Subject : Design of Digital Circuits

B.Tech Semester - 3
Department of Computer Engineering
Dharmsinh Desai University, Nadiad

Taught by :

Prof. Jatayu Baxi,
Assistant Professor,
Computer Engineering Department,
DDU, Nadiad.
E-Mail : jatayubaxi.ce@ddu.ac.in

Chapter -5

Combinational Circuits using MSI and LSI

Introduction

- Combinational vs. sequential systems.
- How to decide criteria for low cost circuits?
- It is not always the case that fewer the gates, lowest is the cost of the circuits.
- With the use of integrated circuits, number of ICs, type of ICs and number of external interconnections decide overall cost of the system.

Standard Method

- Construction of circuits using standard technique is :
  - Preparation of truth table
  - Simplification
- Often very complex when number of variables are more.
- Large number of gates may be required.

Integrated Circuit

- An **integrated circuit** (also referred to as an **IC**, a **chip**, or a **microchip**) is a set of electronic circuits on one small plate ("chip") of semiconductor material, normally silicon. This can be made much smaller than a discrete circuit made from independent components.

SSI, MSI & LSI

- In SSI(Small Scale Integration) —10–100 transistors/chip or 3 - 30 gates /chip(logic gates,flip flops)
- In MSI(Medium Scale Integration) —100–1000 transistors/chip or 30 - 300 gates /chip(counters,multiplexers,registers)
- In LSI(Large Scale Integration) —1000–10,000 transistors/chip or 300 - 3000 gates /chip(8 bit processors)
- In VLSI(Very Large Scale Integration) —10,000–1,00,000 transistors/chip or morethan 3000 gates /chip.(16 bit and 32 bit processors)

Alternate Method

- Numerous MSI devices are available commercially.
- These devices perform specific digital functions commonly employed in the circuit design.
- Selection of MSI over SSI is preferable as it results into reduction of IC packages.

What you will learn in this chapter ?

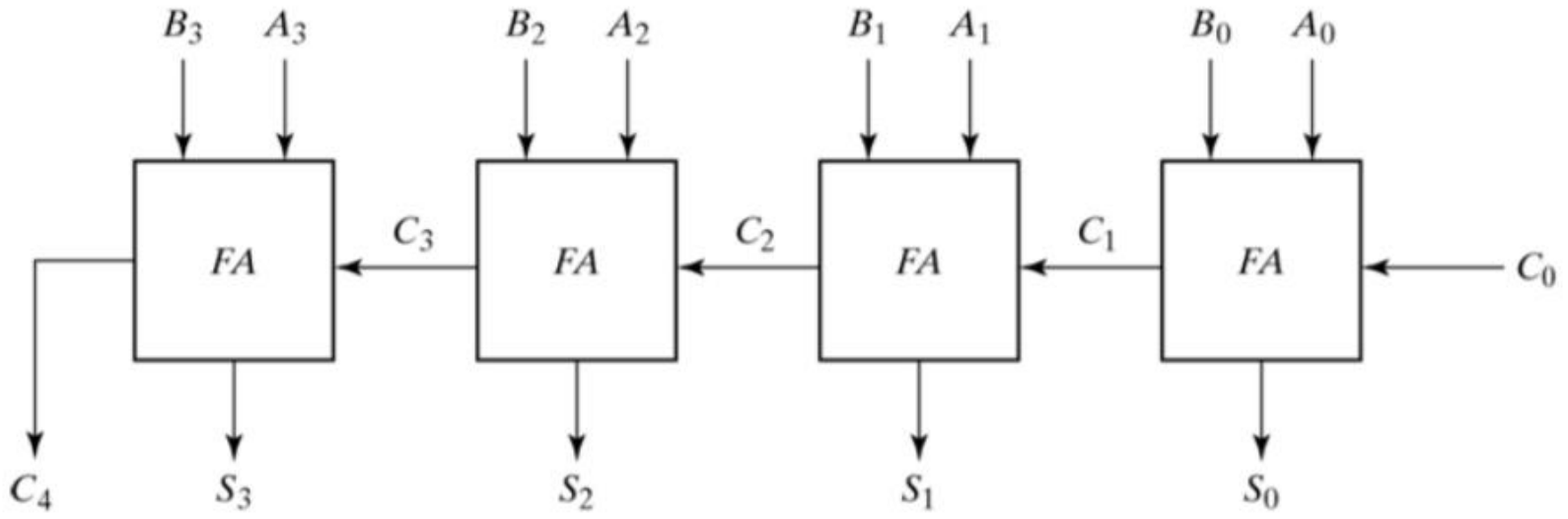
- Examples of combinational circuits with the methods other than classic methods.
- Internal constructions of existing MSI functions.
- Design of combinational circuits using MSI and LSI with the help of
  - Decoder, multiplexer, ROM, PLA (programmable logic array)

4-Bit adder Circuit

- A Full adder is used to add two (1-bit) numbers.

Like 1+ 0

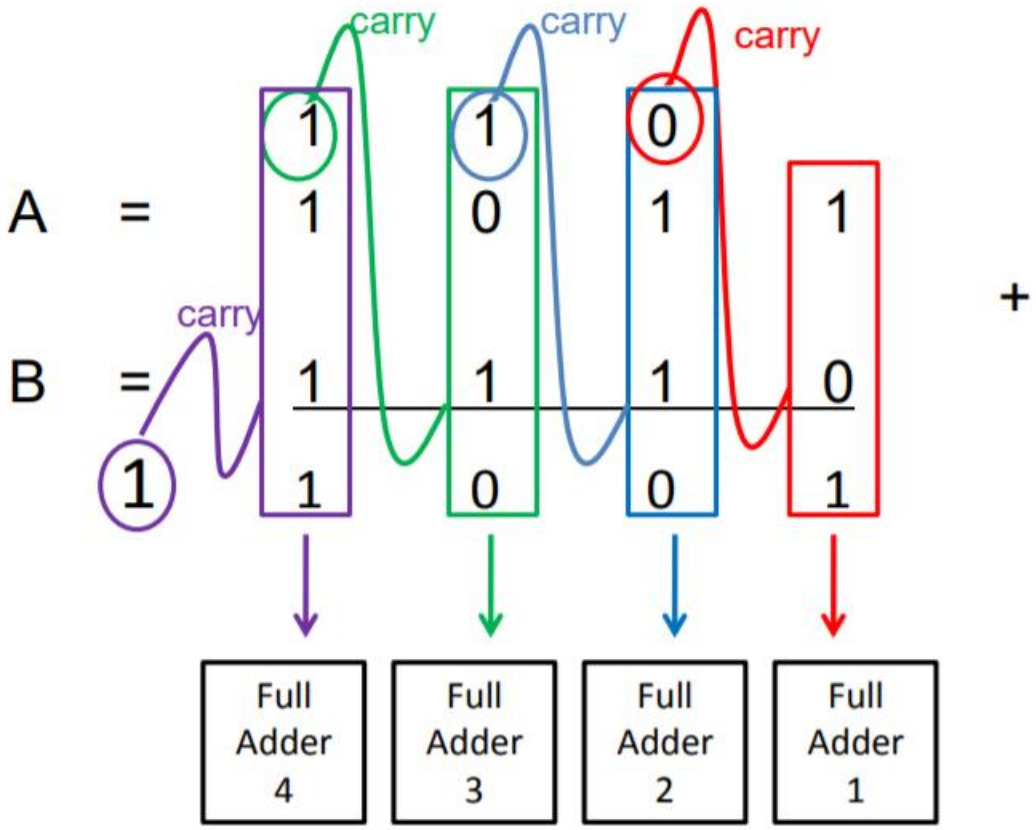
- If we want to sum two (4-bit) numbers: like 1110+ 1011
  - We use 4 (1-bit)adders.



4-Bit Adder

Example

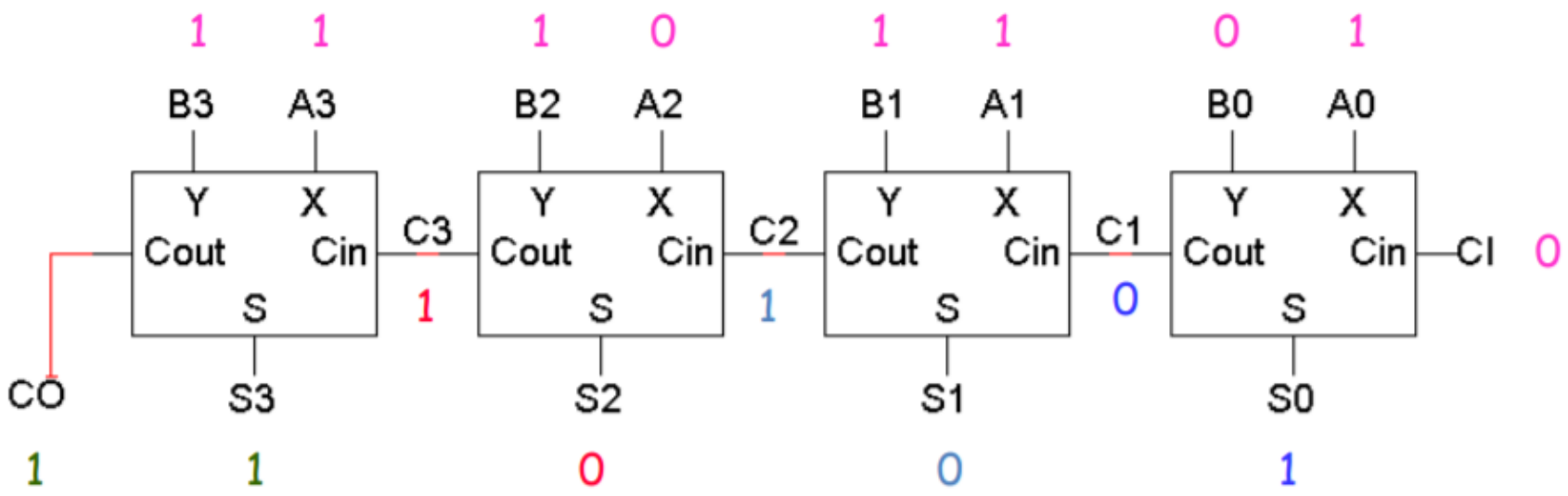
- If we want to find A+B, where A=1011 , B=1110



+

To implement this We use:

– Find A+B, where A=1011 (eleven), B=1110 (fourteen).

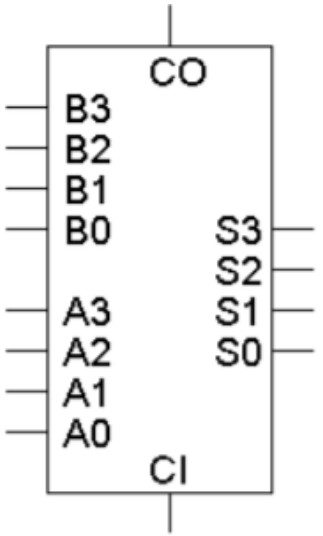


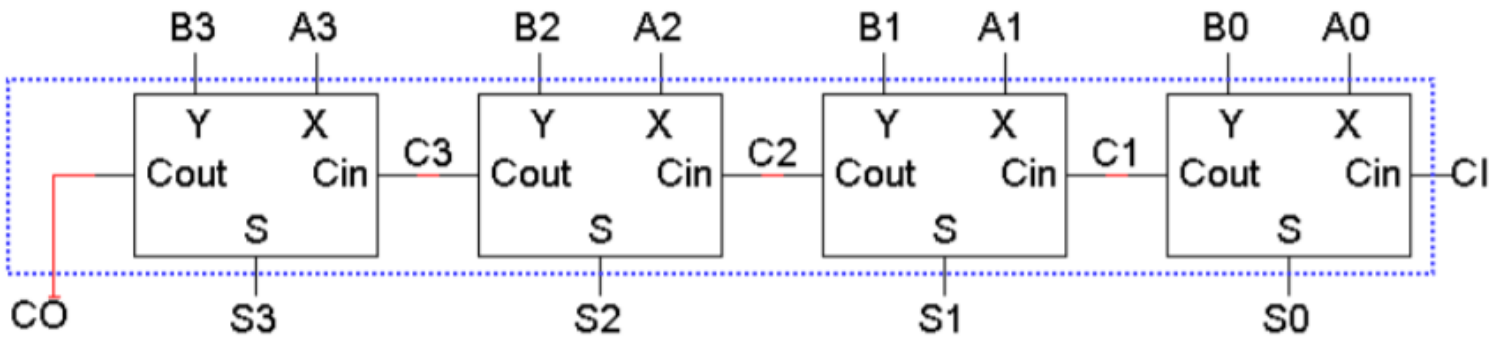
1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

Woohoo! The final answer is 11001 (twenty-five).

4 bit parallel adder

- Four full adders together make a 4-bit adder.
- There are nine total inputs:
  - Two 4-bit numbers, A3 A2 A1 A0 and B3 B2 B1 B0
  - An initial carry in, C_i
- The five outputs are:
  - A 4-bit sum, S3 S2 S1 S0
  - A carry out, C_o



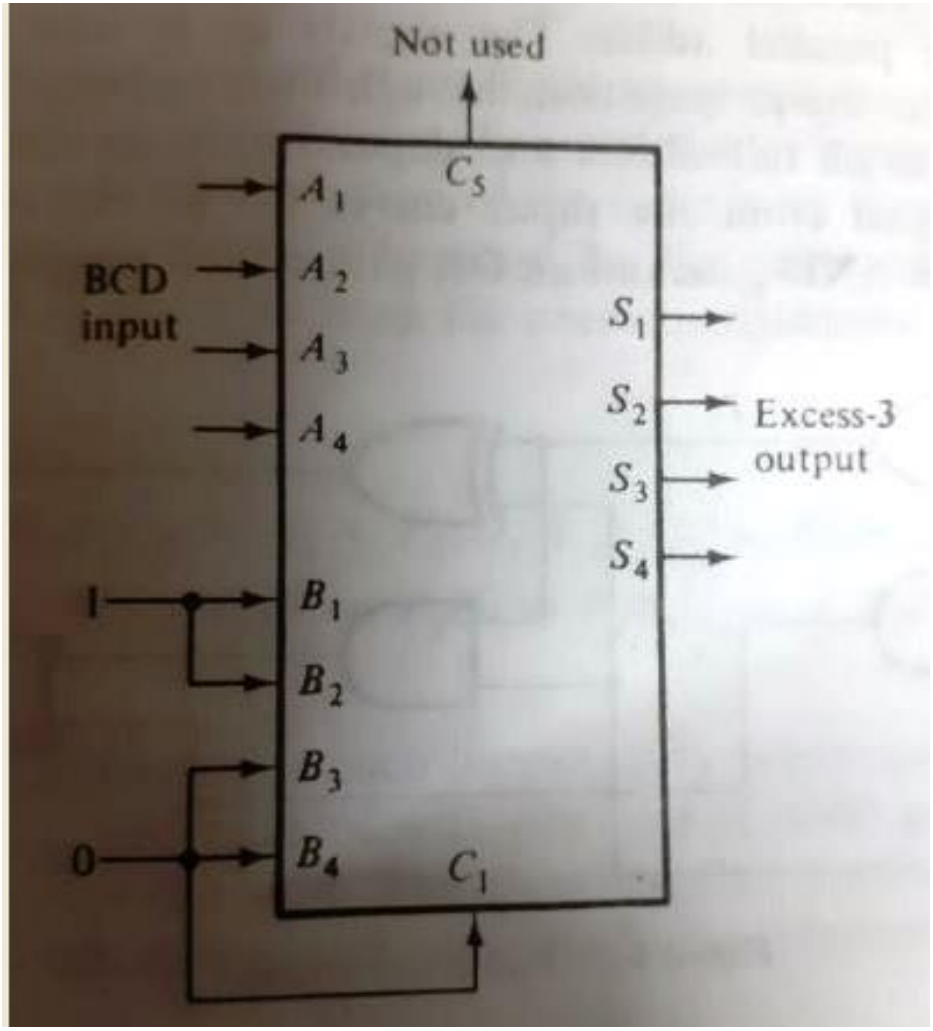


BCD to excess-3 code converter

- BCD + 0011 = excess-3 representation

- This addition can be implemented by means of 4 bit full adder.

