CANARA ENGINEERING COLLEGE

BANTWAL, MANGALURU 574219

COMPUTER SCIENCE & ENGINEERING

CLASS

Computer **L**earning **A**dventure in **S**ocial **S**phere

COURSE TITLE (CODE)	ANALOG & DIGITAL ELECTRONICS(18CS33)
MODULE/ CHAPTER	MODULE 3
TOPICS	COMBINATION CIRCUITS
FACULTY	ANUPAMA V

"QUALITY EDUCATION AT AFFORDABLE COST"

0824-2278666 /67/68 www.canaraengineering.in canarait@gmail.com

Module 3

Combinational circuit design and simulation using gates

&

Multiplexers, Decoders and Programmable Logic Devices

Syllabus Blowup

Review of Combinational circuit design, design of circuits with limited Gate Fan-in ,Gate delays and Timing diagrams, Hazards in combinational Logic, simulation and testing of logic circuits

Multiplexers, three state buffers, decoders and encoders, Programmable Logic devices, Programmable Logic Arrays, Programmable Array Logic.

REVIEW OF COMBINATIONAL CIRCUIT DESIGN

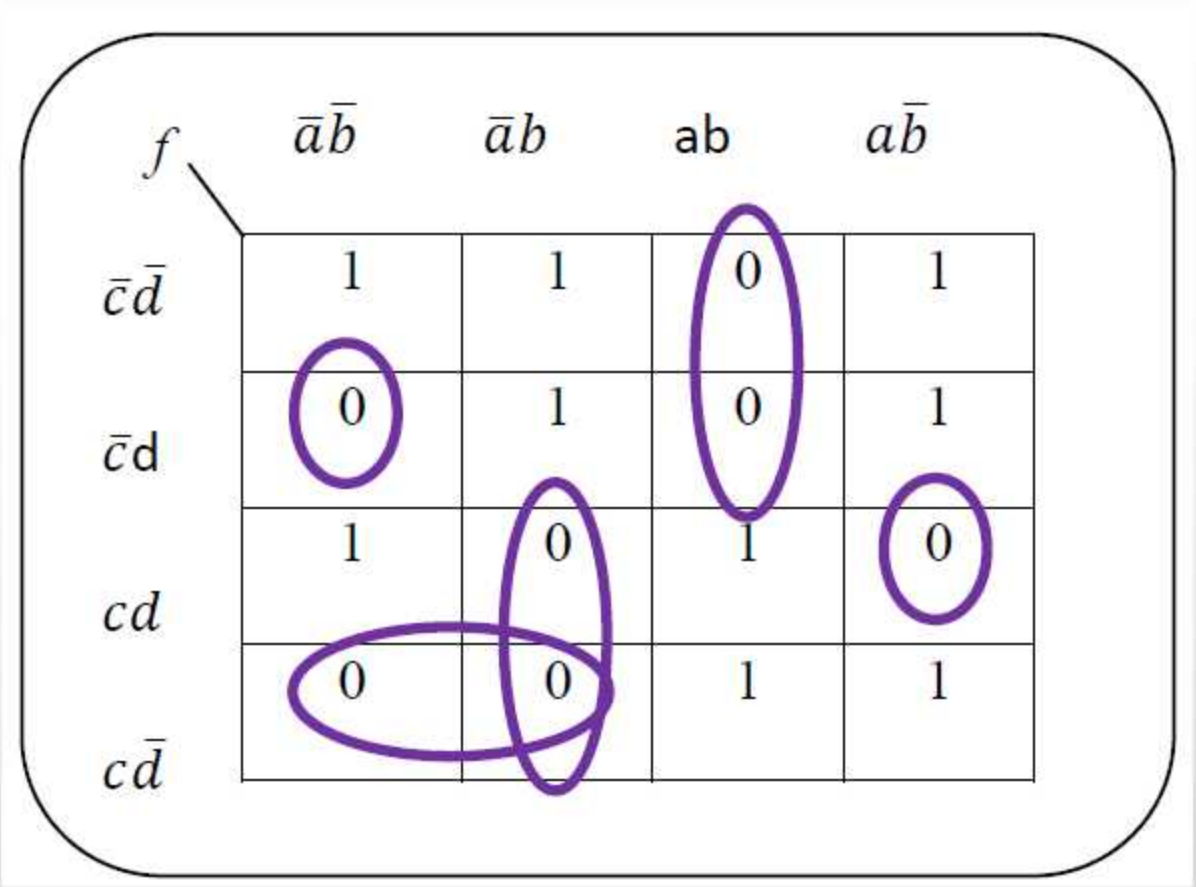
Steps involved in the design of a combinational switching circuit

- Set up a truth table
- Derive simplified algebraic expressions
- Reduce the total number of gates or gate inputs
- Realize using SOP or POS method

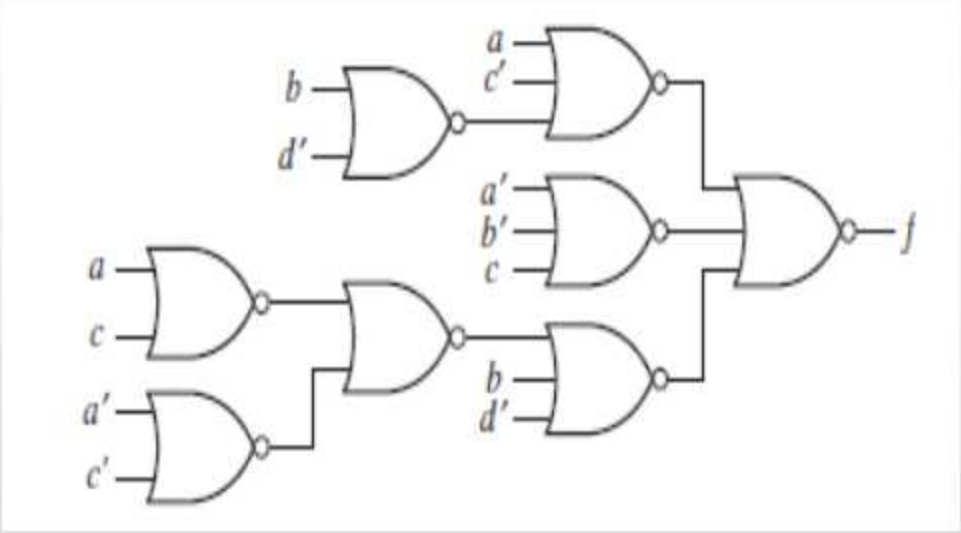
DESIGN OF CIRCUITS WITH LIMITED GATE FAN-IN

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.

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

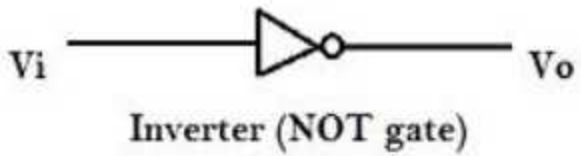


The product-of-sum equation is: $f = a'+b'+c\ a+b'+c'\ a+c'+d\ a+b+c+d'\ a'+b+c'+d'$ As can be seen from the preceding expression, a two-level realization requires three-three input gates, two four-input gates and one five-input gate. The expression for f' is factored to reduce the maximum number of gate inputs to three and, then, it is complemented. Or $f'= abc'+ a'bc+a'cd'+ a'b'c'd+ab'cd$ i.e., $f'=a\ bc'+ a'c\ b+d' + b'd(a'c'+ ac)$ Or $f=[\ a'+b'+c\ ]\ a+c' + b'd\ [\ b+d' + a+c\ a'+c'\ ]$



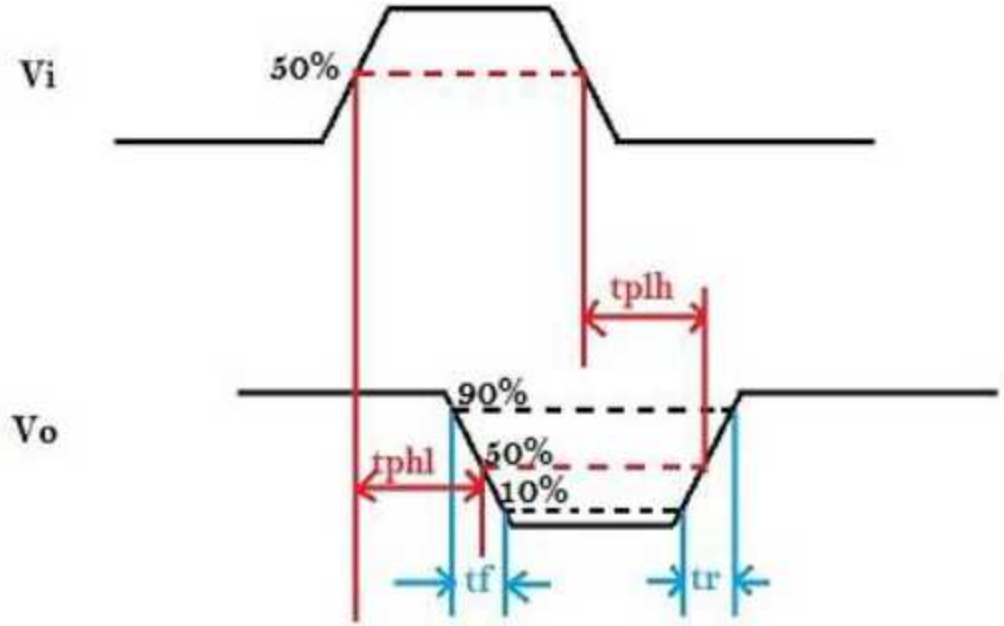
GATE DELAYS AND TIMING DIAGRAMS

When the input to a logic gate is changed, the output will not change instantaneously. The gates take a finite time to react to a change in input, so that the change in the gate output is delayed with respect to the input change



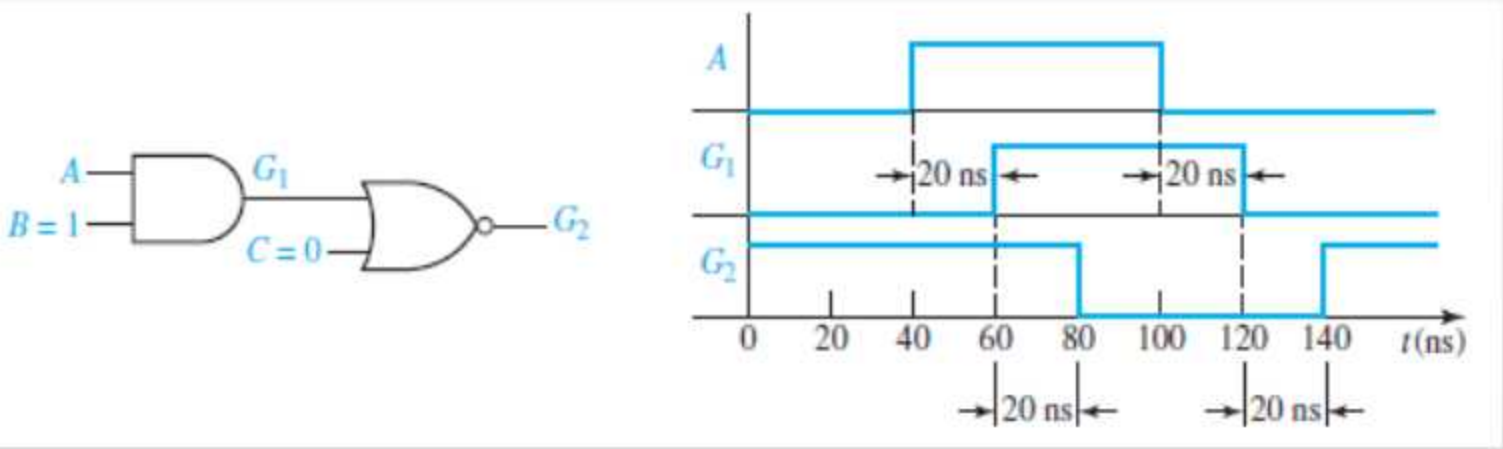
Inverter (NOT gate)

tr = Rise transition time
tf = Fall transition time
tphl = Propagation delay high-low
tplh = Propagation delay low-high



Example

Assume that, each gate has a propagation delay of 20 ns (nanoseconds). This timing diagram indicates what happens when gate inputs B and C are held at constant values 1 and 0, respectively, and input A is changed to 1 at *t = 40 ns* and then changed back to 0 at *t = 100 ns*. The output of gate G1 changes 20 ns after A changes, and the output of gate G2 changes 20 ns after G1 change



Hazards in combinational logic

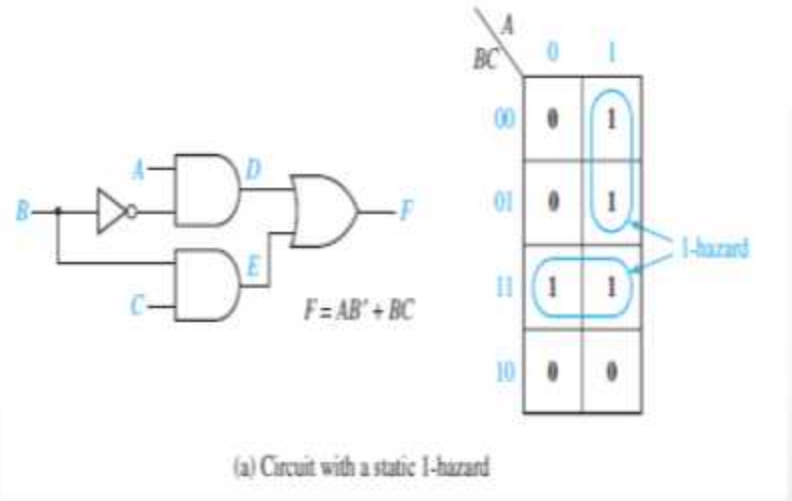
When the input to a combinational circuit changes, unwanted switching transients may appear in the output. These transients occur when different paths from input to output have different propagation delays

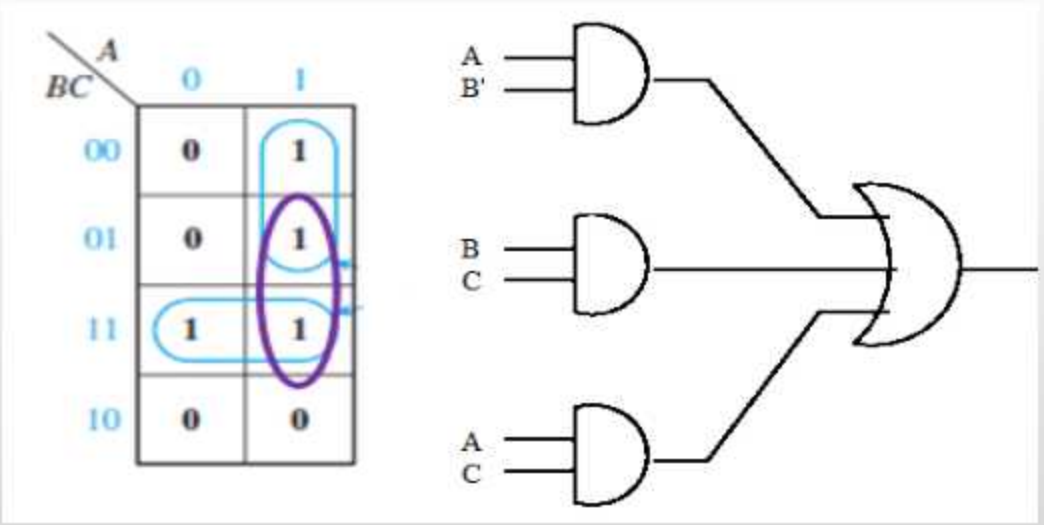
- If, in response to any single input change and for some combination of propagation delays, a circuit output may momentarily go to 0 when it should remain a constant 1, we say that the circuit has a ***Static 1-hazard***.
- Similarly, if the output may momentarily go to 1 when it should remain a 0, 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***.

Hazards

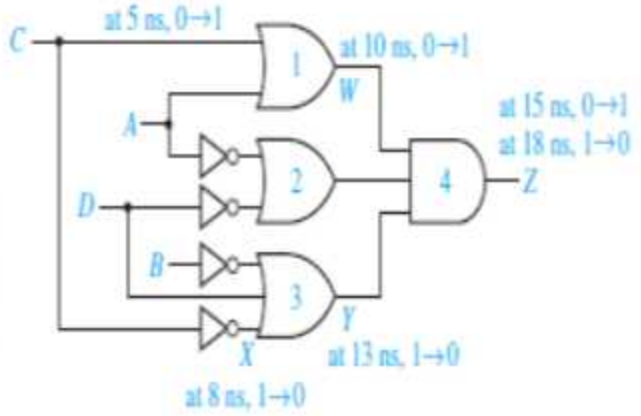
(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazards

Detection of static-1 Hazard and Solution

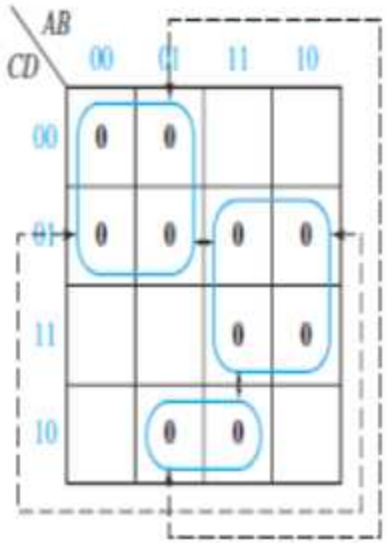




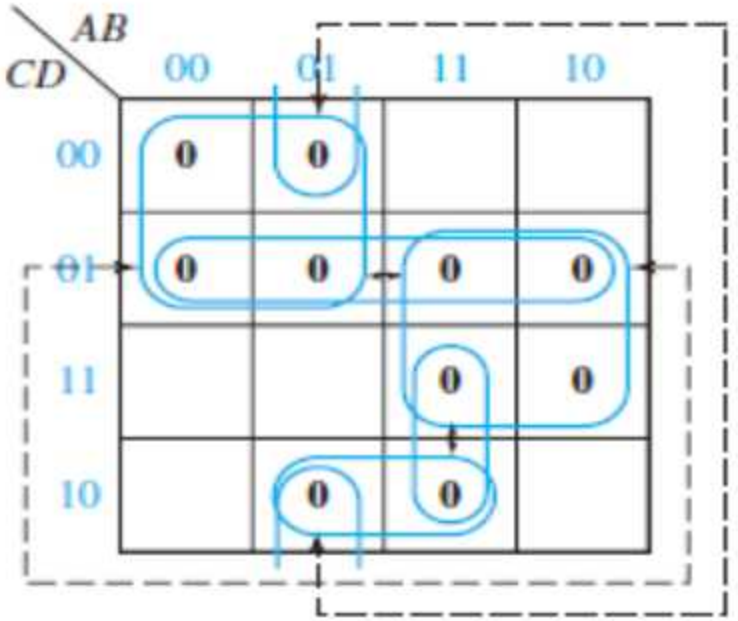
Detection of static-0 Hazard and Solution



(a) Circuit with a static 0-hazard



(b) Karnaugh map for circuit of (a)



Summary

To design a circuit which is free of static and dynamic hazards, the following procedure may be used:

1. Find a sum-of-products expression (F^t) for the output in which every pair of adjacent 1's is covered by a 1-term. (The sum of all prime implicants will always satisfy this condition.) A two-level AND-OR circuit based on this F^t will be free of 1-, 0-, and dynamic hazards.
2. If a different form of the circuit is desired, manipulate F^t to the desired form by simple factoring, DeMorgan's laws, etc. Treat each x_i and x_i' as independent variables to prevent introduction of hazards.

