Unit-4

Combinational Logic

Outline

- Introduction and Design Procedure;
- Adders;
- Subtractors;
- Binary Parallel Adders;
- Encoders and
- Decoders;
- Multiplexers and Demultiplexers;
- Read-Only Memory (ROM);
- Programmable Logic Array (PLA)

Introduction of Combinational Circuit

- A combinational circuit is the digital logic circuit in which the output depends on the combination of inputs at that point of time with total disregard to the past state of the inputs.
- It is made from digital logic gates.

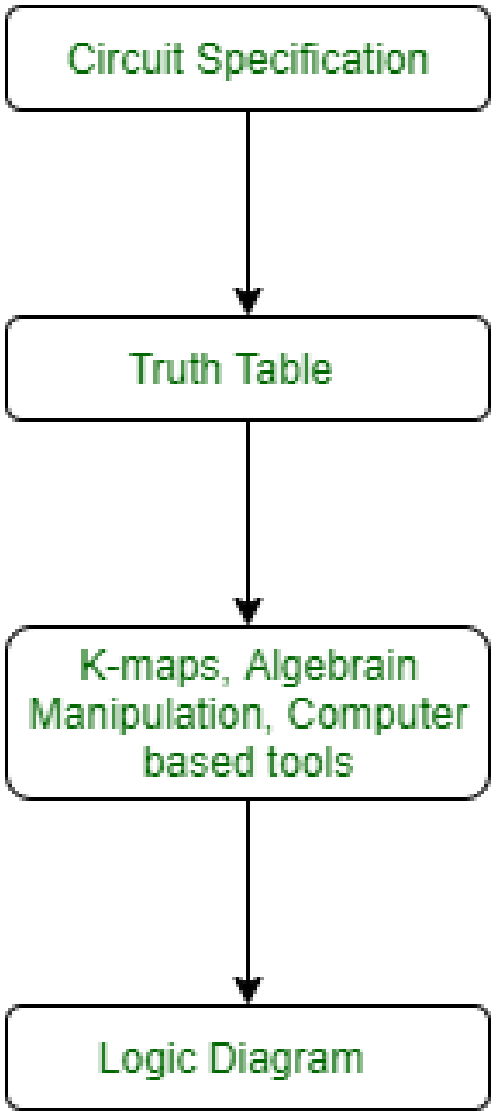
CHARACTERISTICS

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

Design Procedure of Combinational Circuit

To design of combinational circuits, the procedure involves the following steps:

1. Find the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table according to given specifications and function.
3. Using the truth table, obtain simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic circuit diagram.

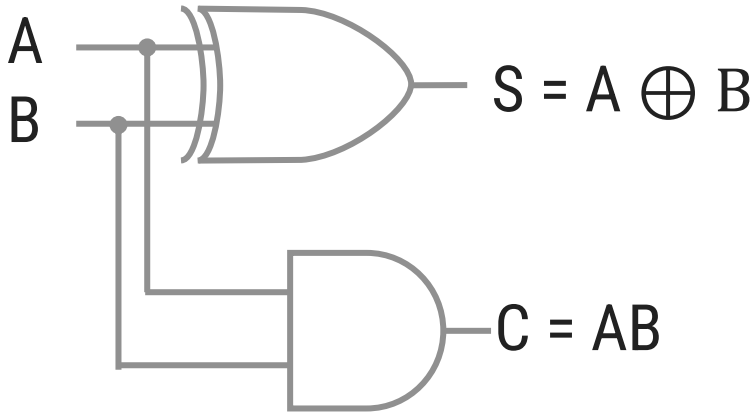


Adders, Subtractors

Half Adder

- A combinational circuit which adds two one-bit binary numbers is called a half-adder.

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- The sum column resembles like an output of the XOR gate.
- The carry column resembles like an output of the AND gate.
- Limitations:
  - In multi-digit addition we have to add two bits along with the carry of previous digit addition. Such addition requires addition of 3 bits. This is not possible in half-adders.

Full Adder

- The full-adder adds the bits A and B and the carry from the previous column called carry-in C_{in} and outputs the sum bit S and the carry bit called carry-out C_{out}.

Inputs			Outputs	
A	**B**	**C_{in}**	**S**	**C_{out}**
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

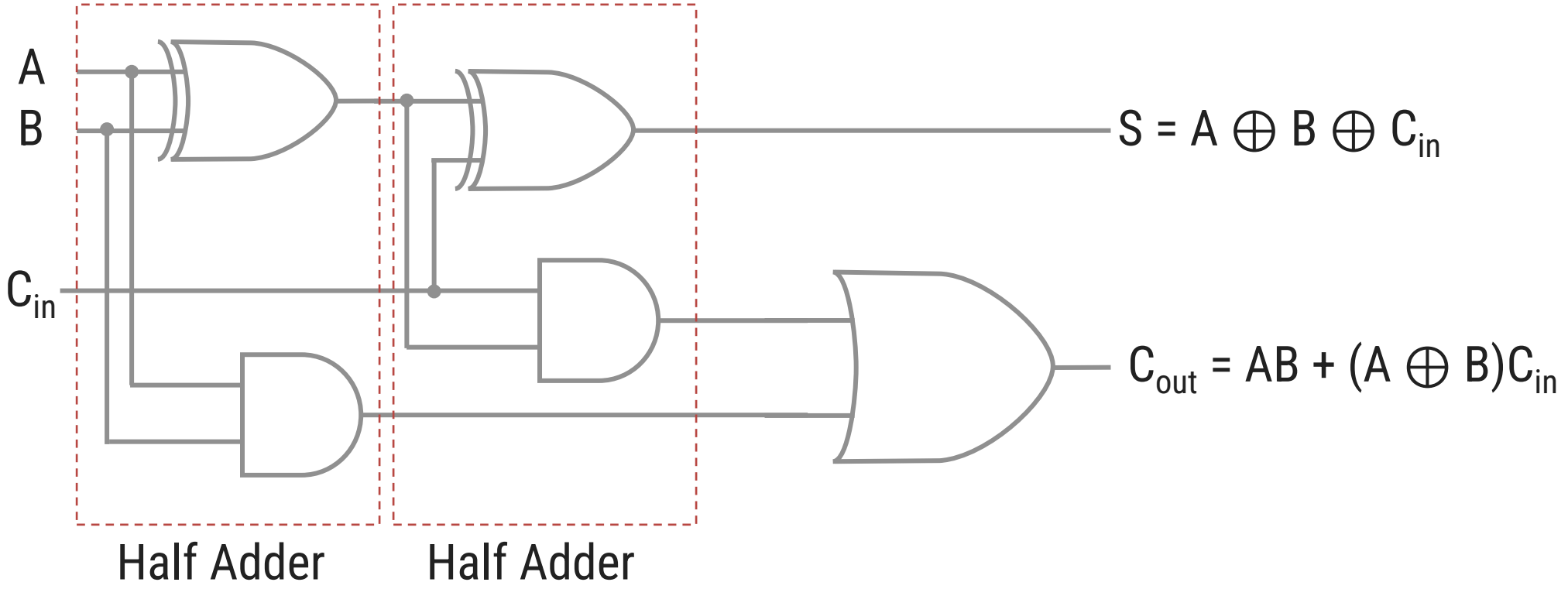
$$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$$
$$= (AB' + A'B)C_{in}' + (AB + A'B')C_{in}$$
$$= (A \oplus B)C_{in}' + (A \oplus B)'C_{in}$$
$$= A \oplus B \oplus C_{in}$$

$$C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$$
$$= AB + (A \oplus B)C_{in}$$

Full Adder – Logical Diagram

$S = A \oplus B \oplus C_{in}$

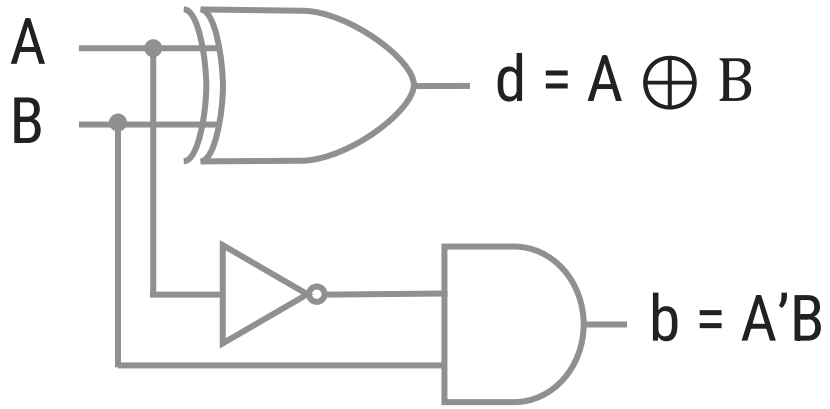
$C_{out} = AB + (A \oplus B)C_{in}$



Half Subtractor

- Subtracts one bit from the other and produces the difference.
- Other output is to specify if 1 is borrowed.

Inputs		Outputs	
A	**B**	**d**	**b**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



- Limitation:
  - Half subtractor can be used only for LSB subtraction.

Full Subtractor

- Full subtractor is a combinational circuit with 3 inputs (A, B, b_i)
- Subtraction = A – B – b_i

Inputs			Outputs	
A	**B**	**b_i**	**d**	**b**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

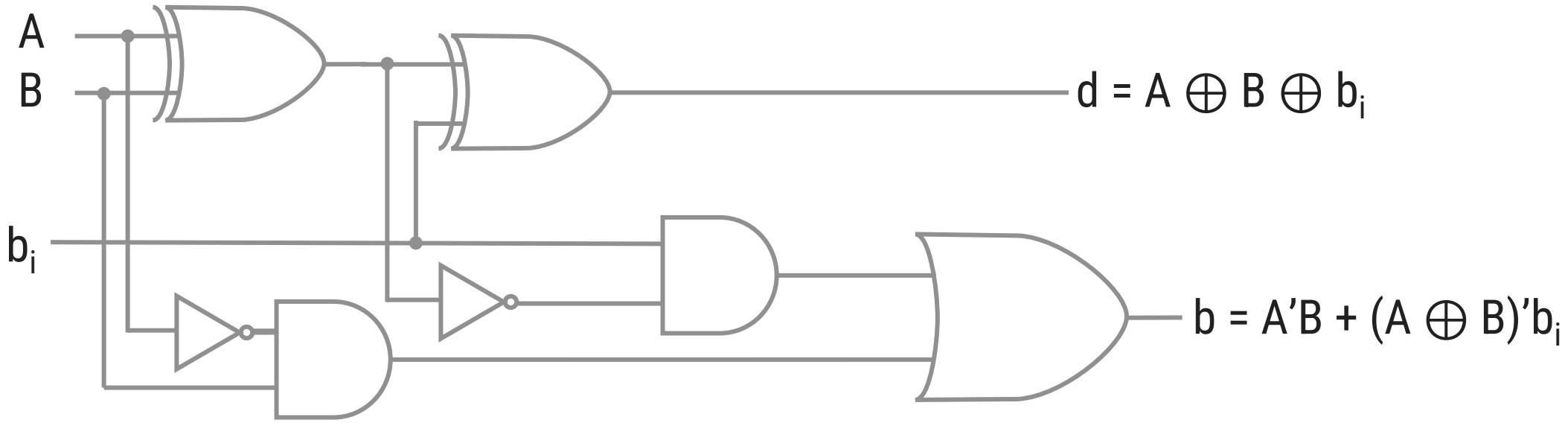
$$d = A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i$$
$$= (AB' + A'B)b_i' + (AB + A'B')b_i$$
$$= (A \oplus B)b_i' + (A \oplus B)'b_i$$
$$= A \oplus B \oplus b_i$$

$$b = A'B'b_i + A'Bb_i' + A'Bb_i + ABb_i$$
$$= A'B(b_i + b_i') + (AB + A'B')b_i$$
$$= A'B + (A \oplus B)'b_i$$

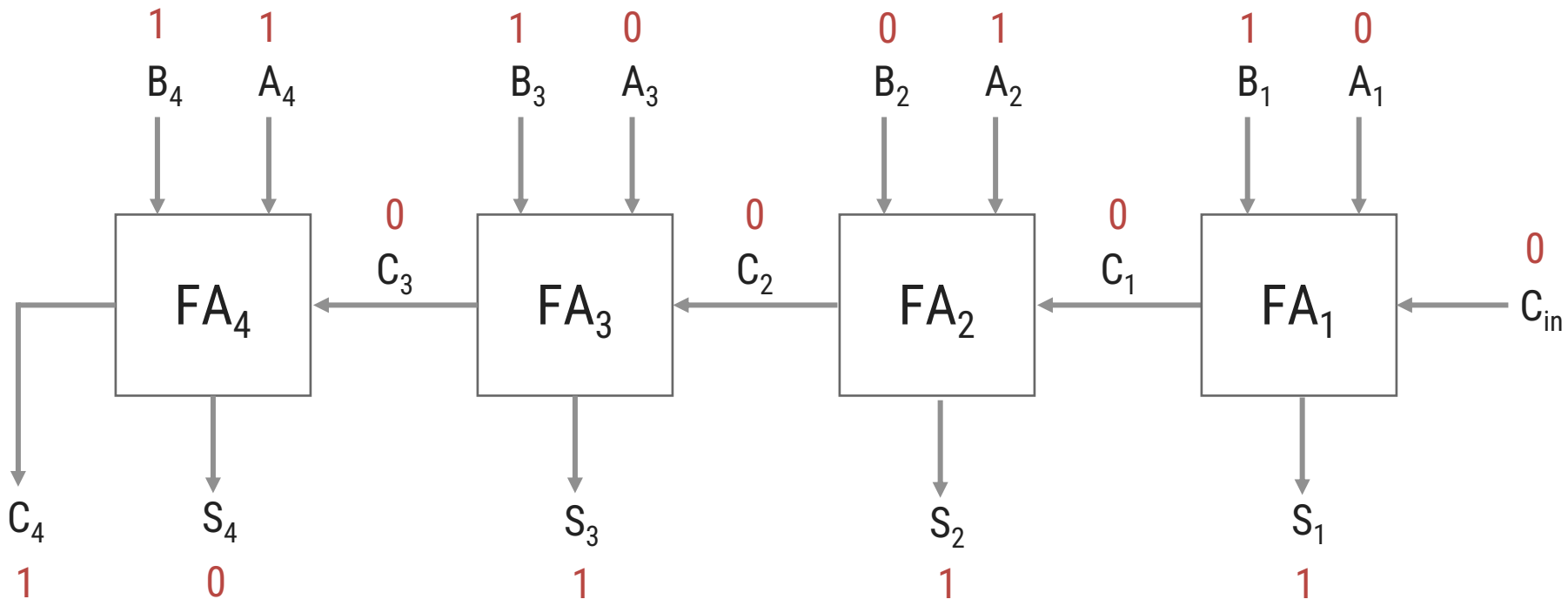
Full Subtractor – Logical Diagram

$d = A \oplus B \oplus b_i$

$b = A'B + (A \oplus B)'b_i$



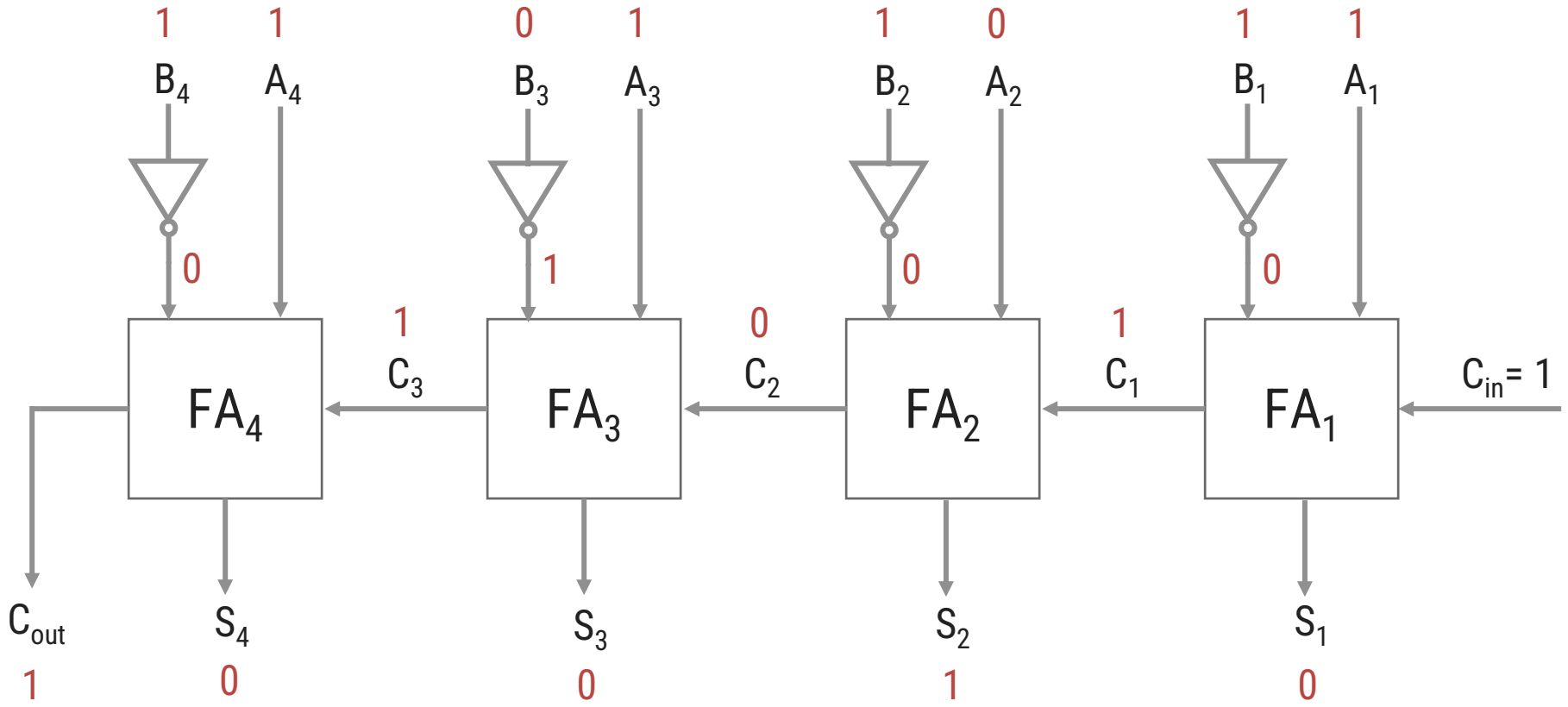
Binary Parallel Adder



Example: A = 1010, B = 1101

$A_4\ A_3\ A_2\ A_1$ = 1010
$B_4\ B_3\ B_2\ B_1$ = 1101

Binary Parallel Subtractor



Example: A = 1101, B = 1011

$A_4\ A_3\ A_2\ A_1$ = 1101
$B_4\ B_3\ B_2\ B_1$ = 1011

Binary to Gray Code Converter

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$

$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$

$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$

$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$

4-bit Binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
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

Binary to Gray Code Converter

- K-Map for G_4, G_3, G_2, G_1 function and their minimization are as follows:

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12: 1	8: 1
01	1	5	13: 1	9: 1
11	3	7	15: 1	11: 1
10	2	6	14: 1	10: 1

$G_4 = B_4$

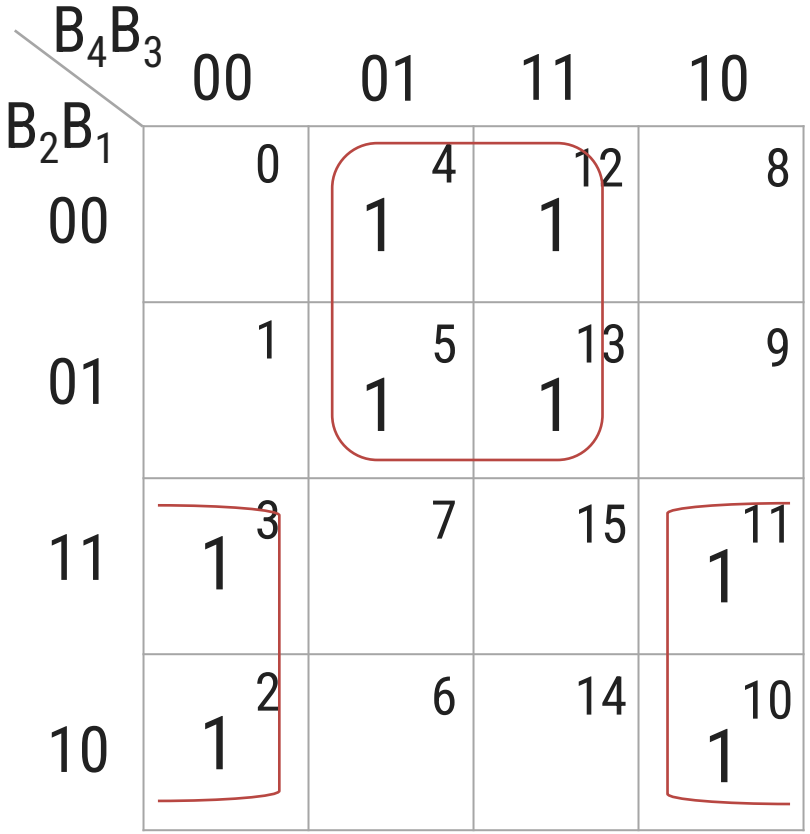
$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4: 1	12	8: 1
01	1	5: 1	13	9: 1
11	3	7: 1	15	11: 1
10	2	6: 1	14	10: 1

$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$

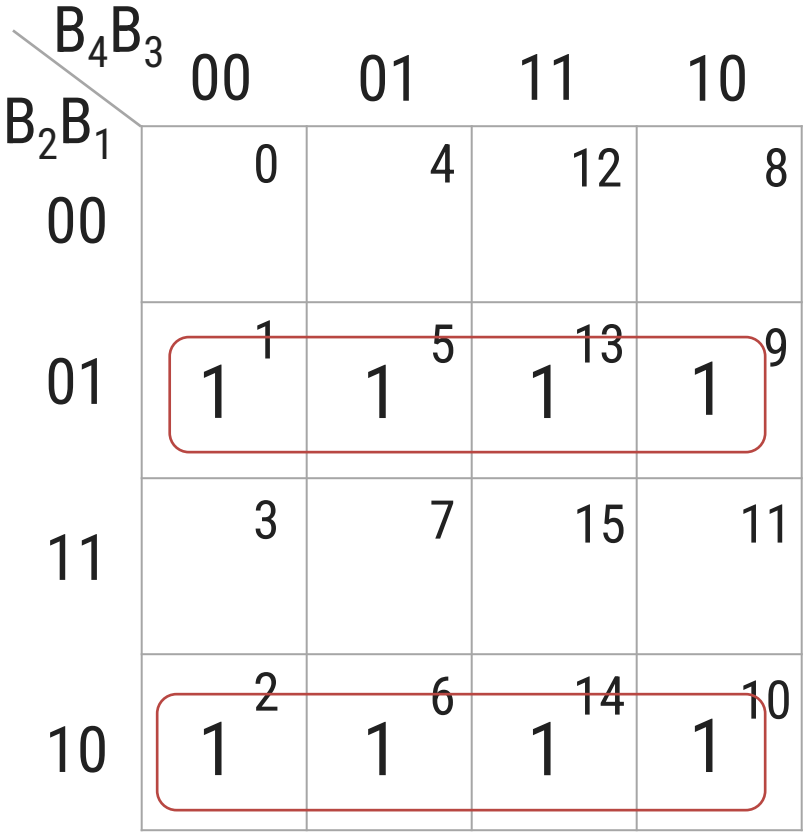
Binary to Gray Code Converter

$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$



$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$

$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$



$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$

Binary to Gray Code Converter

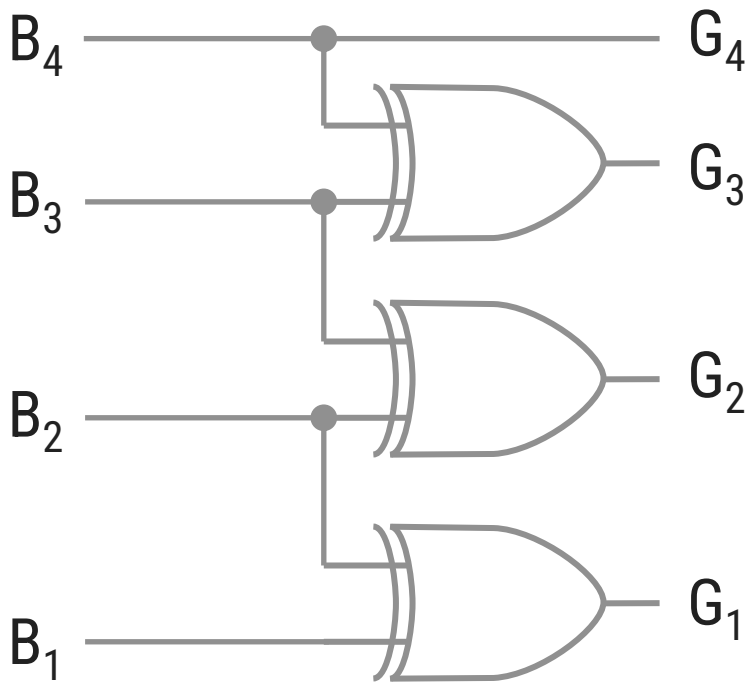
- Logic diagram for binary to Gray code converter is as follows:

$$G_4 = B_4$$

$$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$$

$$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$$

$$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$$



BCD to Excess-3 Code Converter

8421 BCD				XS - 3			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
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

$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$

$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$

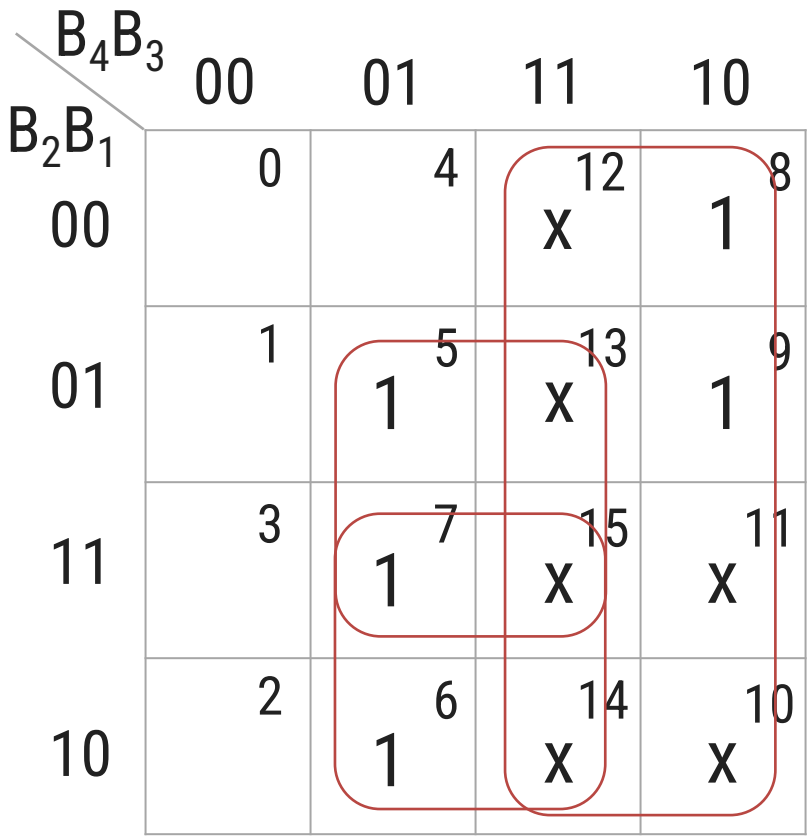
$$X_2 = \sum m(0,3,4,7,8) + d(10,11,12,13,14,15)$$

$$X_1 = \sum m(0,2,4,6,8) + d(10,11,12,13,14,15)$$

BCD to Excess-3 Code Converter

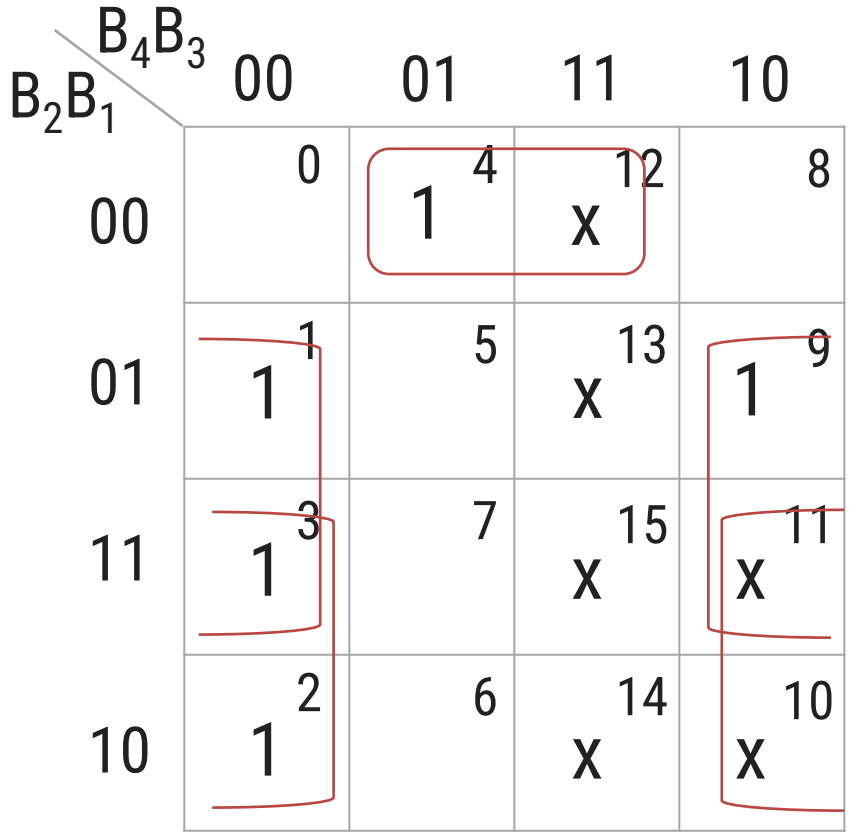
- K-Map for X_4, X_3, X_2, X_1 function and their minimization are as follows:

$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$



$X_4 = B_4 + B_3B_2 + B_3B_1$

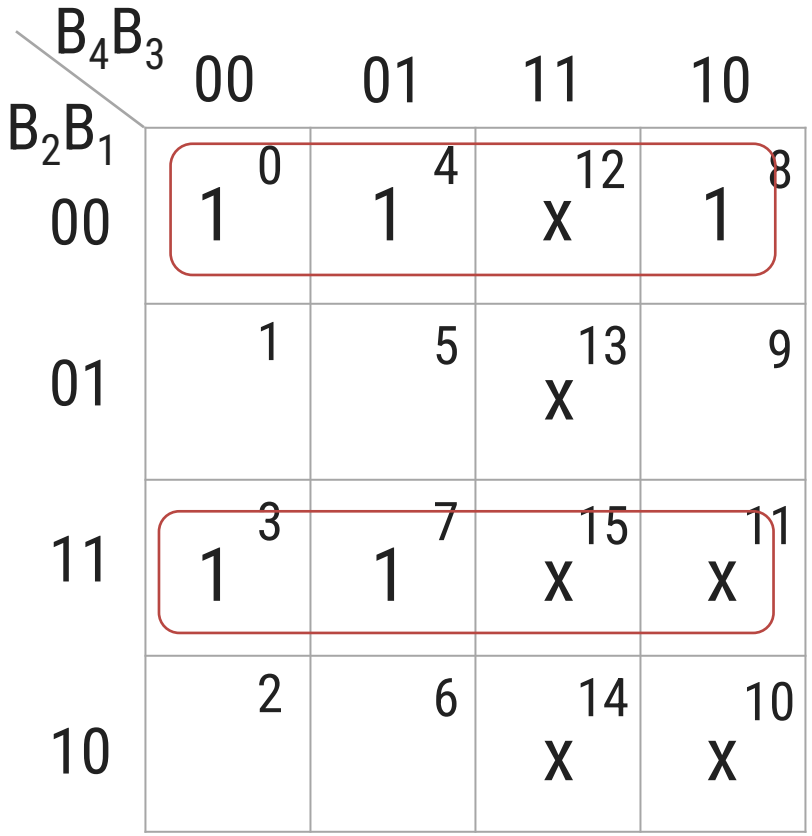
$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$



$X_3 = B_3B_2'B_1' + B_3'B_1 + B_3'B_2$

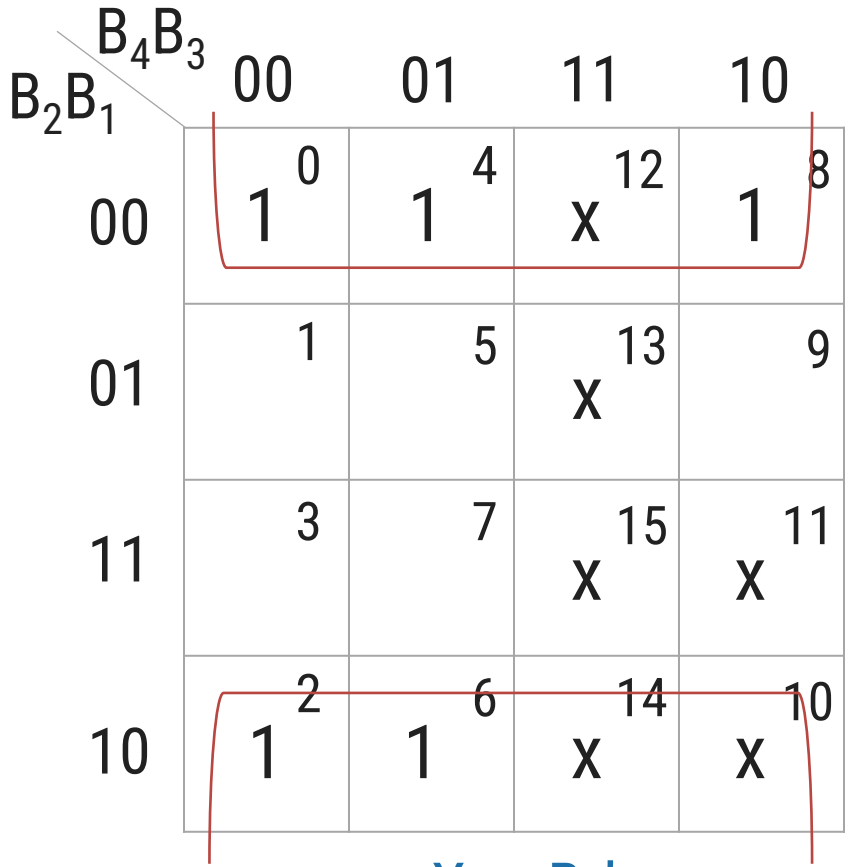
BCD to Excess-3 Code Converter

$$X_2 = \sum m(0,3,4,7,8) + d(10,11,12,13,14,15)$$



$X_2 = B_2'B_1' + B_2B_1$

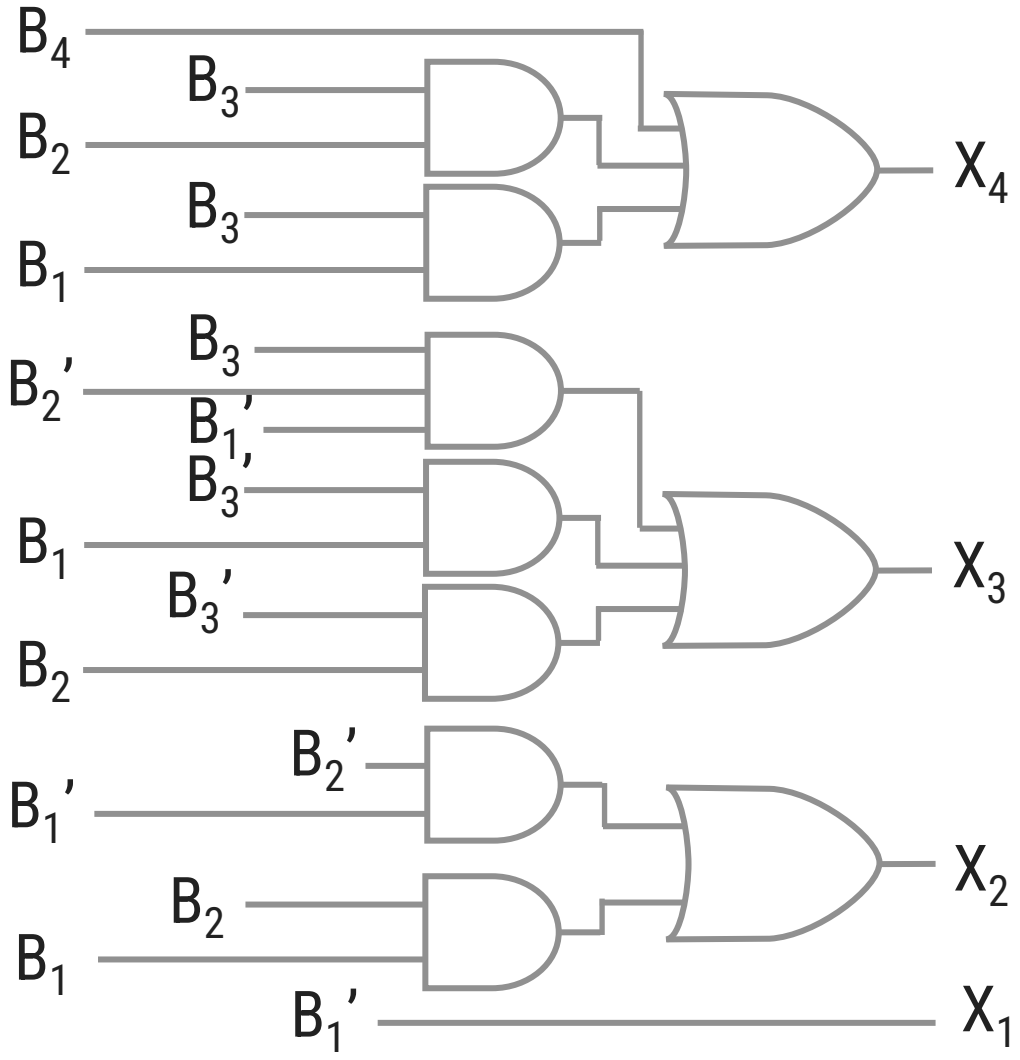
$$X_1 = \sum m(0,2,4,6,8) + d(10,11,12,13,14,15)$$



$X_1 = B_1'$

BCD to Excess-3 Code Converter

- Logic diagram for a BCD to Excess-3 is as follows

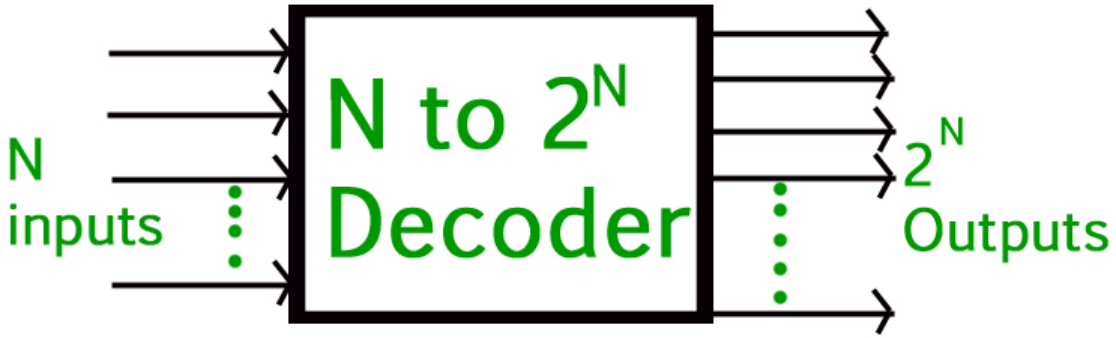


Decoder

- In Digital Electronics, discrete quantities of information are represented by binary codes.
- A binary code of **n bits** is capable of representing up to **2^n distinct elements** of coded information.
- The name **"Decoder"** means to translate or decode coded information from one format into another, so a digital decoder transforms a set of digital input signals into an equivalent decimal code at its output.
- A **decoder** is a **combinational circuit** that converts binary information from **n input lines** to a maximum of **2^n unique output lines**.

Decoder

- One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled.



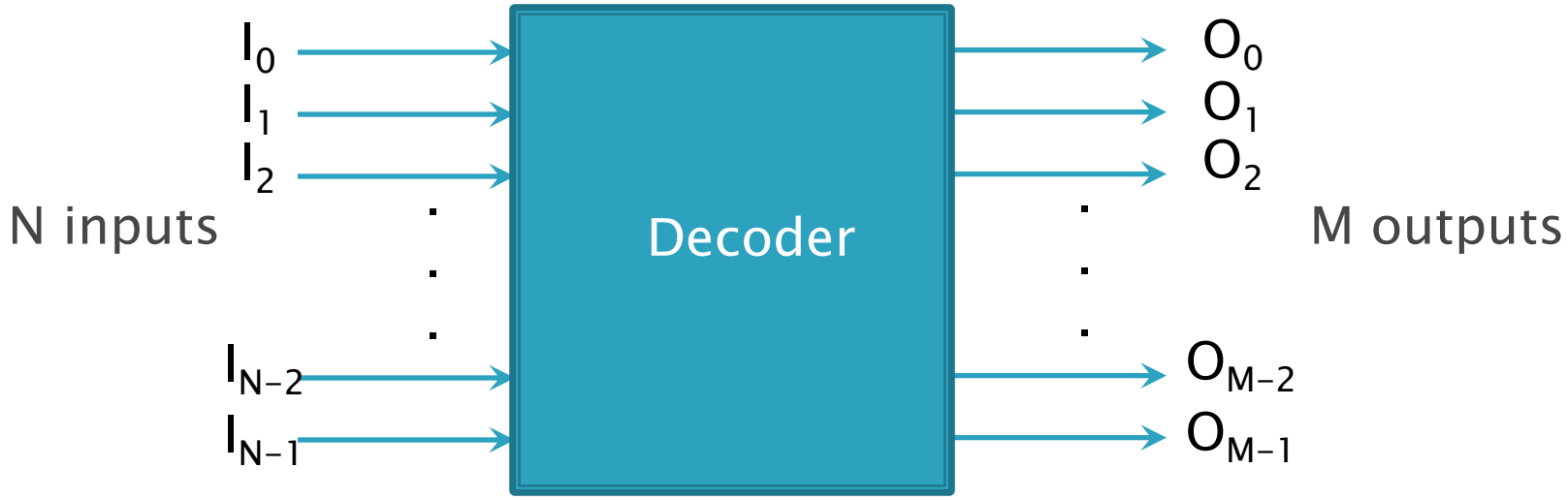
Decoder

- A decoder is a logic circuit that accepts a set of inputs which represents a binary number and activates the only output that corresponds to the input number.

- In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the specific output which corresponds to that number; all other outputs remain inactive.

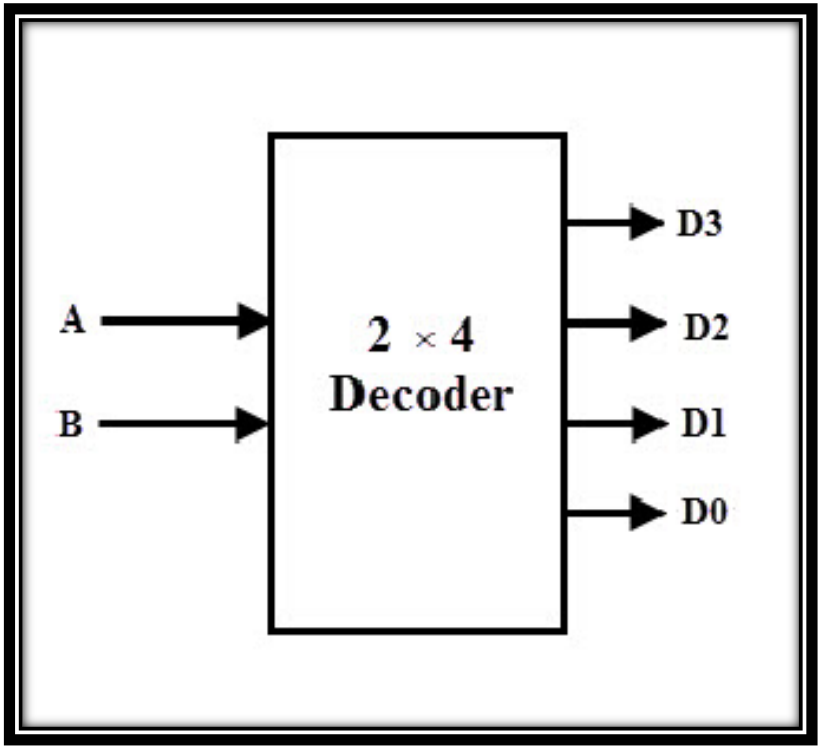
Decoder

- In its general form, a decoder has N input lines to handle N bits and M output lines such that only one output line is activated for each one of the possible combinations of inputs.



$$M = 2^N$$

2 to 4 Decoder



Truth Table

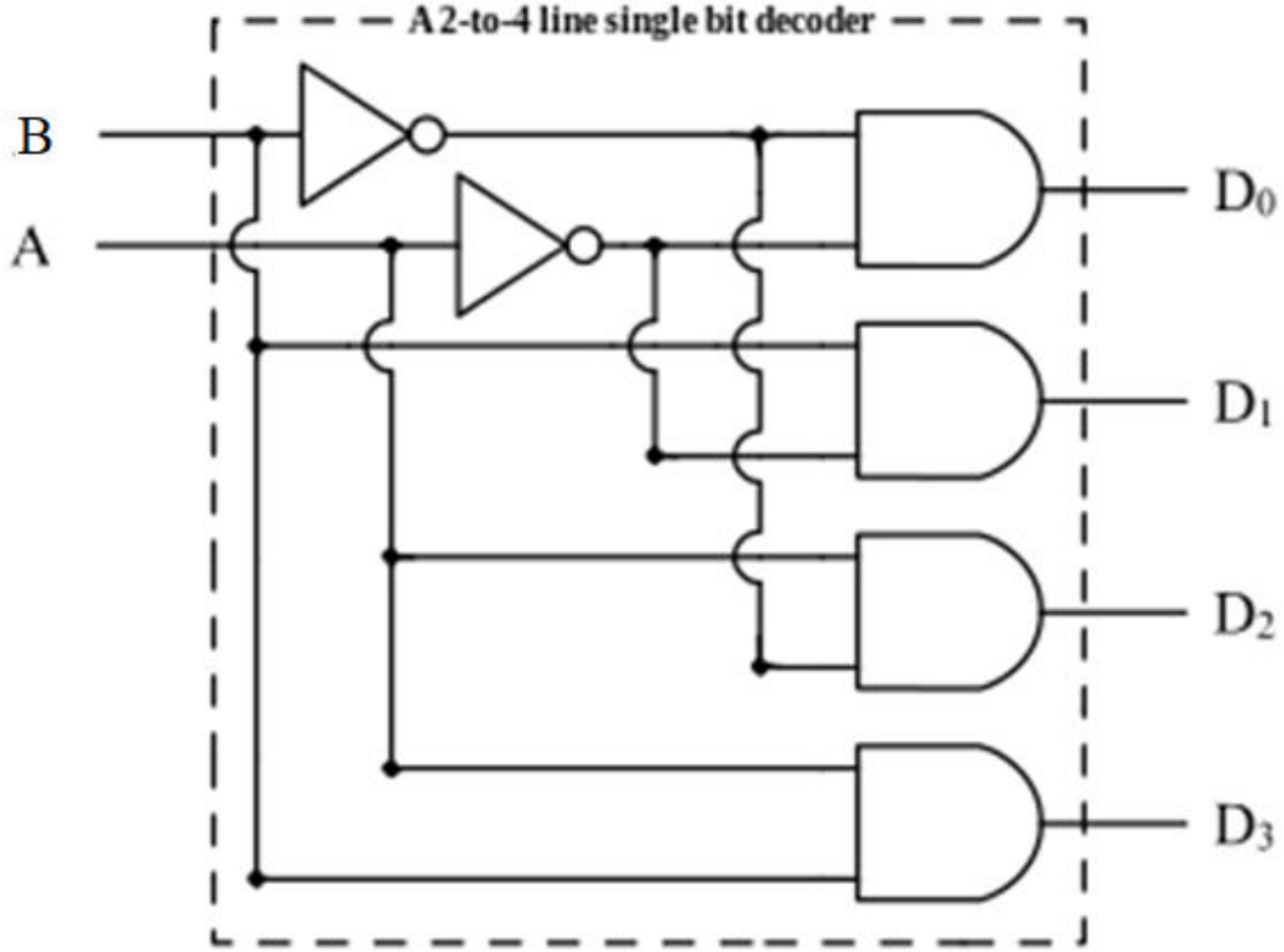
A	B	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$D0 = \overline{A}\,\overline{B}$

$D1 = \overline{A}\,B$

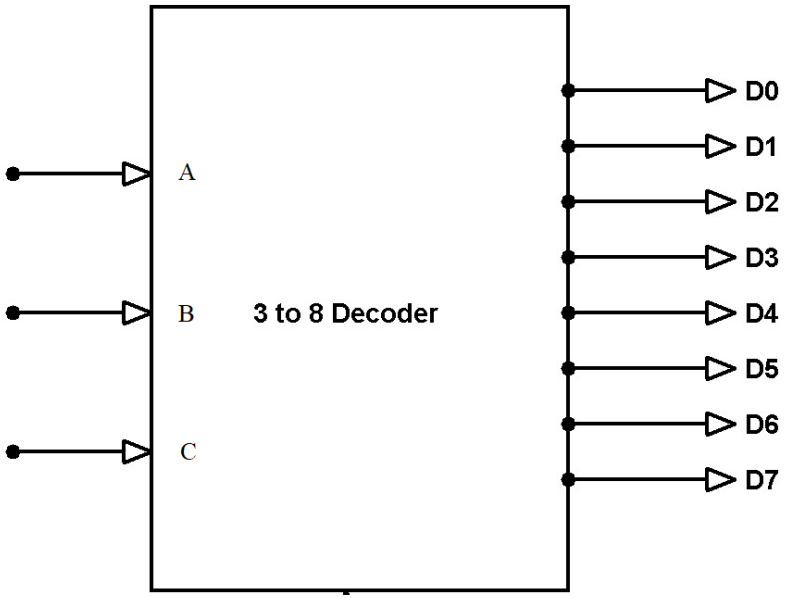
$D2 = A\,\overline{B}$

$D3 = A\,B$


A 2-to-4 line single bit decoder
B
A
D0
D1
D2
D3

3-Line to 8-Line Decoder

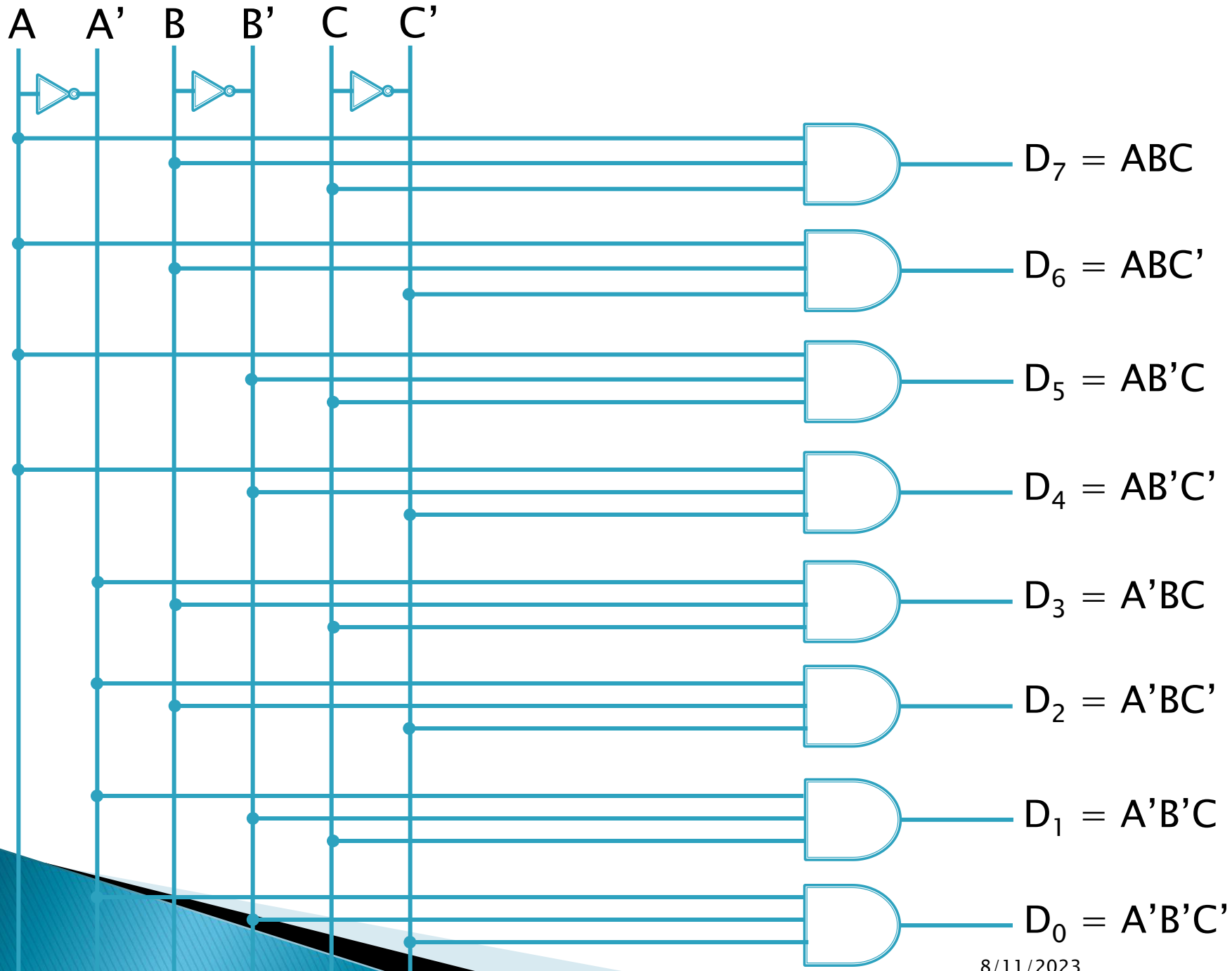
- 3 to 8 line decoder can be implemented using AND gates to achieve active-HIGH output.
- For active-LOW outputs, NAND gates are used.



