), 0	e, -
e	f, -	-, -	g, -	(e), 1
f	(f), 1	h, -	-, -	e, -
g	-, -	h, -	(g), 1	c, -
h	f, -	(h), 1	g, -	-, -

Table 4.37 Primitive flow table

The primitive flow table can be minimized by merging the states 'a', 'b' and 'c' to a single state 'S_0'

$(a, b, c) \rightarrow S_0$

$d \rightarrow S_1$

Also we can merge the states 'e', 'f' and 'h' to a single state 'S_2'

$(e, f, h) \rightarrow S_2$

$g \rightarrow S_3$

Present state	Next state, Output Z			
	TP = 00	TP = 01	TP = 11	TP = 10
S_0	(S_0), 0	(S_0), 0	S_1, -	(S_0), 0
S_1	-, -	S_0, -	(S_1), 0	S_2, -
S_2	(S_2), 1	(S_2), 1	S_3, -	(S_2), 1
S_3	-, -	S_2, -	(S_3), 1	S_0, -

Table 4.38 Reduced Primitive flow table

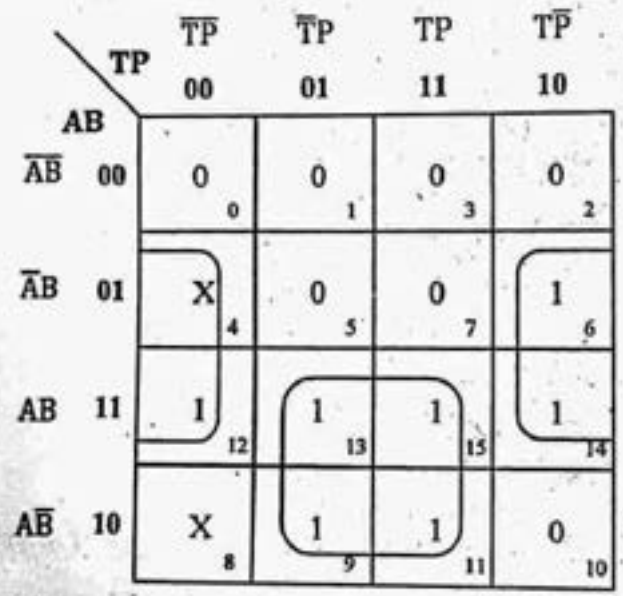
Since there are three states assign, $S_0 = 00$, $S_1 = 01$, $S_2 = 11$ and $S_3 = 10$.

Present state AB	Next state A^+B^+, Output Z			
	TP = 00	TP = 01	TP = 11	TP = 10
00	00, 0	00, 0	01, -	00, 0
01	-, -	00, -	01, 0	11, -
11	11, 1	11, 1	10, -	11, 1
10	-, -	11, -	10, 1	00, -

Table 4.39 State assignment

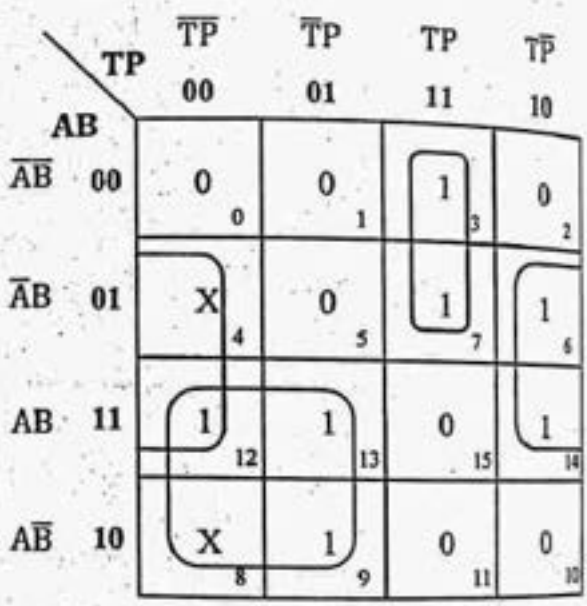
Since we are not designing using flip-flops, assume 'don't cares' for unspecified.

K-map for A^+



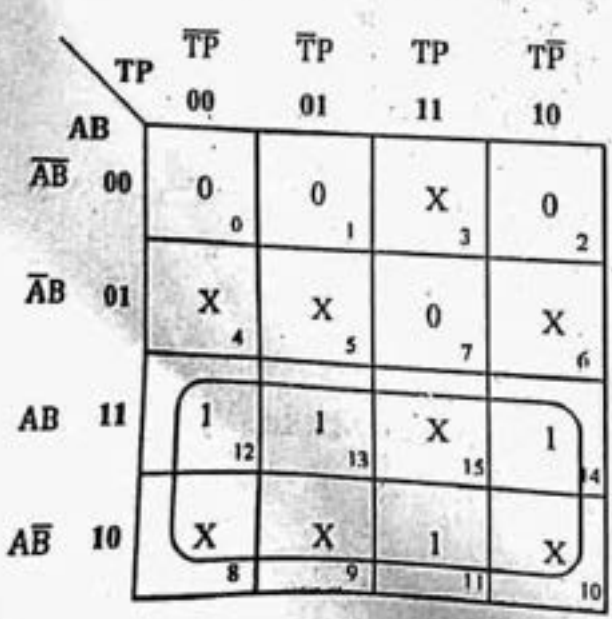
$$A^+ = B\overline{P} + AP$$

K-map for B^+



$$B^+ = A\overline{T} + B\overline{P} + \overline{A}TP$$

K-map for Z



$$Z = A$$

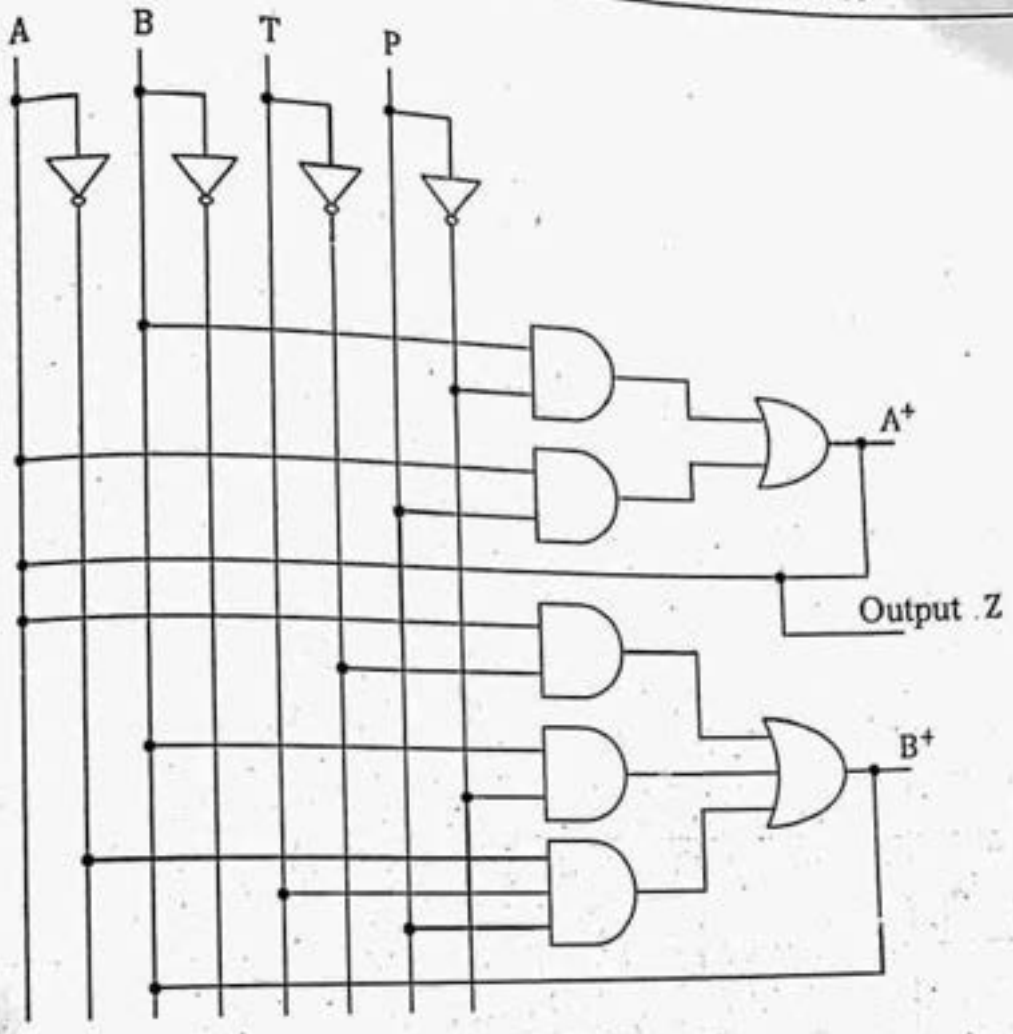


Figure 4.33 Logic diagram

Example 4.8: ***Design an asynchronous sequential circuit with inputs A and B and an output Y. Initially and at any time if both the inputs are 0, the output Y is equal to 0. When A or B becomes 1, Y becomes 1. When the other input also becomes 1, Y becomes 0. The output stays at 0 until circuit goes back to initial state.***

Solution : It is given that A and B are inputs and Y is the output.

If A or B becomes '1', then the output will be Y=1. But if both A and B are '1', then the output will be Y=0. For other cases output will be Y=0.

Step 1: Draw the state diagram.

The state diagram for the given problem can be easily drawn using the diagram shown in figure 4.34. The different states are a, b, c, d, e and f.

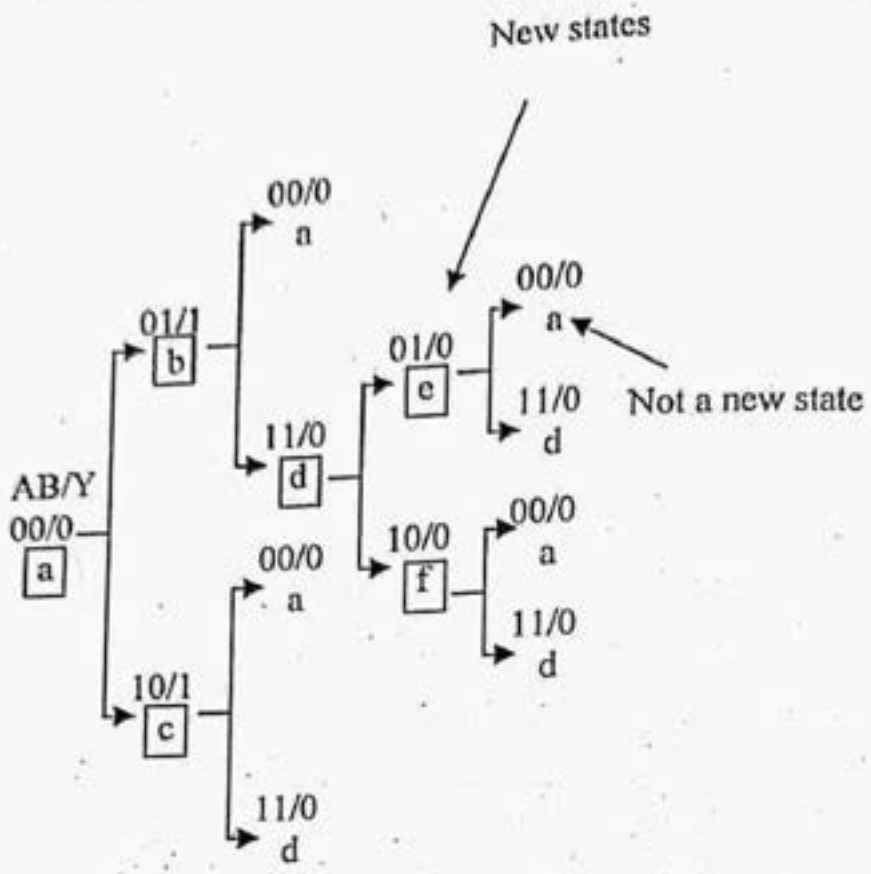


Figure 4.34

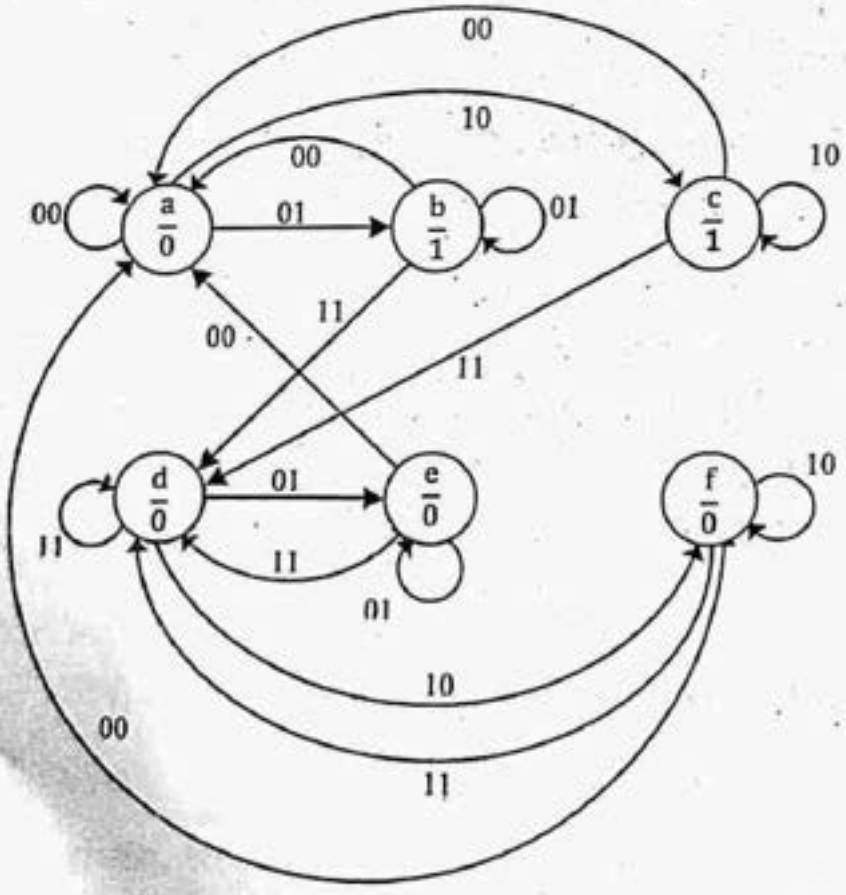


Figure 4.35 State diagram

Step 2: Construct the primitive flow table

Present state	Next state, Output Y			
	AB=00	AB =01	AB=11	AB=10
a	(a), 0	b , -	- , -	c , -
b	a , -	(b), 1	d , -	- , -
c	a , -	- , -	d , -	(c), 1
d	- , -	e , -	(d), 0	f , -
e	a , -	(e), 0	d , -	- , -
f	a , -	- , -	d , -	(f), 0

Table 4.40 Primitive flow table

Here the circle represents that the state is stable.

Step 3: Reduce the primitive flow table

The primitive flow table can be minimized by merging the states a, b, c to a single state 'S_0' because the next state and the output for the states a, b, c are same.

$(a, b, c) \rightarrow S_0$

Also we can merge the states d, e, f to a single state 'S_1' because the next state and the output for the states d, e, f are same.

$(d, e, f) \rightarrow S_1$

Present state	Next state, Output Y			
	AB=00	AB =01	AB=11	AB=10
S_0	(S_0), 0	(S_0), 1	S_1 , -	(S_0), 1
S_1	S_0 , -	(S_1), 0	(S_1), 0	(S_1), 0

Table 4.41 Reduced primitive flow table

Step 4: Assign binary values to states based on race free state assignment.

Here, there are two states S_0 and S_1. So assign S_0=0, S_1=1. Let the present state be Q and the Next state be Q^+. Since the present states and next states differ by single bit, the circuit for this flow table will be free from races.

Present state Q	Next state Q^+, Output Y			
	AB=00	AB =01	AB=11	AB=10
0	0 , 0	0 , 1	1 , -	0 , 1
1	0 , -	1 , 0	1 , 0	1 , 0

Table 4.42 State assignment

Step 5: Find the output table.

By using the excitation table of D flip-flop shown in table 4.43, draw the output table.

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

Table 4.43 Excitation table of D flip-flop

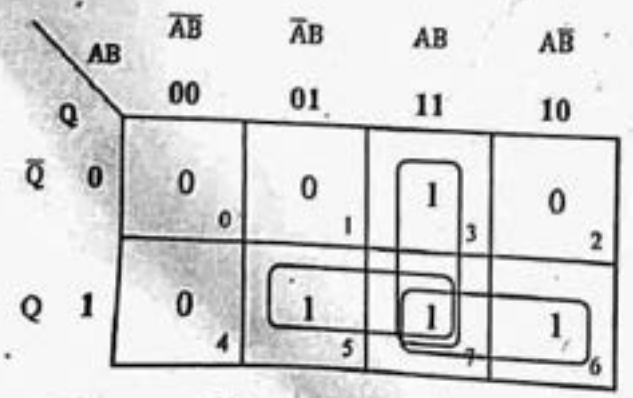
Present state Q	Flip-flop input D, Output Y			
	AB=00	AB =01	AB=11	AB=10
0	0 , 0	0 , 1	1 , -	0 , 1
1	0 , -	1 , 0	1 , 0	1 , 0

Table 4.44 Output table

Step 6: Find the flip-flop input equations and output equations using K-map.

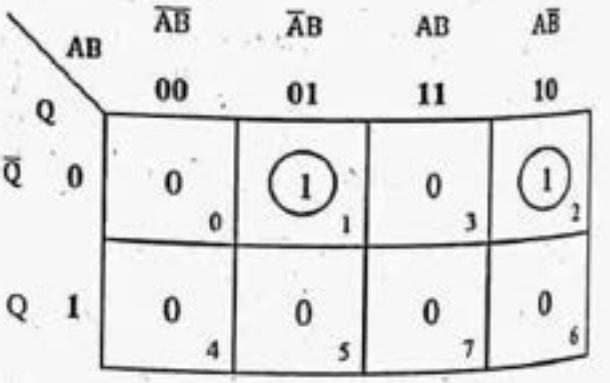
Assume '0' for unspecified outputs.

K-map for D



$D = AB + QB + QA$

K-map for Y



$Y = \overline{Q}\overline{A}B + \overline{Q}A\overline{B}$

$Y = \overline{Q}(\overline{A}B + A\overline{B})$

$Y = \overline{Q}(A \oplus B)$

Step 7: Draw the logic diagram

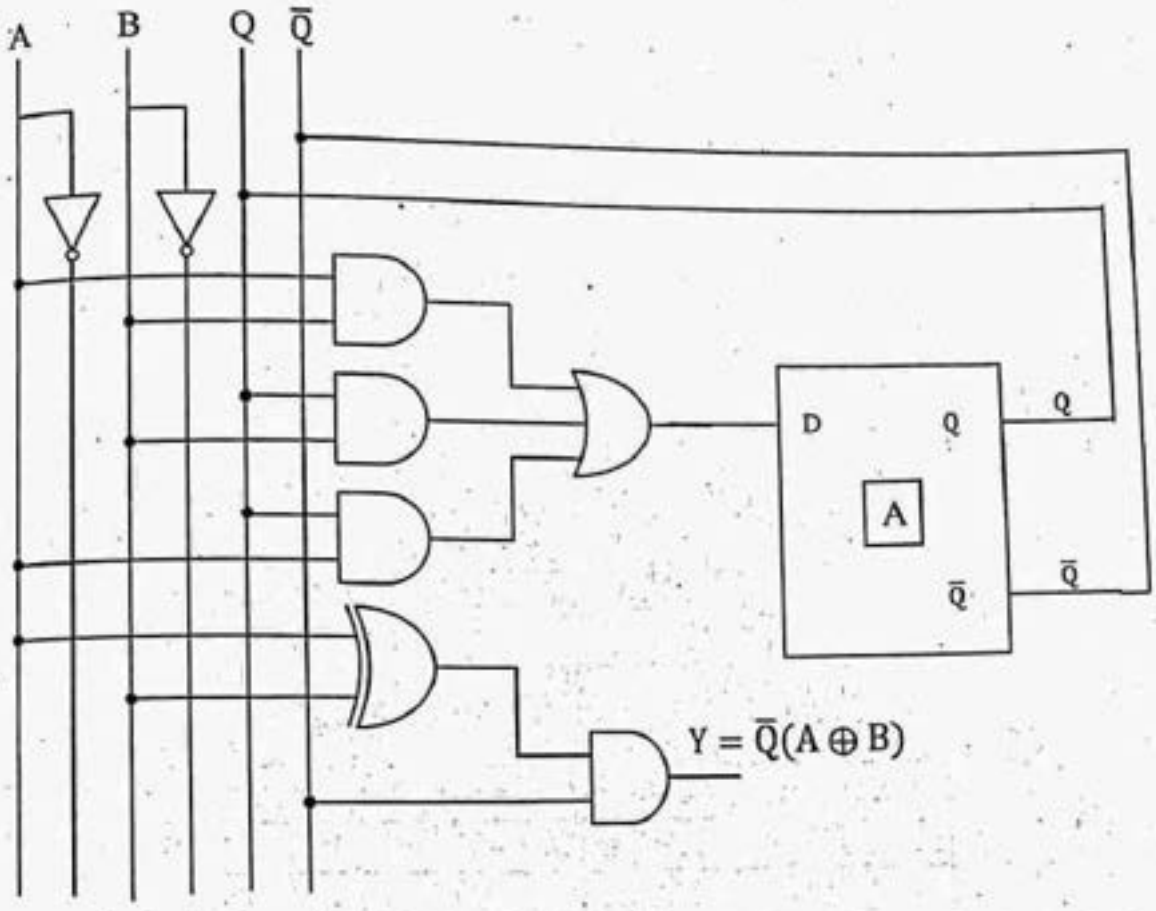


Figure 4.36 Logic diagram

4.7 PROBLEMS IN ASYNCHRONOUS CIRCUITS

4.7.1 Races

If an input change induces change in two or more feedback variables then it is known as race condition.

Races are of two types,

i. Critical race
ii. Non critical race

4.7.1.1 Critical race

If an input change induces the circuit to go to the same wrong state. Then it is termed as critical races. Critical races must be avoided in an asynchronous circuit.

Consider the flow table shown in table 4.45. Assume that the circuit is in state 'e' ($A^+B^+ = 00$), $x_1x_2 = 01$, AB=00. Consider x_2 is changed from 1 to 0.

The circuit begin to switch from state 'e' to state 'a' ($A^+B^+ = 11$). But due to unequal propagation delay one of the state variables become 1 and other remains 0 (the next state A^+B^+ become either $A^+B^+ = 01$ or $A^+B^+ = 10$). If A^+B^+ become 01 the circuit goes to wrong stable state 'b'.

Present state AB	Next state A^+B^+			
	$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 11$	$x_1x_2 = 10$
00	11 (a)	(00) (e)	11 (i)	01 (m)
01	(01) (b)	11 (f)	11 (j)	(10) (n)
11	10 (c)	(11) (g)	(11) (k)	10 (o)
10	(10) (d)	00 (h)	11 (l)	00 (p)

Table 4.45 Flow table

If A^+B^+ become 10 the circuit goes to wrong stable state 'd'. Hence the final state will be wrong for either stable state b=01 or d=10. Because the circuit has to switch from 00 to 11. This situation is called critical race.

4.7.1.2 Non-critical race

Consider the flow table shown in table 4.45. Assume that the circuit is in state 'e' ($A^+B^+ = 00$), $x_1x_2 = 01$, AB=00. Consider x_1 is changed from 0 to 1. The circuit begins to switch from state 'e' to state 'i'. But due to unequal propagation delay one of the state variables become 1 and the other remains 0 (the next state A^+B^+ become either $A^+B^+ = 01$ or $A^+B^+ = 10$).

If A^+B^+ becomes 01 the circuit goes to 11 (state j) which is unstable state and then finally the circuit goes to stable state 'k'.

If A^+B^+ become 10 the circuit goes to 11 (state l) which is also unstable state and then finally the circuit goes to stable state 'k'.

Here the circuit reaches a correct final stable state after transition through unstable states. So we can define critical race as, if an input change induces the circuit to a wrong state which is unstable will finally reaches a correct stable state. This condition is known as non-critical race.

4.7.2 Cycles

Consider the flow table shown in table 4.45. Consider that the circuit is initially stable at state 'k'. Here the input is $x_1x_2 = 11$. Now when the input x_2 changes to 0 ($x_1x_2 = 10$) the next state becomes $A^+B^+ = 10$ (state o) which is unstable. For the present state AB=10 and $x_1x_2 = 10$, the circuit switches to $A^+B^+ = 00$ (state p) which is unstable. For the present state AB=00 and $x_1x_2 = 10$, the circuit switches to $A^+B^+ = 01$ (state m) which is unstable. For the present state AB=01 and $x_1x_2 = 10$, the circuit switches to $A^+B^+ = 10$ (state n) which is stable.

Here the circuit has gone through unstable states o, p, m and finally reaches the stable state n. So if an input change induces a feedback transition through more than one unstable state, then such a situation is known as cycle.

4.8 HAZARDS

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

In a combinational circuit, hazards may cause temporary false output. But in sequential circuits hazards may cause a transition to a wrong state.

A hazard can be classified into,

a. Static hazards.
b. Dynamic hazards
c. Essential hazards

4.8.1 Static Hazards

A static hazard is a hazard that occurs in combinational circuits which results in a single momentary incorrect output due to change in a single input variable when the output is expected to remain in the same state.

The static hazard may be either Static-0 or Static-1

4.8.1.1 Static-0 Hazards

Due to a change in single input variable, if the output momentarily goes to state-1 when the output is expected to remain in state-0. Then such type of hazard is said to be static-0.

Consider the following combinational circuit shown in figure 4.37.

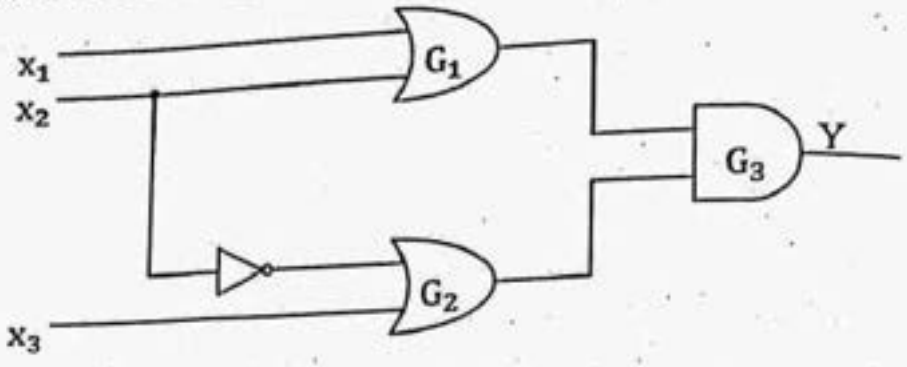


Figure 4.37

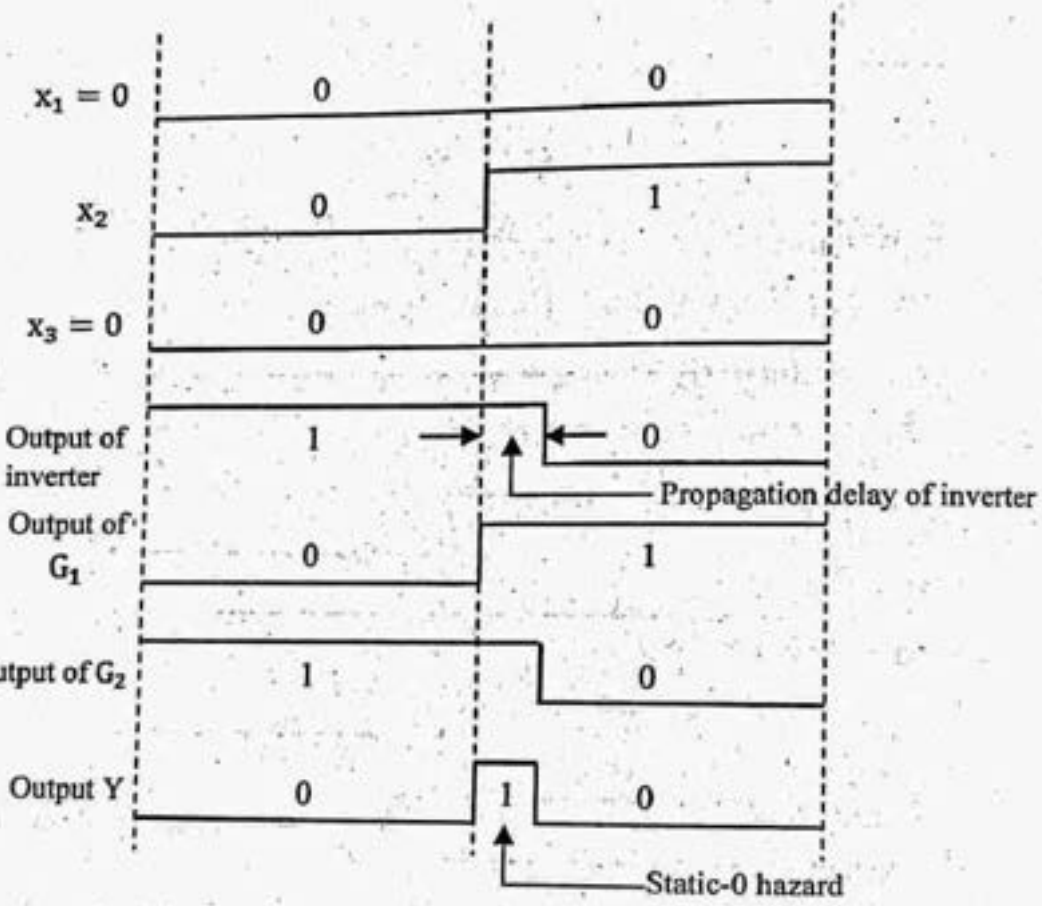


Figure 4.38 Timing diagram

Initially assume $x_1 = 0$, $x_2 = 0$ and $x_3 = 0$. Therefore the output of G_1 is 0, the output of G_2 is 1 and the circuit output is 0. Now consider x_2 is changed from 0 to 1. Then the output of G_1 changes to 1, the output of G_2 changes to 0 and the circuit output remains 0. However, the output momentarily goes to 1, if the propagation delay through the inverter is taken into consideration. The delay in the inverter causes the output of G_2 to remain 1 until the propagation delay completes. Hence during the propagation delay, the output of G_1 is 1 and the output of G_2 is 1. Hence the output of the circuit is 1, during which causes static-0 hazard

4.8.1.2 Static-1 Hazards

Due to a change in single input variable, if the output momentarily goes to state-0 when the output is expected to remain in state-1. Then such type of hazard is said to be Static-1 hazard.

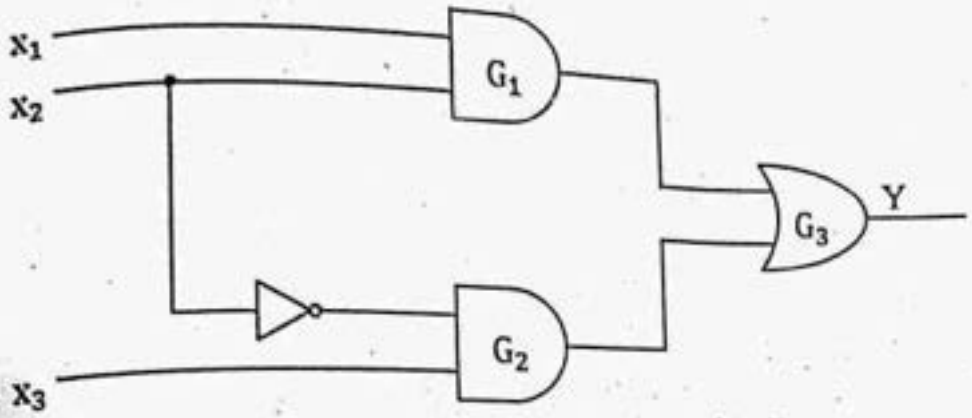


Figure 4.39

Consider the combinational circuit shown in figure 4.39

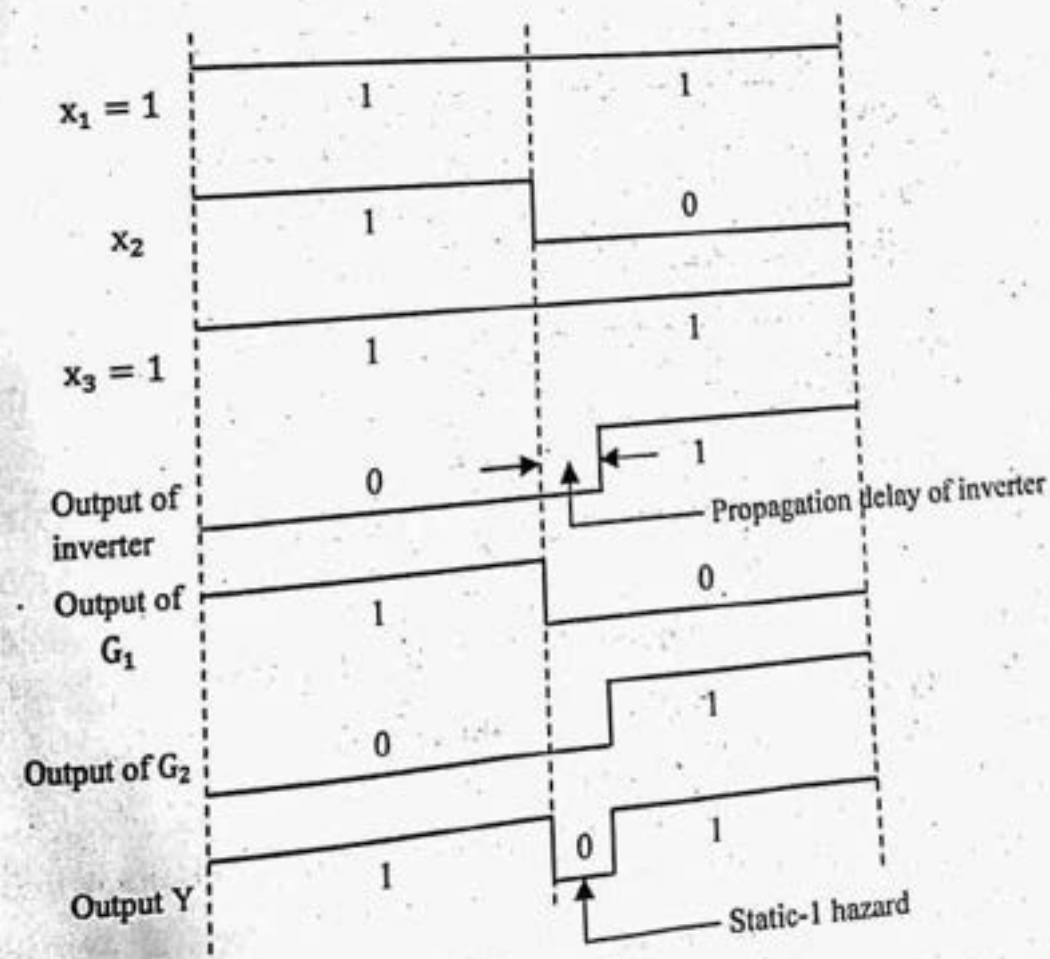


Figure 4.40 Timing diagram

Initially assume $x_1 = 1$, $x_2 = 1$ and $x_3 = 1$. Therefore the output of G_1 is 1,the output of G_2 is 0 and the circuit output is 1

Now consider x_2 is changed from 1 to 0. Then the output of G_1 changes to 0, the output of G_2 changes to 1 and the circuit output remains 1.

However, the output momentarily goes to 0, if the propagation delay through the inverter is taken into consideration. The delay in the inverter causes the output of G_2 to remain 0 until the propagation delay completes. Hence during the propagation delay, the output of G_1 is 0 and the output of G_2 is 0. Hence the output of the circuit is 0, during which causes Static-1 hazard

4.8.2 Dynamic Hazards

The output changes three or more times when it is supposed to change from 0 to 1 or from 1 to 0.

Figure 4.41 Dynamic Hazards

4.8.3 Essential Hazards

Essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks. It is an operational error generally caused by an excessive delay to a feedback variable in response to an input change. Since the essential hazard is caused by unequal delay along two or more paths. These hazards cannot be eliminated by adding redundant gates as in static hazards. Such hazards can be eliminated by adjusting the amount of delays in the affected path.

4.8.4 Design of Hazard Free switching circuits

The hazard exists because of the change of input results in a different product terms covering two minterms or different sum terms covering two maxterms. Static and dynamic hazards can be eliminated by enclosing two minterms or maxterms. The static-1 hazard occurring in the circuit shown in figure 4.39 can be eliminated as follows. The output equation of the given circuit is $Y = x_1x_2 + \bar{x}_2x_3$

The K-map for the above expression can be drawn as follows.

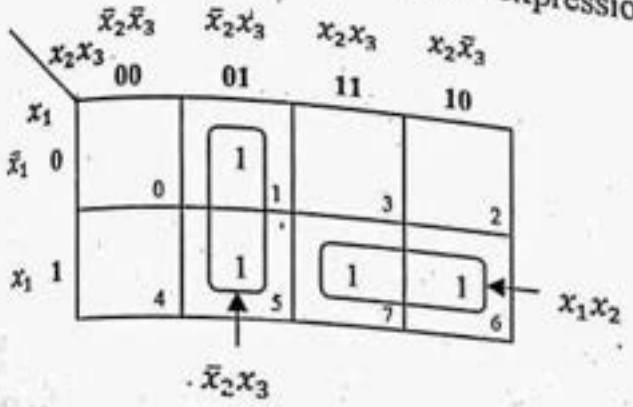


Figure 4.42

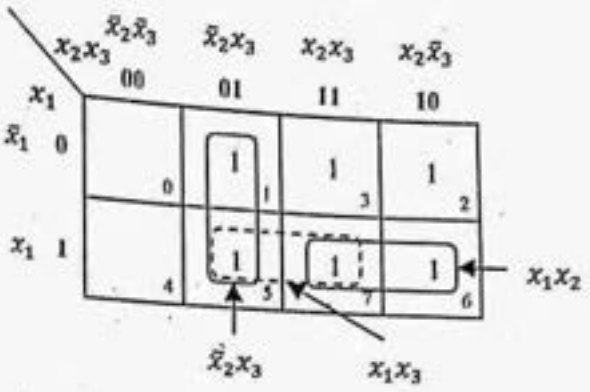


Figure 4.43

By enclosing the two minterms by another minterms x_1x_3, we get

$$Y = x_1x_2 + x_1x_3 + \bar{x}_2x_3$$

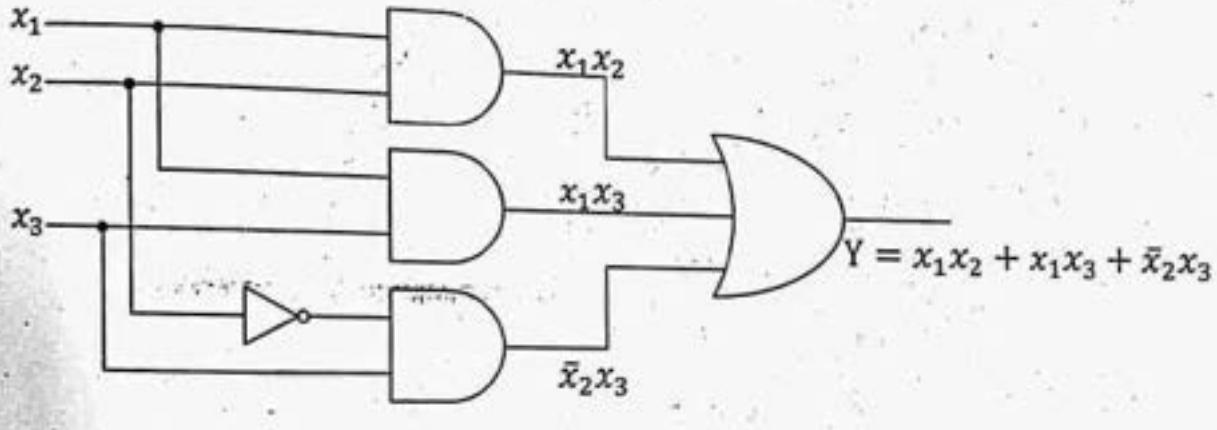


Figure 4.44 Hazard free circuit

Example 4.9: For a given Boolean function obtain the hazard free circuit.

$$F(A, B, C, D) = \sum_m(1, 3, 6, 7, 13, 15)$$

Solution: Hazard free circuit can be designed using following steps.

