S J P N TRUST
HIRASUGAR INSTITUTE OF TECHNOLOGY

SUB: ANALOG AND DIGITAL ELECTRONICS

CODE:18CS33

Module 3

Prof. N K Honnagoudar A.P

MODULE – 3

COMBINATIONAL LOGIC CIRCUITS

REVIEW OF COMBINATIONAL CIRCUIT DESIGN:

Steps involved in the design of a combinational switching circuit:

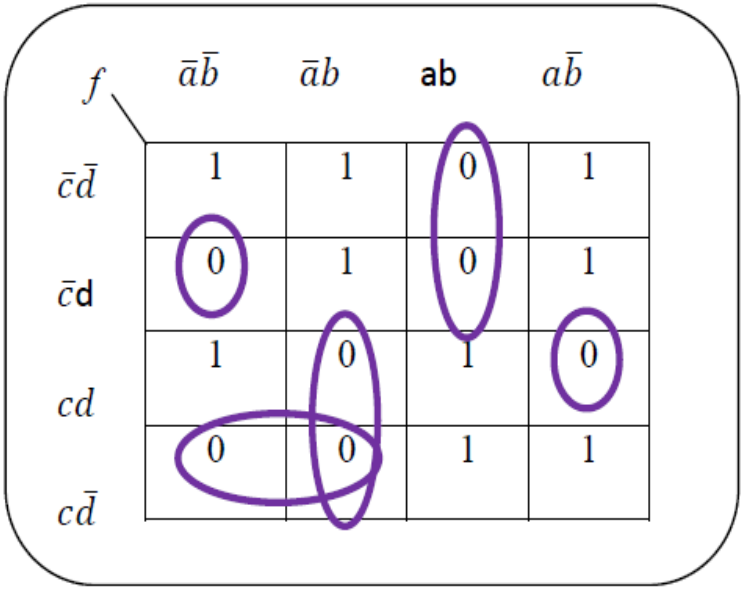
1. Set up a truth table which specifies the output(s) as a function of the input variables. If a given combination of values for the input variables can never occur at the circuit inputs, the corresponding output values are don't-cares.
2. Derive simplified algebraic expressions for the output functions using Karnaugh Maps, or Quine-McCluskey method, or any other similar procedure. The resulting algebraic expressions are then manipulated into the proper form, depending on the type of gates to be used in realizing the circuit.
3. When a circuit has two or more outputs, common terms in the output functions can often be used to reduce the total number of gates or gate inputs.
4. Minimum two-level AND-OR, or NAND-NAND circuits can be realized using the minimum sum-of-products. Minimum two-level OR-AND, or NOR-NOR circuits can be realized using the minimum product-of-sums.

DESIGN OF CIRCUITS WITH LIMITED GATE FAN-IN:

In practical logic design problems, the maximum number of inputs on each gate (or the fan-in) is limited. Depending on the type of gates used, this limit may be two, three, four, eight, or some other number. If a two-level realization of a circuit requires more gate inputs than allowed, factoring the logic expression to obtain a multi-level realization is necessary.

Example: *Realize* $f(a,b,c,d) = \Sigma m(0,3,4,5,8,9,10,14,15)$ *using three input NOR gates.*

Solution:



The product-of-sum equation is:

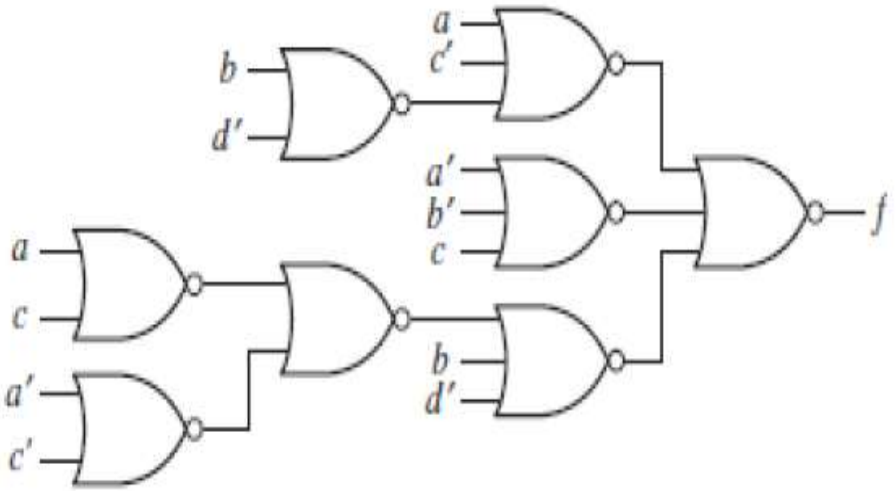
$$f = (a' + b' + c)(a + b' + c')(a + c' + d)(a + b + c + d')(a' + b + c' + d')$$

As can be seen from the preceding expression, a two-level realization requires three-three input gates, two four-input gates and one five-input gate. The expression for f' is factored to reduce the maximum number of gate inputs to three and, then, it is complemented.

Or $f' = abc' + a'bc + a'cd' + a'b'c'd + ab'cd$

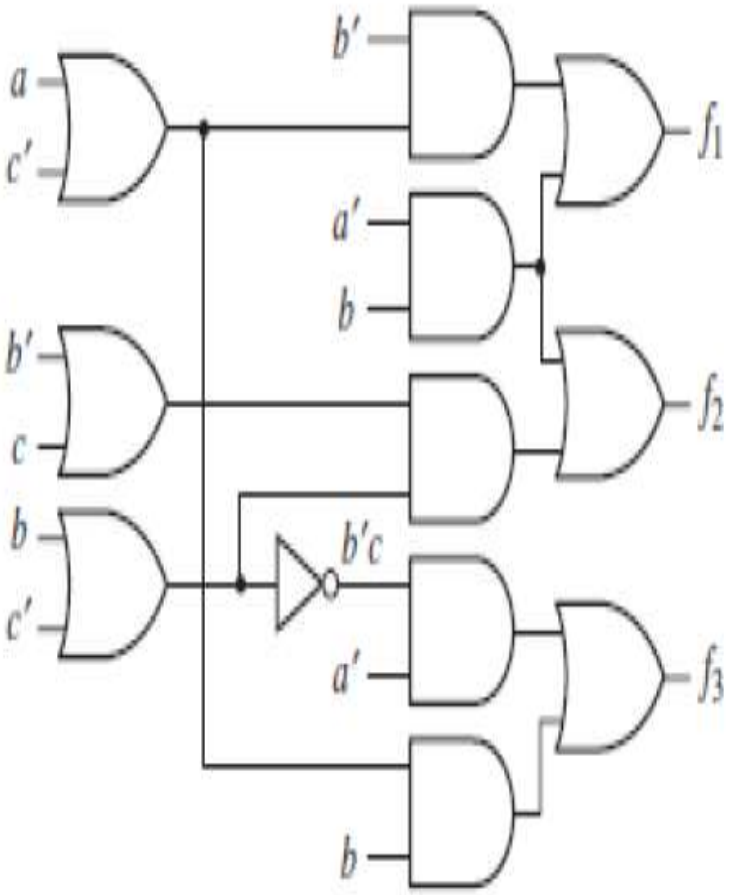
i.e., $f' = abc' + a'c(b + d') + b'd(a'c' + ac)$

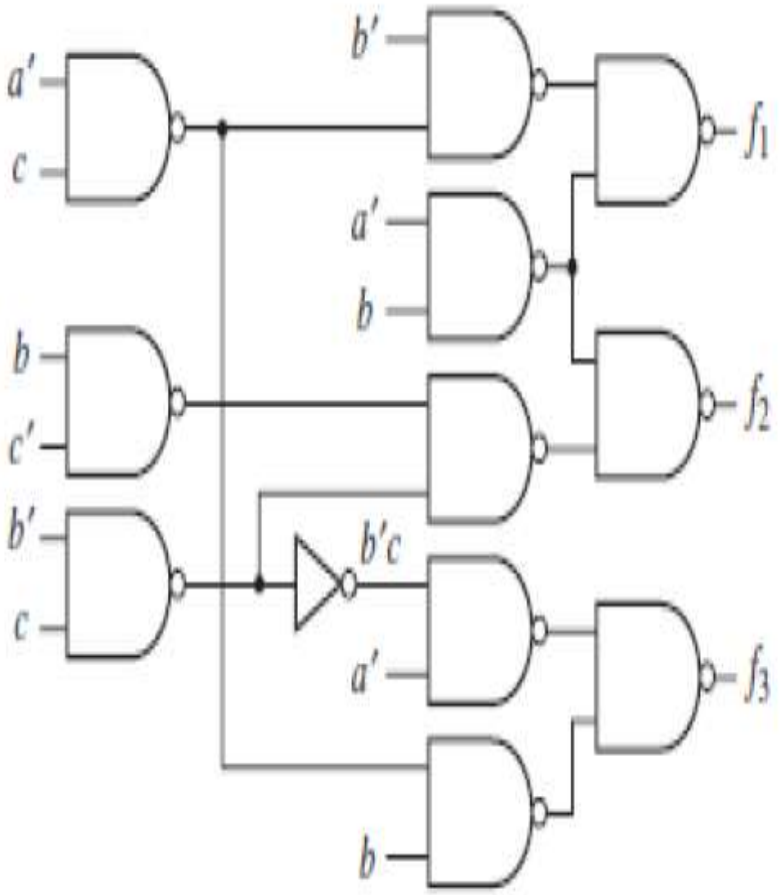
Or $f = [(a' + b' + c)][(a + c') + (b'd)][(b + d') + (a + c)(a' + c')]$