3-Line to 8-Line Decoder

Truth Table

Inputs			Outputs							
A	B	C	D_7 ABC	D_6 ABC'	D_5 AB'C	D_4 AB'C'	D_3 A'BC	D_2 A'BC'	D_1 A'B'C	D_0 A'B'C'
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0


A
A'
B
B'
C
C'
$D_7 = ABC$
$D_6 = ABC'$
$D_5 = AB'C$
$D_4 = AB'C'$
$D_3 = A'BC$
$D_2 = A'BC'$
$D_1 = A'B'C$
$D_0 = A'B'C'$

Assignment

- Design a **4x16 Decoder** with block diagram, truth table and circuit diagram.
- Design a **BCD to Decimal Decoder** with block diagram, truth table and circuit diagram.
- Application of **Encoder** and **Decoder**

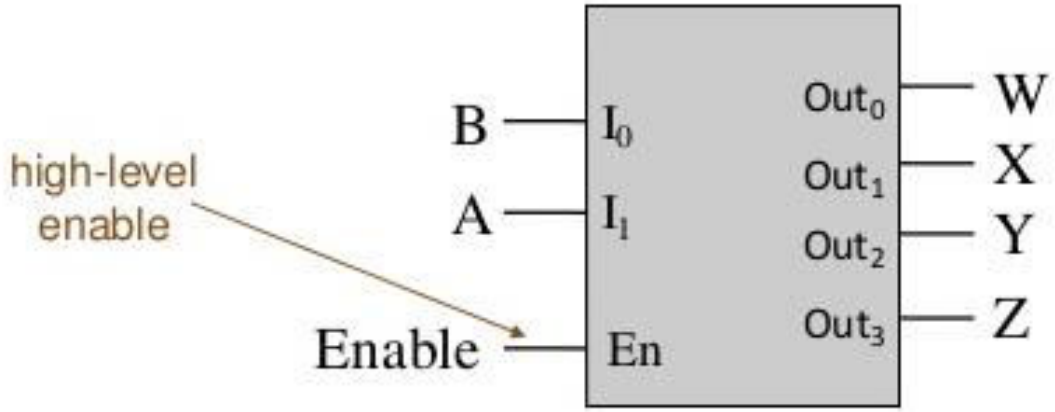
BCD to Decimal Decoder

D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	1	0 *
1	0	1	1	0	0	0	0	0	0	0	0	0	1 *
1	1	0	0	0	0	0	0	0	0	0	0	1	0 *
1	1	0	1	0	0	0	0	0	0	0	0	0	1 *
1	1	1	0	0	0	0	0	0	0	0	0	1	0 *
1	1	1	1	0	0	0	0	0	0	0	0	0	1 *

4x16 Decoder

W	X	Y	Z	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Decoder with Enable



	En	A	B	W	X	Y	Z
enabled	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
disabled	0	x	x	0	0	0	0

Design Full Adder using Decoder

1.Block Diagram:

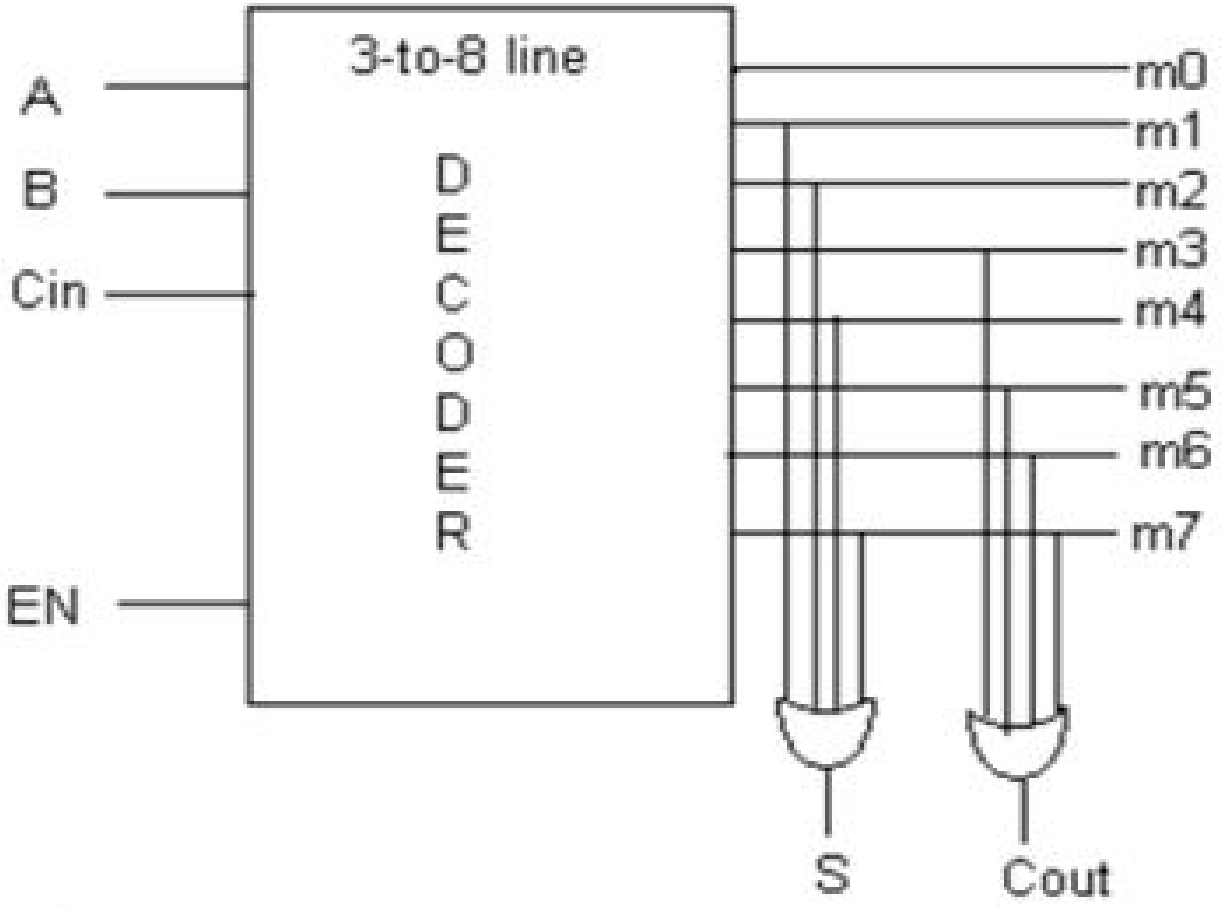
2.Truth Table:

Inputs			Outputs	
A	**B**	**C_{IN}**	**S**	**C_{OUT}**
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**

3.Expression for Sum (S): $S(A, B, C_{IN}) = \sum m(1, 2, 4, 7)$

4.Expression for Carry (C_{OUT}): $C_{OUT}(A, B, C_{IN}) = \sum m(3, 5, 6, 7)$

5.Logical circuit:



EXPANSION OF DECODERS:

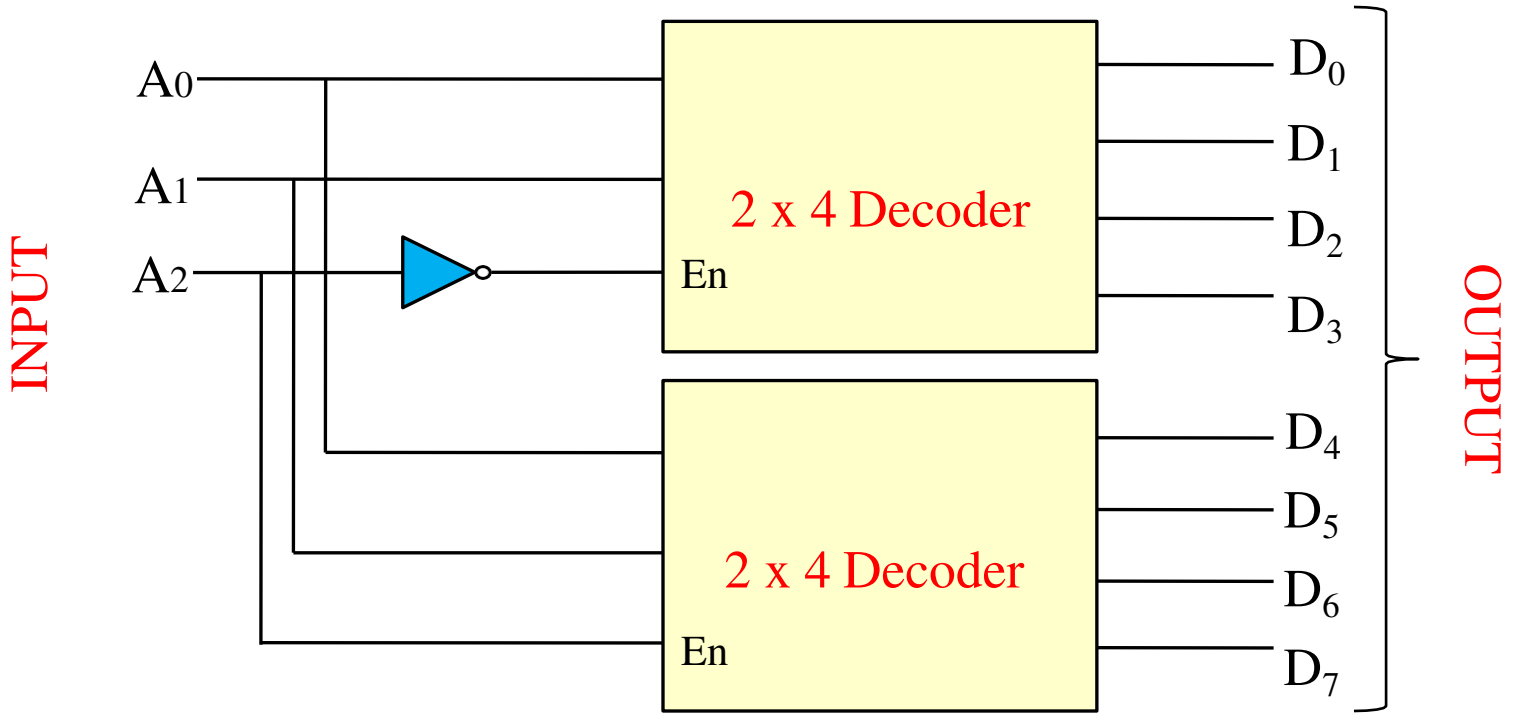
The number of lower order Decoder for implementing higher order Decoder can be find as

No. of lower order required = m_2/m_1

Where, m_1=No. of Outputs of lower order Decoder

m_2=No. of Outputs of higher order Decoder

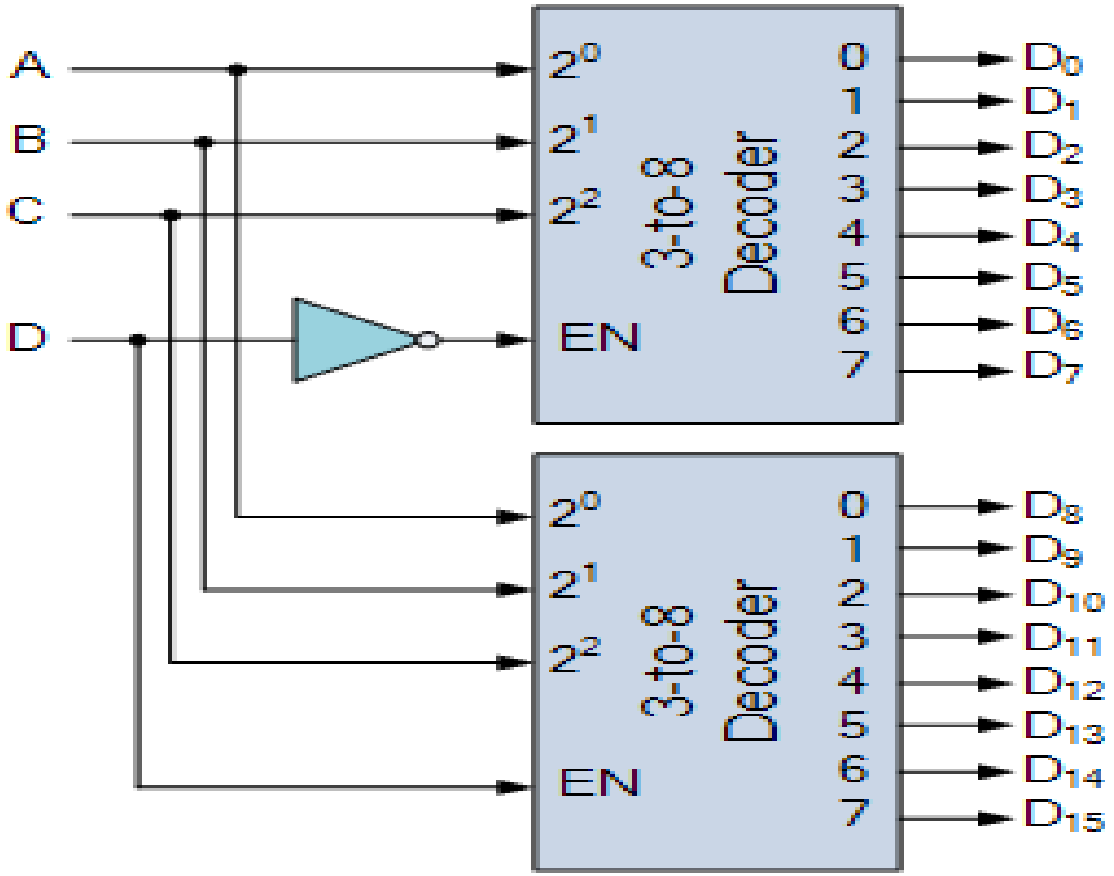
3 x 8 Decoder From 2 x 4 Decoder:



Inputs			Outputs							
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	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0

Inputs		Outputs			
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Construct 4x16 Decoder using 3x8 Decoder



4-to-16 Line Decoder Implemented with two 3-to-8 Decoders

- Inputs A, B, C are used to select which output on either decoder will be at logic “1” (HIGH) and input D is used with the enable input to select which encoder either the first or second will output the “1”.
- However, there is a limit to the number of inputs that can be used for one particular decoder, because as *n* increases, the number of AND gates required to produce an output also becomes larger resulting in the fan-out of the gates used to drive them becoming large.
- This type of active-“HIGH” decoder can be implemented using just Inverters, (NOT Gates) & AND gates. It is convenient to use an AND gate as the basic decoding element for the output because it produces a “HIGH” or logic “1” output only when all of its inputs are logic “1”.

- But some binary decoders are constructed using NAND gates instead of AND gates for their decoded output, since NAND gates are cheaper to produce than AND's as they require fewer transistors to implement within their design.
- The use of NAND gates as the decoding element, results in an active-"LOW" output while the rest will be "HIGH". As a NAND gate produces the AND operation with an inverted output, the NAND decoder looks like this with its inverted truth table.

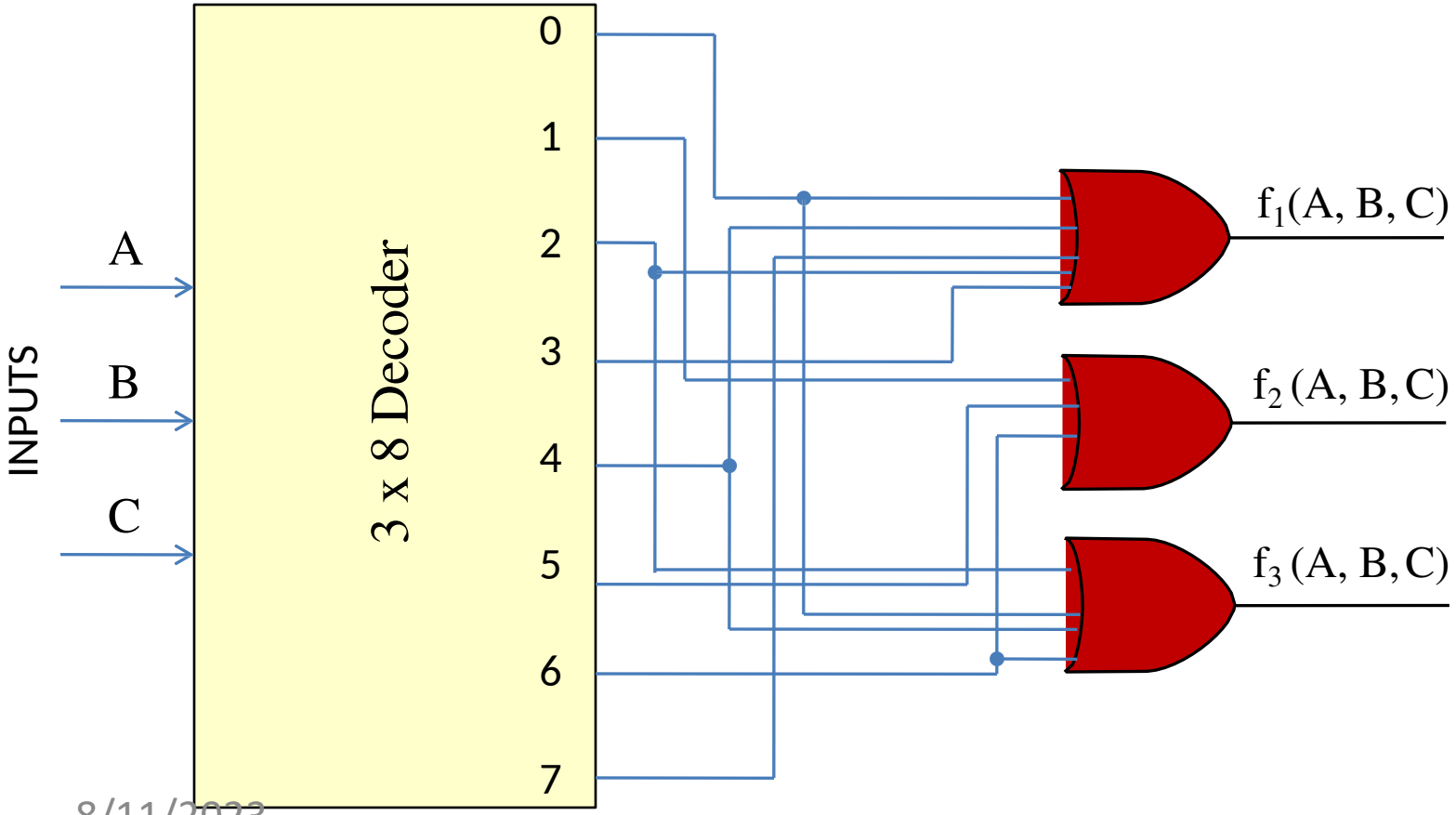
Example: Implement the following multiple output function using a suitable Decoder.

$f_1(A, B, C) = \sum m(0,4,7) + d(2,3)$

$f_2(A, B, C) = \sum m(1,5,6)$

$f_3(A, B, C) = \sum m(0,2,4,6)$

Solution: f1 consists of don't care conditions. So we consider them to be logic 1.



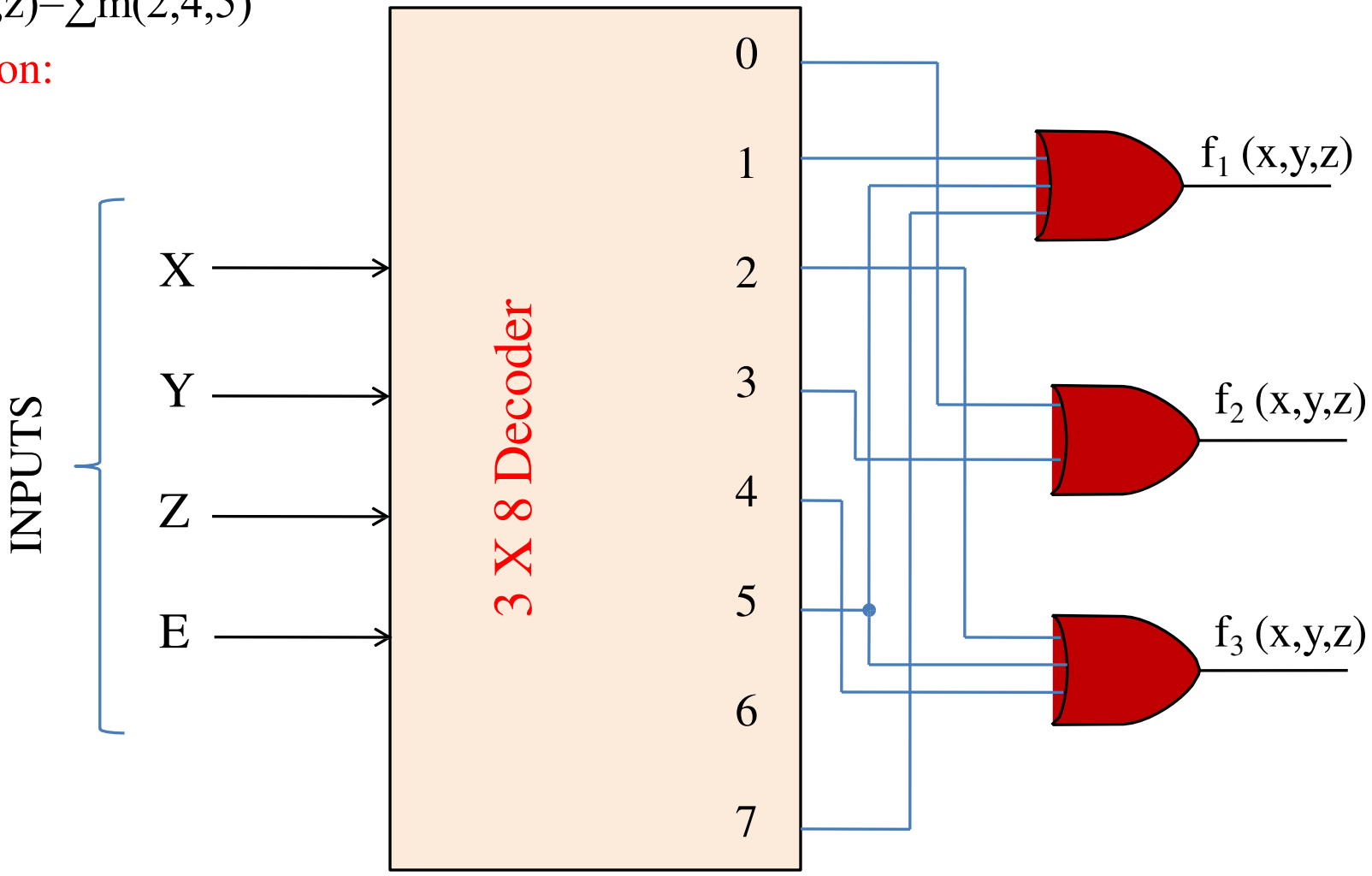
EXAMPLE: Implement the following Boolean function using suitable Decoder.

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

$f_2(x,y,z)=\sum m(0,3)$

$f_3(x,y,z)=\sum m(2,4,5)$

Solution:



EXAMPLE: A combinational circuit is defined by the following Boolean function. Design circuit with a Decoder and external gate.

$$F_1(x, y, z) = \bar{x}\,\bar{y}\,\bar{z} + x\,z$$

$$F_2(x, y, z) = x\,y\,\bar{z} + \bar{x}\,z$$

SOLUTION: STEP 1: Write the given function F_1 in SOPform

$$F_1(x, y, z) = \bar{x}\,\bar{y}\,\bar{z} + (y + \bar{y})\,x\,z$$

$$F_1(x, y, z) = \bar{x}\,\bar{y}\,\bar{z} + x\,y\,z + x\,\bar{y}\,z$$

$$F_1(x, y, z) = \Sigma m\,(0,5,7)$$

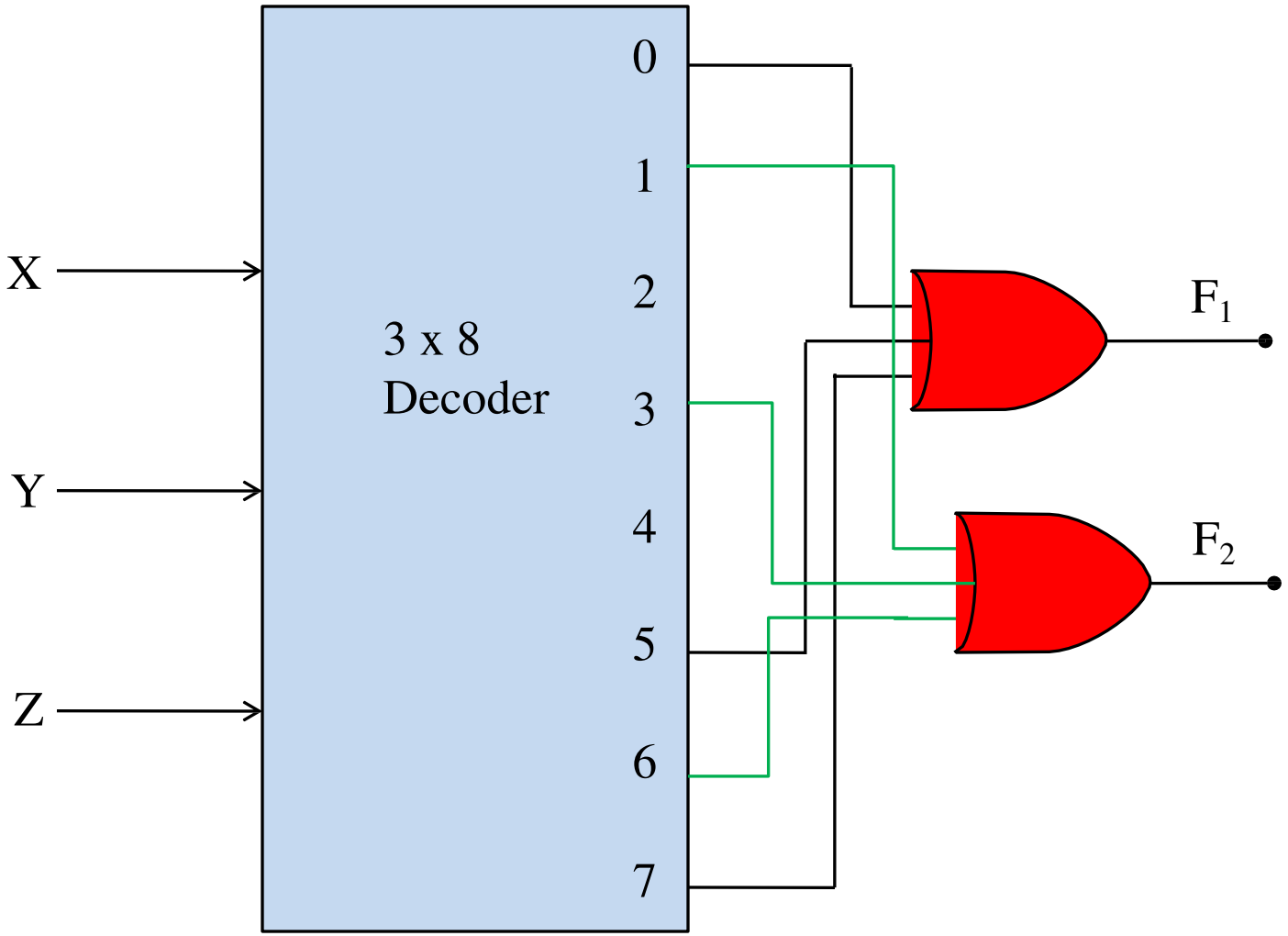
$$F_2(x, y, z) = x\,y\,\bar{z} + \bar{x}\,z$$

$$F_2(x, y, z) = x\,y\,\bar{z} + (y + \bar{y})\,\bar{x}\,z$$

$$F_2(x, y, z) = x\,y\,\bar{z} + \bar{x}\,y\,z + \bar{x}\,\bar{y}\,z$$

$$F_2(x, y, z) = \Sigma m\,(1,3,6)$$

Boolean Function using Decoder:



Encoder

PARAMETER	ENCODER	DECODER
Input applied	Original message signal	Coded binary input
Output generated	Coded binary output	Original message
Input lines	2^n	n
Output lines	n	2^n
Operation	Simple	Complex
Basic logic element	OR gate	AND gate along with NOT gate
Applications	E-mail , video encoders etc.	Microprocessors, memory chips etc.

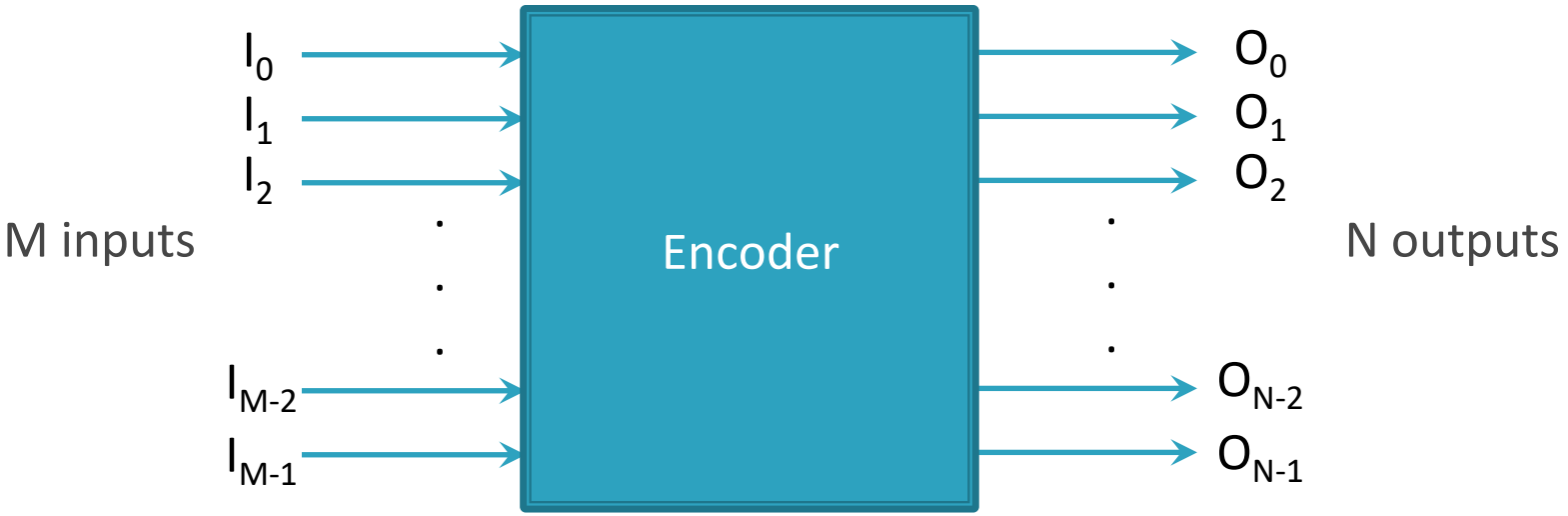
Encoder

- An Encoder is a **combinational circuit** which 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.
- It will produce a binary code equivalent to the input, which is active High.
- Therefore, the encoder encodes 2^n input lines with ‘n’ bits.
- Encoder performs the reverse operation of Decoder.
- **Limitations** : Only one input can be high at a time. If two inputs are enabled at the same time then output is not correct.

- An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting from familiar symbols or numbers to a coded format is called encoding.

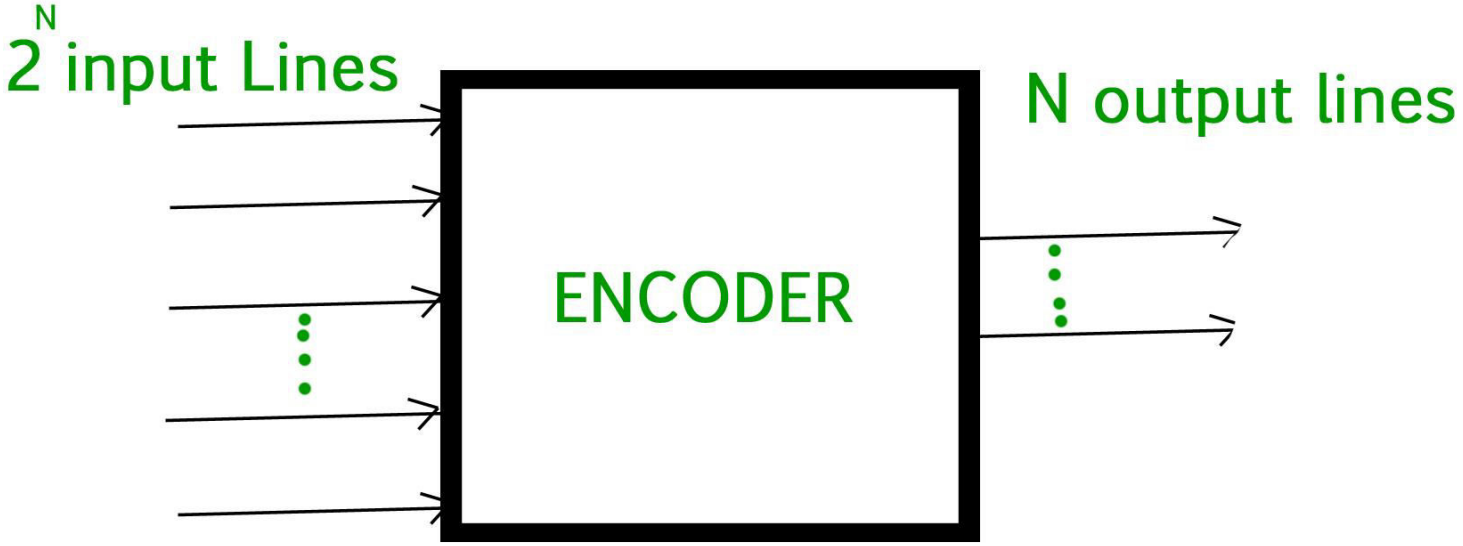
Encoder

- Device to convert familiar numbers or symbols into coded format.
- It has a number of input lines, only one of which is activated at a given time, and produces an N-bit output code depending on which input is activated.
- Figure shows the block diagram of an encoder with M inputs and N outputs.