(a) Group the minterms using K-map.

(b) Enclose the grouped minterms and find the expression.

(c) Draw the hazard free circuit.

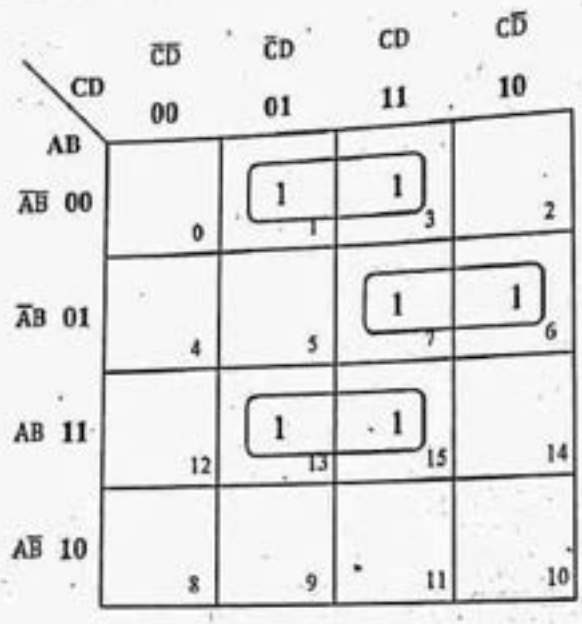


Figure 4.45

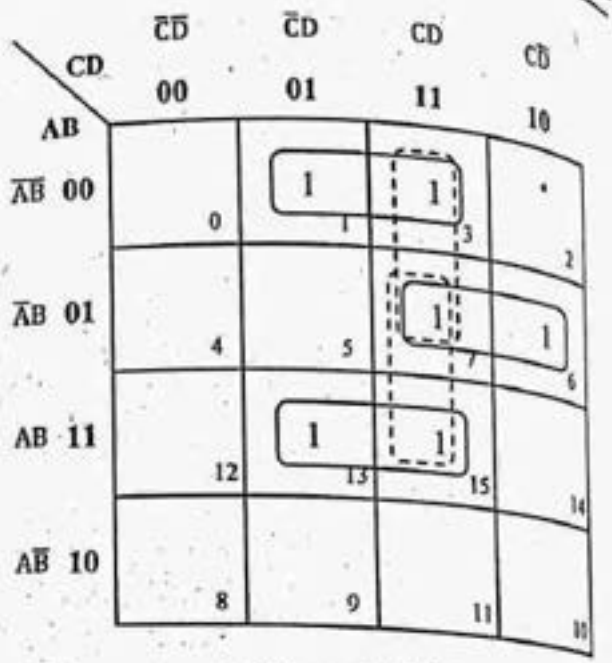


Figure 4.46

$$F(A,B,C,D)=\bar{A}\bar{B}D + \bar{A}BC + ABD + A\bar{C}D + BCD$$

Logic diagram

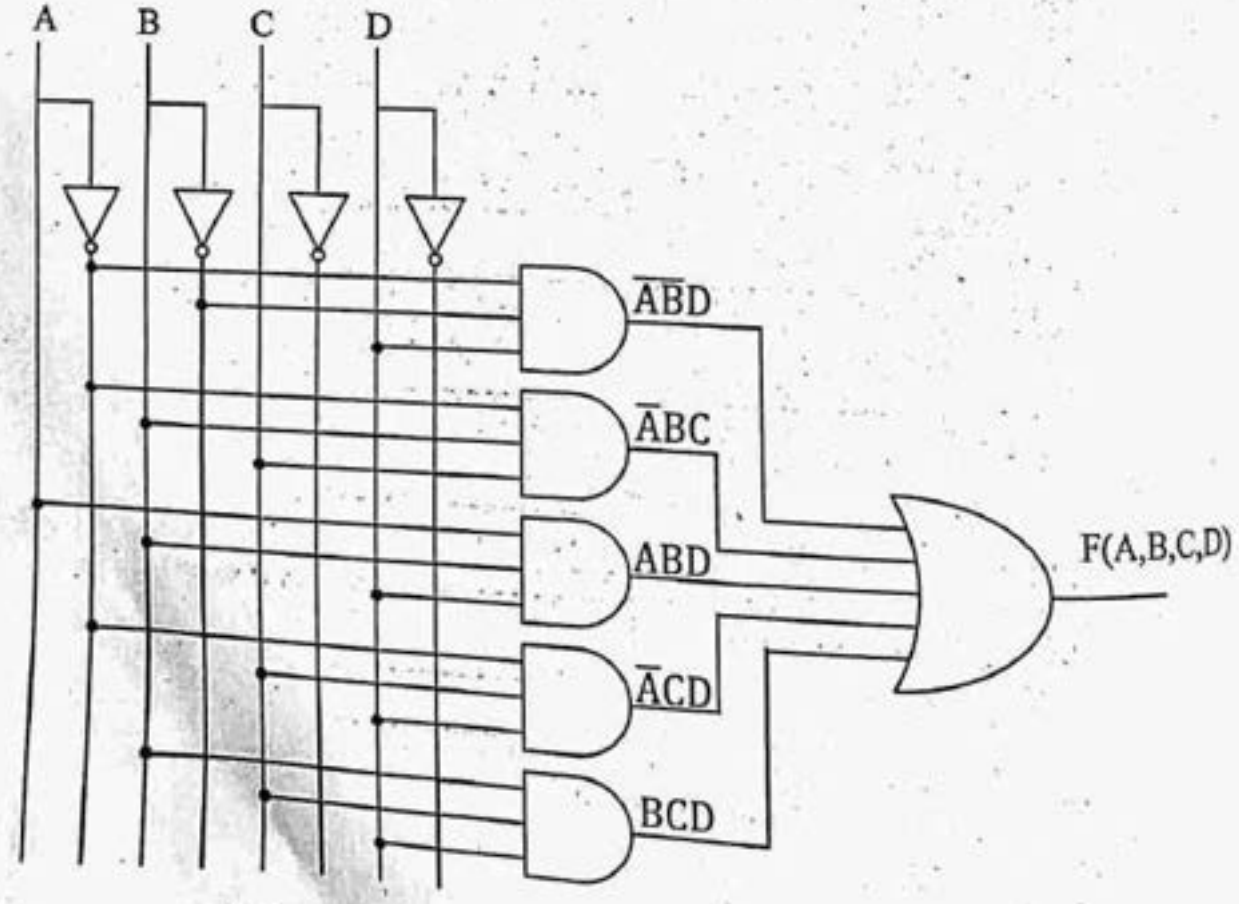


Figure 4.47 Logic diagram of hazard free circuit

Example 4.10: ***Implement the following logic and analyze for the pressure of any hazard*** $f = x_1x_2 + x_1'x_3$ ***. If hazard is present briefly explain the type of hazard and design-free circuit.***

Solution

The logic diagram for the given expression $f = x_1x_2 + x_1'x_3$ is

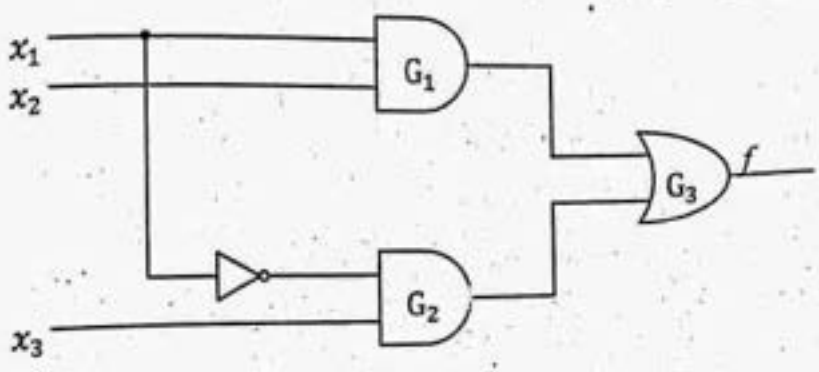


Figure 4.48

Due to a change in single input variable, if the output momentarily goes to state-0 when the output is expected to remain in state-1. Then such type of hazard is said to be Static-1 hazard.

Consider the combinational circuit shown in figure 4.48

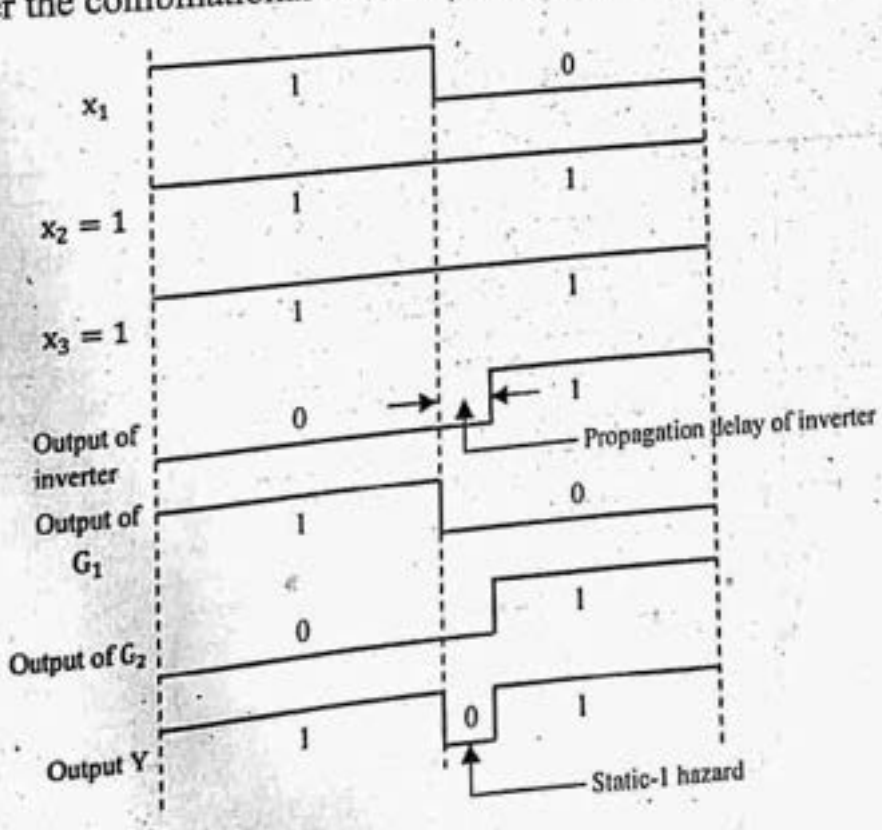


Figure 4.49 Timing diagram

Initially assume $x_1 = 1, x_2 = 1$ and $x_3 = 1$. Therefore the output of G_1 is 1, the output of G_2 is 0 and the circuit output is 1

Now consider x_1 is changed from 1 to 0. Then the output of G_1 changes to 0, the output of G_2 changes to 1 and the circuit output remains 1.

However, the output momentarily goes to 0, if the propagation delay through the inverter is taken into consideration. The delay in the inverter causes the output of G_2 to remain 0 until the propagation delay completes. Hence during the propagation delay, the output of G_1 is 0 and the output of G_2 is 0. Hence the output of the circuit is 0, during which causes Static-1 hazard

The static-1 hazard occurring in the circuit shown in figure 4.48 can be eliminated as follows. The output equation of the given circuit is

$$f = x_1x_2 + x_1'x_3$$

The K-map for the above expression can be drawn as follows.

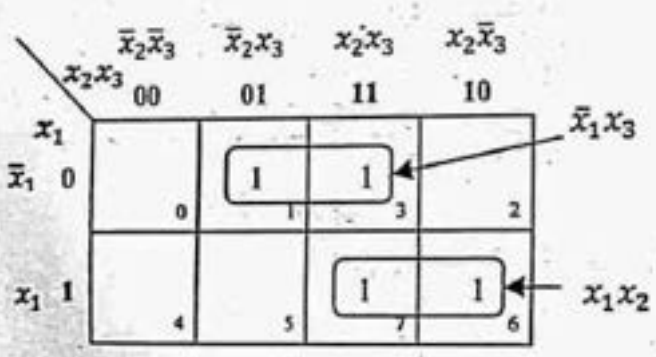


Figure 4.50

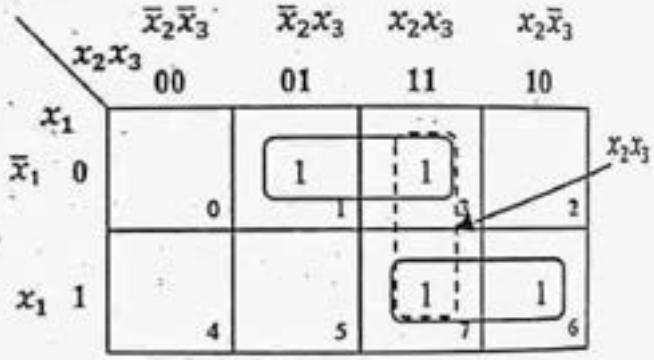


Figure 4.51

By enclosing the two minterms by another minterms x_2x_3, we get

$$Y = \bar{x}_1x_3 + x_1x_2 + x_2x_3$$

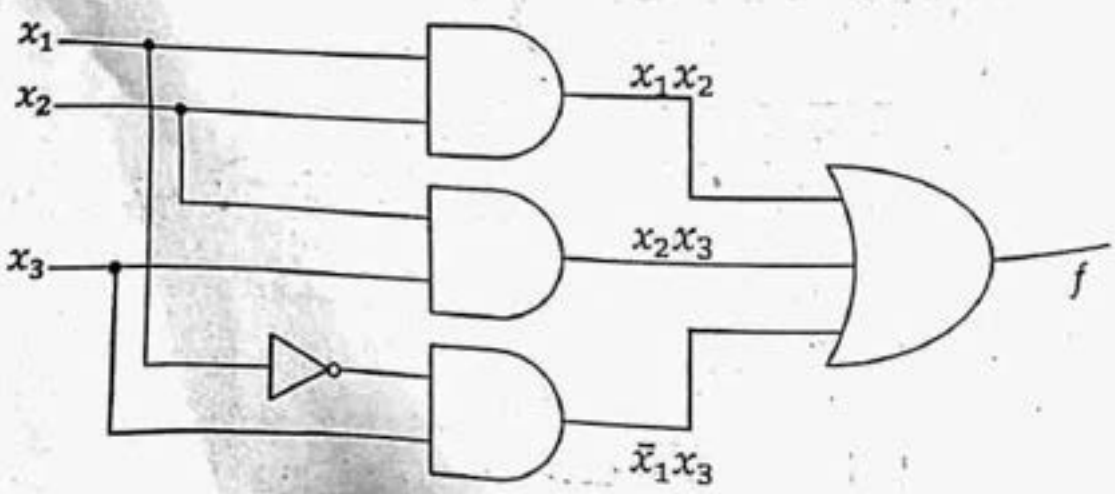


Figure 4.52 Hazard free circuit

Example 4.11: Implement the switching function $F(A,B,C,D) = \sum_m(1,3,5,7,8,9,14,15)$ *by a static hazard free two level AND-OR gate network.*

Solution: Hazard free circuit can be designed using following steps.

(a) Group the minterms using K-map.
(b) Enclose the grouped minterms and find the expression.
(c) Draw the hazard free circuit.

Given $F(A,B,C,D) = \sum_m(1,3,5,7,8,9,14,15)$

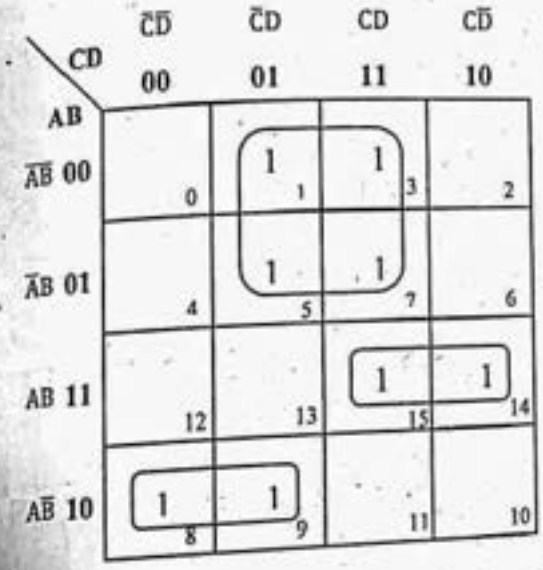


Figure 4.53

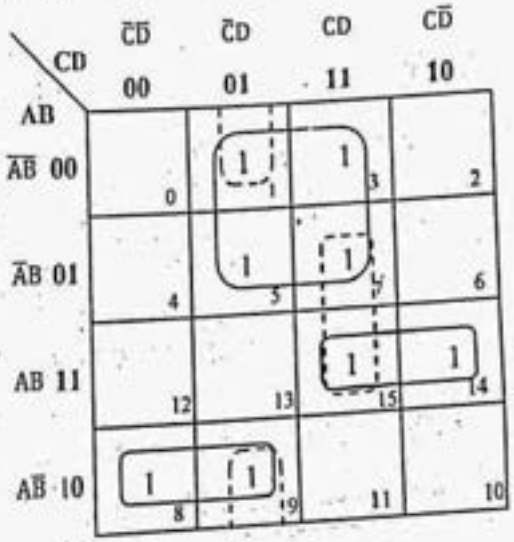


Figure 4.54

$$F(A,B,C,D) = \bar{A}D + ABC + A\bar{B}\bar{C} + BCD + \bar{B}\bar{C}D$$

Logic diagram

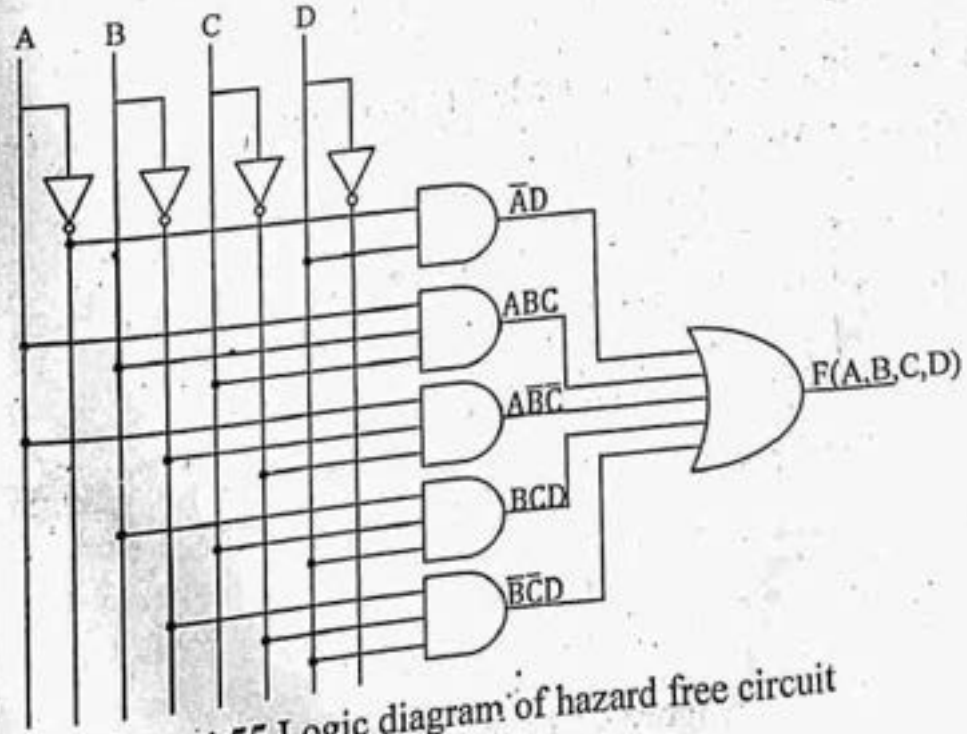


Figure 4.55 Logic diagram of hazard free circuit

Example 4.12: Give hazard free realization for the following Boolean function

$$F(A,B,C,D) = \sum_m(0,2,6,7,8,10,12)$$

Solution: Hazard free circuit can be designed using following steps.

(a) Group the minterms using K-map.
(b) Enclose the grouped minterms and find the expression.
(c) Draw the hazard free circuit.

$$F(A,B,C,D) = \sum_m(0,2,6,7,8,10,12)$$

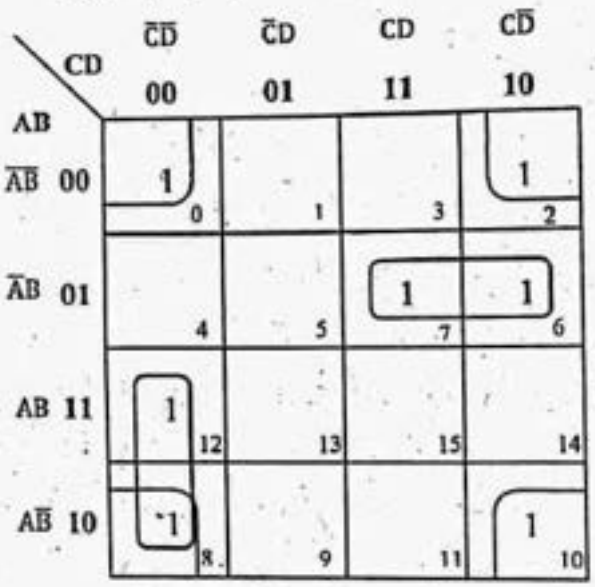


Figure 4.56

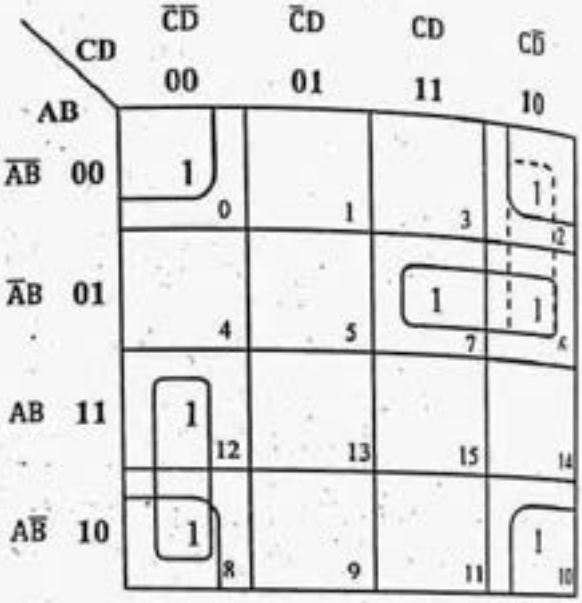


Figure 4.57

$$F(A,B,C,D) = \bar{B}\bar{D} + \bar{A}BC + A\bar{C}\bar{D} + \bar{A}C\bar{D}$$

Logic diagram

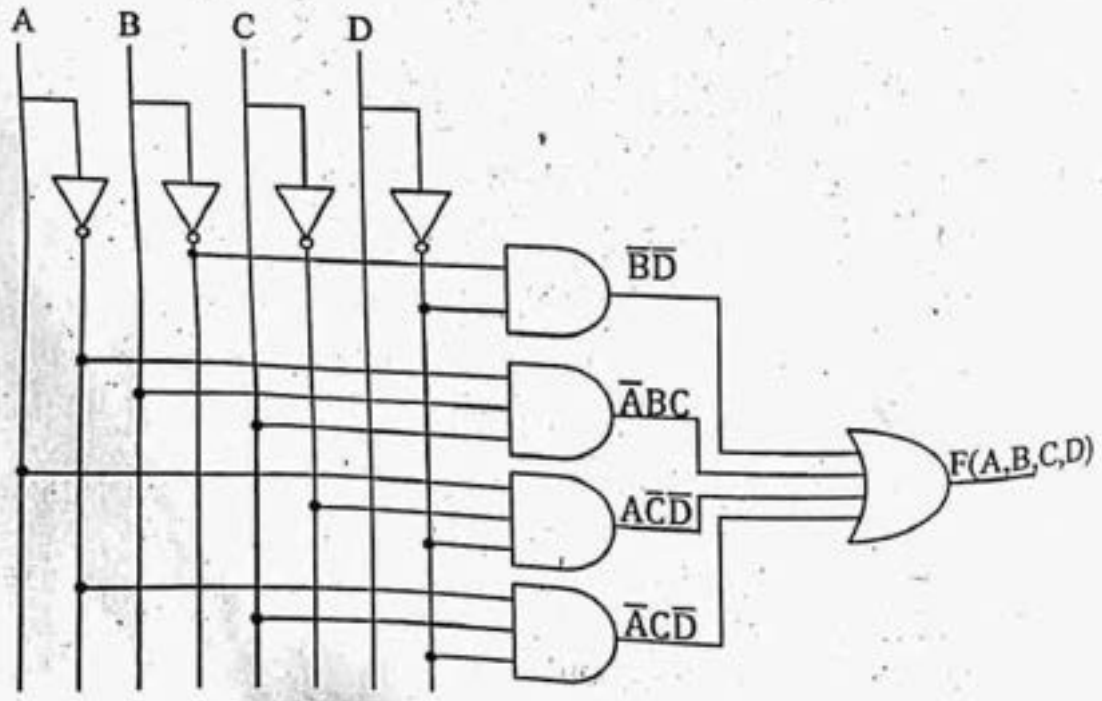


Figure 4.58 Logic diagram of Hazard free circuit

4.9 ALGORITHMIC STATE MACHINES (ASM)

An ASM is a flow chart representation that describes the behavior of a sequential circuit. An ASM chart differs from a conventional flow chart, a conventional flow chart describes the procedure steps and decision path for an algorithm without concern for their time relationship. An ASM chart also describes the timing relationship between the states and outputs.

ASM chart

An ASM chart mainly consists of three elements.

1. State box
2. Decision box
3. Conditional box

State box

State box represents one state of ASM. It is rectangle in shape. It has single entry path and an exit path. The name of the state is written on the left side of the state box. The binary value of the state is placed at the upper right corner. The output is specified inside the rectangular box.

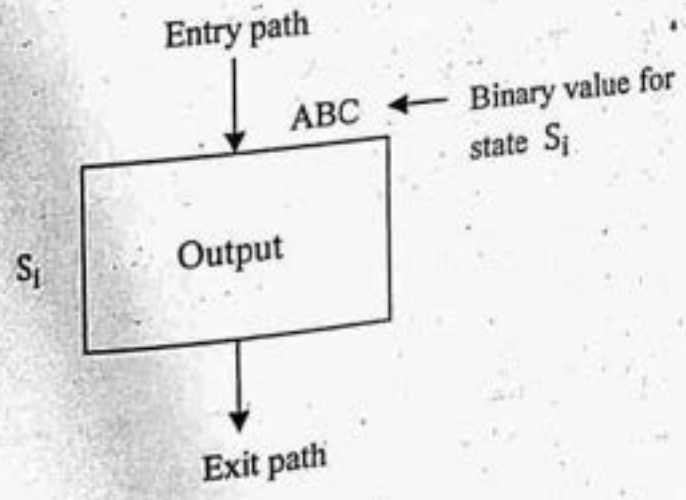


Figure 4.59 State box

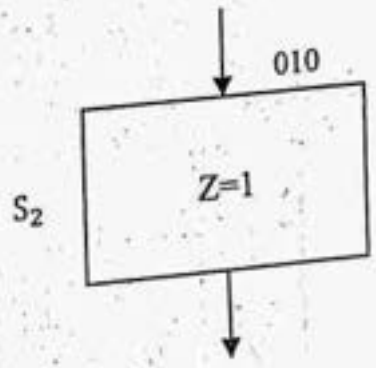


Figure 4.60 Example

Decision box

Decision box describes the effect of an input on the control subsystem. Decision box is a diamond shape box with single entry path and two or more exit path. The input condition to be tested is written inside the diamond box. If the condition is true one exit path is taken, and if the condition is false another exit path is taken.

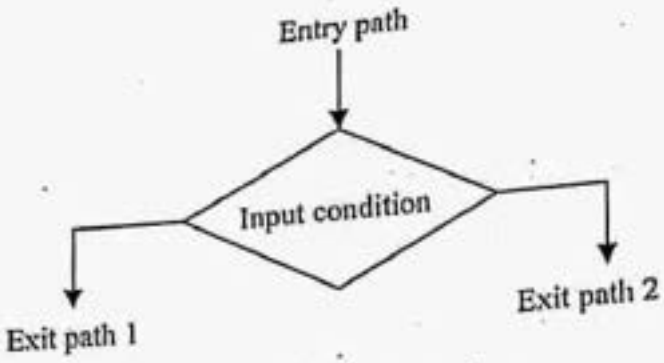


Figure 4.61 Decision box

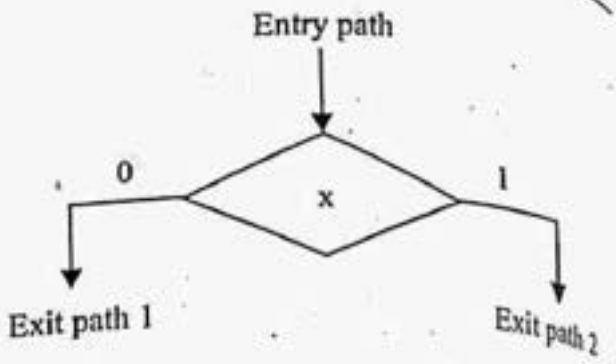


Figure 4.62 Example

Conditional box

Conditional box is oval in shape. The entry path of a conditional box always comes from one of the exit paths of a decision box. The conditional outputs are written inside the conditional box.

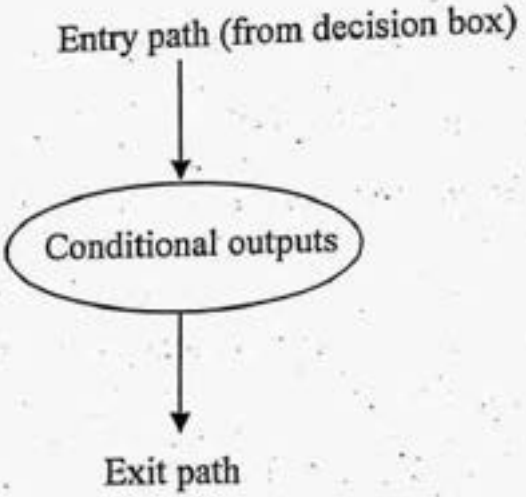


Figure 4.63 Conditional box

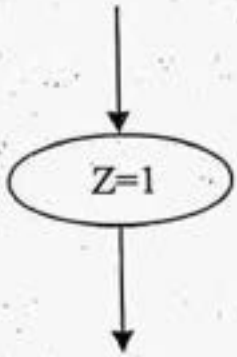


Figure 4.64 Example

Example 4.13: Draw the ASM chart for a mod-5 counter.

Solution:

Let $a = 000;$

$b = 001;$

$c = 010;$

$d = 011;$

$e = 100$

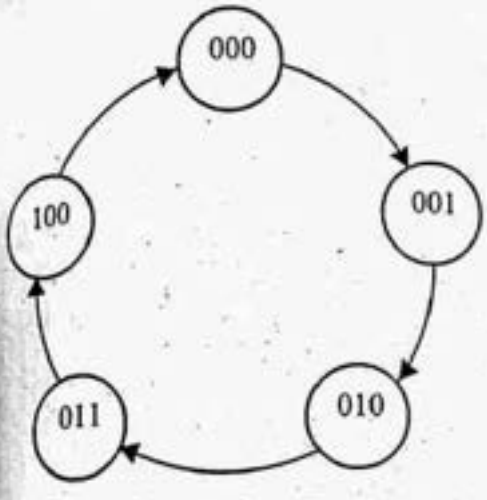


Figure 4.65 State diagram

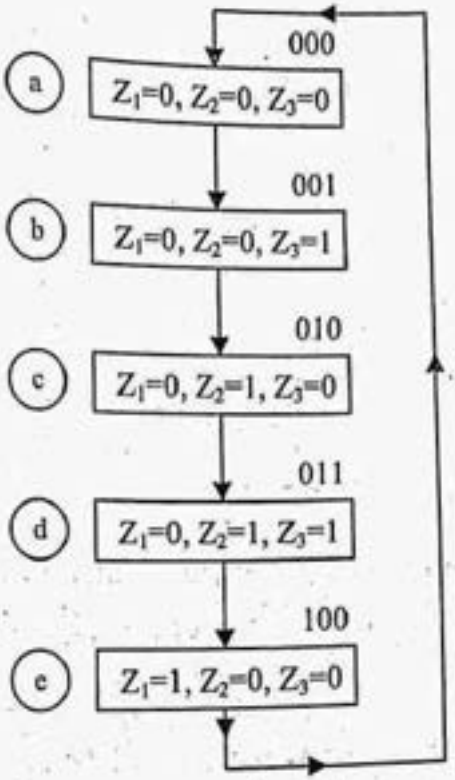


Figure 4.66 ASM chart

Example 4.14: Draw the ASM chart for a 3-bit up/down counter.

Solution:

Consider a input 'x'. If the is input x=1, the counter works as a up counter. If the input is x=0, the counter works as a down counter.