BCD to excess-3



Ripple Carry Adder

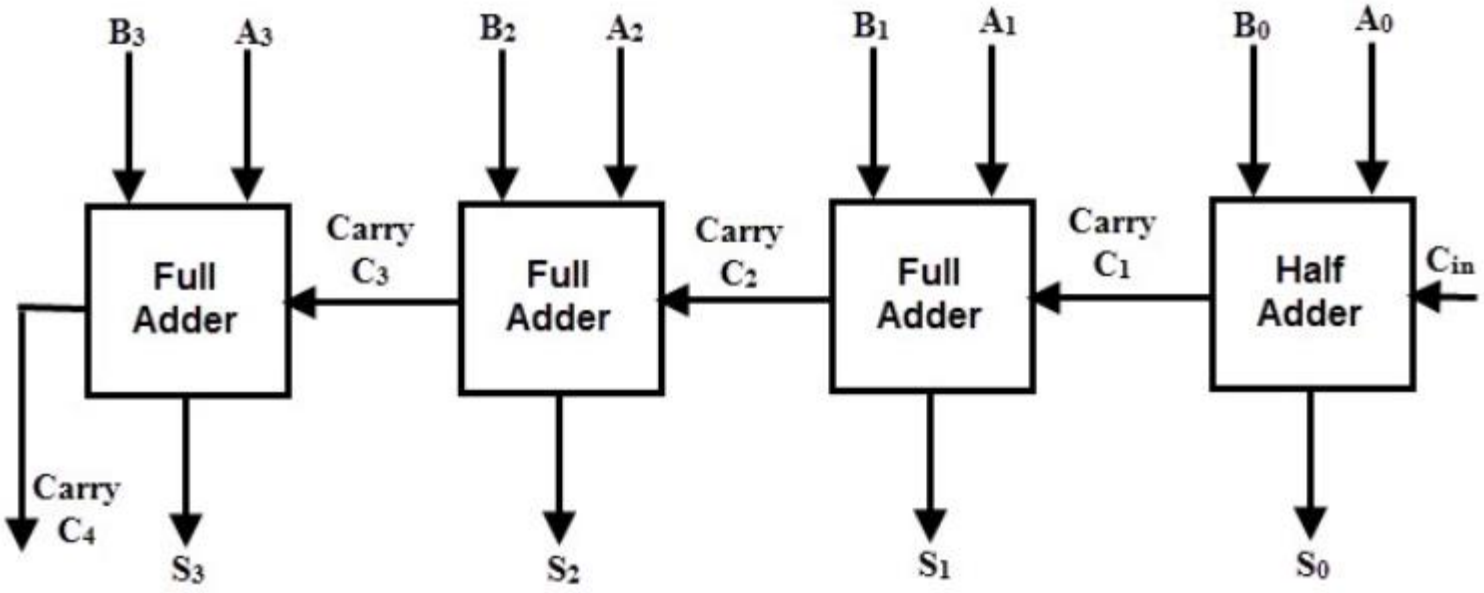
- In case of parallel adders, the binary addition of two numbers is initiated when all the bits of the augend and the addend must be available at the same time to perform the computation. In a parallel adder circuit, the carry output of each full adder stage is connected to the carry input of the next higher-order stage, hence it is also called as ripple carry type adder.

Carry Propagation Delay

- In such adder circuits, it is not possible to produce the sum and carry outputs of any stage until the input carry occurs. So there will be a considerable time delay in the addition process , which is known as , carry propagation delay.

- sum S4 is produced by the corresponding full adder as soon as the input signals are applied to it. But the carry input C4 is not available on its final steady state value until carry c3 is available at its steady state value. Similarly C3 depends on C2 and C2 on C1. Therefore, carry must propagate to all the stages in order that output S4 and carry C5 settle their final steady-state value.

Parallel Adder



- The propagation time is equal to the propagation delay of the typical gate times the number of gate levels in the circuit. For example, if each full adder stage has a propagation delay of 20n seconds, then S4 will reach its final correct value after 80n (20 × 4) seconds. If we extend the number of stages for adding more number of bits then this situation becomes much worse.

- So the speed at which the number of bits added in the parallel adder depends on the carry propagation time.

Ways to reduce delay

- By employing faster gates with reduced delays, we can reduce the propagation delay. But there will be a capability limit for every physical logic gate.
- Another way is to increase the circuit complexity in order to reduce the carry delay time. There are several methods available to speeding up the parallel adder, one commonly used method employs the principle of look ahead-carry addition by eliminating inter stage carry logic.

Carry Look ahead Adder

- A carry-Lookahead adder is a fast parallel adder as it reduces the propagation delay by more complex hardware, hence it is costlier.
- This method makes use of logic gates so as to look at the lower order bits of the augend and addend to see whether a higher order carry is to be generated or not.

Full Adder Truth Table

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry Propogation

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry out is 1 in case :

- A and B both are 1
- One of A and B is 1 and Cin is 1

- $Cout = AB + (A \oplus B) Cin$

First term is called carry generation Gi and second term is called carry propagation Pi

$Cout = Gi + Pi\ Cin$

Equations

- $P_i = A_i \oplus B_i$
- Gi = Ai Bi
- Si = Pi $\oplus$ Ci
- C i +1 = Gi + Pi Ci

Carry Equations

- C1 = G0 + P0 Cin
- C2 = G1 + P1 C1

  = G1 + P1 G0 + P1 P0 Cin
- C3 = G2 + P2 C2

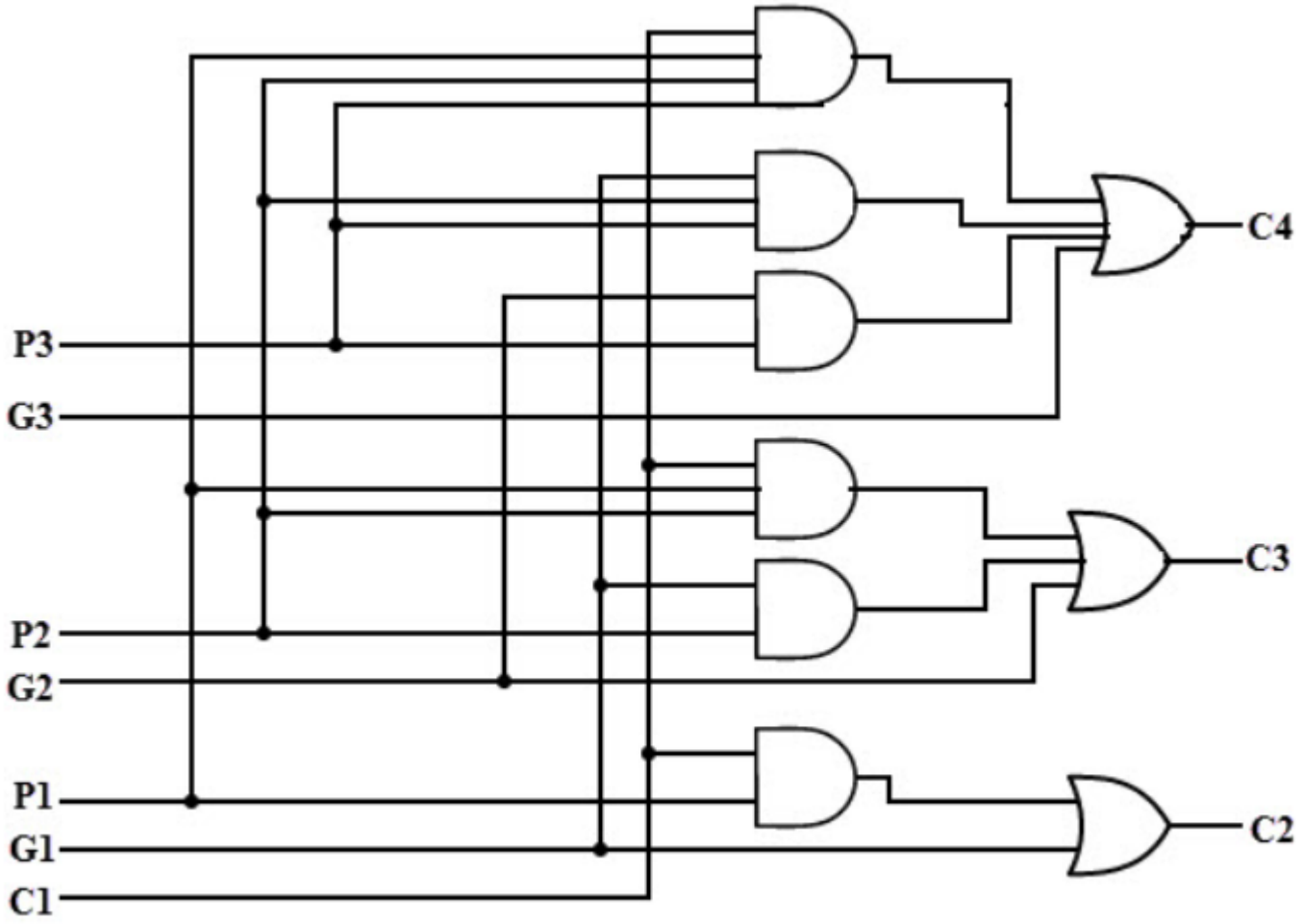
  = G2 + P2 G1+ P2 P1 G0 + P2 P1 P0 Cin
- C4 = G3 + P3 C3

  = G3 + P3 G2+ P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 Cin

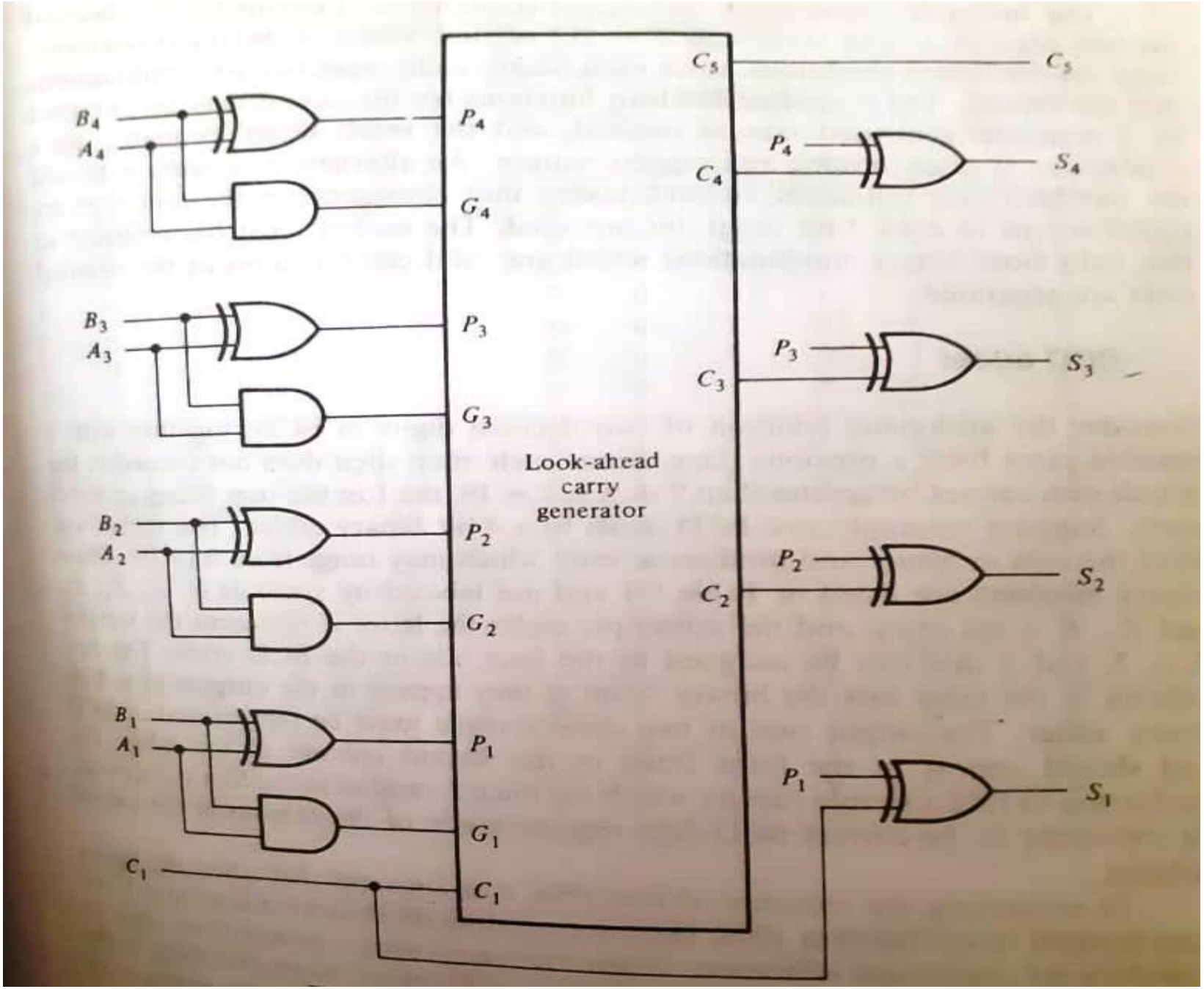
Carry Propogation

- From the above Boolean equations we can observe that C4 does not have to wait for C3 and C2 to propagate but actually C4 is propagated at the same time as C3 and C2. Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

Implementation of Carry Logic



Implementation of 4 Bit carry Look ahead Adder


B_4
A_4
P_4
G_4
B_3
A_3
P_3
G_3
B_2
A_2
P_2
G_2
B_1
A_1
P_1
G_1
C_1
C_1
Look-ahead carry generator
C_5
C_4
C_3
C_2
C_5
P_4
S_4
P_3
S_3
P_2
S_2
P_1
S_1

Decoder

- A binary code of **n bits** is capable of representing up to **2^n distinct elements** of coded information.
- A **decoder** is a **combinational circuit** that converts binary information from **n input lines** to a maximum of **2^n unique output lines**.