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

Checking of circuits

1.The circuit inputs are applied to the first set of gates in the circuit, and the outputs of those gates are calculated.

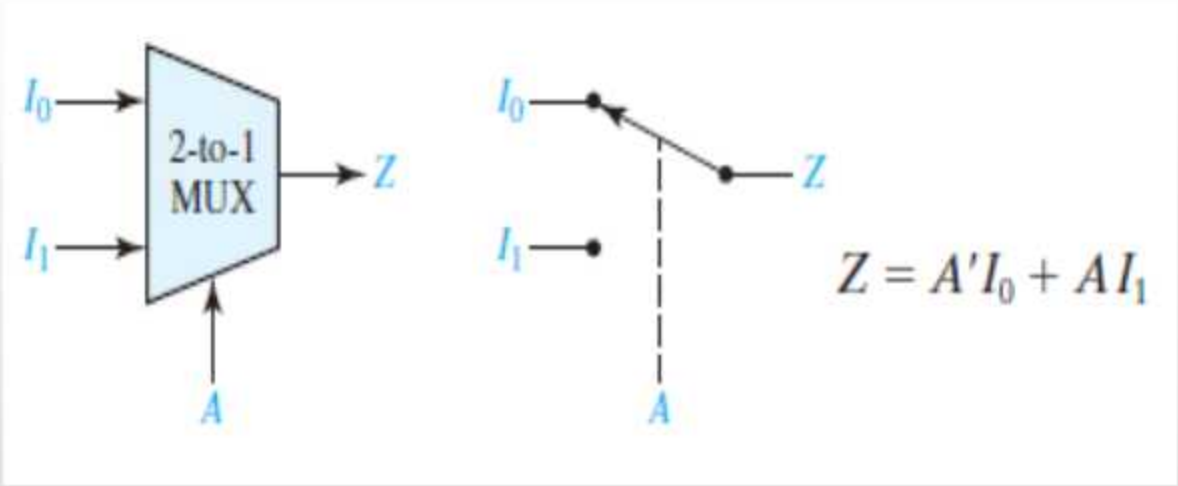
2. The outputs of the gates which changed in the previous step are fed into the next level of gate inputs. If the input to any gate has changed, then the output of that gate is calculated.

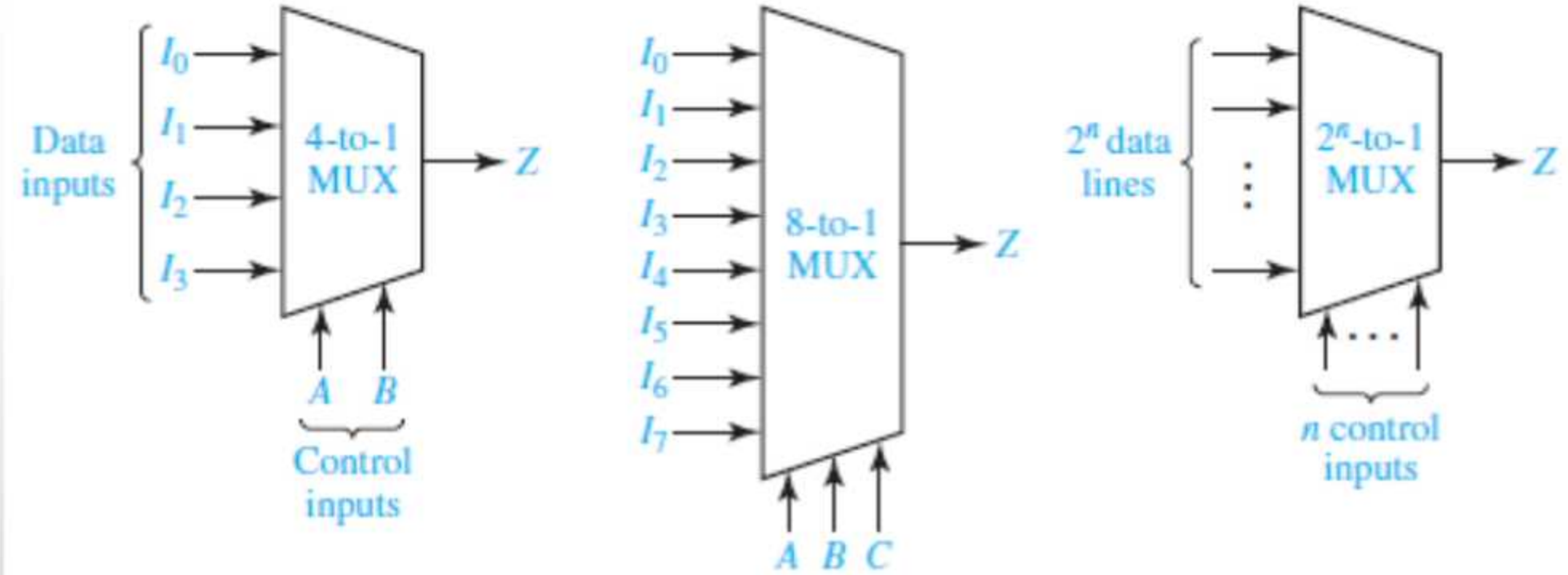
3. Step 2 is repeated until no more changes in gate inputs occur. The circuit is then in a steady-state condition, and the outputs may be read.

4. Steps 1 through 3 are repeated every time a circuit input changes.

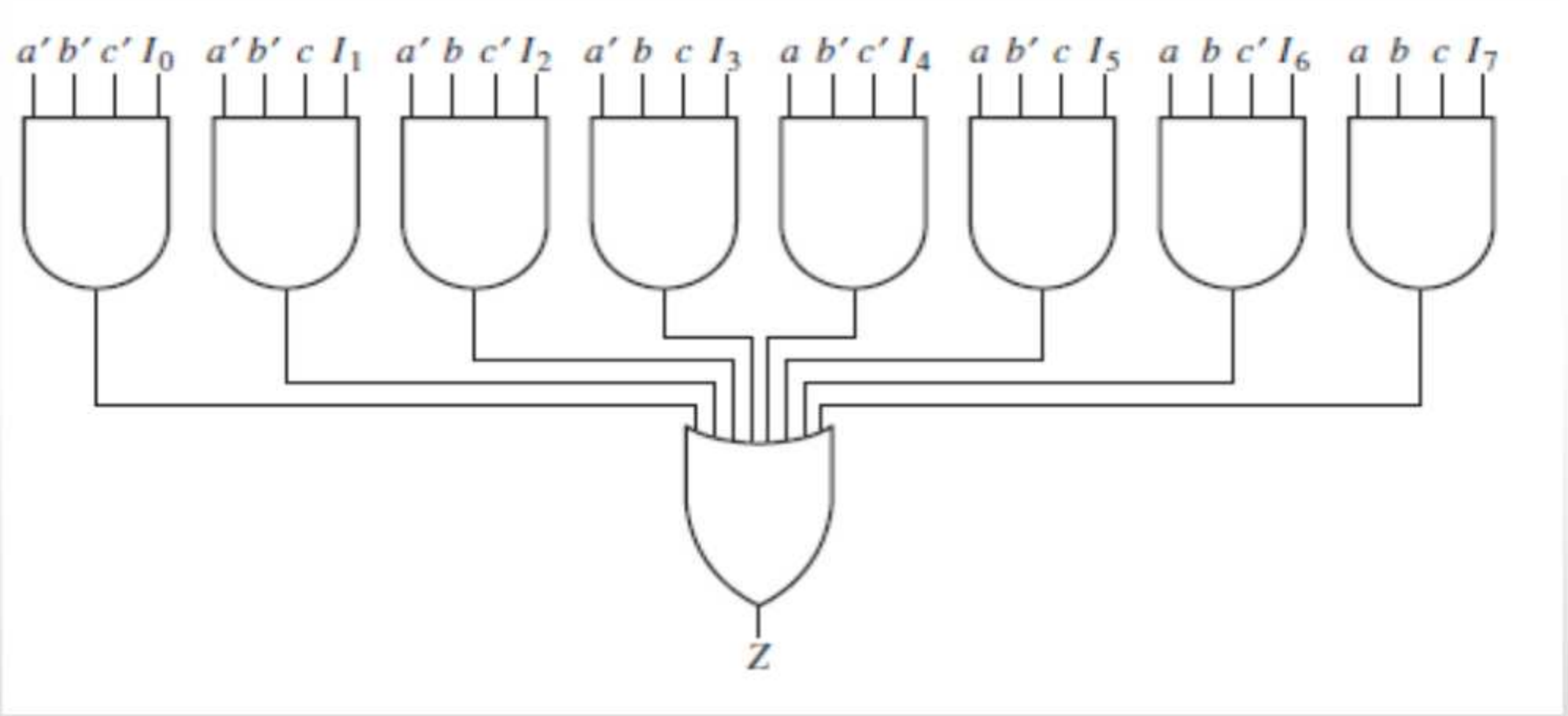
Multiplexers

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




Data inputs
I_0
I_1
I_2
I_3
4-to-1 MUX
Z
A B
Control inputs
I_0
I_1
I_2
I_3
I_4
I_5
I_6
I_7
8-to-1 MUX
Z
A B C
2^n data lines
2^n-to-1 MUX
Z
n control inputs

Internal circuitry of 8:1 MUX

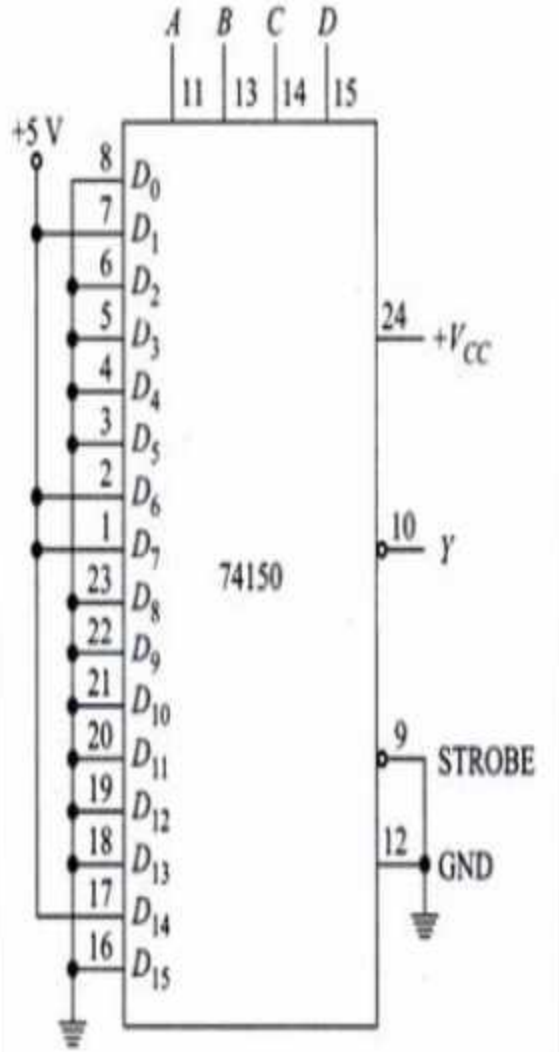


Multiplexer logic

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.

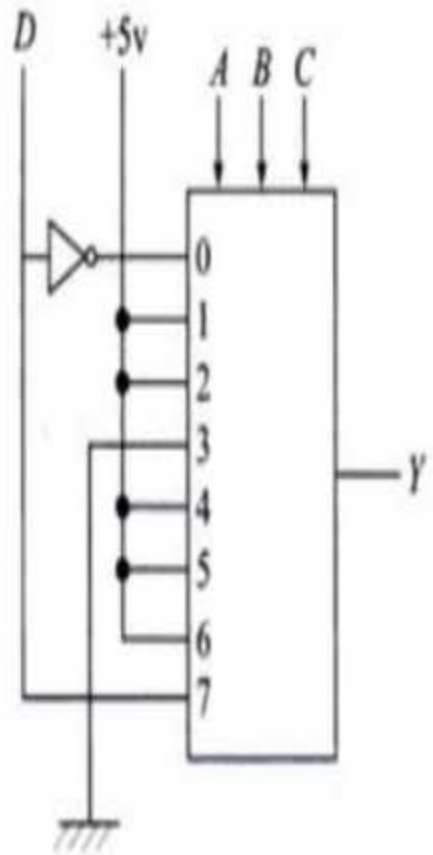
Implement Y (A, B, C, D) = Σm (0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15) using 16-to-1 multiplexer (IC 74150) & 8-to-1 multiplexer.
rd method is the *multiplexer solution*

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	

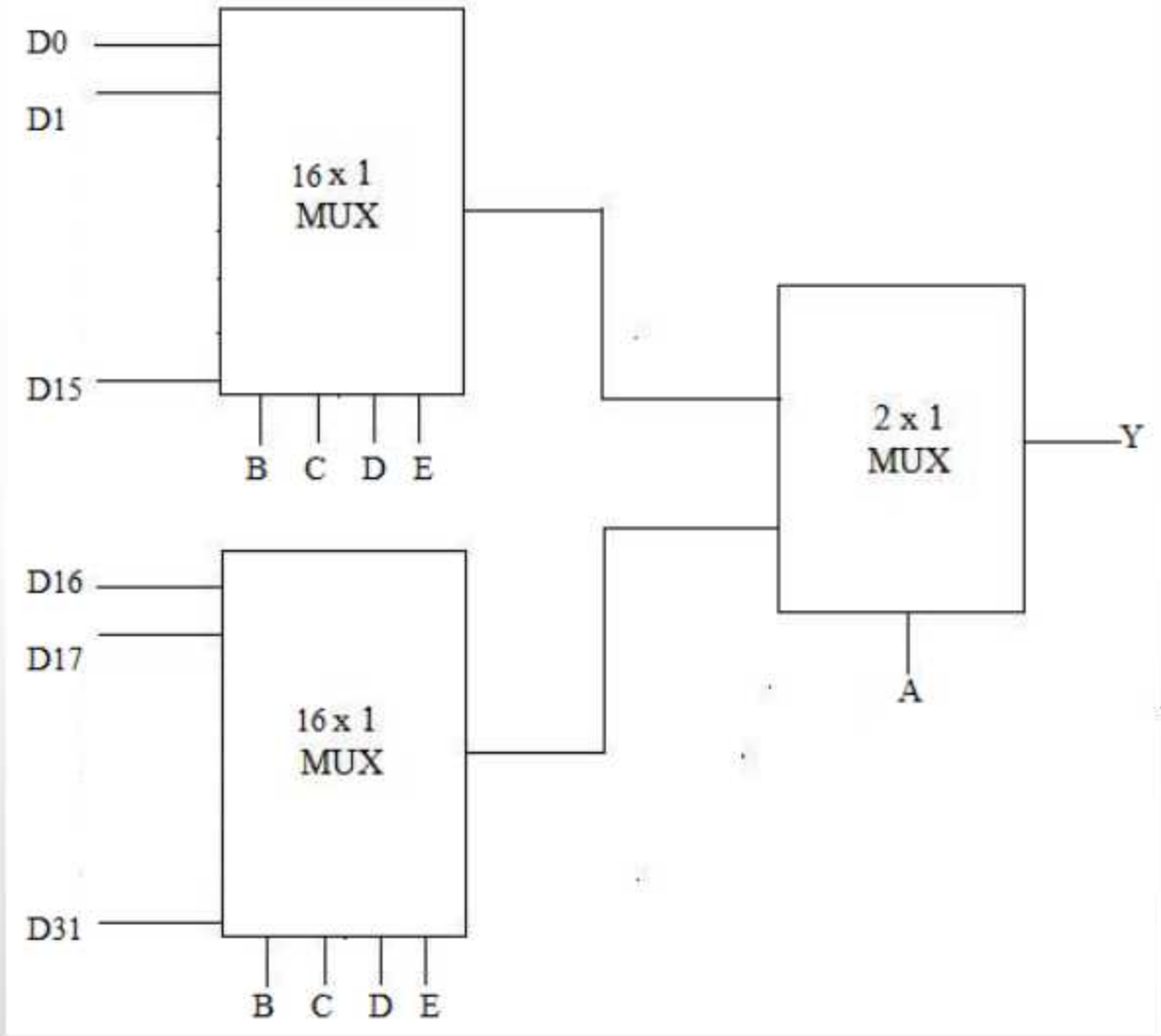


We follow a procedure that is similar to the one that we adopted in entered variable map method to implement Y using 8-to-1 MUX.