Example: Realize the following functions using only two-input NAND gates and inverters

The following Figures show the resulting circuits:

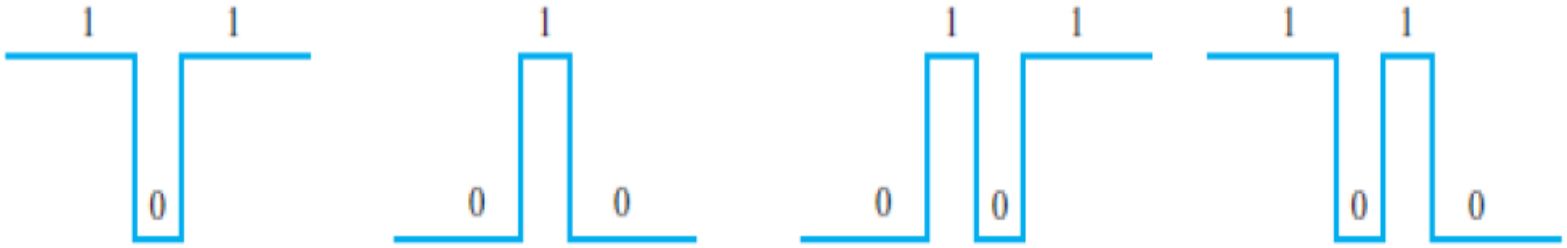




When the input to a combinational circuit changes, unwanted switching transients may appear in the output. These transients occur when different paths from input to output have different propagation delays.

- If, in response to any single input change and for some combination of propagation delays, a circuit output may momentarily go to 0 when it should remain a constant 1, we say that the circuit has a ***static 1-hazard***.
- Similarly, if the output may momentarily go to 1 when it should remain a 0, we say that the circuit has a ***static 0-hazard***.
- If, when the output is supposed to change from 0 to 1 (or 1 to 0), the output may change three or more times, we say that the circuit has a ***dynamic hazard***.

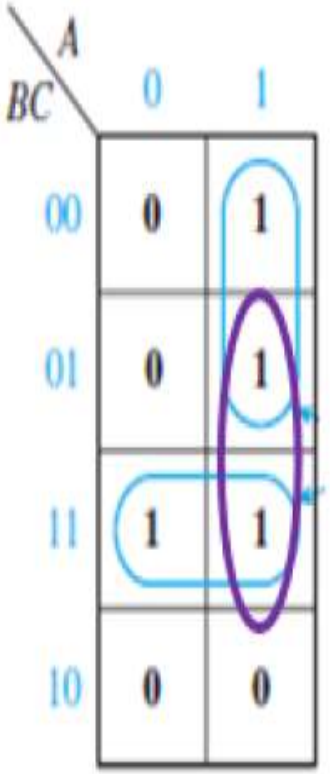
The following Figure shows possible outputs from a circuit with hazards:

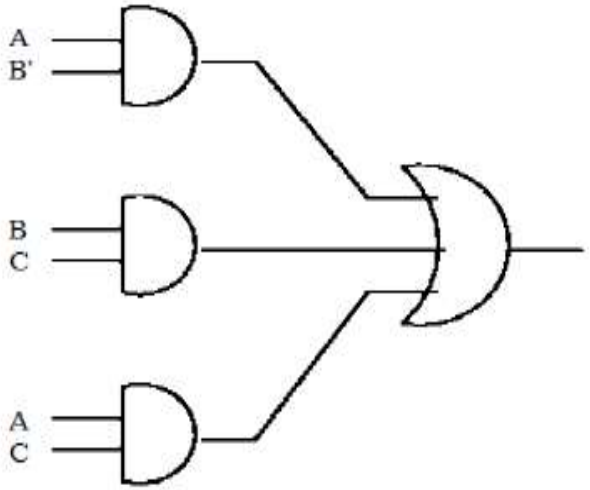


(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazards

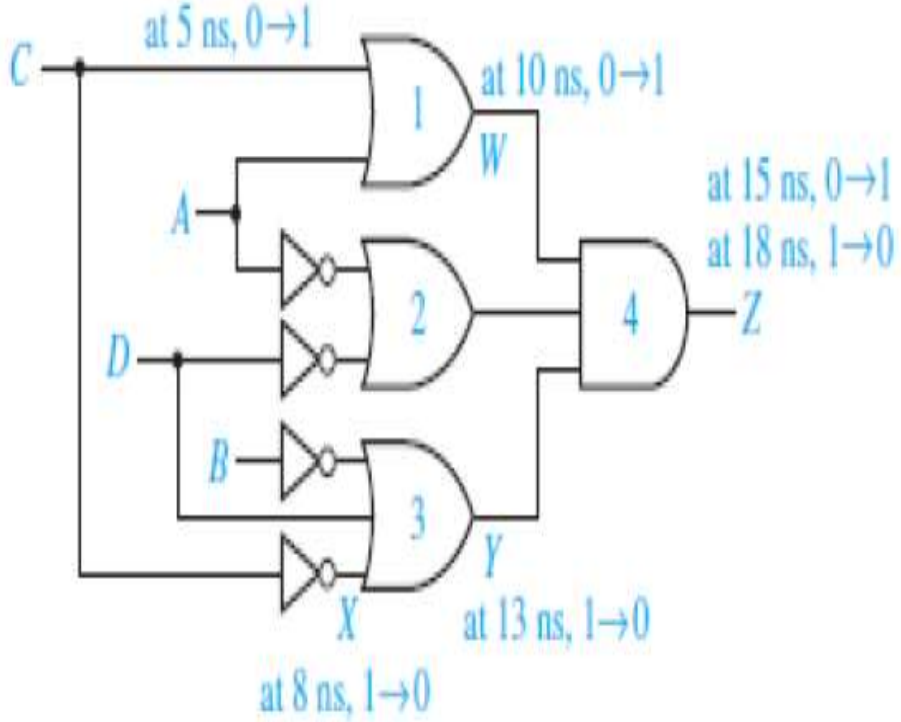
the other $n - 1$ variables are held constant.

To Eliminate Static 1 Hazard: If we add a loop to the map of above Figure and, then, add the corresponding gate to the circuit (as shown in the following Figure), this eliminates the hazard. The term AC remains *1* while B is changing, so no glitch can appear in the output. Note that F is no longer a minimum sum of products.

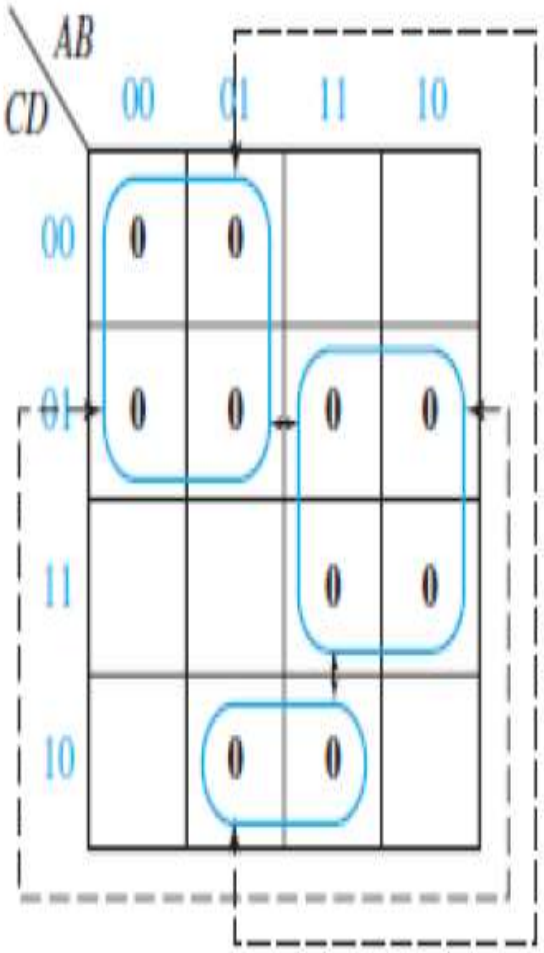




Detection of Static 0 Hazard: The following Figure shows a circuit with several 0-hazards. The product-