3 to 8 Decoder

- A 3 to 8 decoder has three inputs (A,B,C) and eight outputs (DO to D7).
- Based on the 3 inputs one of the eight outputs is selected.
- Each output represents corresponding min term.

Truth Table

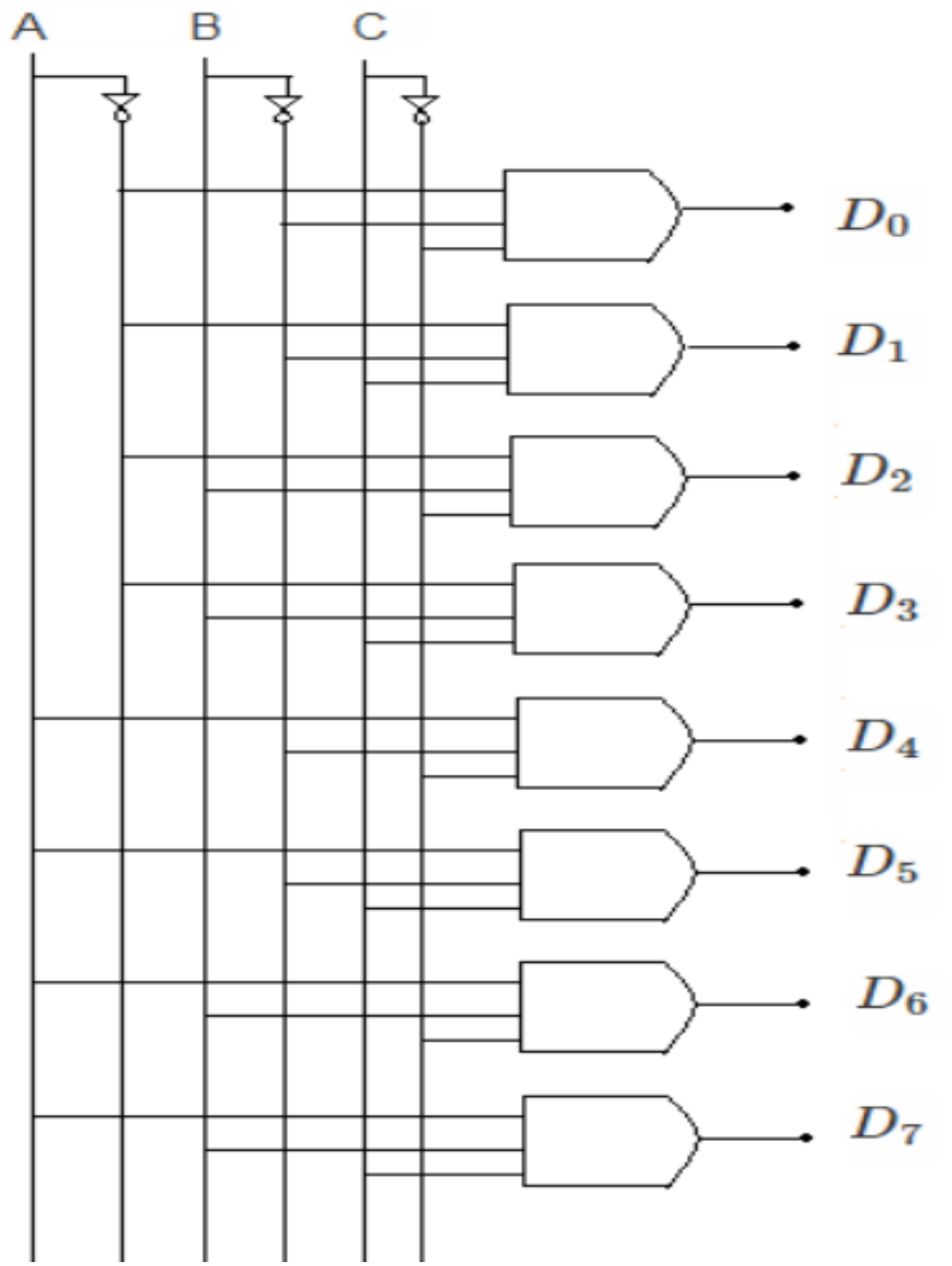
A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Expressions

$$D_0 = \bar{A}\bar{B}\bar{C}, \quad D_1 = \bar{A}\bar{B}C, \quad D_2 = \bar{A}B\bar{C},$$

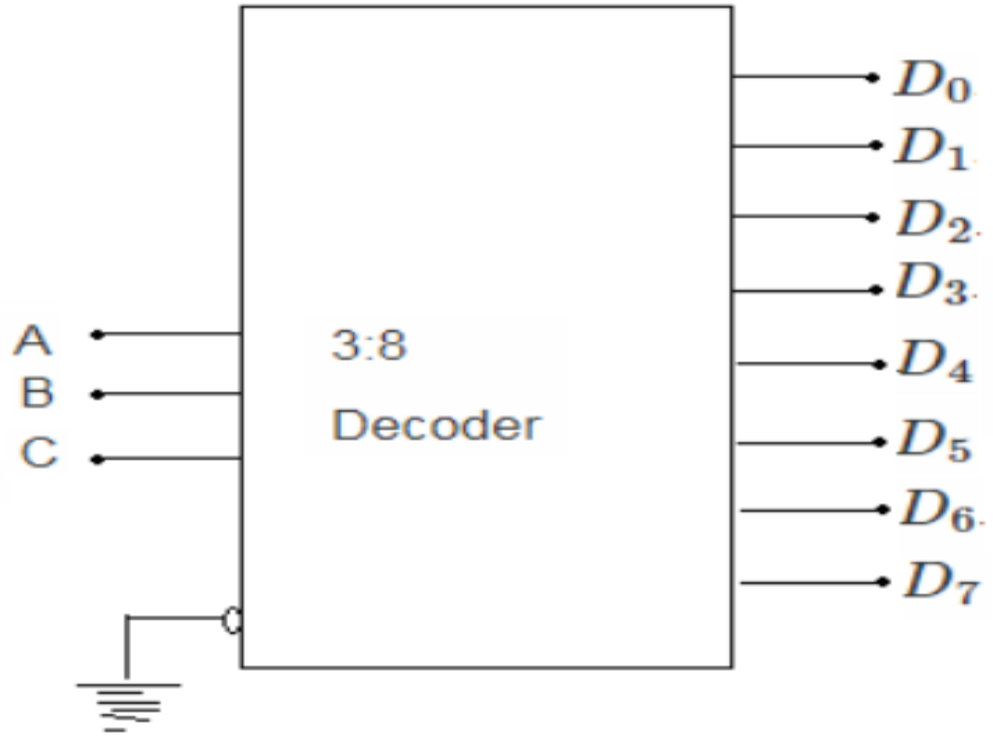
$$D_3 = \bar{A}BC, \quad D_4 = A\bar{B}\bar{C}, \quad D_5 = A\bar{B}C,$$

$$D_6 = AB\bar{C}, \quad D_7 = ABC$$



Implementation of 3 to 8 Decoder

- It is also called a binary-to-octal decoder, since the inputs represent 3-bit binary numbers and the outputs represent the eight digits in the octal number system.



Design of combinational circuits using decoders

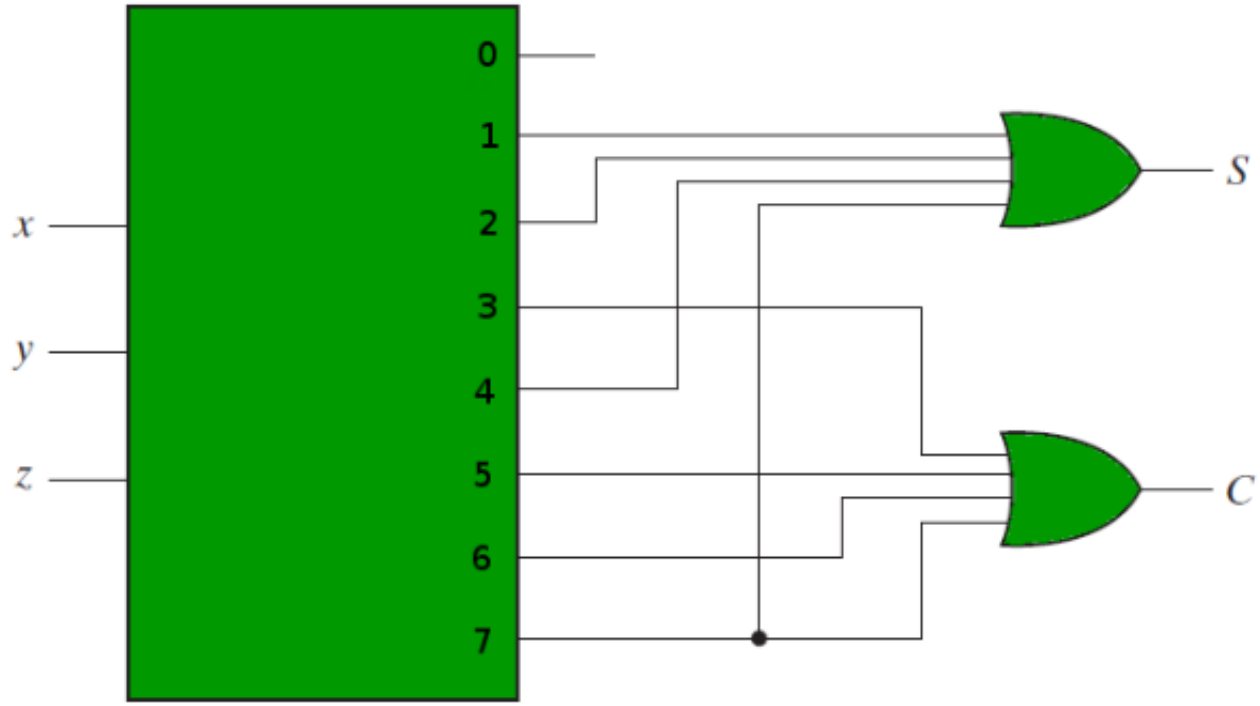
- Design Full Adder Circuit.
- What we know :
  - Decoder outputs Min terms.
  - Truth table of Full adder.
  - Min terms representing sum and carry part.

What we know

x	y	z	S	C
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

$$S = \sum(1, 2, 4, 7)$$
$$C = \sum(3, 5, 6, 7)$$

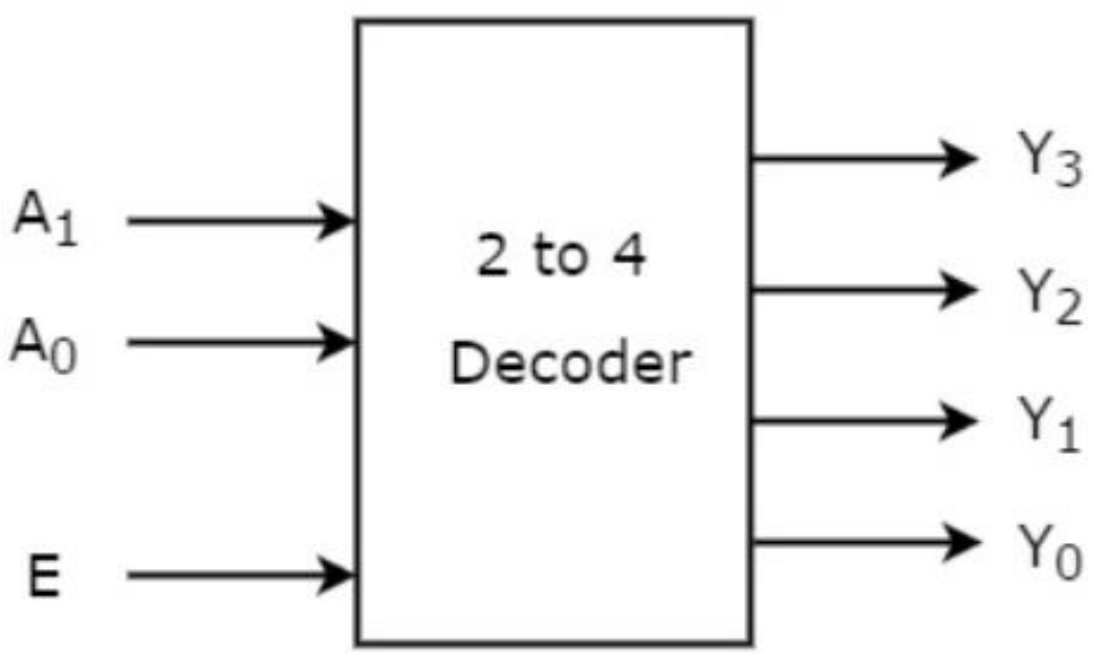
Full adder using 3 to 8 Decoder



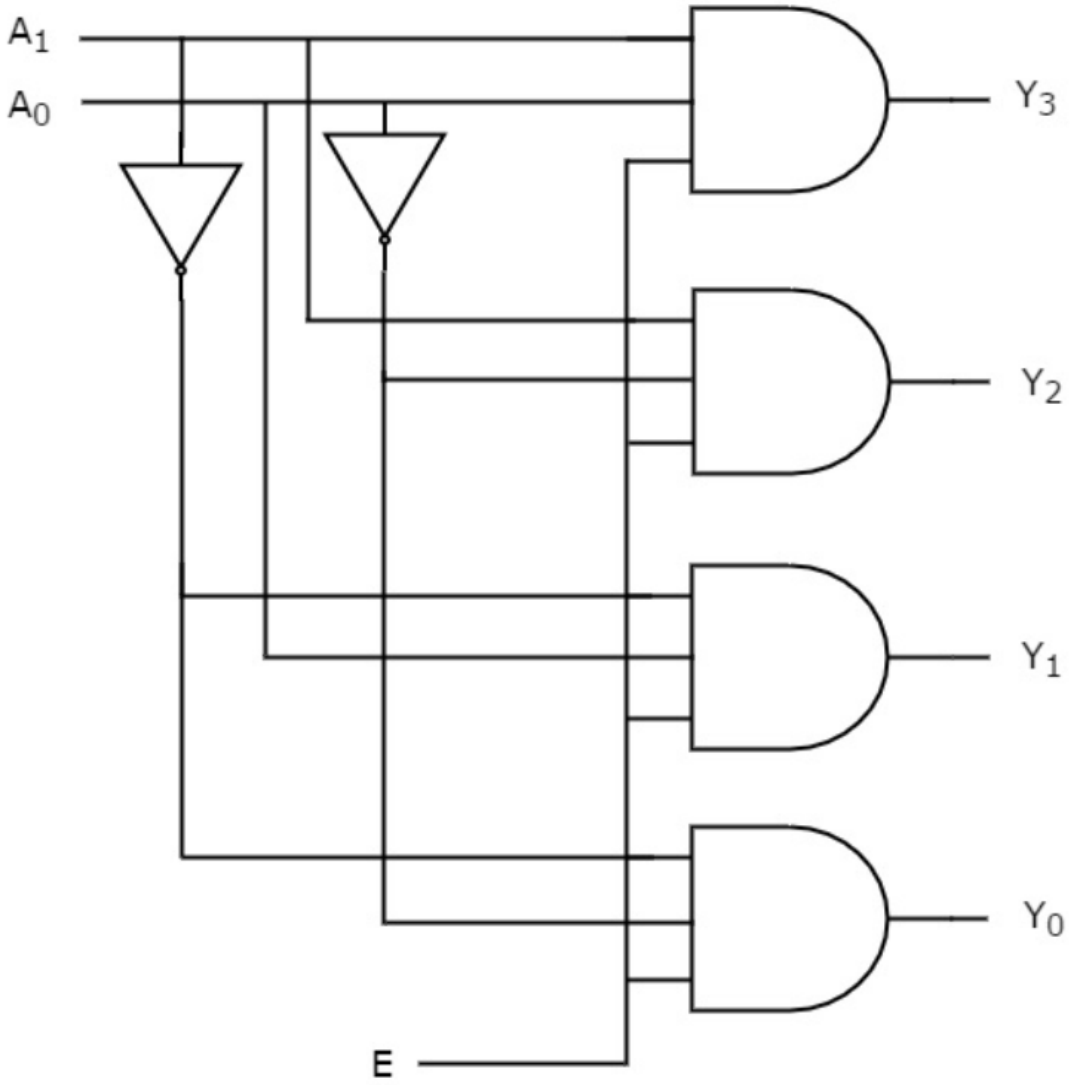
Decoder with Enable Input

- We can given one additional input Enable to the decoder.
- When enable input in high(1), then and then decoder should be functional.
- When enable is low(0), all outputs must be zero.
- Enable input is simply ANDed with other inputs.