Let $a = 000; b = 001;\ c = 010; d = 011; e = 100; f = 101;$

$g = 110; h = 111$

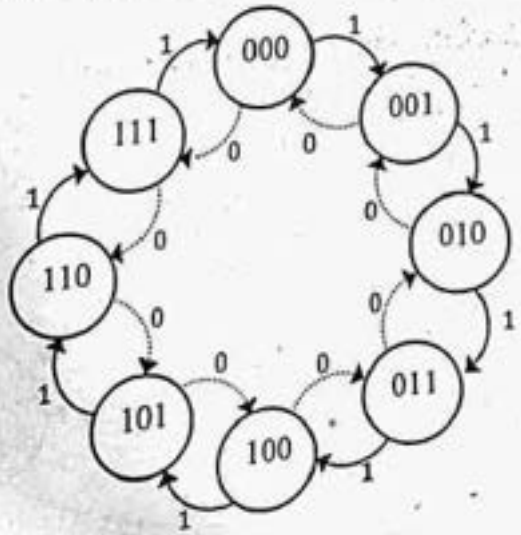


Figure 4.67 State diagram

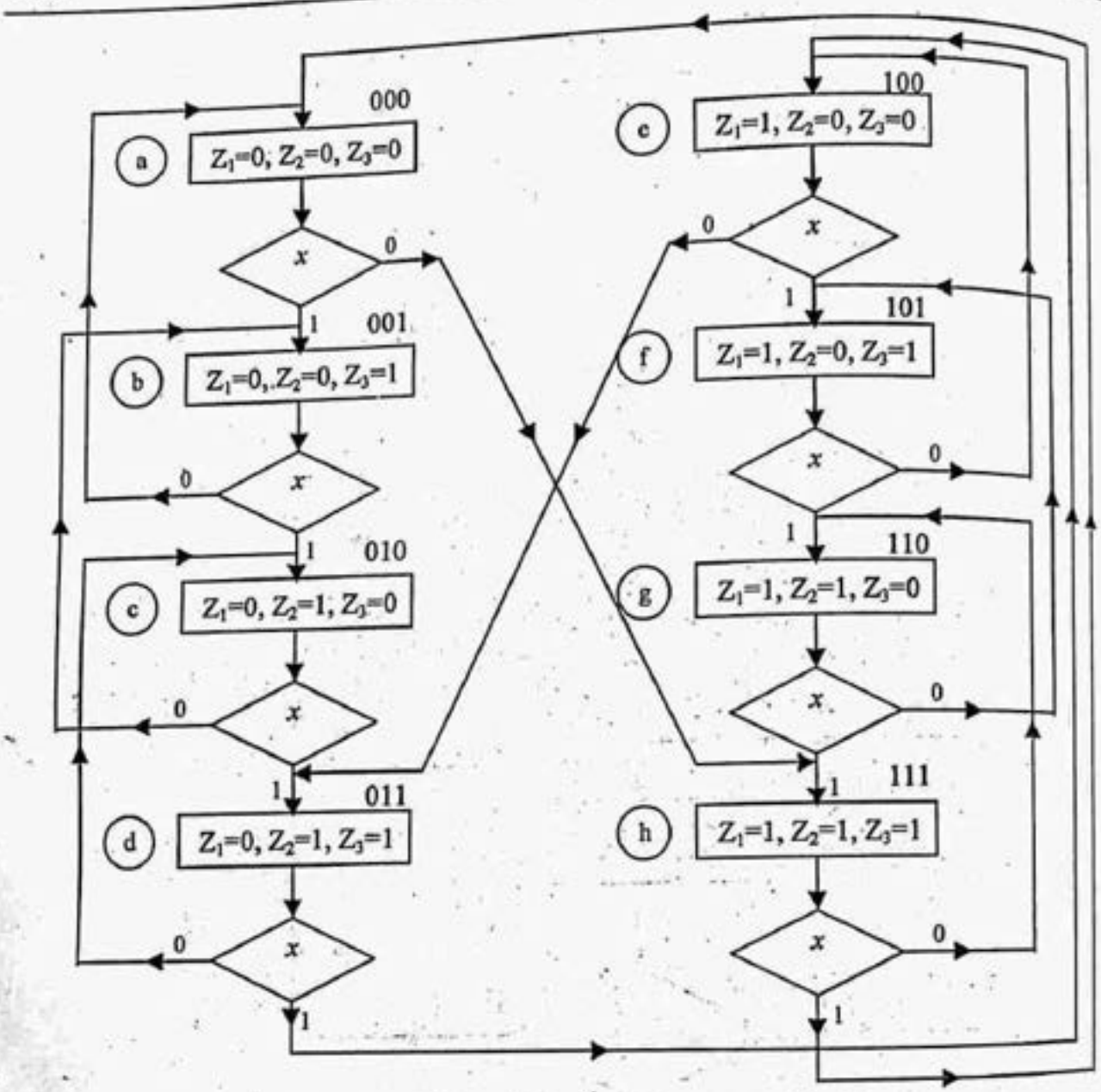


Figure 4.68 ASM chart for 3 bit up/down counter

Example 4.15: *Draw the ASM chart for the state diagram shown in figure 4.69.*

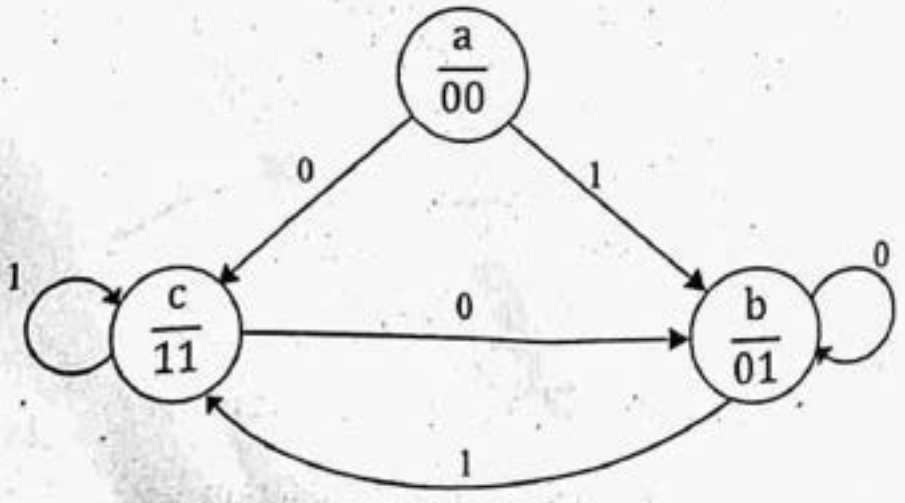


Figure 4.69 State diagram

Solution: Let Z_1, Z_2 be the outputs and x be the input

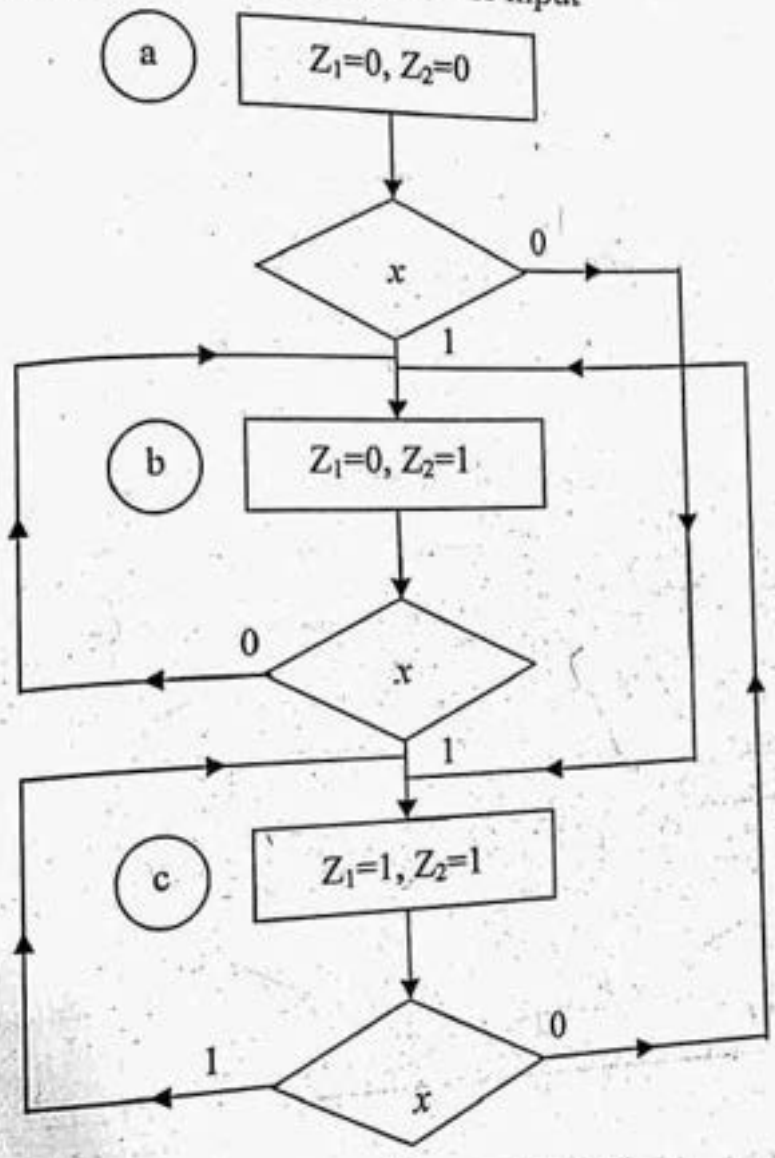


Figure 4.70 ASM chart

Example 4.16 : Draw the ASM chart for the state diagram shown in figure 4.71.

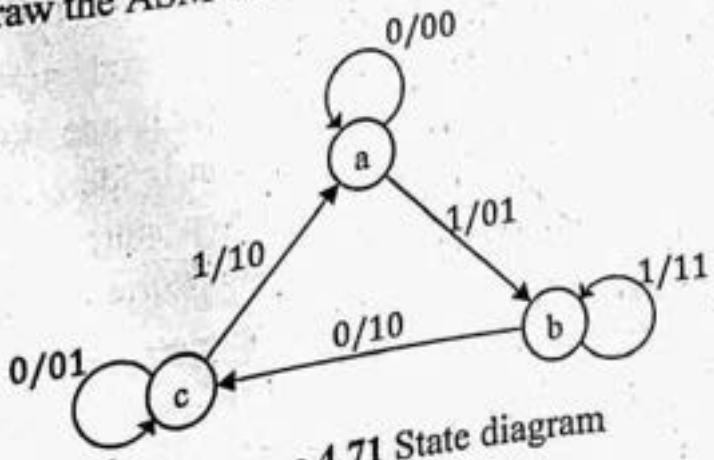


Figure 4.71 State diagram

Solution: Let Z_1, Z_2 be the outputs and x be the input

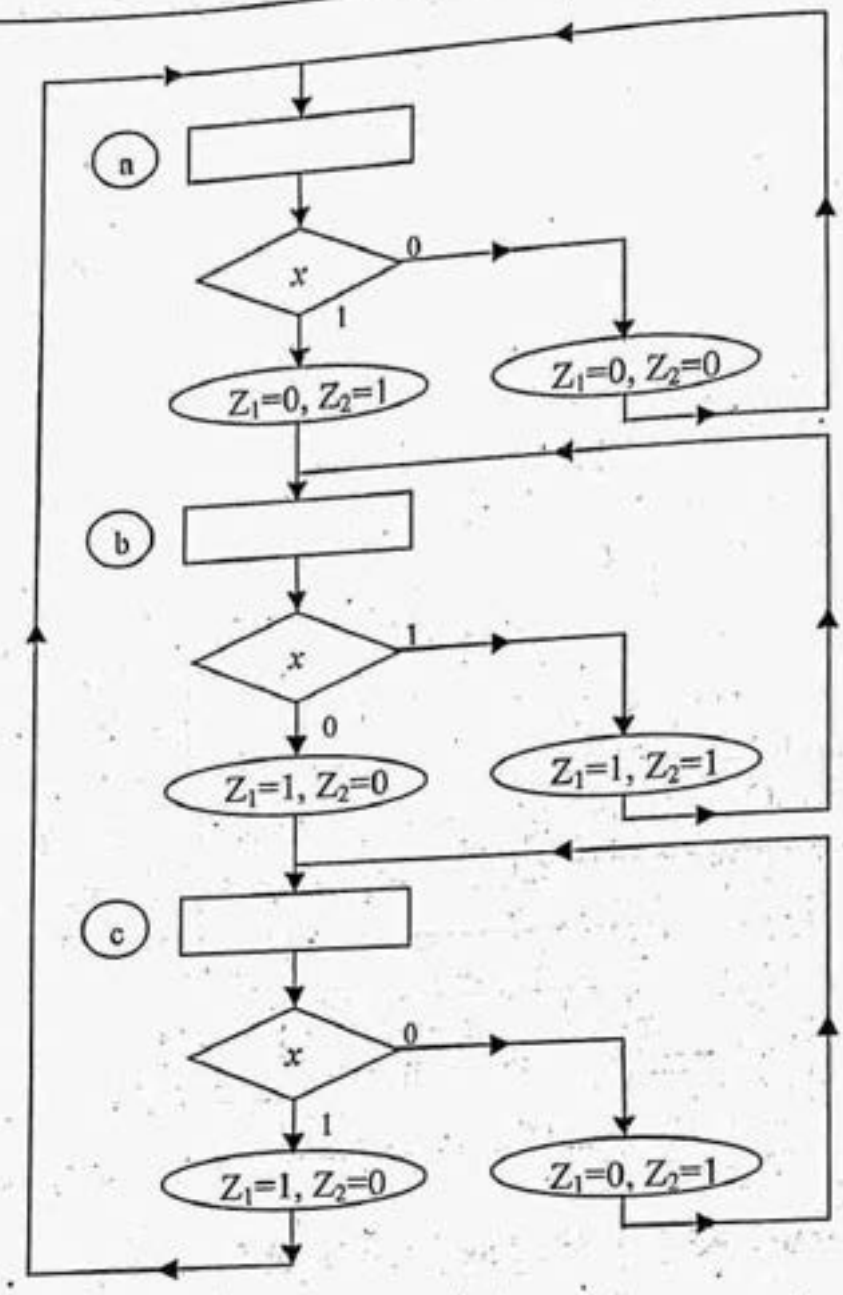


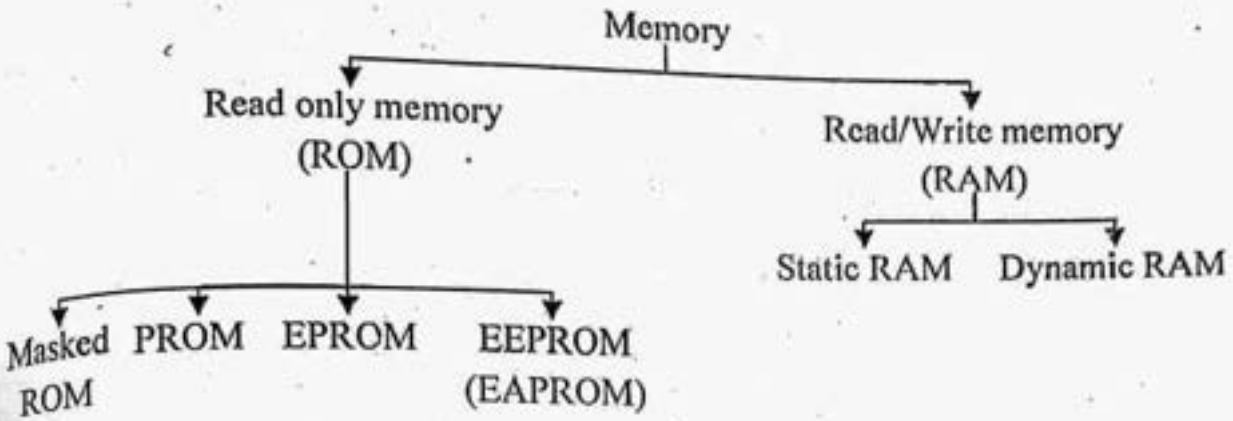
Figure 4.72 ASM chart

4.10 MEMORY

Memories are made up of registers. Each register is one storage location and each location can store one or more bits. Each location is identified by an address. Generally, the total number of bits that a memory can store is its capacity. Each register in memories consists of storage elements which stores one bit of data.

4.11 CLASSIFICATION OF MEMORIES

Memories are classified as follows.



4.11.1 Read only memory (ROM)

ROM is a non-volatile memory (it can hold data even if power is turned off). We can't write data in this memory. Initially the binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the memory unit even when power is turned off and on again. The binary information which is already stored inside the memory can be retrieved or read at any time.

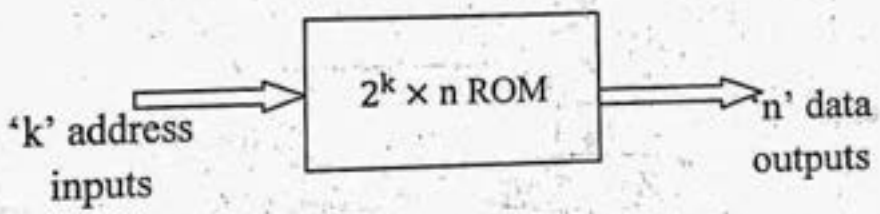


Figure 4.73 Block diagram of ROM memory

The block diagram of a ROM is shown in figure 4.73. It consists of 'k' inputs and 'n' outputs.

4.11.1.1 ROM Organization

The input provides the address for the memory and the output gives the data bits of the stored word which is selected by the address. The number of words in a ROM is determined from the fact that 'k' address input lines are needed to specify 2^k words. Consider a 32×8 ROM. The unit consists of 32 words of 8 bits each. There are 5 input lines that form the binary numbers form 00000 through 11111 for the address.

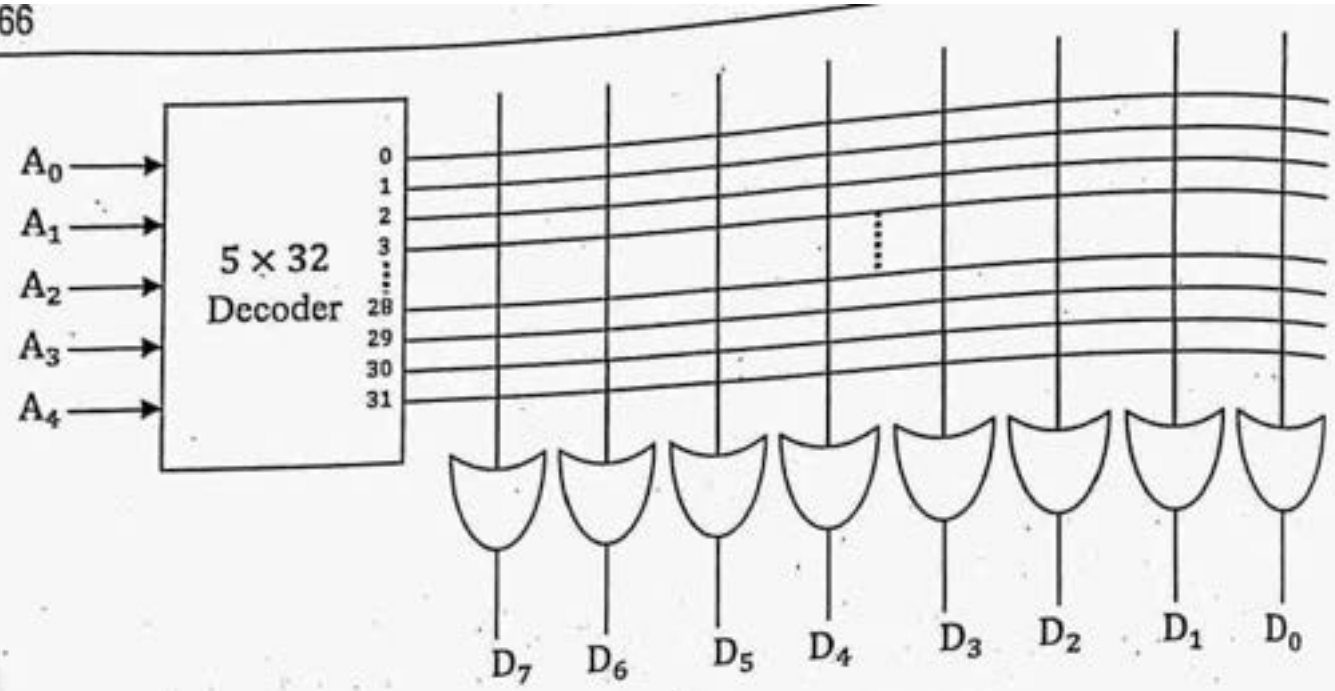


Figure 4.74 ROM memory unit

Figure 4.74 shows the internal logic construction of the ROM. The five inputs are decoded into 32 distinct outputs. Each output of the decoder represents a memory address. The 32 outputs are connected to each of the eight OR gates. Each OR gate must be considered as having 32 inputs. In general $2^k \times n$ ROM will have an internal $k \times 2^k$ decoder and 'n' OR gates. The 256 intersections in figure 4.75 are programmable. A programmable connection between the lines are logically equivalent to a switch that can be altered to either be close (horizontal and vertical lines are connected) or open (horizontal and vertical lines are disconnected).

Address inputs					Data outputs							
A_4	A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1	0	1	1	0
0	0	0	0	1	0	1	1	0	1	0	1	0
0	0	0	1	0	1	1	0	1	0	1	1	1
0	0	0	1	1	1	1	1	0	1	1	1	0
⋮					⋮							
1	1	1	0	0	0	1	0	1	1	0	1	1
1	1	1	0	1	0	0	0	1	1	0	1	1
1	1	1	1	0	1	1	1	0	0	1	0	1
1	1	1	1	1	1	1	1	1	1	1	0	1

Table 4.46 Truth table

The programmable intersection between two lines is some times called a cross point. Normally a fuse is used which connects the two lines. It can be opened or "blown" by applying a high voltage pulse into the fuse. The fuse links are "blown" according to the truth table.

Consider the truth table shown in table 4.46

Every '0' listed in the truth table specifies no connection and every '1' listed specifies a path that is obtained by a connection. The 0's in the word are programmed by blowing the fuse links.

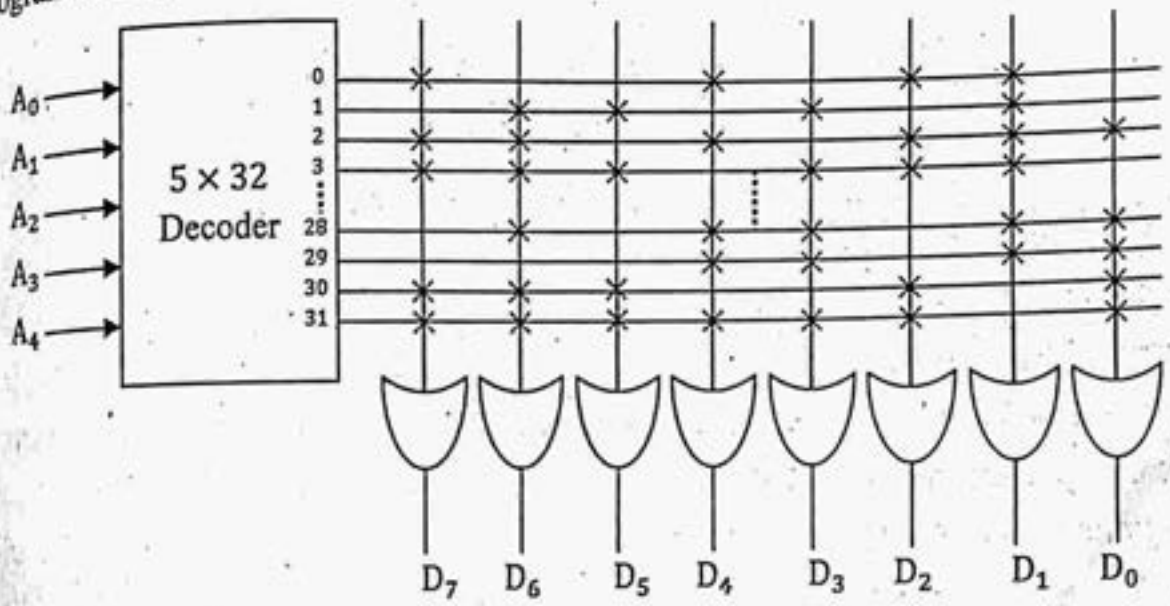


Figure 4.75 ROM

If an address 00011 is applied to the address line, the data output will be '11101110'.

4.11.2 Types of ROM

4.11.2.1 Masked ROM

In integrated circuits, a thin metalized layer connects the gates of some transistors to the row select lines. The gate connections of MOS transistors depend on the data to be stored in the ROM. Therefore, according to the user truth table, manufacturer can deposit thin layer of metal to connect gates of the transistors. Once the pattern/mask is decided, it is possible to make thousands of such ROMs, such ROMs are called mask programmed ROMs.

4.11.2.2 PROM (Programmable Read Only Memory)

When ordered, a PROM unit contains all the fuses intact giving all 1's in the bits of the stored words. The fuses in the PROM are blown by application of a high voltage through a special pin. A blown fuse defines a binary '0' and an intact fuse gives a binary '1'. This allows the user to program the PROM in the laboratory to achieve the desired relationship between input address and stored words.

4.11.2.3 EPROM (Erasable Programmable Read Only Memory)

The EPROM can be restructured to the initial state even though it has been programmed previously. When the EPROM is placed under a special ultraviolet light for a given period of time, the short wave radiation through its quartz window discharges the internal floating gates that serve as the programmed connections. After erasure, the EPROM return to its initial state and can be reprogrammed to a new set of values. It is not possible to erase selective information, when erased the entire information is lost.

4.11.2.4 EEPROM (Electrically Erasable PROM) /EAPROM (Electrically Alterable PROM)

EEPROM is similar to EPROM except that the previously programmed connections can be erased with an electrical signal instead of ultraviolet light. The advantage is that the device can be erased without removing it from its socket.

4.11.3 Random Access memory (RAM)

RAM is a volatile memory (it cannot hold data when power is turned off). The communication between a memory and its environment is achieved through data line, address selection lines and control lines. The control lines specify the direction of data transfer. The data lines are bidirectional which means that data can go in either direction. The direction of data transfer will be decided by the control inputs. The address lines specify the particular word chosen among the many variable. The two control inputs (READ or WRITE) specify the direction of transfer desired.

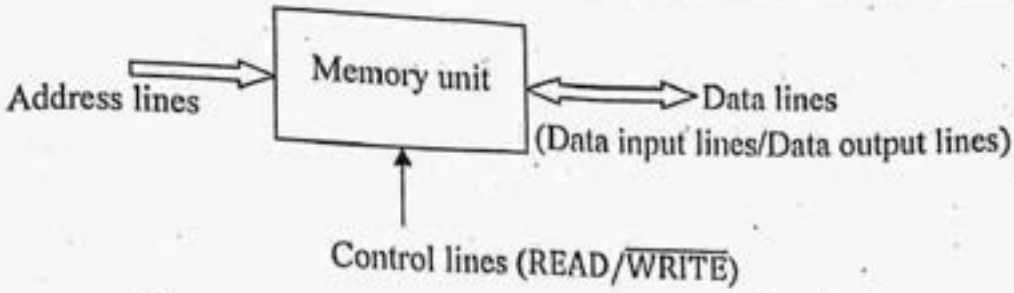


Figure 4.76 Block diagram of a Memory Unit

4.11.3.1 RAM Organization

Consider a memory unit contains 'n' data lines and 'k' address lines. A 'k' address lines can select 2^k locations in the memory.The process of storing the information to the memory is referred to as memory write operation. The process of transferring the stored information out of memory is referred to as memory read operation. RAM can perform both the write and read operation.

The number of word in a RAM is determined from the fact that 'k' address input lines are needed to specify 2^k words. Consider a 1024×16 RAM shown in figure 4.77. The memory unit consists of 1024 words of 16 bits each. In general $2^k \times n$ RAM will have $k \times 2^k$ decoder. Here 1024×16 RAM consists of 10 to 1024 decoder ($2^{10} = 1024$), where the decoder inputs are the 10 address lines $(A_9, A_8 \ldots \ldots A_0)$. A decoder accepts the address lines $(A_9, A_8 \ldots \ldots A_0)$ and provides the paths needed to select the word specified.

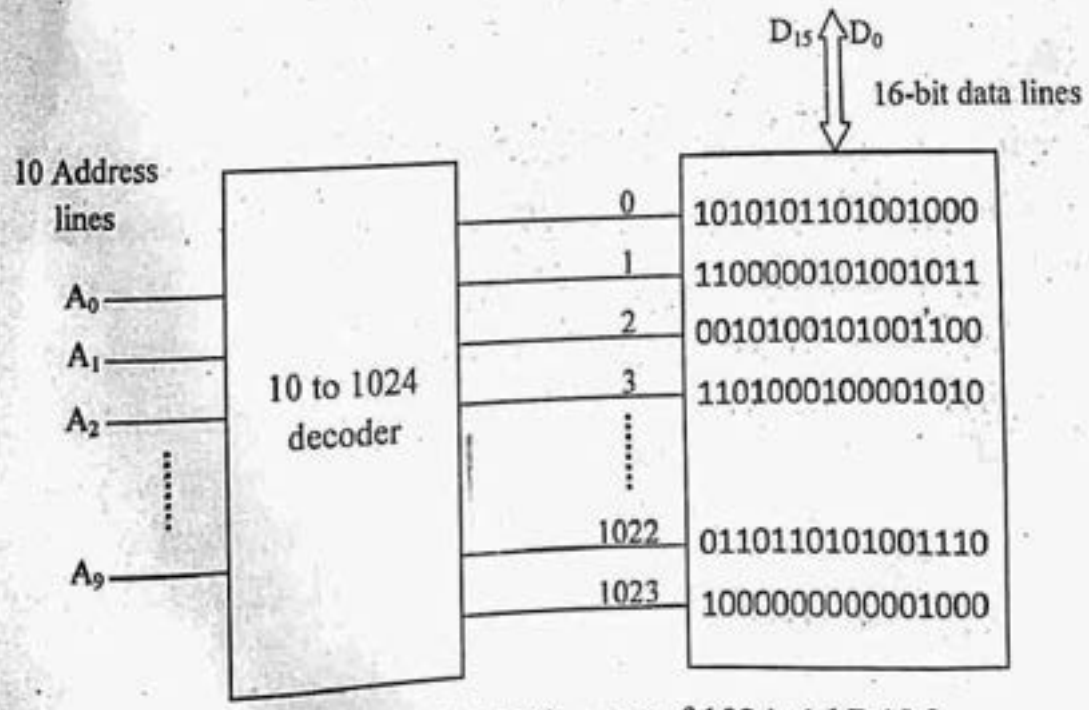


Figure 4.77 Block diagram of 1024×16 RAM

4.11.3.2 Write and read operation

The two operations that a RAM can perform are write and read operations.

The steps to perform memory write operation are as follows.

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data lines.
3. Activate the write input (READ/$\overline{\text{WRITE}}$=0)

The steps to perform memory read operation are as follows.

1. Apply the binary address of the desired word to the address lines.
2. Activate the read input (READ/$\overline{\text{WRITE}}$=1)

   The memory unit will then take the bits from the word that has been selected by address and apply them to the output data lines.

4.12 TYPES OF RAM

The random access memory is classified into two types based on its operating modes.

1. Static RAM
2. Dynamic RAM

4.12.1 Static RAM

Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories. These are random access memories and hence known as static RAM memories.

The static RAM (SRAM) consists essentially of internal latches or flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The memory cell can be implemented using MOS technology and Bipolar technology. The MOS technology uses Enhancement mode MOSFET transistors and the bipolar technology uses TTL (Transistor-Transistor-Logic) multiple emitter technology.

4.12.1.1 Bipolar RAM cell

The Bipolar RAM cell is implemented using TTL (Transistor- Transistor-Logic) multiple emitter technology. It stores 1 bit of information. It can store either 0 or 1 as long as power is applied. It can set to store 1 or it can reset to store 0. It is nothing but a flip-flop. Figure 4.78 shows a simplified schematic of a bipolar RAM cell.

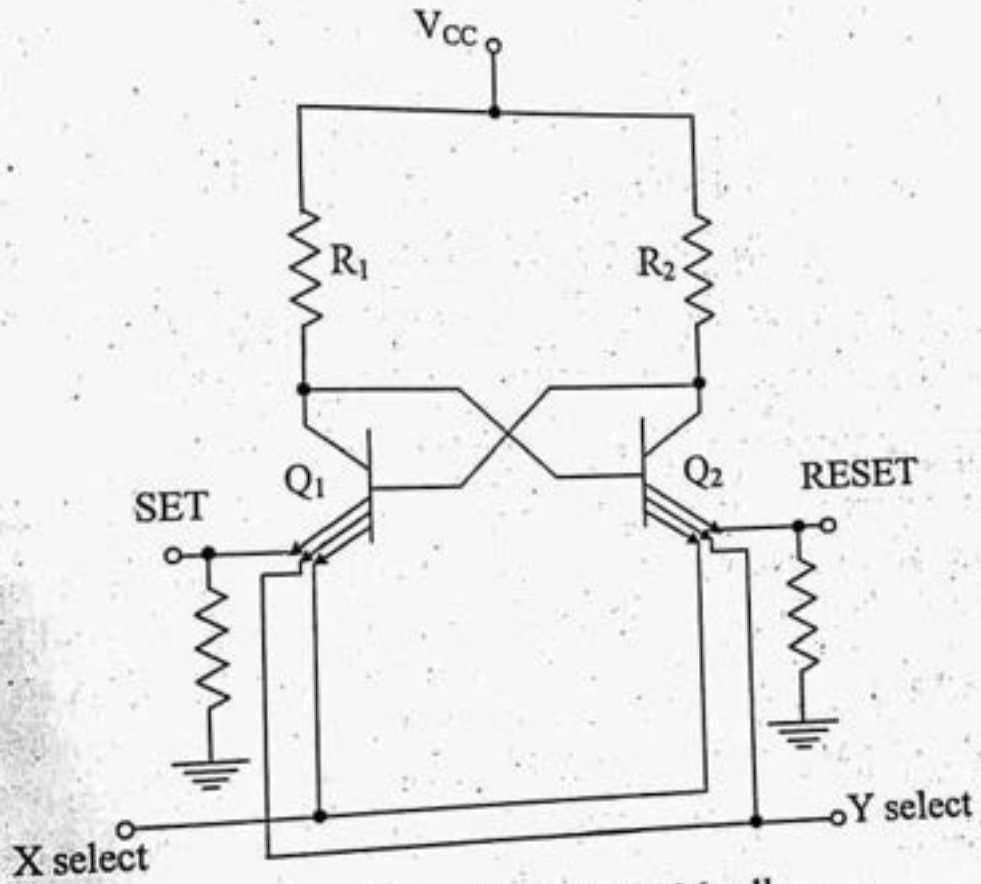


Figure 4.78 TTL RAM cell

Operation :

The X select line and Y select lines select a cell. The Q_1 and Q_2 are cross coupled inverters, hence one is always OFF while the other is ON. A "1" is stored in the cell if Q_1 is conducting and Q_2 is OFF. A "0" is stored in the cell if Q_2 is conducting and Q_1 is OFF. The state of the cell is changed to a "0" by pulsing a HIGH on the Q_1 emitter (SET). This turns OFF Q_1 and turns ON Q_2. A "1" can be rewritten by pulsing a HIGH on the Q_2 emitter (RESET).

4.12.1.2 MOSFET RAM cell

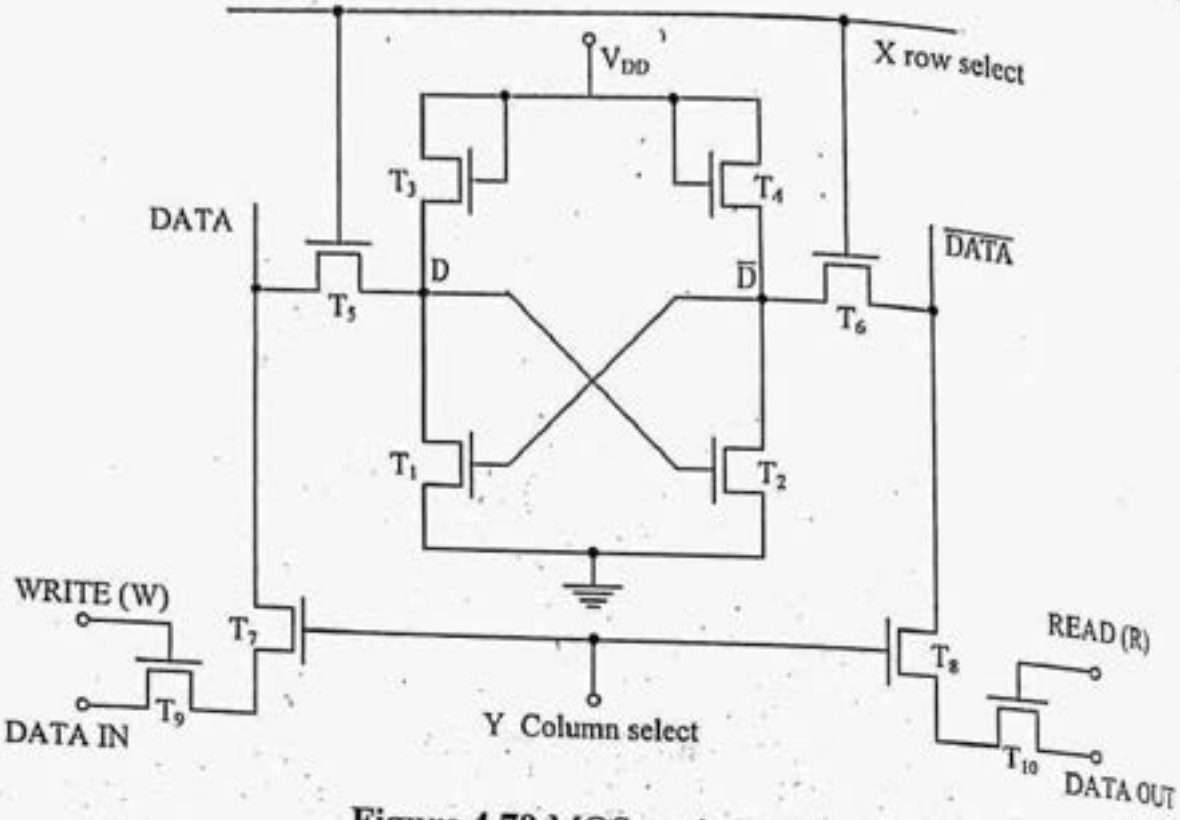


Figure 4.79 MOS static RAM cell

Enhancement mode MOSFET transistors are used to make MOSFET RAM cell. Here T_1 and T_2 form the basic cross coupled inverters, T_3 and T_4 act as load resistors for T_1 and T_2. X row select line and Y column select line are used for addressing the cell. When X row select line and Y column select line are HIGH, cell is selected. When X row select line is HIGH, T_5 and T_6 get ON and the cell is connected to the DATA and $\overline{DATA}$ line. When Y column select line is HIGH, T_7 and T_8 get ON. Due to this, either read or write operation is possible.

Write operation:

Write operation can be enabled by making WRITE(W) signal HIGH. With write operation enabled, if DATA IN signal is logic 1, node D is also at logic 1. This turns ON T_2 and T_1 is cutoff. If the DATA IN is logic '0', T_2 will be cutoff and T_1 will be turned ON.

Read operation:

Read operation can be enabled by making READ signal High. With read operation enabled, T_{10} becomes ON. This connects the data output ($\overline{DATA}$) line to the DATA OUT and thus the complement of the bit stored in the cell is available at the output.

4.12.2 Dynamic RAM

Dynamic RAM stores the data as a charge on the capacitor. The stored charge on the capacitors tends to discharge with time and the capacitors must be periodically recharged by using a refreshing circuit. This refreshing is done by cycling through the words for every few milliseconds to restore the decaying charge. Dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. Figure 4.80 shows a dynamic RAM cell.

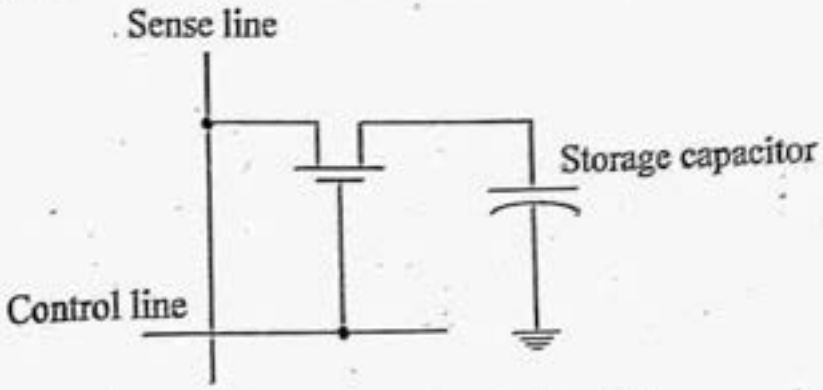


Figure 4.80 Dynamic RAM

Memory units that lose stored information when power is turned off are said to be volatile. Both static and dynamic RAM is of this category.

The differences between static RAM and dynamic RAM are shown in table 4.47.

SLNO.	Static RAM	Dynamic RAM
1.	Static RAM contains less memory cells per unit area.	Dynamic RAM contains more memory cells per unit area as compared to static RAM.
2.	Static RAM consists of flip-flops. Each flip-flop stores one bit of binary information.	Dynamic RAM stores the data as a charge on the capacitor. It consists of MOSFET and capacitor on each cell.
3.	Its access time is less and so it is having faster memories.	Its access time is greater that static RAM.
4.	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors after every few milliseconds. Extra hardware is required to control refreshing. This makes system design complicated.
5.	Cost of static RAM is more	Cost of Dynamic RAM is less

Table 4.47 Difference between SRAM and DRAM

4.13 MEMORY CYCLES AND TIMING WAVEFORMS

Access time: The access time is the time taken to read a stored word after applying the address bits.

Cycle time

Read cycle time: It is the minimum time for which an address must be held stable on the address bus for read cycle.

Write cycle time: It is the minimum time for which an address must be held stable on the address bus for write cycle.

The operation of the memory unit is controlled by an external device such as CPU(Central processing Unit). The CPU is synchronized by its own clock but the memory does not employ an internal clock. The CPU must provide the memory control signal in such a way as to synchronize its internal operations with the read and write operation of memory.

Assume that a CPU operates with a clock frequency of 50MHz and the corresponding period for one clock cycle is 20ns. Let the access time and cycle time does not exceed 50ns interval (i.e. read or write operation can be performed within 3 clock cycles).