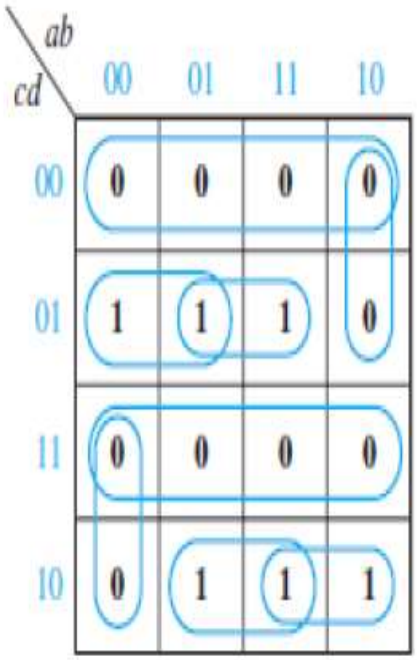
(a) Circuit with a static 0-hazard



(b) Karnaugh map for circuit of (a)

Dynamic Hazard: A dynamic hazard exists if there is a term of the form $xx'\alpha$ and two conditions are satisfied: (1) There are adjacent input combinations on the Karnaugh map differing in the value of x, with $\alpha = 1$ and with opposite function values, and (2) for these input combinations the change in x propagates over at least three paths through the circuit.

Consider the following expression and its Karnaugh map;



$$f = a'c'd + bc'd + bcd' + acd'$$

$$f = (c' + ad' + bd')(c + a'd + bd)$$

$$= cc' + acd' + bcd' + a'c'd + aa'dd' + a'bdd' + bc'd + abdd' + bdd'$$

$$= cc' + acd' + bcd' + a'c'd + aa'dd' + bc'd + bdd'$$

SIMULATION AND TESTING OF LOGIC CIRCUITS:

An important part of the logic design process is verifying that the final design is correct and debugging the design if necessary. Logic circuits may be tested either by actually building them or by simulating them on a computer. Simulation is generally easier, faster, and more economical. As logic circuits become more and more complex, it is very important to simulate a design before actually building it. This is particularly true when the design is built in integrated circuit form, because fabricating an integrated circuit may take a long time and correcting errors may be very expensive.

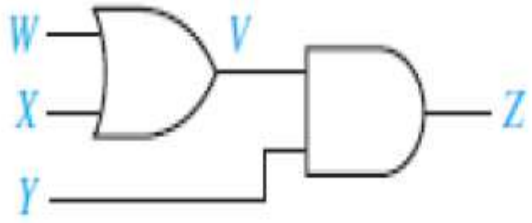
Simulation is done for following reasons: (1) Verification that the design is logically correct, (2) Verification that the timing of the logic signals is correct, and (3) Simulation of faulty components in the circuit as an aid to finding tests for the circuit.

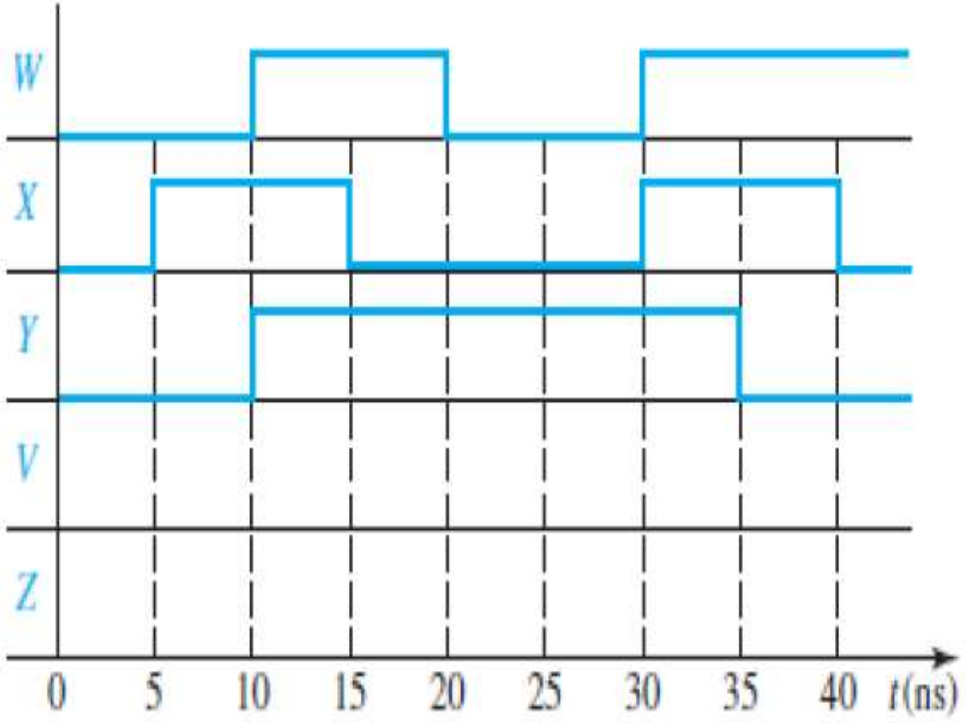
1. The output of gate 7 (F) is wrong, but this wrong output is consistent with the inputs to gate 7, that is, 1 + 0 = 1.Therefore, one of the inputs to gate 7 must be wrong.
2. In order for gate 7 to have the correct output (F = 0), both inputs must be 0. Therefore, the output of gate 5 is wrong. However, the output of gate 5 is consistent with its inputs because 1.1.1 = 1. Therefore, one of the inputs to gate 5 must be wrong.
3. Either the output of gate 3 is wrong, or the A or B input to gate 5 is wrong. Because $C'D + CD' = 0$, the output of gate 3 is wrong.
4. The output of gate 3 is not consistent with the outputs of gates 1 and 2 because $0 + 0 \neq 1$. Therefore, either one of the inputs to gate 3 is connected wrong, or gate 3 is defective, or one of the input connections to gate 3 is defective.

ANALOG AND DIGITAL ELECTRONICS

Problem: *Complete the timing diagram for the given circuit. Assume that both gates have a propagation delay of 5 ns.*

Solution:





Problem: *Consider the logic function: F (A, B, C, D) = ∑m (0, 4, 5, 10, 11, 13, 14, 15).*

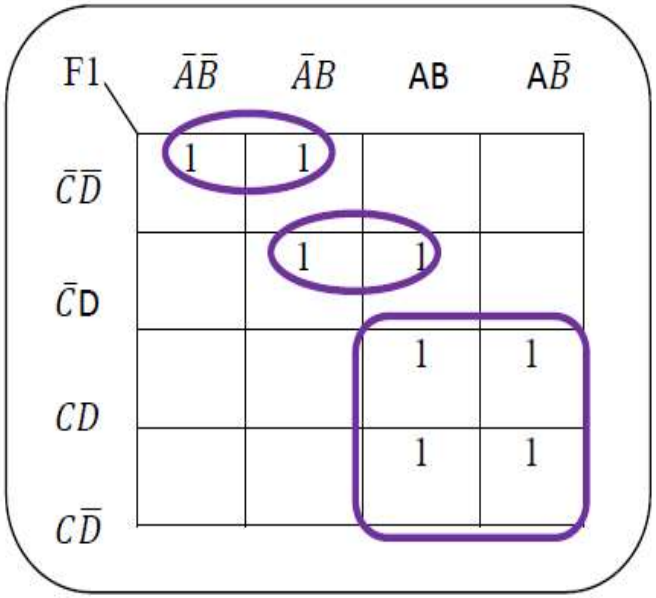
(a) Find two different minimum circuits which implement F using AND and OR Gates. Identify two hazards in each circuit.

(b) Find an AND-OR circuit for F which has no hazard.

(c) Find an OR-AND circuit for F which has no hazard.