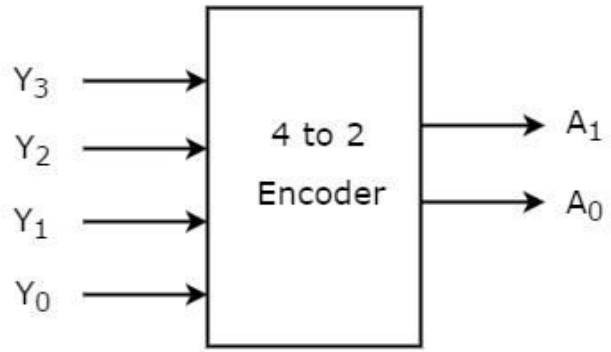


$M = 2^N$

Block Diagram

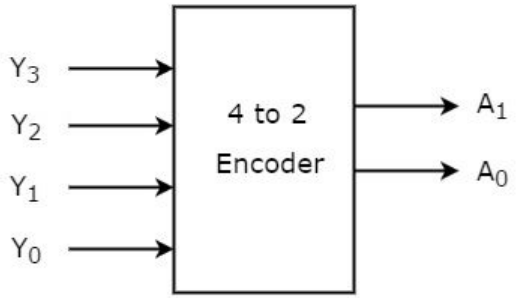


Block Diagram of 4-to-2 Encoder



4 to 2 Encoder

- Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0.
- The **block diagram** of 4 to 2 Encoder is shown in the following figure.



- At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.

- The **Truth table** of 4 to 2 encoder is shown below.

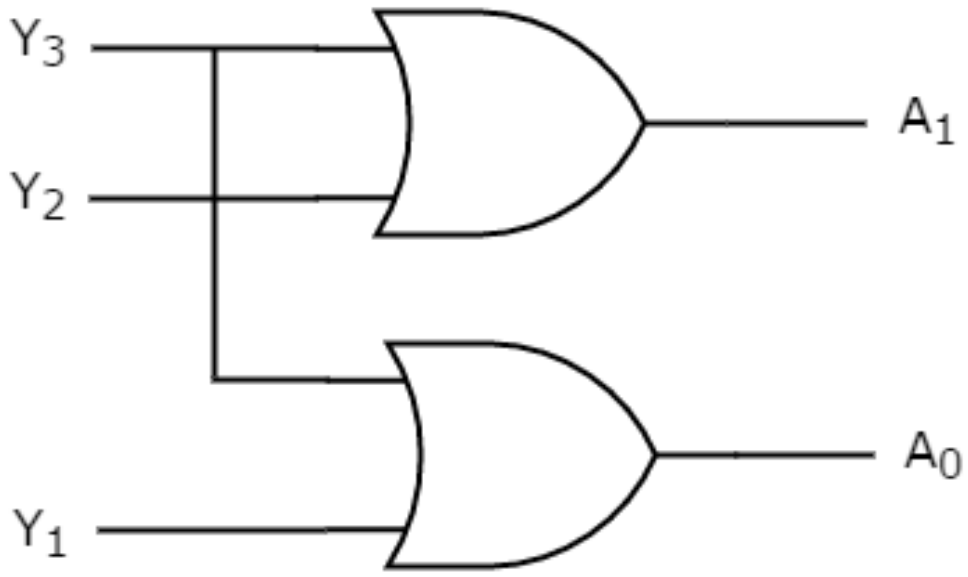
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

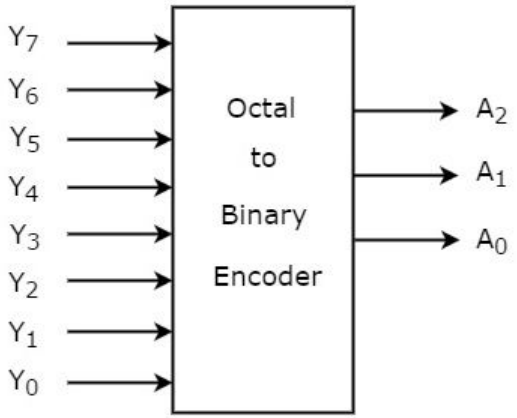
$$A_0 = Y_3 + Y_1$$

- We can implement the above two Boolean functions by using two input OR gates.
- The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



8 : 3 Encoder (Octal to Binary)

- Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0.
- Octal to binary encoder is nothing but 8 to 3 encoder.
- The **block diagram** of octal to binary Encoder is shown in the following figure.



- At any time, only one of these eight inputs can be ‘1’ in order to get the respective binary code.
- The **Truth table** of octal to binary encoder is shown below.

Inputs								Outputs		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- From Truth table, we can write the Boolean functions for each output as

$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

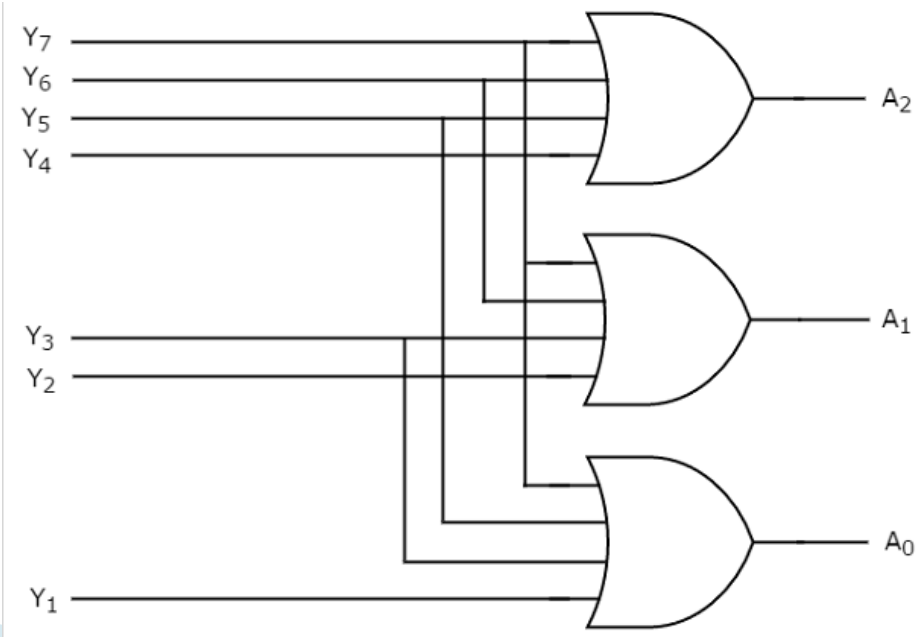
$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

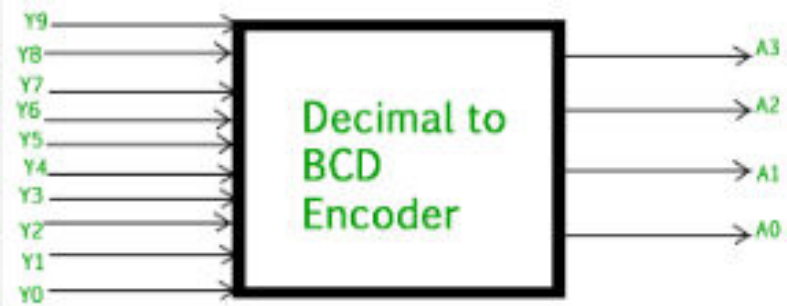
$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

- We can implement the above Boolean functions by using four input OR gates.
- The **circuit diagram** of octal to binary encoder is shown in the following figure.



Decimal to BCD Encoder

- The decimal to binary encoder usually consists of **10 input lines** and **4 output lines**.
- Each input line corresponds to the each decimal digit and 4 outputs correspond to the BCD code.
- This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines.
- The figure below shows the logic symbol of decimal to BCD encoder :



- The truth table for decimal to BCD encoder is as follows:

INPUTS										OUTPUTS			
Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

- From Truth table, we can write the Boolean functions for each output as

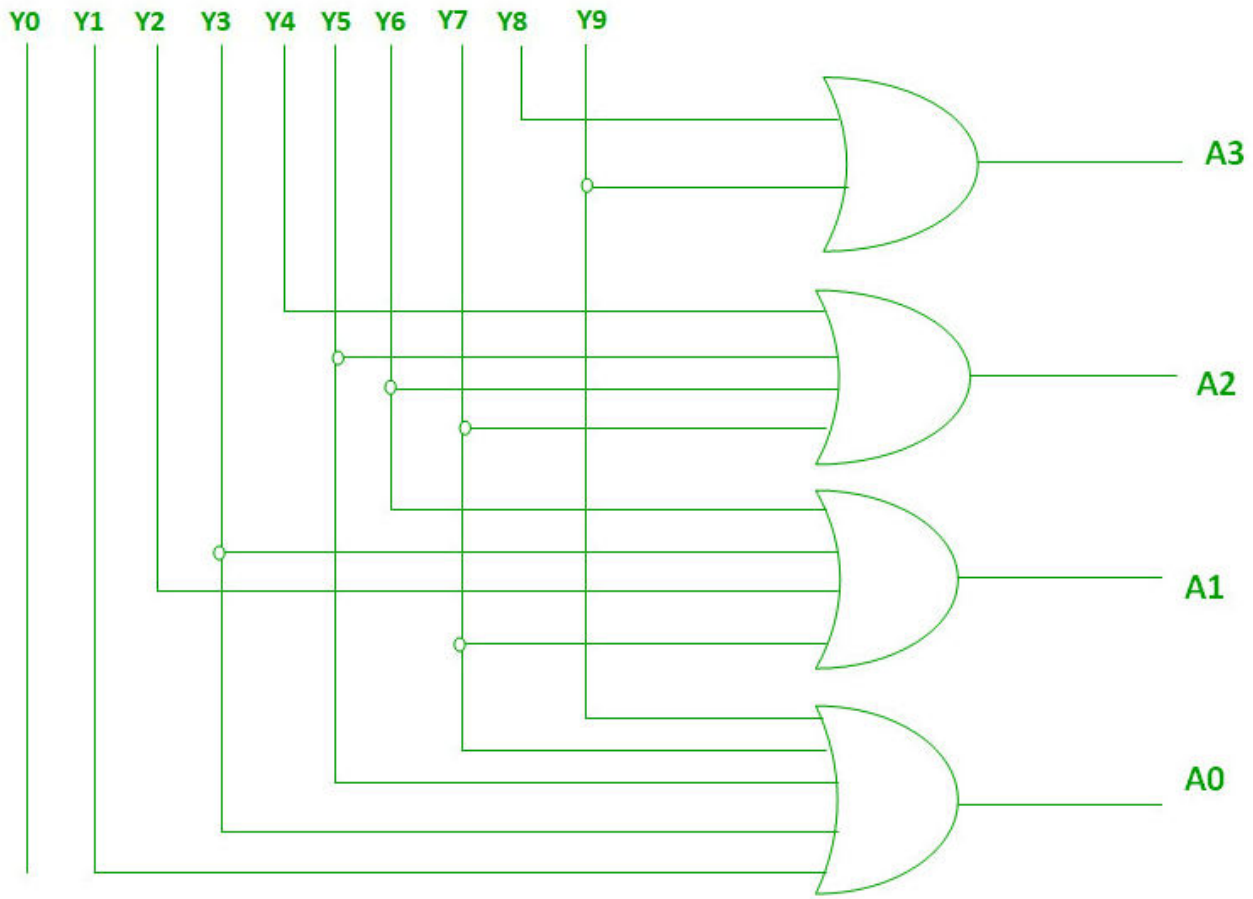
$$A3 = Y9 + Y8$$

$$A2 = Y7 + Y6 + Y5 + Y4$$

$$A1 = Y7 + Y6 + Y3 + Y2$$

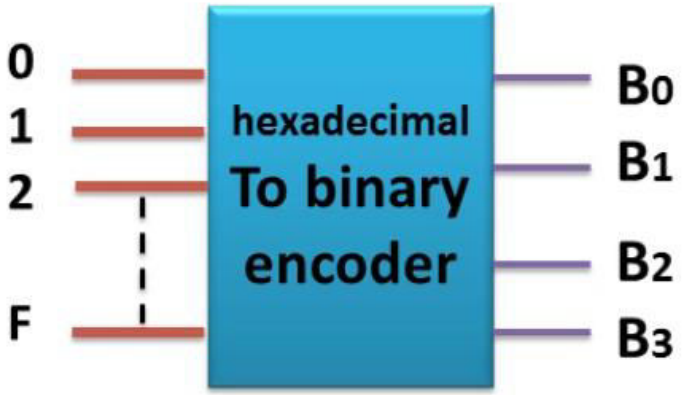
$$A0 = Y9 + Y7 + Y5 + Y3 + Y1$$

- The above Boolean functions can be implemented using OR gates



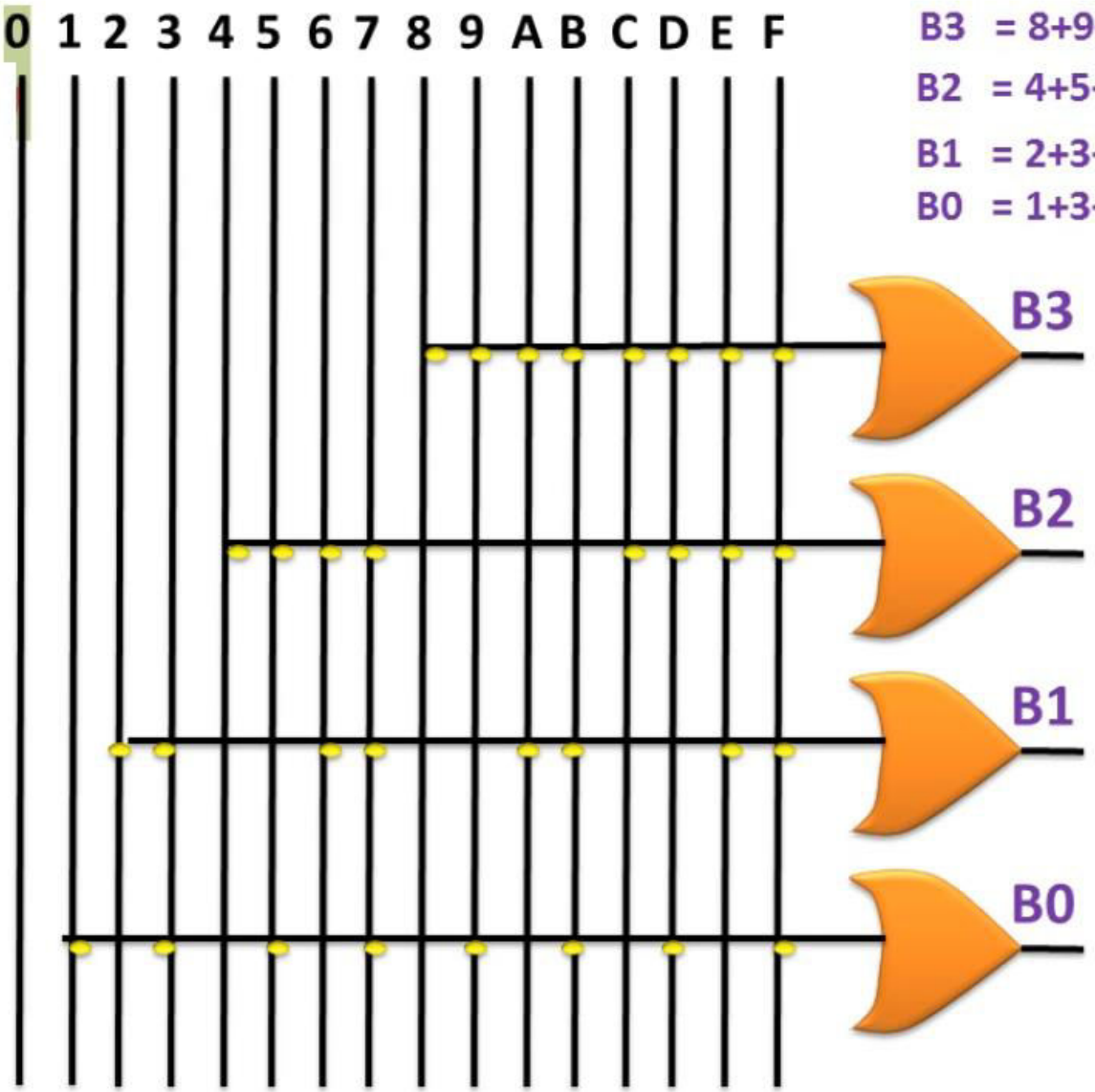
Assignment

- Draw the block diagram, truth table and logic diagram of hexadecimal to binary encoder



INPUT	OUTPUT			
	B3	B2	B1	B0
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
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

B3 = 8+9+A+B+C+D+E+F
B2 = 4+5+6+7+C+D+E+F
B1 = 2+3+6+7+A+B+E+F
B0 = 1+3+5+7+9+B+D+F



B3 = 8+9+A+B+C+D+E+F

B2 = 4+5+6+7+C+D+E+F

B1 = 2+3+6+7+A+B+E+F

B0 = 1+3+5+7+9+B+D+F

Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a priority encoder.
- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position.

- A 4 to 2 priority encoder has **4 inputs** : Y3, Y2, Y1 & Y0 and **2 outputs** : A1 & A0.
- Here, the input, Y3 has the **highest priority**, whereas the input, Y0 has the **lowest priority**.
- In this case, even if more than one input is ‘1’ at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**.

We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

Priority Encoder (Summary)

- A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH.
- The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded.
- For example, if both decimal 3 and decimal 4 are activated simultaneously, then a priority encoder would encode decimal 4.

4x2 Priority Encoder

Truth Table

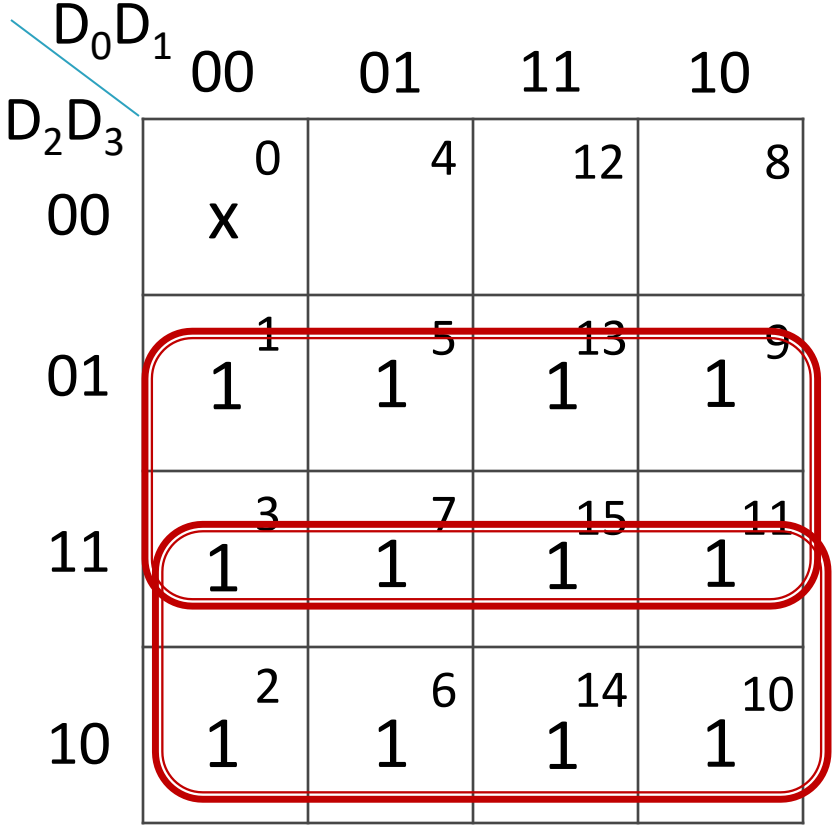
Inputs				Outputs		
D_3	D_2	D_1	D_0	A	B	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

$$A = \sum m(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15) \qquad B = \sum m(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

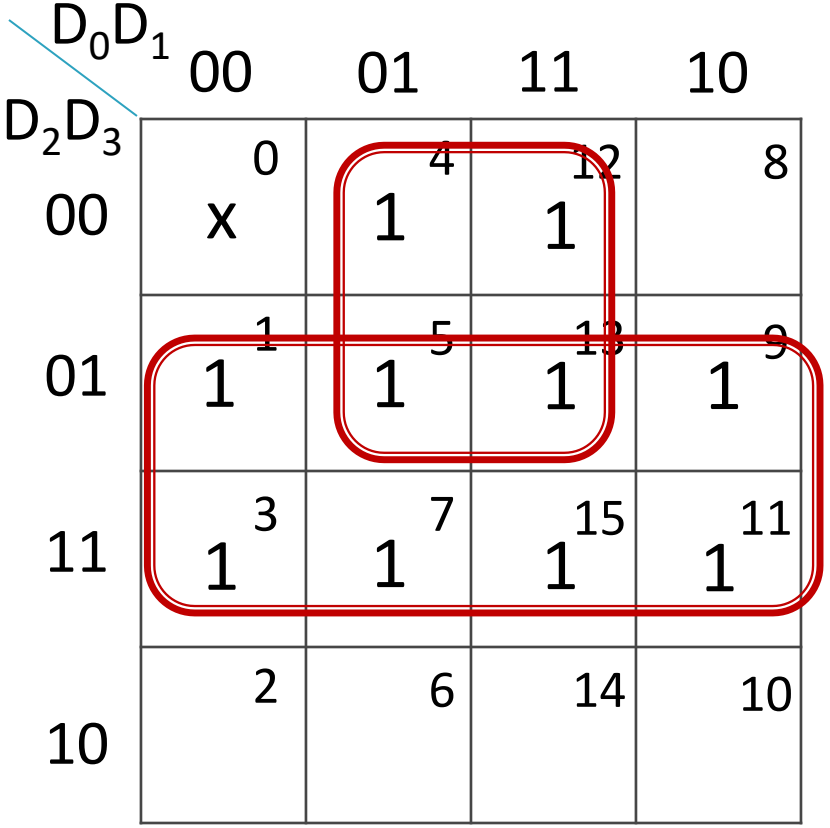
$$V = \sum m(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

Priority Encoder

$$A = \sum m(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15) \qquad B = \sum m(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$



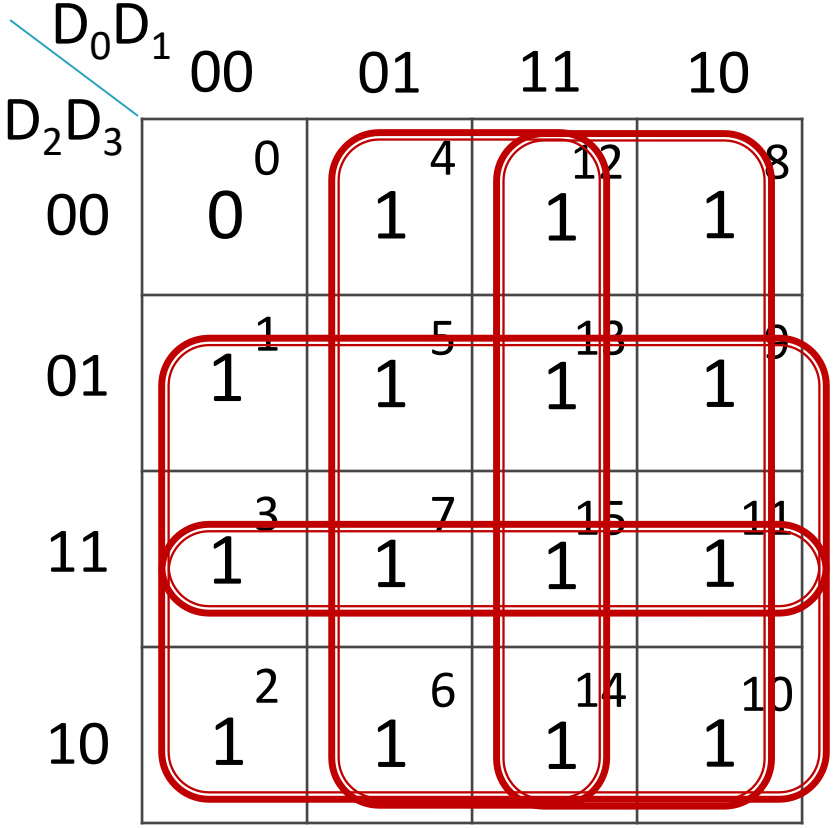
$A = D_3 + D_2$



$B = D_3 + D_2' D_1$

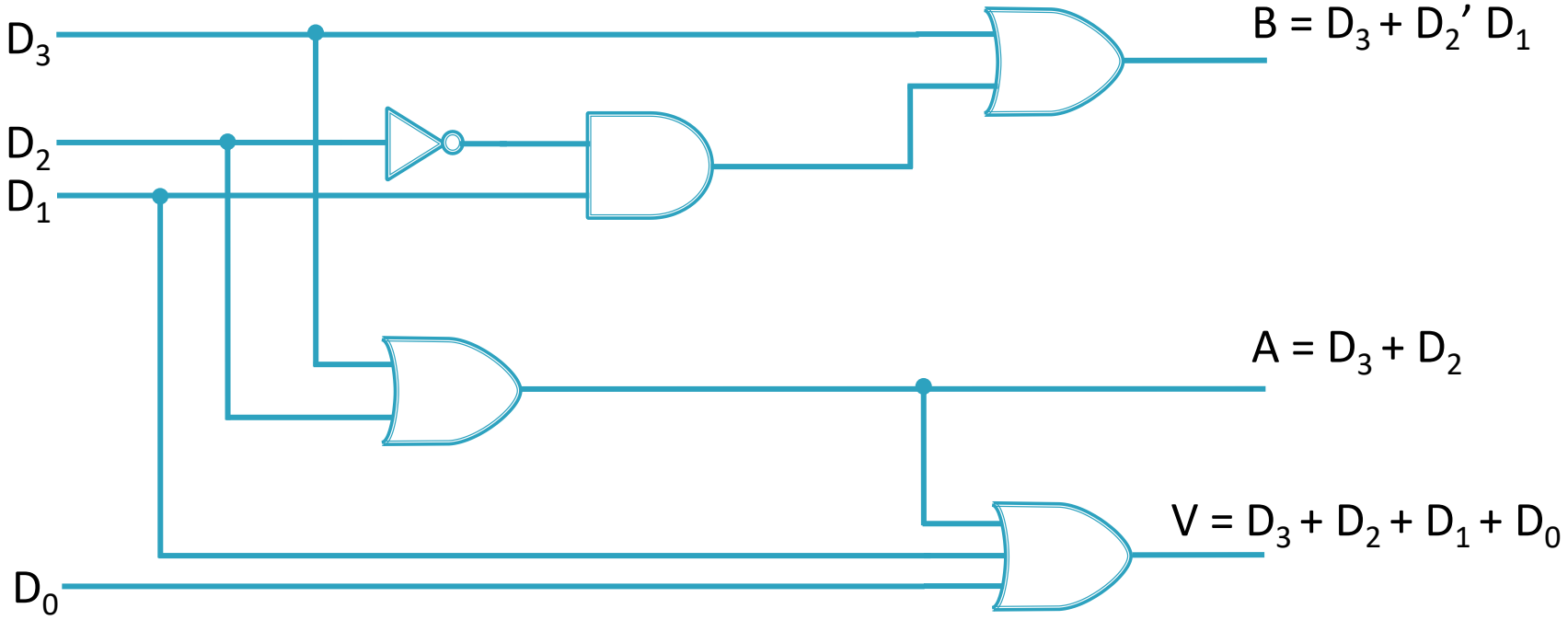
Priority Encoder

$$V = \sum m(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$



$$V = D_3 + D_2 + D_1 + D_0$$

Priority Encoder



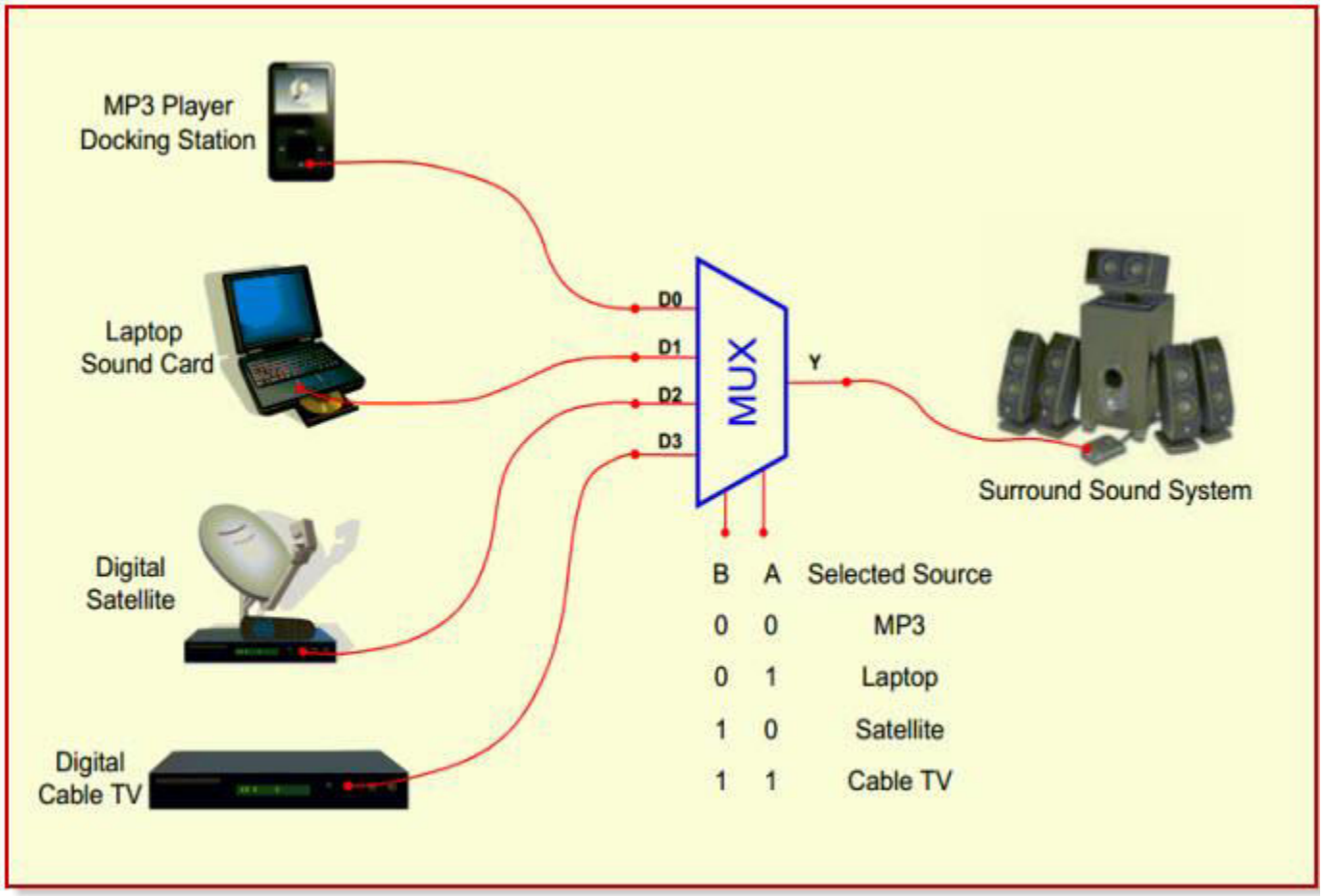
Drawbacks of Normal Encoders

- There is an ambiguity, when all outputs of encoder are equal to zero.
- Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For example, if both Y3 and Y6 are '1', then the encoder produces 111 at the output. This is neither equivalent code corresponding to Y3, when it is '1' nor the equivalent code corresponding to Y6, when it is '1'.