Write cycle

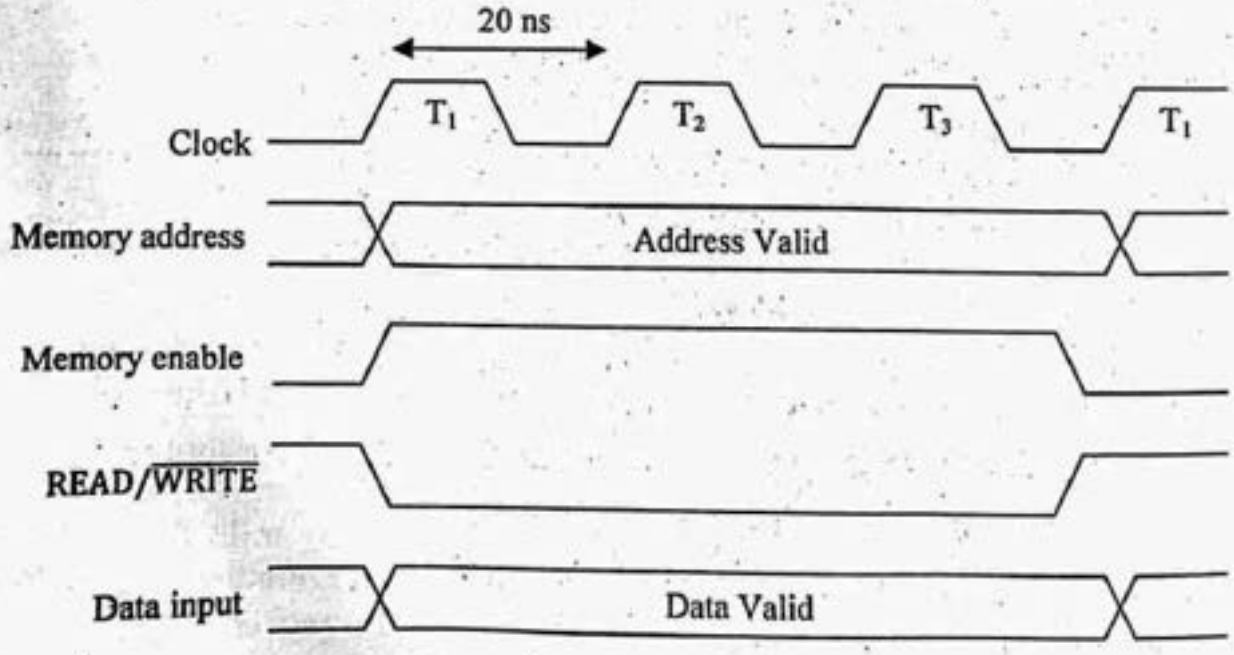


Figure 4.81 Memory write cycle

Figure 4.81 shows the write cycle timing diagram for a CPU with 50MHz clock. Let the maximum write cycle time be 50ns. The write operation requires three 20ns cycles T_1, T_2 and T_3 for a write operation. The CPU must provide the address and data input to the memory. This is done at the beginning of T_1. The memory enable signal changes to the high level and the READ/$\overline{\text{WRITE}}$ signal must go low to perform write operation. These two signals must be active for a period of 50ns. The memory address and data signals must remain active for a short time after the control signals are deactivated. At the completion of the 3rd clock cycle, the memory write operation is completed and the CPU can access the data from the corresponding memory location.

Read cycle

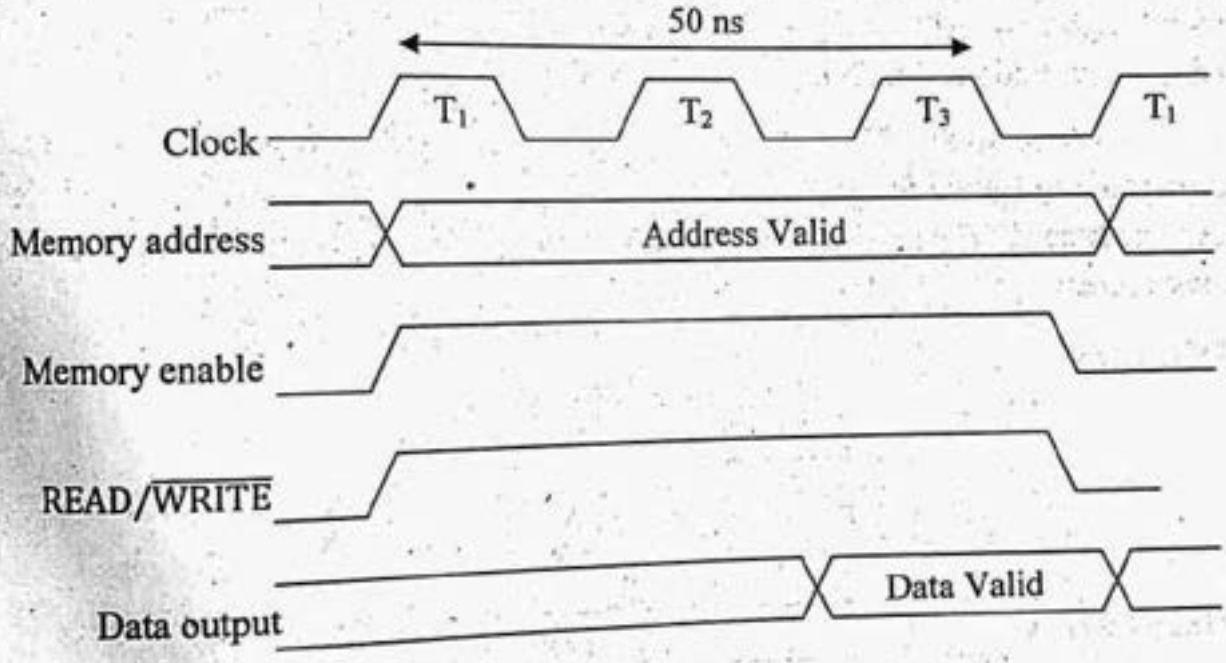


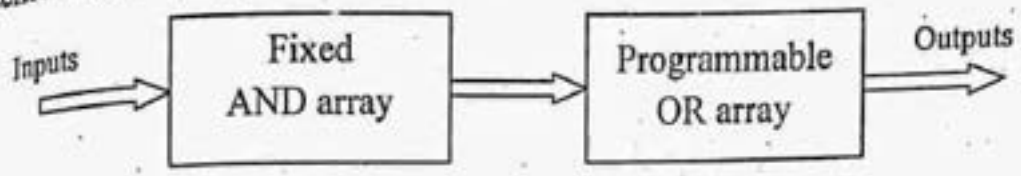
Figure 4.82 Memory read cycle

Figure 4.82 shows the read cycle. While performing read operation, the required address for the memory is provided by the CPU. The memory enable and READ/$\overline{\text{WRITE}}$ must be in logic high for read operation. The memory places the data of the word selected by the address lines into the output data lines within a 50ns interval from the time that the memory enable is activated.

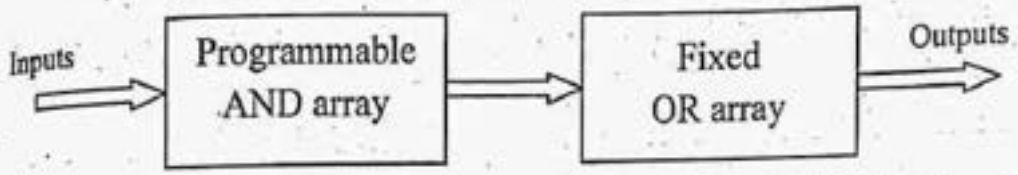
4.16 PROGRAMMABLE LOGIC DEVICES (PLDS)

A programmable logic device (PLD) is an integrated circuit with programmable gates that contain AND array and OR array.

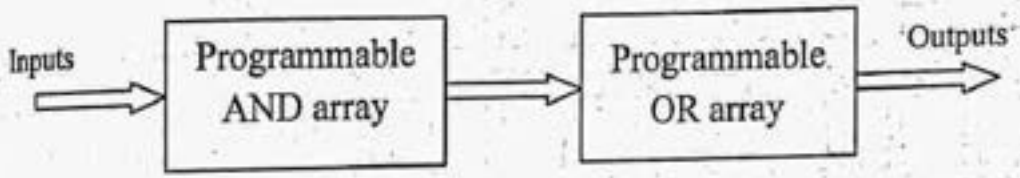
There are three major types of combinational PLDs and they differ in the placement of the programmable connections in the AND-OR array.



a. Programmable Read-only memory (PROM)



b. Programmable array logic (PAL)



c. Programmable Logic array (PLA)

Figure 4.89 Basic configuration of PROM, PAL and PLA

The programmable read only memory (PROM) has a fixed AND array and programmable OR array. The programmable array logic (PAL) has a programmable AND array and a fixed OR array. The most flexible PLD is the programmable logic array(PLA), where both the AND and OR arrays can be programmed.

4.16.1 Programmable Logic Array (PLA)

In PLA, both AND and OR arrays can be programmed. So both AND and OR gates have fuses. The product terms constitute a group of 'n' AND gates and the sum terms constitute a group of 'm' OR gates. Fuses are inserted between all 'n' inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates. The third set of fuses between the output of OR gates and the input of inverters that allows the output function to be generated either in the AND-OR form or in the AND-OR INVERT form. When inverter is bypassed by link, we get AND-OR implementation. When the inverter is not bypassed, we get AND-OR INVERT implementation.

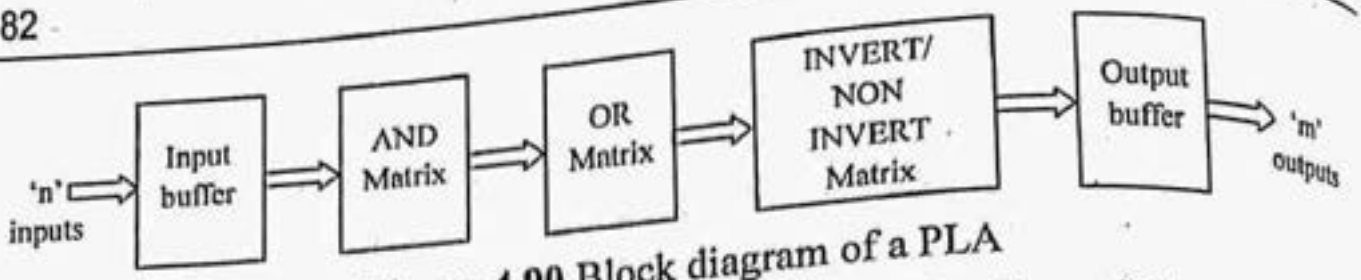


Figure 4.90 Block diagram of a PLA

Input buffers are provided in the PLA to limit loading of the sources that drive the inputs. They provide two outputs, one output in the inverted form of its input and the other output in the non inverted form of its input.

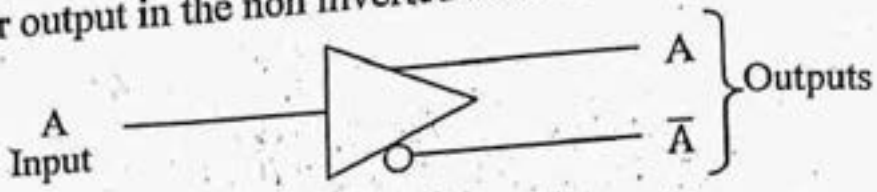


Figure 4.91 Buffer

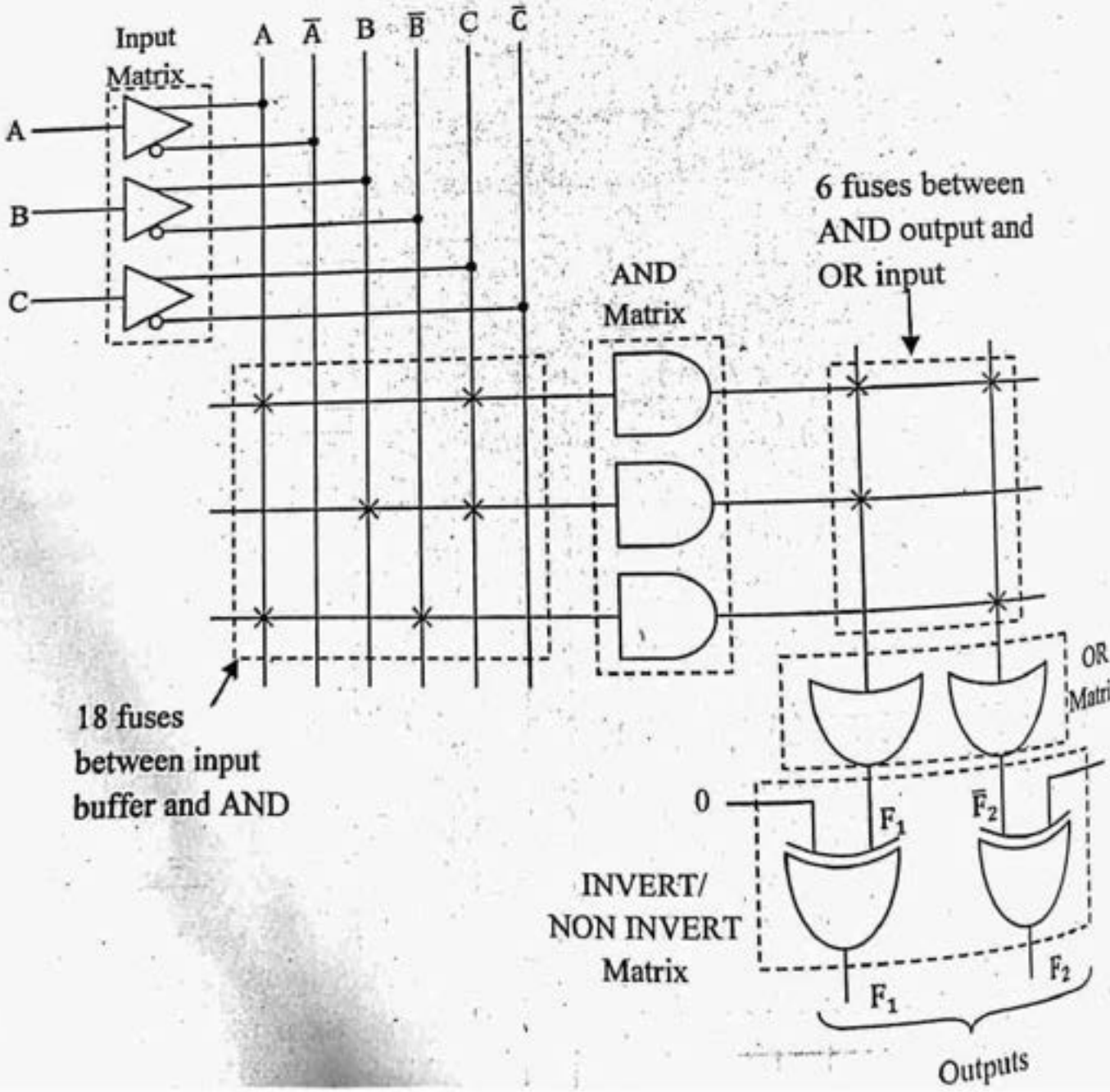


Figure 4.92 shows a PLA with 3-Inputs and 2 outputs. Here the output of OR gates are F_1 and $\overline{F}_2$ therefore the output of F_1 is not to be inverted. So the other input of XOR is connected to logic '0'. But the output of second OR gate $\overline{F}_2$ is to be inverted. So the other terminal of XOR is connected to logic '1' to obtain the real output F_2.

4.16.2 Programmable Array Logic (PAL)

The programmable array logic (PAL) is a programmable logic device with a fixed OR array and a programmable AND array. Figure 4.93 shows the logic configuration of a typical PAL. The PAL shown in figure 4.93 has four inputs and four outputs. Each input has a buffer- inverter gate and each output is generated by a fixed OR gate.

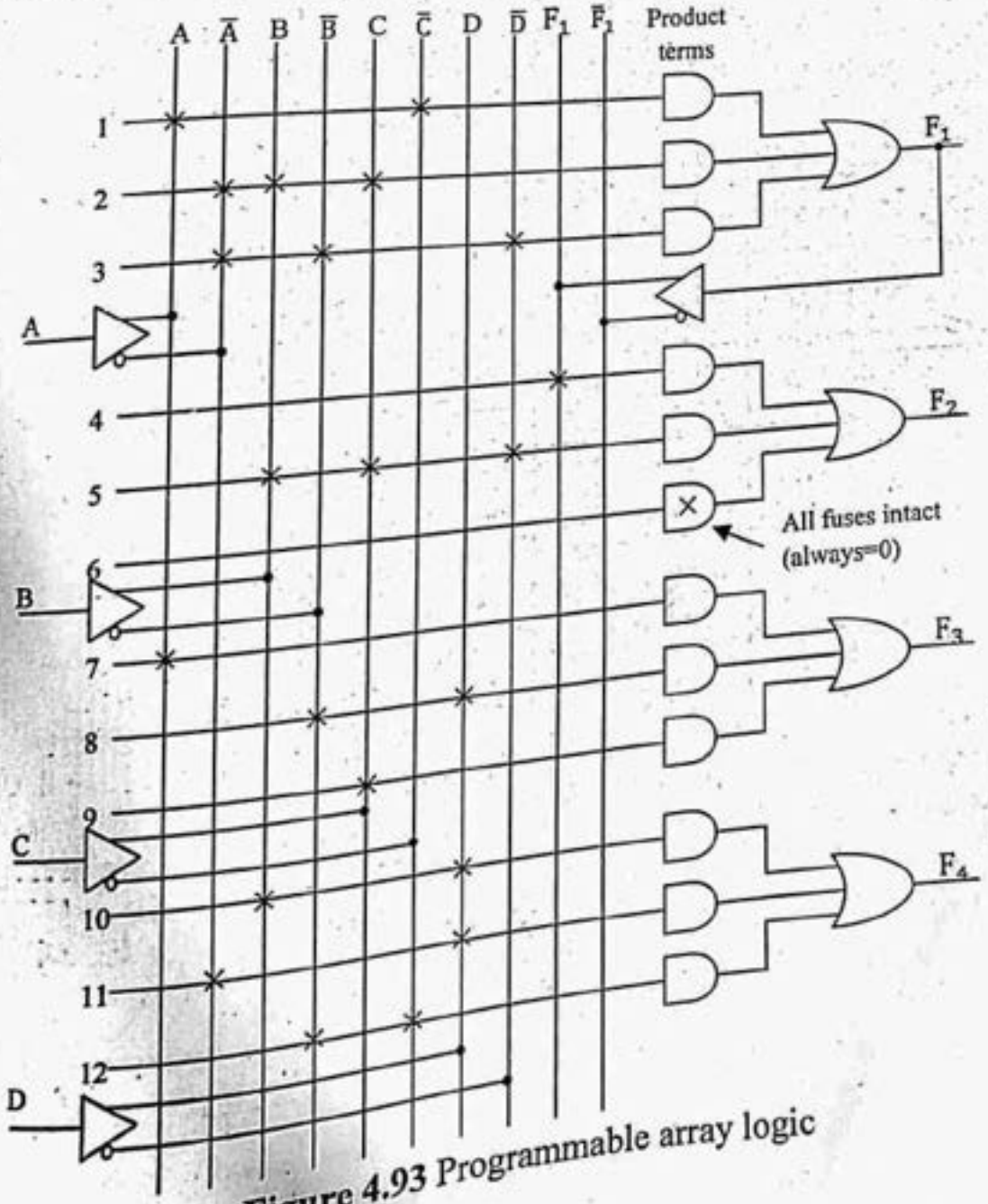


Figure 4.93 Programmable array logic

There are four sections in the unit each being composed of three wide AND-OR array. This is the term used to indicate that there are 3 programmable AND gates in each section and one fixed OR gate. Each AND gate has 10 programmable input connections. This is shown in the figure 4.93 by 10 vertical lines. The horizontal line symbolizes the multiple input configuration of the AND gate. The function F_1 is used by the function F_2, so the output F_1 is connected to a buffer-inverter gate and then fed back to the input of the AND gate present in F_2.

4.16.3 Programmable Read Only Memory (PROM)

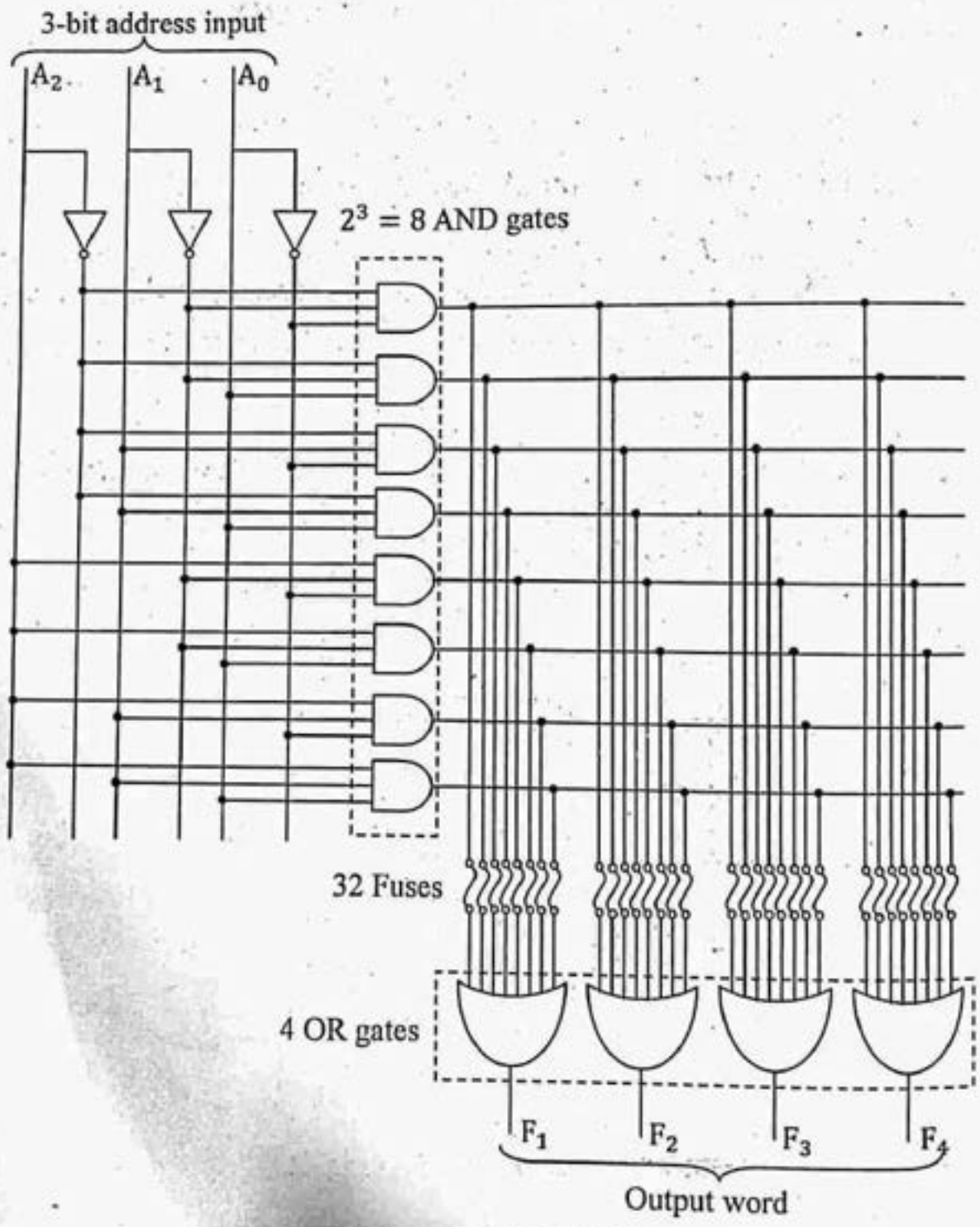


Figure 4.94 PROM

A PROM consists of 'n' input lines and 'm' output lines. Each bit combination of the input variables is called an address. Each bit combination that comes out of the output lines is called a word. The number of bits per word is equal to the number of output line 'm'.

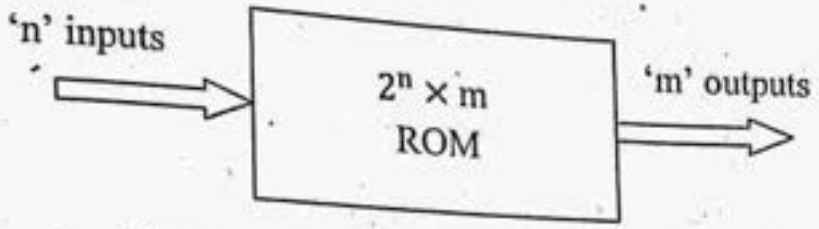


Figure 4.95 Block diagram of PROM

A 8×4 PROM consists of 3 inputs, 4 outputs. This PROM contains 8 AND gates, 4 OR gates and 32 fuses which is shown in figure 4.94.

4.16.4 Implementation of combinational circuits using PLA

Example 4.18: ***A combinational circuit is defined by the functions.*** $F_1 = \sum_m(3,5,7)$; $F_2 = \sum_m(4,5,7)$. ***Implement the circuit with a PLA having 3 inputs, 3 product terms and 2 outputs.***

Solution:

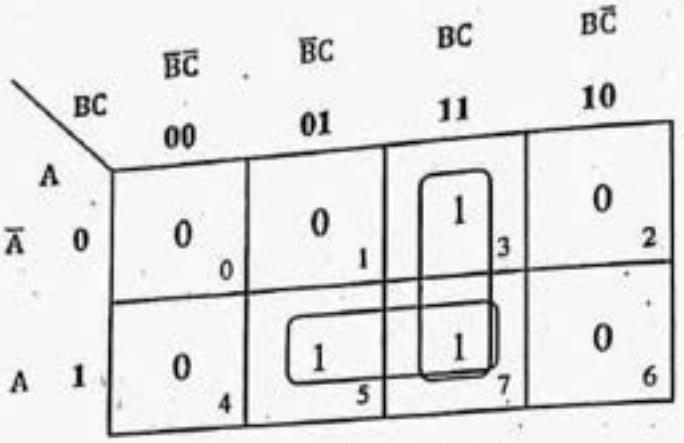
The truth table for the given Boolean functions is shown in table 4.48.

Let, $F_1(A,B,C) = \sum_m(3,5,7)$; $F_2(A,B,C) = \sum_m(4,5,7)$

Inputs			Outputs	
A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

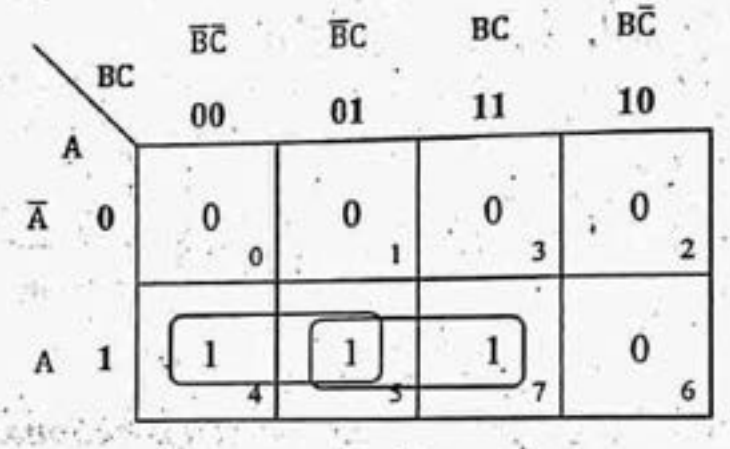
Table 4.48 Truth table

K-map for F_1



$$F_1 = AC + BC$$

K-map for F_2



$$F_2 = A\overline{B} + AC$$

Here $F_1 = AC + BC$ and $F_2 = A\overline{B} + AC$. The two functions F_1 and F_2 totally have 3 different product terms AC, BC and $A\overline{B}$.

Product terms	Inputs			Outputs	
	A	B	C	F_1	F_2
AC	1	-	1	1	1
BC	-	1	1	1	-
$A\overline{B}$	1	0	-	-	1
				T	T

Table 4.49 PLA program table

Under each output variable, we write a 'T' if the output inverter is to be bypassed and 'C' if the function is to be complemented with the output inverter.

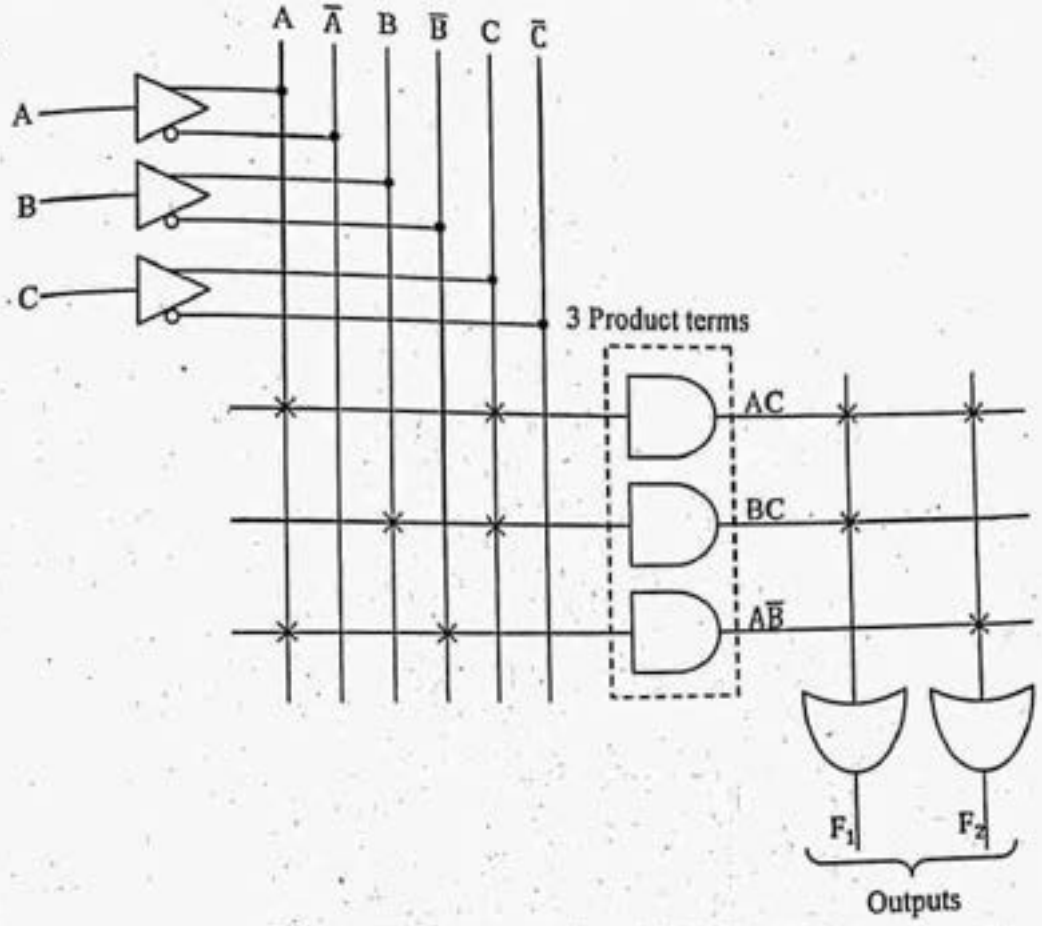


Figure 4.96 PLA

Example 4.19: Illustrate how a PLA can be used for combinational logic design with reference to the functions.

$$F_1(a,b,c) = \sum_m(0,1,3,4)$$

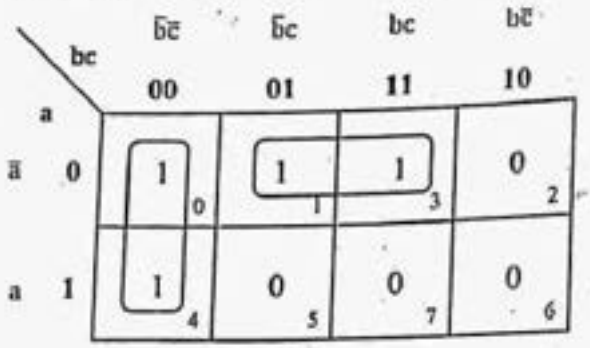
$$F_2(a,b,c) = \sum_m(1,2,3,4,5)$$

Solution: The truth table for the given function is shown in table 4.50

Inputs			Outputs	
a	b	c	F_1	F_2
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

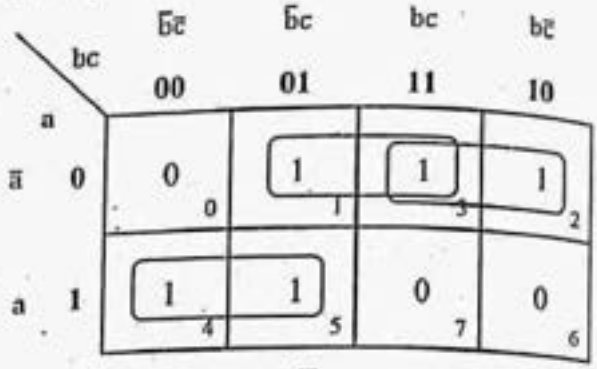
Table 4.50 Truth table

K-map for F_1



$$F_1 = \bar{b}\bar{c} + \bar{a}c$$

K-map for F_2



$$F_2 = a\bar{b} + \bar{a}c + \bar{a}b$$

The two functions F_1 and F_2 totally have 4 different product terms $\bar{b}\bar{c}$, $\bar{a}c$, $a\bar{b}$ and $\bar{a}b$.

Product terms	Inputs			Outputs	
	a	b	c	F_1	F_2
$\bar{b}\bar{c}$	-	0	0	1	-
$\bar{a}c$	0	-	1	1	1
$a\bar{b}$	1	0	-	-	1
$\bar{a}b$	0	1	-	-	1
				T	T

Table 4.51 Program table

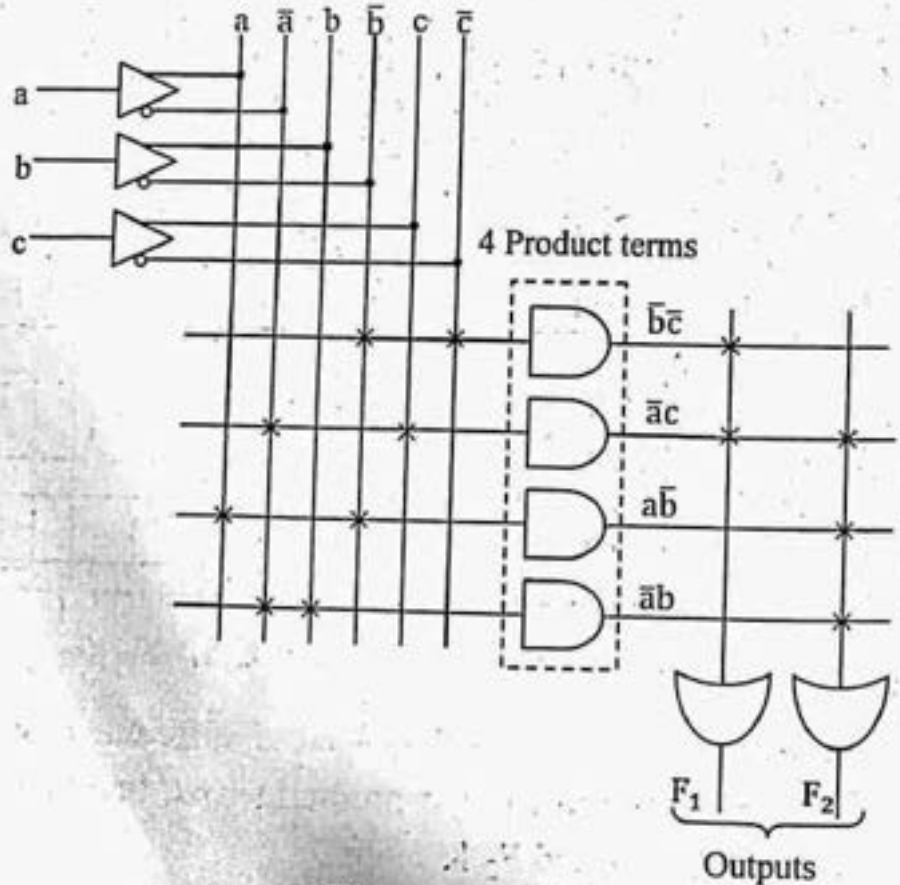


Figure 4.97 PLA

Example 4.20: Implement the following multi-Boolean function using 3 × 4 × 2 PLA.

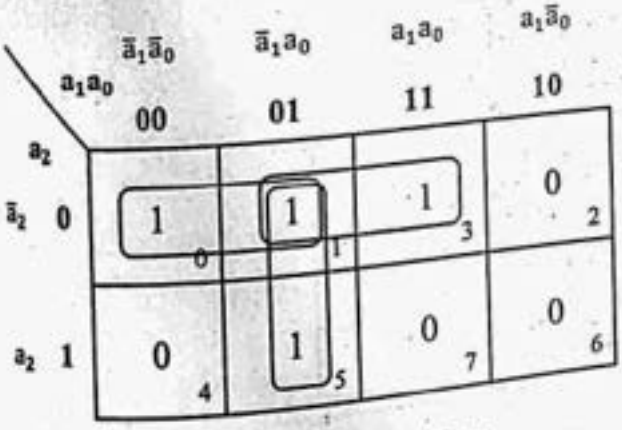
$$F_1(a_2, a_1, a_0) = \sum_m(0,1,3,5);$$
$$F_2(a_2, a_1, a_0) = \sum_m(3,5,7)$$

Solution: The Truth table for the given function is shown in table 4.52

Inputs			Outputs	
a_2	a_1	a_0	F_1	F_2
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1

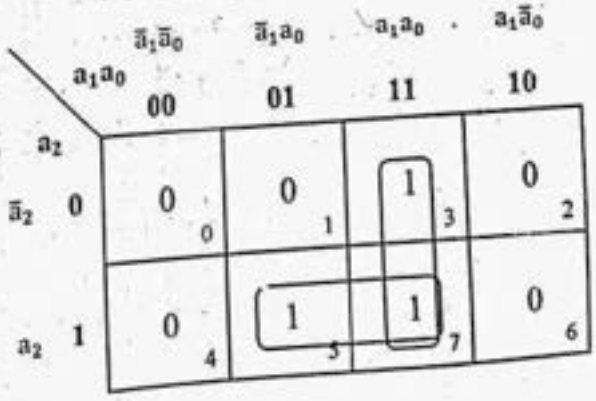
Table 4.52 Truth table

K-map for F_1



$$F_1 = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$$

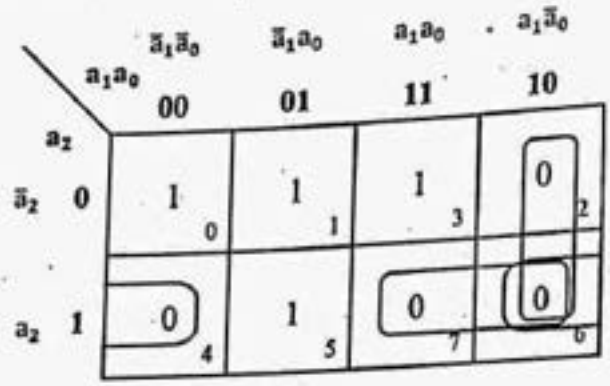
K-map for F_2



$$F_2 = a_2a_0 + a_1a_0$$

Here the two functions F_1 and F_2 totally have 5 different product terms. In order to implement the given function using 4 product terms, find $\bar{F}_1$ and $\bar{F}_2$ by just grouping the zeros present in the F_1 and F_2 K-maps.