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

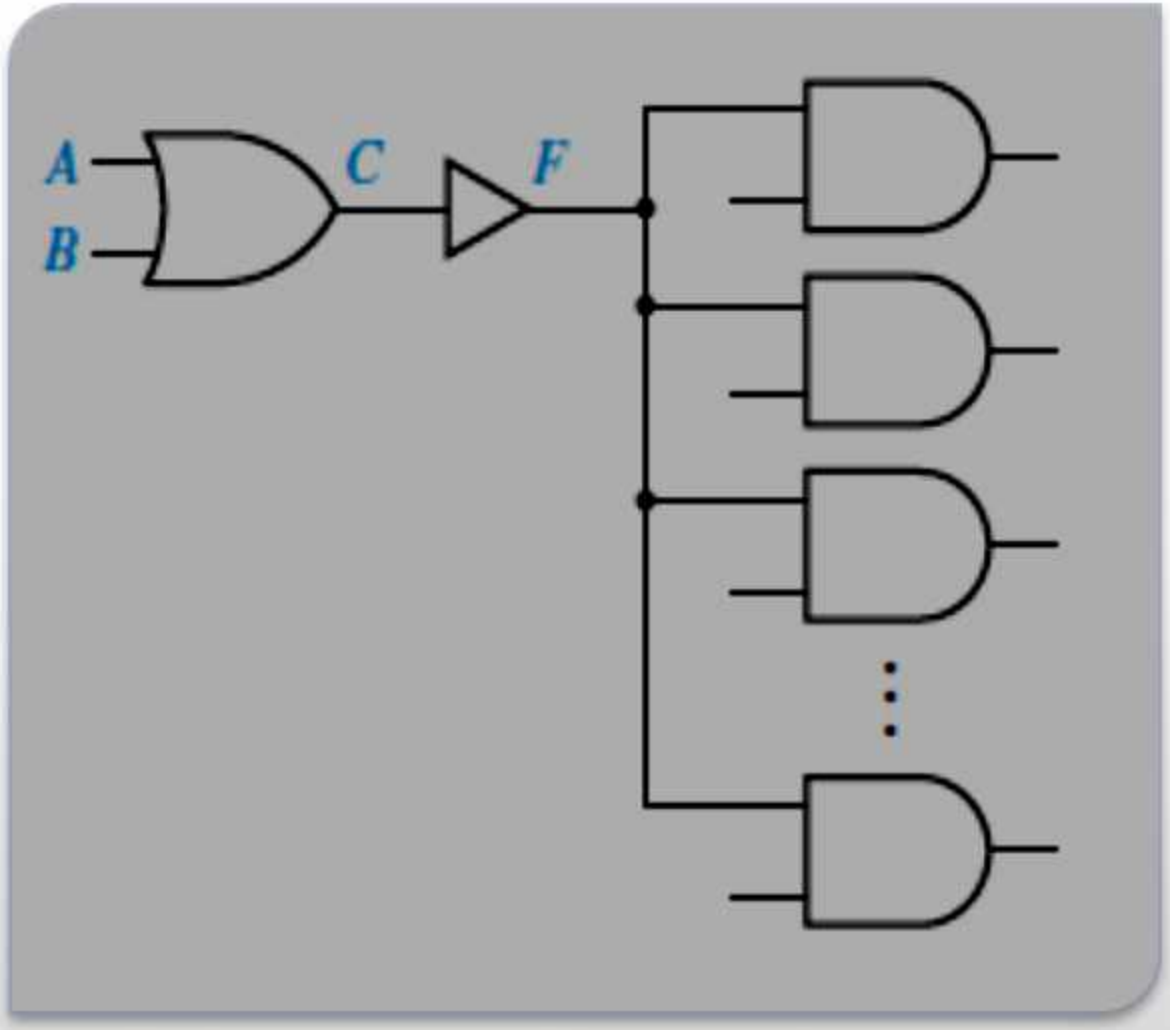


Design a 32-to-1 multiplexer using two 16-to-1 multiplexer and one 2-to-1 multiplexer



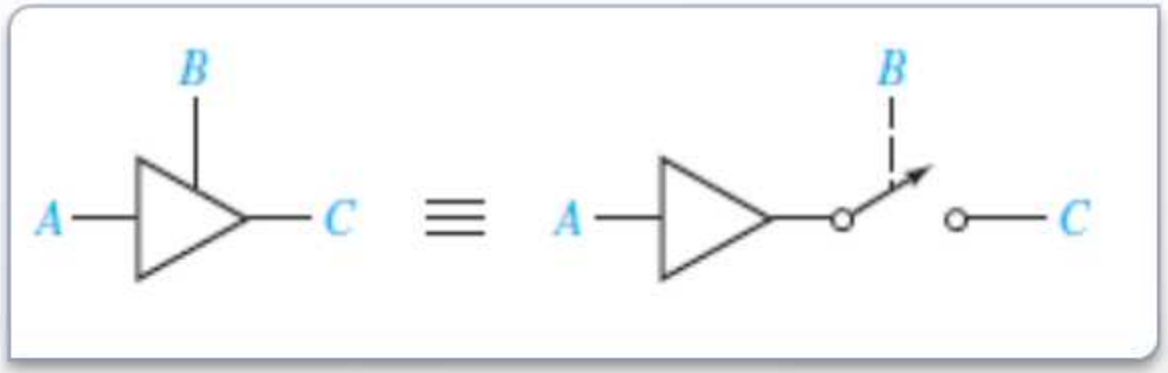
Three state buffers

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, $F = C$.

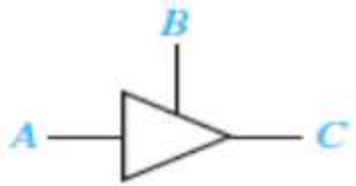


Tri-state buffer

When the enable input B is 1, the output C equals A; when B is 0, the output C acts like an open circuit. In other words, when B is 0, the output C is effectively disconnected from the buffer output so that no current can flow. This is often referred to as a Hi-Z (high-impedance) state of the output because the circuit offers a very high resistance or impedance to the flow of current. Three-state buffers are also called *tri-state buffers*.

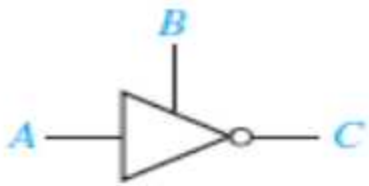


Four types of Tri-state buffer



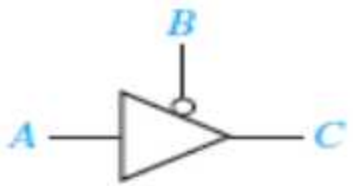
B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

(a)



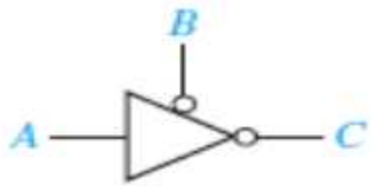
B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

(b)



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

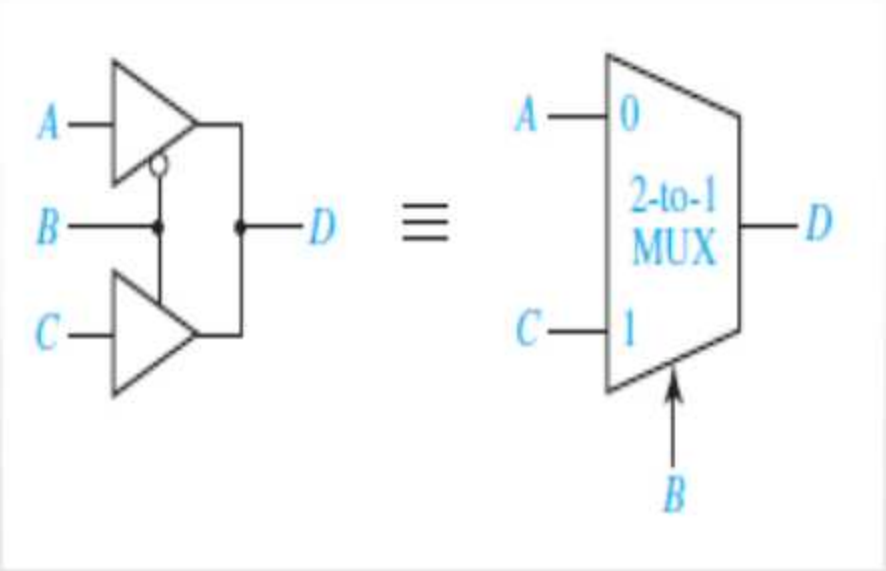
(c)



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

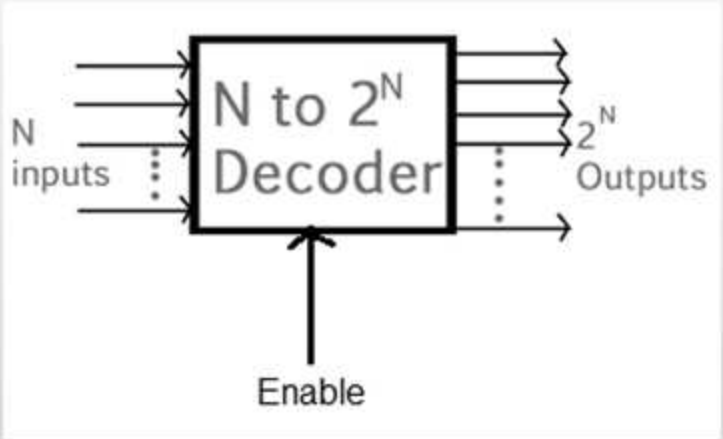
(d)

In the following figure, the outputs of two three-state buffers are tied together. When B = 0, the top buffer is enabled, so that D = A; when B = 1, the lower buffer is enabled, so that D = C. Therefore, $D = B'A + BC$. this is logically equivalent to using a 2-to-1 multiplexer to select the A input when B = 0 and the C input when B = 1.



Decoders

- Decoder is a combinational circuit that has ‘n’ input lines and maximum of 2^n output lines.
- One of these outputs will be ACTIVATED based on the combination of inputs present, when the decoder is enabled.
- Decoder detects a particular code.
- The outputs of the decoder are the minterms of ‘n’ input variables.

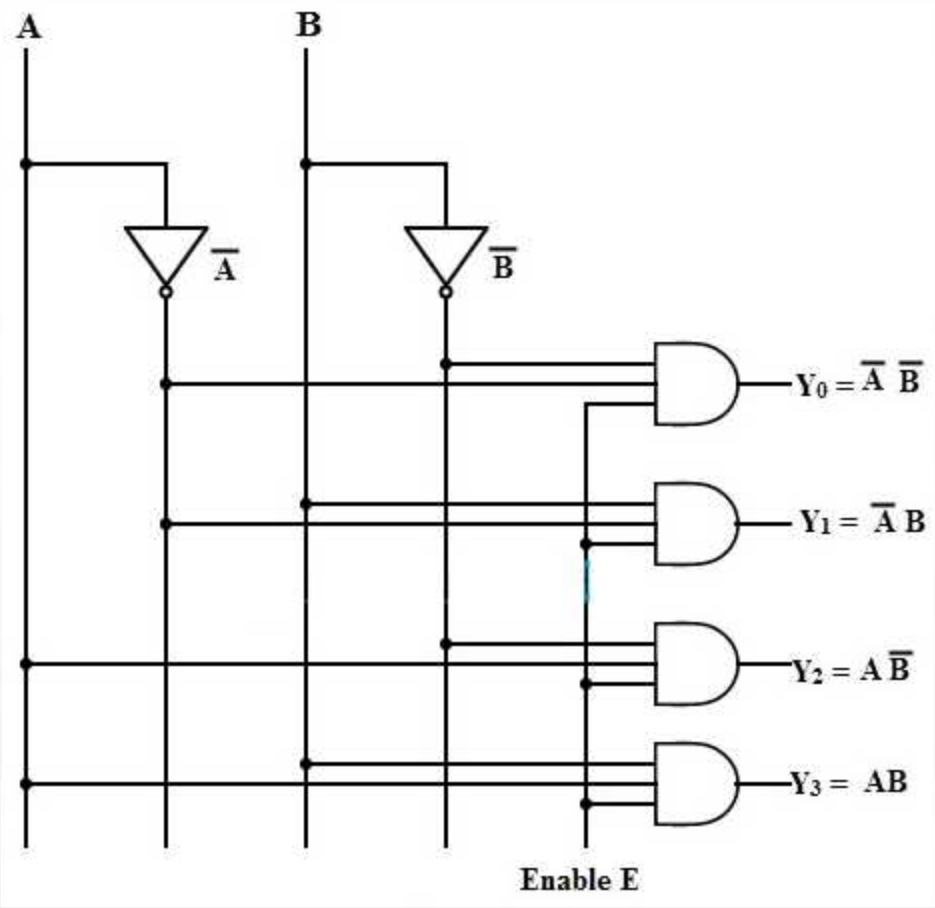


2 to 4 Decoder

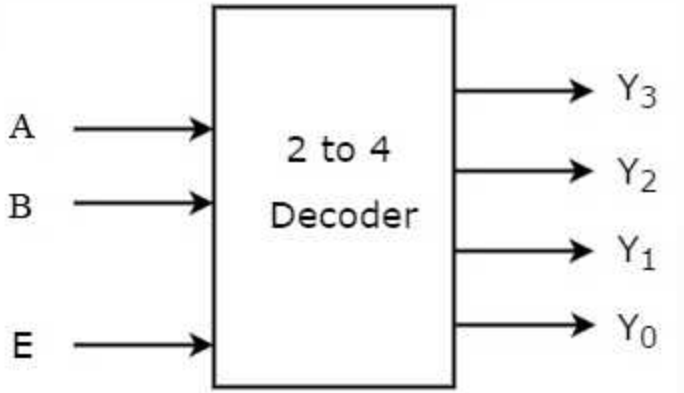
- Is a combinational circuit that has ‘2’ input lines and 4 output lines.
- One of the 4 outputs will be activated based on the combination of inputs present, when the decoder is enabled.

2 to 4 Decoder

Logic Diagram



Block Diagram



Truth Table

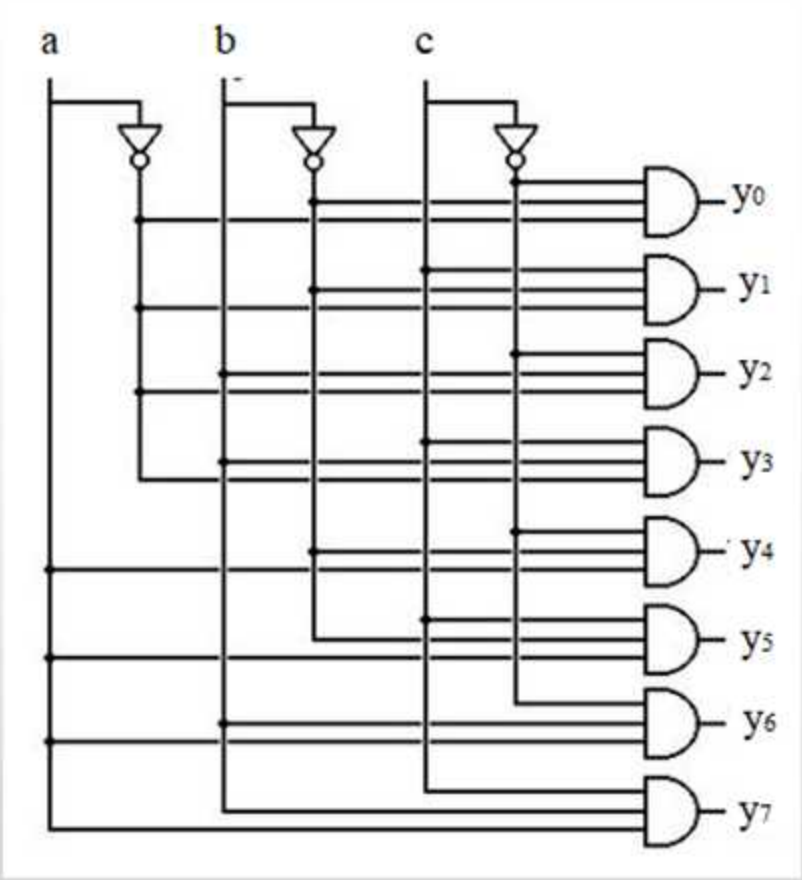
E	A	B	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

3 to 8 Decoder

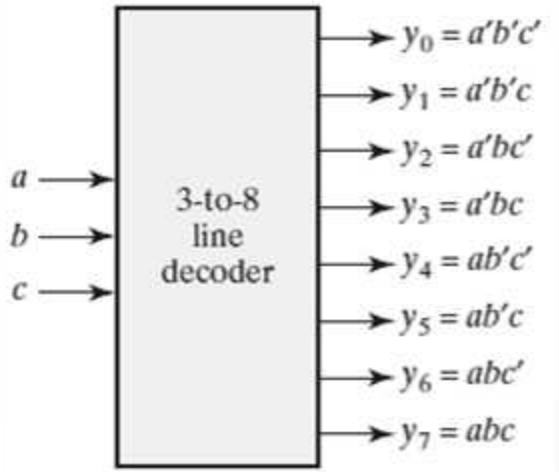
- Is a combinational circuit that has ‘3’ input lines and 8 output lines.
- One of the 8 outputs will be activated based on the combination of inputs present, when the decoder is enabled.