Application of Priority Encoder

- This type of encoders is used to reduce the number of connections required in a particular application in which there are multiple inputs are present.
- Used to control interrupt requests by acting on the highest priority request.
- To encode the output of a flash analog to digital converter

Multiplexer



Multiplexer

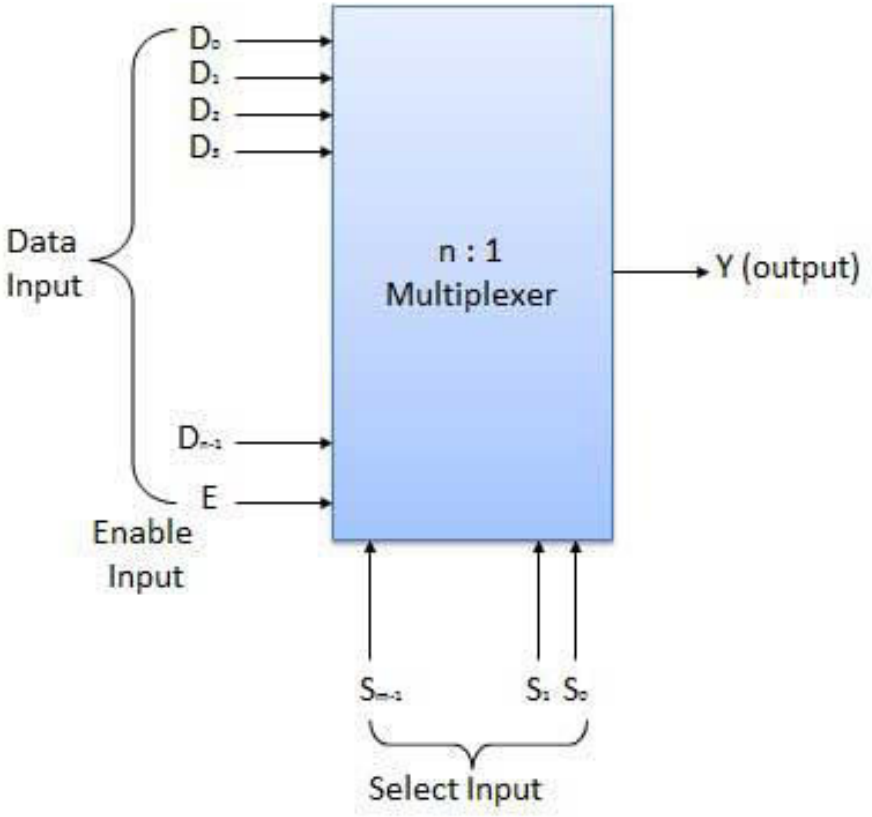
- A multiplexer(MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Consider an integer ‘m’, which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.
- A **m-to-1** Multiplexer has
  - m Inputs: $I_0, I_1, I_2, \ldots\ldots\ldots\ldots I_{(m-1)}$
  - One Output: Y
  - n Control inputs: $S_0, S_1, S_2, \ldots\ldots S_{(n-1)}$
  - One (or more) Enable input(s)

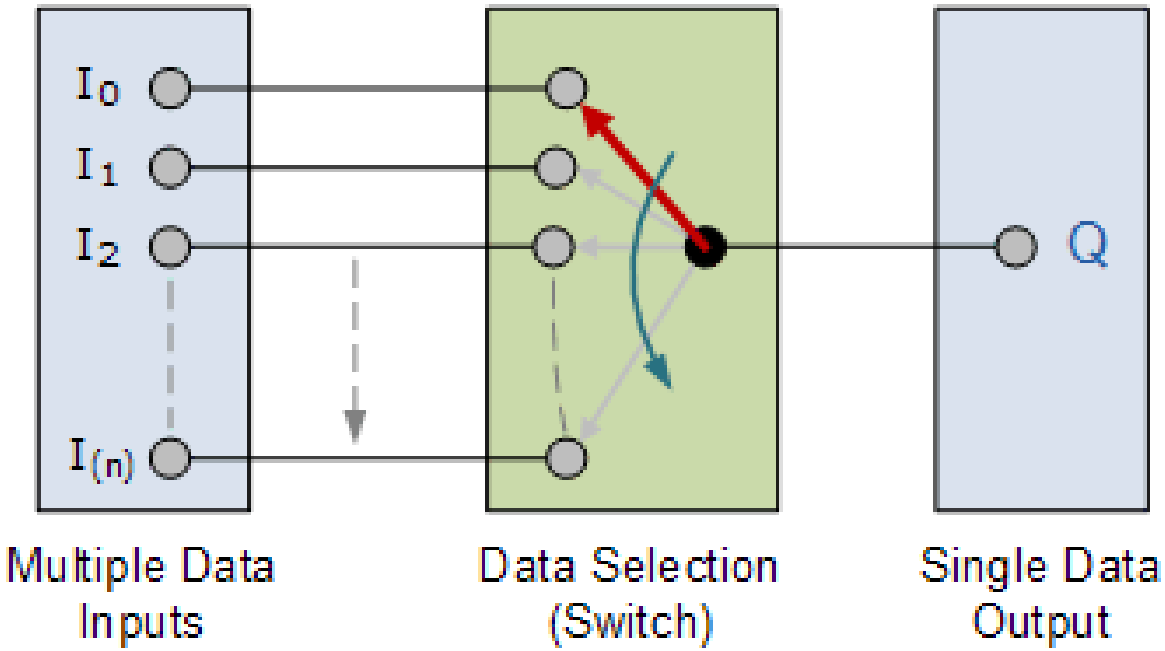
  such that Y may be equal to one of the inputs, depending upon the control inputs.

- **Multiplexer** is a combinational circuit that has maximum of 2^n data inputs, ‘n’ selection lines and single output line.
- One of these data inputs will be connected to the output based on the values of selection lines.
- Since there are ‘n’ selection lines, there will be 2^n possible combinations of zeros and ones.
- So, each combination will select only one data input.
- Multiplexer is also called as **Mux**.

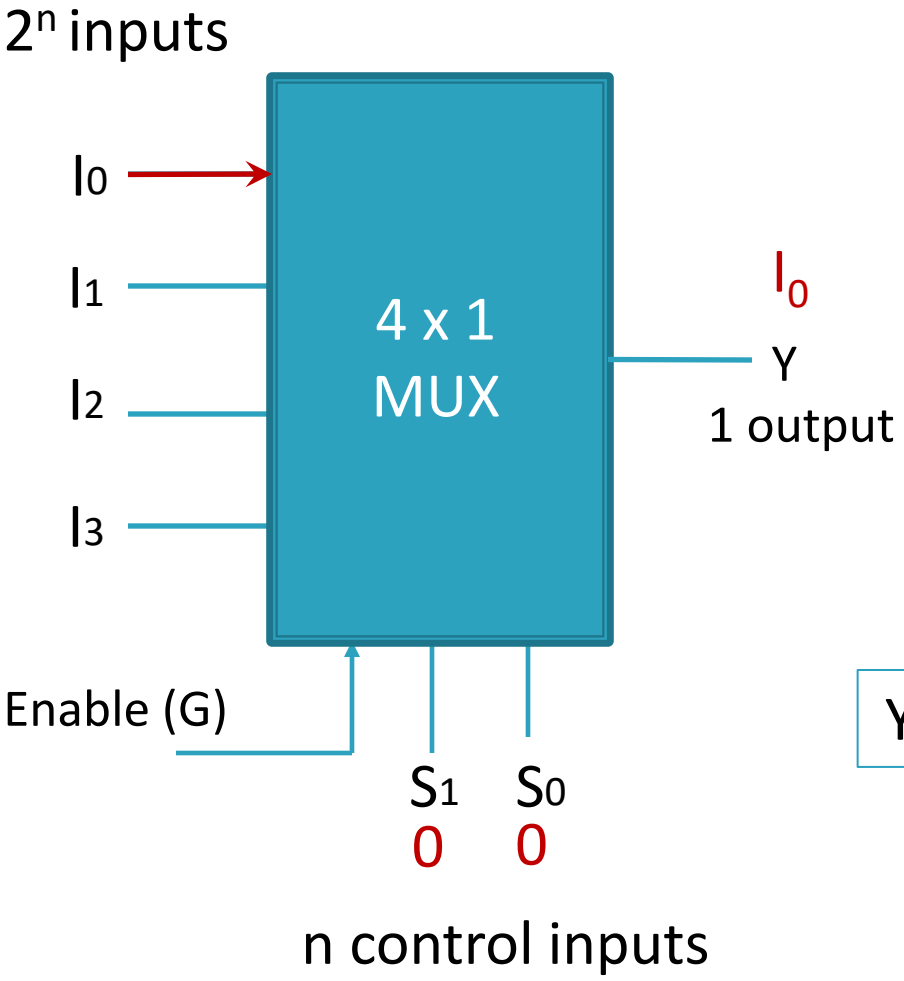
Block Diagram of Multiplexer



How Select Line works



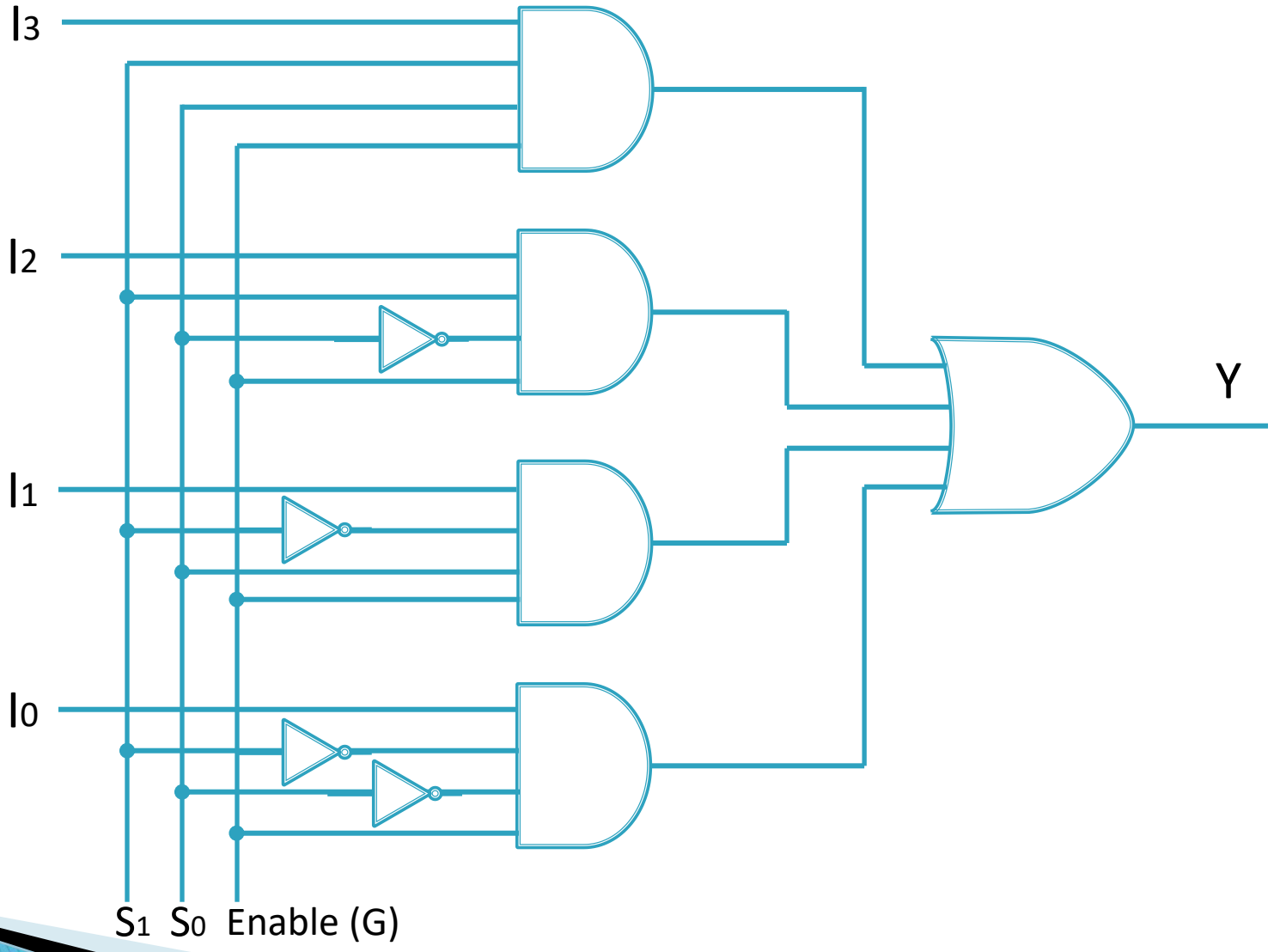
4-to-1 Multiplexer



Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

4 x 1 MUX Actual Circuit



Self Check Exercise (HW)

- Implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Application of Multiplexer

- Logic function generation
- Data selection
- Data routing
- Operation sequencing
- Parallel-to-serial conversion
- Waveform generation

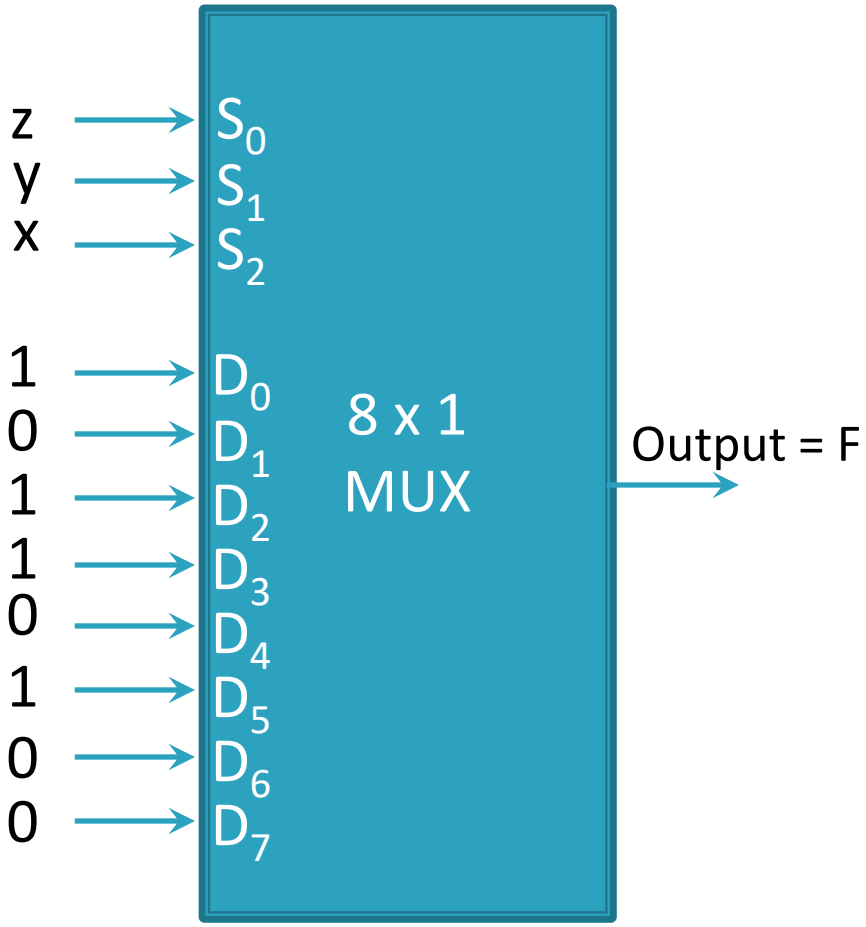
Implementing Boolean Functions with Multiplexers

Logic function generator

- Implement the following function using 8 to 1 MUX

$$F(x,y,z) = \Sigma\, m(0,2,3,5)$$

S_2	S_1	S_0	F
x	y	z	
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Logic function generator

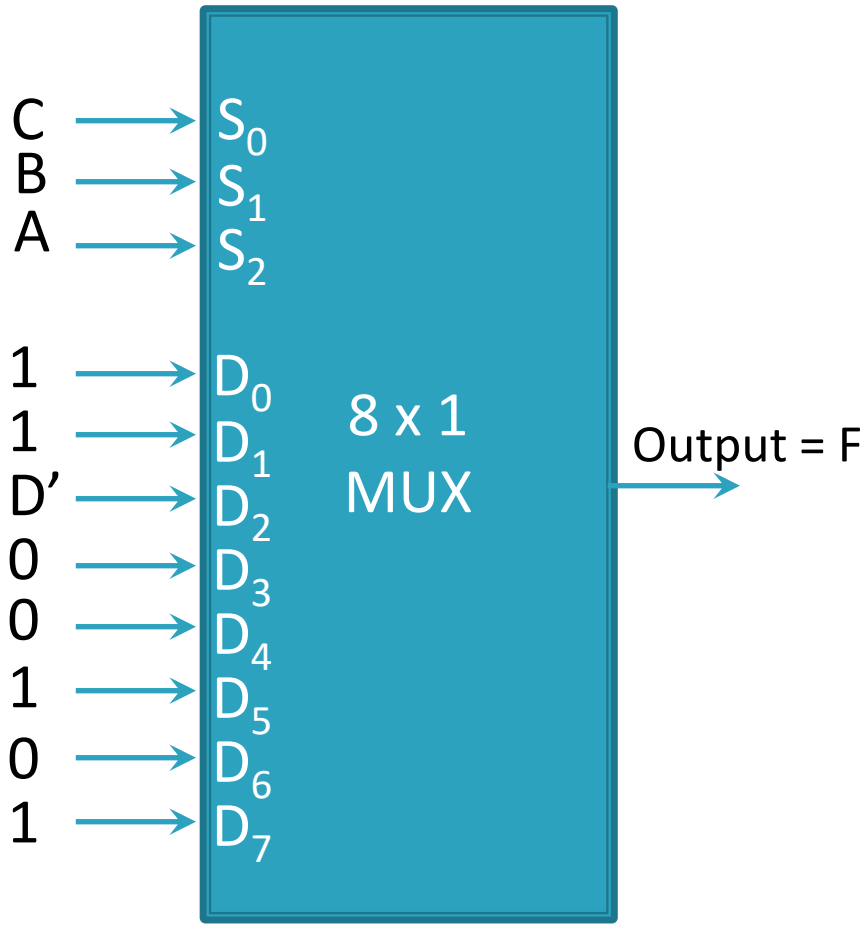
- Multiplexer with n-data select inputs can implement any function of n + 1 variables.
- The first n variables of the function as the select inputs and to use the least significant input variable and its complement to drive some of the data inputs.
- If the single variable is denoted by D, each data output of the multiplexer will be D, D’, 1, or 0.
- Suppose, we wish to implement a 4-variable logic function using a multiplexer with three data select inputs.
- Let the input variables be A, B, C, and D; D is the LSB.

Logic function generator

- A truth table for the function F(A, B, C, D) is constructed with ABC has the same value twice once with D = 0 and again with D = 1.
- The following rules are used to determine the connections that should be made to the data inputs of the multiplexer.
  1. If F = 0 both times when the same combination of ABC occurs, connect logic 0 to the data input selected by that combination.
  2. If F = 1 both times when the same combination of ABC occurs, connect logic 1 to the data input selected by that combination.
  3. If F is different for the two occurrences of a combination of ABC, and if F = D in each case, connect D to the data input selected by that combination.
  4. If F is different for the two occurrences of a combination of ABC, and if F = D in each case, connect D to the data input selected by that combination.

Implement the following function using 8x1 MUX $F = \Sigma\, m(0,1,2,3,4,10,11,14,15)$

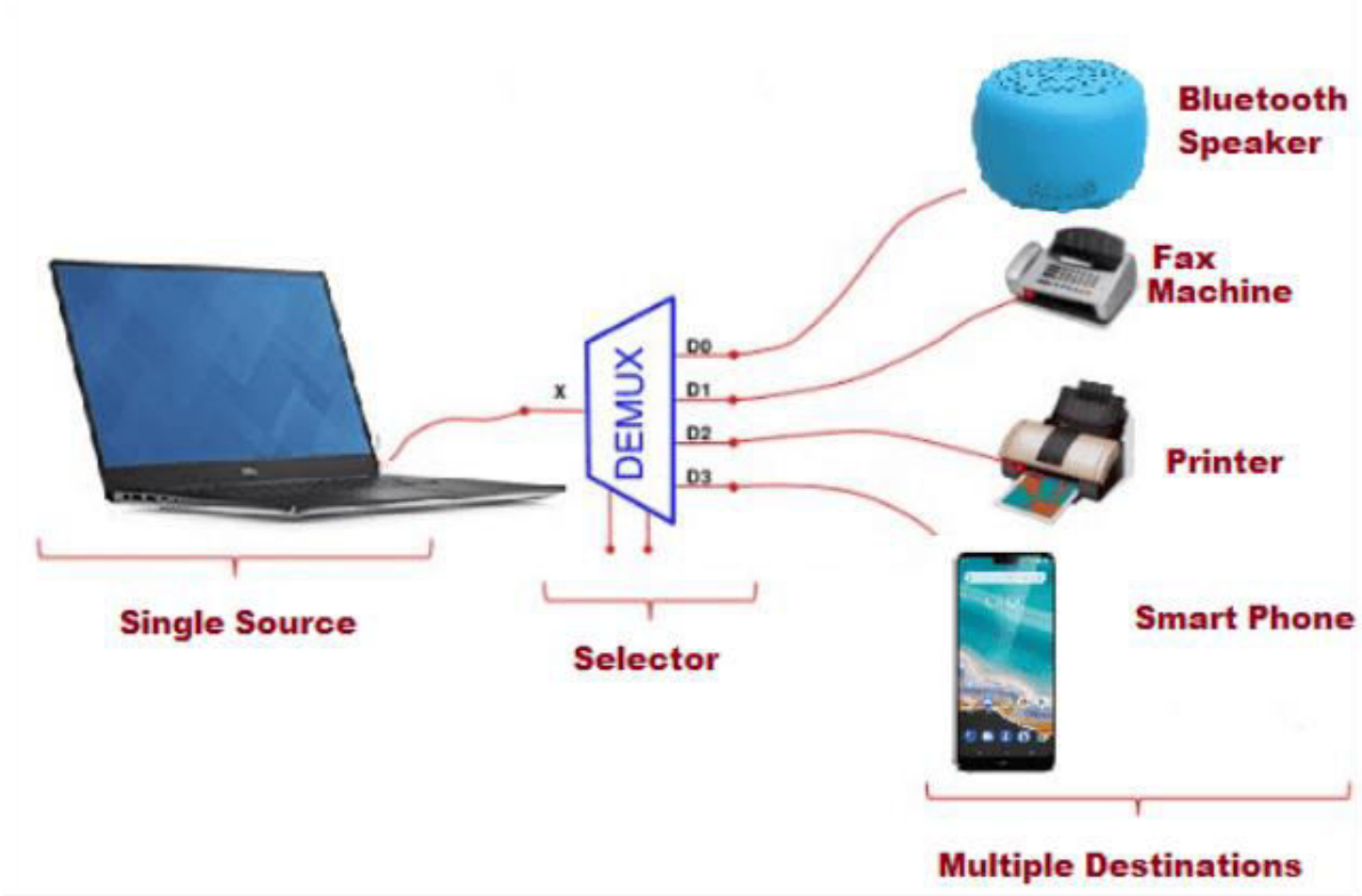
S_2	S_1	S_0	D	F
A	B	C		
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



Self Check Exercise

1. Implement F = Σ m(1,2,4,7) using 8x1 MUX
2. Implement F = Σ m(1,2,3,6,7) using 8x1 MUX
3. Implement F = Σ m(6,8,10,11,12) using 8x1 MUX
4. Implement 4x1 MUX using 2x1 MUX
5. Implement 8x1 MUX using 4x1 MUX
6. Implement 8x1 MUX using 4x1 MUX and 2x1 MUX

Demultiplexer



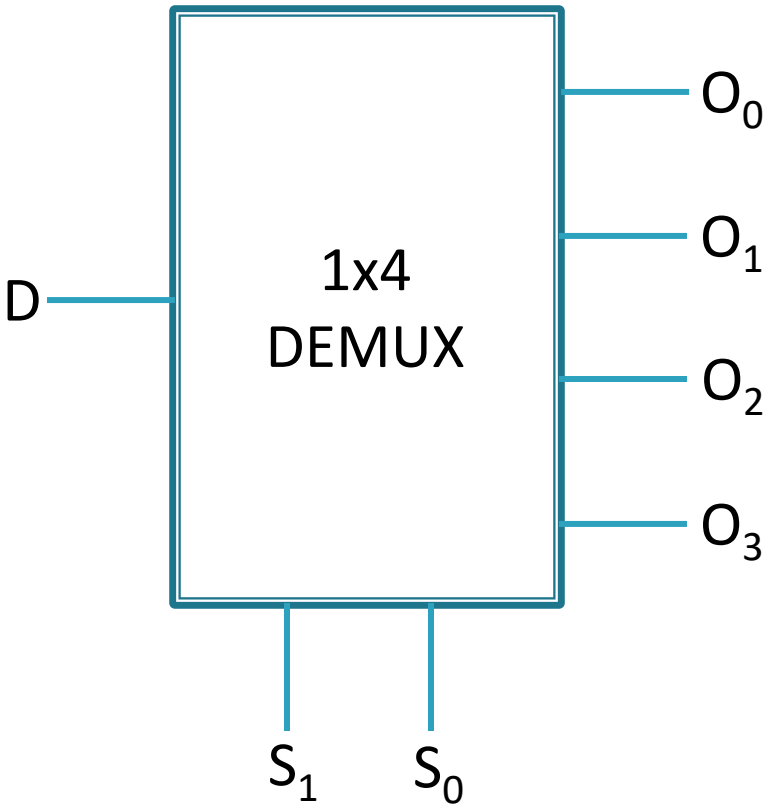
Demultiplexer

- A demultiplexer(DEMUX) is a device that allows digital information from one source to be routed onto a multiple lines for transmission over different destinations.
- Consider an integer 'm', which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.
- A **1-to-m** Demultiplexer has
  - One Input: D
  - m Outputs: O_0, O_1, O_2, ... $O_{(m-1)}$
  - n Control inputs: S_0, S_1, S_2, ... $S_{(n-1)}$
  - One (or more) Enable input(s)

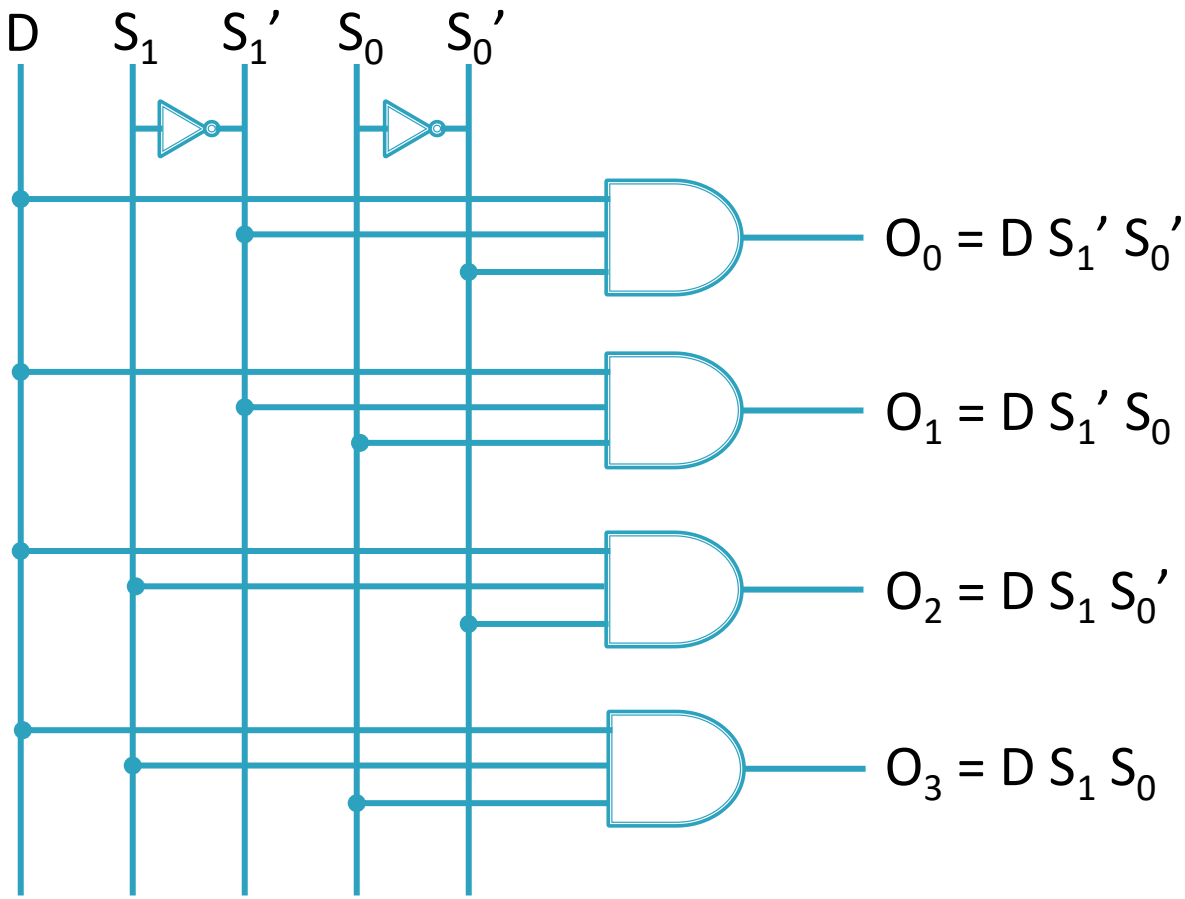
  Such that D may be transfer to one of the outputs, depending upon the control inputs.