K-map for $\bar{F}_1$

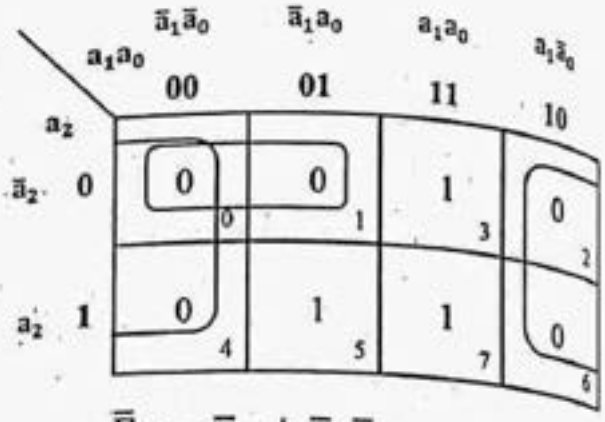


$$\bar{F}_1 = a_2\bar{a}_0 + a_2a_1 + a_1\bar{a}_0$$

$$F_1 = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$$

$$F_2 = a_2a_0 + a_1a_0 \quad .$$

K-map for $\bar{F}_2$



$$\bar{F}_2 = \bar{a}_0 + \bar{a}_2\bar{a}_1$$

$$\bar{F}_1 = a_2\bar{a}_0 + a_2a_1 + a_1\bar{a}_0$$

$$\bar{F}_2 = \bar{a}_0 + \bar{a}_2\bar{a}_1$$

The functions F_1 and F_2 totally have 5 different product terms. The functions $\bar{F}_1$ and F_2 totally have 5 different product terms. The functions $\bar{F}_1$ and $\bar{F}_2$ totally have 5 different product terms. But the functions F_1 and $\bar{F}_2$ totally have 4 different product terms $\bar{a}_1a_0$, $\bar{a}_2\bar{a}_1$, $\bar{a}_2a_0$ and $\bar{a}_0$. So consider the functions

$$F_1 = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$$

$$\bar{F}_2 = \bar{a}_0 + \bar{a}_2\bar{a}_1$$

The PLA program table for the functions F_1 and $\bar{F}_2$ is shown in table 4.53.

Product terms	Inputs			Outputs	
	a_2	a_1	a_0	F_1	F_2
$\bar{a}_1a_0$	-	0	1	1	-
$\bar{a}_2\bar{a}_1$	0	0	-	1	1
$\bar{a}_2a_0$	0	-	1	1	-
$\bar{a}_0$	-	-	0	-	1
				T	C

Table 4.53 PLA program table

Here C represents that the output of $\bar{F}_2$ is to be complemented in order to make it a real output function F_2. Under each output variable, we write a 'T' if the output inverter is to be bypassed and 'C' if the function is to be complemented with the output inverter.

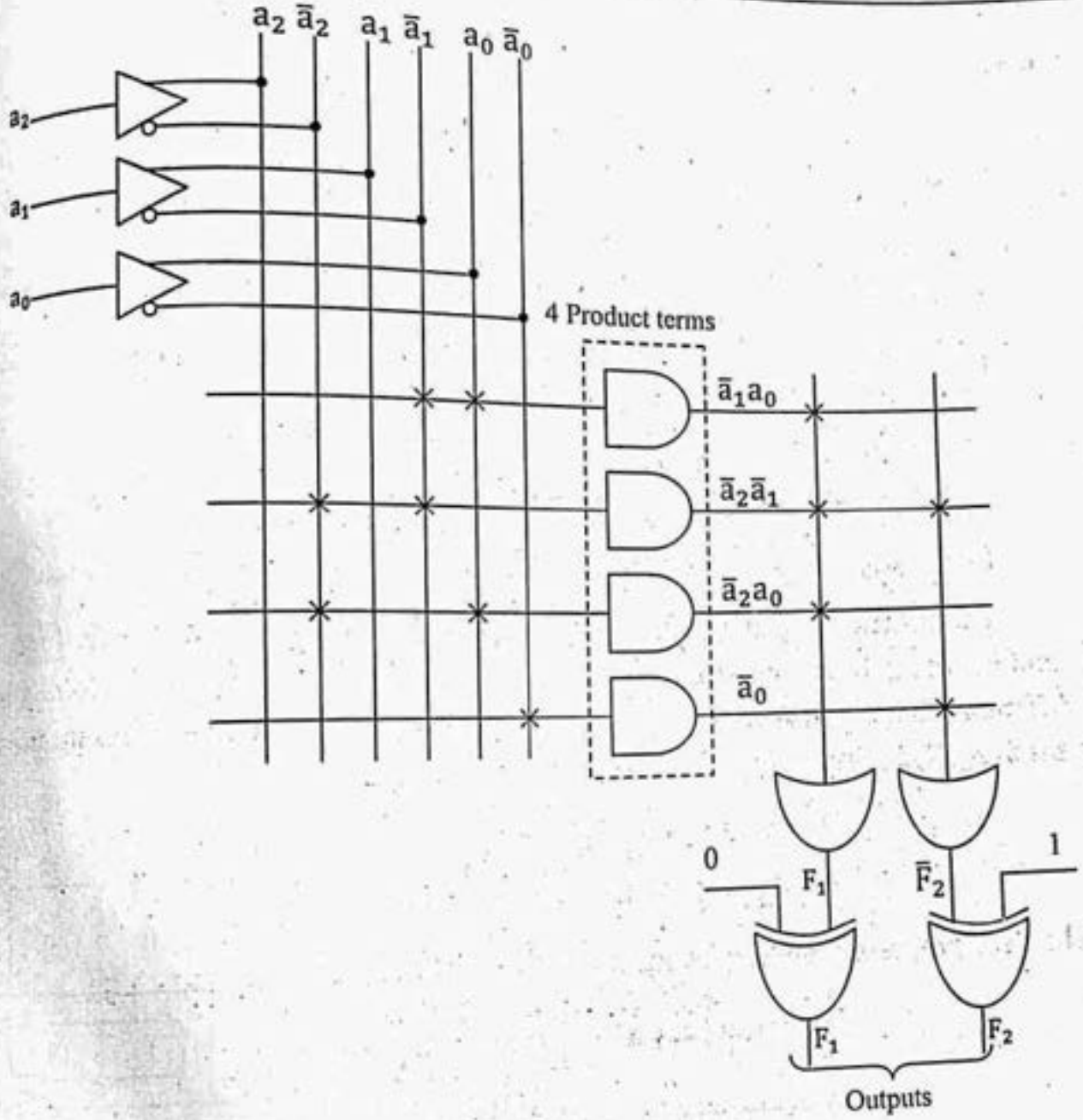


Figure 4.98 PLA

Connect an XOR gate to the complement term $\bar{F}_2$, in which other input of XOR is made high.

Example 4.21: ***Implement the following function using PLA.***

$$A(x,y,z) = \sum_m(1,2,4,6)$$

$$B(x,y,z) = \sum_m(0,1,6,7)$$

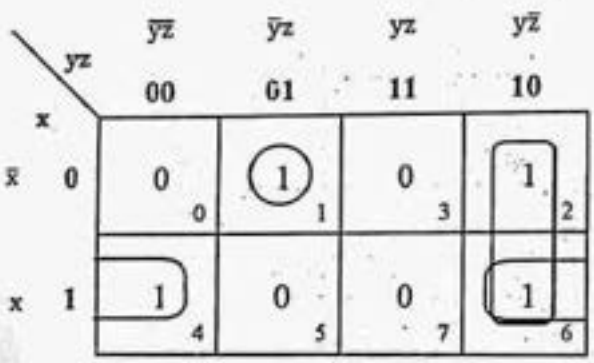
$$C(x,y,z) = \sum_m(2,6)$$

Solution: The truth table for the given Boolean function is shown in table 4.54

Inputs			Outputs		
x	y	z	A	B	C
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	0

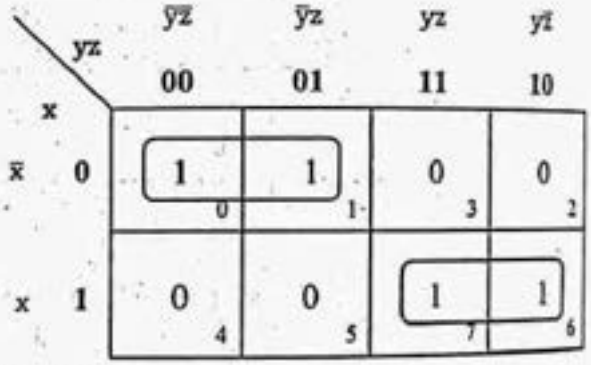
Table 4.54 Truth table

K-map for A



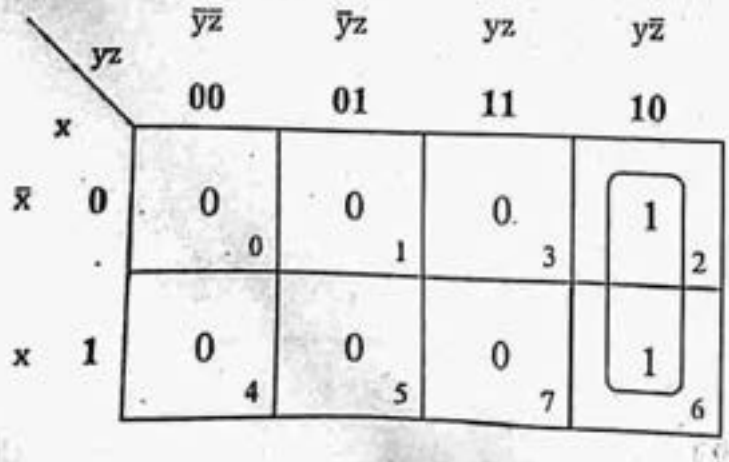
$A = \bar{x}\bar{y}z + x\bar{z} + \bar{y}\bar{z}$

K-map for B



$B = \bar{x}\bar{y} + xy$

K-map for C



$C = y\bar{z}$

Product terms	Inputs			Outputs		
	x	y	z	A	B	C
$\bar{x}\bar{y}z$	0	0	1	1	-	-
$x\bar{z}$	1	-	0	1	-	-
$y\bar{z}$	-	1	0	1	-	1
$\bar{x}\bar{y}$	0	0	-	-	1	-
xy	1	1	-	-	1	-
				T	T	T

Table 4.55 PLA program table

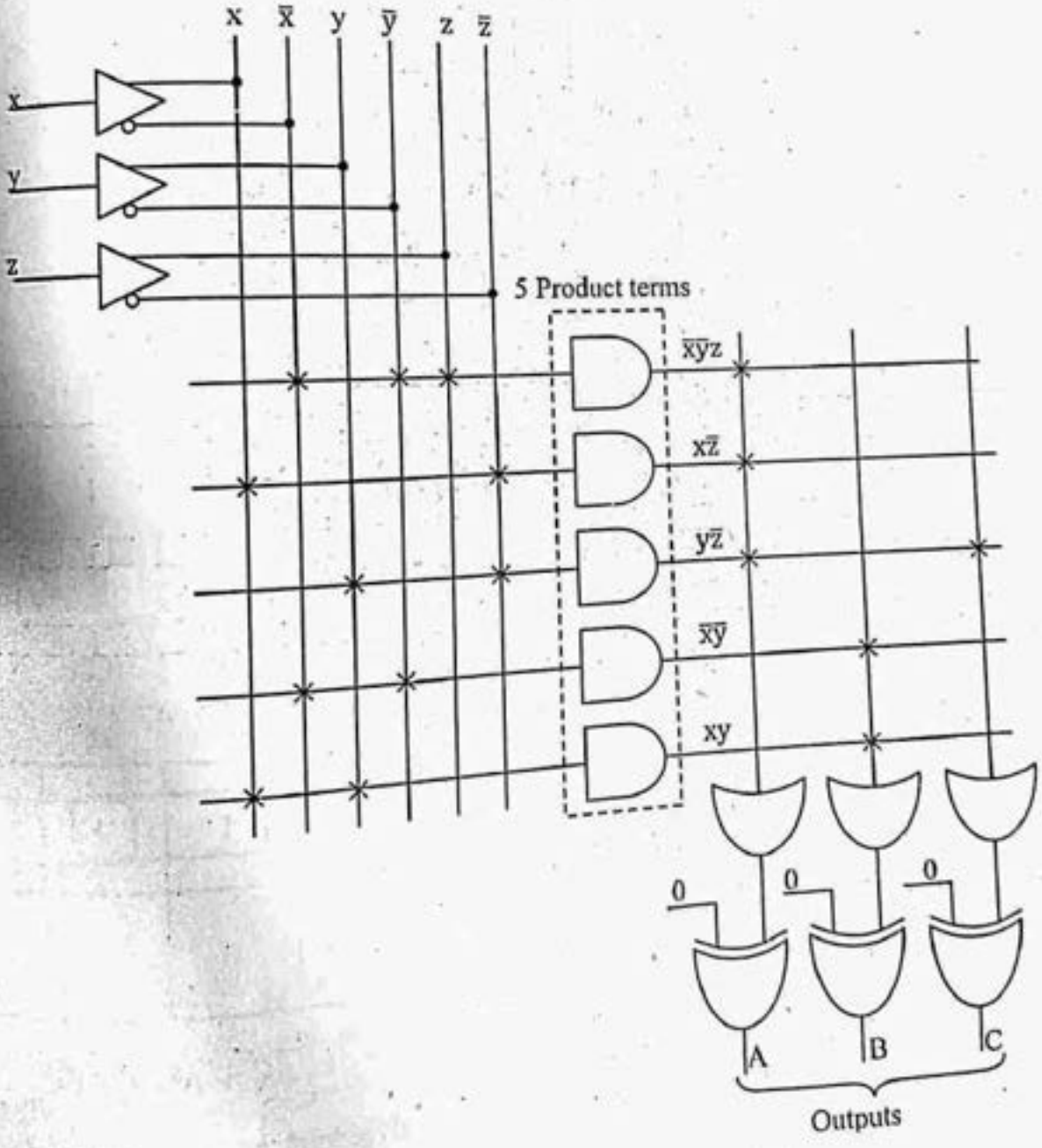


Figure 4.99 PLA

Example 4.22: Use PLA with 3 inputs, 4 AND terms and two outputs to implement the following two Boolean functions. $F_1(A,B,C) = \sum_m(3,5,6,7)$ *and* $F_2(A,B,C) = \sum_m(1,2,3,4)$

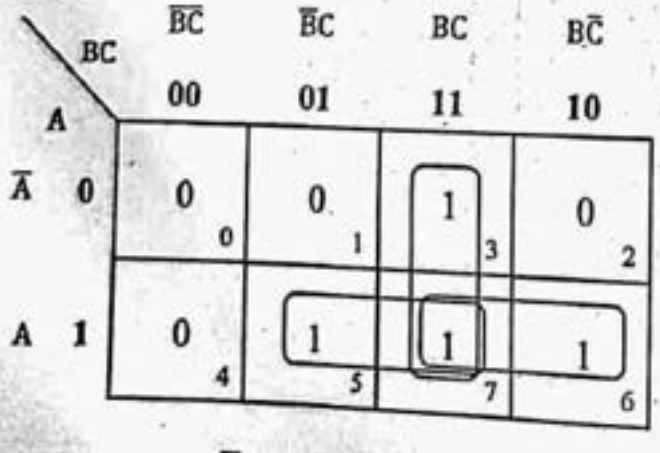
Solution:

The truth table for the given Boolean functions is shown in table 4.56.

Inputs			Outputs	
A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

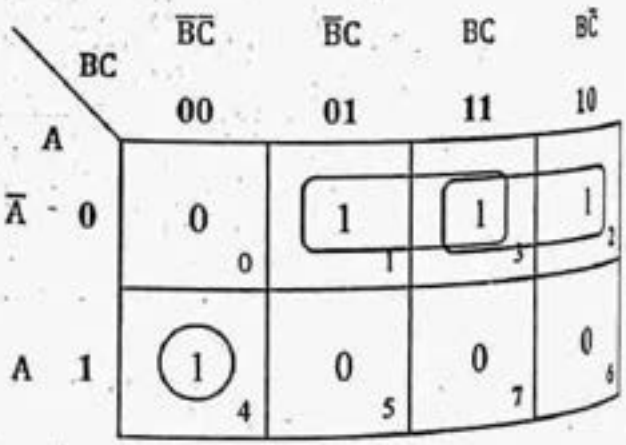
Table 4.56 Truth table

K-map for F_1



$F_1 = BC + AC + AB$

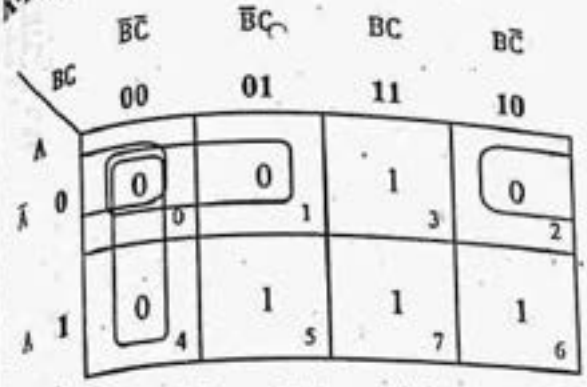
K-map for F_2



$F_2 = \overline{A}C + \overline{A}B + A\overline{B}\overline{C}$

The two functions F_1 and F_2 totally have 6 different product terms. In order to implement the given function using 4 product terms, find $\overline{F}_1$ and $\overline{F}_2$ by just grouping the zeros present in the F_1 and F_2 K-maps.

K-map for $\bar{F}_1$

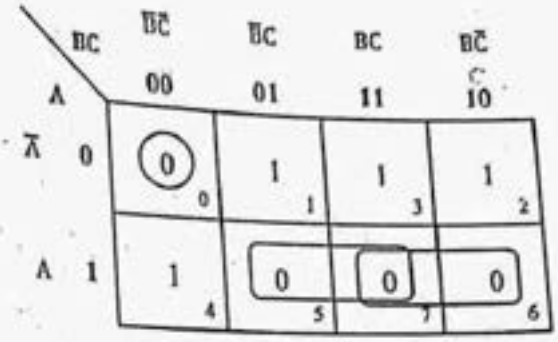


$$\bar{F}_1 = \overline{BC} + \overline{AB} + \overline{AC}$$

$$F_1 = BC + AC + AB$$

$$F_2 = \overline{A}C + \overline{A}B + A\overline{BC}$$

K-map for $\bar{F}_2$



$$\bar{F}_2 = AC + AB + \overline{ABC}$$

$$\bar{F}_1 = \overline{BC} + \overline{AB} + \overline{AC}$$

$$\bar{F}_2 = AC + AB + \overline{ABC}$$

The functions F_1 and F_2 totally have 6 different product terms. The functions $\bar{F}_1$ and F_2 totally have 6 different product terms. The functions $\bar{F}_1$ and $\bar{F}_2$ totally have 6 different product terms. But the functions F_1 and $\bar{F}_2$ totally have 4 different product terms BC, AC, AB and $\overline{ABC}$. So consider the functions $F_1 = BC + AC + AB$ and $\bar{F}_2 = AC + AB + \overline{ABC}$. The PLA program table for the functions F_1 and $\bar{F}_2$ is shown in table 4.57.

Product terms	Inputs			Outputs	
	A	B	C	F_1	F_2
BC	-	1	1	1	-
AC	1	-	1	1	1
AB	1	1	-	1	1
$\overline{ABC}$	0	0	0	-	1
				T	C

Table 4.57 PLA program table

Here C represents that the output of $\bar{F}_2$ is to be complemented in order to make it a real output function F_2. Under each output variable, we write a 'T' if the output inverter is to be bypassed and 'C' if the function is to be complemented with the output inverter.

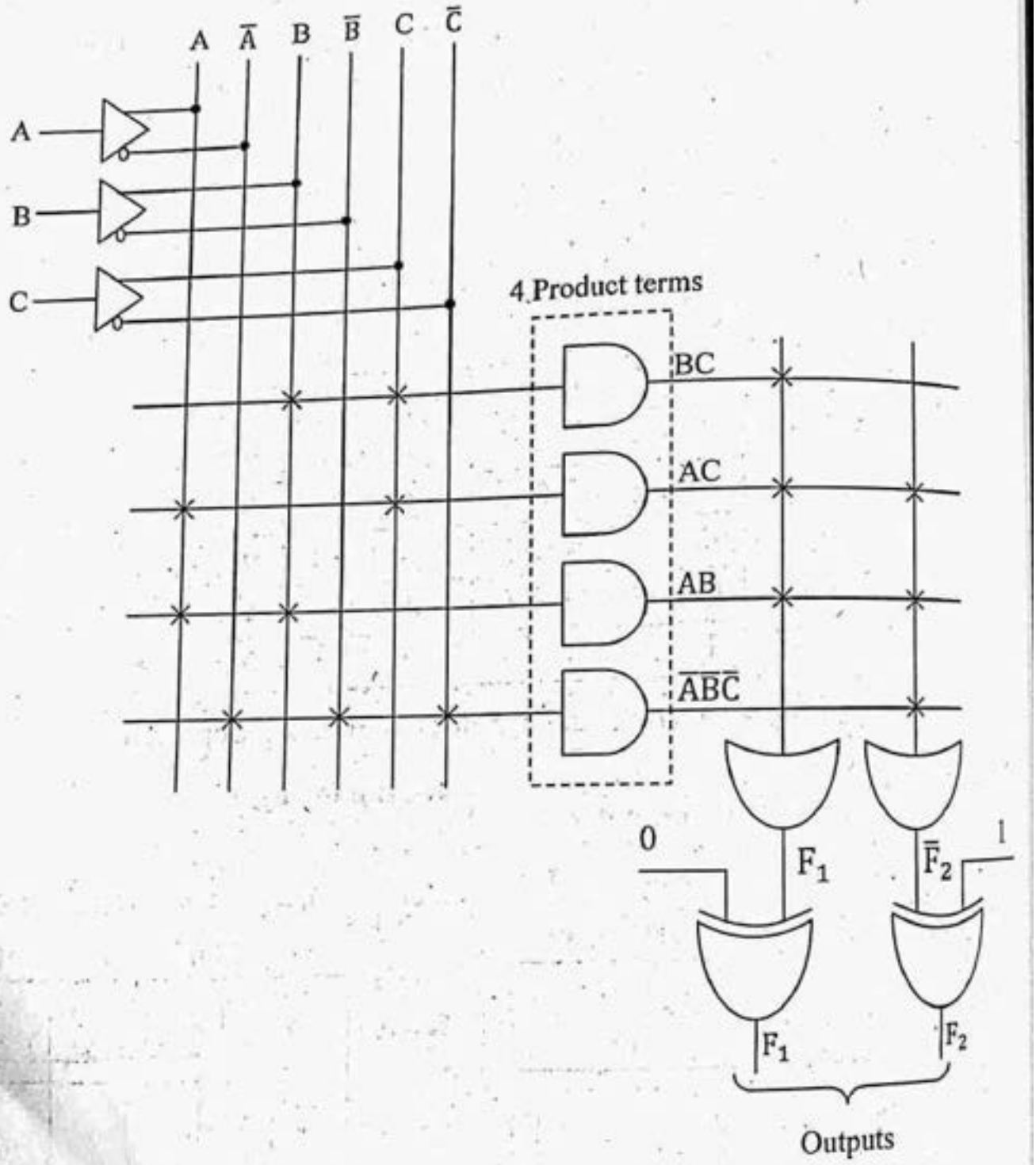


Figure 4.100 PLA

Connect an XOR gate to the complement term $\overline{F}_2$, in which other input of XOR is made high.

Example 4.23: Design a BCD to Excess-3 code converter and implement using suitable PLA.

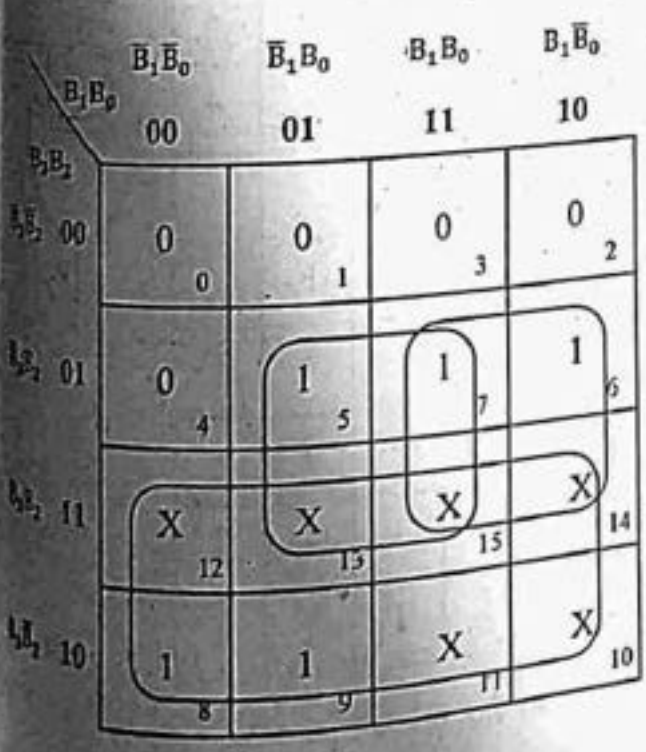
Solution:

The truth table of BCD to Excess-3 code converter is shown in table 4.58

BCD code				Excess-3 code			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

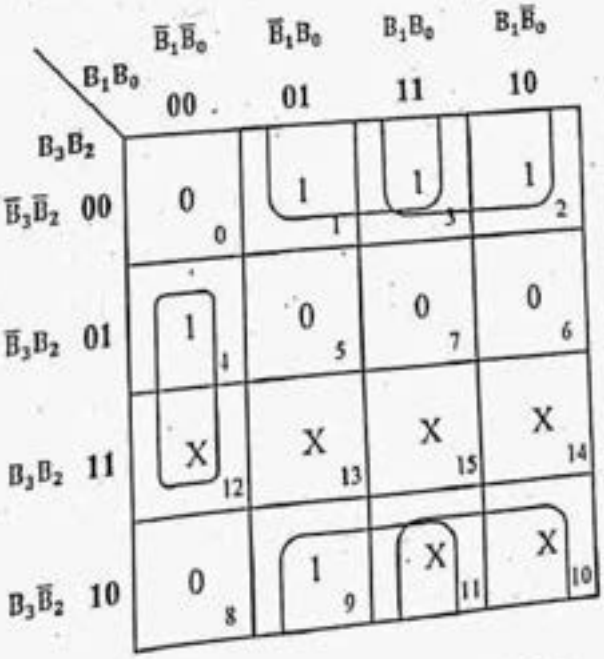
Table 4.58 Truth table for BCD to Excess-3 Converter

K-map for E_3



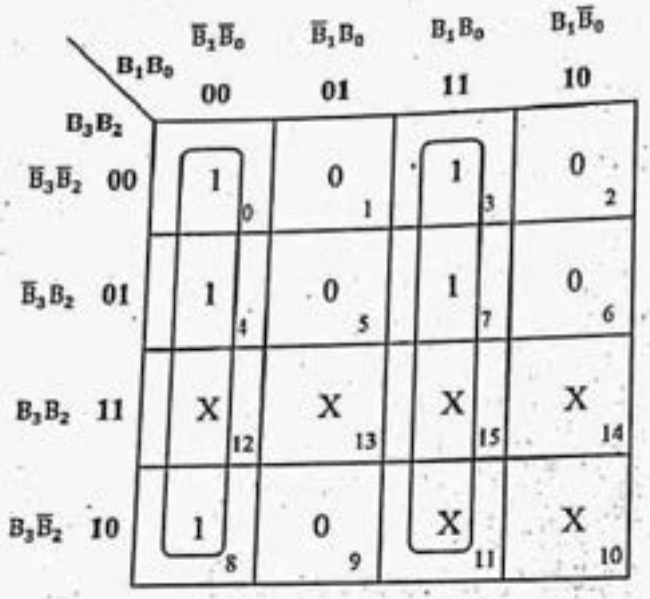
$$E_3 = B_3 + B_2\bar{B}_0 + B_2B_1$$

K-map for E_2



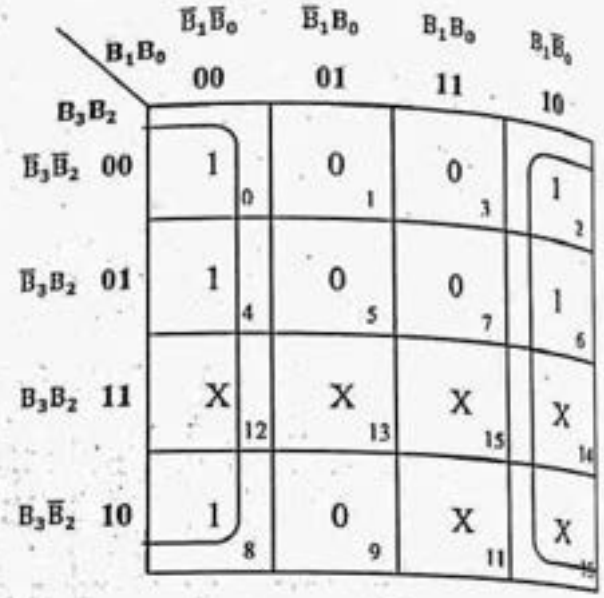
$$E_2 = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$$

K-map for E_1



$$E_1 = \overline{B}_1\overline{B}_0 + B_1B_0$$

K-map for E_0



$$E_0 = \overline{B}_0$$

Product terms	Inputs				Outputs			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
B_3	1	-	-	-	1	-	-	-
B_2B_0	-	1	-	1	1	-	-	-
B_2B_1	-	1	1	-	1	-	-	-
$B_2\overline{B}_1\overline{B}_0$	-	1	0	0	-	1	-	-
$\overline{B}_2B_0$	-	0	-	1	-	1	-	-
$\overline{B}_2B_1$	-	0	1	-	-	1	-	-
$\overline{B}_1\overline{B}_0$	-	-	0	0	-	-	1	-
B_1B_0	-	-	1	1	-	-	1	-
$\overline{B}_0$	-	-	-	0	-	-	-	1
					T	T	T	T

Table 4.59 PLA program table

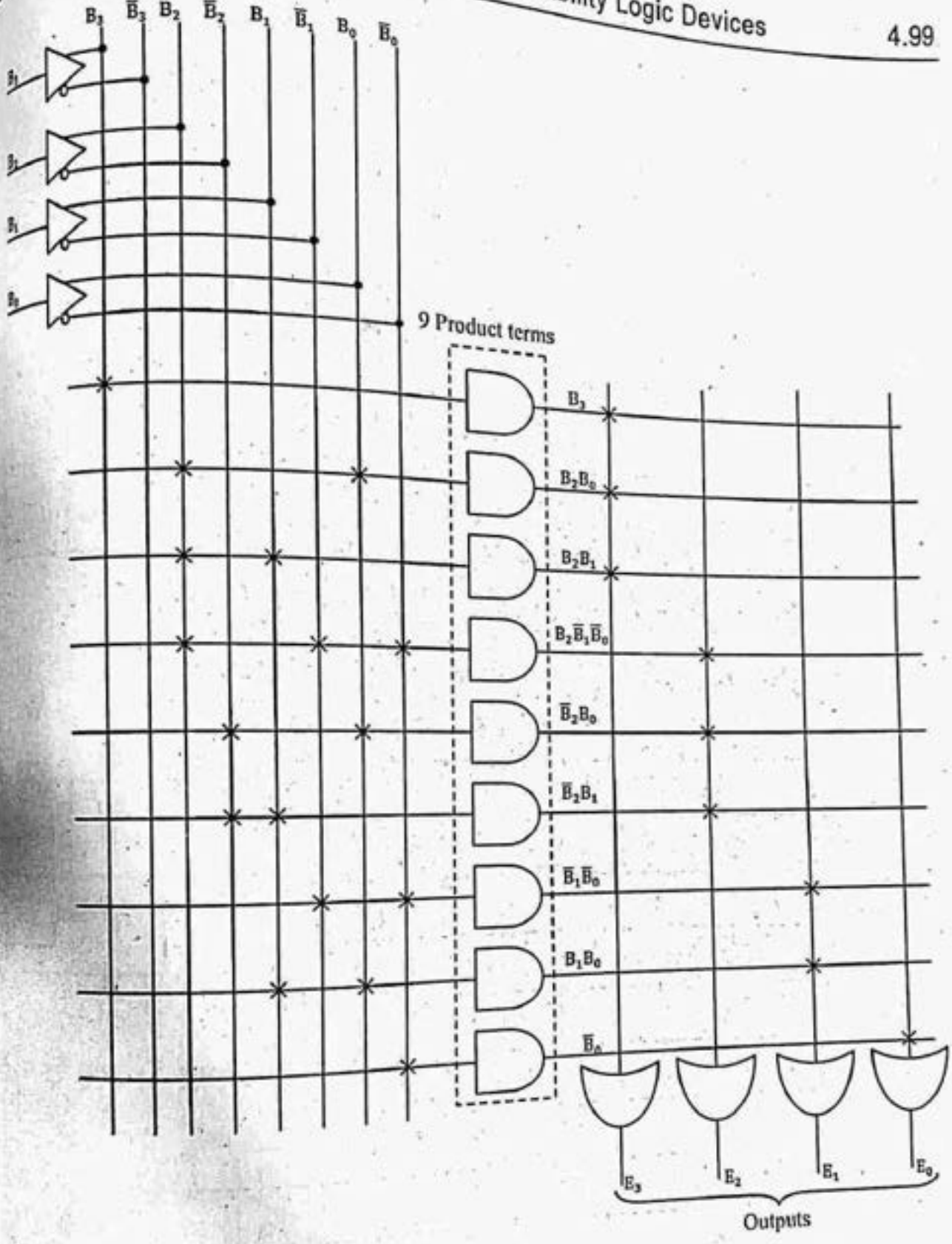


Figure 4.101 PLA

Example 4.24: *Implement the following functions using programmable logic array*

$$f_1(x, y, z) = \sum m(0, 1, 3, 5, 7)$$

$$f_2(x, y, z) = \sum m(2, 4, 6)$$

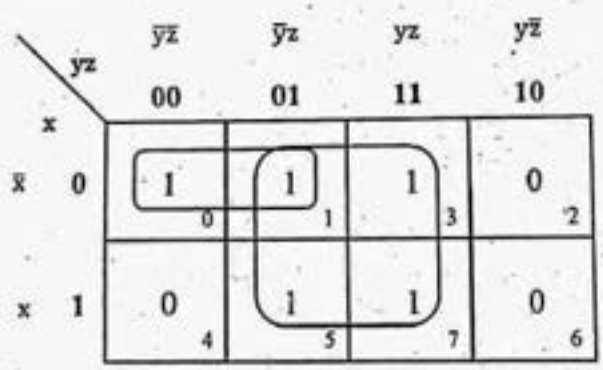
Solution: The truth table for the given Boolean functions is shown in table 4.60.

Let, $F_1(x,y,z) = \sum_m(0,1,3,5,7)$; $F_2(x,y,z) = \sum_m(2,4,6)$

Inputs			Outputs	
x	y	z	F_1	F_2
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

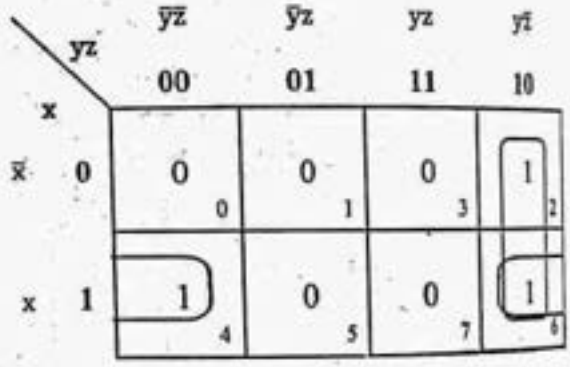
Table 4.60 Truth table

K-map for F_1



$F_1 = z + \bar{x}\bar{y}$

K-map for F_2



$F_2 = y\bar{z} + x\bar{z}$

Here $F_1 = z + \bar{x}\bar{y}$ and $F_2 = y\bar{z} + x\bar{z}$. The two functions F_1 and F_2 totally have 4 different product terms z, $\bar{x}\bar{y}$, $y\bar{z}$ and $x\bar{z}$.

Product terms	Inputs			Outputs	
	x	y	z	F_1	F_2
z	-	-	1	1	-
$\bar{x}\bar{y}$	0	0	-	1	-
$y\bar{z}$	-	1	0	-	1
$x\bar{z}$	1	-	0	-	1
				T	T

Table 4.61 PLA program table

Under each output variable, we write a 'T' if the output inverter is to be bypassed and 'C' if the function is to be complemented with the output inverter.