3 to 8 Decoder

Logic Diagram



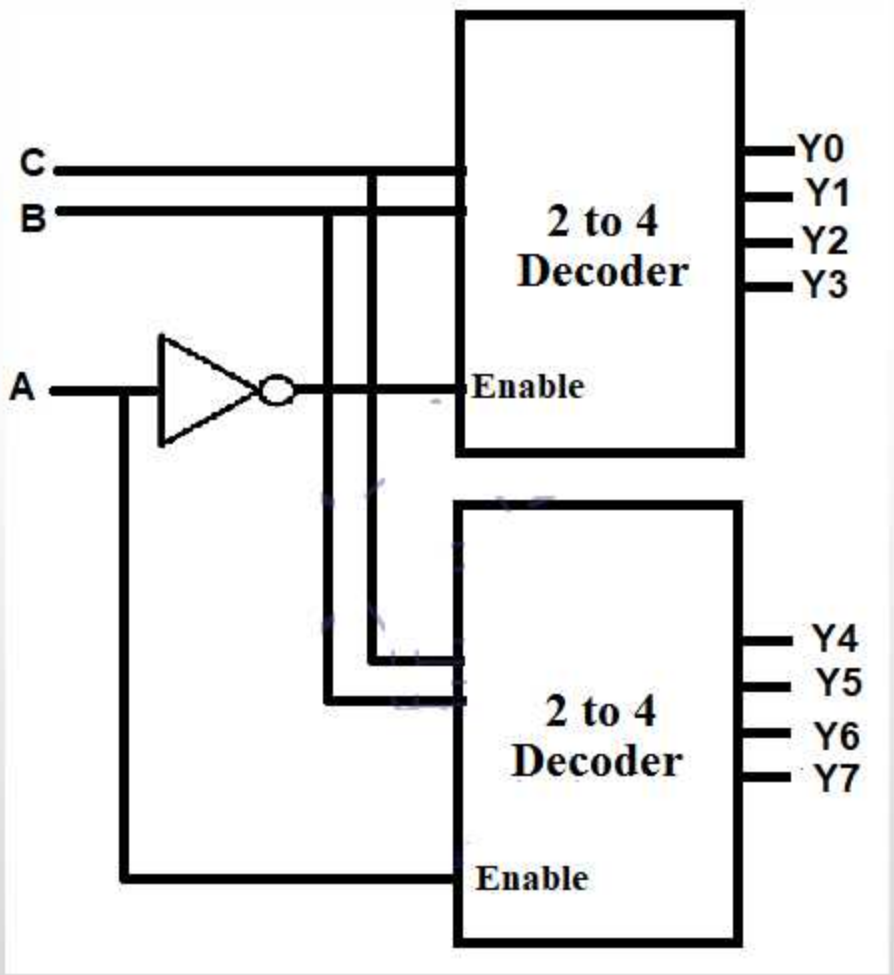
Block Diagram



Truth Table

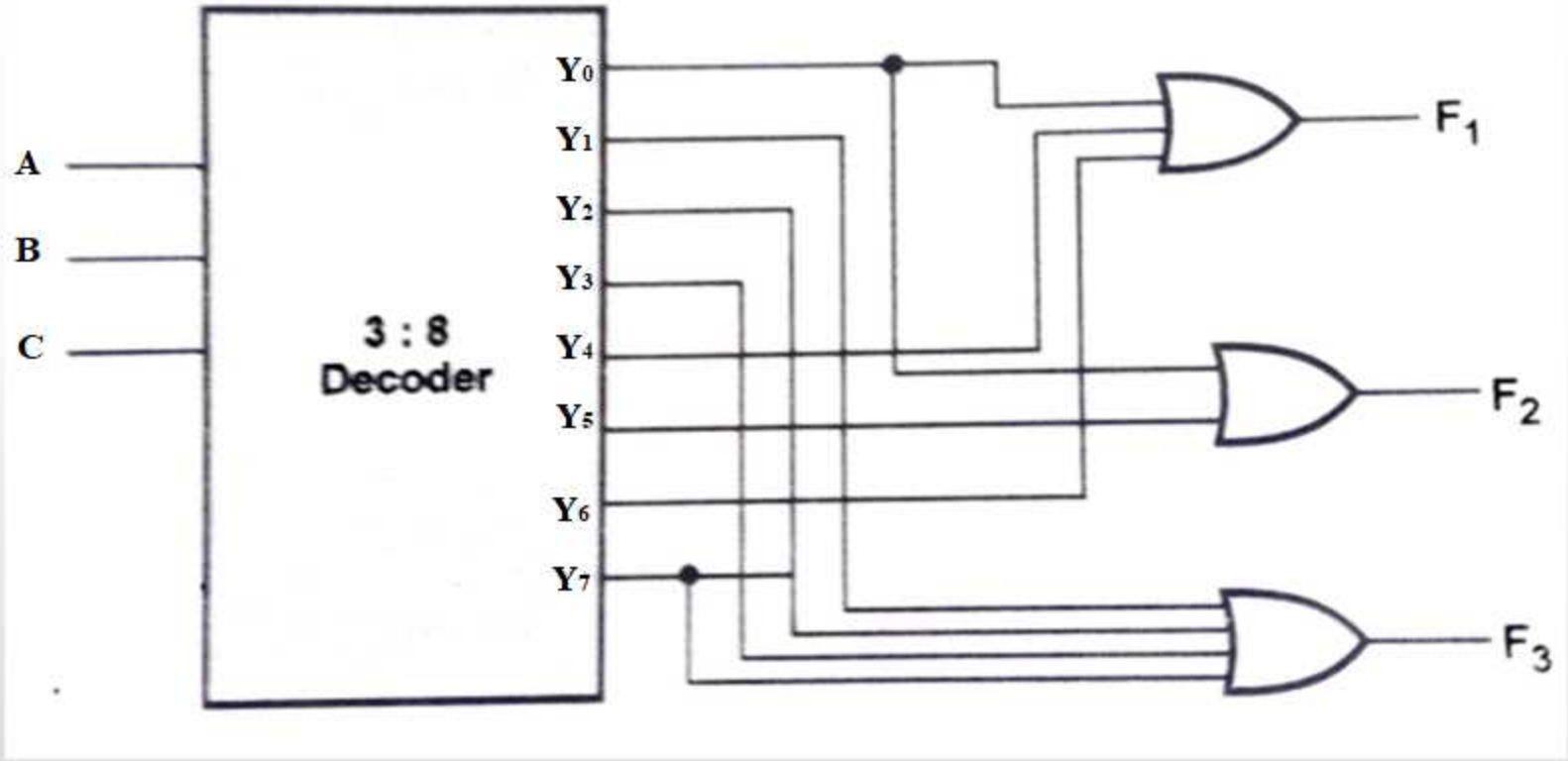
a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Construct 3 to 8 decoder using 2 to 4 decoder

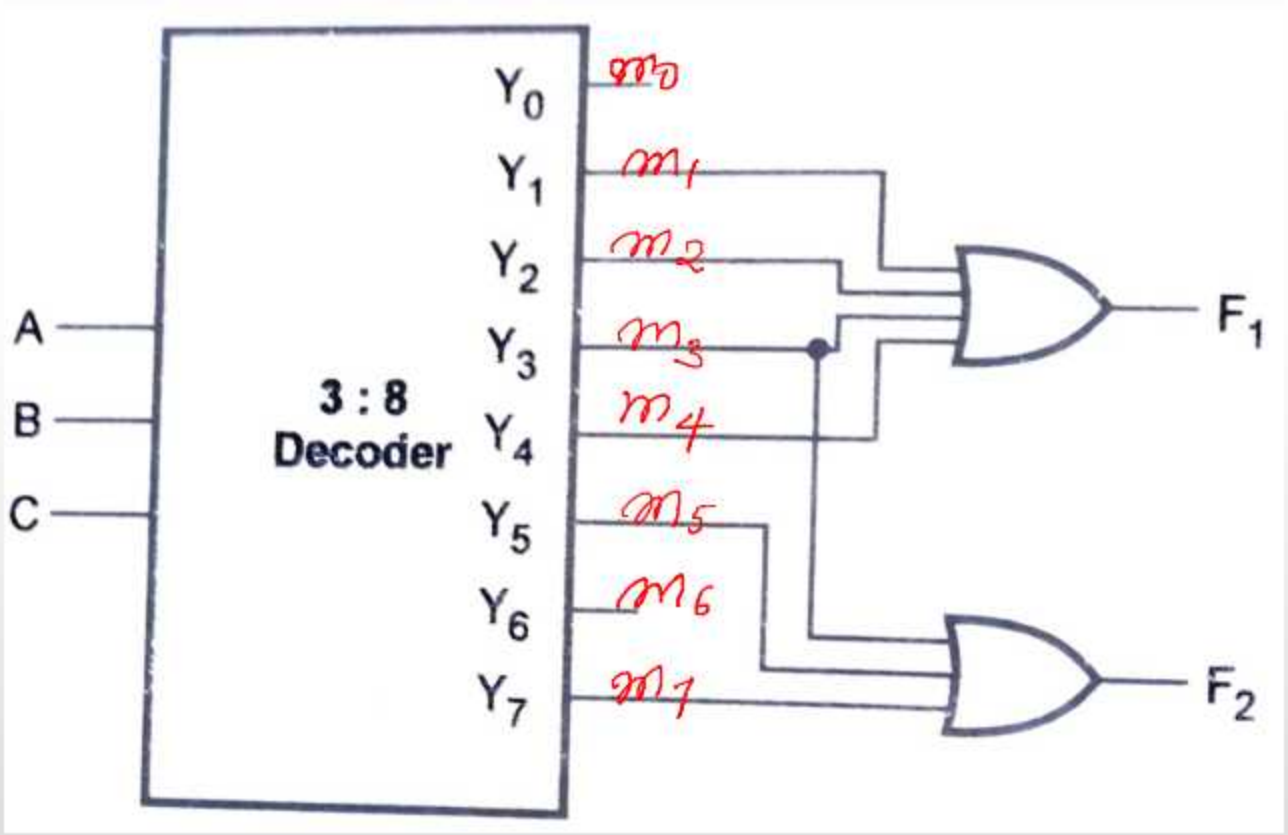


Show how using a 3-to-8 decoder and multi-input OR gates following Boolean expressions can be realized simultaneously.

F_1 (A, *B*, C) = Σ*m*(*O*, 4, 6); F_2(*A*, *B*, C) = Σ*m*(*0*, 5); F_3 (*A*, *B*, C) = Σ*m* (1, 2, 3, 7)

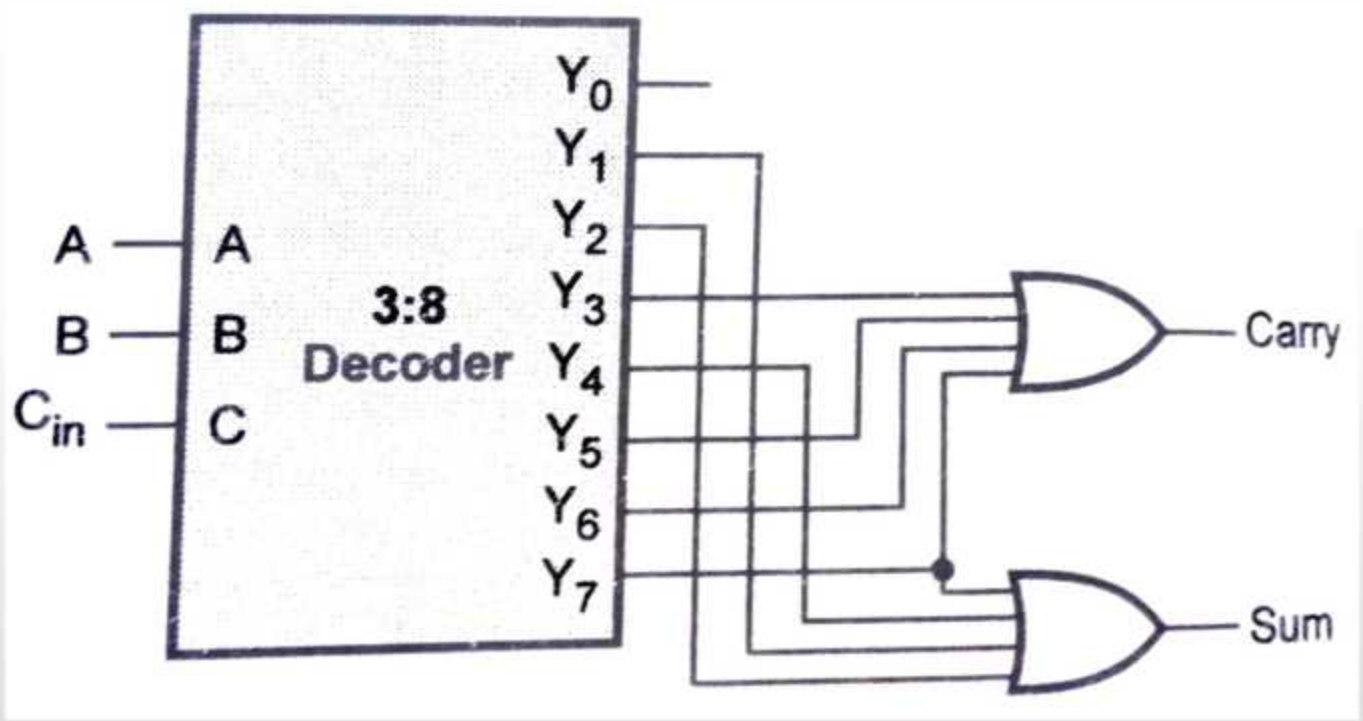


Realize the following Boolean expressions using the 3-to-8 decoder.
F_1 (A, *B*, C) = Σ*m*(1,2,3,4); F_2(*A*, *B*, C) = Σ*m*(3,5,7);



Implement a full adder using a 3-to-8 decoder.

INPUT			OUTPUT	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Sum = Σm (1, 2, 4, 7)

Carry = Σm (3, 5, 6, 7)

Binary coded decimal - BCD

- The BCD code expresses each digit in a decimal number by its nibble equivalent.
- BCD digits are from 0000 to 1001.
- 1010 to 1111 cannot exist in the BCD.
- Decimal number 429 is changed to its BCD form as follows:

4	2	9
↓	↓	↓
0100	0010	1001

- Decimal equivalent of $(0101\ 0111\ 1000)_{BCD}$ is 578

0101	0111	1000
↓	↓	↓
5	7	8

4 to 10 Decoders

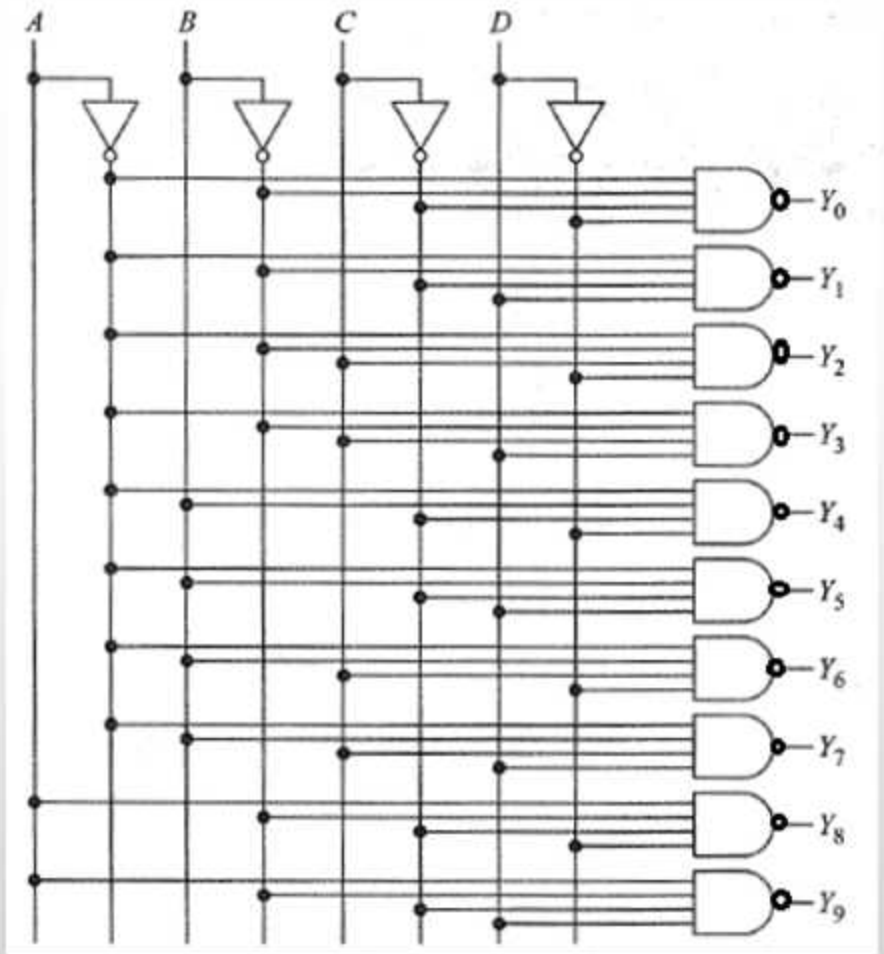
$2^4 = 16$

$10 < 16$

- Called as 1-of-10 decoder.
- Called 4 to 10 line decoder.
- Called as BCD-to-decimal decoder.
- Circuit is also called a BCD-to-decimal converter.
- Is a combinational circuit that has ‘4’ input lines and ‘10’ output lines.
- One of the 10 outputs will be activated based on the combination of inputs present, when the decoder is enabled.

4 to 10 Decoders

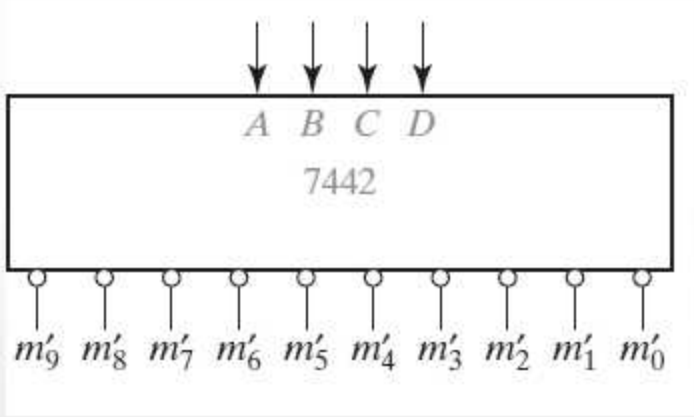
Circuit Diagram



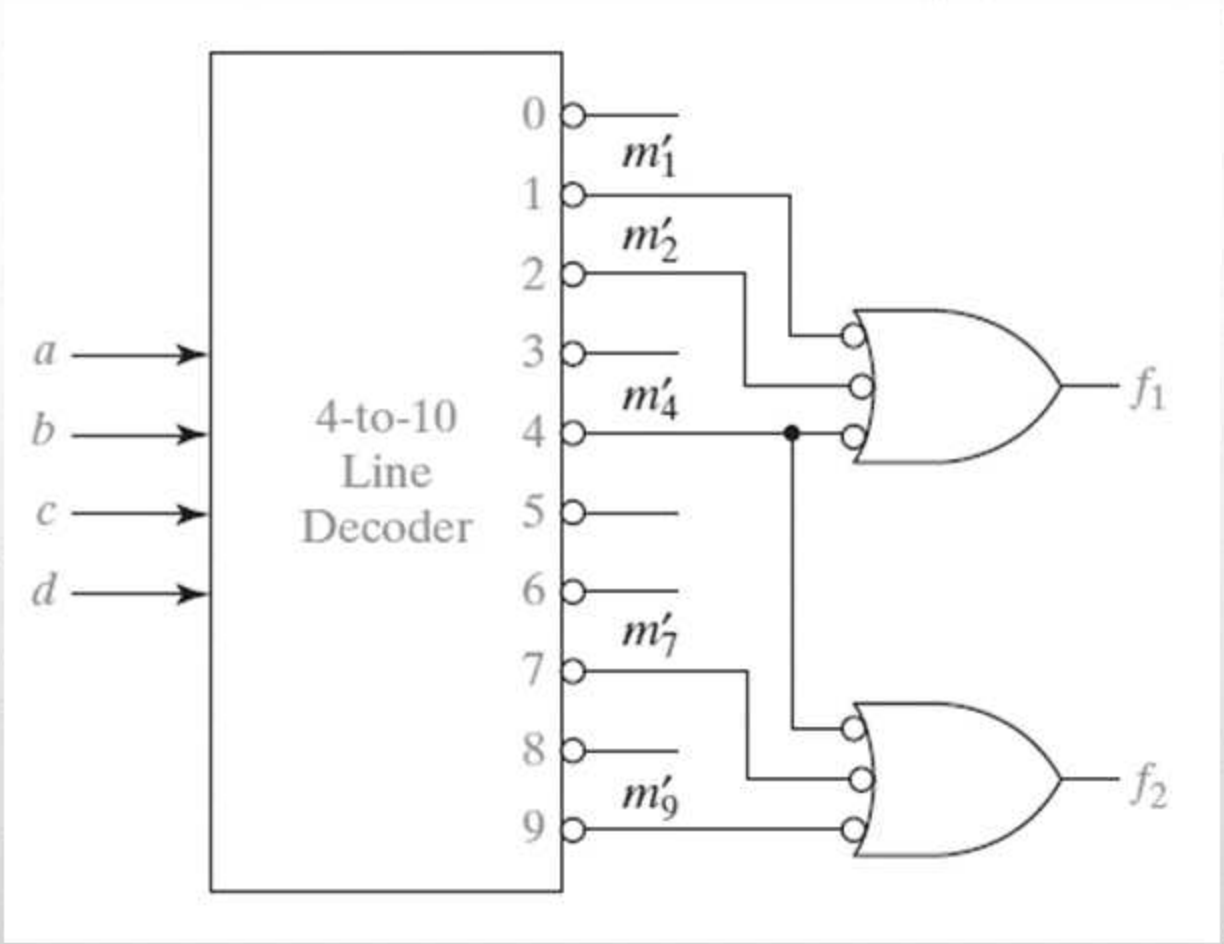
Truth Table

BCD Input				Decimal Output									
A	*B*	*C*	*D*	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