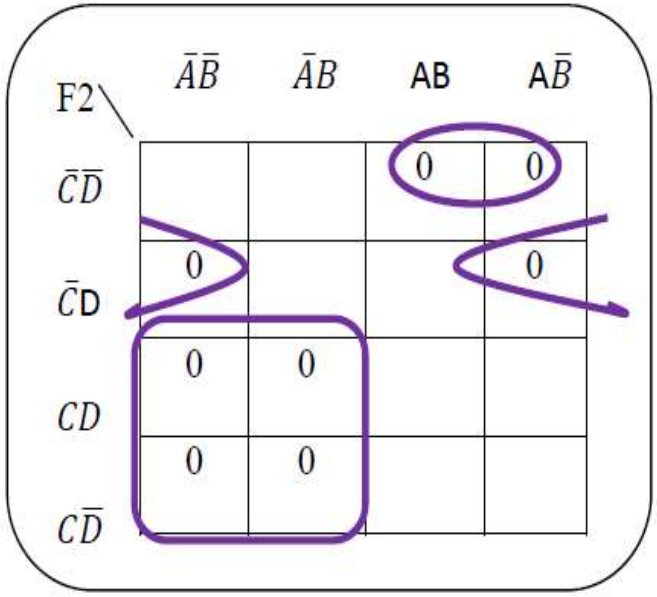
Solution: Given, F (A, B, C, D) = ∑m (0, 4, 5, 10, 11, 13, 14, 15)

(a) K-Map1:



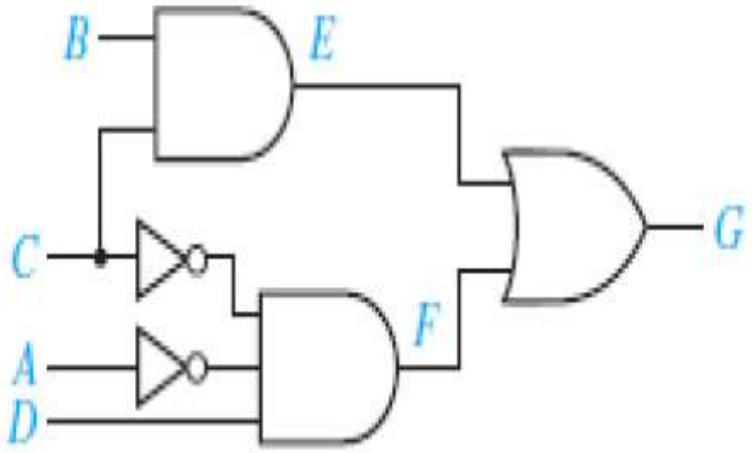
F1 =

K-Map2:



F2 =

Problem: *For the following circuit:*

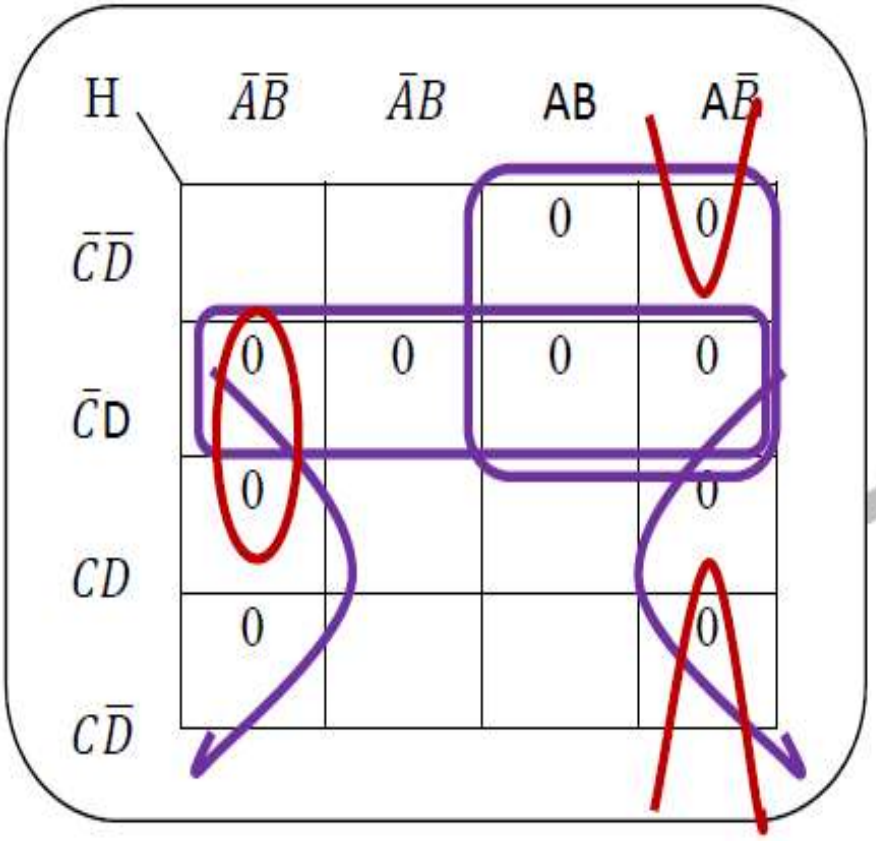


(a) *Assume that the inverters have a delay of 1 ns and the other gates have a delay of 2 ns. Initially, A = 0 and B = C = D = 1, and C changes to 0 at time = 2 ns. Draw a timing diagram and identify the transient that occurs.*

(b) *Modify the circuit to eliminate the hazard.*

(b) Expression for $H(A,B,C,D) = (A' + C)(C + D')(B + C')$

K-Map (OR-AND):

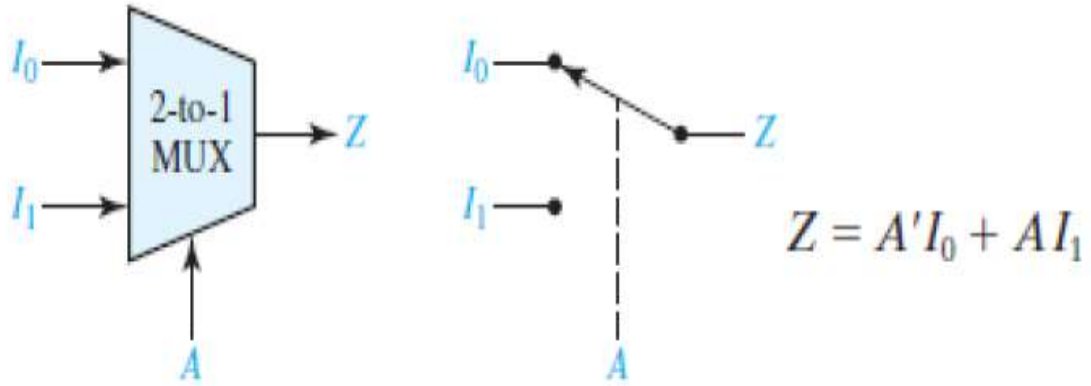


H =

Hazard free Circuit Diagram:

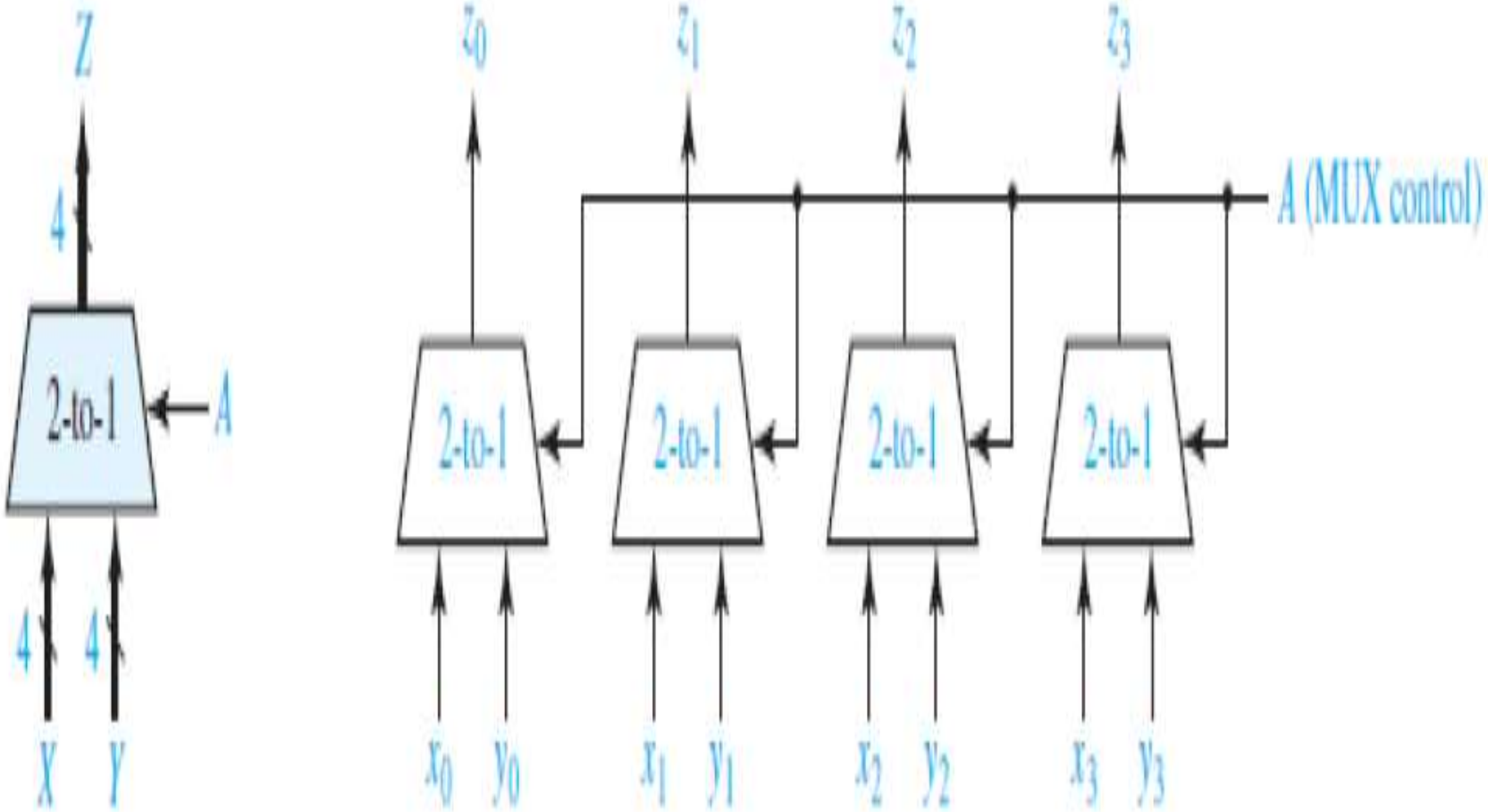
MULTIPLEXERS:

A *multiplexer* (or *data selector*, abbreviated as *MUX*) has a group of data inputs and a group of control inputs. The control inputs are used to select one of the data inputs and connect it to the output terminal. The following Figure shows a 2-to-1 multiplexer.



When the control input A is 0, the switch is in the upper position and the MUX output is $Z = I_0$; when A is 1, the switch is in the lower position and the MUX output is $Z = I_1$. In other words, a MUX acts like a switch that selects one of the data inputs (I_0 or I_1) and transmits it to the output. The logic equation for the 2-to-1 MUX is therefore: $Z = A' I_0 + AI_1$

The following Figure shows diagrams for a 4-to-1 multiplexer, 8-to-1 multiplexer, and 2^n-to-1

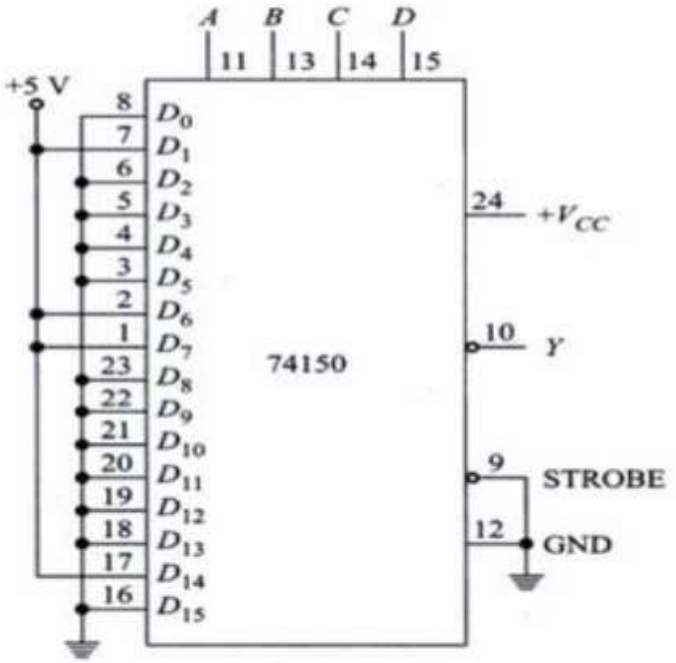

Z
4
2-to-1
A
4
4
X
Y
z_0
z_1
z_2
z_3
A (MUX control)
2-to-1
2-to-1
2-to-1
2-to-1
x_0
y_0
x_1
y_1
x_2
y_2
x_3
y_3

Multiplexer Logic: A digital design usually begins with a truth table. The problem is to come up with a logic circuit that has the same truth table. We have two standard methods for implementing a truth table – the SOP and the POS solution. The third method is the *multiplexer solution*.

Problem: *Implement Y (A, B, C, D) = ∑m (0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15) using 16-to-1 multiplexer (IC 74150) & 8-to-1 multiplexer.*

Solution: Notice that, output is an active low signal in IC 74150.

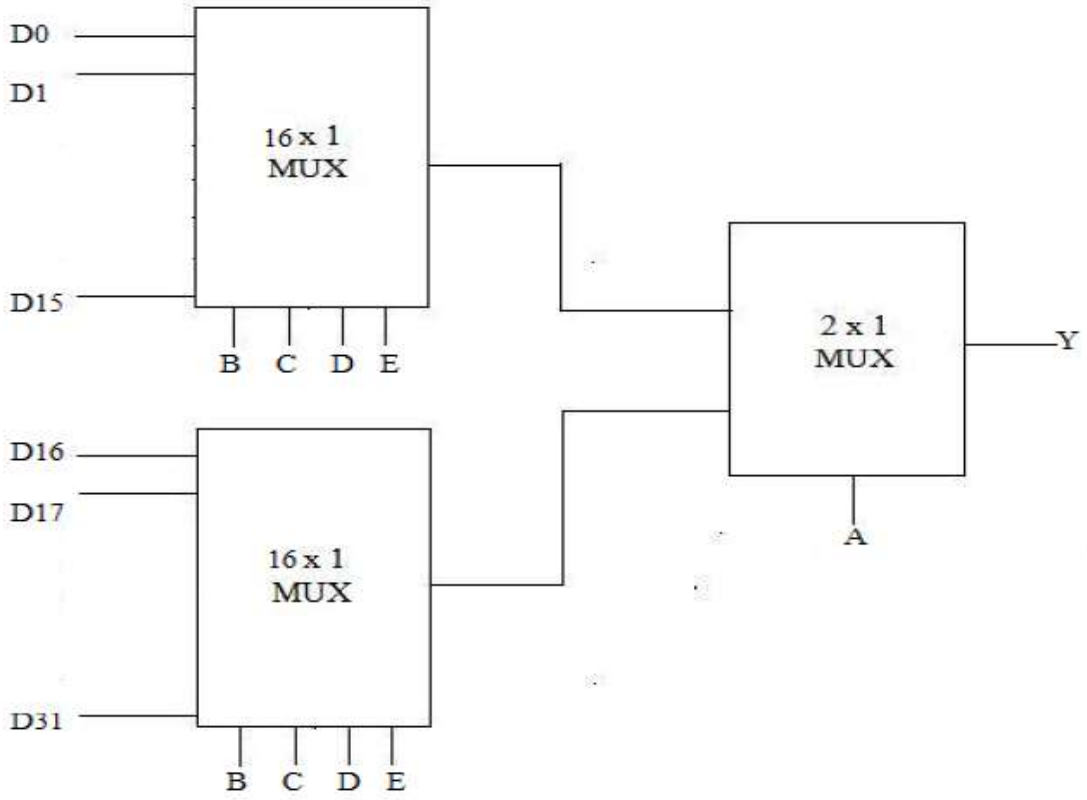
A	B	C	D	Y	8-to-1 MUX Data Inputs
0	0	0	0	1	$\overline{D}$
0	0	0	1	0	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	1	1
0	1	0	1	1	
0	1	1	0	0	0
0	1	1	1	0	
1	0	0	0	1	1
1	0	0	1	1	
1	0	1	0	1	1
1	0	1	1	1	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	0	D
1	1	1	1	1	



We follow a procedure that is similar to the one that we adopted in Entered Variable Map method to implement Y using 8-to-1 MUX.

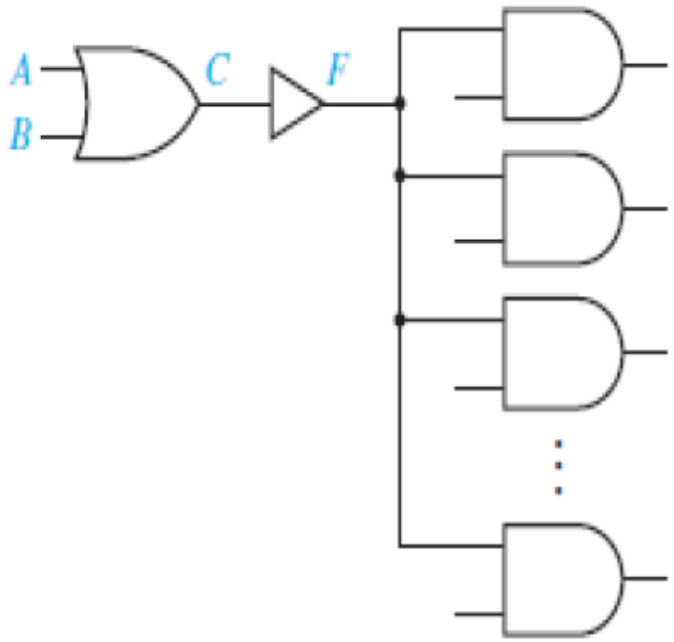
Problem: *Design a 32-to-1 multiplexer using two 16-to-1 multiplexer and one 2-to-1 multiplexer.*

Solution: The circuit diagram is shown in the following Fig. A 32-to-1 multiplexer required 5 ($\log_2 32$) select lines (say, ABCDE). The lower four select lines (BCDE) chose 16-to-1 multiplexer outputs. The 2-to-1 multiplexer chooses one of the output of two 16-to-1 multiplexers, depending on the 5th select line (A).