1-to-4 Demultiplexer



Input	Select code		Outputs			
	S_1	S_0	O_3	O_2	O_1	O_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

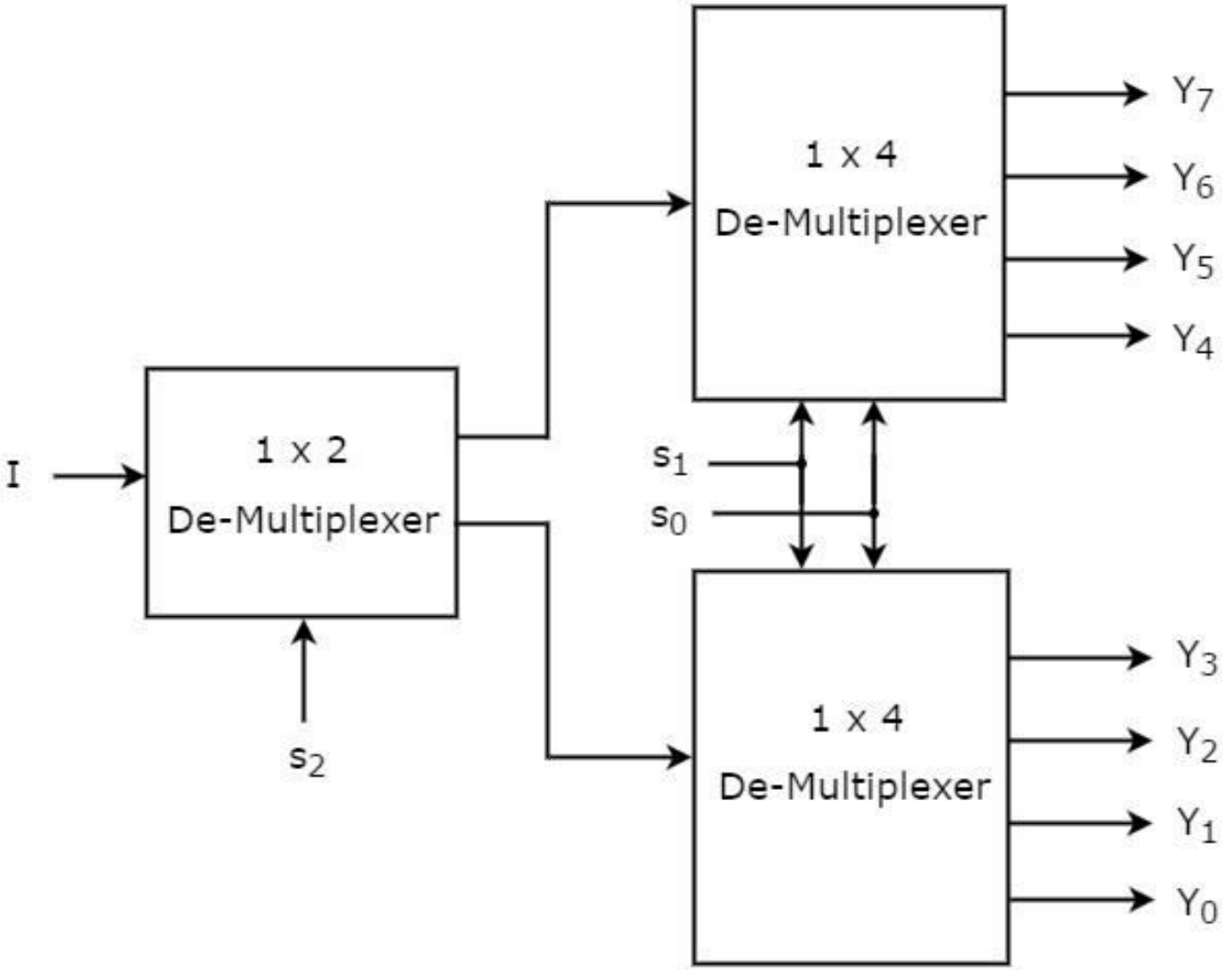
1-to-4 Demultiplexer



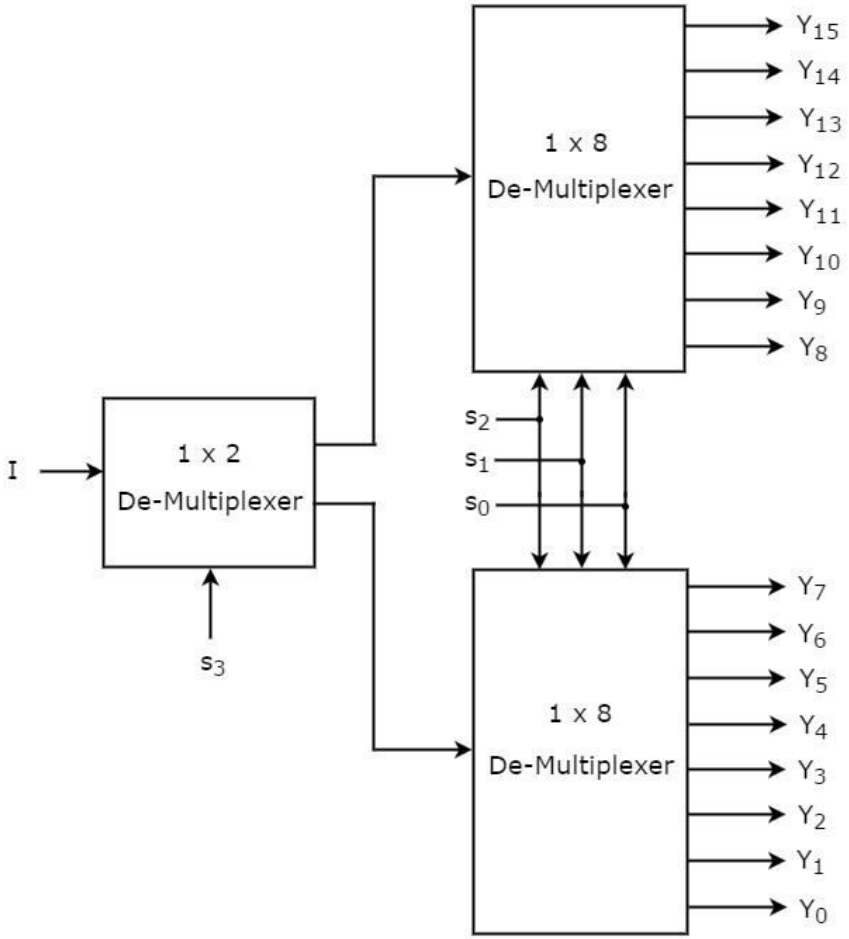
Self Check Exercise

- Implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.
- Implement 1x8 De-Multiplexer using lower order Multiplexers.
- Implement 1x16 De-Multiplexer using lower order Multiplexers

1x8 De-Multiplexer using lower order Multiplexers



1x16 De-Multiplexer using lower order Multiplexers



Programmable Logic Devices(PLDs)

- A programmable logic device is an IC that is user configurable and is capable of implementing logic functions.
- It is an LSI chip that contains a ‘regular’ structure and allows the designer to customize it for any specific application.
- Advantages over fixed function ICs
  - Low development cost
  - Less space requirement
  - Less power requirement
  - High reliability
  - Easy circuit testing
  - Easy design modification
  - High design security

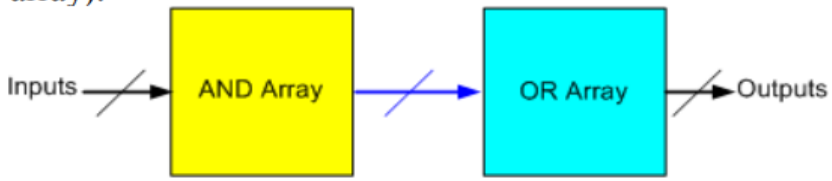
Programmable Logic Device (PLD)

- A logic device is an electronic component which performs a definite function which is decided at the time of manufacture and will never change.
- For example, a not gate always inverts the logic level of the input signal and does/can-do-nothing else.
- On the other hand, **Programmable Logic Devices** (PLDs) are the components which do not have a specific function associated with them.
- These can be configured to perform a certain function by the user, on a need basis and can further be changed to perform some other function at the later point of time, i.e. these are re-configurable. However, the amount of flexibility offered depends on their type.

- Programmable Logic Devices **(PLDs)** are the integrated circuits which contain an array of AND gates & another array of OR gates.
- The process of entering the information into these devices is known as **programming**.
- Here, the term programming refers to hardware programming but not software programming.
- The internal logic gates AND/OR connections of PLDs can be changed/configured by a programming process

- One of the simplest programming technologies is to use fuses.
- In the original state of the device, all the fuses are intact.
- Programming the device involves blowing those fuses along the paths that must be removed in order to obtain the particular configuration of the desired logic function.

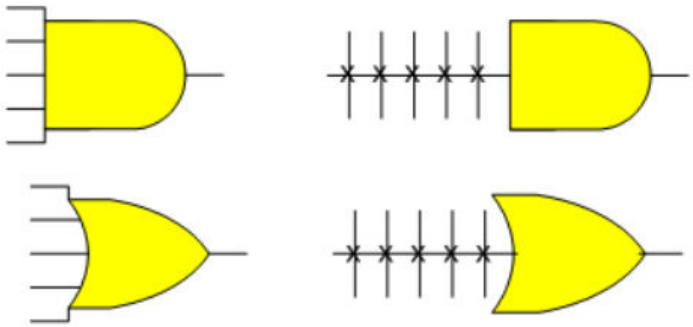
PLDs are typically built with an *array* of AND gates (AND-array) and an *array* of OR gates (OR-array).



Array logic

- A typical programmable logic device may have hundreds to millions of gates interconnected through hundreds to thousands of internal paths.
- In order to show the internal logic diagram in a concise form, it is necessary to employ a special gate symbol applicable to array logic.

Figure shows the conventional and array logic symbols for a multiple input AND and a multiple input OR gate.



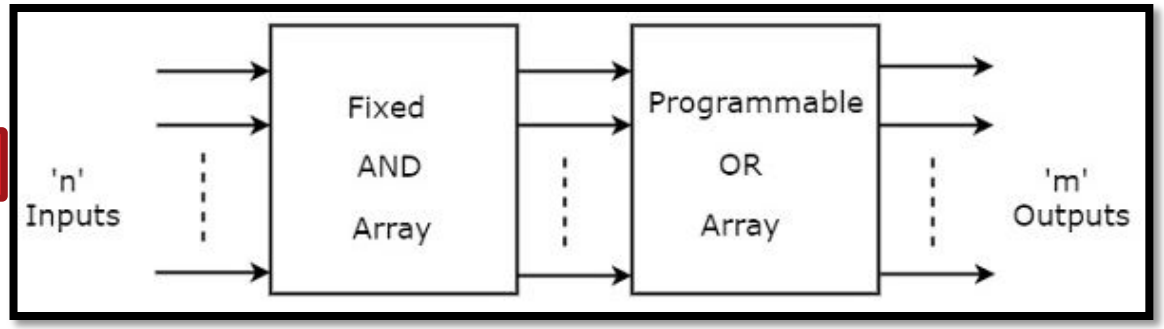
(a) Conventional Symbol

(b) Array Logic Symbol

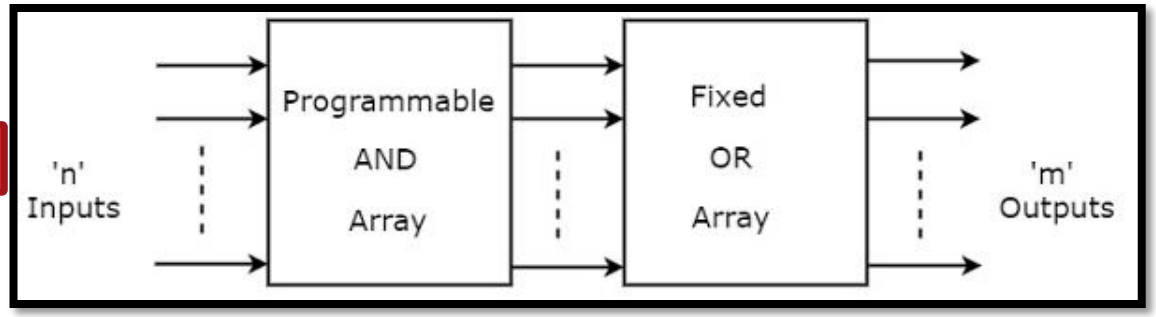
Types

- Programmable Read Only Memory (PROM)
- Programmable Array Logic (PAL)
- Programmable Logic Array (PLA)

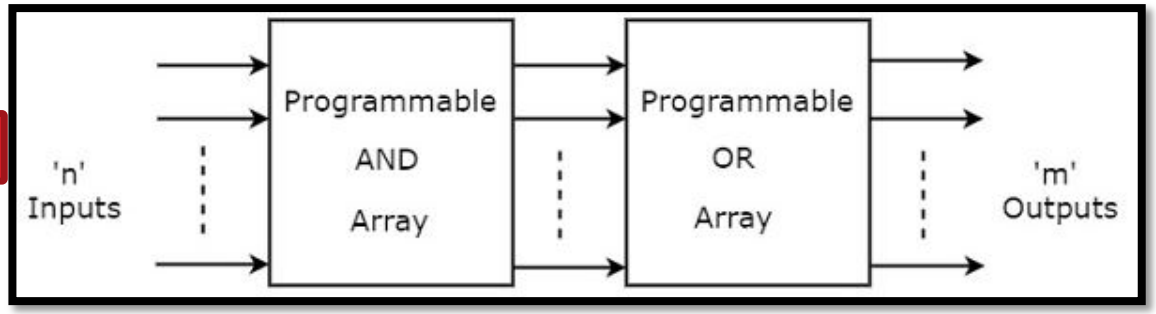
PROM


'n'
Inputs
Fixed
AND
Array
Programmable
OR
Array
'm'
Outputs

PAL


'n'
Inputs
Programmable
AND
Array
Fixed
OR
Array
'm'
Outputs

PLA


'n'
Inputs
Programmable
AND
Array
Programmable
OR
Array
'm'
Outputs

Summary

ORGANIZATION	AND ARRAY	OR ARRAY
PROM	FIXED	PROG.
PAL	PROG.	FIXED
PLA	PROG.	PROG.

x denotes Programmable Connection

● denotes Fixed Connection

SC Exercise

- Advantages of using PLDs

Read-Only Memory (ROM)

- A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored.
- 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 unit when the power is turned off and on again.
- A ROM which can be programmed is called a PROM. The process of entering information in a ROM is known as programming.
- ROMs are used to store information which is of fixed type, such as tables for various functions, fixed data and instructions.
- ROMs can be used for designing combinational logic circuits.

Read-Only Memory

- Advantages of using a ROM as a PLD
  - Ease of design since no simplification or minimization of logic function is required
  - Designs can be changed, modified easily
  - Faster than discrete MSI/SSI circuit
  - Cost is reduced
- Disadvantages of ROM based circuits
  - Non-utilization of complete circuit
  - Increased power requirement
  - Enormous increase in size with increase in number of input variables

Types of ROM

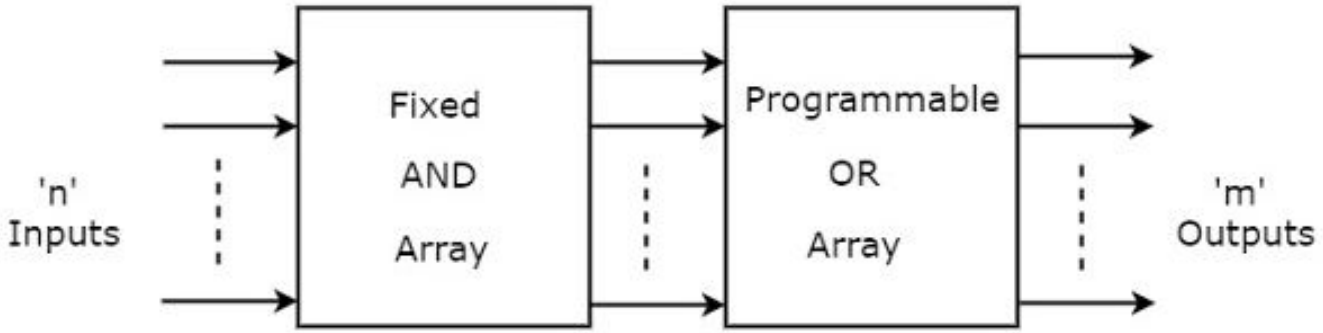
- Mask programmable read-only memory (MROM)
- Programmable read-only memory (PROM)
- Erasable read-only memory (EPROM)
- Electrically erasable and programmable read-only memory (EEPROM or E^2PROM)

Programmable Read Only Memory (PROM)

- Programmable ROM is a one-time programmable chip that, once programmed, cannot be erased or altered.
- PROM is also referred as **One Time Programmable(OTP)**
- These memories are bought in blank form and are programmed using a special PROM programmer.
- Typically a PROM will consist of an array(collection) of fusible links some of which are “blown” during the programming process to provide the required data pattern.
- For every bit of PROM, these exist a fuse. PROM is programmed by blowing the fuse.

- A typical PROM comes with all bits reading as "1".
- Burning a fuse bit during programming causes the bit to read as "0".
- The memory can be programmed just once after manufacturing by "blowing" the fuses, which is an irreversible process.
- In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1.

- PROM is a programmable logic device that has fixed AND array & Programmable OR array.
- The **block diagram** of PROM is shown in the following figure.



- Here, the inputs of AND gates are not of programmable type. So, we have to generate 2^n product terms by using 2^n AND gates having n inputs each. We can implement these product terms by using $nx2^n$ decoder. So, this decoder generates ‘n’ **min terms**.
- Here, the inputs of OR gates are programmable. That means, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of **sum of min terms**.

PROM Organization

- n inputs – provide address for the memory
- m outputs – data bits of the stored word selected by address
- n address input lines specify 2^n words
- ROM does not have data inputs because it does not have write operation.

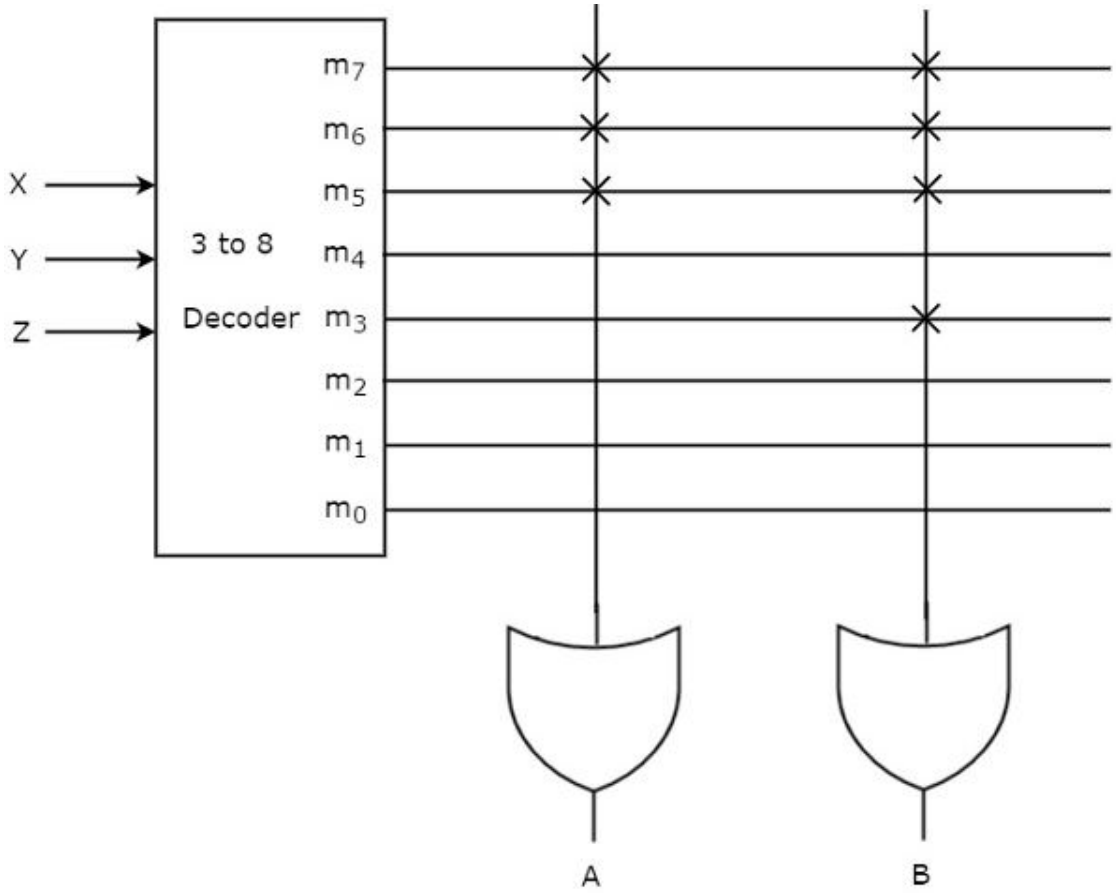
Example-1

- Let us implement the following **Boolean functions** using PROM

$$A(X, Y, Z) = \sum m\,(5, 6, 7)$$

$$B(X, Y, Z) = \sum m\,(3, 5, 6, 7)$$

- The given two functions are in sum of min terms form and each function is having three variables X, Y & Z. So, we require a 3 to 8 decoder and two programmable OR gates for producing these two functions.


X
Y
Z
3 to 8
Decoder
m7
m6
m5
m4
m3
m2
m1
m0
A
B

- Here, 3 to 8 decoder generates eight min terms. The two programmable OR gates have the access of all these min terms.
- But, only the required min terms are programmed in order to produce the respective Boolean functions by each OR gate.
- The symbol ‘X’ is used for programmable connections.

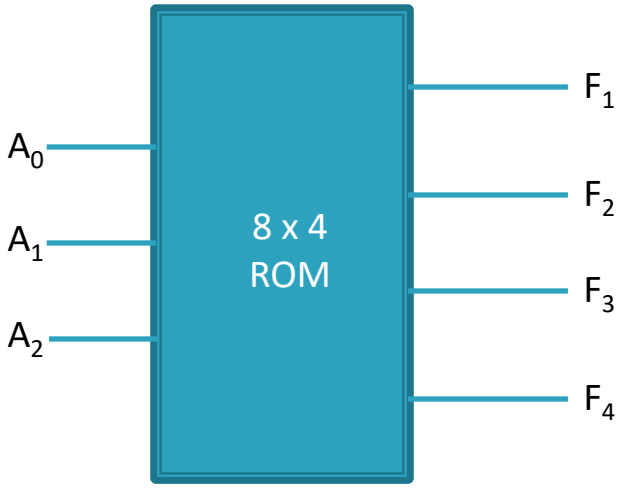
Example-2

- Implement following functions using PROM.

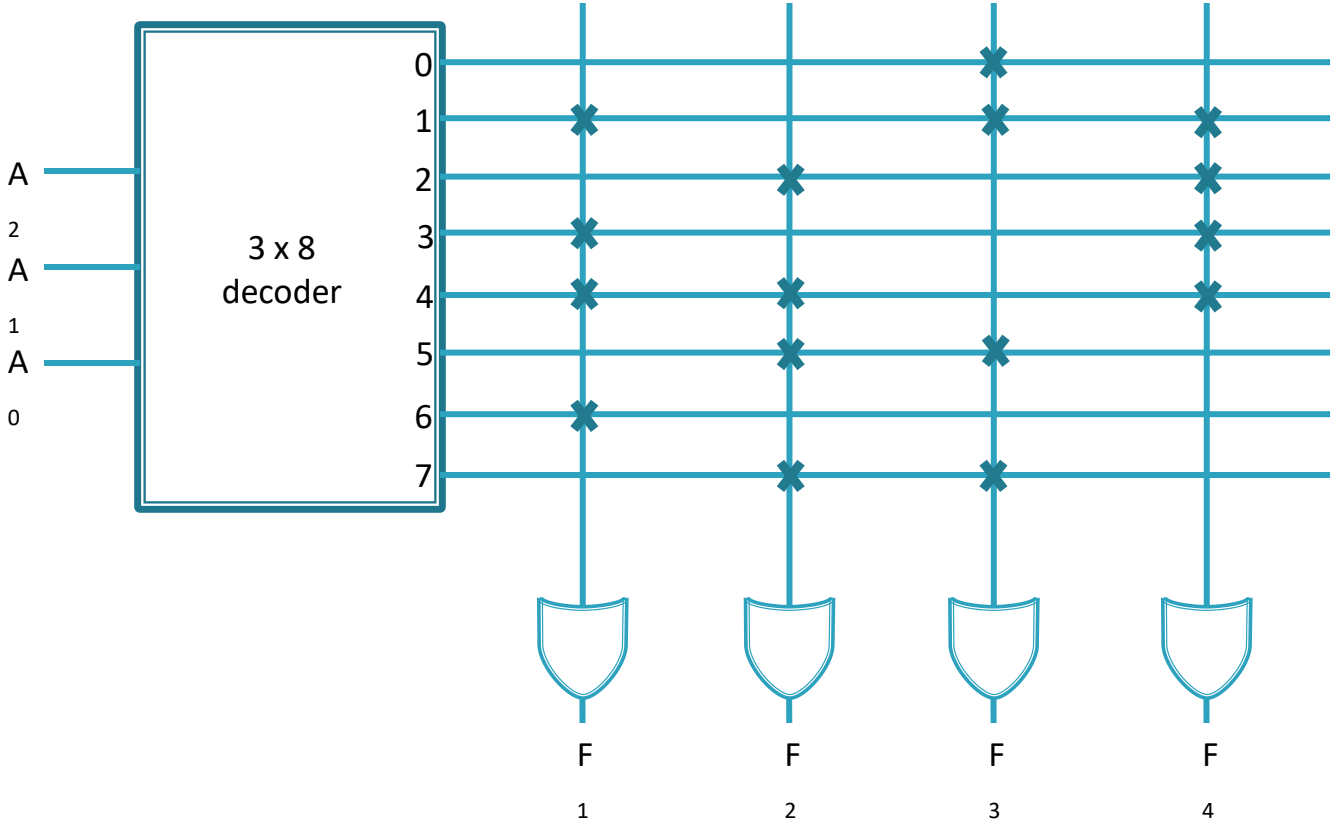
$F1 = \sum m(1, 3, 4, 6)$ $F2 = \sum m(2, 4, 5, 7)$

$F3 = \sum m(0, 1, 5, 7)$ $F4 = \sum m(1, 2, 3, 4)$

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	1	1	0	0	1
1	0	0	1	1	0	1
1	0	1	0	1	1	0
1	1	0	1	0	0	0
1	1	1	0	1	1	0



Example-2



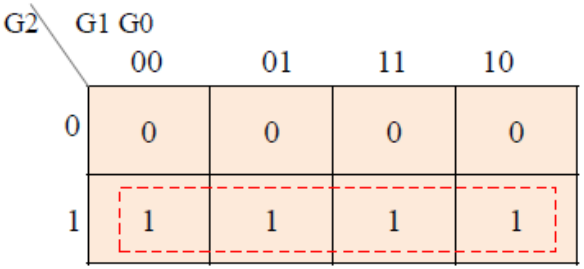
EXAMPLE-3: Design a combinational circuit using PROM to convert gray code into binary code.