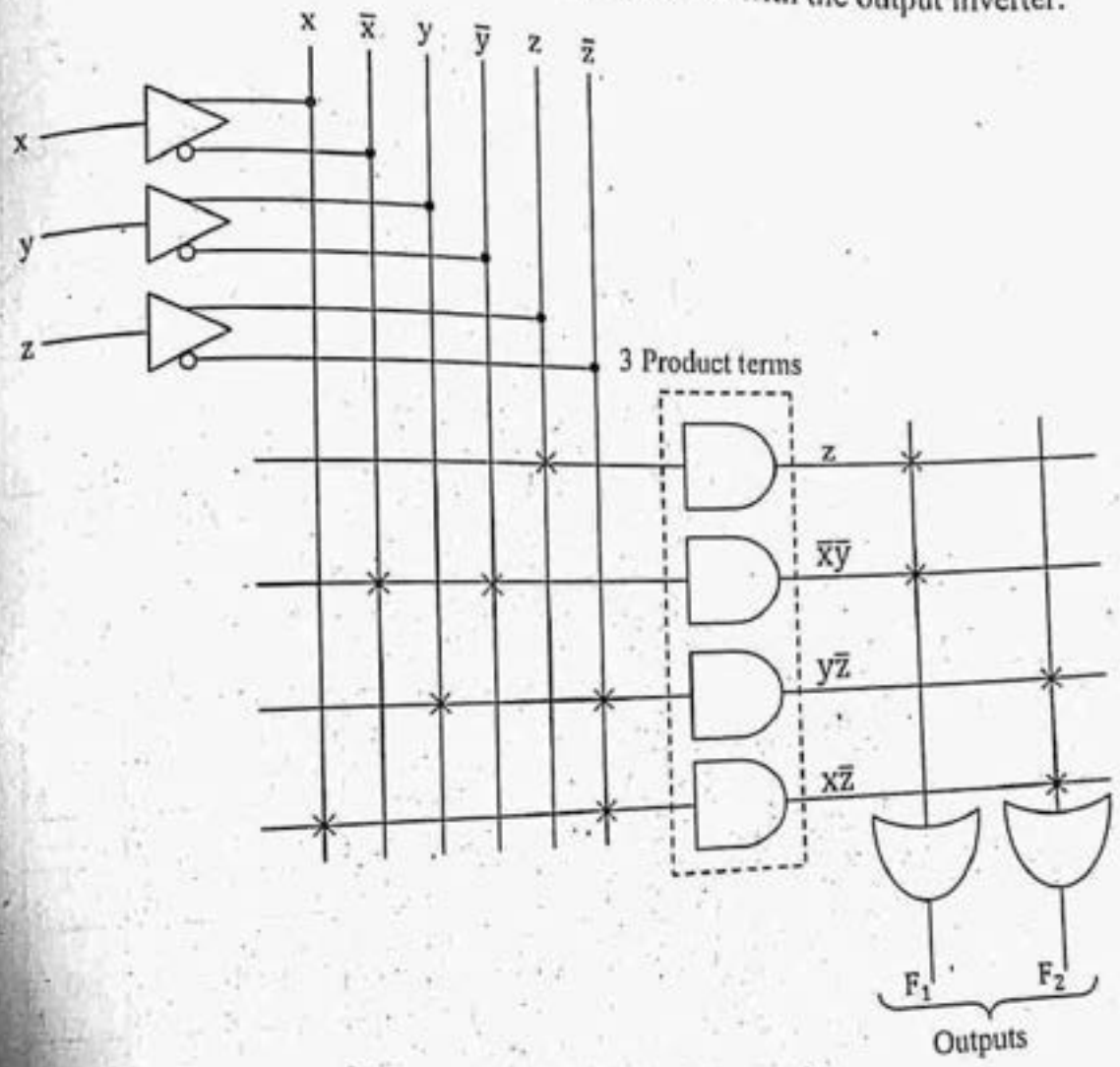


Figure 4.102 PLA

Example 4.25: Design a PLA structure using AND and OR logic for the following functions

$F_1 = \sum m(0,1,2,3,4,7,8,11,12,15)$ $\quad F_2 = \sum m(2,3,6,7,8,9,12,13)$

$F_3 = \sum m(1,3,7,8,11,12,15)$ $\quad F_4 = \sum m(0,1,4,8,11,12,15)$

Solution:

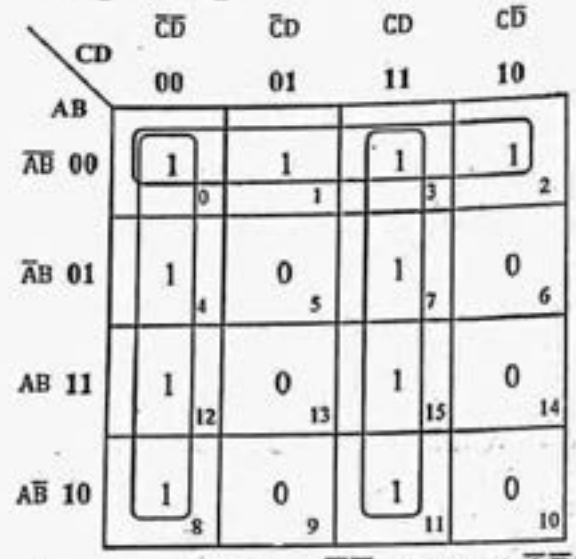
$F_1(A,B,C,D) = \sum m(0,1,2,3,4,7,8,11,12,15)$

$F_2(A,B,C,D) = \sum m(2,3,6,7,8,9,12,13)$

$F_3(A,B,C,D) = \sum m(1,3,7,8,11,12,15)$

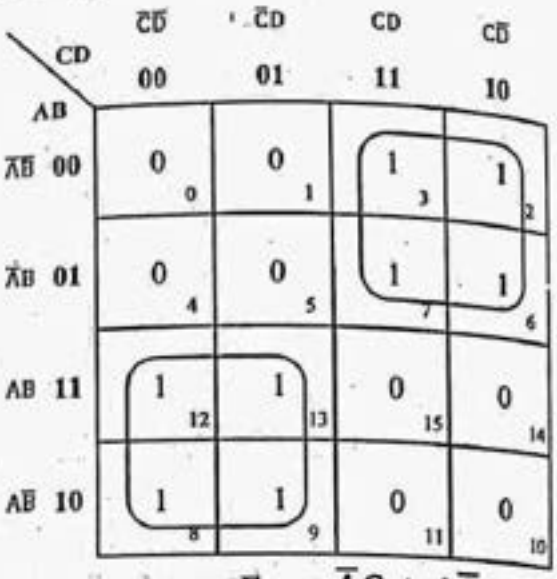
$F_4(A,B,C,D) = \sum m(0,1,4,8,11,12,15)$

K-map for F_1



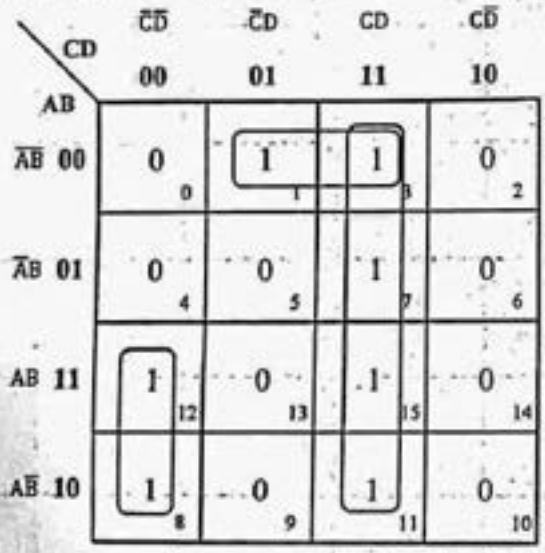
$F_1 = \bar{C}\bar{D} + CD + \bar{A}\bar{B}$

K-map for F_2



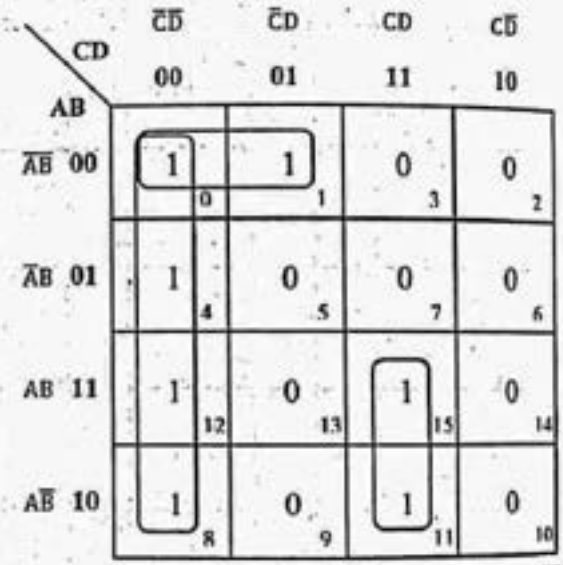
$F_2 = \bar{A}C + A\bar{C}$

K-map for F_3



$F_3 = CD + A\bar{C}\bar{D} + \bar{A}\bar{B}D$

K-map for F_4



$F_4 = \bar{C}\bar{D} + ACD + \bar{A}B\bar{C}$

Product terms	Inputs				Outputs			
	A	B	C	D	F_1	F_2	F_3	F_4
$\bar{C}\bar{D}$	-	-	0	0	1	-	-	1
CD	-	-	1	1	1	-	1	-
$\bar{A}\bar{B}$	0	0	-	-	1	-	-	-
$\bar{A}C$	0	-	1	-	-	1	-	-
$A\bar{C}$	1	-	0	-	-	1	-	-
$A\bar{C}\bar{D}$	1	-	0	0	-	-	1	-
$\bar{A}\bar{B}D$	0	0	-	1	-	-	1	-
ACD	1	-	1	1	-	-	-	1
$\bar{A}B\bar{C}$	0	0	0	-	-	-	-	1
					T	T	T	T

Table 4.62 PLA Program table

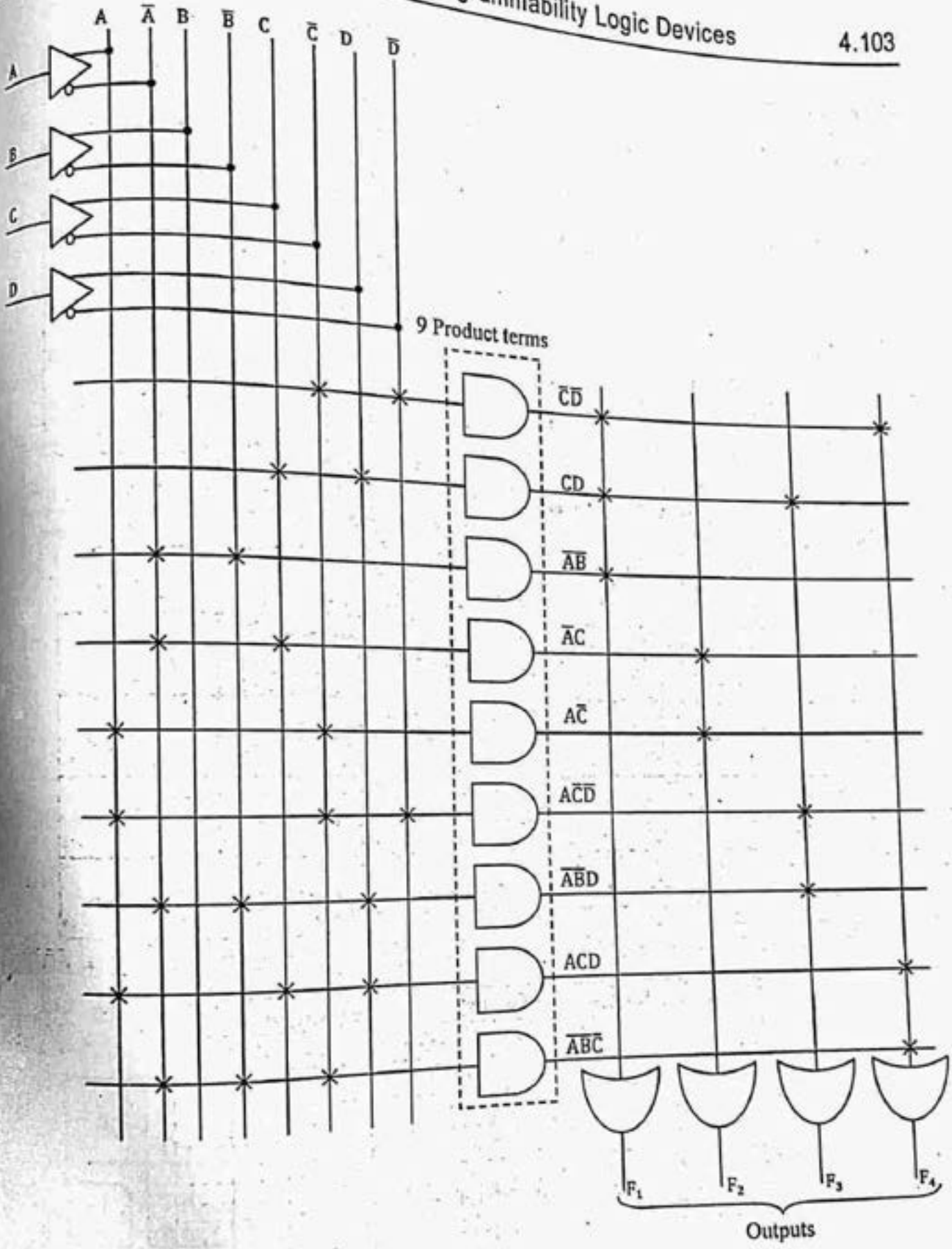


Figure 4.103 PLA

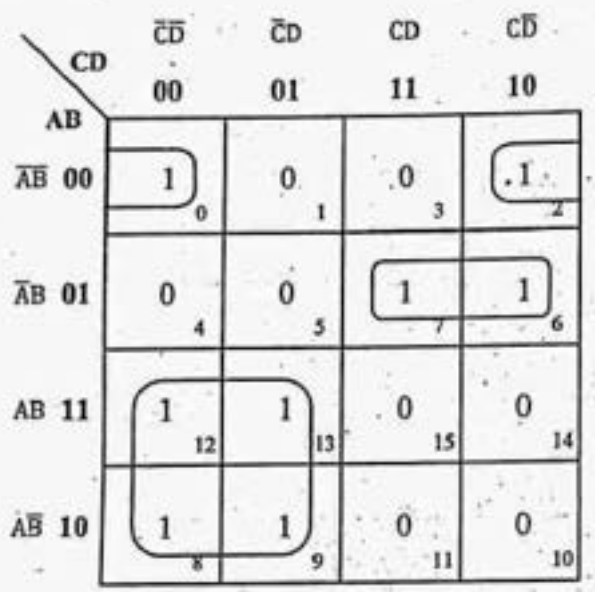
4.16.5 Implementation of combinational circuits using PAL

Example 4.26: Implement the following Boolean function using PAL.

$$w(A,B,C,D) = \sum_m(0,2,6,7,8,9,12,13)$$
$$x(A,B,C,D) = \sum_m(0,2,6,7,8,9,12,13,14)$$
$$y(A,B,C,D) = \sum_m(2,3,8,9,10,12,13)$$
$$z(A,B,C,D) = \sum_m(1,3,4,6,9,12,14)$$

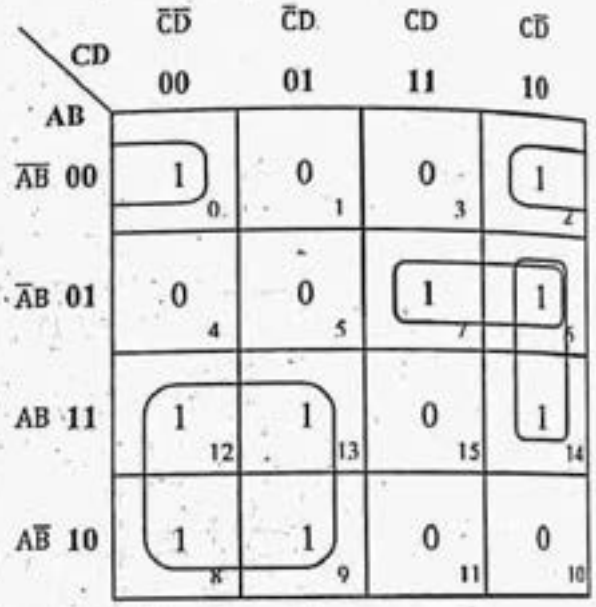
Solution:

K-map for w



$$w = A\overline{C} + \overline{A}BC + \overline{A}B\overline{D}$$

K-map for x

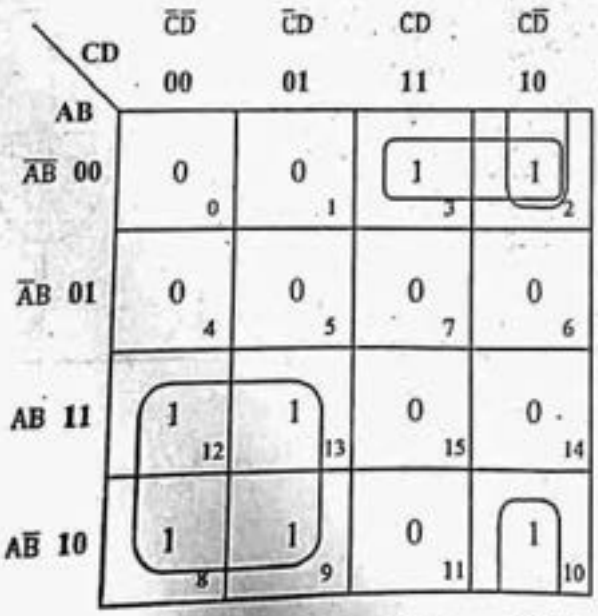


$$x = A\overline{C} + \overline{A}BC + \overline{A}B\overline{D} + BC\overline{D}$$

Substitute $w = A\overline{C} + \overline{A}BC + \overline{A}B\overline{D}$ in x

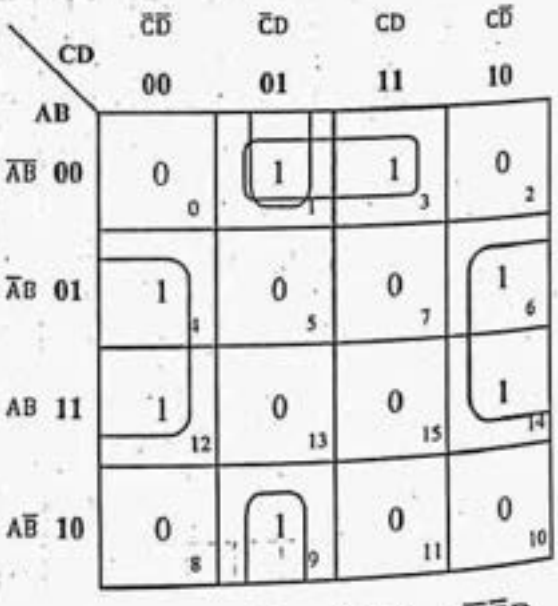
We get $x = w + BC\overline{D}$

K-map for y



$$y = A\overline{C} + \overline{A}B\overline{C} + \overline{B}C\overline{D}$$

K-map for z



$$z = B\overline{D} + \overline{A}B\overline{D} + \overline{B}C\overline{D}$$

The program for PAL is shown in table 4.63

Product term	AND inputs					Outputs
	A	B	C	D	w	
$A\bar{C}$	1	-	0	-	-	$w = A\bar{C} + \bar{A}BC + \bar{A}B\bar{D}$
$\bar{A}BC$	0	1	1	-	-	
$\bar{A}B\bar{D}$	0	0	-	0	-	
w	-	-	-	-	1	$x = w + BC\bar{D}$
$BC\bar{D}$	-	1	1	0	-	
-	-	-	-	-	-	
$A\bar{C}$	1	-	0	-	-	$y = A\bar{C} + \bar{A}BC + \bar{B}C\bar{D}$
$\bar{A}BC$	0	0	1	-	-	
$\bar{B}C\bar{D}$	-	0	1	0	-	
$B\bar{D}$	-	1	-	0	-	$z = B\bar{D} + \bar{A}BD + \bar{B}C\bar{D}$
$\bar{A}BD$	0	0	-	1	-	
$\bar{B}C\bar{D}$	-	0	0	1	-	

Table 4.63 PAL program table

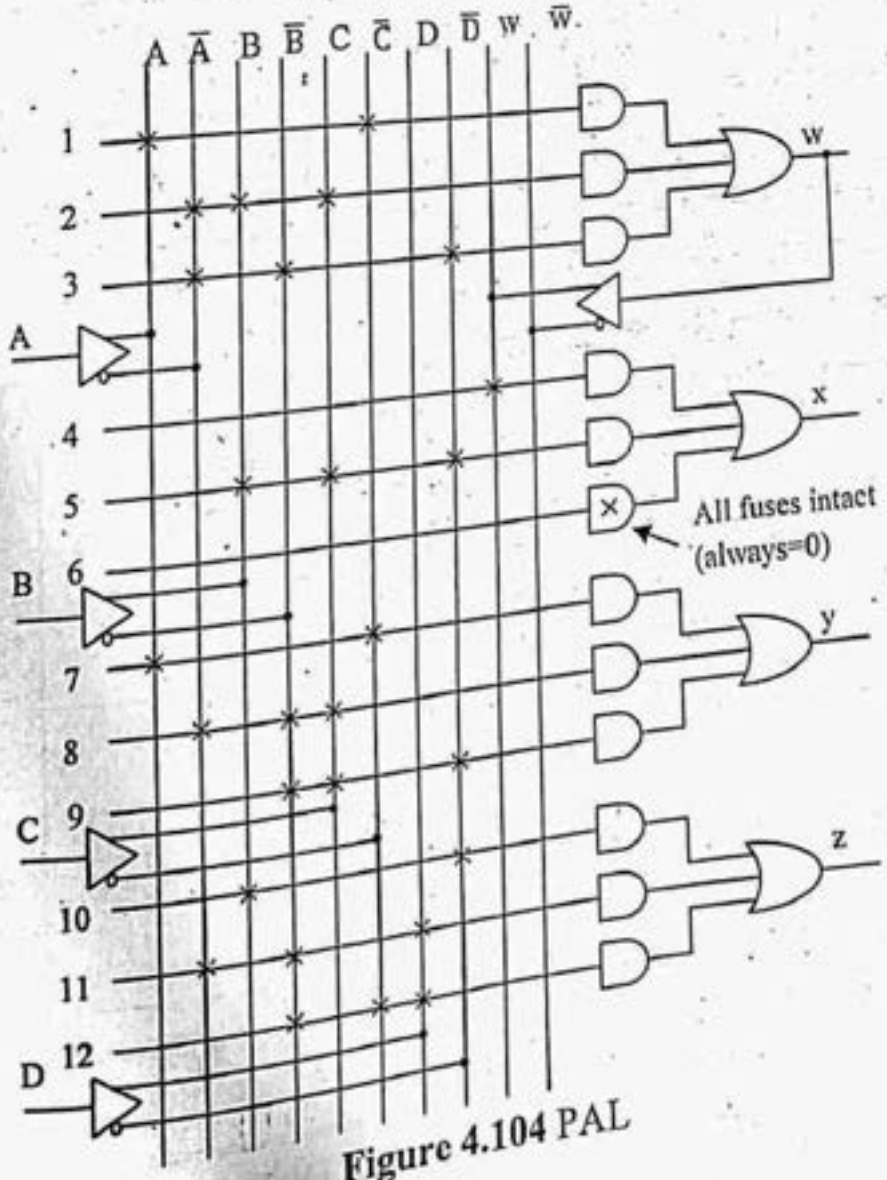
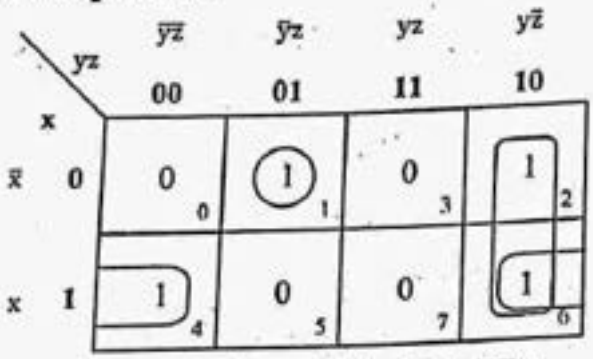


Figure 4.104 PAL

Example 4.27: Tabulate the PAL programmable table for the four Boolean functions listed below. $A(x,y,z) = \sum_m(1,2,4,6)$, $B(x,y,z) = \sum_m(0,1,6,7)$, $C(x,y,z) = \sum_m(2,6)$, $D(x,y,z) = \sum_m(1,2,3,5,7)$

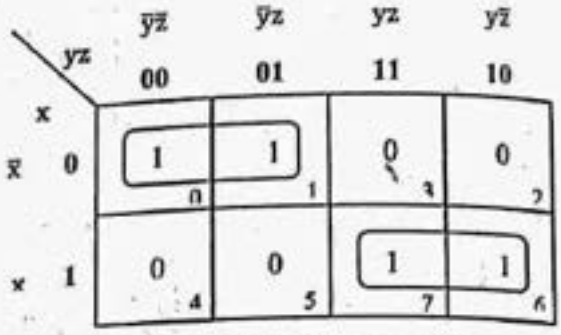
Solution :

K-map for A



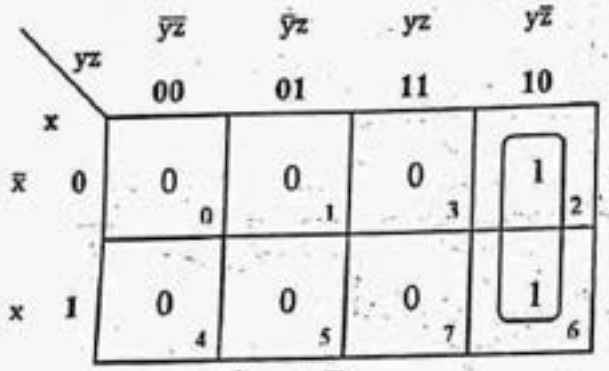
$A = y\bar{z} + x\bar{z} + \bar{x}\bar{y}z$

K-map for B



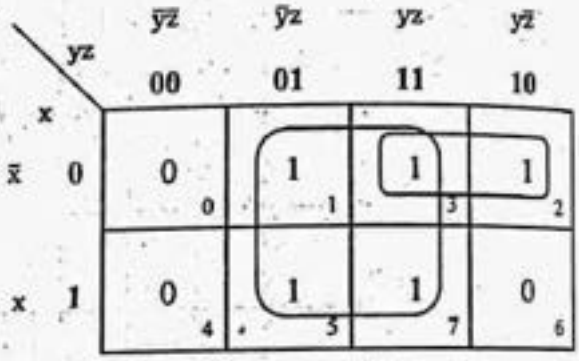
$B = \bar{x}\bar{y} + xy$

K-map for C



$C = y\bar{z}$

K-map for D



$D = z + \bar{x}y$

Here $C = y\bar{z}$

Therefore $A = C + x\bar{z} + \bar{x}\bar{y}z$; $B = \bar{x}\bar{y} + xy$; $C = y\bar{z}$; $D = z + \bar{x}y$

Product term	AND inputs				Outputs
	x	y	z	C	
C	-	-	-	1	$A = C + x\bar{z} + \bar{x}\bar{y}z$
$x\bar{z}$	1	-	0	-	
$\bar{x}\bar{y}z$	0	0	1	-	
$\bar{x}\bar{y}$	0	0	-	-	$B = \bar{x}\bar{y} + xy$
xy	1	1	-	-	
-	-	-	-	-	
$y\bar{z}$	-	1	0	-	$C = y\bar{z}$
-	-	-	-	-	
-	-	-	-	-	
z	-	-	1	-	$D = z + \bar{x}y$
$\bar{x}y$	0	1	-	-	
-	-	-	-	-	

Table 4.64 PAL program table

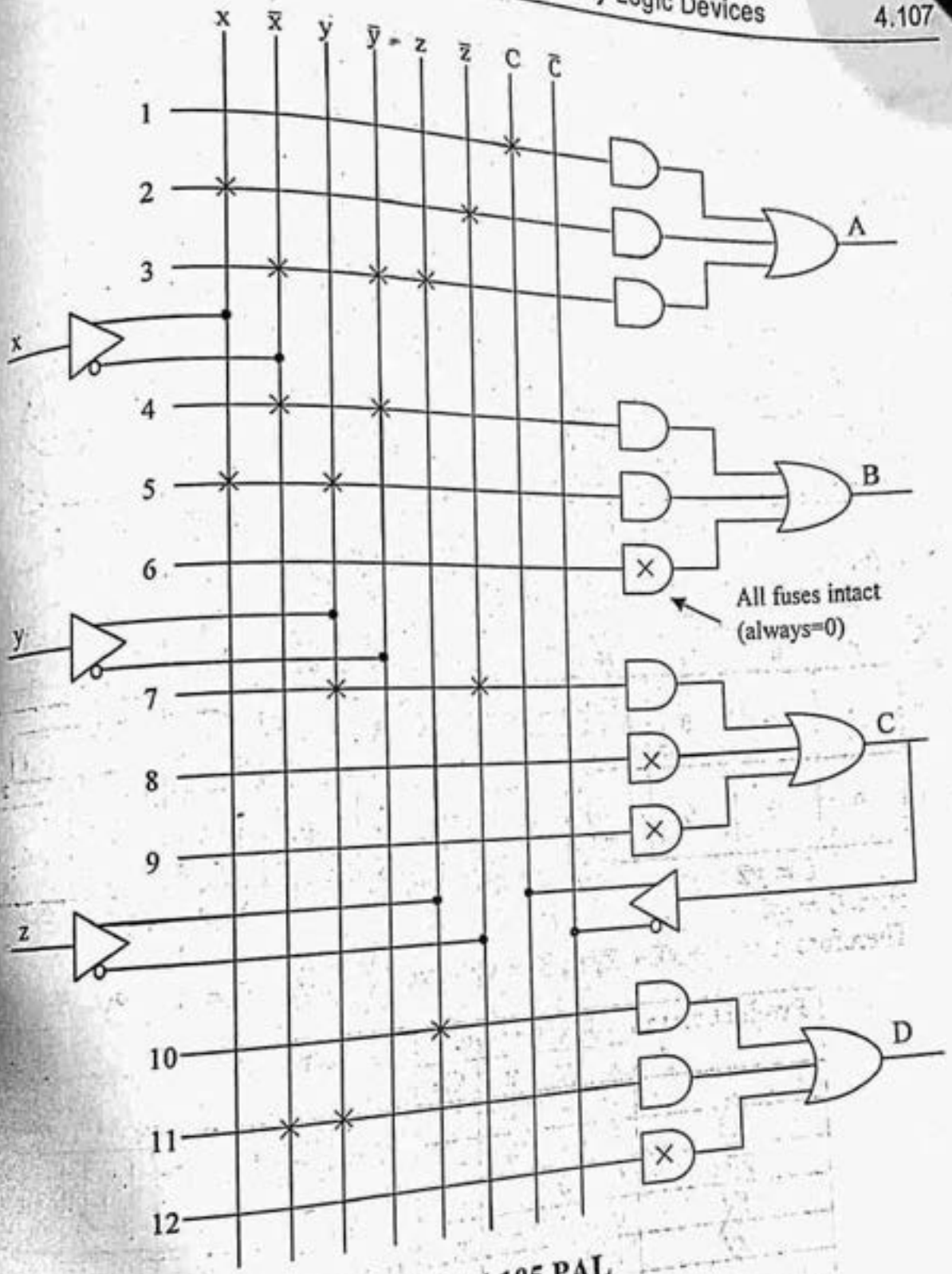


Figure 4.105 PAL

Example 4.28: Implement a 3 bit up/down counter using PAL devices.

Solution:

Consider a control input 'x'. If x = 1, the counter will works as a up counter. If x = 0, the counter will works as a down counter. The state diagram of 3 bit up/down counter is shown in figure 4.106.

Present state			Input	Next state			Flip-flop inputs		
A	B	C	x	A^+	B^+	C^+	D_A	D_B	D_C
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	1	0
0	1	0	0	0	0	1	0	0	1
0	1	0	1	0	1	1	0	1	1
0	1	1	0	0	1	0	0	1	0
0	1	1	1	1	0	0	1	0	0
1	0	0	0	0	1	1	0	1	1
1	0	0	1	1	0	1	1	0	1
1	0	1	0	1	0	0	1	0	0
1	0	1	1	1	1	0	1	1	0
1	1	0	0	1	0	1	1	0	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	0	1	1	0
1	1	1	1	0	0	0	0	0	0

Table 4.65 Transition table

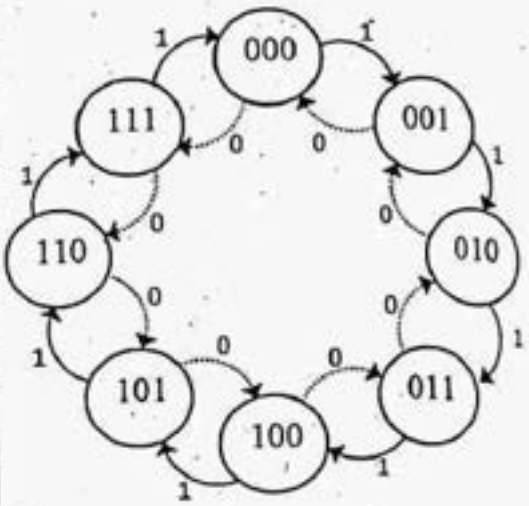
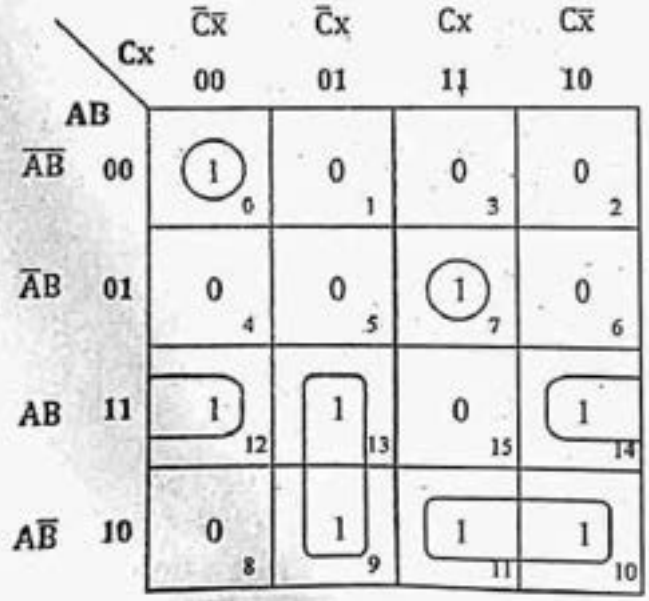


Figure 4.106 State diagram

The transition table of a 3 bit up/down counter is shown in table 4.65.

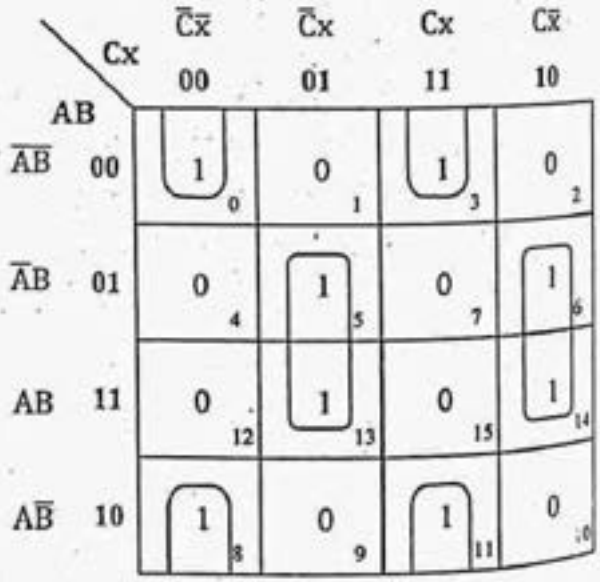
The equations for D_A, D_B and D_C can be determined using K-map as shown below.

K-map for D_A



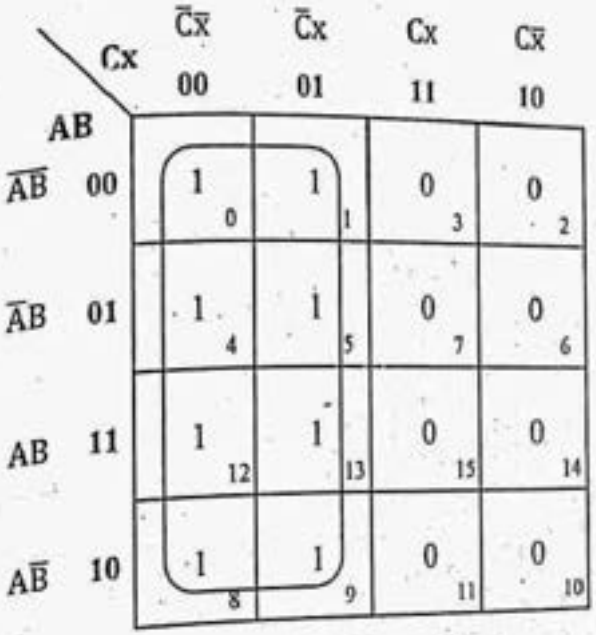
$$D_A = A\bar{C}x + A\bar{B}C + AB\bar{x} + \bar{A}B\bar{C}\bar{x} + \bar{A}BCx$$

K-map for D_B



$$D_B = B\bar{C}x + BC\bar{x} + \bar{B}C\bar{x} + \bar{B}Cx$$

Map for D_C



$$D_C = \bar{C}$$

The program table for PAL is shown in table 4.66.