Example 2 to 4 Decoder



Implementation Diagram



Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$$Y_3 = E.A_1.A_0$$

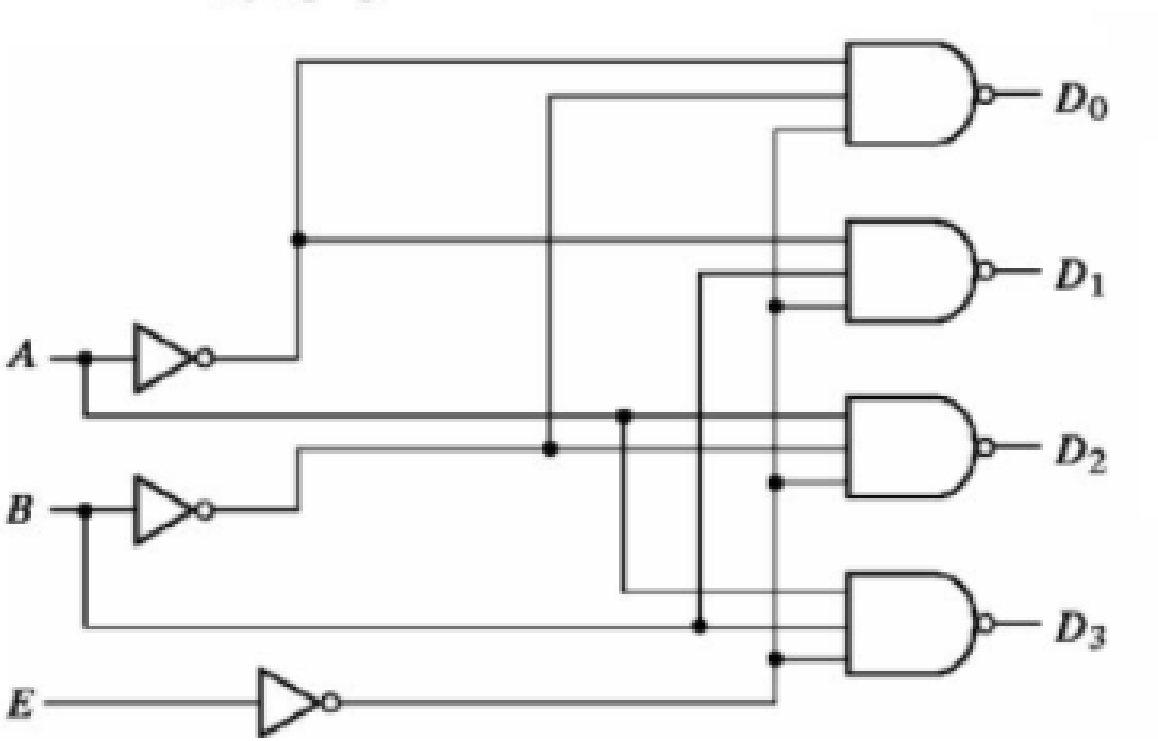
$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

Decoder with NAND Gate

- The special input “enable” controls the circuit behaviour.



Truth Table

- All outputs are equal to 1 if Enable signal is 1, regardless of values for A and B.
- When enable is 0, then the circuit behaves like decoder with complemented output.

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

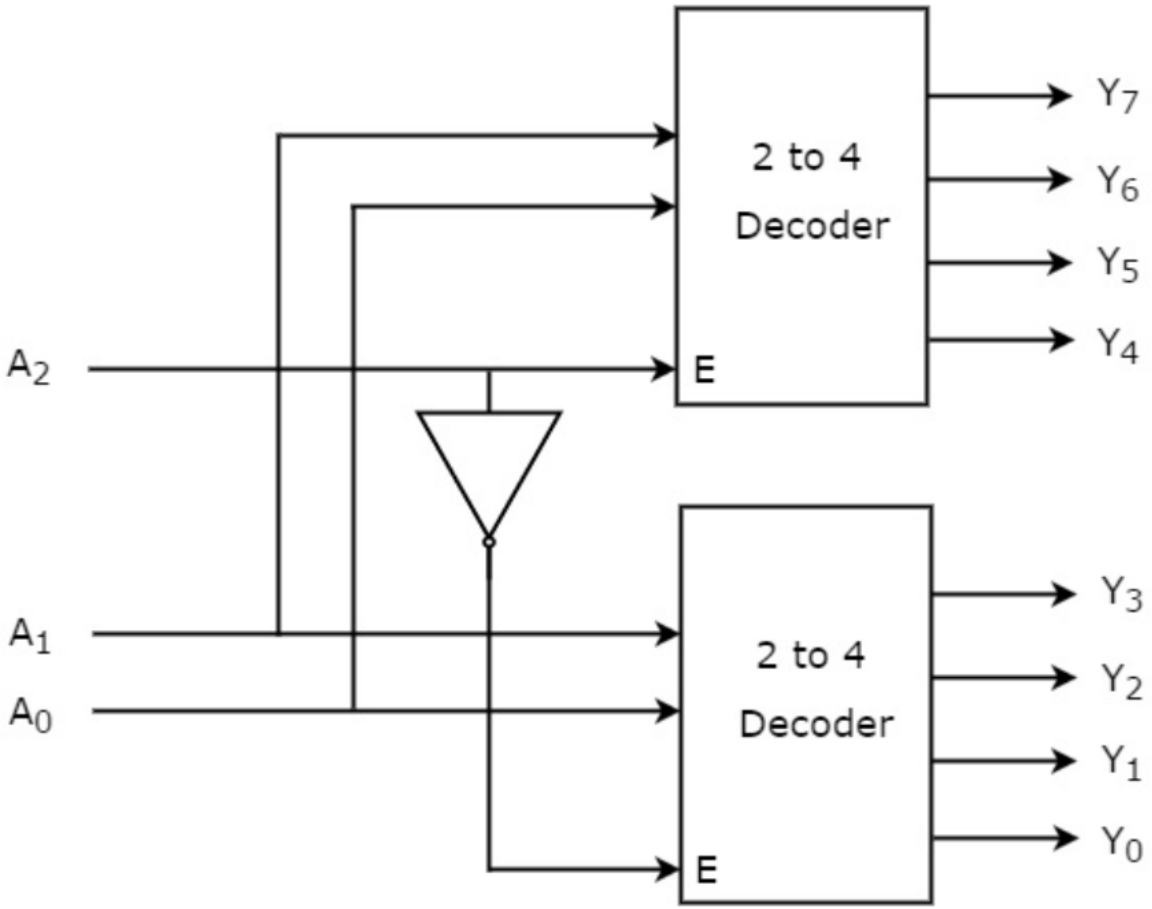
Demultiplexer

- Decoder with enable input is called demultiplexer.
- E is referred as Input and A and B are referred as select lines.
- Based on the bits specified by A and B, the input E is passed to one of the four output lines.

Implementation of Higher-order Decoders using lower order decoders

- We can find the number of lower order decoders required for implementing higher order decoder using the following formula :
  - Required number of lower order decoders = m2 /m1
- m1 is the number of outputs of lower order decoder and m2 is the number of outputs of higher order decoder.
- Implement **3 to 8 decoder using 2 to 4 decoders**.
  - So m1 = 4, m2 =8

3 to 8 Decoder using 2 to 4 Decoder



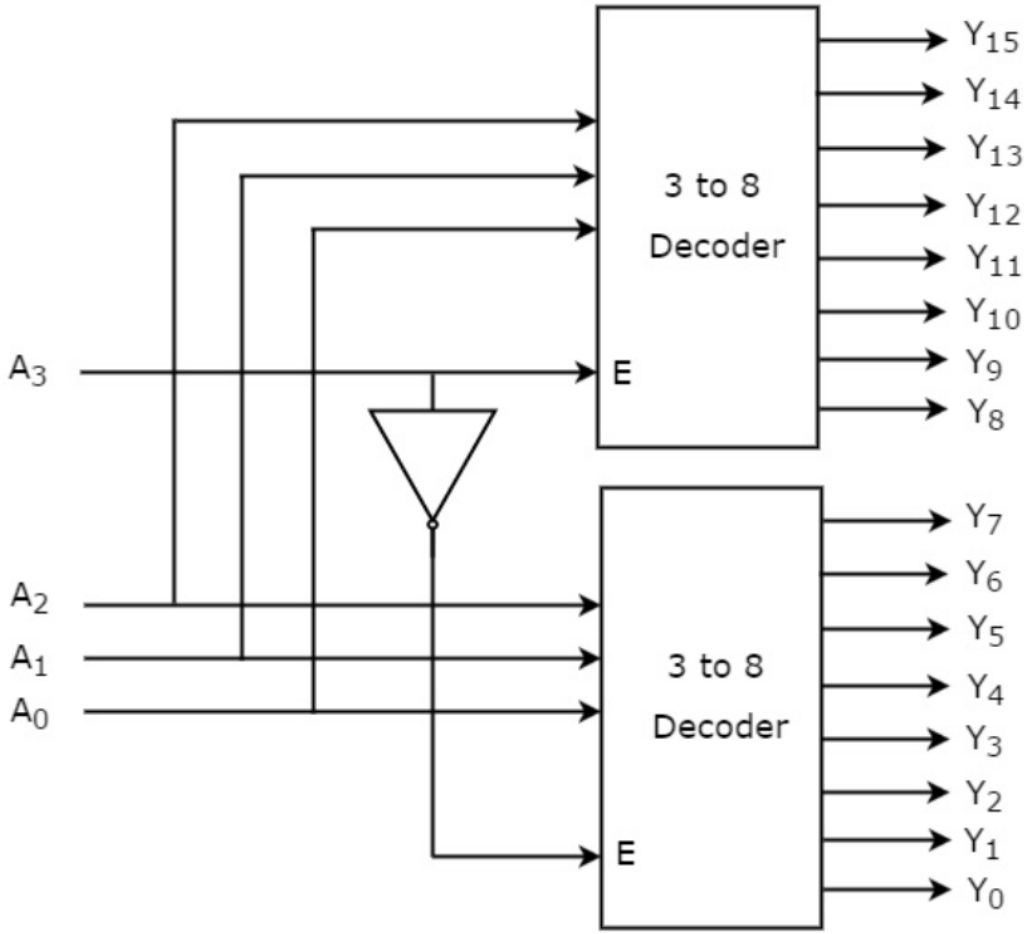
- The higher order bit A2 acts as enable pin for both decoders.
- When A2 is 0, then upper decoder will be disabled and lower decoder will generate min terms Do to D3.
- When A2 is 1, then lower decoder will be disabled and upper decoder will generate min terms D4 to D7.

Truth Table

A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	
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

- Construct 4 to 16 decoder using 3 to 8 decoder.
- How many 3 to 8 decoders will be used ?
- Draw block diagram.

Block Diagram

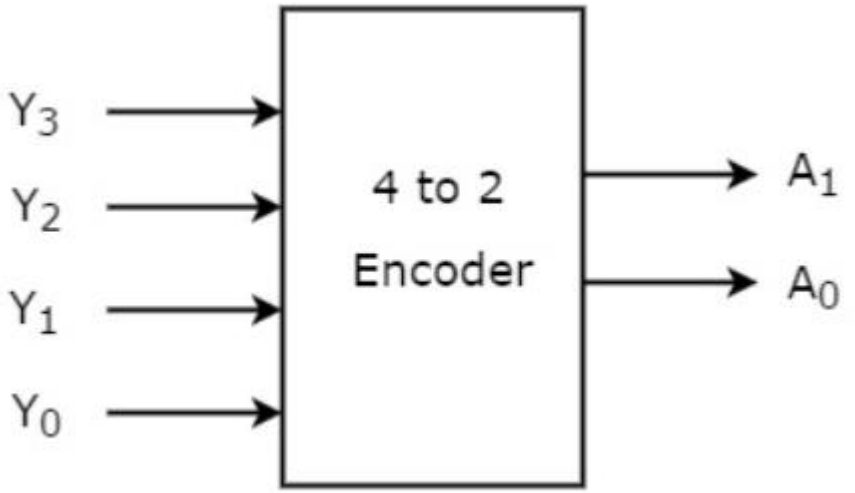


Encoders

- An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and ‘n’ output lines. It will produce a binary code equivalent to the input, which is active High.

4 to 2 Encoder

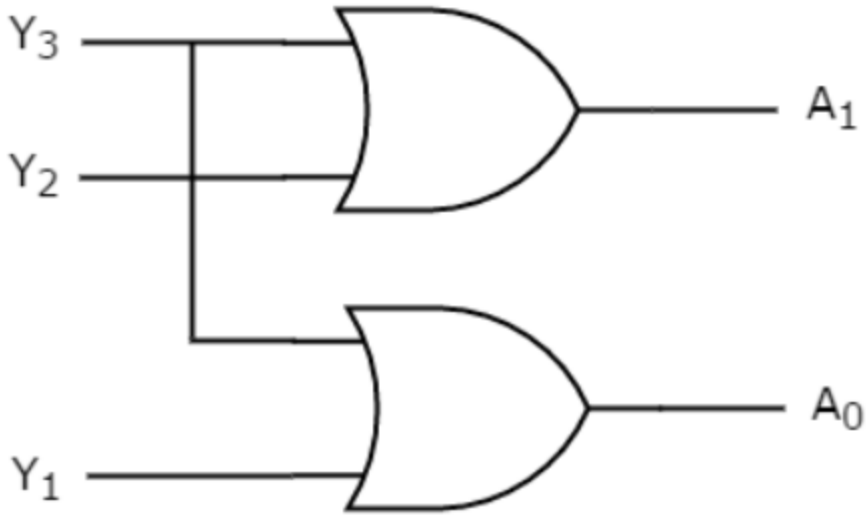
- Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0.



