Chapter-5

Combinational Logic with MSI and LSI

Introduction

There are several combinational circuits that are employed extensively in the design of digital systems. These circuits are available in integrated circuits and are classified as MSI components. MSI components perform specific digital functions commonly needed in the design of digital systems.

Combinational circuit-type MSI components that are readily available in IC packages are binary adders, subtractors, comparators, decoders, encoders, and multiplexers. These components are also used as standard modules within more complex LSI and VLSI circuits and hence used extensively as basic building blocks in the design of digital computers and systems

Binary Adder

This circuit sums up two binary numbers A and B of n-bits using full-adders to add each bit-pair & carry from previous bit position. The sum of A and B can be generated in two ways: either in a serial fashion or in parallel.

The serial addition method uses only one full-adder circuit and a storage device to hold the generated output carry. The pair of bits in A and B are transferred serially, one at a time, through the single full-adder to produce a string of output bits for the sum. The stored output carry from one pair of bits is used as an input carry for the next pair of bits.

The parallel method uses n full-adder circuits, and all bits of A and B are applied simultaneously. The outputs carry from one full-adder is connected to the input carry of the full-adder one position to its left. As soon as the carries are generated, the correct sum bits emerge from the sum outputs of all full adders

Binary Parallel adder

A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers in parallel. It consists of full-adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

Diagram below shows the interconnection of four full-adder (FA) circuits to provide a 4-bit binary parallel adder. The augend bits of A and the addend bits of B are designated by subscript numbers from right to left. The carries are connected in a chain through the full-adders. The S outputs generate the required sum bits. The input carry to the adder is C1 and the output carry is C5.

When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augend bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries.

Binary Parallel adder

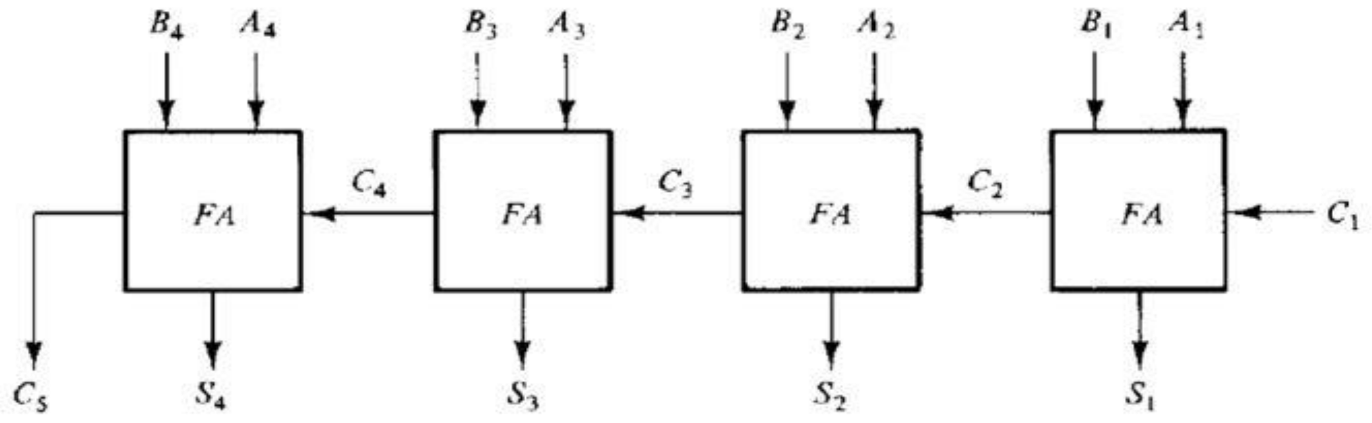


Fig: 4-bit parallel binary adder

The 4-bit binary-adder is a typical example of an MSI function. It can be used in many applications involving arithmetic operations. Observe that the design of this circuit by the classical method would require a truth table with $2^9 = 512$ entries, since there are 9 inputs to the circuit. By using an iterative method of cascading an already known function, we were able to obtain a simple and well-organized implementation.

Binary Parallel adder

The carry propagation time is a limiting factor on the speed with which two numbers are added in parallel. Although a parallel adder, or any combinational circuit, will always have some value at its output terminals, the outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs. Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is very critical. An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays. But physical circuits have a limit to their capability. Another solution is to increase the equipment complexity in such a way that the carry delay time is reduced.

There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of look-ahead carry.