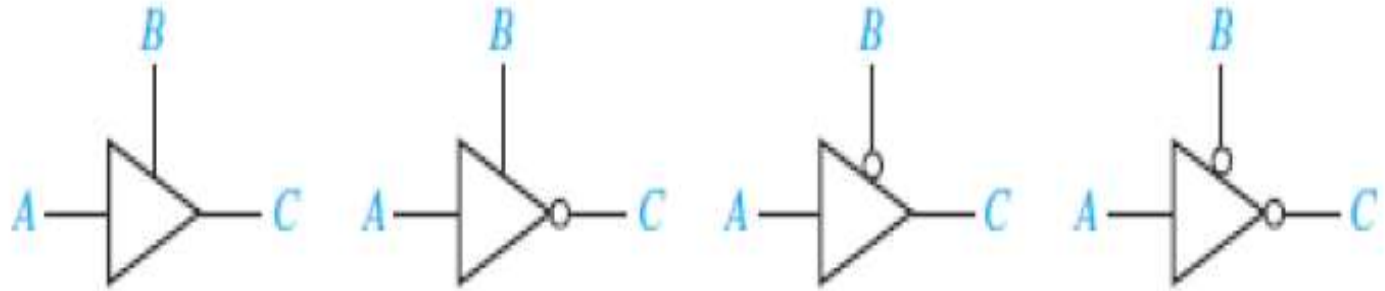


THREE STATE BUFFERS:

A gate output can only be connected to a limited number of other device inputs without degrading the performance of a digital system. A simple buffer may be used to increase the driving capability of a gate output. The following Figure shows a buffer connected between a gate output and several gate inputs. Because no bubble is present at the buffer output, this is a non-inverting buffer, and the logic values of the buffer input and output are the same, that is, $F = C$.



ANALOG AND DIGITAL ELECTRONICS



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

(b)

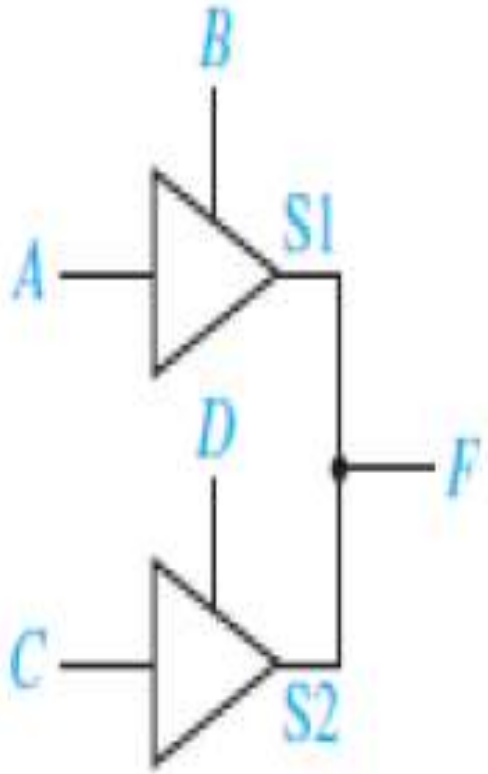
B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

(c)

B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

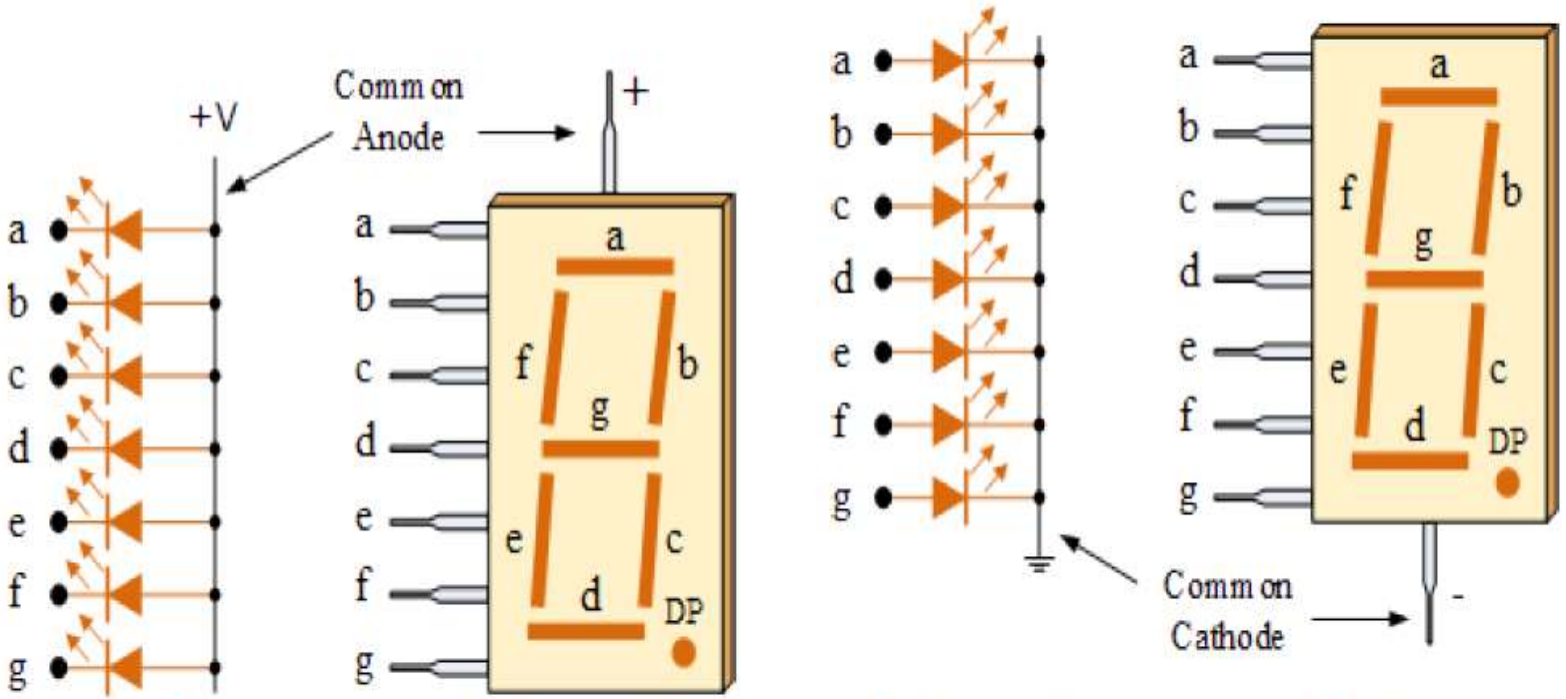
(d)