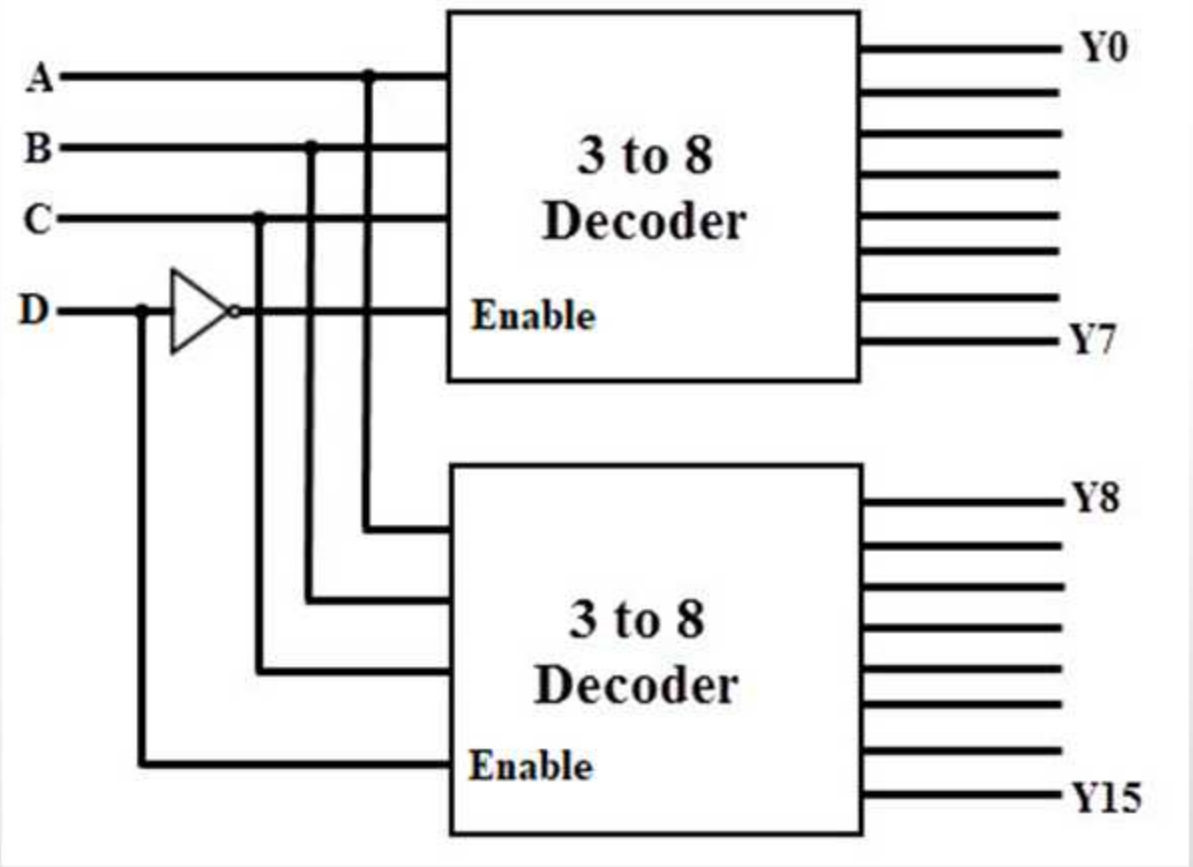
Block Diagram



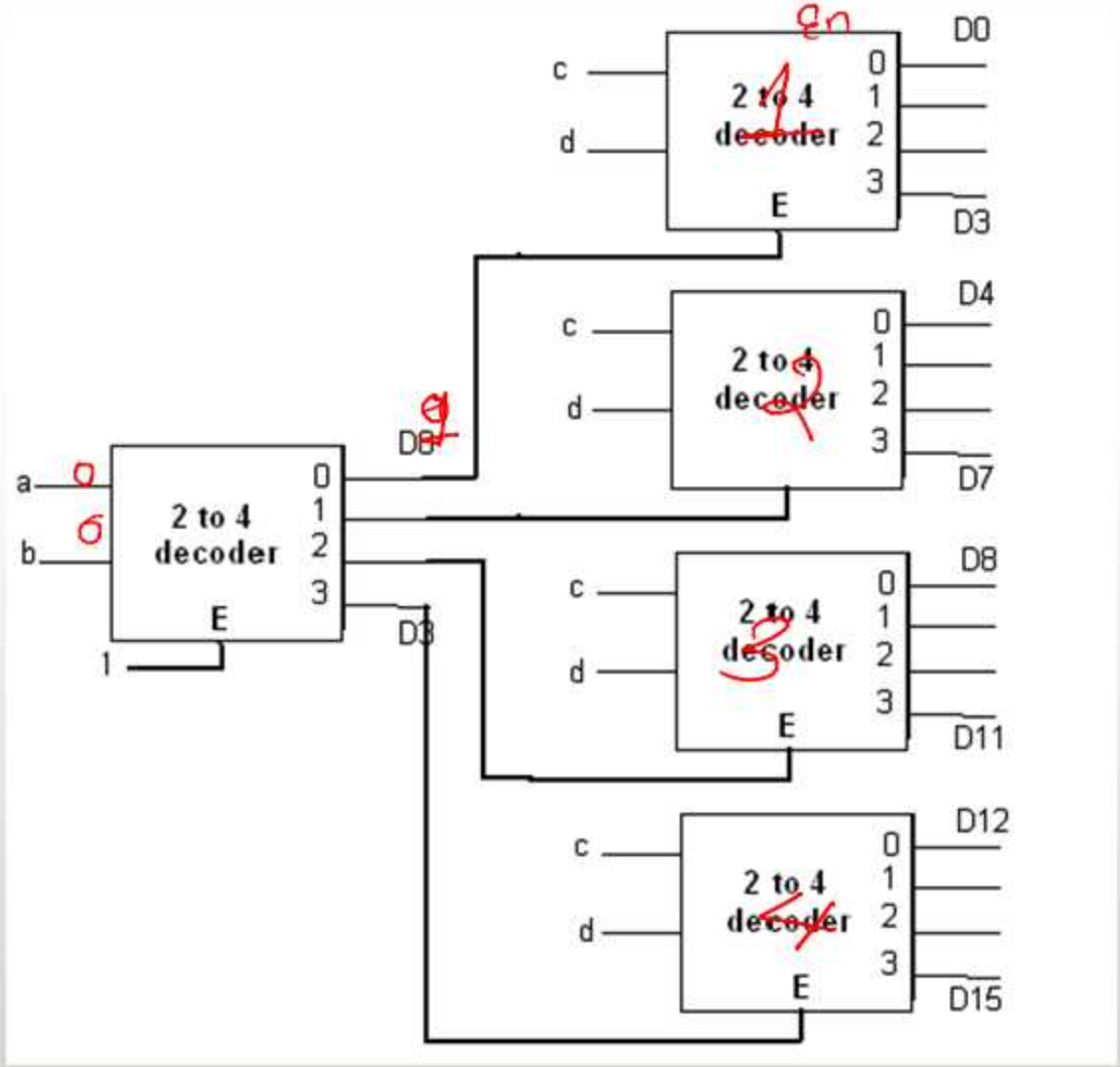
Realize the following Boolean expressions using the 4 to 10 line decoder.
f_1 (a, *b*, c, d) = Σ*m(1,2,4)*; f_2(*a, b,* c,d) = Σ*m(4,7,9)*;



Construct 4 to 16 decoder using 3 to 8 decoder



Construct 4 to 16 decoder using 2 to 4 decoder

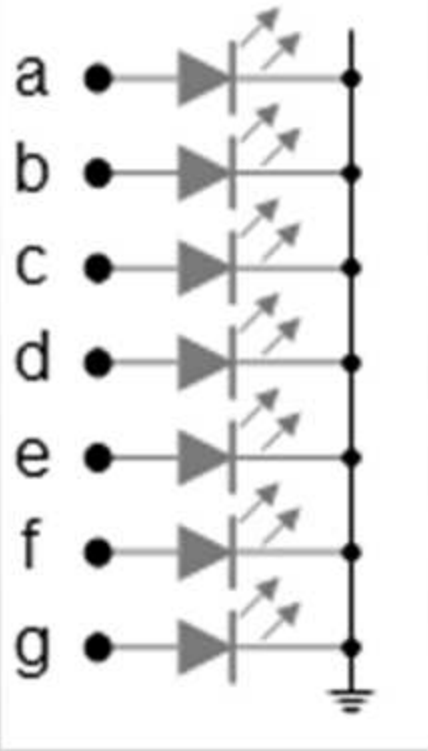


Seven-segment display

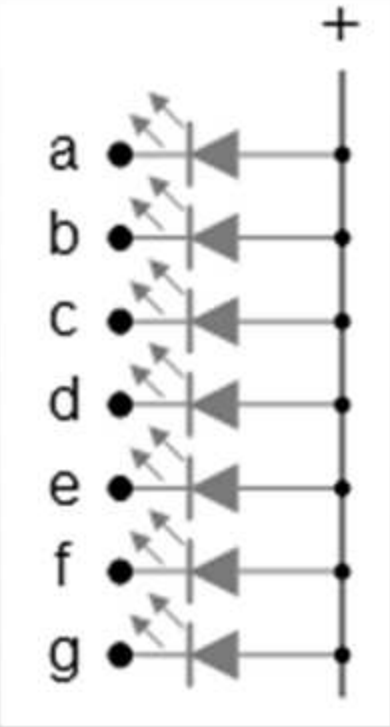
- An electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in a some definite pattern (common cathode or common anode type), which is used to display BCD numerals.
- Two types of 7-segment LED digital display:
  1) *The Common Cathode Display (CCD)*
     - All the cathode connections of the led's are joined together to logic "0" or ground.
     - Individual segments are illuminated by application of a "high", logic "1" signal to the individual anode terminals.
  2) *The Common Anode Display (CAD)*
     - All the anode connections of the led's are joined together to logic "1"
     - Individual segments are illuminated by connecting the individual cathode terminals to a "LOW", logic "0" signal.

Seven-segment display

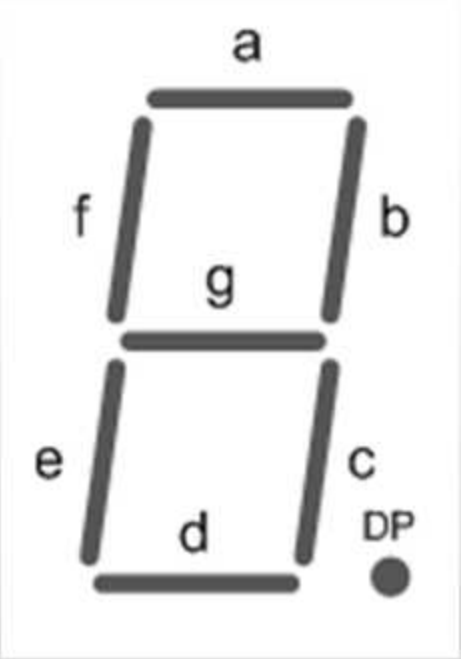
Common Cathode



Common Anode

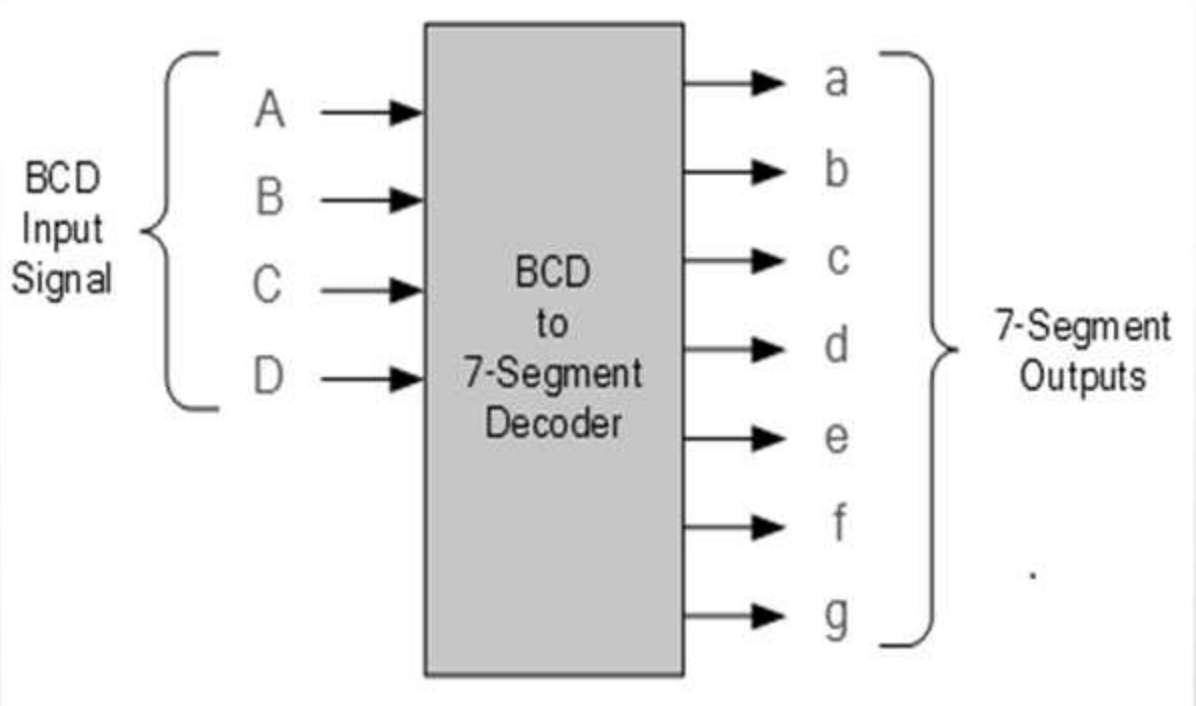


7-Segment Display Format



Seven-segment decoder

Block Diagram



Truth table

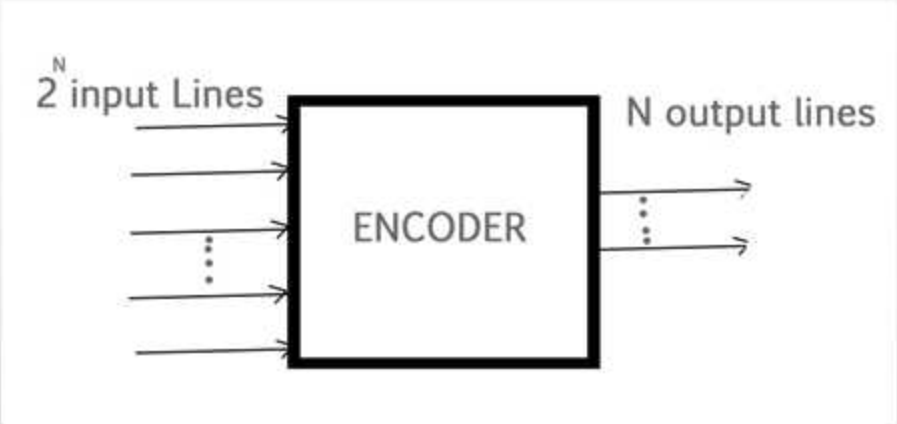
Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	

Summary

- Decoder definition
- Operation of 3 to 8 decoder, BCD to decimal decoder, seven segment decoder
- Cascading of decoder
- Function realization using decoder

Encoders

- An Encoder is a *combinational circuit* that performs the inverse function of a decoder.
- An encoder converts an active input signal into a coded output signal.
- It has maximum of 2^n input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code.

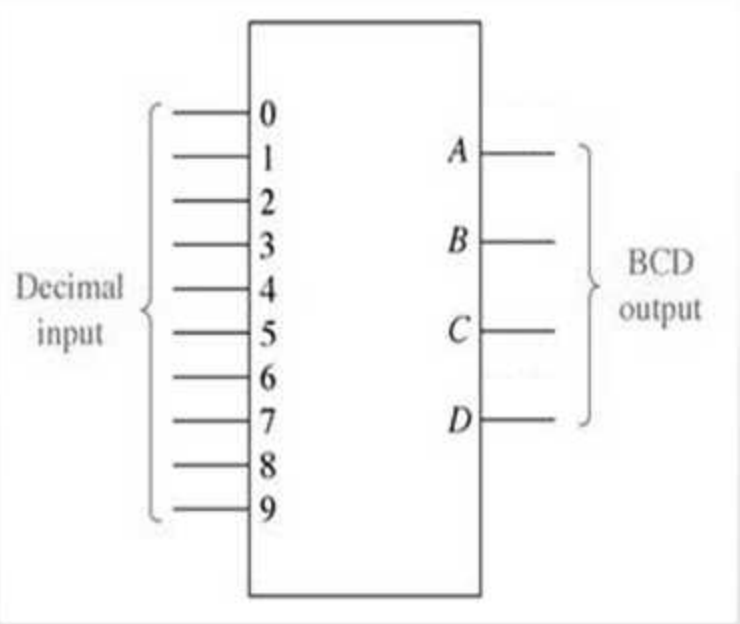


Decimal-to-BCD Encoder

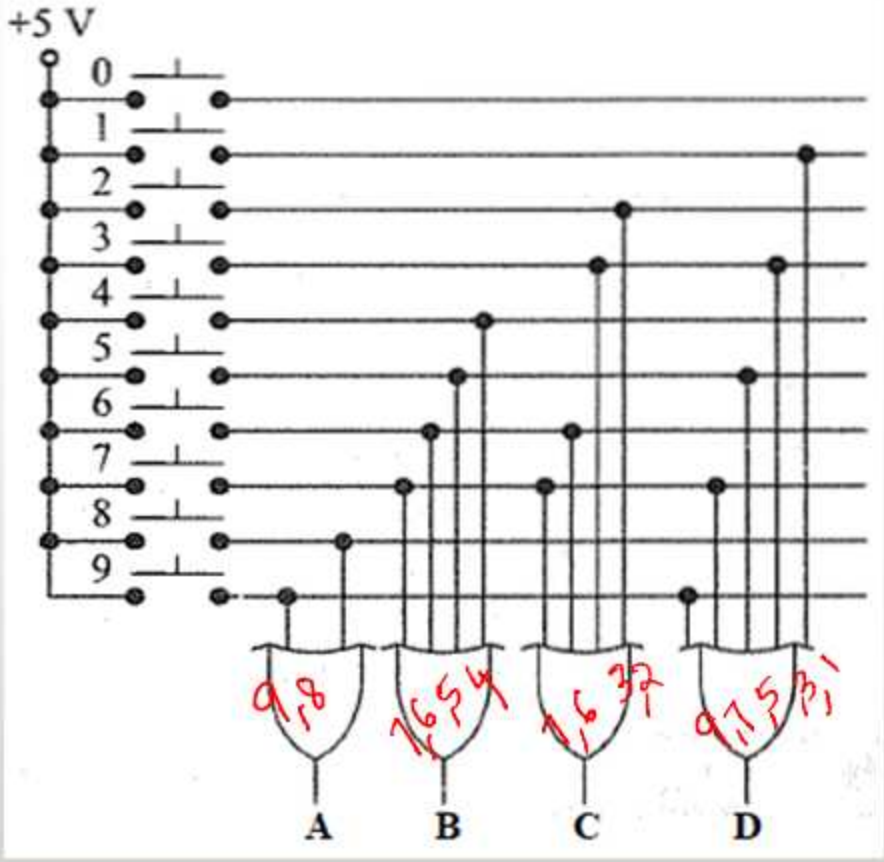
- Called as 10-line to 4-line encoder.
- It accepts 10 inputs and produces a 4-bit output corresponding to the activated **decimal** input.