Truth Table

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

Implementation

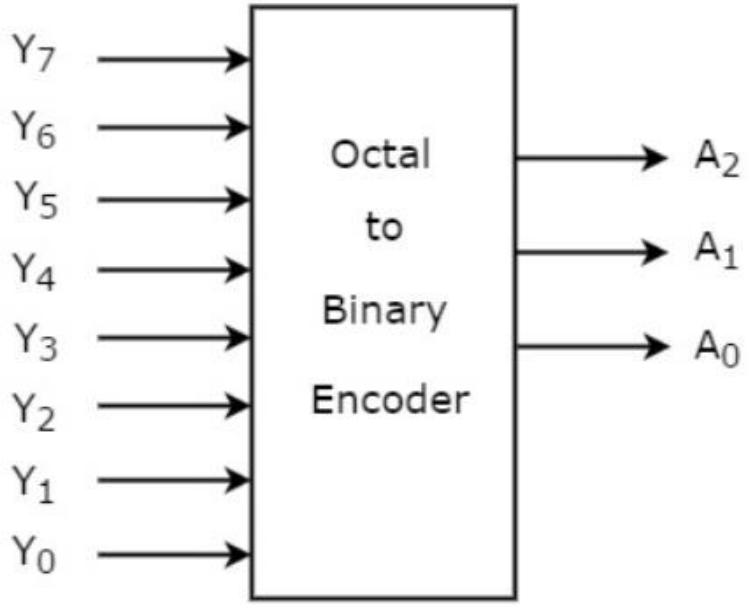


$$A_1 = Y_3 + Y_2$$

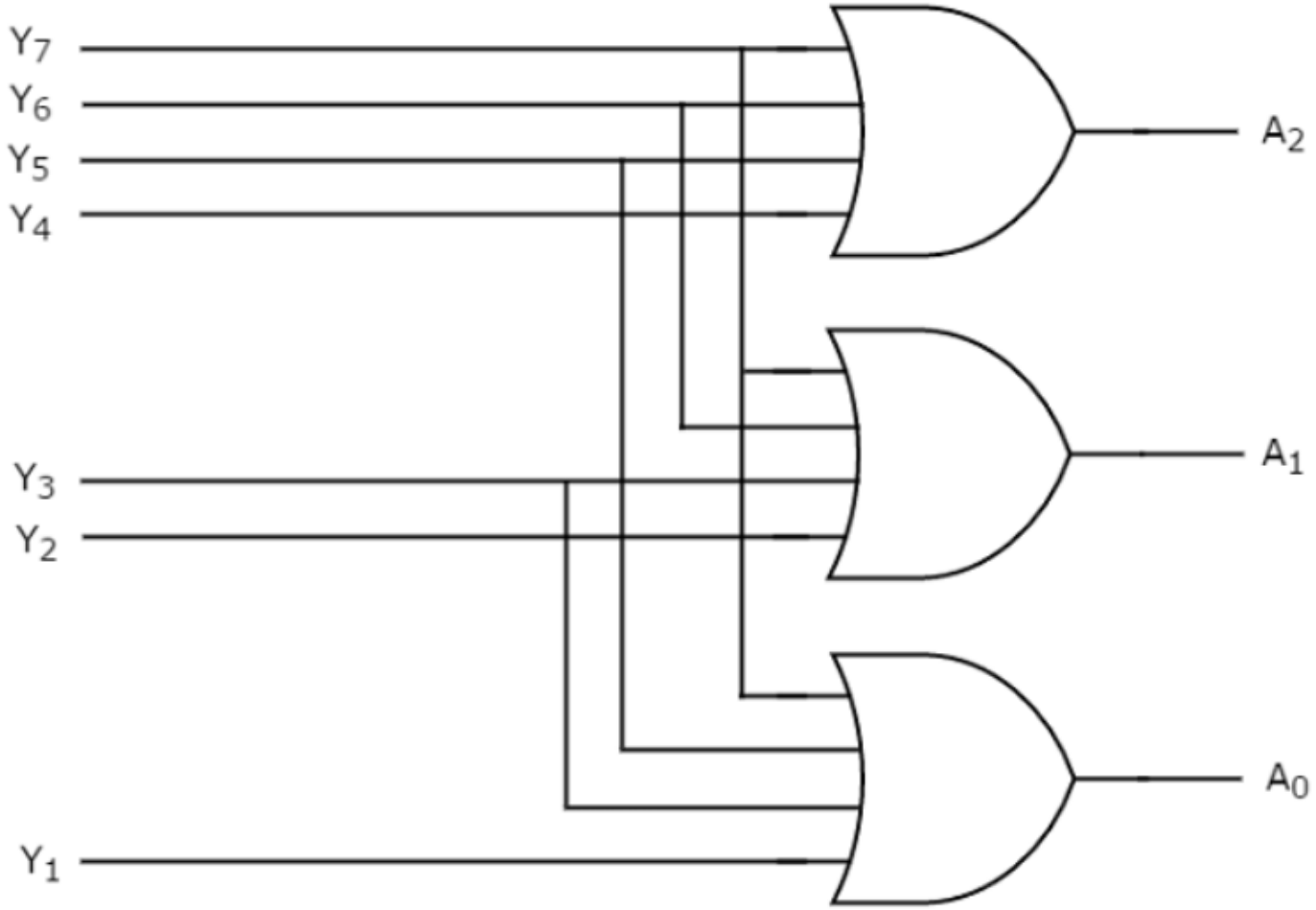
$$A_0 = Y_3 + Y_1$$

Octal to Binary Encoder

- 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.



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


Y7
Y6
Y5
Y4
Y3
Y2
Y1
A2
A1
A0

Drawbacks of Encoder

- 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 Y_3 and Y_6 are '1', then the encoder produces 110 at the output.

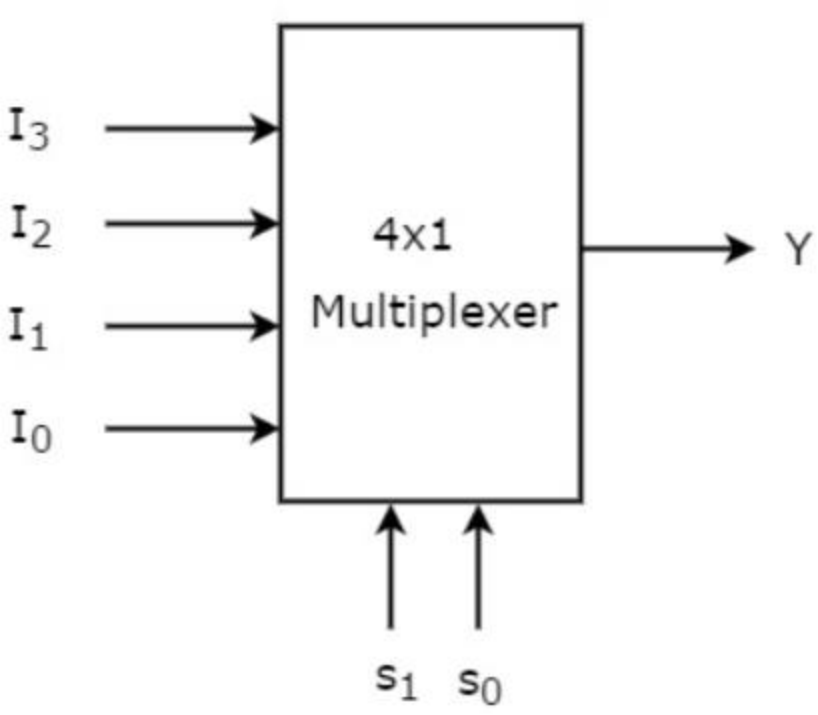
- To overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the binary code corresponding to the active High inputs, which has higher priority. This encoder is called as **priority encoder**.

Multiplexers

- **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.

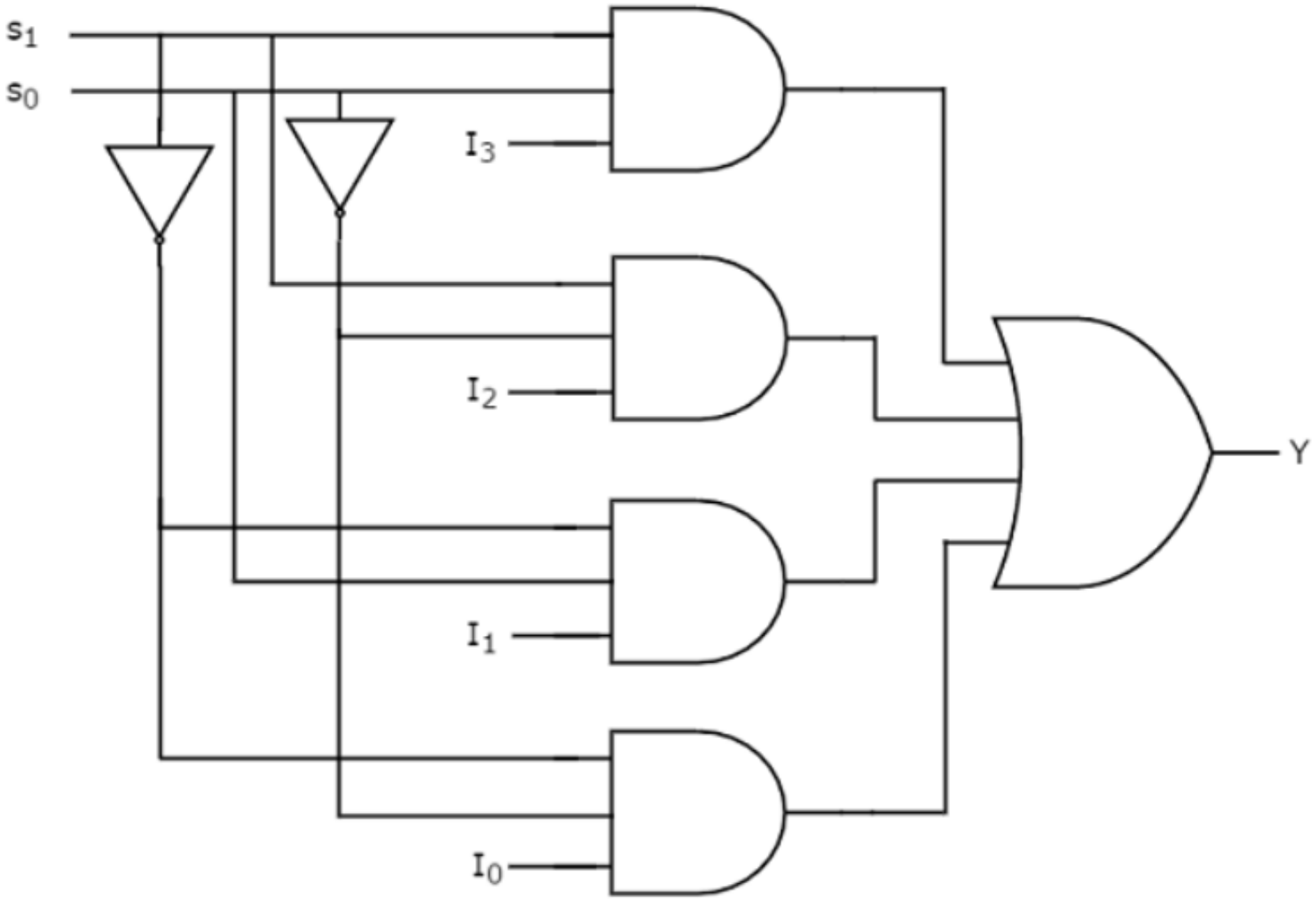
4x1 Multiplexer

- 4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y.



Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Y=S1’S0’I0+S1’S0I1+S1S0’I2+S1S0I3


s_1
s_0
I_3
I_2
I_1
I_0
Y

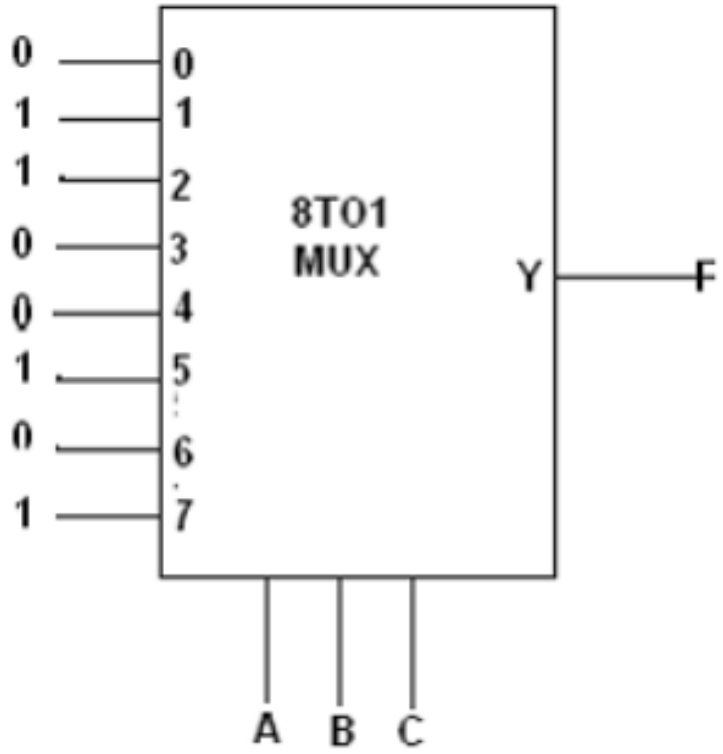
Quadrupal 2-to-1 line Mux

- It has 4 multiplexers.
- Each capable of selecting one of two input lines.
- Output Y1 can be selected to be equal to either A1 or B1.
- Output Y2 can be selected to be equal to either A2 or B2 and so on.

Implementation of Boolean Function using MUX

implement the function F(A, B, C)= Σ (1, 2, 5, 7) using 8 to 1 MUX

- We can implement it using all three variables at selection lines. We put 1 on the min term lines which are present in functions and 0 on the rest.