ANALOG AND DIGITAL ELECTRONICS



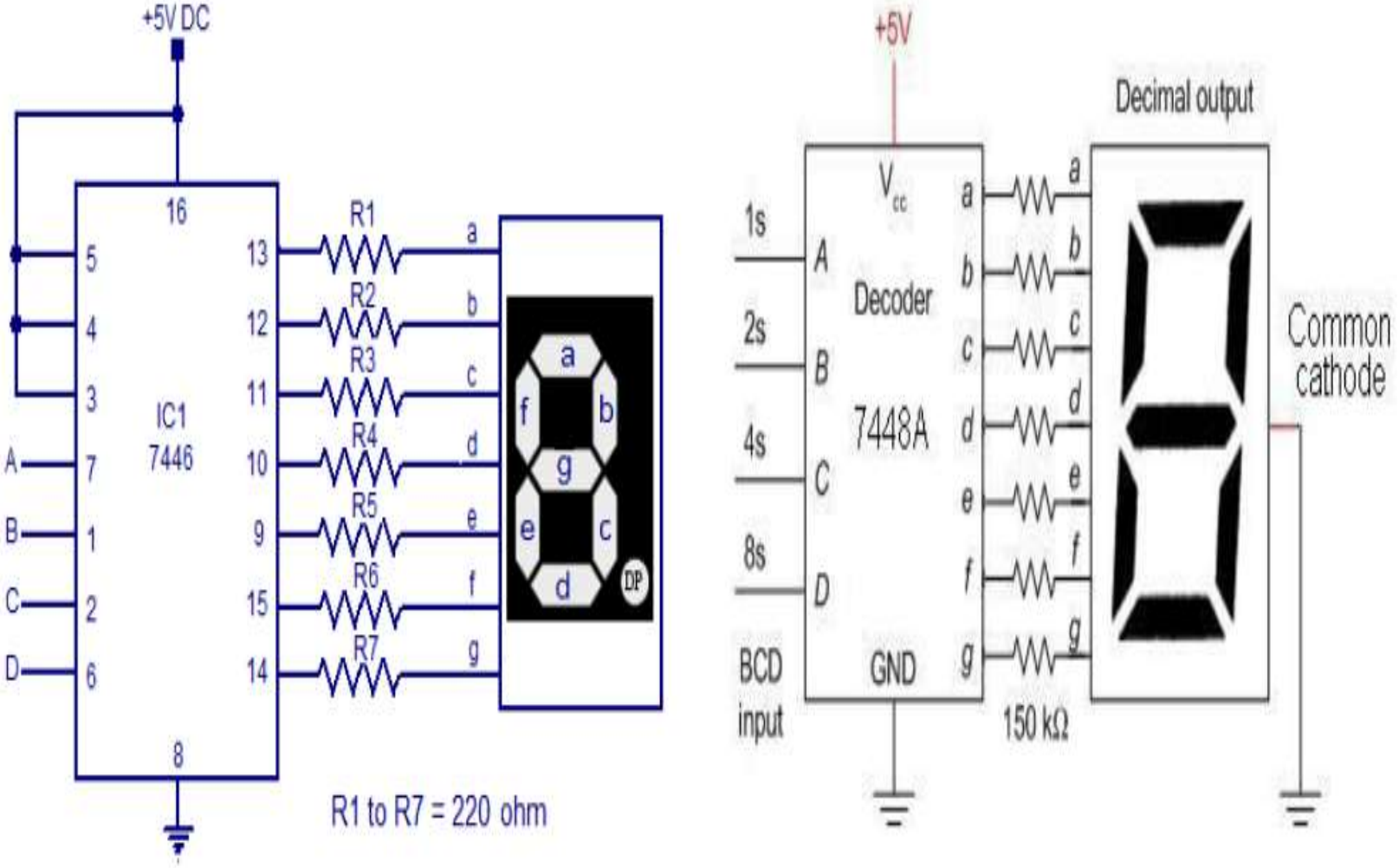
	S2			
S1	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Seven-Segment Decoders: The following Fig shows a *seven-segment indicator*, i.e. seven LEDs labeled *a* through *g* (actually, eight LEDs labeled through *a* through *h*). By forward biasing the LEDs, we can display the digits 0 through 9. For example, to display the digit 0, we need to light-up the segments *a*, *b*, *c*, *d*, *e*, and *f*. Similarly, to light-up the digit 5, we need segments *a*, *c*, *d*, *f*, and *g*.



Seven-Segment Indicator: Common Anode Type & Common Cathode Type

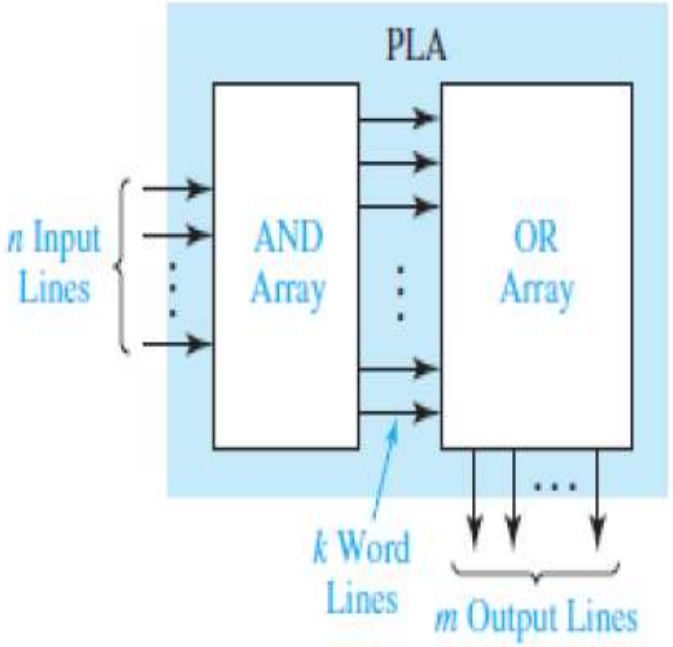
Seven-segment indicators may be *common-anode* type; where all anodes are connected together (as