Decimal-to-BCD Encoder

Block Diagram



Logic Diagram



Truth Table

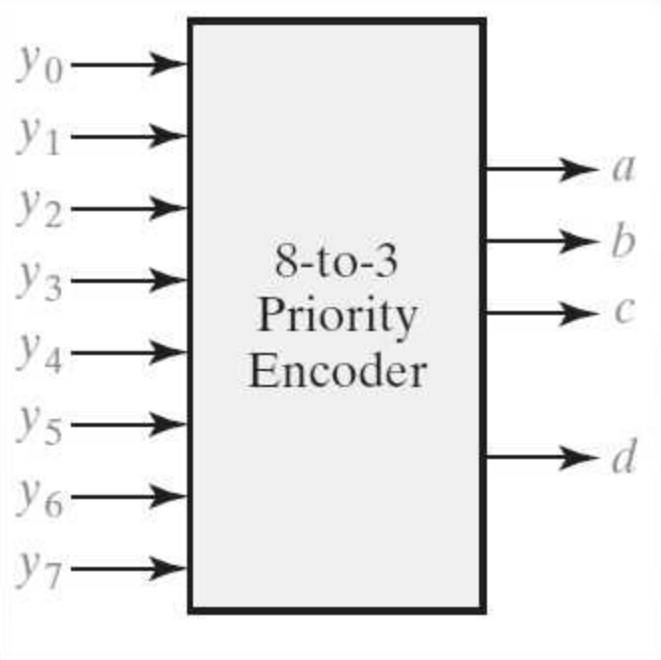
Decimal Digit	BCD Code			
	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

8-to-3 Priority Encoder

- Take all of their data inputs one at a time and converts them into an equivalent binary code at its output.
- It generates an output code based on the highest prioritized input.
- If two or more inputs are equal to 1 at the same time, the input having highest priority takes the precedence.

8-to-3 Priority Encoder

Block Diagram



Truth Table

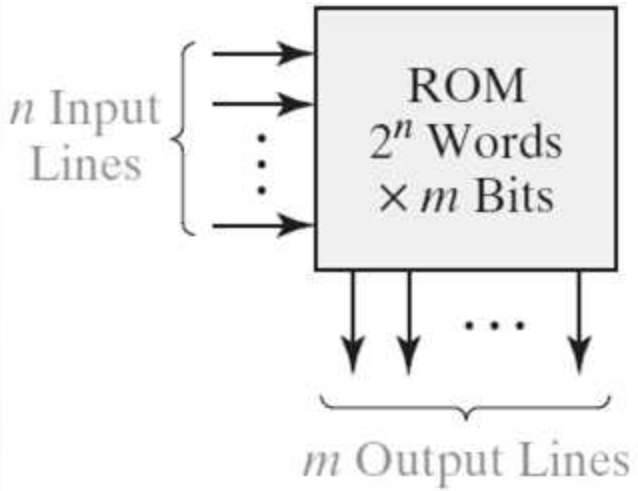
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	v
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

Read Only Memories/ ROM

- Consists of an array of semiconductor devices that are interconnected to store an array of binary data.
- Binary data is stored can be read out whenever desired, but the data that is stored cannot be changed.
- A ROM which has *n* input lines and *m* output lines contains an array of 2^n words, and each word is m bits long. The input lines serve as an address to select one of the 2^n words.
- A ROM basically consists of a decoder and a memory array.
- A $2^n \times m$ ROM can realize m functions of n variables because it can store a truth table with 2^n rows and m columns.

Read Only Memories/ ROM

ROM with *n* Inputs & *m* Outputs

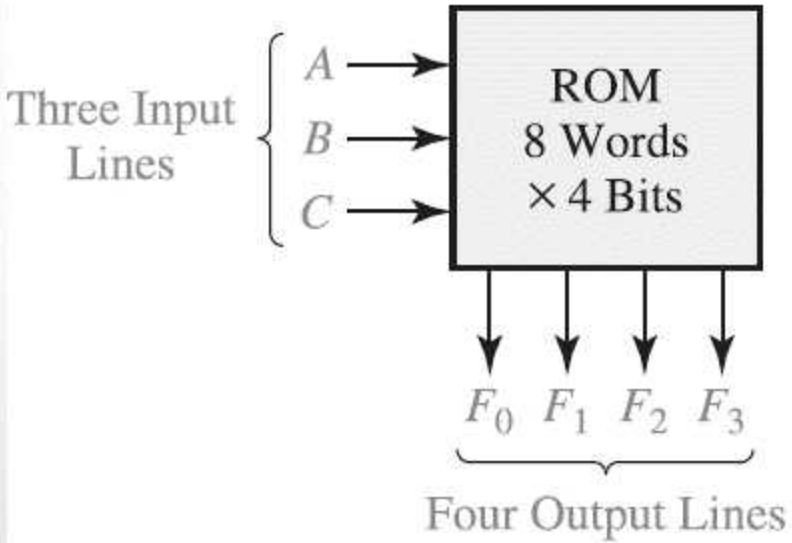


n Input Variables	*m* Output Variables
00 · · · 00	100 · · · 110
00 · · · 01	010 · · · 111
00 · · · 10	101 · · · 101
00 · · · 11	110 · · · 010
⋮	⋮
11 · · · 00	001 · · · 011
11 · · · 01	110 · · · 110
11 · · · 10	011 · · · 000
11 · · · 11	111 · · · 101

Typical Data Array Stored in ROM (2^n words of m bits each)

Read Only Memories/ ROM

An 8-Word × 4-Bit ROM



(a) Block diagram

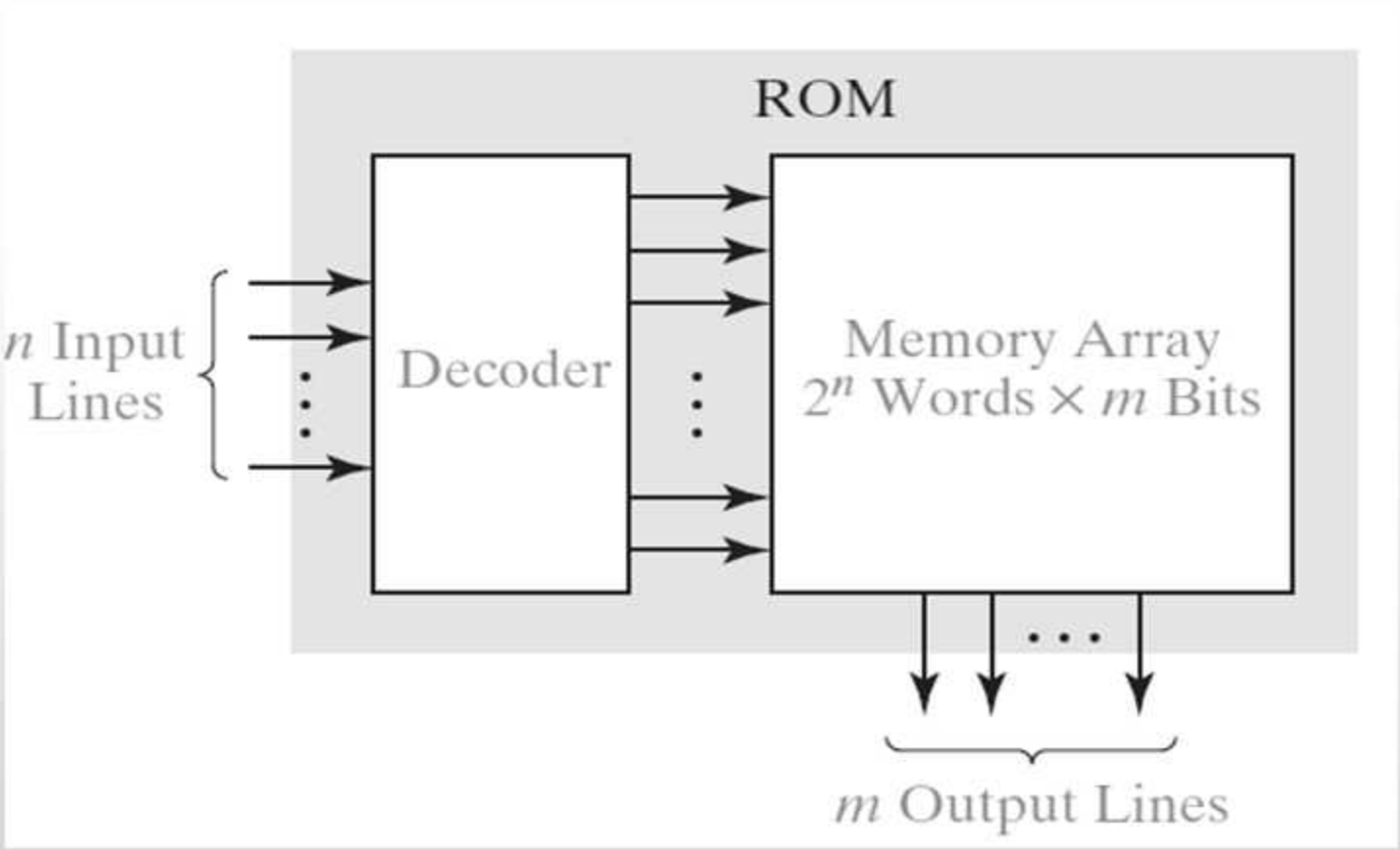
A B C	F_0 F_1 F_2 F_3
0 0 0	1 0 1 0
0 0 1	1 0 1 0
0 1 0	0 1 1 1
0 1 1	0 1 0 1
1 0 0	1 1 0 0
1 0 1	0 0 0 1
1 1 0	1 1 1 1
1 1 1	0 1 0 1

Typical Data Stored in ROM (2^3 words of 4 bits each)

(b) Truth table for ROM

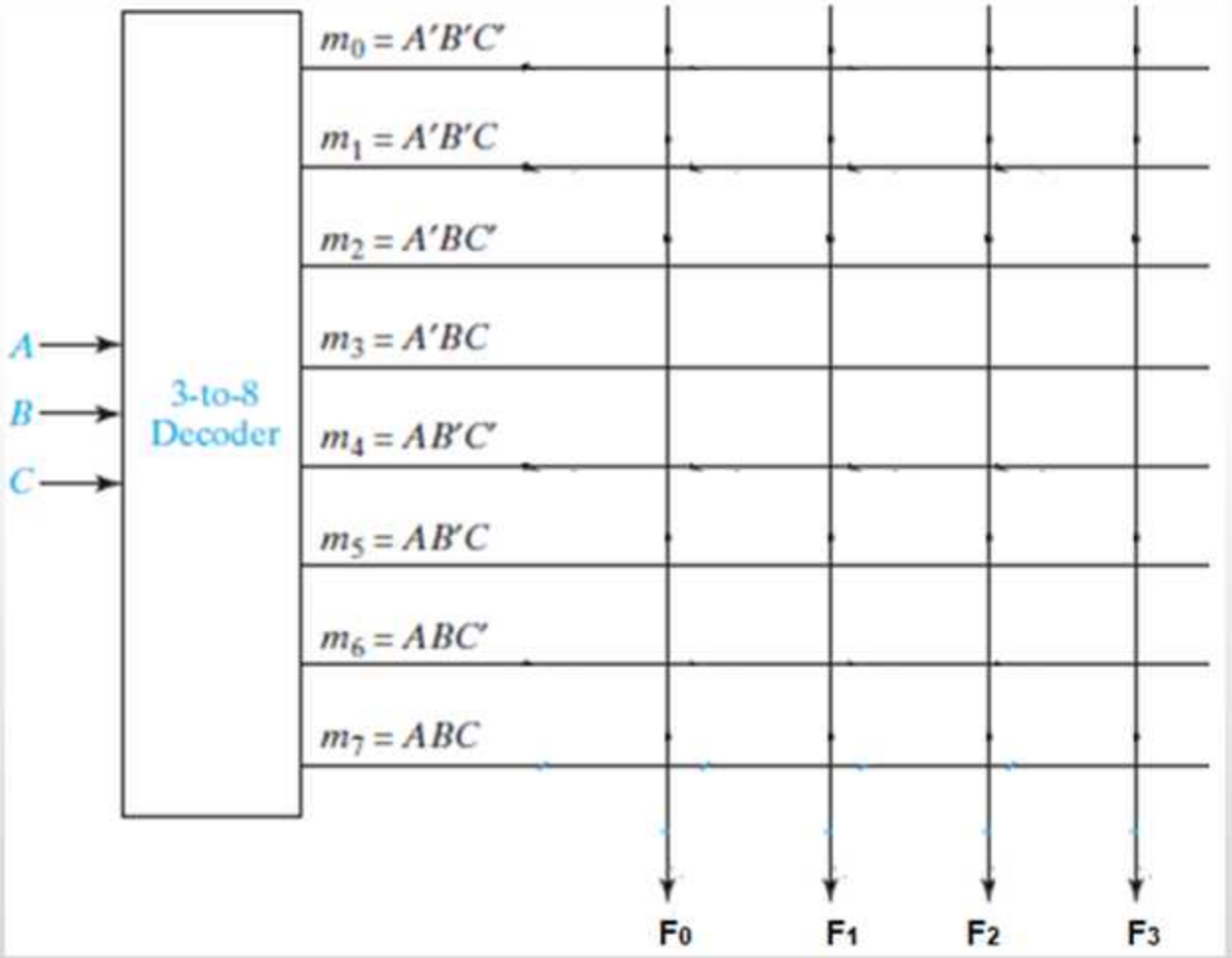
Read Only Memories/ ROM

Basic ROM Structure



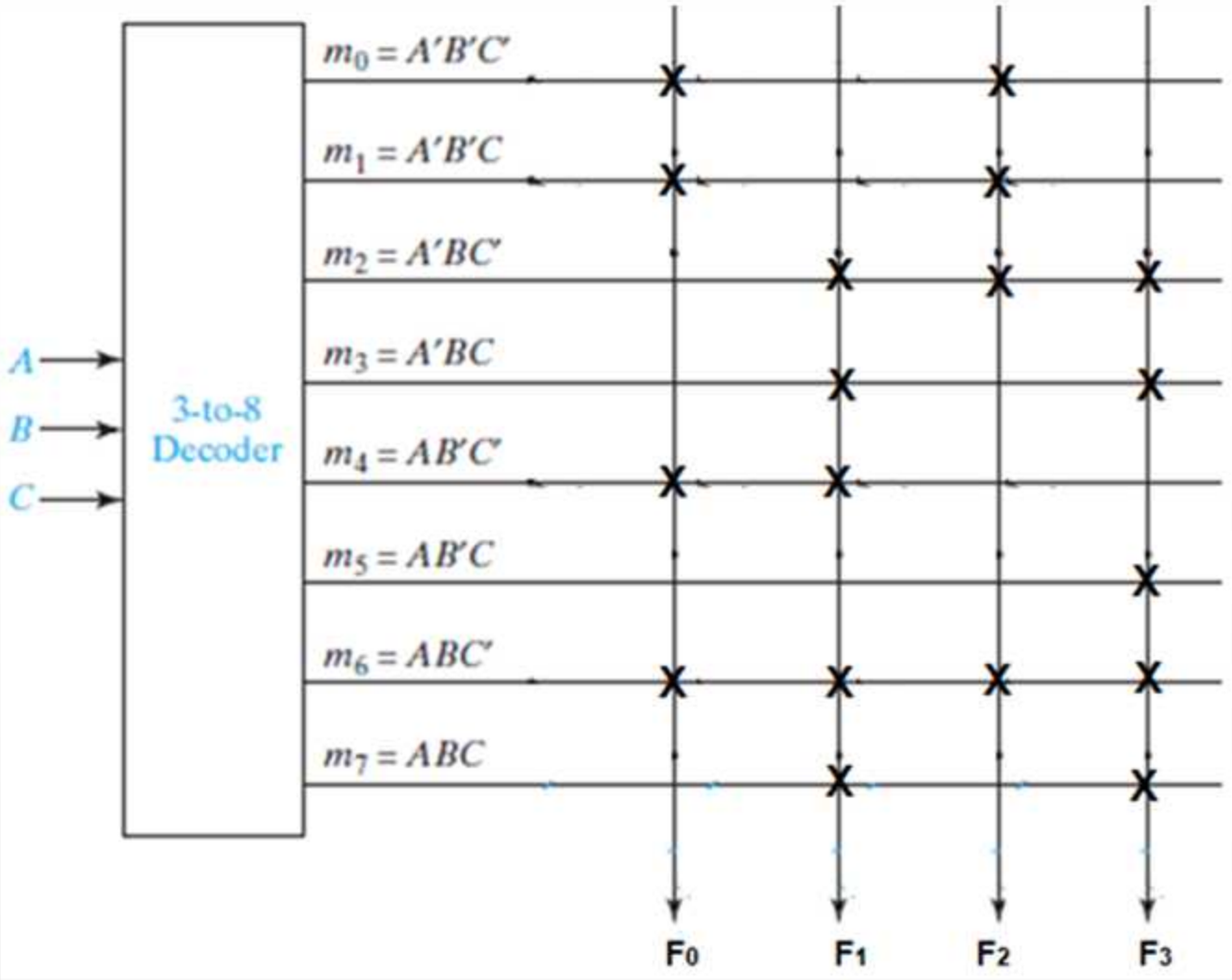
Implement the following functions using an 8-Word × 4-Bit ROM

$F_0 = \Sigma m(0, 1, 4, 6)$, $F_1 = \Sigma m(2, 3, 4, 6, 7)$, $F_2 = \Sigma m(0, 1, 2, 6)$, $F_3 = \Sigma m(2, 3, 5, 6, 7)$



Implement the following functions using an 8-Word × 4-Bit ROM

$F_0 = \Sigma m(0, 1, 4, 6)$, $F_1 = \Sigma m(2, 3, 4, 6, 7)$, $F_2 = \Sigma m(0, 1, 2, 6)$, $F_3 = \Sigma m(2, 3, 5, 6, 7)$