0
1
1
0
0
1
0
1
0
1
2
3
4
5
6
7
8TO1
MUX
Y
F
A
B
C

implement the function F(A, B, C)= Σ (1, 2, 5, 7) using 4 to 1 MUX

Implementing boolean function using Multiplexer

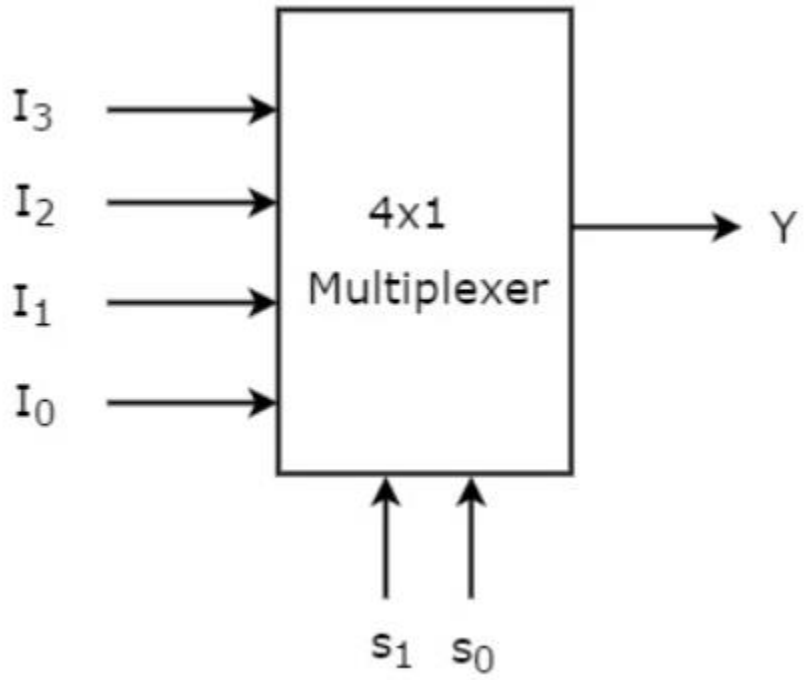
- While implementing any function using MUX, if we have N variables in the function then we take (N-1) variables on the selection lines and 1 variable is used for inputs of MUX.
- As we have N-1 variables on selection lines we need to have 2^{N-1} to 1 MUX.

- N=3 so we use $2^{N-1} = 2^{2} = 4$ to 1 MUX.
- Suppose we connect B,C to the selection lines and A to the input lines.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Implementation

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Alternative Method

- Take one variable for input lines and rest of the term for selection lines.
- Then list the min terms with the variable selected in complimented form in 1st row.
- List The min terms with variable selected in un-complimented form in 2nd row.
- Then encircle the min terms which are present in the function.
  - If we have no circled variable in the column, then we put 0 on the corresponding line
  - If we have both circled variables, then we put 1 on the line
  - If bottom variable is circled and top is not circled, apply A to input line
  - If bottom variable is not circled and top is circled, apply A’ to input line

- Take one variable for input lines and rest of the term for selection lines.
  - A Input Line and B, C Selection line.

- Then list the min terms with the variable selected in complimented form in 1st row.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

	I0	I1	I2	I3
A'	0	1	2	3
A				

- List The min terms with variable selected in un-complimented form in 2nd row.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

	I0	I1	I2	I3
A'	0	1	2	3
A	4	5	6	7

F(A, B, C)= Σ (1, 2, 5, 7)

	I0	I1	I2	I3
A'	0	1	2	3
A	4	5	6	7

- If we have no circled variable in the column, then we put 0 on the corresponding line
- If we have both circled variables, then we put 1 on the line
- If bottom variable is circled and top is not circled, apply A to input line
- If bottom variable is not circled and top is circled, apply A’ to input line

	I0	I1	I2	I3
A'	0	1	2	3
A	4	5	6	7

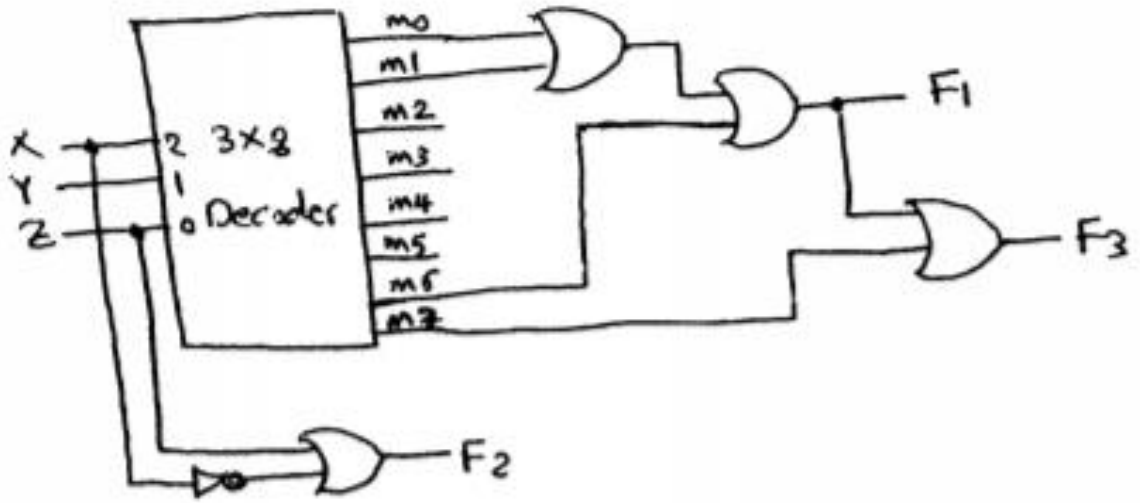
I0 = 0 , I1 = 1, I2 = A’, I3 = A

Exercise-1

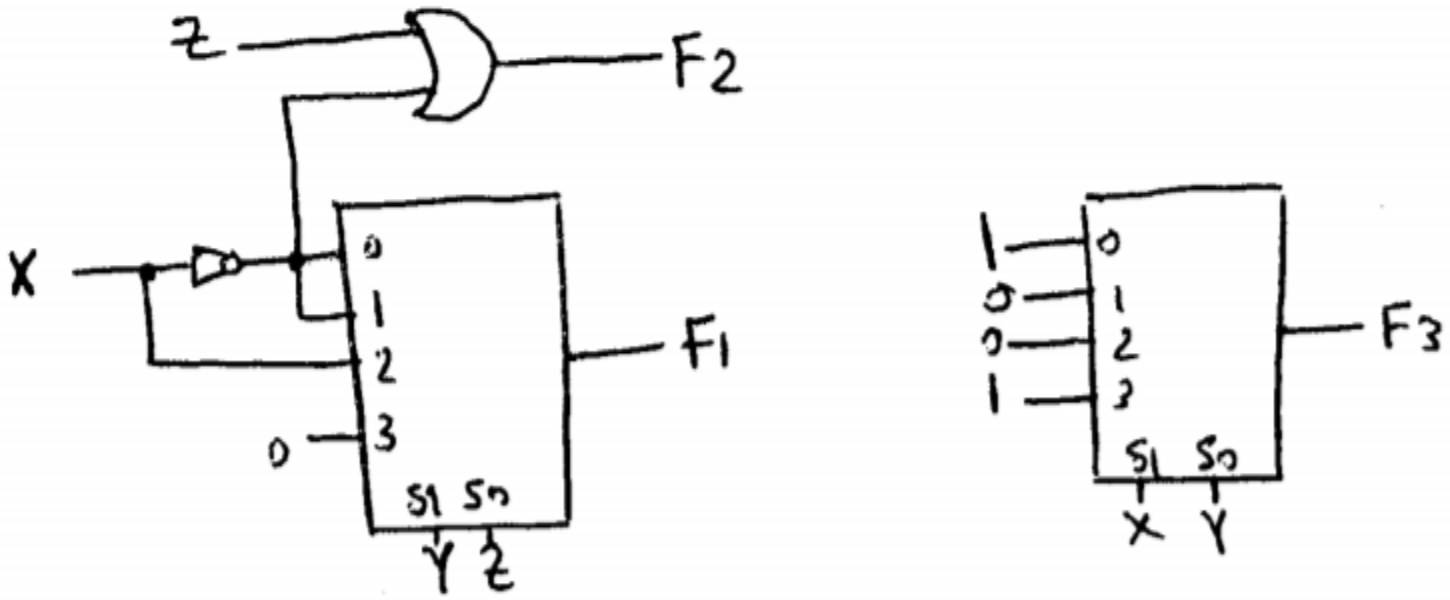
- A combinational circuit is defined by following 3 functions :
  - F1 = x'y' + xyz'
  - F2 = x' + y
  - F3 = xy + x'y'
    - Design above circuit with decoder and external gates.
    - Design with 4 to 1 MUX

Solution

(i) $F_1 = \sum m(0,1,6)$, $F_3 = \sum m(0,1,6,7)$



MUX Solution



Exercise-2

- Specify the truth table of an octal to binary priority encoder. Provide an output V to indicate that at least one of the inputs is present. The input with the highest subscript number has the highest priority. What will be the value of the four outputs if inputs D5 and D3 are 1 at the same time?

Solution

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z	V
0	0	0	0	0	0	0	0	X	X	X	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

The value of the four outputs if inputs D_5 and D_3 are 1 at the same time will be X=1, Y=0, Z=1, V=1.

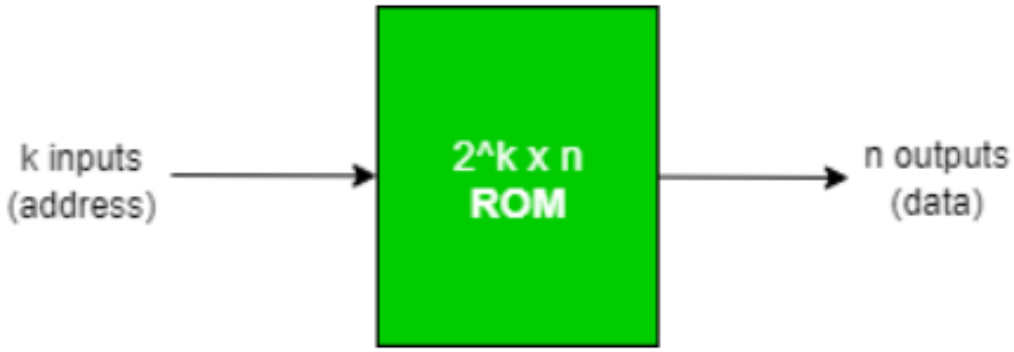
ROM(Read Only Memory)

- Read-Only Memory (ROM) is the primary memory unit of any computer system along with the Random Access Memory (RAM), but unlike RAM, in ROM, the binary information is stored permanently . Now, this information to be stored is provided by the designer and is then stored inside the ROM . Once, it is stored, it remains within the unit, even when power is turned off and on again .

- The information is embedded in the ROM, in the form of bits, by a process known as programming the ROM . Here, programming is used to refer to the hardware procedure which specifies the bits that are going to be inserted in the hardware configuration of the device . And this is what makes ROM a Programmable Logic Device (PLD) .

- A Programmable Logic Device (PLD) is an IC (Integrated Circuit) with internal logic gates connected through electronic paths that behave similar to fuses . In the original state, all the fuses are intact, but when we program these devices, we blow away certain fuses along the paths that must be removed to achieve a particular configuration. And this is what happens in ROM, ROM consists of nothing but basic logic gates arranged in such a way that they store the specified bits.