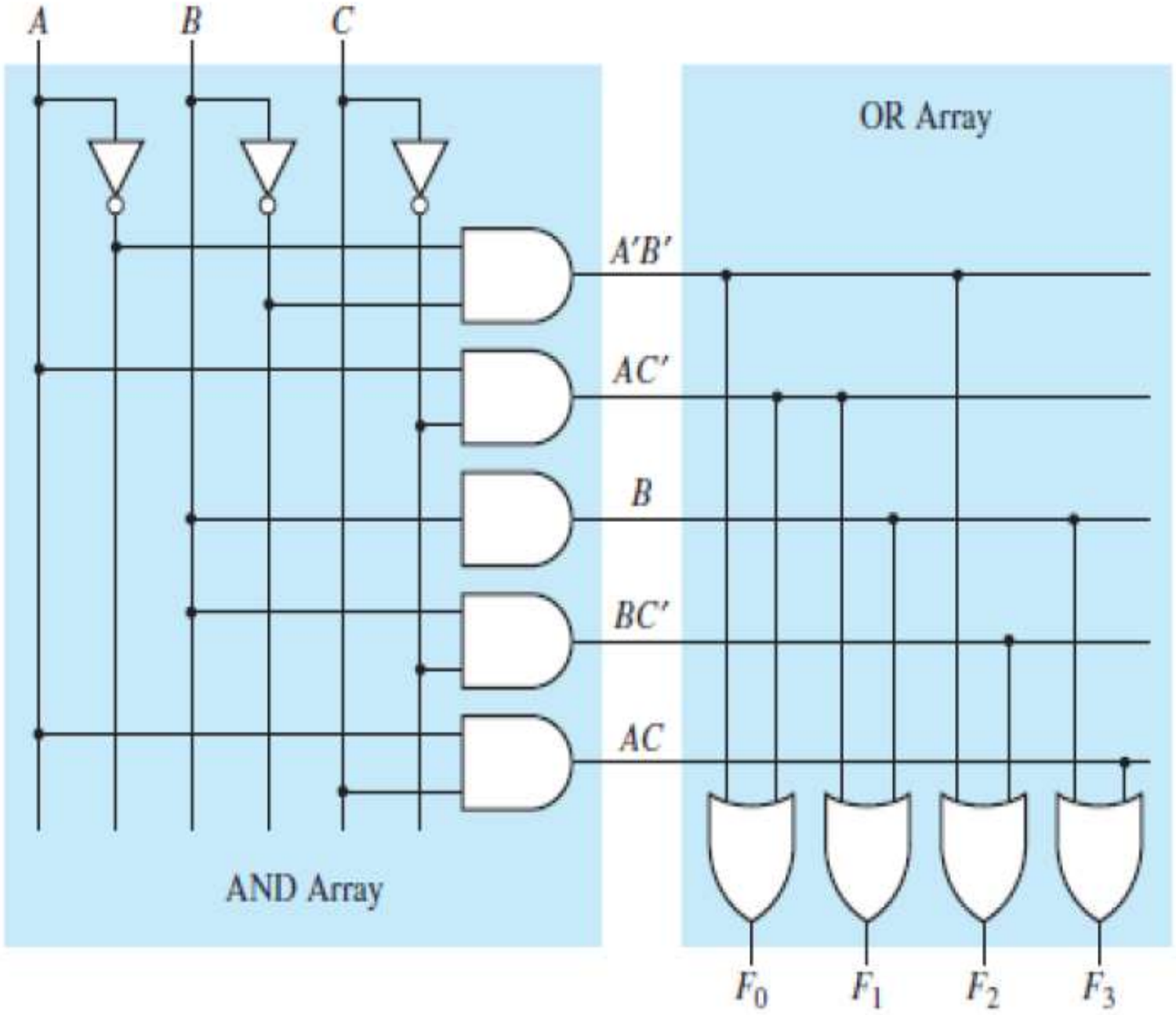


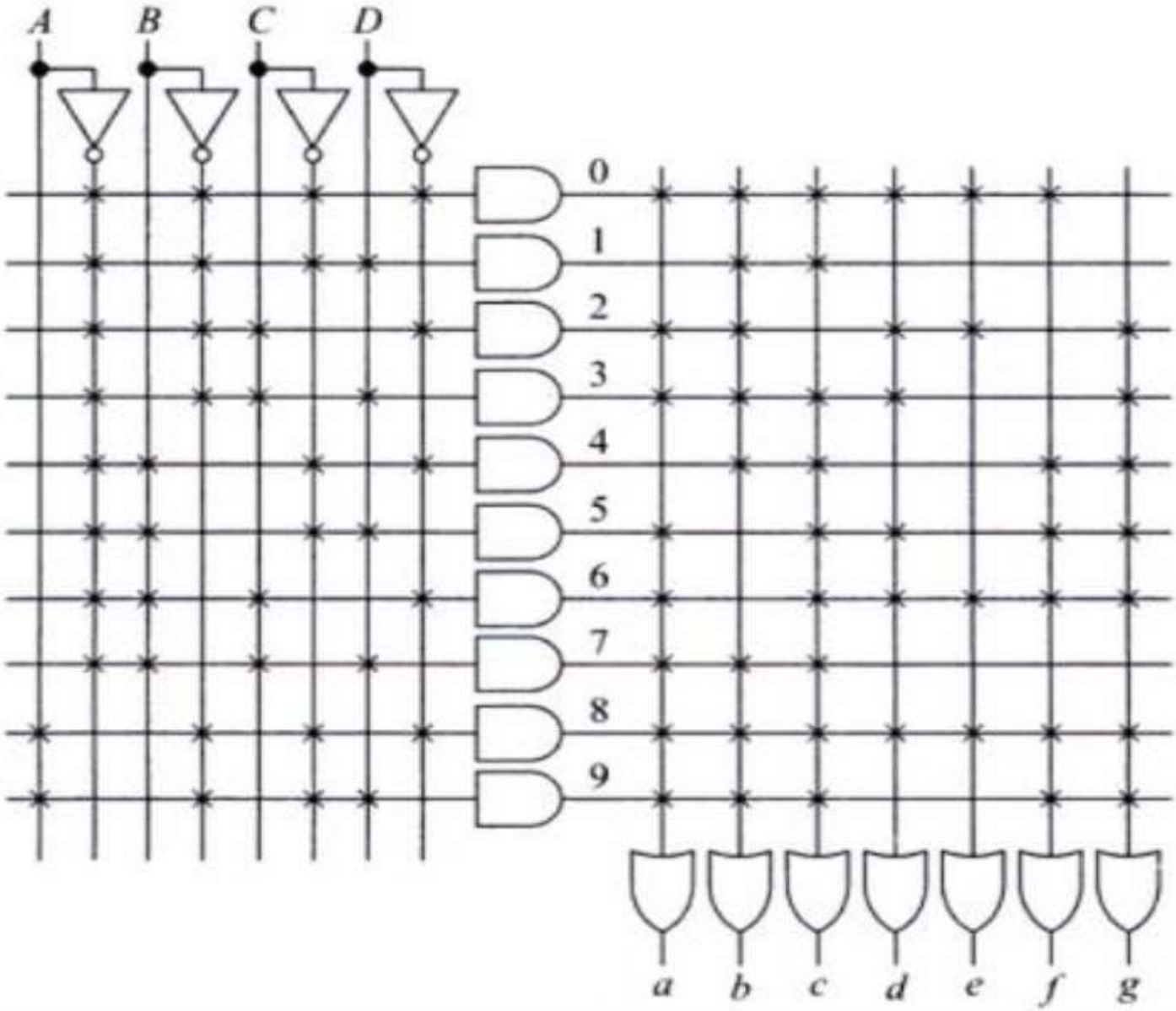
7446 Decoder-driver & 7448 Decoder-driver

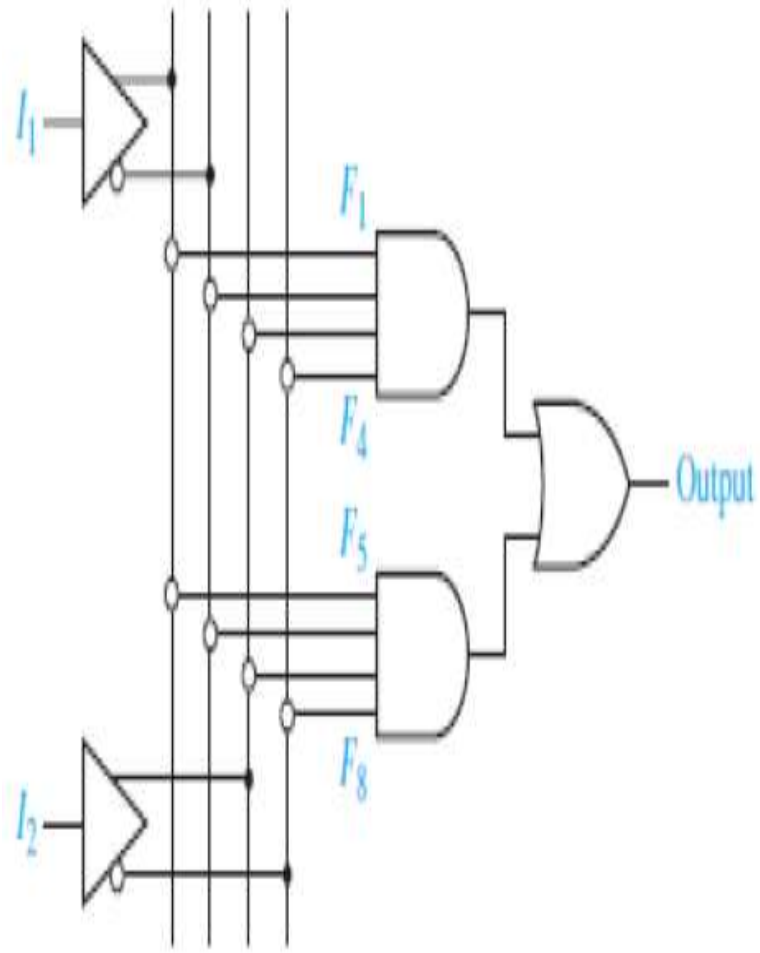
Programmable Logic Arrays (PLA):

A *PLA* with n inputs and m outputs (see the following Figure) can realize m functions of n variables. In PLA, the product terms of the input variables is realized by an AND array; and the OR array ORs together the product terms needed to form the output functions. Hence, a PLA implements a sum-of-products expression.

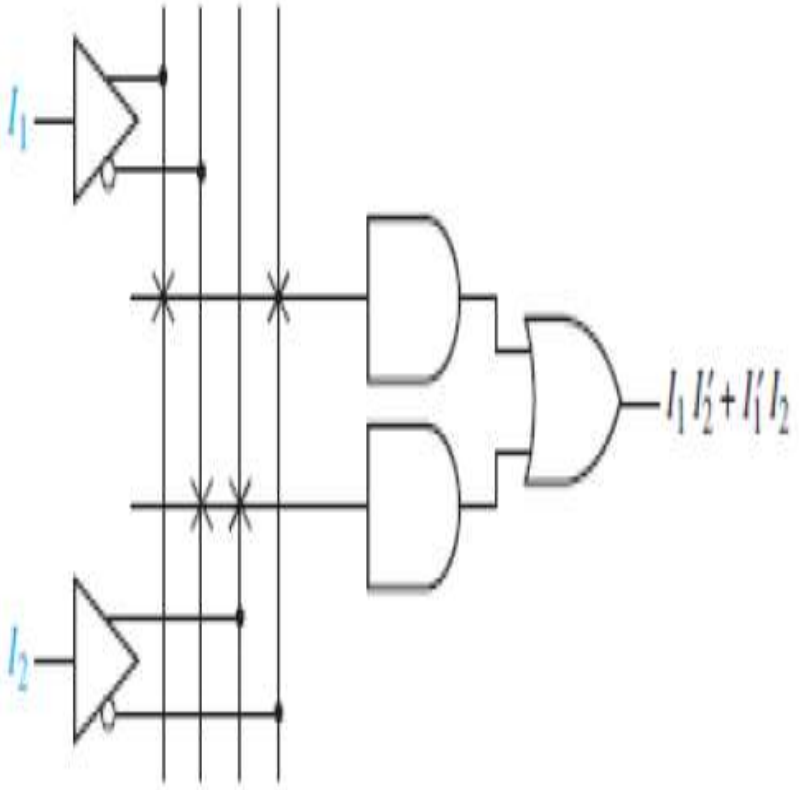



A
B
C
OR Array
A'B'
AC'
B
BC'
AC
AND Array
F0
F1
F2
F3


A
B
C
D
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
g



(a) Unprogrammed



(b) Programmed

diagram show the connections that are programmed into the PAL to implement the full adder equations.