Product term	AND inputs				Outputs
	A	B	C	x	
$A\bar{C}x$	1	-	0	1	$D_A = A\bar{C}x + A\bar{B}C + AB\bar{x} + \bar{A}\bar{B}\bar{C}\bar{x} + \bar{A}BCx$
$A\bar{B}C$	1	0	1	-	
$AB\bar{x}$	1	1	-	0	
$\bar{A}\bar{B}\bar{C}\bar{x}$	0	0	0	0	
$\bar{A}BCx$	0	1	1	1	
$B\bar{C}x$	-	1	0	1	$D_B = B\bar{C}x + BC\bar{x} + \bar{B}\bar{C}\bar{x} + \bar{B}Cx$
$BC\bar{x}$	-	1	1	0	
$\bar{B}\bar{C}\bar{x}$	-	0	0	0	
$\bar{B}Cx$	-	0	1	1	
-	-	-	-	-	$D_C = \bar{C}$
$\bar{C}$	-	-	0	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	

Table 4.66 PAL program table

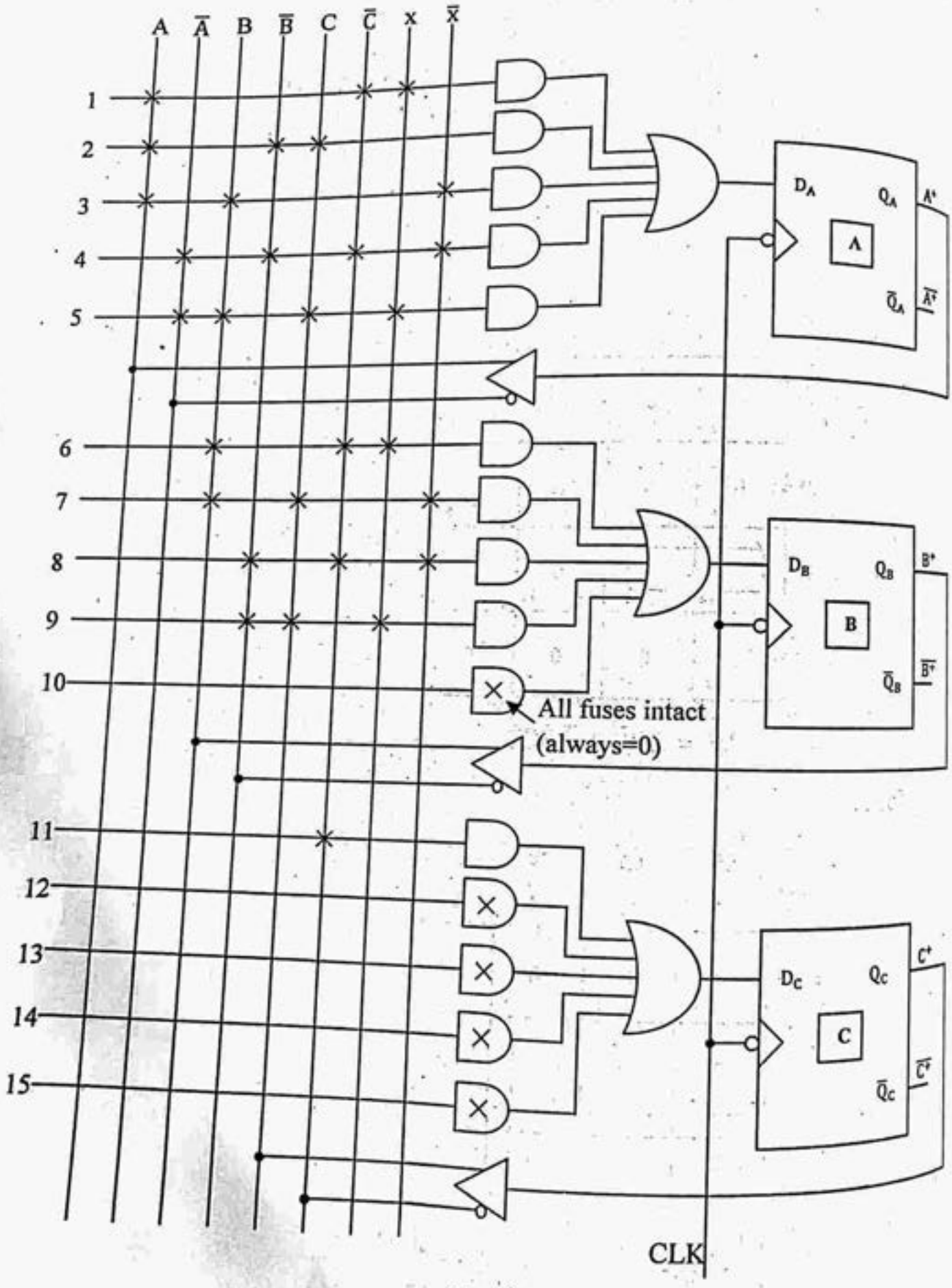


Figure 4.107 PAL

Example

$W(A,B,C,D) = \sum_m(2,12,13)$

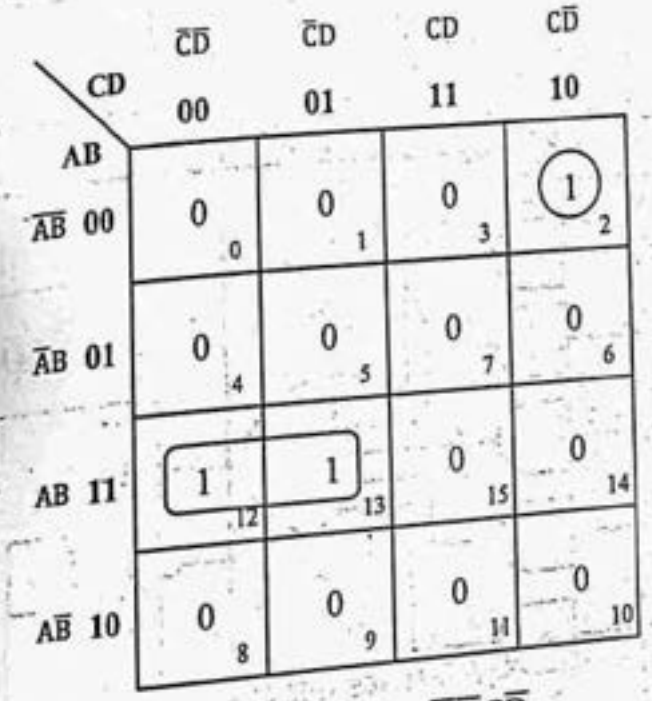
$X(A,B,C,D) = \sum_m(7,8,9,10,11,12,13,14,15)$

$Y(A,B,C,D) = \sum_m(0,2,3,4,5,6,7,8,10,11,15)$

$Z(A,B,C,D) = \sum_m(1,2,8,12,13)$

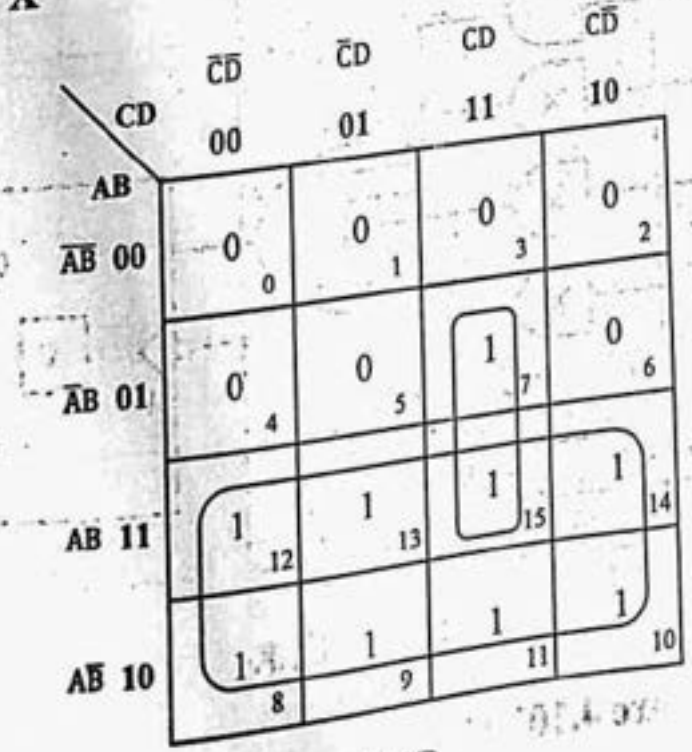
Solution:

K-map for W



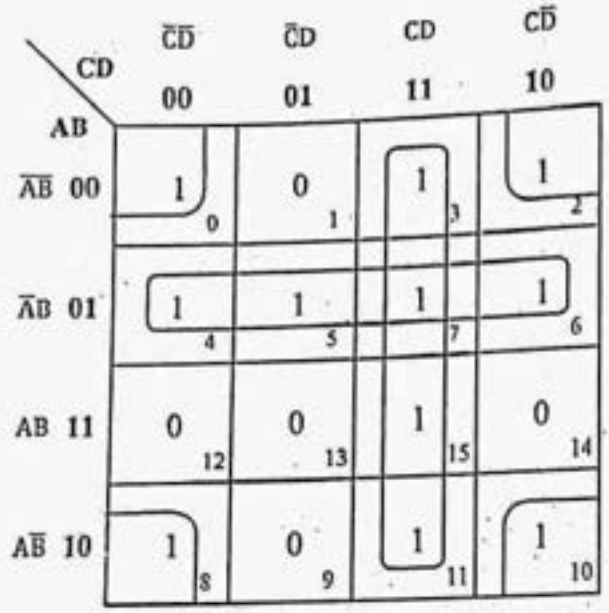
$W = AB\bar{C} + \bar{A}BC\bar{D}$

K-map for X



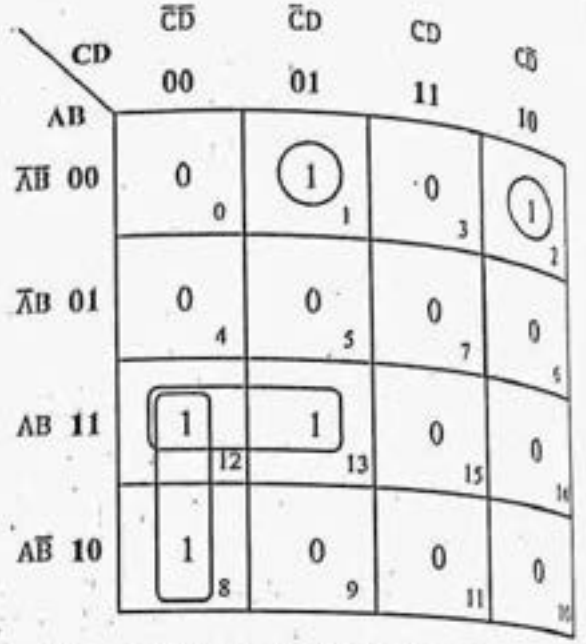
$X = A + BCD$

K-map for Y



$Y = \bar{A}B + CD + \bar{B}\bar{D}$

K-map for Z



$Z = A\bar{C}\bar{D} + AB\bar{C} + \bar{A}B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

Substituting $W = AB\bar{C} + \bar{A}B\bar{C}\bar{D}$ in Z, we get

$$Z = A\bar{C}\bar{D} + W + \bar{A}\bar{B}\bar{C}D$$

Product term	AND inputs					Outputs
	A	B	C	D	W	
$AB\bar{C}$	1	1	0	-	-	$W = AB\bar{C} + \bar{A}B\bar{C}\bar{D}$
$\bar{A}B\bar{C}\bar{D}$	0	0	1	0	-	
-	-	-	-	-	-	
A	1	-	-	-	-	$X = A + BCD$
BCD	-	1	1	1	-	
-	-	-	-	-	-	
$\bar{A}B$	0	1	-	-	-	$Y = \bar{A}B + CD + \bar{B}\bar{D}$
CD	-	-	1	1	-	
$\bar{B}\bar{D}$	-	0	-	0	-	
$A\bar{C}\bar{D}$	1	-	0	0	-	$Z = A\bar{C}\bar{D} + W + \bar{A}\bar{B}\bar{C}D$
W	-	-	-	-	1	
$\bar{A}\bar{B}\bar{C}D$	0	0	0	1	-	

Table 4.67 PAL program table

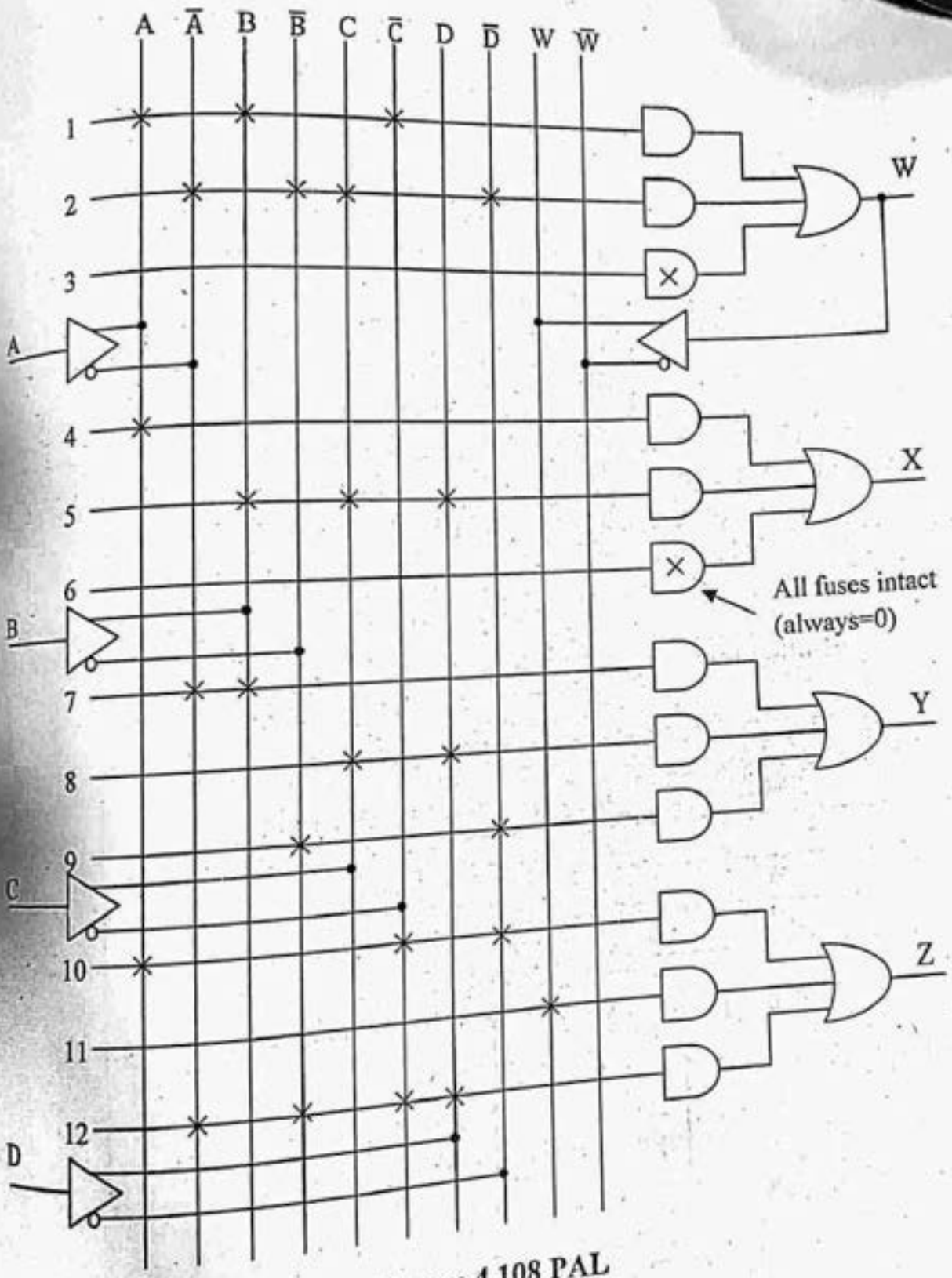


Figure 4.108 PAL

Example 4.30: *Implement the following function using PLA and PAL:*

$$f(x,y,z) = \sum m(0,1,3,5,7)$$

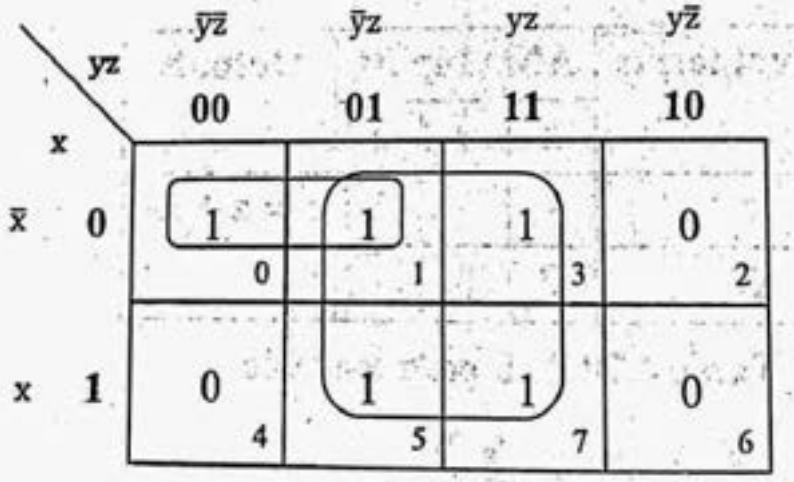
Solution:

The truth table for the given Boolean function is shown in table 4.68.

Inputs			Outputs
x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 4.68 Truth table

K-map for f



$f = z + \bar{x}\bar{y}$

PLA Implementation

Product terms	Inputs			Outputs
	x	y	z	f
z	-	-	1	1
$\bar{x}\bar{y}$	0	0	-	1
				T

Table 4.69 PLA program table

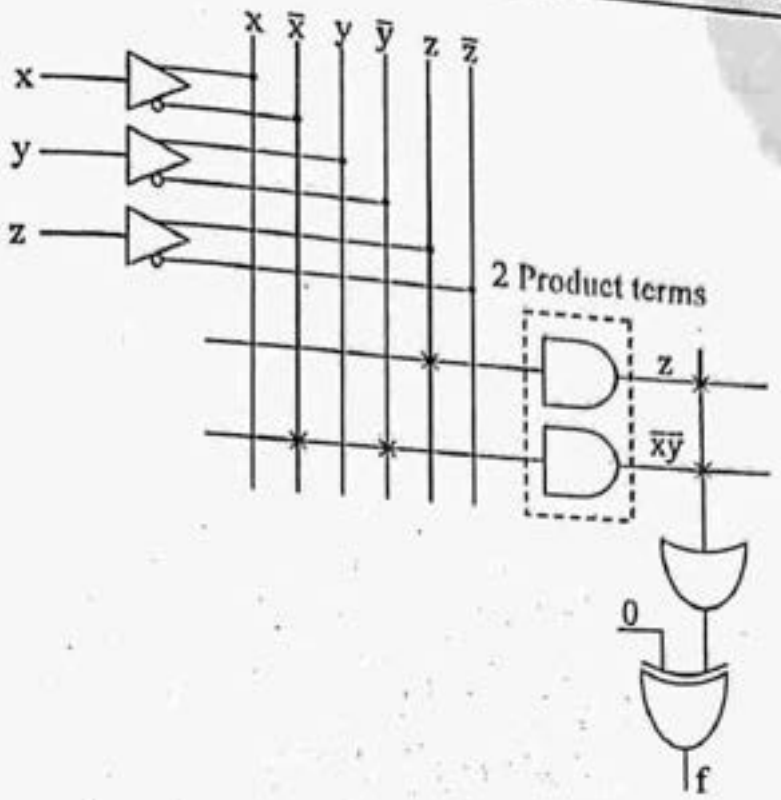


Figure 4.109 PLA

PAL Implementation

Product terms	AND Inputs			Outputs
	x	y	z	
z	-	-	1	$f = z + \bar{x}\bar{y}$
$\bar{x}\bar{y}$	0	0	-	

Table 4.70 PAL program table

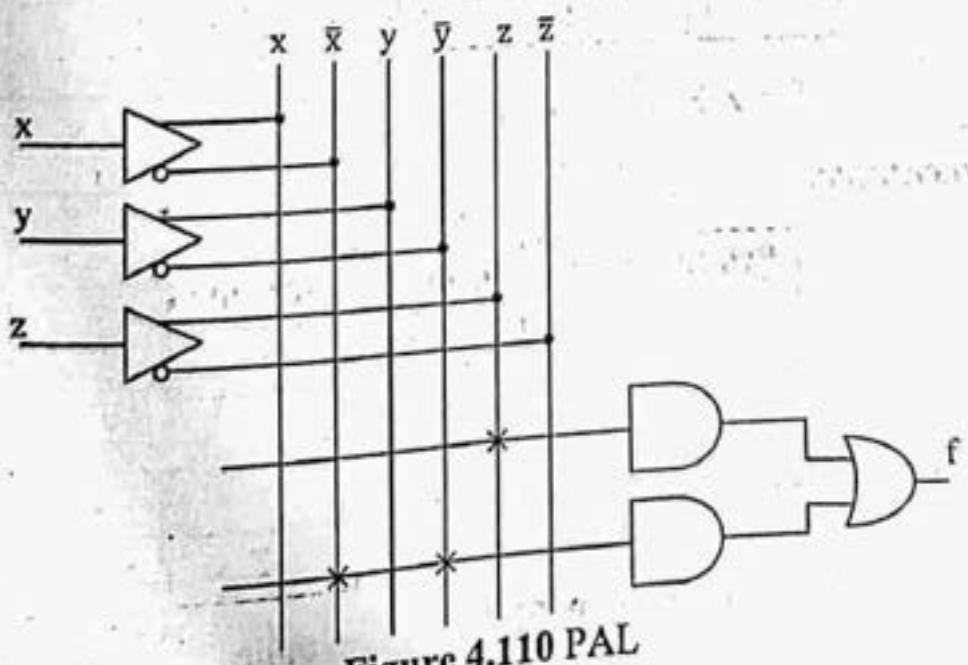


Figure 4.110 PAL

4.16.6 Implementation of combinational circuits using PROM

Example 4.31: Using PROM realize the following expression

$$F_1(a,b,c) = \sum_m(0,1,3,5,7)$$
$$F_2(a,b,c) = \sum_m(1,2,5,6)$$

Solution:

The given function F_1 and F_2 have 3 inputs a, b, and c. They generate $2^3 = 8$ minterms and hence the PROM have 8 AND gates. Since there are 2 outputs the PROM have two OR gates.

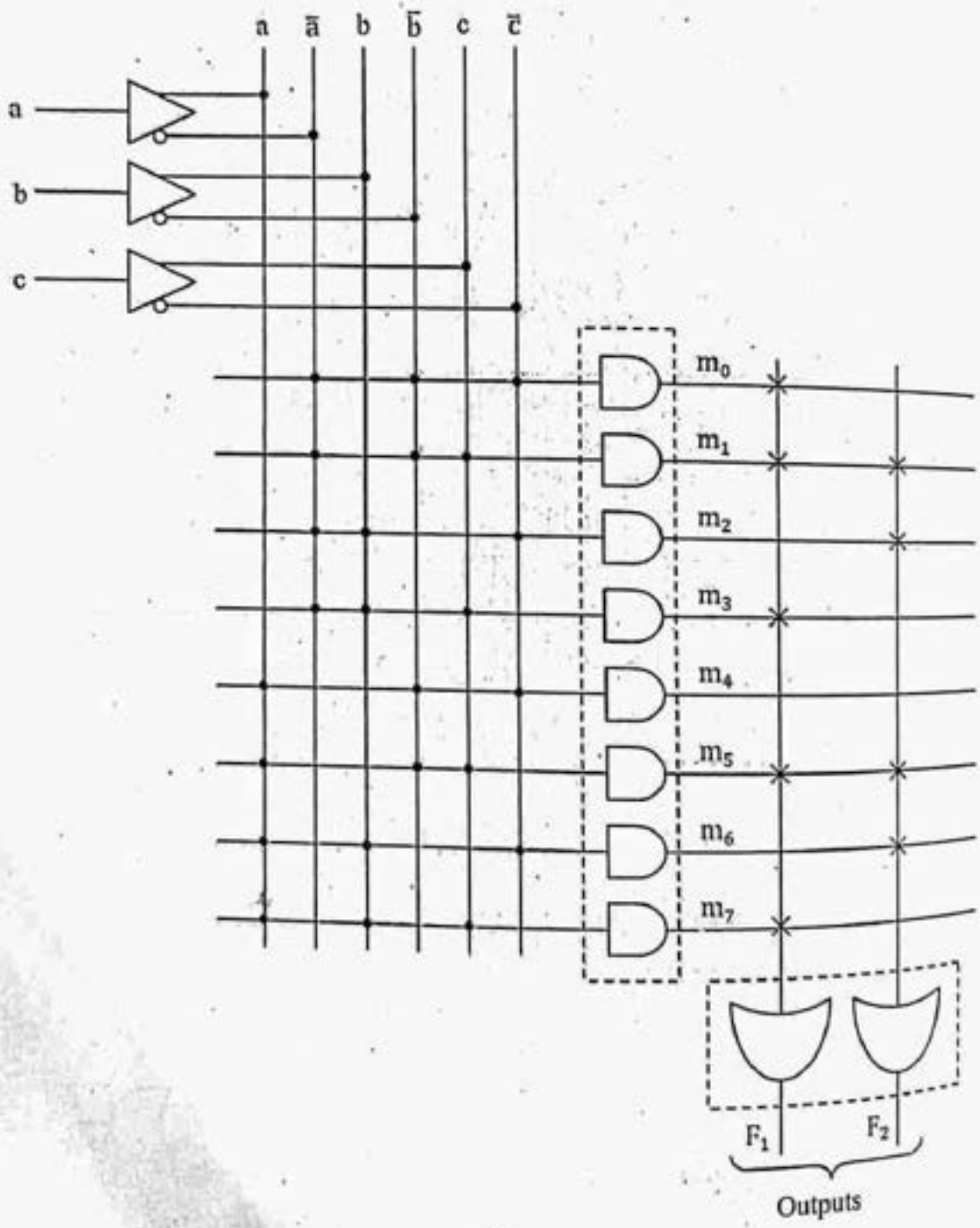


Figure 4.111 PROM

Example 4.32: Design a combinational circuit using a PROM. The circuit accepts 3 bit binary number and generates its equivalent Excess-3 code.

Solution:

The truth table of a 3 bit binary to its equivalent Excess-3 code is shown in table 4.71.

Binary input			Excess-3 output			
B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Table 4.71 Truth table of Binary to Excess-3 Conversion

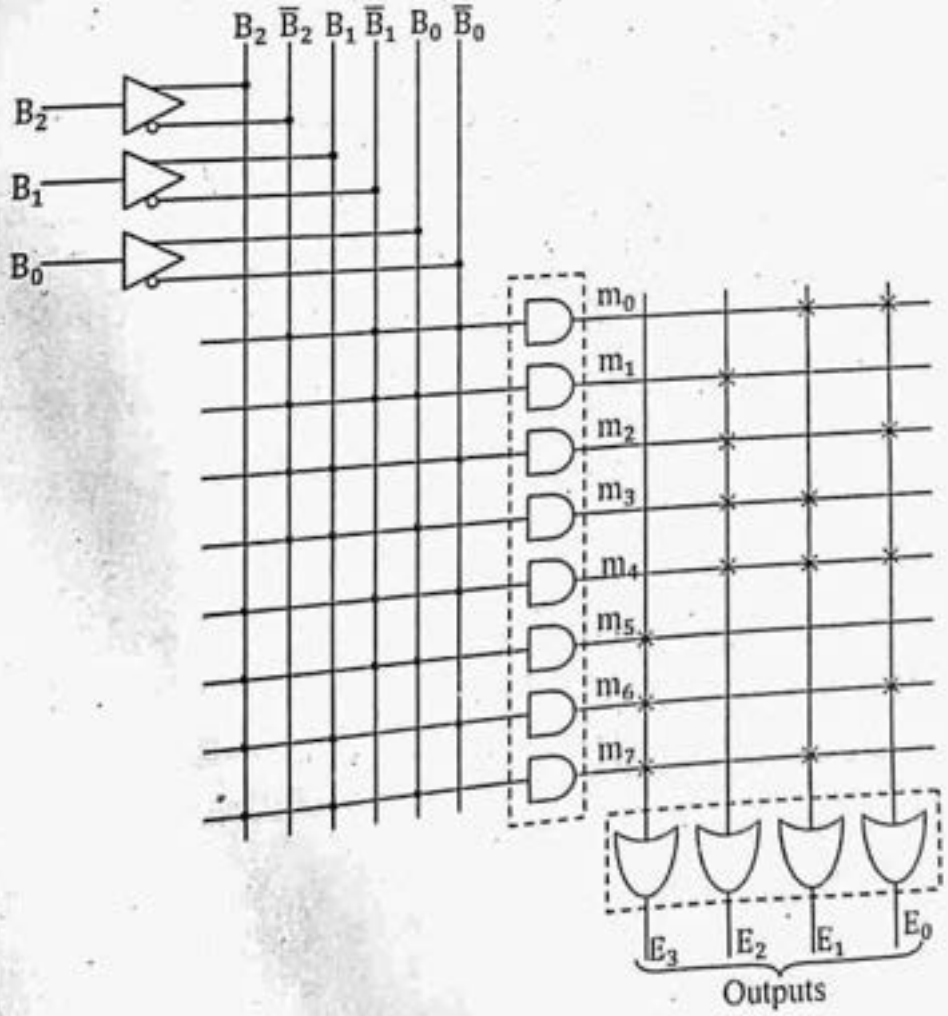


Figure 4.112 PROM

Example 4.33: We have found a minimum sum of products expression for each of two functions, F and G, minimizing them individually (no sharing)

$F = WY' + XY'Z$

$G = WX'Y' + X'Y + W'Y'Z$

(i) Implement them with a ROM

(ii) Implement them in the PLA

Solution: The given functions are first converted to canonical SOP form as follows

$$F = W\bar{Y}(X + \bar{X})(Z + \bar{Z}) + X\bar{Y}Z(W + \bar{W})$$

$$F = (W\bar{Y}X + W\bar{Y}\,\bar{X})(Z + \bar{Z}) + X\bar{Y}ZW + X\bar{Y}Z\bar{W}$$

$$F = WX\bar{Y}Z + WX\bar{Y}\bar{Z} + W\bar{X}\bar{Y}Z + W\bar{X}\bar{Y}\bar{Z} + WX\bar{Y}Z + \bar{W}X\bar{Y}Z$$

$$F = m_{13} + m_{12} + m_9 + m_8 + m_5$$

$$F = \sum_m(5,8,9,12,13)$$

Similarly by converting the function

$$G = WX'Y' + X'Y + W'Y'Z$$

$$G = W\bar{X}\bar{Y}(Z + \bar{Z}) + \bar{X}Y(W + \bar{W})(Z + \bar{Z}) + \bar{W}\bar{Y}Z(X + \bar{X})$$

$$G = W\bar{X}\bar{Y}Z + W\bar{X}\bar{Y}\bar{Z} + (\bar{X}YW + \bar{X}Y\bar{W})(Z + \bar{Z}) + \bar{W}\bar{Y}ZX + \bar{W}\bar{Y}Z\bar{X}$$

$$G = W\bar{X}\bar{Y}Z + W\bar{X}\bar{Y}\bar{Z} + W\bar{X}YZ + W\bar{X}Y\bar{Z} + \bar{W}\bar{X}YZ + \bar{W}\bar{X}Y\bar{Z} + \bar{W}X\bar{Y}Z + \bar{W}\bar{X}\bar{Y}Z$$

$$G = m_9 + m_8 + m_{11} + m_{10} + m_3 + m_2 + m_5 + m_1$$

$$G = \sum_m(1,2,3,5,8,9,10,11)$$

(i) Implementing the given function with a PROM

The given functions F and G have 4 inputs W, X, Y and Z. They generate $2^4 = 16$ minterms and hence the PROM have 16 AND gates. Since there are 2 outputs the PROM have two OR gates.

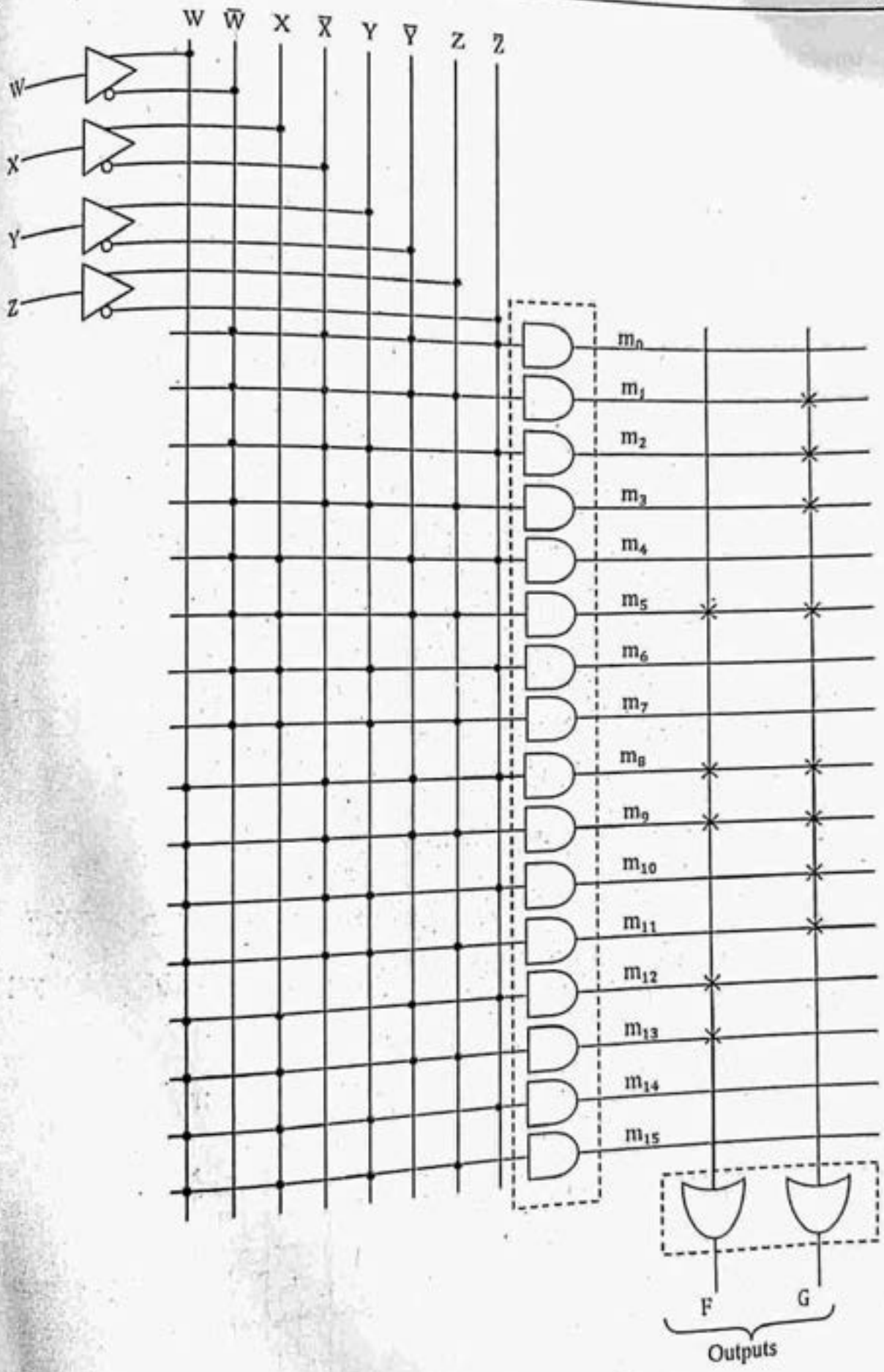


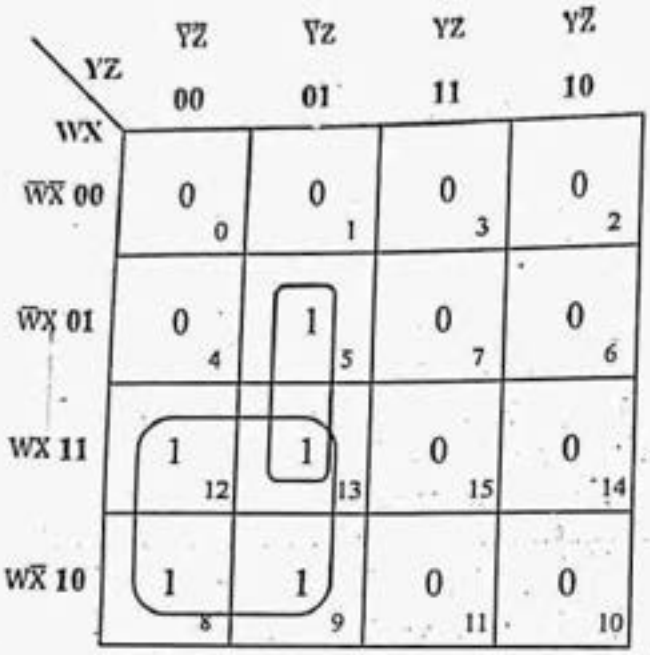
Figure 4.113 PROM

(ii) Implement them in the PLA.

We know that $F(W, X, Y, Z) = \sum_m(5,8,9,12,13)$

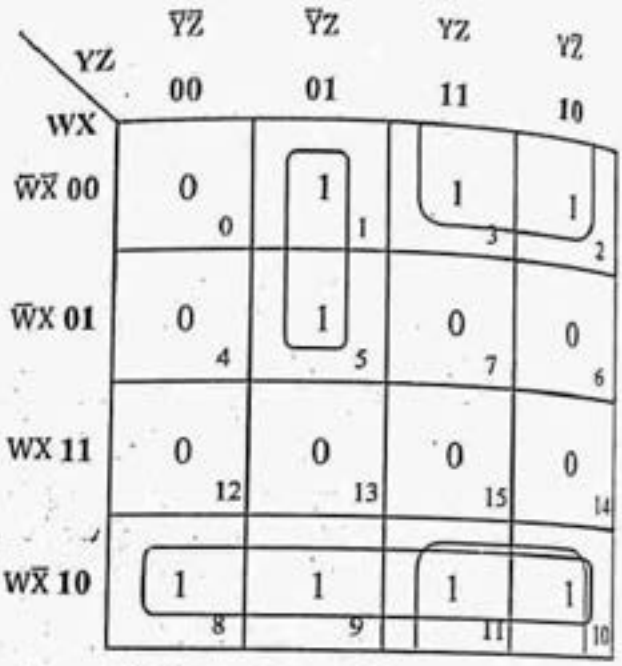
$G(W, X, Y, Z) = \sum_m(1,2,3,5,8,9,10,11)$

K-map for F



$F = W\bar{Y} + X\bar{Y}Z$

K-map for G



$G = W\bar{X} + \bar{W}\bar{Y}Z + \bar{X}Y$

Product terms	Inputs				Outputs	
	W	X	Y	Z	F	G
$W\bar{Y}$	1	-	0	-	1	-
$X\bar{Y}Z$	-	1	0	1	1	-
$W\bar{X}$	1	0	-	-	-	1
$\bar{W}\bar{Y}Z$	0	-	0	1	-	1
$\bar{X}Y$	-	0	1	-	-	1
					T	T

Table 4.72 PLA program table

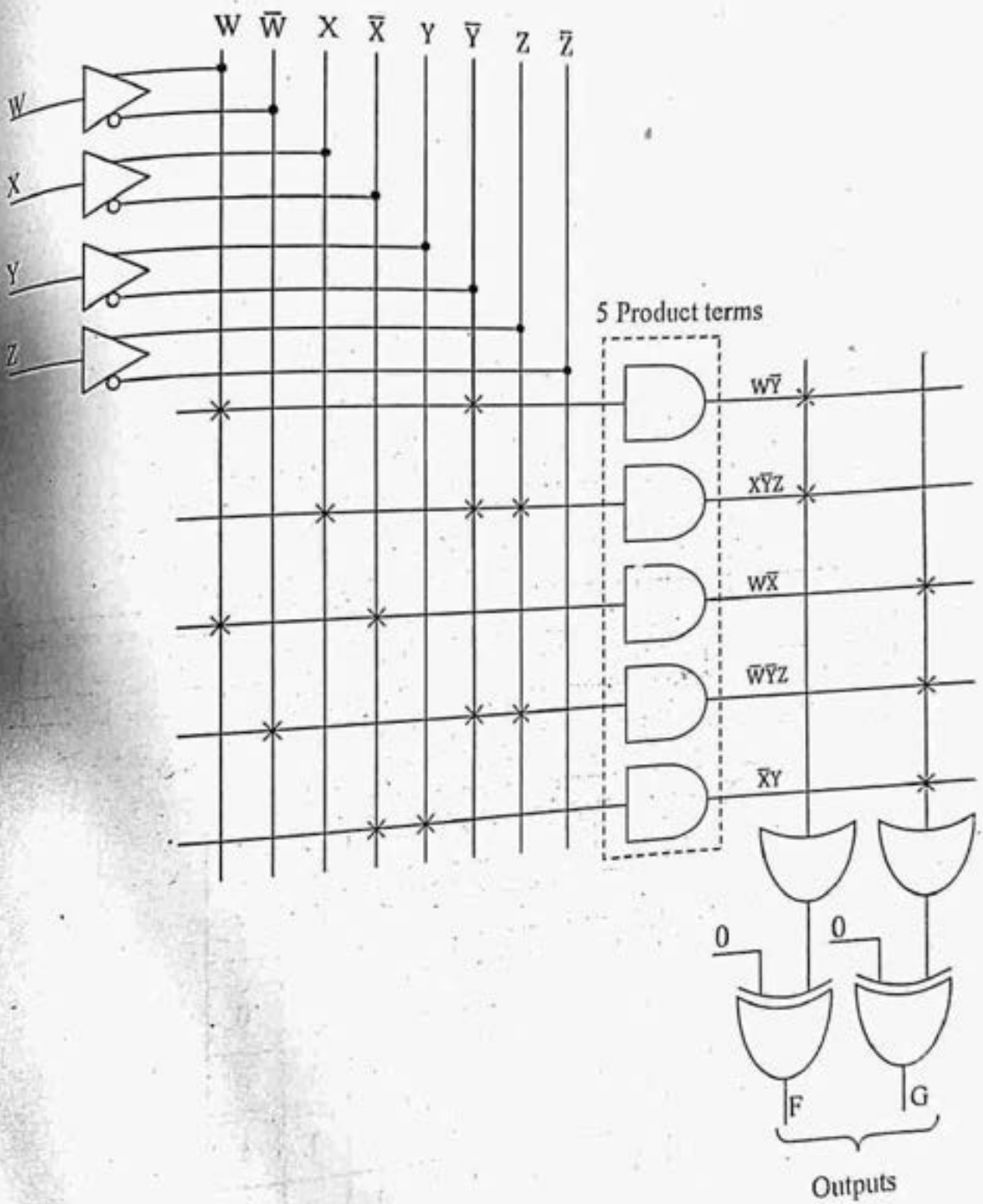


Figure 4.114 PLA

***Example 4.34:** Implement binary to Gray code converter using PROM devices.*

Solution :

The truth table of a binary to gray code converter is shown in table 4.73.