Structure of ROM

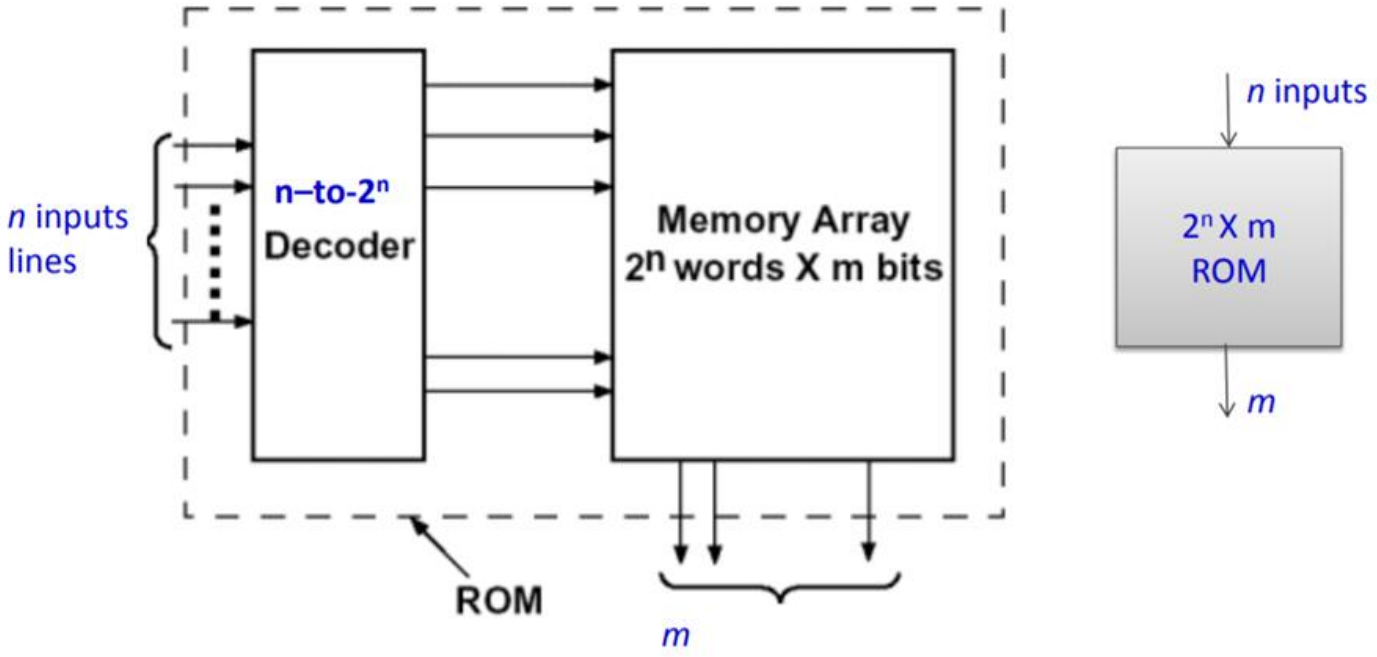


ROM Structure

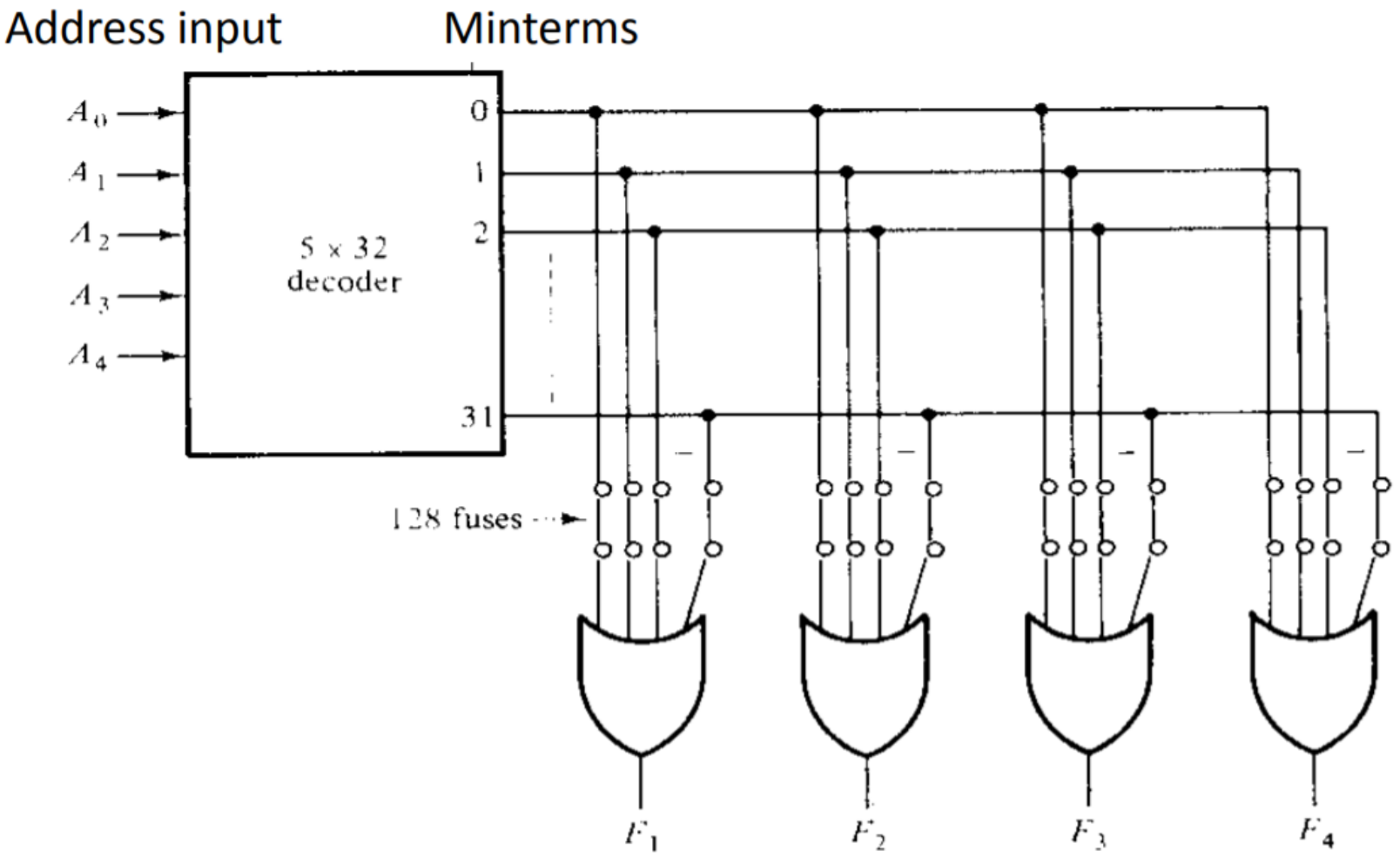
- It consists of k input lines and n output lines .
- The k input lines is used to take the input address from where we want to access the content of the ROM .
- Since each of the k input lines can be either 0 or 1, so there are 2 total addresses which can be referred to by these input lines and each of these addresses contain n bit information, which is given out as the output of the ROM.
- Such a ROM is specified as 2 x n ROM .

- It consists of two basic components – Decoder and OR gates .
- Decoder outputs min terms.
- Each Min term is fused to output level OR gates.
- Number of output gates determine output lines of ROM.

$2^n \times m$ ROM



Logical construction of a 32 X 4 ROM



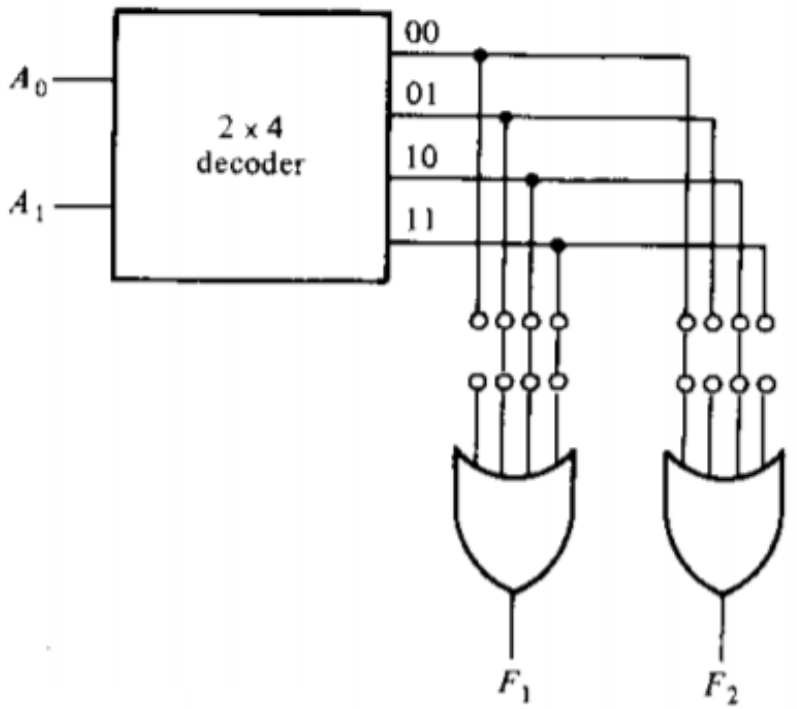
Combinational-circuit implementation with a 4 X 2 ROM

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

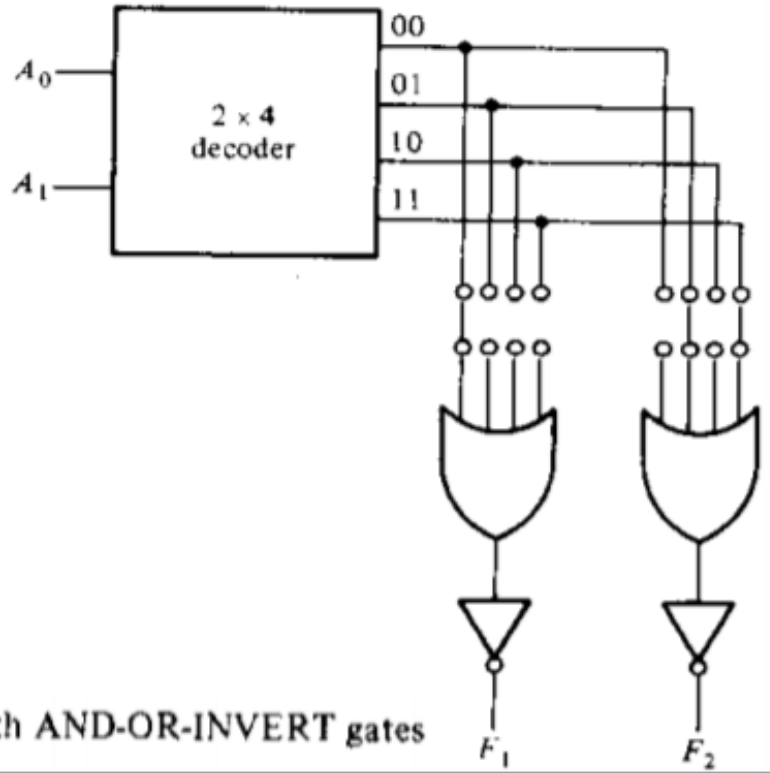
(a) Truth table

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$

$$F_2(A_1, A_0) = \Sigma(0, 2)$$

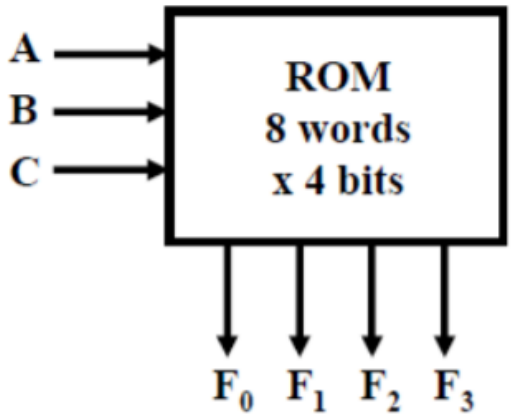


(b) ROM with AND-OR gates



(c) ROM with AND-OR-INVERT gates

ROM Realization of Logic Functions



a. Block diagram

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

b. Truth table for ROM

$F_0 = A'B' + AC'$

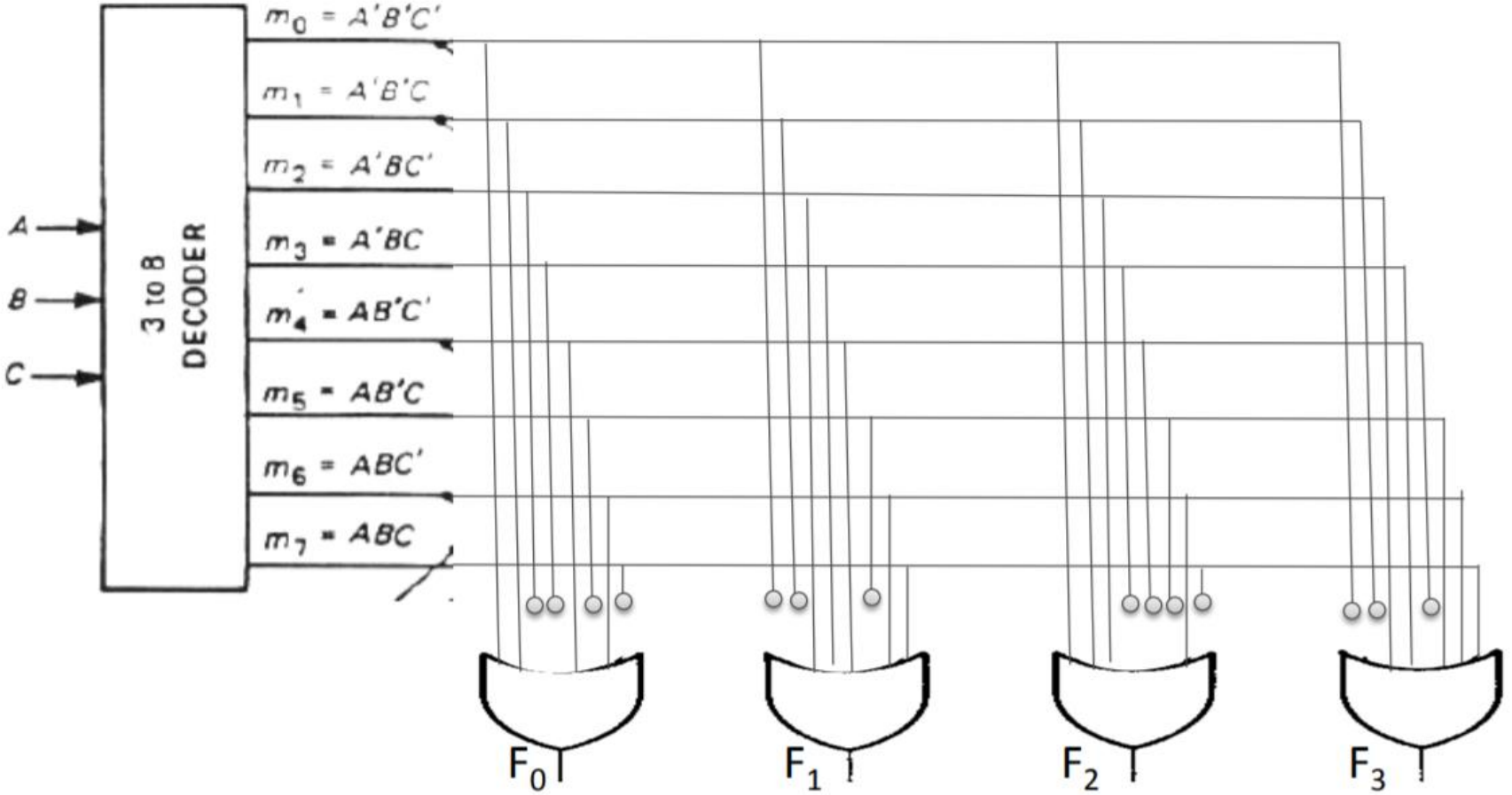
$F_1 = B + AC'$

$F_2 = A'B' + BC'$

$F_3 = AC + B$

ROM Realization of Logic Functions

- ROM consists of a decoder and a memory array.
- When a particular input sequence is applied to the n decoder inputs, exactly one of the 2^n outputs is set to 1.
- This output line selects one of the words in the memory array.

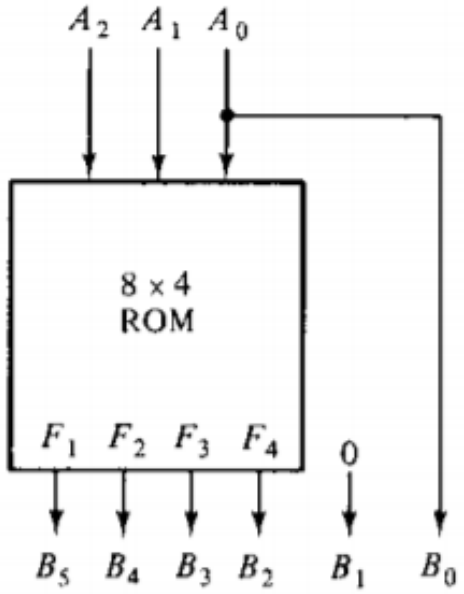


Exercise: Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

- First step is to derive the truth table for the combinational circuit
  - In some case, we can fit a small truth table for ROM by using certain properties in the truth table

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Block diagram



ROM truth table

A_2	A_1	A_0	F_1	F_2	F_3	F_4
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

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Types of ROM

- Mask programming
- Programmable read only memory
- Erasable PROM

Mask programming

- Created by manufacturer during the fabrication process
- Customer provides required truth table in specified format
- The manufacturer makes the mask for the paths to produce one and zeros
- It is costly process
- Suitable for large production

PROGRAMMABLE READ ONLY MEMORY

- Initially it contains either all 0's or all 1's
- The links are broken by applying current pulses
- User can program in his own way
- Programming process is hardware based.
- The programming is irreversible
- The programming done cannot be changed

EPROM

- They can be structured to their initial values
- This process is done by the application of ultraviolet light
- This type of ROM can be reprogrammed by the user.
- If electrical signals are used instead of ultraviolet lights then the ROM is called electrically alterable ROM

APPLICATIONS OF ROM

- Implementation of any combination circuit
  - Each output terminal is considered as the output of Boolean function expresses in sum of mean terms
- Storage unit
  - Stores fixed pattern of bits
  - Each combination of bits represent one word (address)
  - Data stored at specific address is given as a output

APPLICATIONS OF ROM

- Conversion of one binary code to other
- Arithmetic operations such as multiplier
- Design of control units of digital systems
  - A control unit that utilizes a ROM to store binary control information is called micro programmed control unit

ROM Types

- **Mask-programmable (ROM)**
  - Permanent programming done at fabrication time
  - Fabrication take place at factory as per customer order
  - Very expensive and therefore feasible only for large quantity orders
- **Programmable ROM (PROM)**
  - User programmed after purchase, called field-programmable ROM (FPROM)
  - Reprogrammable by user, erasable by UV emission, called erasable, programmable ROM (EPROM)
- **Electrically erasable, programmable ROM (EEPROM)**
  - User can erase individual words; switching elements can be enabled/disabled
  - Can be erased and reprogrammed limited number of times, typically 100 to 1000 times
- **Flash memory**
  - Like EEPROM, but all words (or large blocks of words) can be erased simultaneously
  - Become common relatively recently (late 1990s)

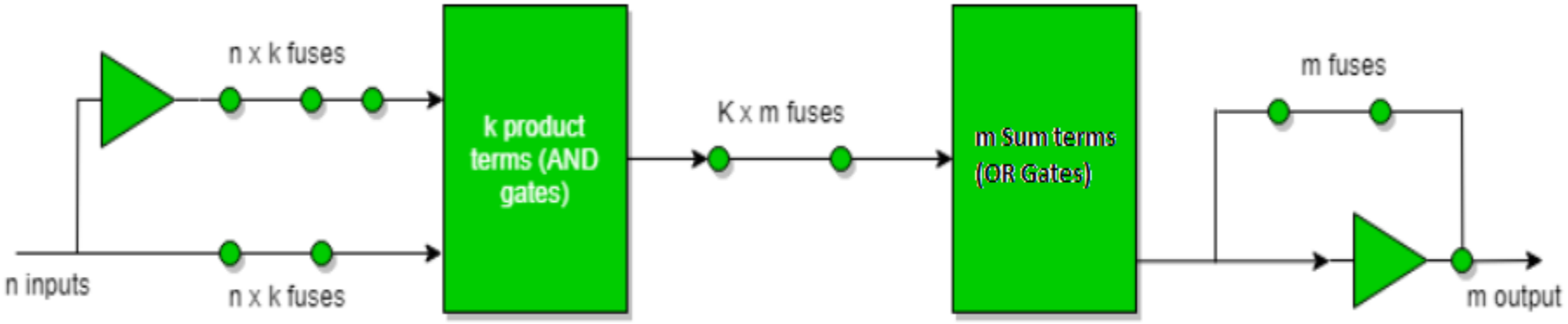
Programmable Logic Array

- Programmable Logic Array(PLA) is a fixed architecture logic device with programmable AND gates followed by programmable OR gates.
- PLA is basically a type of programmable logic device used to build reconfigurable digital circuit. PLDs have undefined function at the time of manufacturing but they are programmed before made into use. PLA is a combination of memory and logic.