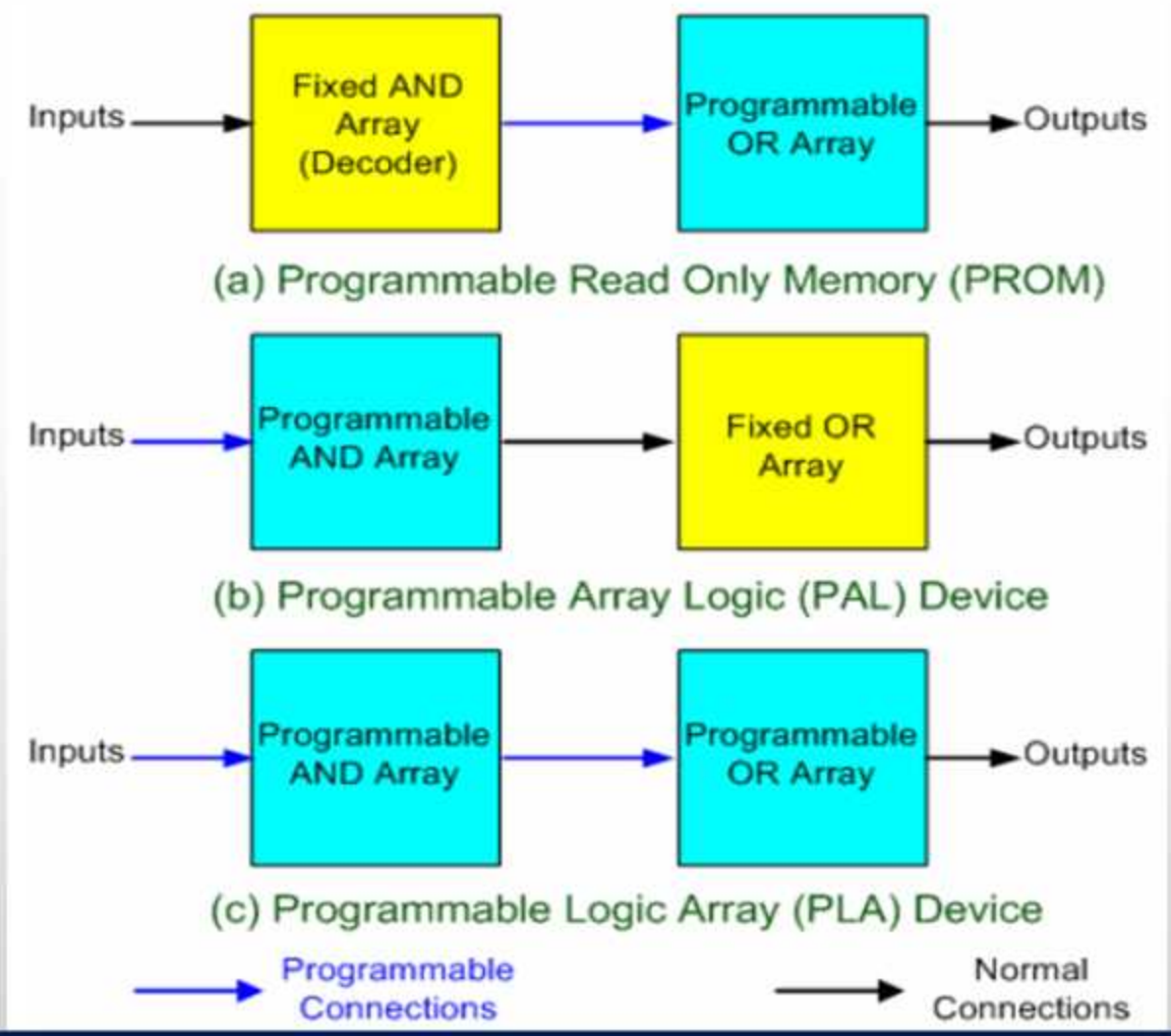
Summary

- Encoder definition
- Operation of decimal to BCD encoder, 8 to 3 priority encoder
- ROM structure
- Function realization using ROM

Programmable Logic Device/ PLD

- A logic device capable of being programmed to provide a variety of different logic functions.
- Contain an array of AND gates & another array of OR gates.
- Three fundamental types of PLDs:
  - Programmable Read Only Memory - PROM
  - Programmable Array Logic - PAL
  - Programmable Logic Arrays - PLA

Programmable Logic Device/ PLD

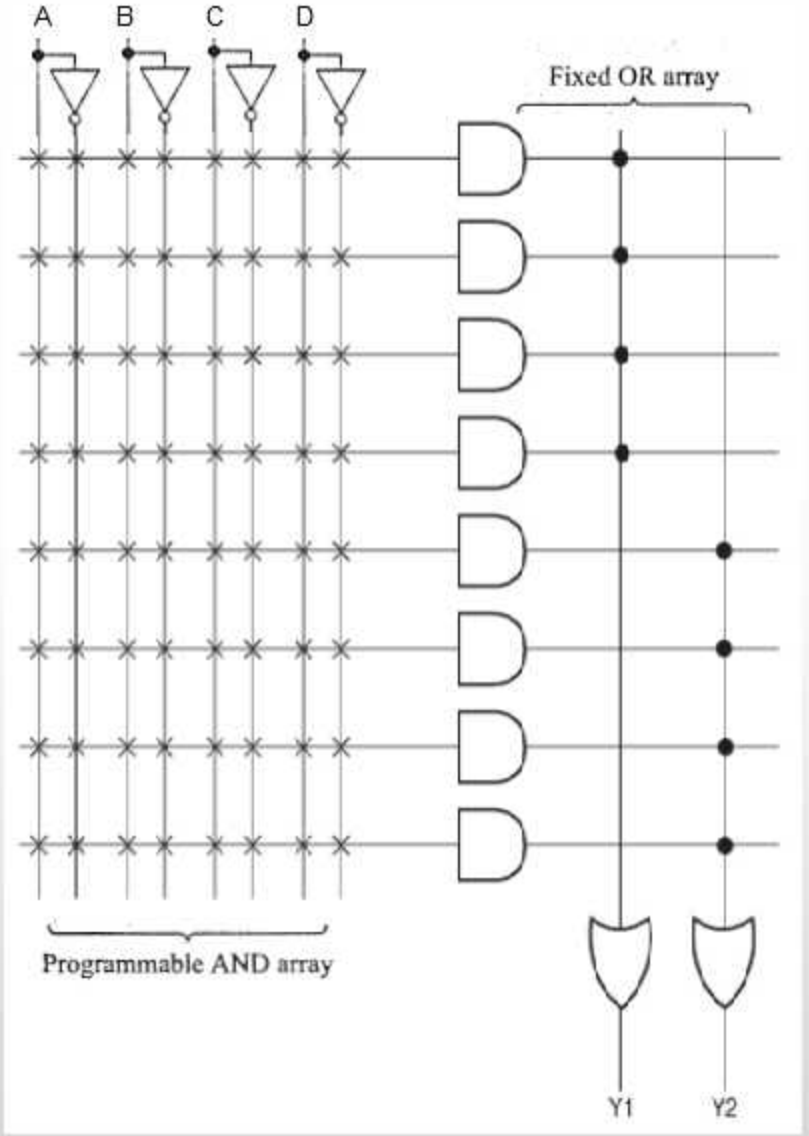


Programmable Array Logic/ PAL

- Is a programmable logic device that has a programmable AND array and a fixed OR array.
- Inputs of AND gates are programmable here.
- Can generate only the required product terms
- Inputs of the OR gates are not of programmable type.
- Number of inputs to each of the OR gates is of fixed type.
- Output of PAL is represented as a sum of product terms.

Programmable Array Logic/ PAL

- A PAL with 4 inputs and 2 outputs.



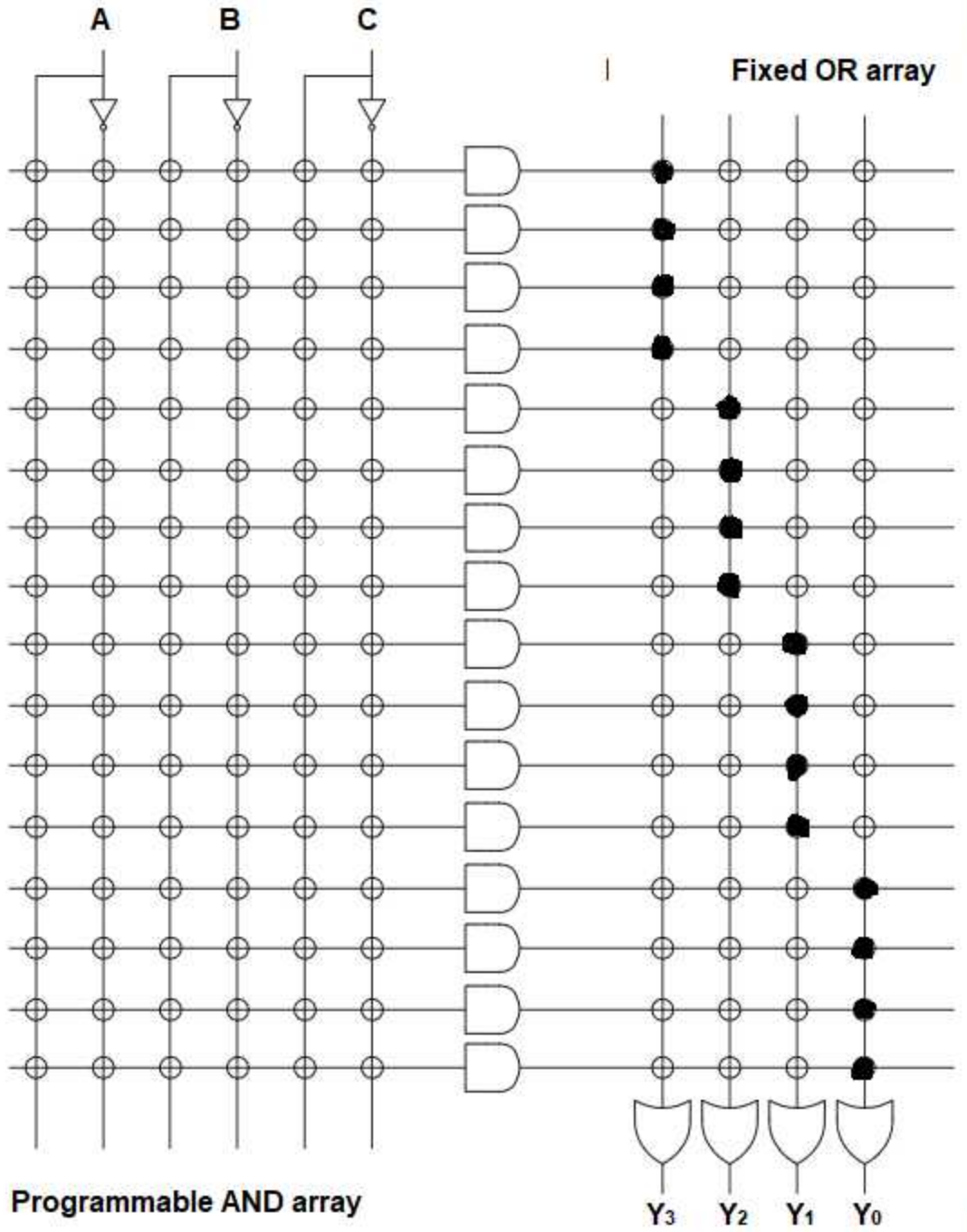
Implement following Boolean functions using PAL.

$$Y_3 = \bar{A}B\bar{C}$$

$$Y_2\ A\bar{B}C + ABC$$

$$Y_1 = A\bar{B}C + \bar{A}BC + ABC$$

$$Y_0 = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$$

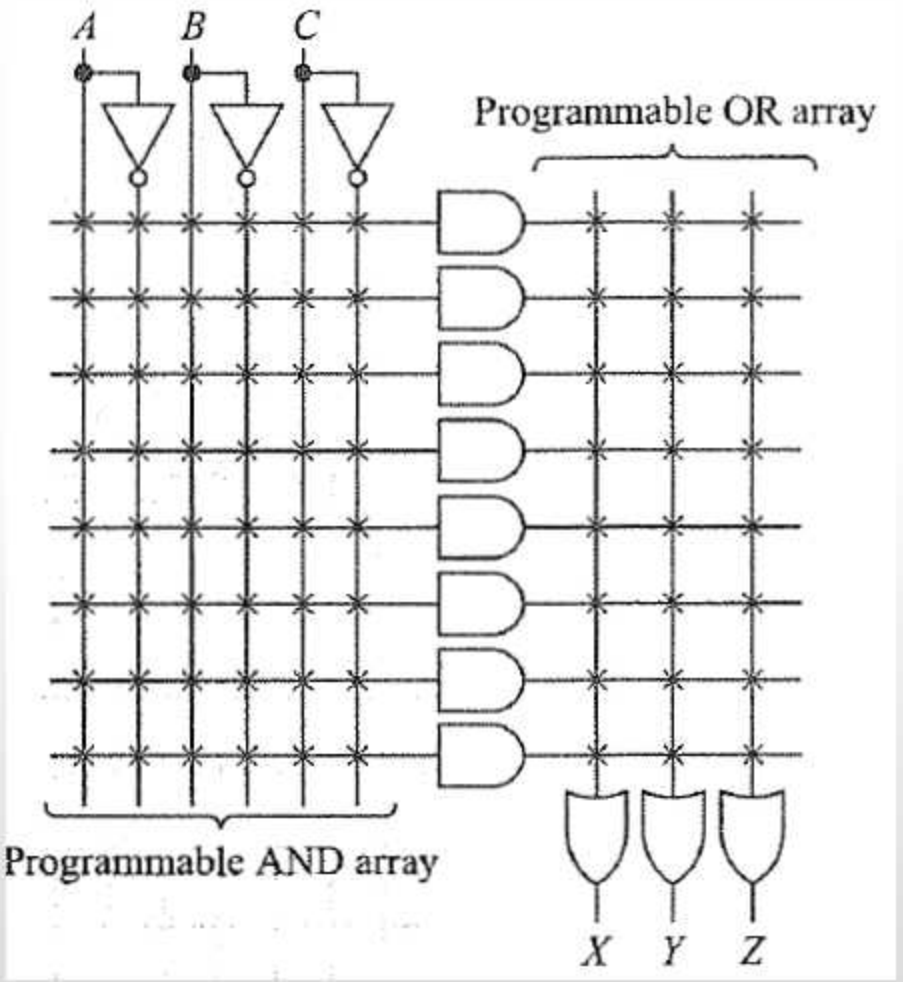


Programmable Logic Array / PLA

- Is a programmable logic device that has a programmable AND array and a programmable OR array.
- Inputs of the AND gates are programmable.
- Can generate the required **product terms** using the AND Array.
- Inputs of the OR gates are also programmable.
- Output of PAL will be as a **sum of product terms**.

Programmable Logic Array / PLA

- A PLA with 3 inputs and 3 outputs.



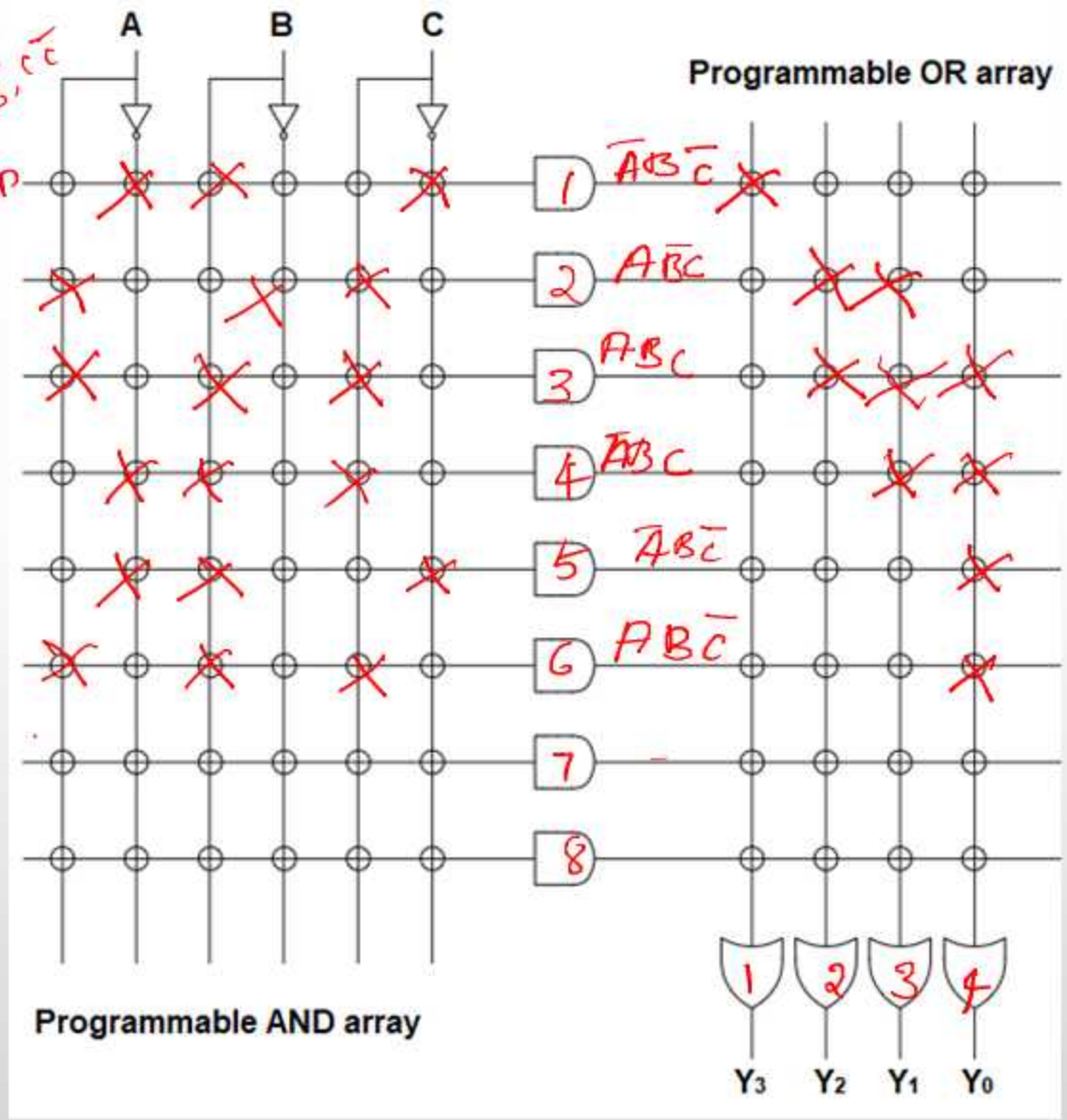
Implement following Boolean functions using PLA.