Input Binary code				Output Gray code			
b_3	b_2	b_1	b_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Table 4.73 Truth table for Binary to Gray code converter

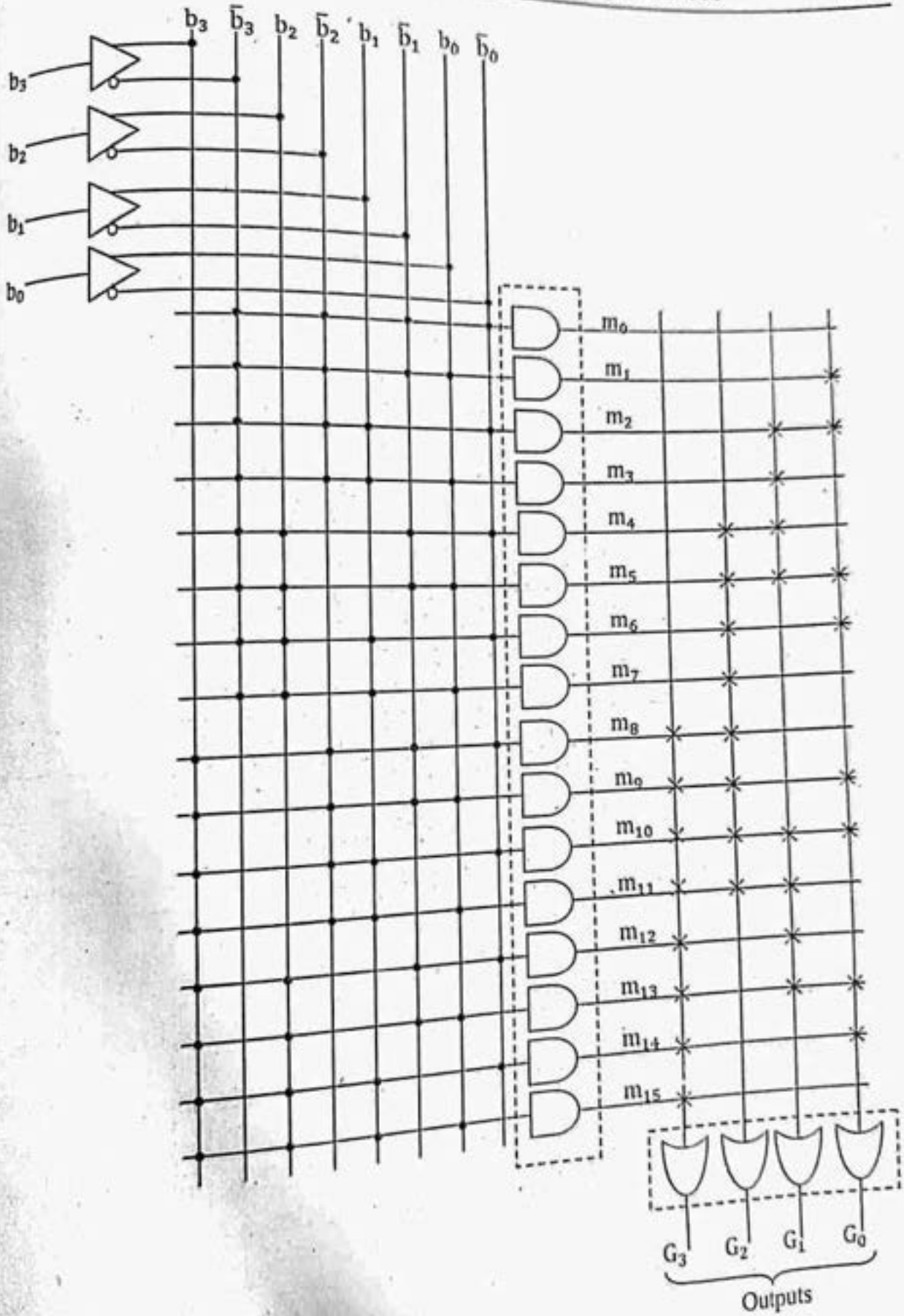


Figure 4.115 PROM

4.17 SEQUENTIAL PROGRAMMABLE DEVICES

The combinational PLD (PLA, PAL and PROM) consists of only gates. But most of the programmable devices are designed with flip-flops and gates, so it is necessary to include an external flip-flop with the combinational PLD when they are used in the design of programmable devices.

The sequential programmable devices are mainly classified into three types. They are,

1. Sequential programmable logic devices (SPLD)
2. Complex programmable logic devices (CPLD)
3. Field-programmable gate array (FPGA)

4.17.1 Sequential programmable logic devices (SPLD)

Sequential programmable logic devices (SPLD) are also known as simple PLD. The SPLD includes flip-flops, in addition to the AND-OR array, with in the integrated circuit chip. A PAL or PLA is modified by including a number of flip-flops connected to form a register.

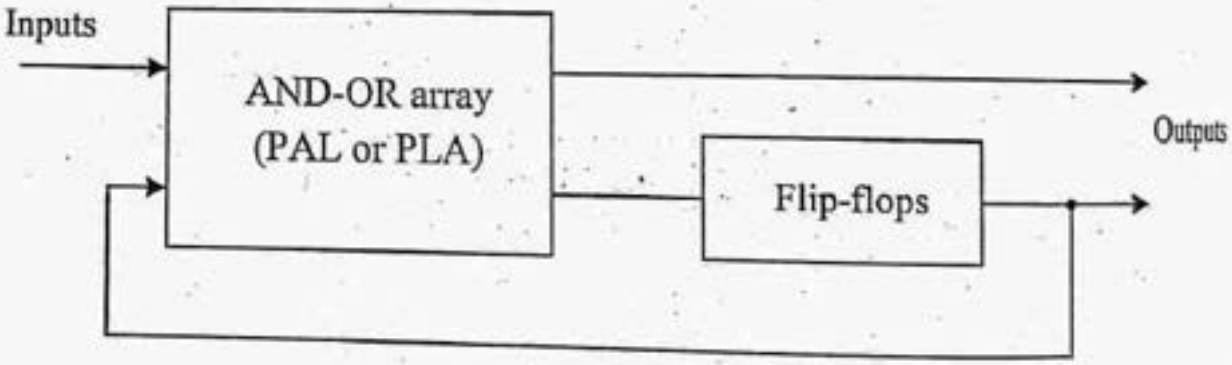


Figure 4.116 Block diagram of a sequential programmable Logic device

The outputs of sequential programmable logic devices can be taken from the OR gates of the PAL or PLA or from the outputs of the flip-flops. The flip-flops may be D Flip-flop or JK flip-flop. The flop-flops are flexible such that they can be programmed to operate as either JK flip-flop or D flip-flop.

Registered PAL configuration

The configuration mostly used in the SPLD is the combinational PAL together with D flip-flops. A PAL that includes flip-flops is referred to as a registered PAL. Each section of SPLD is called a macrocell.

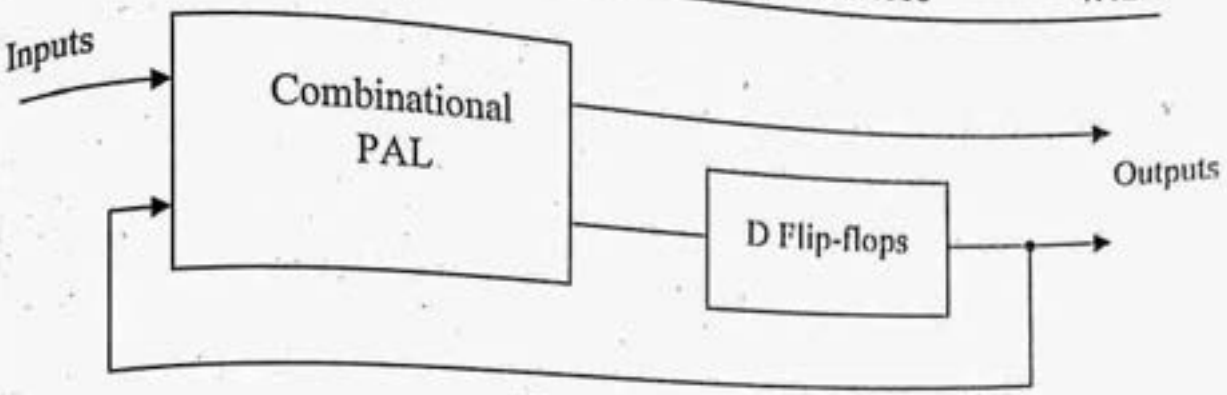


Figure 4.117 Block diagram of Registered PAL

A macrocell contains a sum of products logic function and an optional flip-flop.

Figure 4.118 shows the logic diagram of a basic macrocell. The output is driven by a edge triggered D flip-flop connected to a common clock inputs. The output of the flip-flop is connected to a tri-state buffer (or inverter) controlled by an output enable signal. The output of the flip-flop is fed back to the programmable AND gates to provide the present state condition for the sequential circuit. All the flip-flops are connected to the common CLK input, and all the tri-state buffers are controlled by the output enable signal.

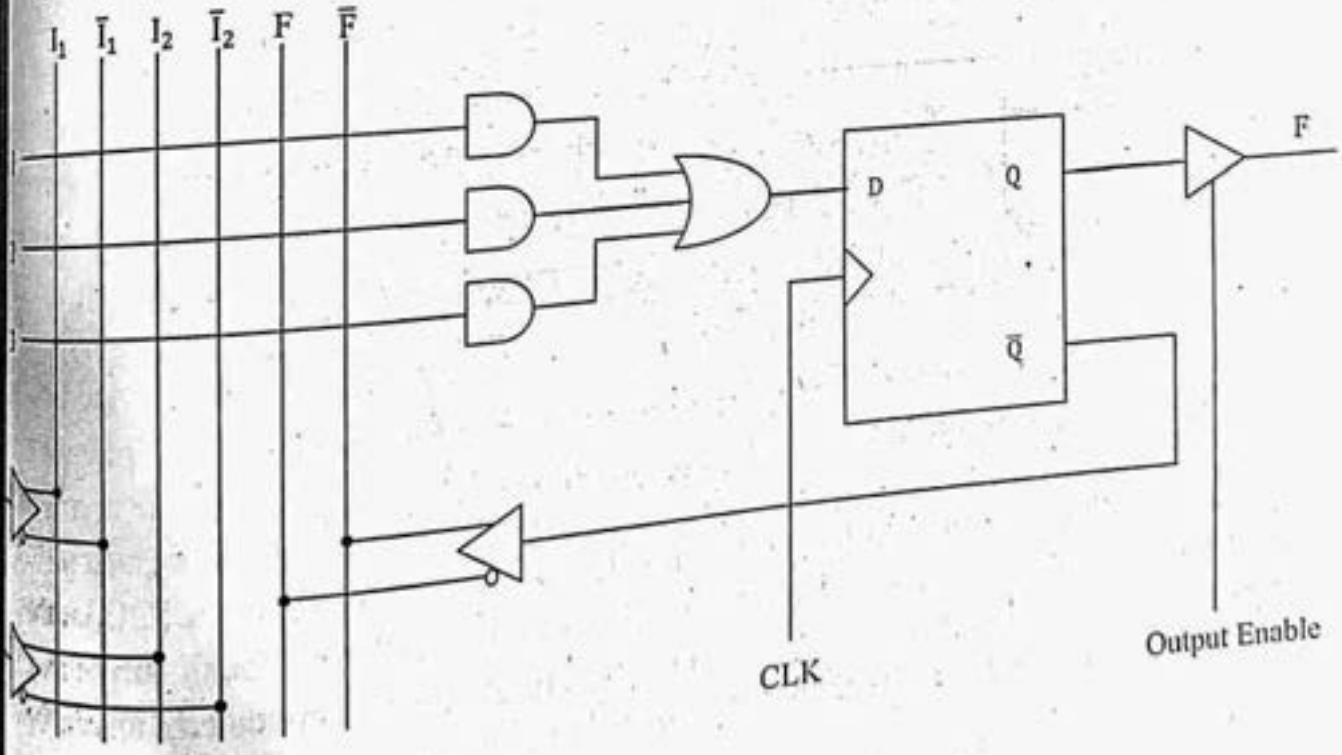


Figure 4.118 Basic macrocell logic

4.17.2 Complex Programmable logic devices (CPLD)

A CPLD consists of multiple PLDs interconnected through a programmable switch matrix as shown in figure 4.119

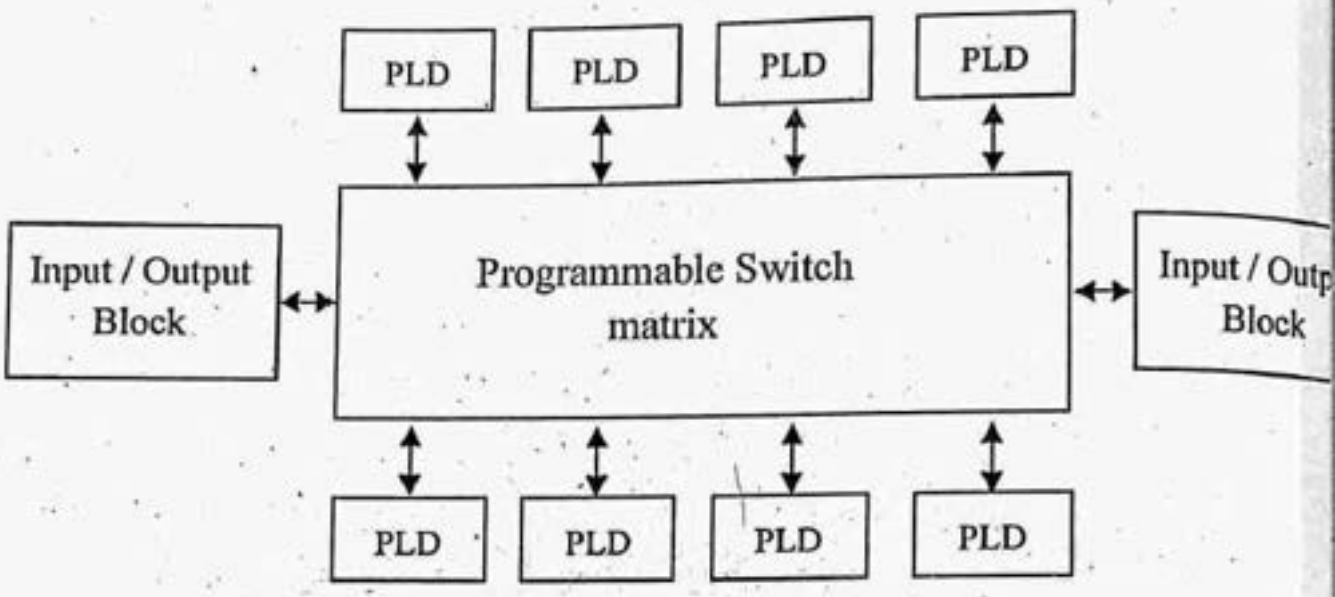


Figure 4.119 Block diagram of complex programmable Logic devices (CPLD)

The input output block (I/O block) provides the input output connection to the programmable switch matrix. Each I/O block is driven by a tri-state buffer and can be programmed to act as input or output pin. The programmable switch matrix receives inputs from the I/O block and directs them to the individual macrocells or PLDs. Similarly selected outputs from the macro cells or PLDs are sent to the outputs through programmable switch matrix to the I/O block. Each PLD typically contains 8 to 16 macrocells. In some cases the macrocell flip-flop is programmed in such a way that it can act as JK, D or T flip-flop.

4.17.3 Field Programmable gate Array (FPGA)

A field programmable gate array (FPGA) is a programmable device that can be programmed at the user's location. The word 'field' in the name refers to the ability of the gate arrays to be programmed for a specific function by the user instead of by the manufacturer of the devices. The word 'array' is used to indicate a series of rows or columns of gates that can be programmed by the end user. A FPGA consists of an array of hundred or thousands of Programmable logic blocks surrounded by programmable input and output blocks (IOB) and connected together via programmable interconnections known as switching matrix.

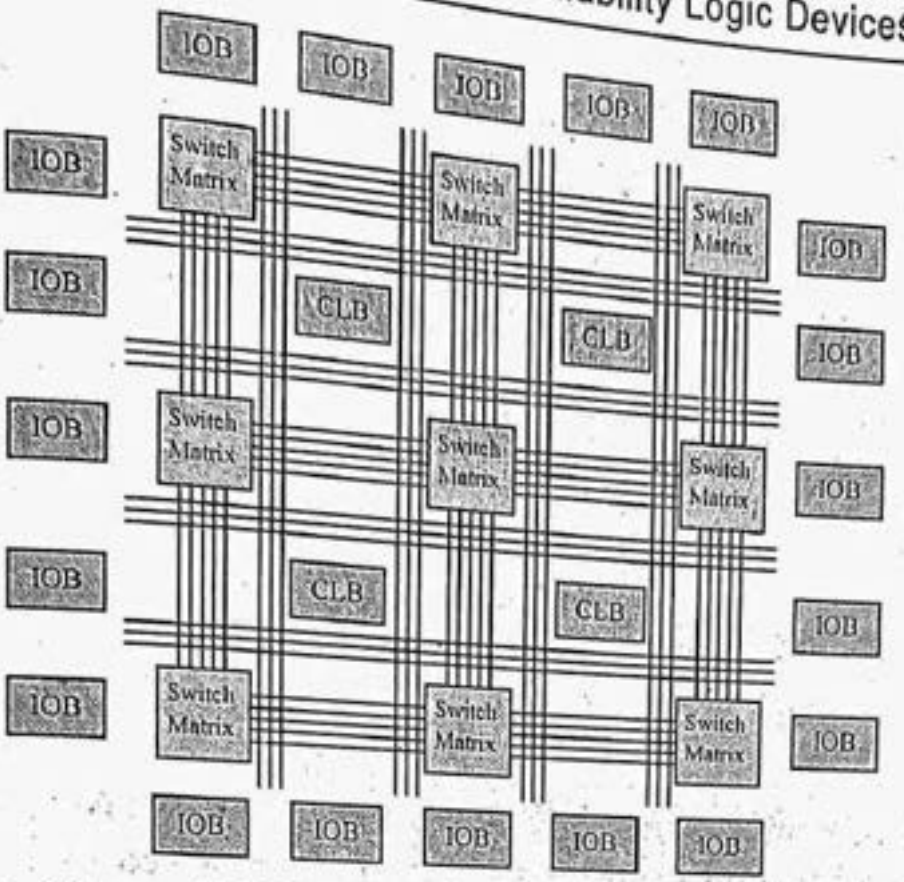


Figure 4.120 General FPGA chip architecture

The programmable logic block consists of lookup tables, multiplexers, gates and flip-flops. A lookup table is a truth table stored in a SRAM and provides the function for the logic block.

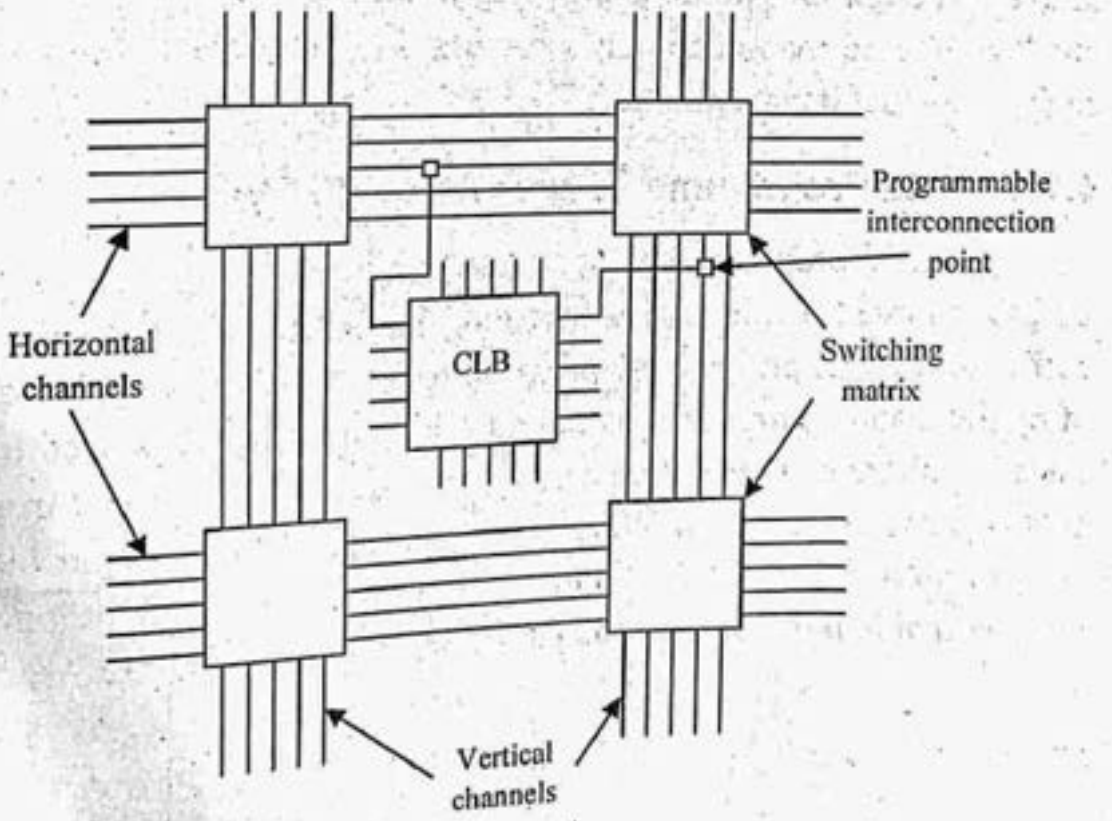


Figure 4.121 Switching matrices and programmable interconnect points

The programmable logic blocks of FPGAs are called configurable logic blocks (CLBs) also known as Logic elements (LE). The interconnection are made between the CLB using programmable interconnect known as switching matrix as shown in figure 4.122.

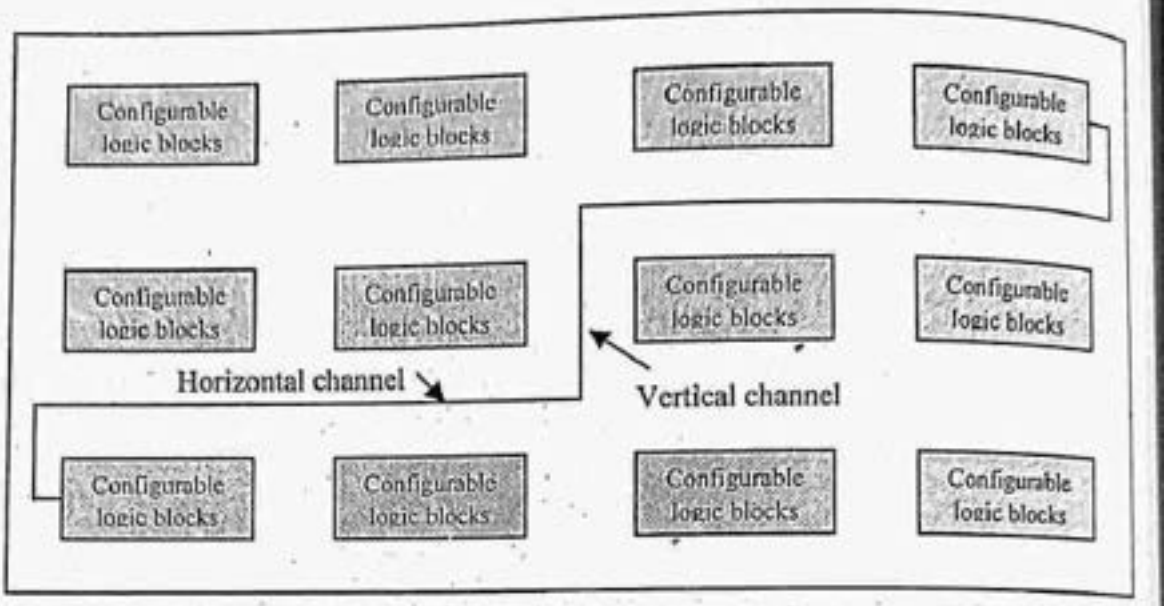


Figure 4.122 Interconnection between CLBs

FPGAs typically offer several types of interconnect depending on the distance between CLBs. Connections between CLB may requires complex paths, since the CLBs are arranged in two dimensional structure. We therefore need to make connections not just between CLBs and wires but also between the wires. Wires are typically organized in wiring channels or routing channels that run horizontally and vertically through the chip. The horizontal channel and vertical channel contains several wires and the programmer chooses which wire will be used in each channel to carry the signal.

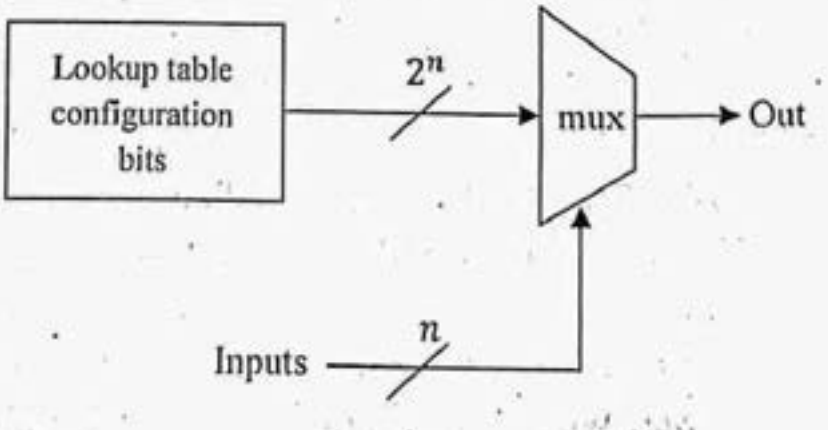


Figure 4.123 Configurable logic block (CLB)

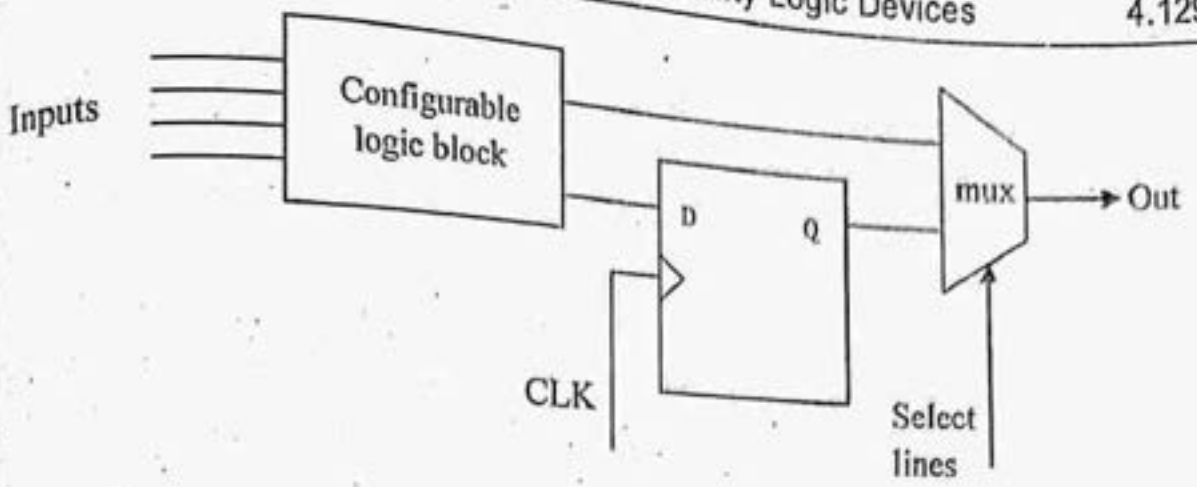


Figure 4.124 Configurable logic block with Flip-flop

The lookup table in a CLB is an SRAM that is used to implement a truth table. Each address in the SRAM represents a combination of inputs to the CLBs. A n-input function requires an SRAM with 2^n locations. Logic elements or CLBs generally contain registers, flip-flops and latches as well as combinational logic as shown in figure 4.124.

The FPGA can be easily reprogrammed. The I/O pins connect it to the outside world. Output pins provide buffers with sufficient drive to produce adequate signals on the pins. Each I/O block is driven by a tri-state buffer and can be programmed to act as input or output.

4.18 APPLICATION SPECIFIC INTEGRATED CIRCUITS (ASIC)

Application specific integrated circuit is an integrated circuit (IC) customized for a particular application rather than intended for general purpose use.

Examples of ASIC are an IC designed to run a digital voice recorder, a chip for a toy bear that talks, a chip designed to handle the interface between memory and a microprocessor etc.

4.18.1 Full custom ASIC

In a full custom ASIC, the designer should design some or all the logic cells, circuits or layout. We uses a full custom design if there are no suitable existing cell libraries available that can be used for the entire design. This might be because existing cell libraries are not fast enough or the logic cells are not small enough or consume too much power. We also uses a full custom design if the ASIC technology is new or so specialized that there are no existing cell libraries.

4.18.2 Standard-cell based ASIC

A cell based ASIC (Cell based IC or CBIC) uses predefined logic cells such as AND gate, OR gates, multiplexers, flip-flop etc. these predefined logic cells are known as standard cells. The standard cell areas may be used in combination with larger predefined cells such as microprocessor or microcontrollers known as megacells. Megacells are also called megafunctions, full-custom blocks, system level macros (SLMs), fixed blocks, cores or Functional standard Blocks (FSBs).

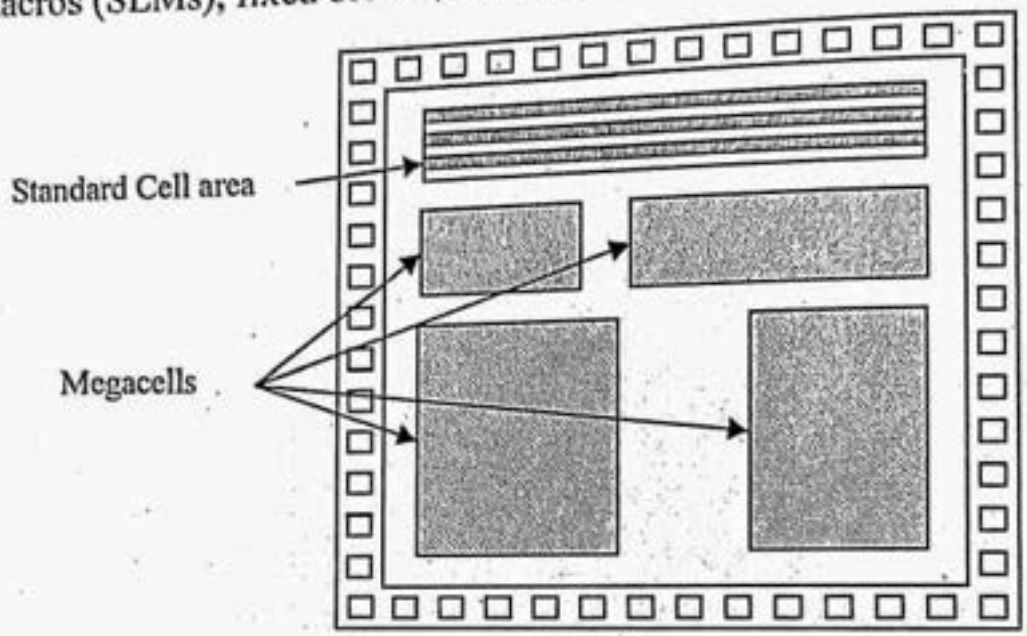


Figure 4.125 Standard-cell based ASIC

The figure 4.125 shows a standard-cell based ASIC. In this, the ASIC designer define only the placement of the standard cells and the interconnect in the cell based IC. However the standard cell can be placed anywhere on the silicon. The advantage of CBIC is that designers save time, money and reduce risk by using those predefined or predesigned standard cell libraries. The disadvantage of cell based IC is the time or expense of designing the standard cell library is high.

Each standard cell library is designed using full custom methods. The power and ground lines run horizontally on metal lines inside the cells. Usually the designer manually controls the number of power and ground lines.

Standard cell libraries may contain hundreds of different logic cells including combinational functions (NAND, NOR, AND, OR gates etc) with multiple inputs as well as latches and flip-flops with different combinations of reset, preset and clocking options.

4.18.3 Gate array based ASIC

In a gate array based ASIC the transistors are predefined on the silicon wafer. The predefined pattern of transistors on a gate array is the base array. The smallest element that is replicated to make the base array is the base cell. Only the top few layers of metal, which define the interconnect between transistors are defined by the designer. The designer chooses the predefined logic cells from a gate array library. The logic cells in the gate array-library are called macros.

The gate array base ASIC are classified into 3 types.

1. Channeled gate array
2. Channel less gate array
3. Structured gate array

4.18.3.1 Channeled gate array

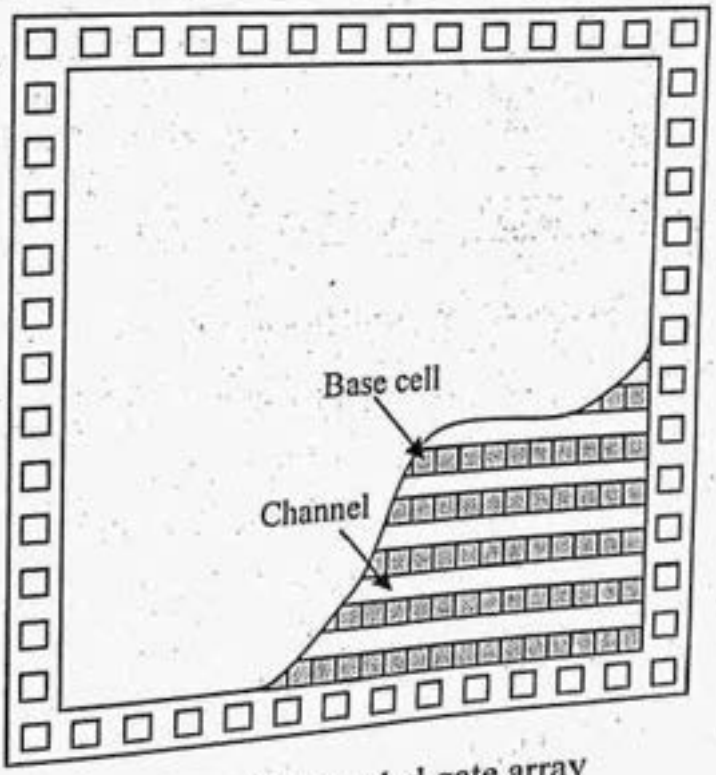


Figure 4.126 Channeled gate array

In a channeled gate array, rows of cells are separated by channels used for interconnect. The space for interconnect between the rows of cells are fixed in height as shown in figure 4.126.

4.18.3.2 Channel-less gate array

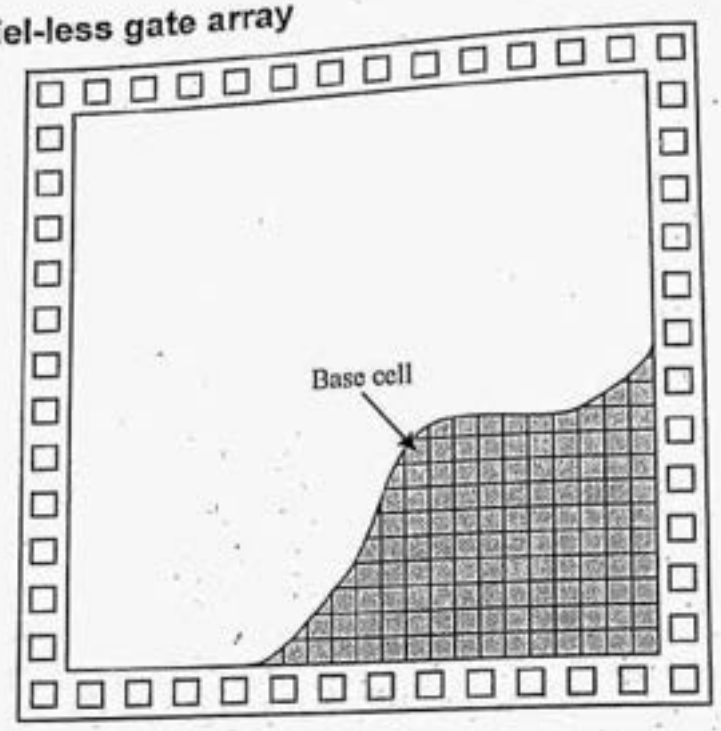


Figure 4.127 Channel-less gate array

In a channel-less gate array, there is no predefined areas (channel) for interconnection between the cells as shown in figure 4.127. Here the routing or interconnection is done on the top of the gate array device. The amount of logic cell that can be implemented in a given silicon area is higher for channel-less gate array than for channeled gate array.

4.18.3.3 Structured gate array

Structured gate array is also known as embedded gate array. It combines some of the features of CBIC (cell based IC) and gate array based ICs. In a structured gate array we set some of the IC area and dedicate it to a specific function as shown in figure 4.128. This embedded area either can contain a different base cell that is more suitable for building memory cells, or it can contain a complex circuit block such as microcontrollers.

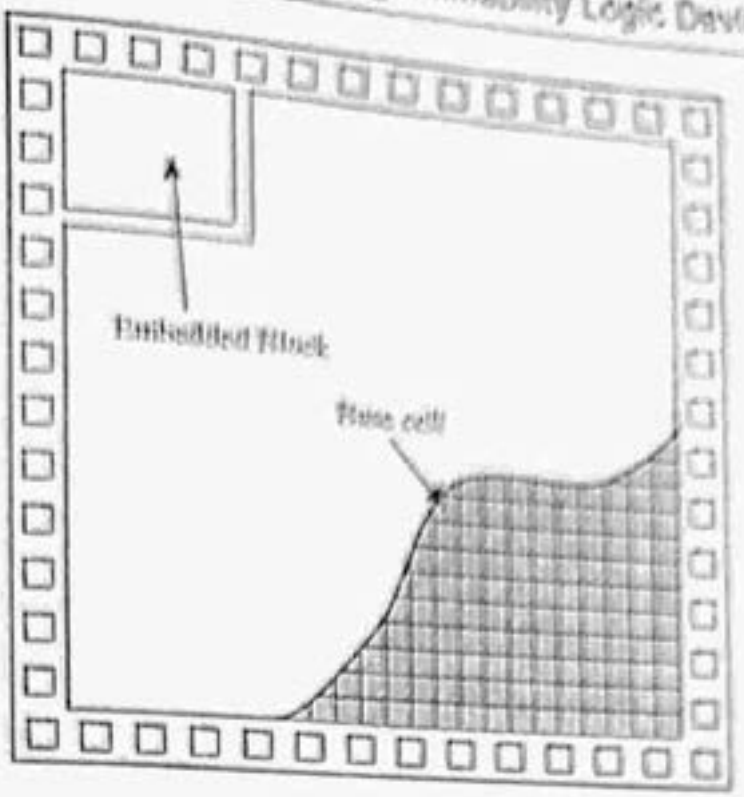


Figure 4.128 Structured gate array