PLA

- **Comparison with other Programmable Logic Devices:**
  - PLA has programmable AND gate array and programmable OR gate array.
  - PAL has programmable AND gate array but fixed OR gate array.
  - ROM has fixed AND gate array but programmable OR gate array.
- PLA is similar to a ROM in concept; however it does not provide full decoding of variables and does not generate all minterms as in the ROM.

Basic block diagram for PLA



Example

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

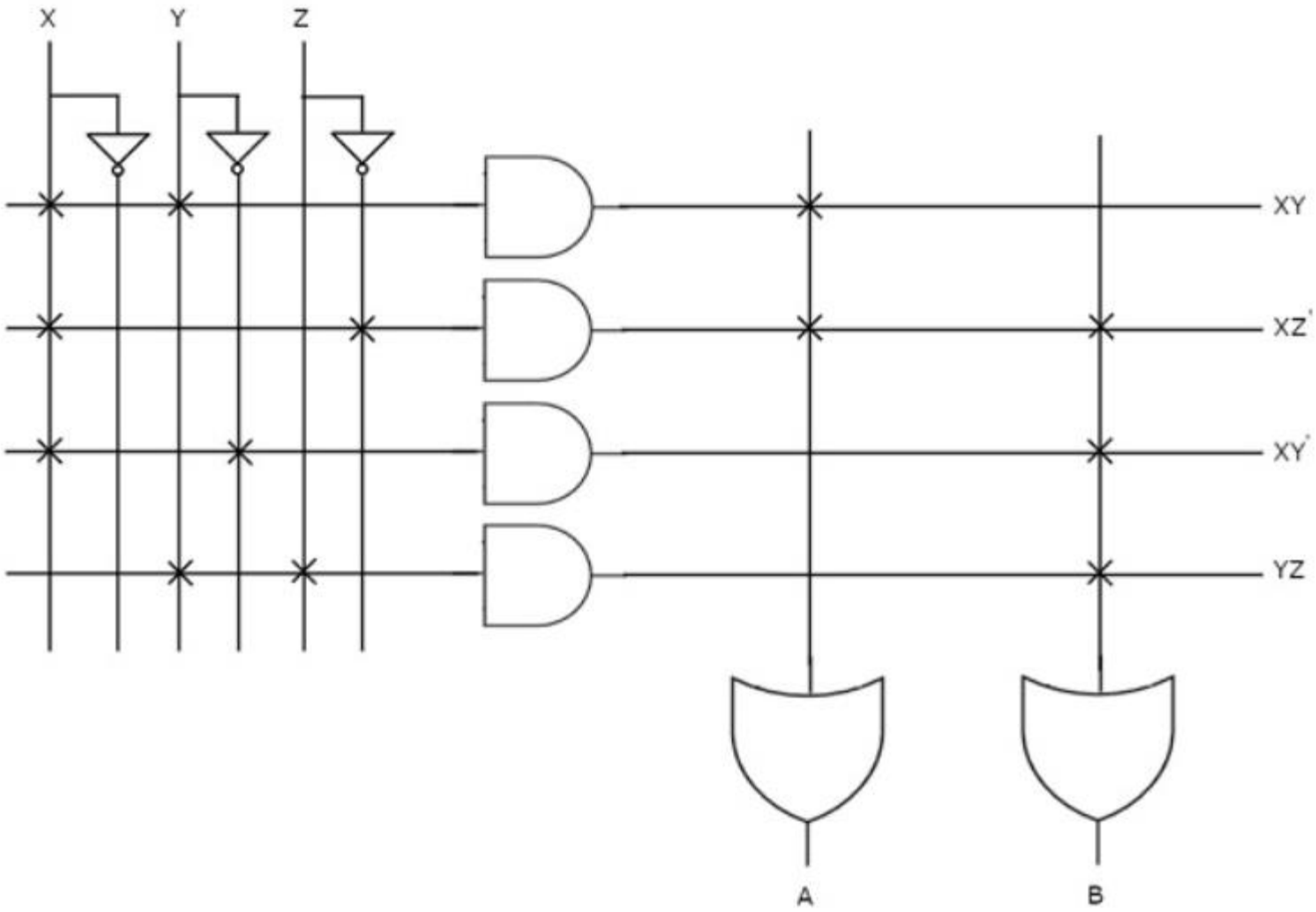
$$A = XY + XZ'$$

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

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, $Z'X$ is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding **PLA** is shown in the following figure.

Logic Diagram



- The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′, Y, Y′, Z & Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.
- All these product terms are available at the inputs of each **programmable OR gate**. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol ‘X’ is used for programmable connections.

Example -2

- F1 = AB’ + AC
- F2 = AC + BC

Logic Diagram

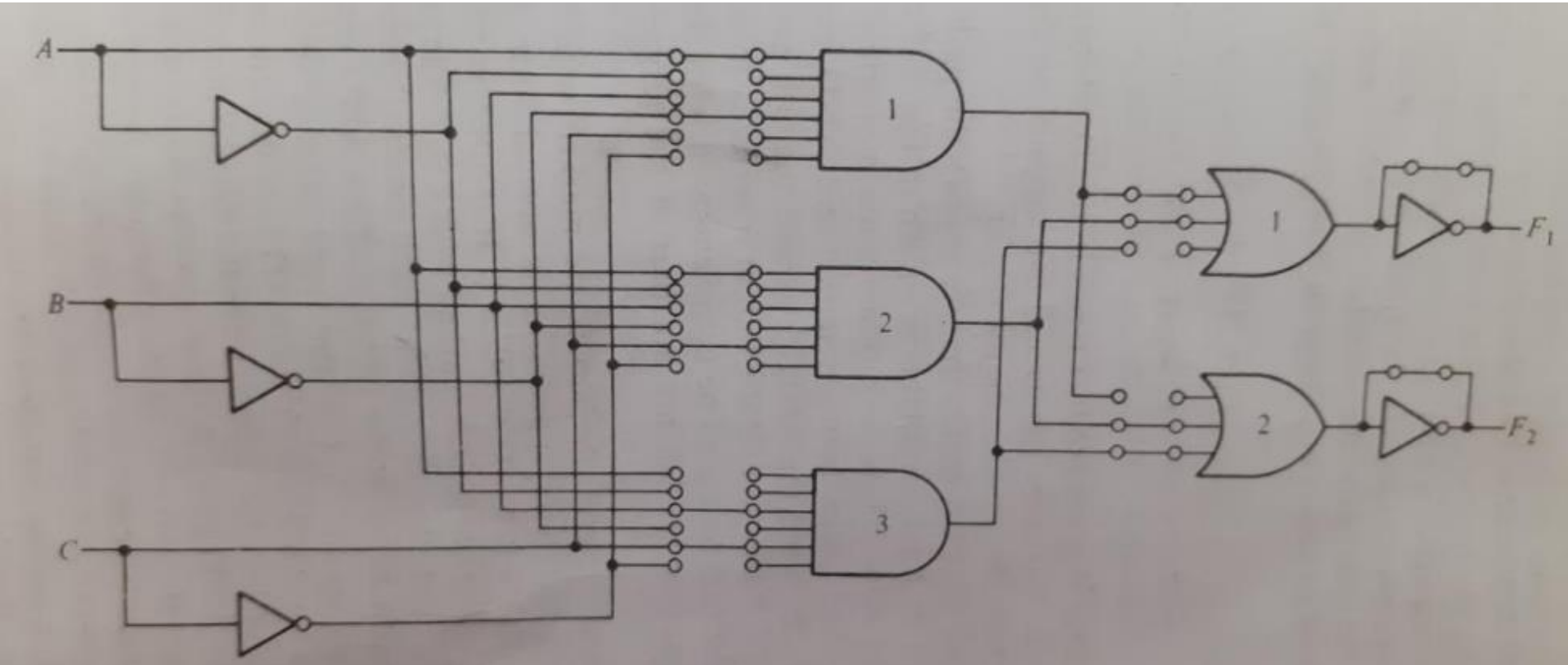


Figure 5-26 PLA with 3 inputs. 3 product terms, and 2 outputs; it implements the combinational circuit specified in Fig. 5-27

Truth Table

	Product term	Inputs A	B	C	Outputs F_1	F_2	
AB'	1	1	0	–	1	–	
AC	2	1	–	1	1	1	
BC	3	–	1	1	–	1	
					T	T	T/C

(c) PLA program table.

A	B	C	F1	F2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

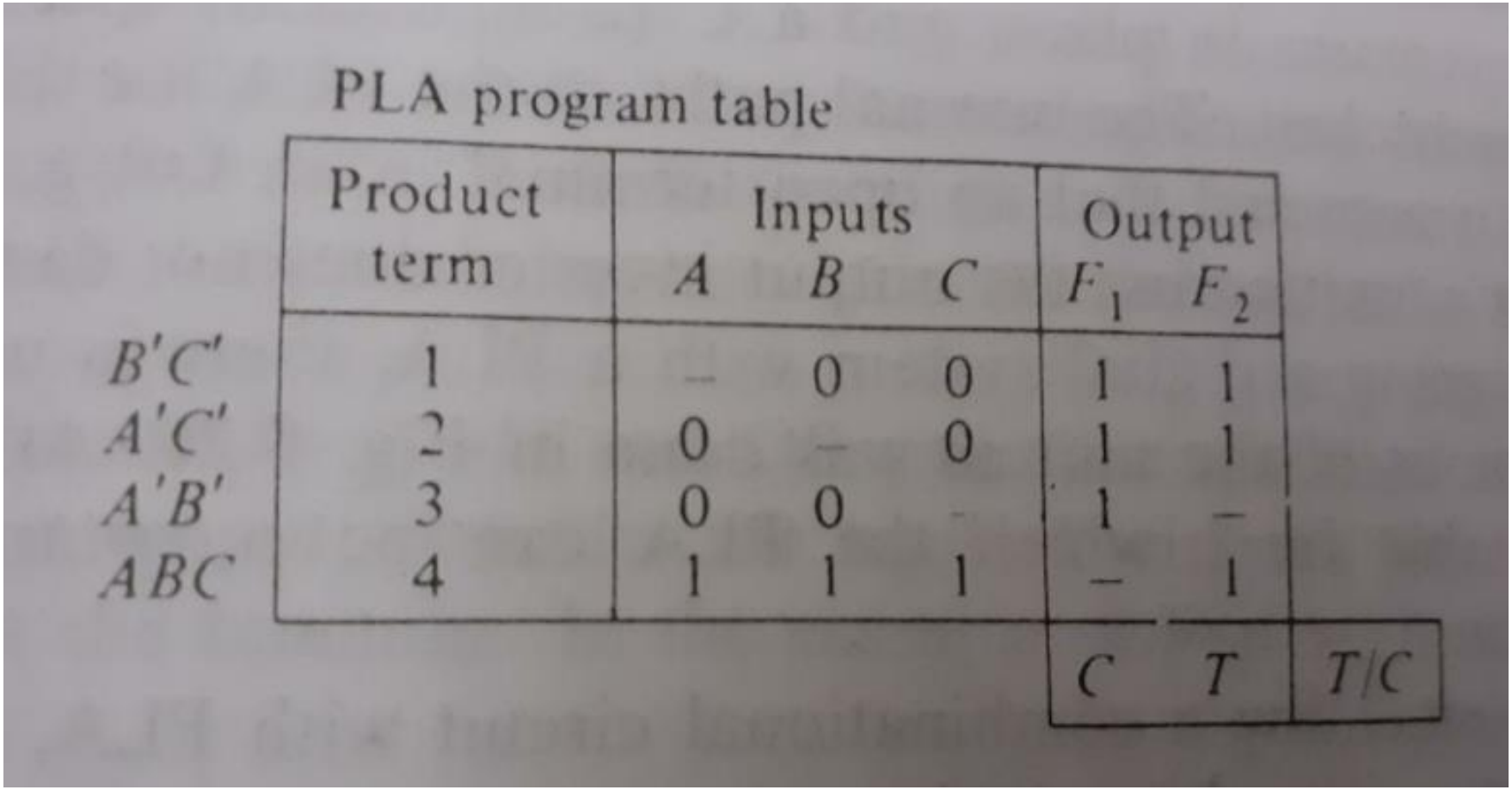
Example

- A combinational circuit is defined by the function :
  - F1(A,B,C) = minterms(3,5,6,7)
  - F2(A,B,C) = minterms(0,2,4,7)
- Implement the circuit with a PLA having 3 Inputs, Four product terms, and two outputs.

Example

- F1 = AC + AB + BC
- F2 = B’C’ + A’C’ + ABC
- F1’ = B’C’ + A’C’ + A’B’
- F2’ = B’C + A’C + ABC’

Truth Table



PLA program table

	Product term	Inputs A	Inputs B	Inputs C	Output F_1	Output F_2
$B'C'$	1	–	0	0	1	1
$A'C'$	2	0	–	0	1	1
$A'B'$	3	0	0	–	1	–
ABC	4	1	1	1	–	1
					C	T

T/C