SOLUTION:

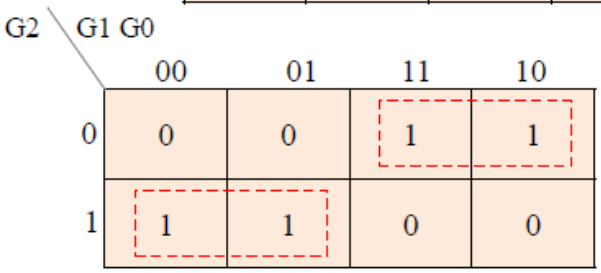
STEP 1: Truth Table for Grey to Binary code

GRAY INPUT			BINARY OUTPUT (Y)		
G2	G1	G0	B2	B1	B0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	1	0	1	0
0	1	0	0	1	1
1	1	0	1	0	0
1	1	1	1	0	1
1	0	1	1	1	0
1	0	0	1	1	1

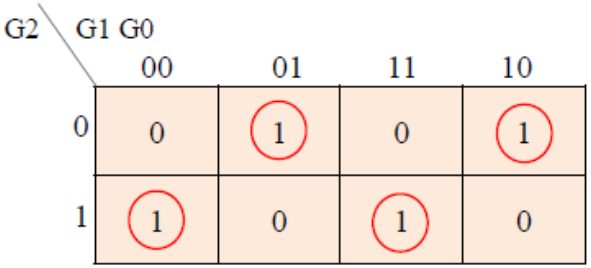
Step 2: Simplification by using K-map:



$$B_2 = G_2 \text{ ------(1)}$$



$$B_1 = G_2\,\overline{G}_1 + \overline{G}_2\,G_1 \text{ ------(2)}$$



$$B_0 = \overline{G}_2\overline{G}_1G_0 + \overline{G}_2G_1\overline{G}_0 \text{ -----(3)}$$

STEP 3:

- No. of inputs = 3
- No. of address lines (locations) = $2^3 = 8$
- Each location can store 4 bit words
- No. of outputs = 3 (B_2, B_1, B_0)

ROM Programming Table: (with the help of Eq. 1, 2 and 3)

Location No.	Inputs			Outputs		
	G_2	G_1	G_0	B_2	B_1	B_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	1
5	1	0	1	1	1	0
6	1	1	0	1	0	0
7	1	1	1	1	0	1

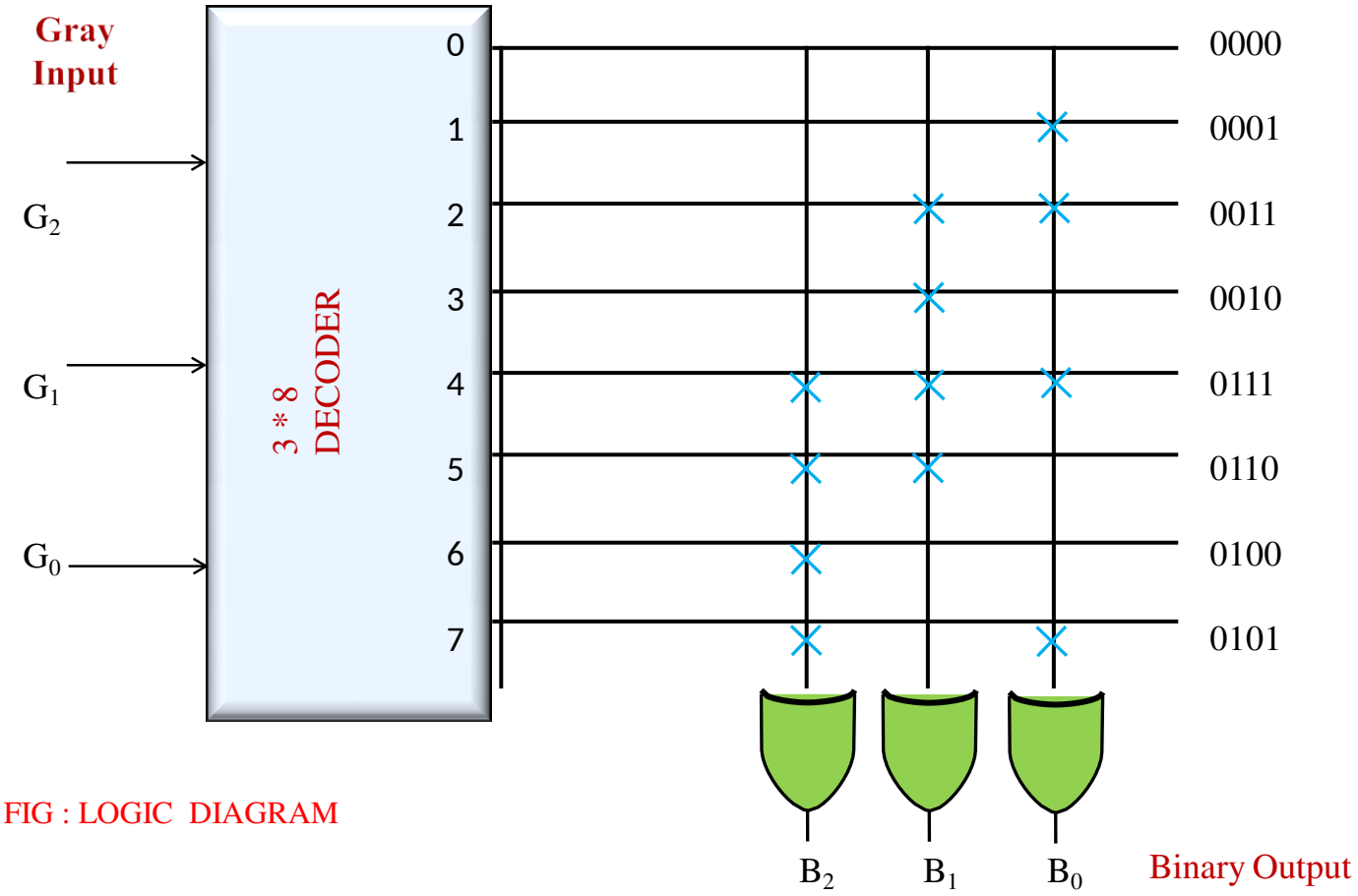
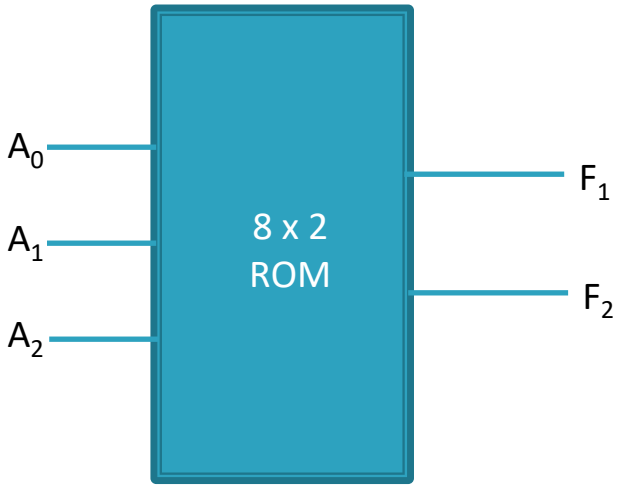


FIG : LOGIC DIAGRAM

Self Check Exercise

- Implement Full Adder Using PROM

Inputs			Outputs	
A	**B**	**C_{in}**	**S**	**C_{out}**
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



Implement Full Adder Using PROM

Block diagram :

Truth table :

i\P'S			o\P'S	
A	B	C	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

- expression for sum is given as: Sum = **∑ m (1,2,4,7)**
- and expression for carry is given as: Carry = **∑ m (3,5,6,7)**
- **Logic Diagram:** There are three input variables = A, B, C, therefore we will be using a 3:8 decoder.

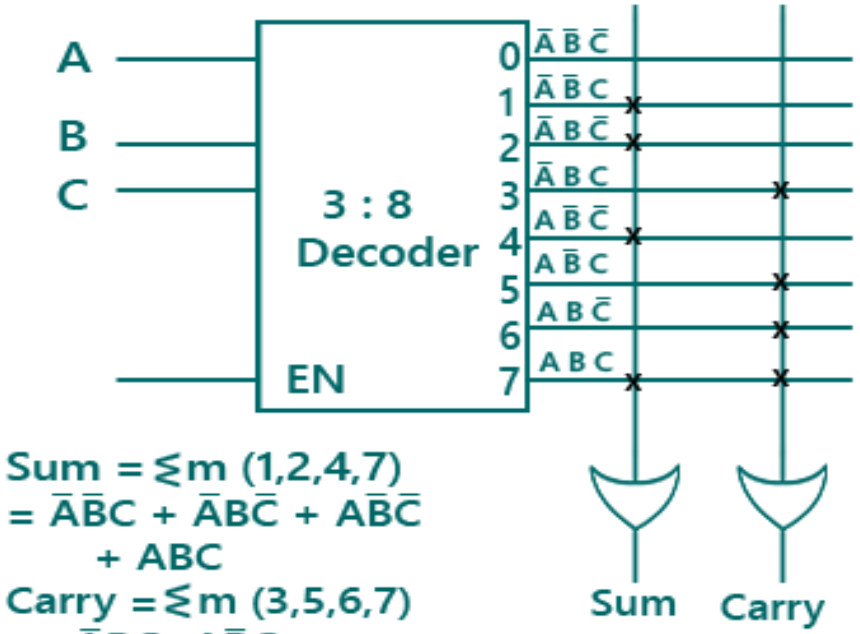
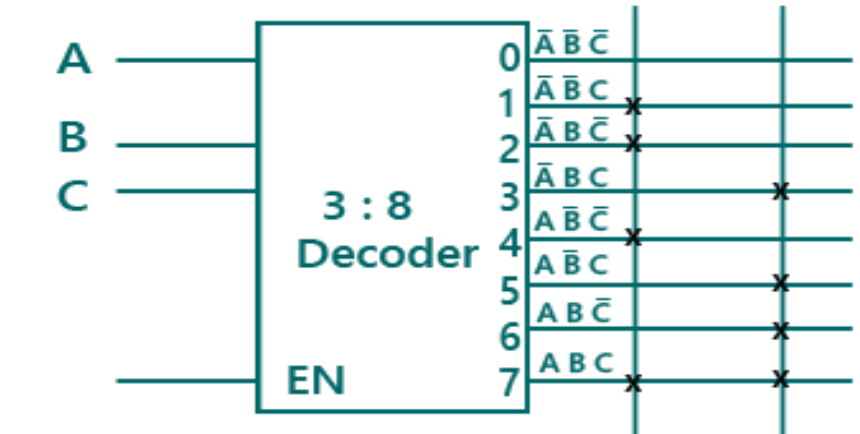


Fig . Full Adder using PROM



$Sum = \sum m\,(1,2,4,7)$
$= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$
$+ ABC$
$Carry = \sum m\,(3,5,6,7)$
$= \bar{A}BC + A\bar{B}C$
$+ AB\bar{C} + ABC$

Fig . Full Adder using PROM

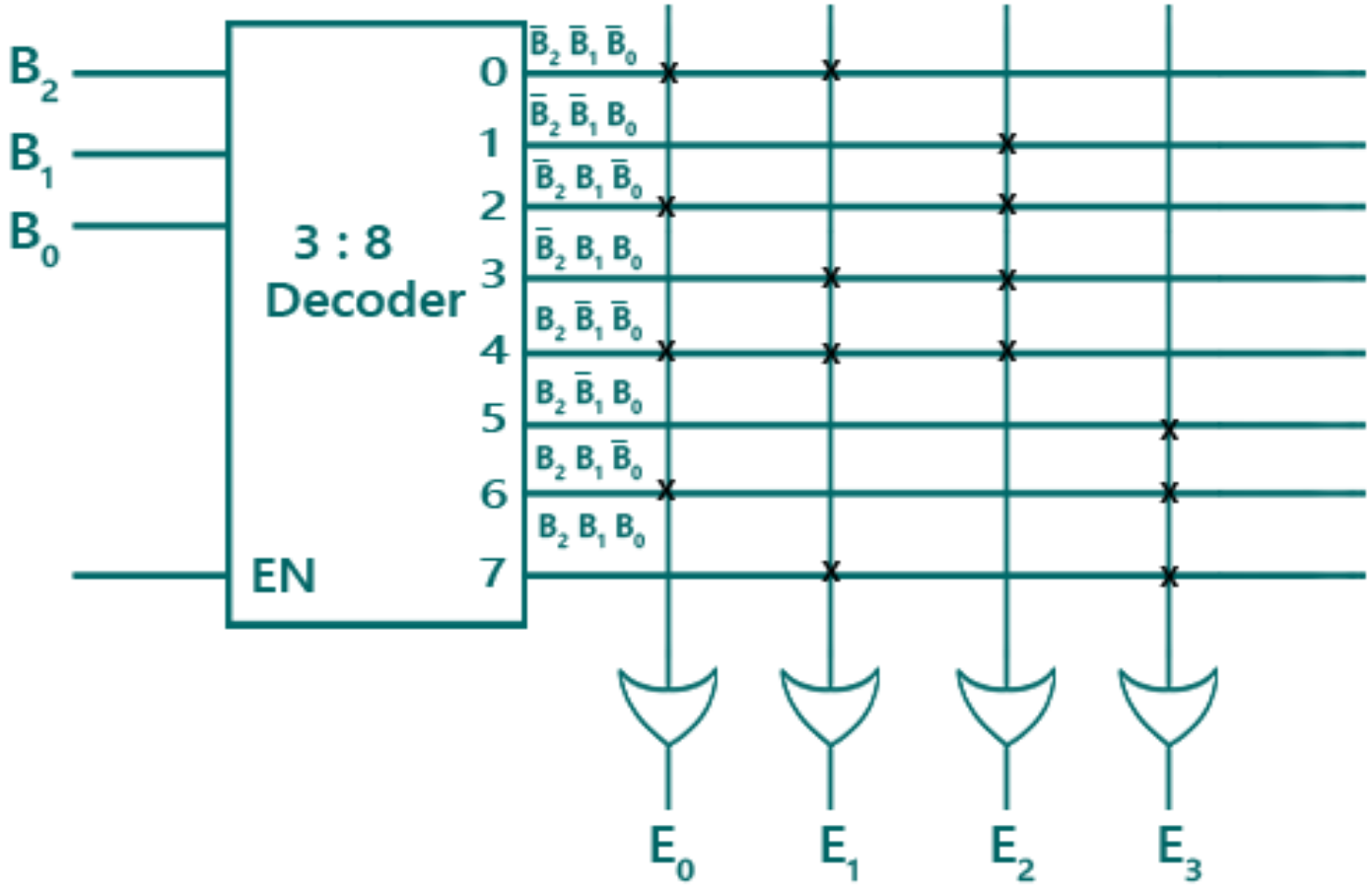
- **Design a combinational circuit using PROM, in which a 3-bit binary number is provided as input and the circuit generates its equivalent Excess-3 code.**

EXCESS-3 :-

Truth table :

I\P (3 bit binary no.)			O\P (EXCESS-3code of i\p)			
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

- $E_0 (B_2, B_1, B_0) = \sum m (0,2,4,6)$
- $E_1 (B_2, B_1, B_0) = \sum m (0,3,4,7)$
- $E_2 (B_2, B_1, B_0) = \sum m (1,2,3,4)$
- $E_3 (B_2, B_1, B_0) = \sum m (5,6,7)$
- **Logic Diagram:** Since, there are three input variables = B_2, B_1, B_0, therefore we will be using a 3:8 decoder.


B2
B1
B0
3 : 8
Decoder
EN
0
1
2
3
4
5
6
7
B̄2 B̄1 B̄0
B̄2 B̄1 B0
B̄2 B1 B̄0
B̄2 B1 B0
B2 B̄1 B̄0
B2 B̄1 B0
B2 B1 B̄0
B2 B1 B0
E0
E1
E2
E3

Self Check Exercise

1) Realize two outputs F_1 and F_2 using a 4 x 2 PROM:

$F_1 (A_1, A_0) = \sum m(0, 2)$

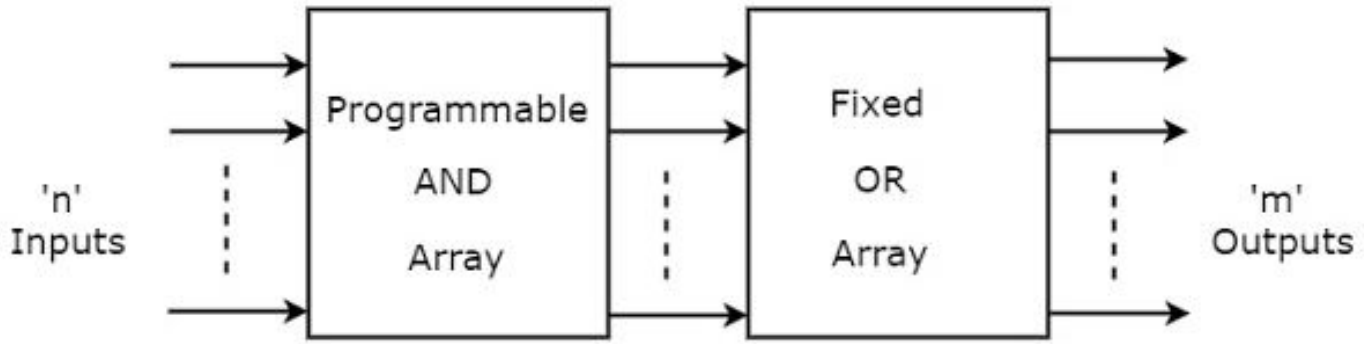
$F_2 (A_1, A_0) = \sum m(0, 1, 3)$

Programmable Array Logic (PAL)

- Programmable array logic (a registered trade mark of Monolithic Memories) is a particular family of programmable logic devices (PLDs) that is widely used and available from a number of manufacturers.
- The PAL circuits consist of a set of AND gates whose inputs can be programmed and whose outputs are connected to an OR gate, i.e. the inputs to the OR gate are hard-wired.
- The PAL is easier to program as only the AND gates are programmable.

Programmable Array Logic (PAL)

- PAL is a programmable logic device that has Programmable AND array & fixed OR array.
- The advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the min terms by using programmable AND gates.
- The **block diagram** of PAL is shown in the following figure.



- Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

- Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of **sum of products form**.

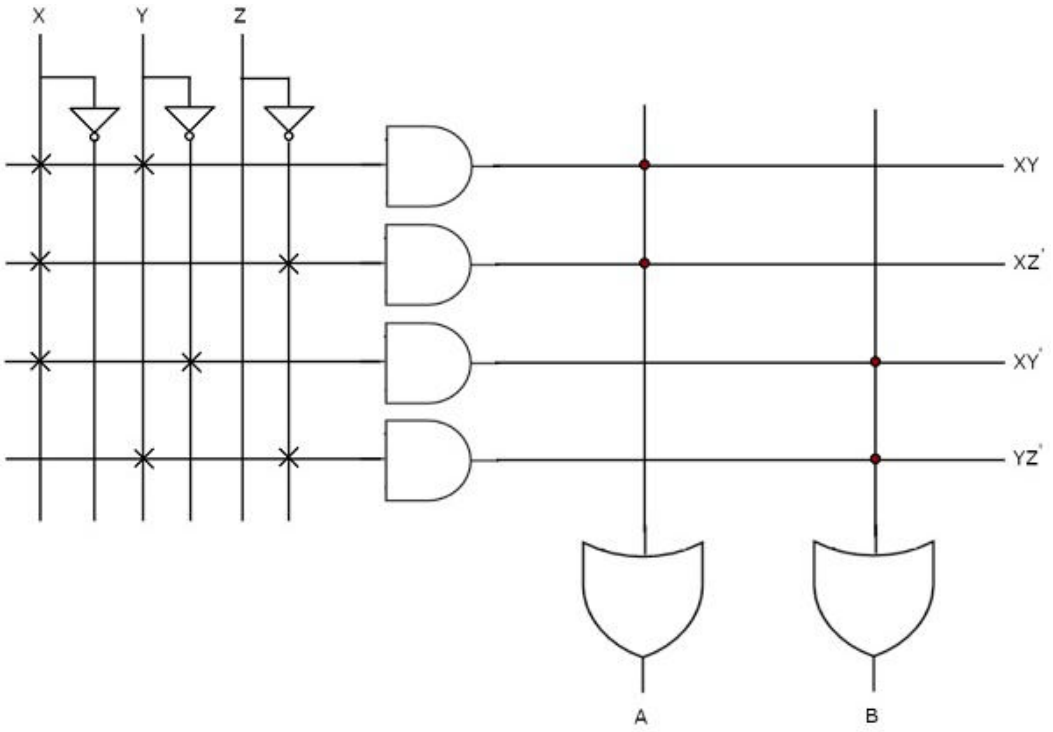
Example-1

- Let us implement the following **Boolean functions** using PAL

$$A = XY + XZ'$$

$$B = XY' + YZ'$$

- The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions.


X
Y
Z
XY
XZ'
XY'
YZ'
A
B

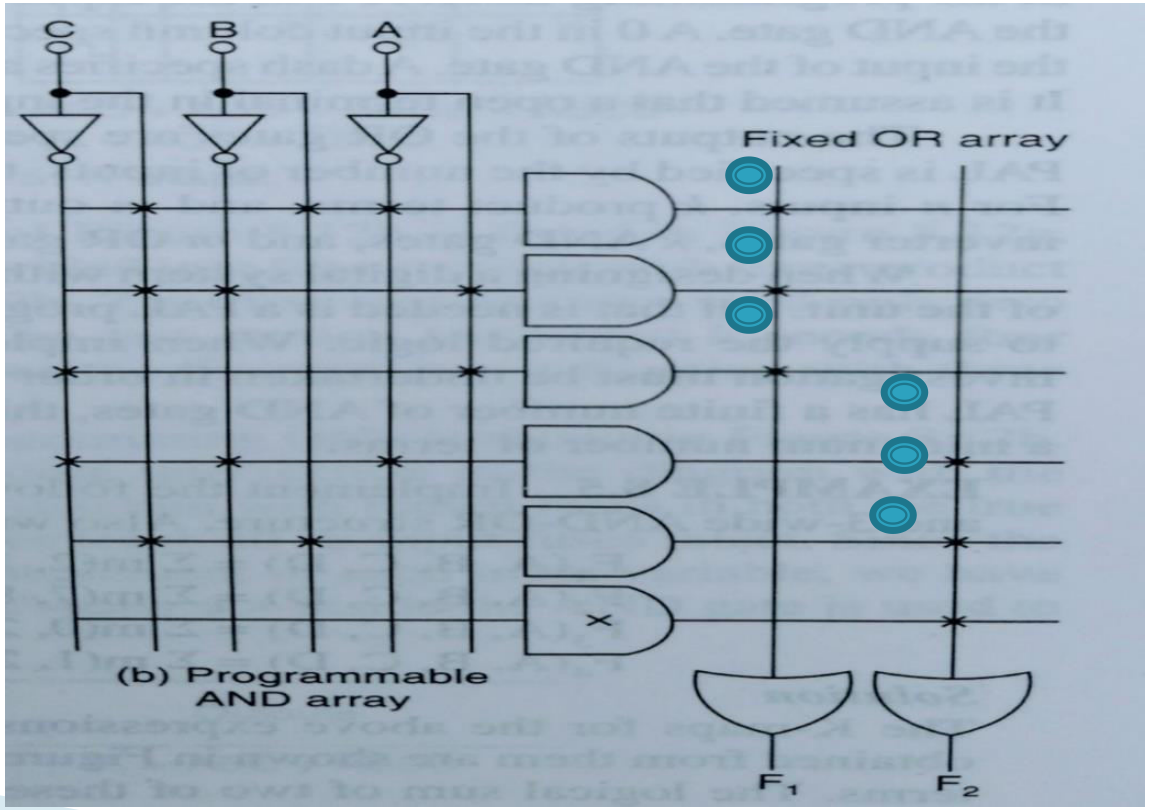
Example-2:- Implement following functions using PAL.
F1 = A'BC + AC' + AB'C, F2 = A'B'C' + BC

- Class Work

Example-2

- Implement following functions using PAL.