$$Y_3 = \bar{A}B\bar{C}$$

$$Y_2 = A\bar{B}\bar{C} + AB\bar{C}$$

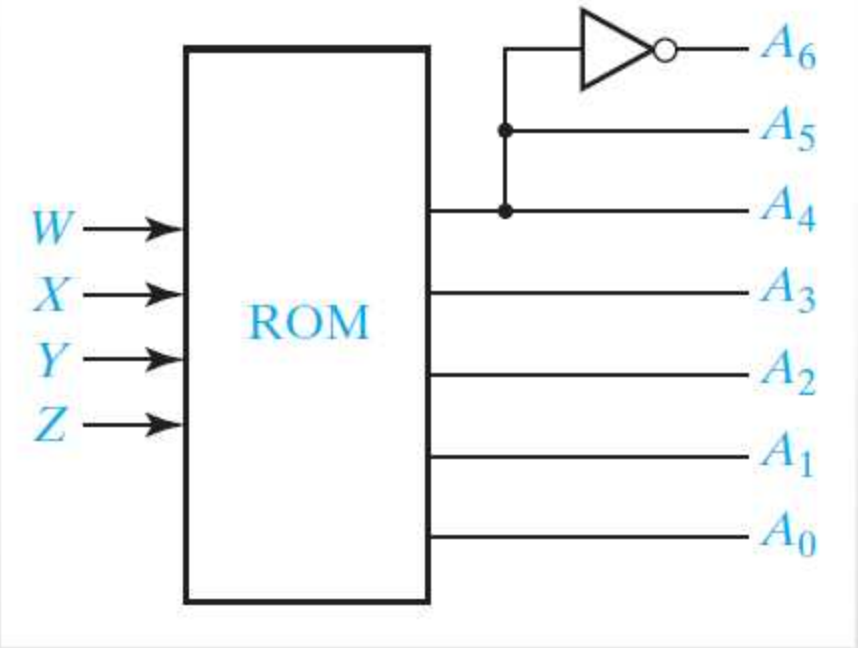
$$Y_1 = A\bar{B}C + \bar{A}B\bar{C} + ABC$$

$$Y_0 = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$$



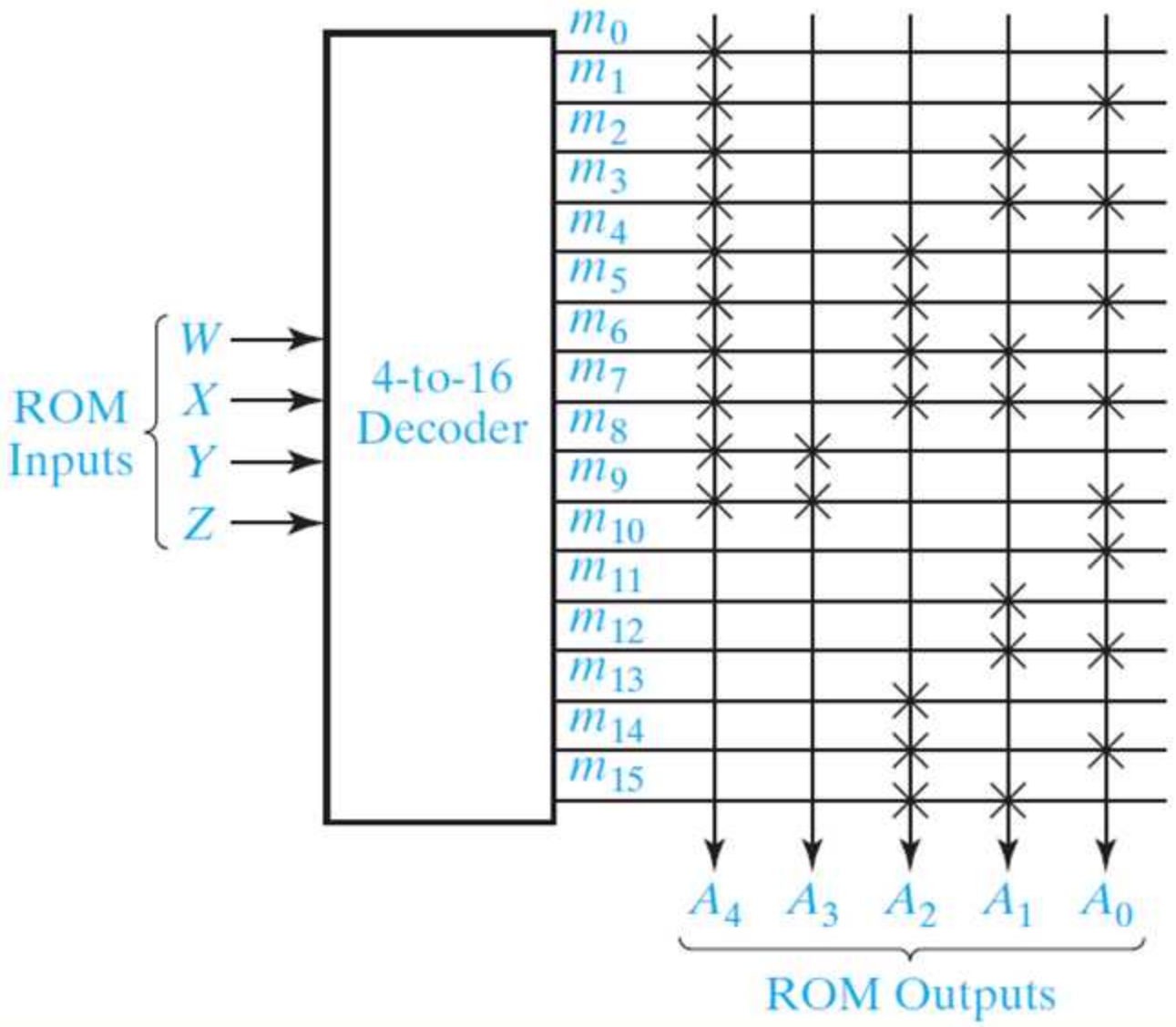
Realize Hexadecimal to ASCII Code Converter using ROM

W	X	Y	Z	Digit	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1
0	0	1	0	2	0	1	1	0	0	1	0
0	0	1	1	3	0	1	1	0	0	1	1
0	1	0	0	4	0	1	1	0	1	0	0
0	1	0	1	5	0	1	1	0	1	0	1
0	1	1	0	6	0	1	1	0	1	1	0
0	1	1	1	7	0	1	1	0	1	1	1
1	0	0	0	8	0	1	1	1	0	0	0
1	0	0	1	9	0	1	1	1	0	0	1
1	0	1	0	A	1	0	0	0	0	0	1
1	0	1	1	B	1	0	0	0	0	1	0
1	1	0	0	C	1	0	0	0	0	1	1
1	1	0	1	D	1	0	0	0	1	0	0
1	1	1	0	E	1	0	0	0	1	0	1
1	1	1	1	F	1	0	0	0	1	1	0



Realize Hexadecimal to ASCII Code Converter using ROM

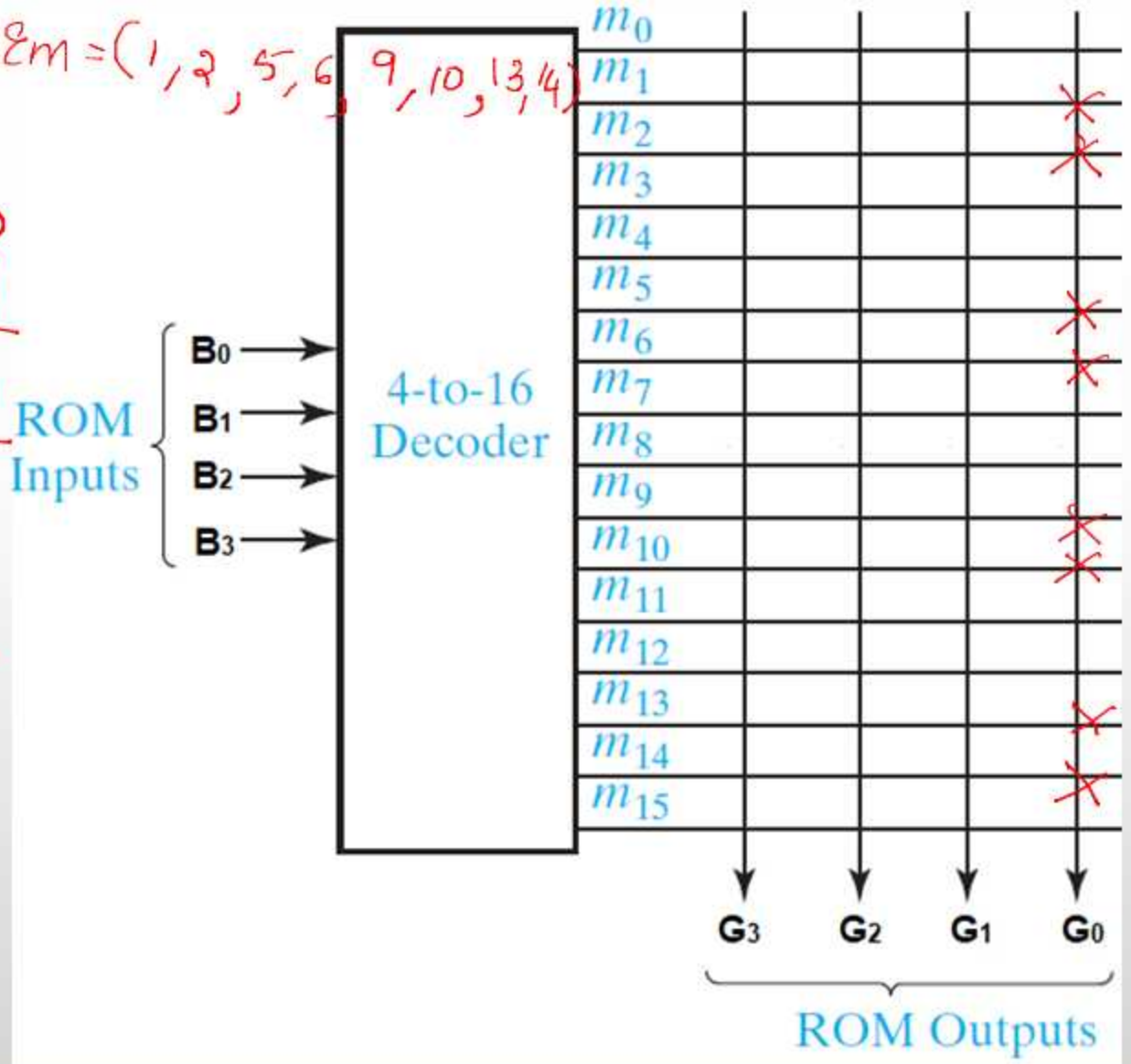
W	X	Y	Z	Digit	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1
0	0	1	0	2	0	1	1	0	0	1	0
0	0	1	1	3	0	1	1	0	0	1	1
0	1	0	0	4	0	1	1	0	1	0	0
0	1	0	1	5	0	1	1	0	1	0	1
0	1	1	0	6	0	1	1	0	1	1	0
0	1	1	1	7	0	1	1	0	1	1	1
1	0	0	0	8	0	1	1	1	0	0	0
1	0	0	1	9	0	1	1	1	0	0	1
1	0	1	0	A	1	0	0	0	0	0	1
1	0	1	1	B	1	0	0	0	0	1	0
1	1	0	0	C	1	0	0	0	0	1	1
1	1	0	1	D	1	0	0	0	1	0	0
1	1	1	0	E	1	0	0	0	1	0	1
1	1	1	1	F	1	0	0	0	1	1	0



Realize Binary to Gray Code Converter using ROM

4 bit Binary Number $B_3B_2B_1B_0$	4 bit Gray Code $G_3G_2G_1G_0$
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

$G_0 = \Sigma m = (1, 2, 5, 6, 9, 10, 13, 14)$



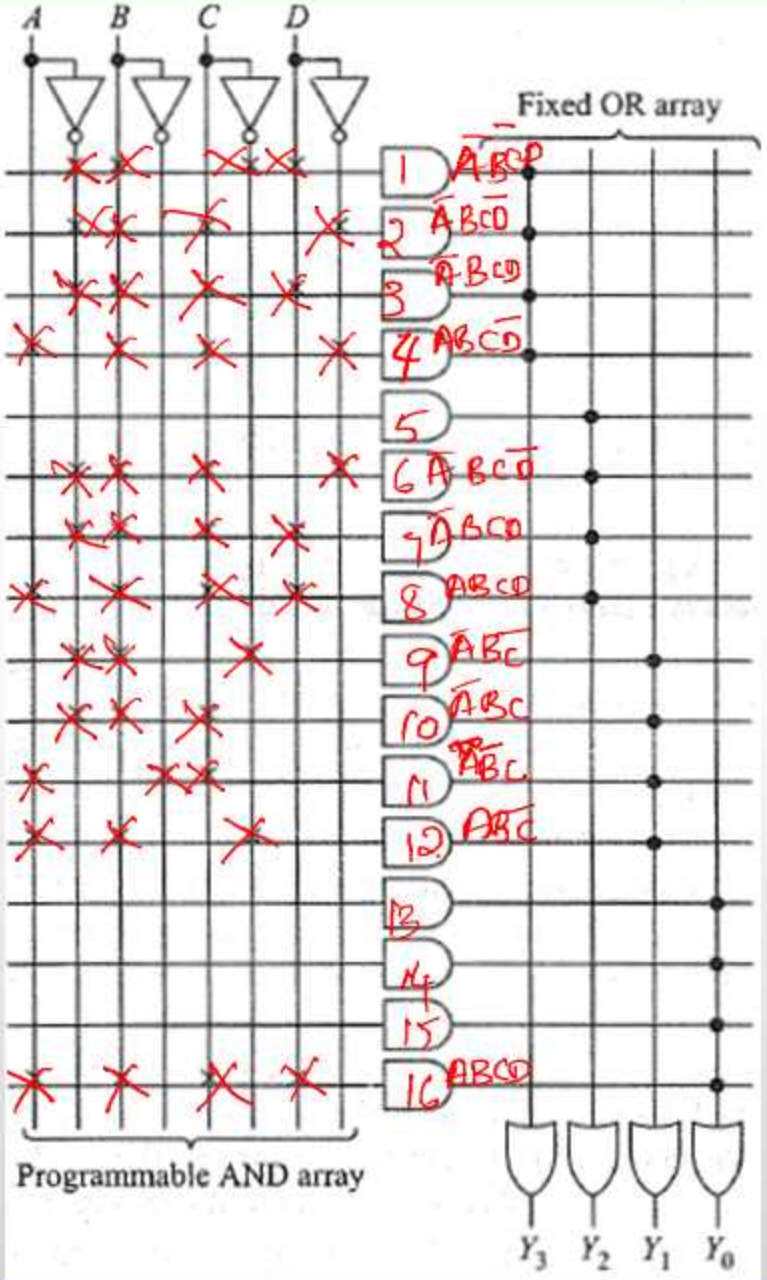
Implement following Boolean functions using PAL

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$

$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$

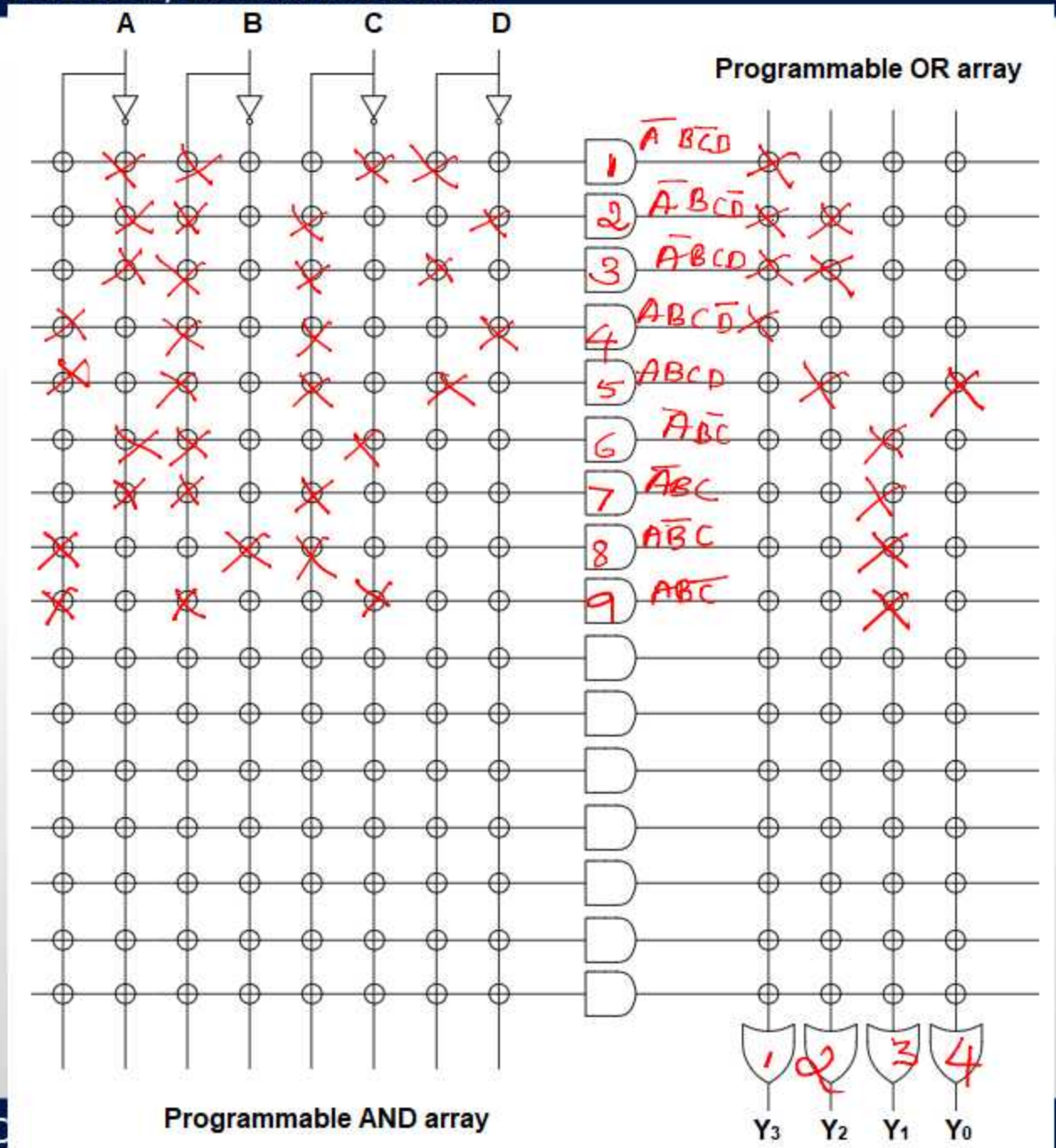
$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$

$$Y_0 = ABCD$$

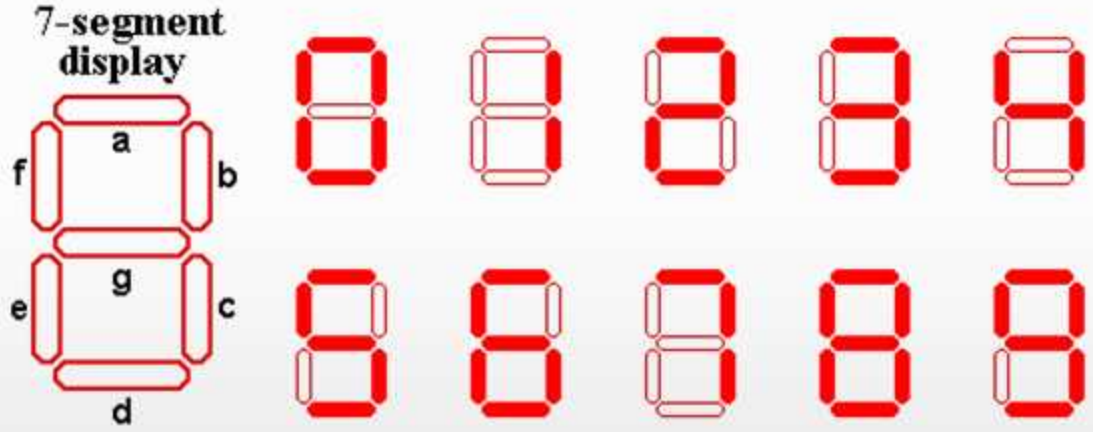


Implement following Boolean functions using PLA

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$
$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$
$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$
$$Y_0 = ABCD$$



Implement 7-segment decoder using PLA.



Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	

Implement 7-segment decoder using PLA.

SOP 0 - $\overline{V}$
1 - V

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9