F1 = A'BC + AC' + AB'C F2 = A'B'C' + BC



Example-3

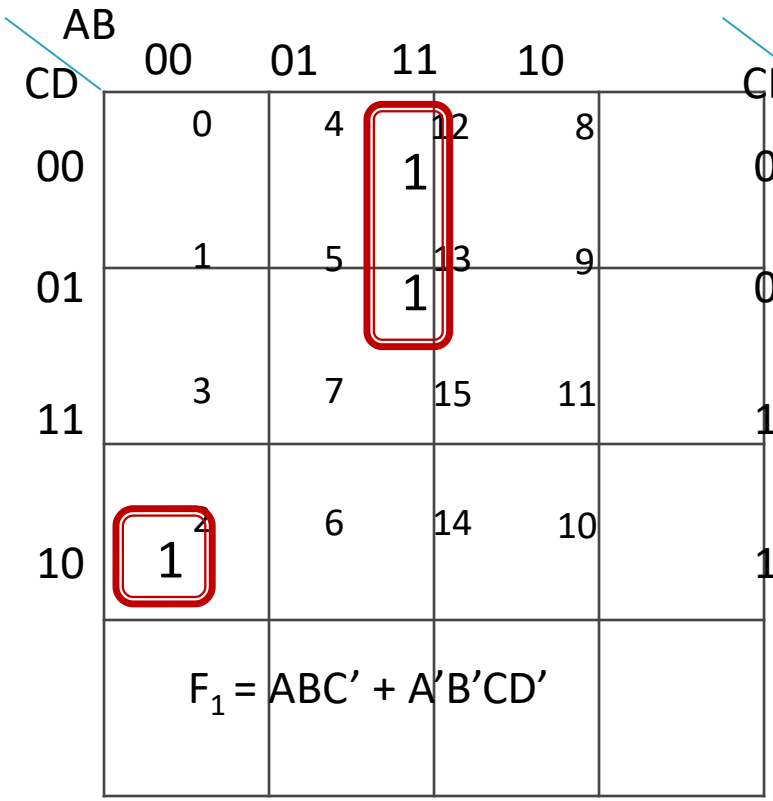
- Implement the following Boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

  $F_1(A, B, C, D) = \sum_m(2, 12, 13)$

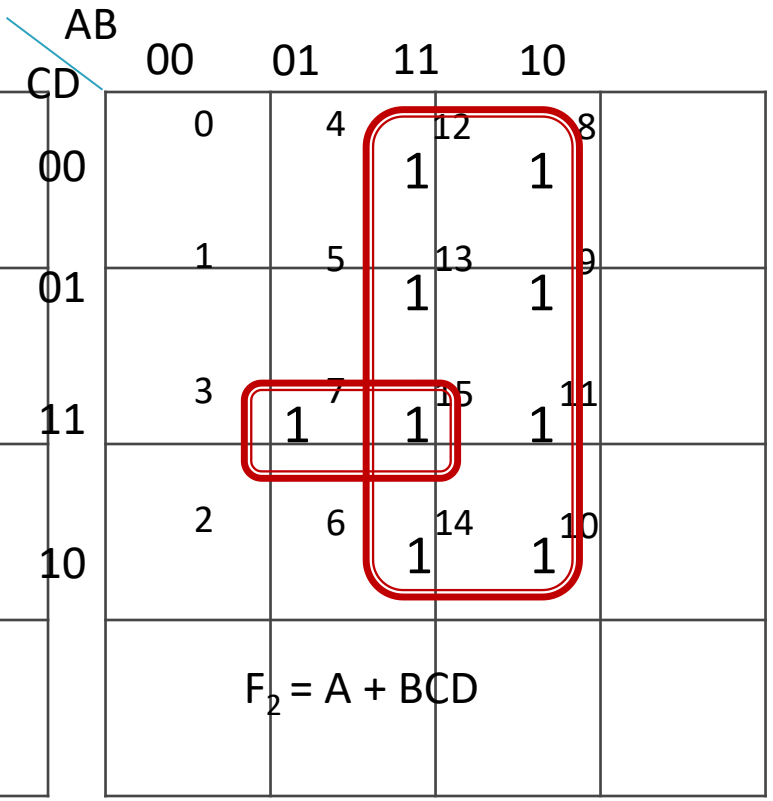
  $F_2(A, B, C, D) = \sum_m(7, 8, 9, 10, 11, 12, 13, 14, 15)$

  $F_3(A, B, C, D) = \sum_m(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$

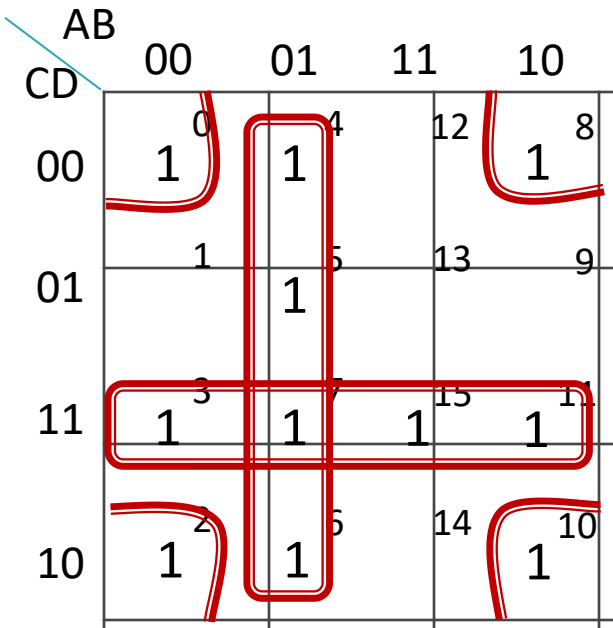
  $F_4(A, B, C, D) = \sum_m(1, 2, 8, 12, 13)$



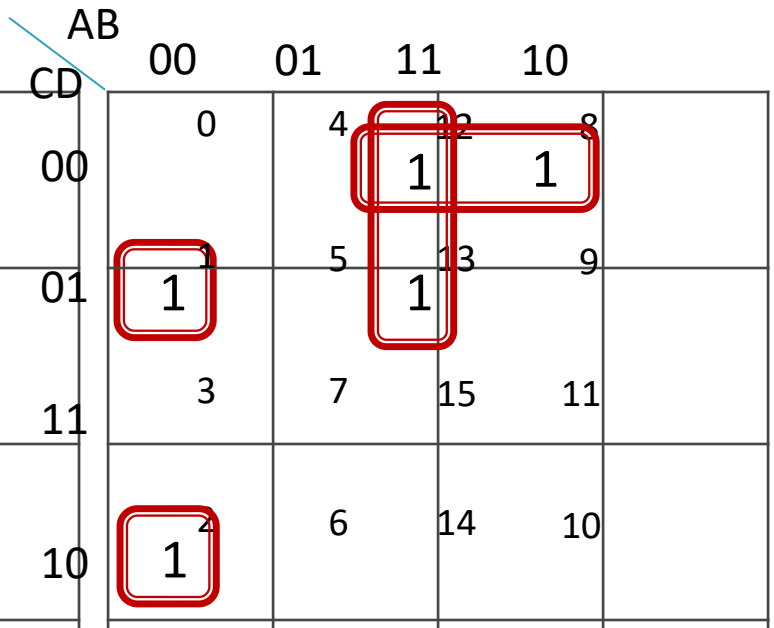
$F_1 = ABC' + A'B'CD'$



$F_2 = A + BCD$



$F_3 = CD + A'B + B'D'$

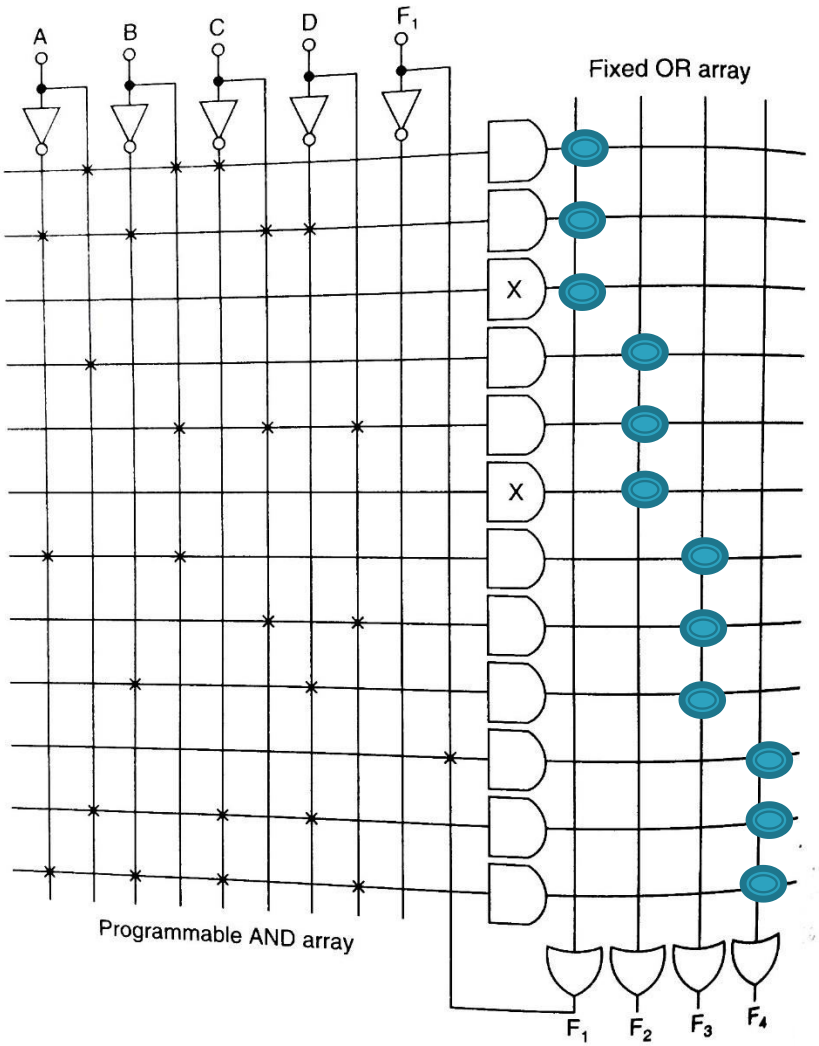


$F_4 = ABC' + AC'D' + A'B'C'D + A'B'CD'$

$= F_1 + AC'D' + A'B'C'D$

PAL Programming Table

Product term		AND Inputs					Outputs
		A	B	C	D	F_1	
1	ABC’	1	1	0	-	-	$F_1 = ABC' + A'B'CD'$
2	A’B’CD’	0	0	1	0	-	
3	-	-	-	-	-	-	
4	A	1	-	-	-	-	$F_2 = A + BCD$
5	BCD	-	1	1	1	-	
6	-	-	-	-	-	-	
7	A’B	0	1	-	-	-	$F_3 = CD + A'B + B'D'$
8	CD	-	-	1	1	-	
9	B’D	-	0	-	0	-	
10	-	-	-	-	-	1	$F_4 = F_1 + AC'D' + A'B'C'D$
11	AC’D’	1	-	0	0	-	
12	A’B’C’D	0	0	0	1	-	


A
B
C
D
F1
Fixed OR array
X
X
Programmable AND array
F1
F2
F3
F4

SC Exercise

- Realize the following functions using a PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

  $F_1(A, B, C, D) = \sum_m(6, 8, 9, 12, 13, 14, 15)$

  $F_2(A, B, C, D) = \sum_m(1, 4, 5, 6, 7, 10, 11, 12, 13)$

  $F_3(A, B, C, D) = \sum_m(4, 5, 6, 7, 10, 11)$

  $F_4(A, B, C, D) = \sum_m(4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15)$

SC Exercise

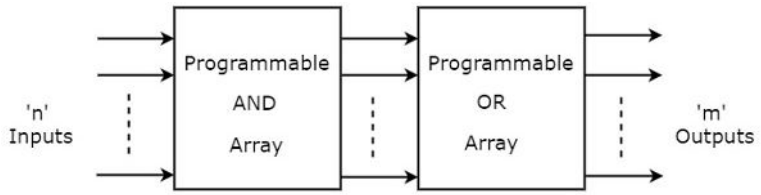
- Implementation of full adder using PAL

Programmable Logic Array (PLA)

- The PLA combines the characteristics of the PROM and the PAL by providing both a programmable OR array and a programmable AND array, i.e. in a PLA both AND gates and OR gates have fuses at the inputs.
- A third set of fuses in the output inverters allows the output function to be inverted if required.

Programmable Logic Array (PLA)

- PLA is a programmable logic device that has both Programmable AND array & Programmable OR array.
- Hence, it is the most flexible PLD.
- The **block diagram** of PLA is shown in the following figure.



- Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

- Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of **sum of products form**.

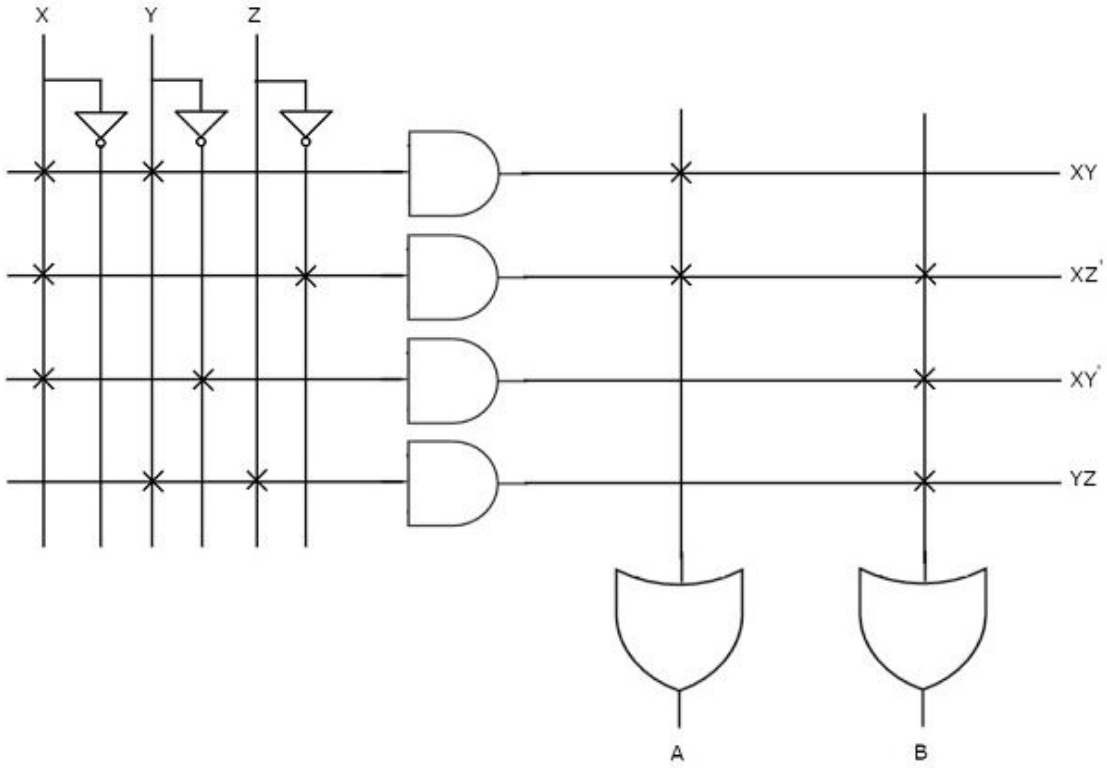
Example-1

- Let us implement the following **Boolean functions** using PLA.

$$A = XY + XZ'$$

$$B = XY' + YZ + XZ'$$

- The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, is common in each function.
- So, we require four programmable AND gates & two programmable OR gates for producing those two functions.

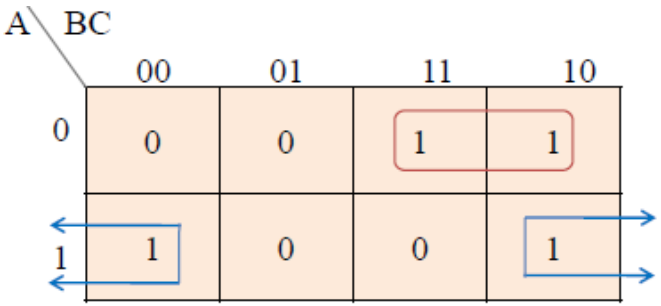

X
Y
Z
XY
XZ'
XY'
YZ
A
B

Example-2: Implement the circuit with PLA of the following functions

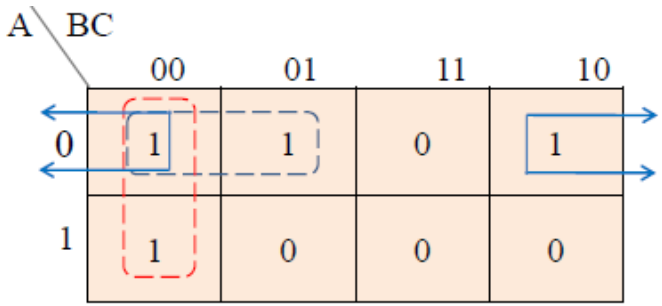
$$F_1(A,B,C) = \sum m\,(2,3,6,4)$$

$$F_2(A,B,C) = \sum m\,(0,1,2,4)$$

- Solution:
  - STEP 1: Write the function in minimize SOP form. (Here we are using K- map)

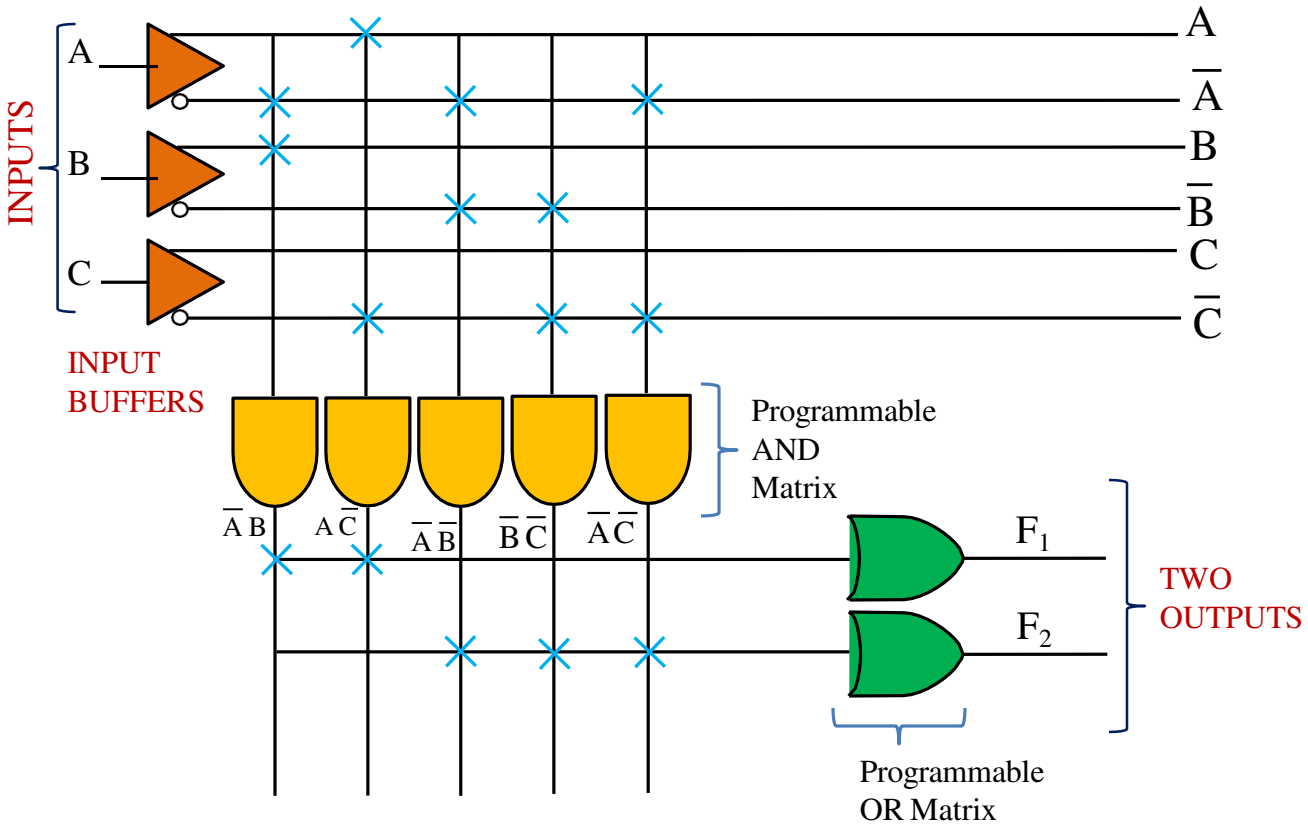


$$F_1 = \overline{A}B + A\overline{C}$$



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

LOGIC DIAGRAM



EXAMPLE -3: A combinational circuit is defined by the following function:

$F1\ (A,B,C)=\sum m(4,5,7)$

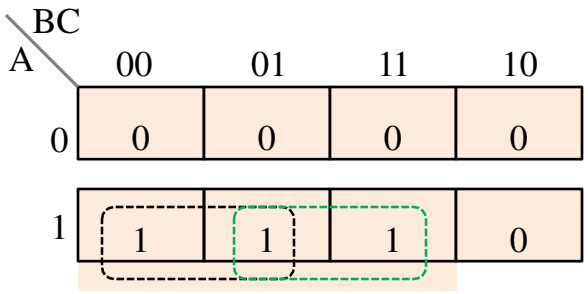
$F_2\ (A,B,C)=\sum m(3,5,7)$

Implement this circuit with a PLA having 3 inputs, 3 product terms and 2 outputs. Also write the PLA programming table.

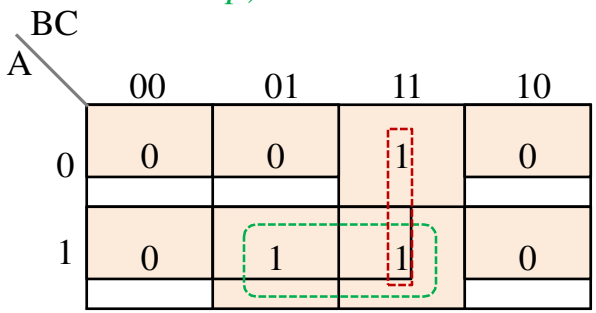
SOLUTION:

STEP 1: Write the Boolean Expression in minimum SOP form.

(Note: use minimization technique. Here we use K-map)



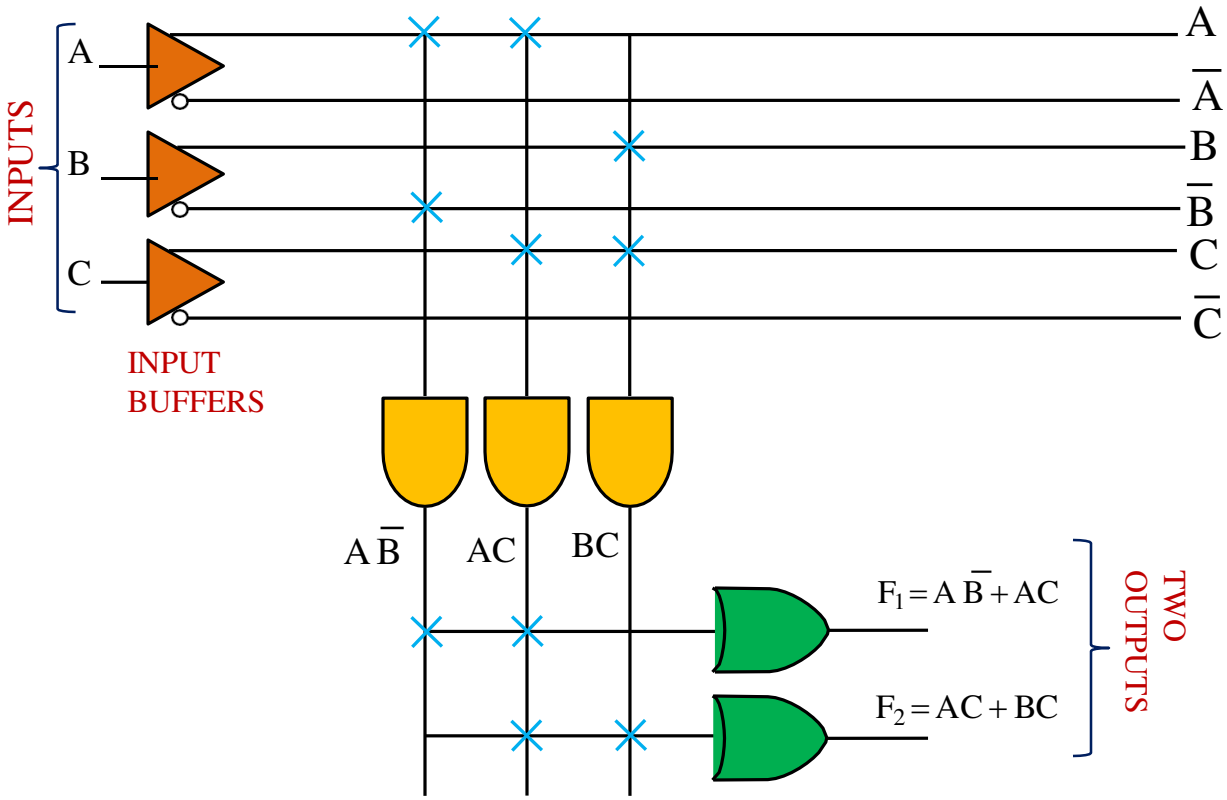
$F_1 = A\,\overline{B} + AC$



$F_2 = AC + BC$

- Number of inputs = 3
- Number of product terms = 4, in which ‘AC’ is common in both function F_1 and F_2 , so number of product terms = 3
- Number of AND gates = Number of product terms = 3
- Number of OR gates = Number of outputs = 2
- Logic diagram is shown in fig

STEP-2 : LOGIC DIAGRAM



PLA Programming Table:

PRODUCT TERMS		INPUTS			OUTPUTS	
		A	B	C	F_1	F_2
1	A B'	1	0	-	1	-
2	AC	1	-	1	1	1
3	BC	-	1	1	-	1

Example-4 : Derive PLA Program Table for below truth table

A	B	C	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
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	1
1	1	1	1	1	0	0

Solution

K-map for F1:

	B′C′	B′C	BC	BC′
A			1	1
A′				

F1= AB

K-map for F2:

	B′C′	B′C	BC	BC′
A	1	1	1	
A′				

F2= AB’+AC

K-map for F3:

	B′C′	B′C	BC	BC′
A		1		
A′			1	

F3= AB’C+A’BC

K-map for F4:

	B′C′	B′C	BC	BC′
A				1
A′				1

F4= BC′

PLA Program Table

Product Term		Input			Output			
		A	B	C	F1	F2	F3	F4
1	AB	1	1	-	1	-	-	-
2	AB’	1	0	-	-	1	-	-
3	AC	1	-	1	-	1	-	-
4	AB’C	1	0	1	-	-	1	-
5	A’BC	0	1	1	-	-	1	-
6	BC’	-	1	0	-	-	-	1

Example-5

- Implement the following two Boolean functions with a PLA:

$F_1(A, B, C) = \sum_m(0, 1, 2, 4)$

$F_2(A, B, C) = \sum_m(0, 5, 6, 7)$

Also draw the Program Table

Class Work

EXAMPLE 0: A combinational logic is defined by the functions

$$F_1 = A\,\overline{B} + A\,C + \overline{A}\,B\,\overline{C}$$

$$F_2 = \overline{(AC + BC)}$$

Implement the circuit with PLA having three inputs, 4 product terms and two outputs.

(Reference: Digital Design by M. Moris Mano, Michael D. Cilleti)

- The size of the PLA is specified by the number of inputs, the number of product terms and the number of outputs.
- PLA Programming Table:

PRODUCT TERMS		INPUTS			OUTPUTS (T) (C)	
		A	B	C	F_1	F_2
1	A B'	1	0	-	1	-
2	A C	1	-	1	1	1
3	B C	-	1	1	-	1
4	A' BC'	0	1	0	1	-

'T' stands for TRUE and 'C' stands for COMPLEMENT

Logic Diagram

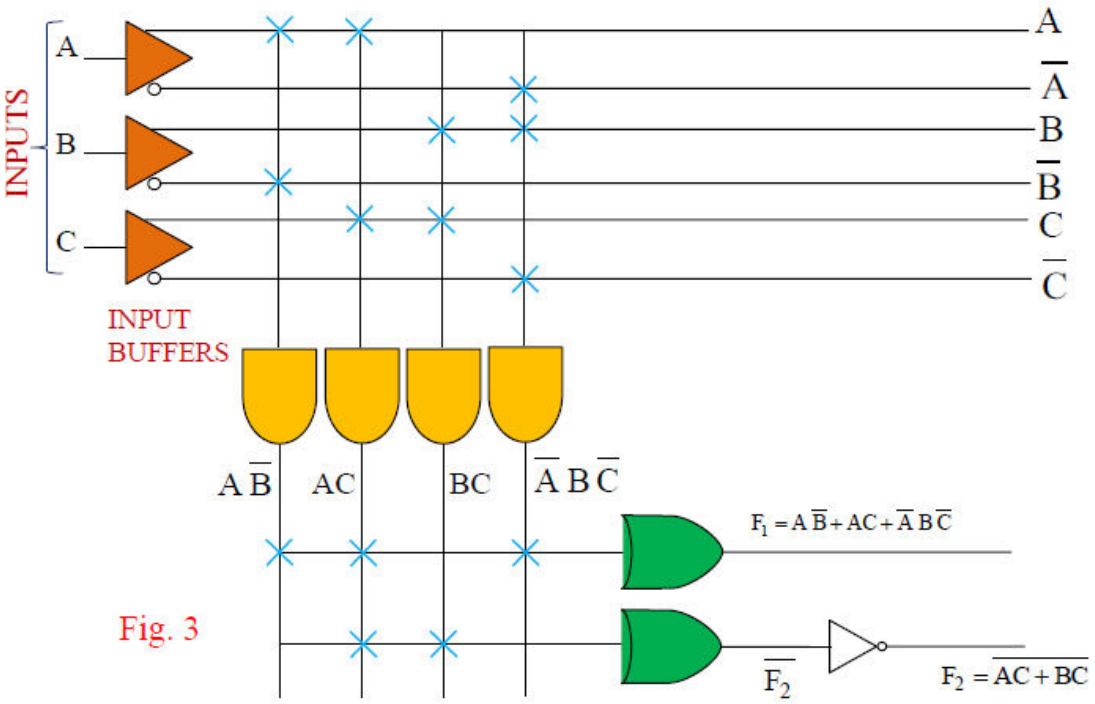


Fig. 3

PROM, PLA and PAL

Comparison of PROM, PLA and PAL

PROM	PLA	PAL
AND array is fixed and OR array is programmable.	Both AND and OR arrays are programmable.	OR array is fixed and AND array is programmable.
Cheaper and simple to use.	Costliest and complex than PAL and PROMs,	Cheaper and simpler.
All minterms are decoded.	AND array can be programmed to get desired minterms.	AND array can be programmed to get desired minterms.
Only Boolean functions in Standard SOP form can be implemented using PROM	Any Boolean functions In SOP form can be implemented using PLA.	Any Boolean functions In SOP form can be implemented using PLA.

SC Exercise

- Difference between PLA, PAL and PROM
- Advantages and disadvantages of PLA, PAL and PROM
- Application of PLA, PAL and PROM
- Implement following Boolean functions using PLA.

$F_1 = A'B + AC' + A'BC'$

$F_2 = AC' + B'C$

Old Question

18. Write difference between PLA and PAL. Design a PLA circuit with given functions. 10 Marks

F1 (A, B, C) = Σ (2, 3, 5)

F2 (A, B, C) = Σ (0, 4, 5, 7). Design PLA program table also.[3+7]

9. Define PLA. Design a PLA circuit with given functions. 10 Marks

F1 (A, B, C) = Σ (3, 5, 6, 7)

F2 (A, B, C) = Σ (0, 2, 4, 7). Design PLA program table also. [3 + 7]

9. Define PLA with its block diagram.Realize BCD to gray code converter using PAL. 10 Marks

Implement Seven Segment Decoder Using PAL