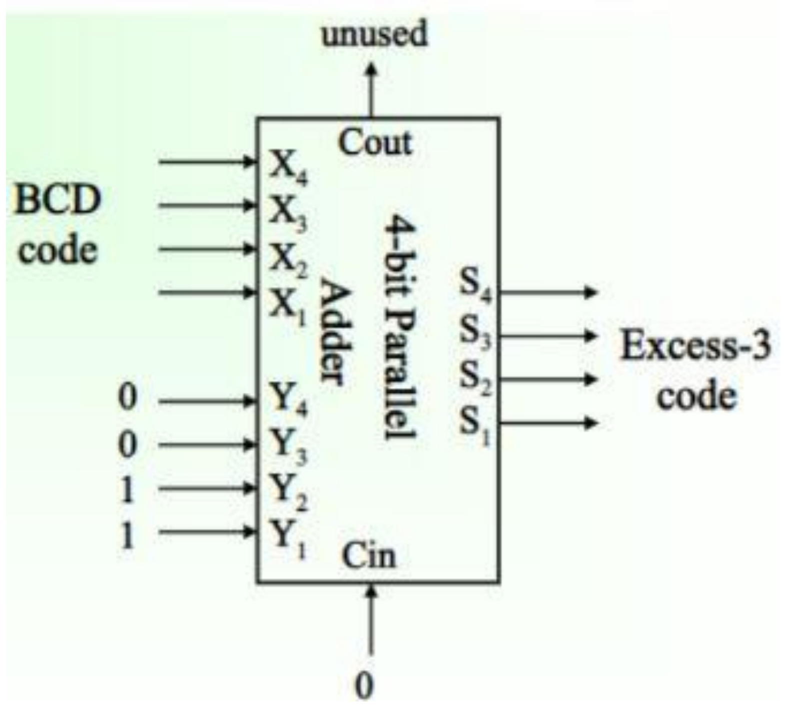
Q. **Design a BCD-to-excess-3 code converter using a 4-bit full adders MSI circuit.**

Excess 3 code = BCD code + $(0011)_2$

Augend bits = X_4 X_3 X_2 X_1 (Input bits)

Addend bits = Y_4 Y_3 Y_2 Y_1 = 0011

Excess-3 code = S_4 S_3 S_2 S_1 (output)



Decimal/BCD Adder

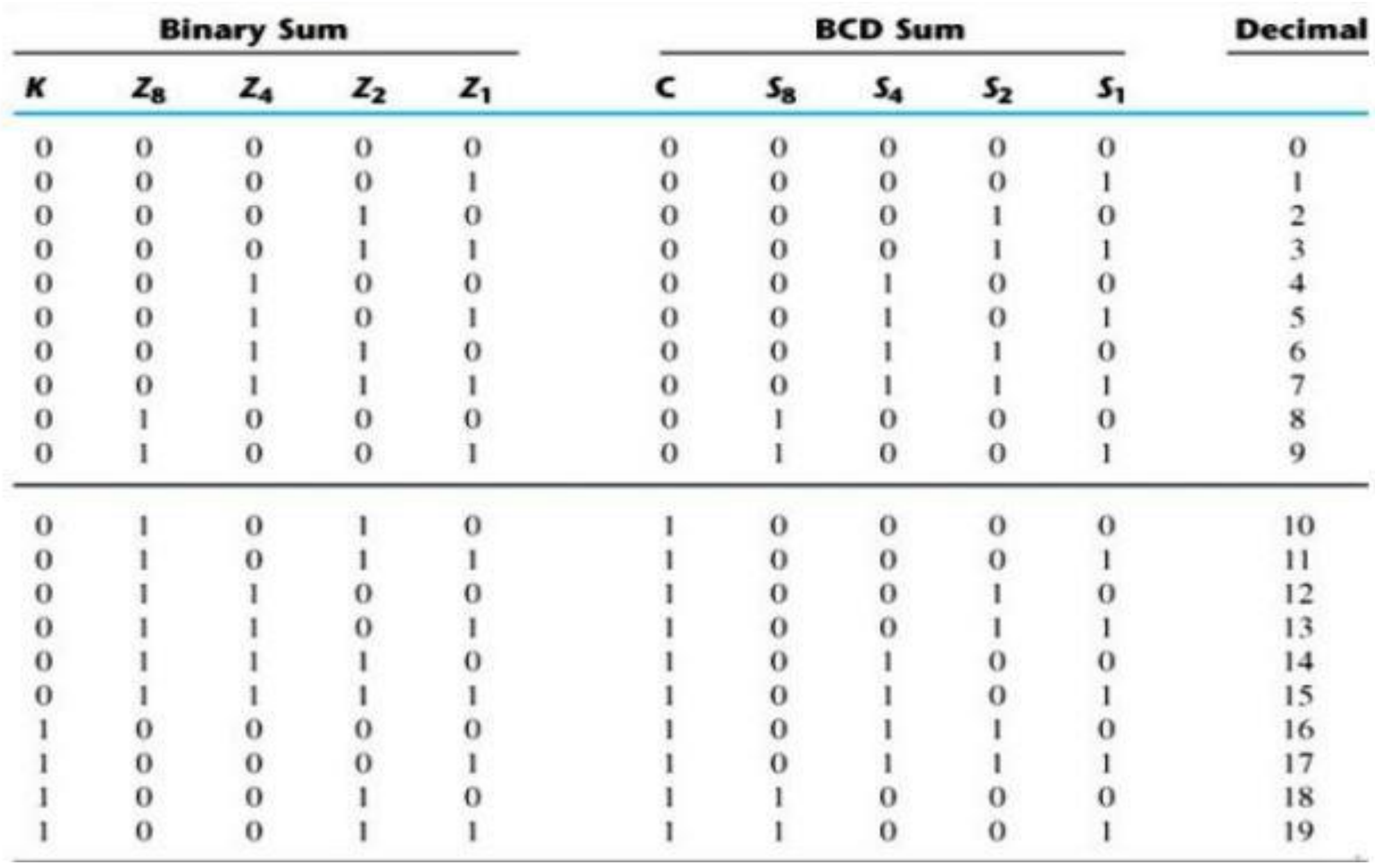
Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary-coded form.

Decimal adder is a combinational circuit that sums up two decimal numbers adopting particular encoding technique. A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input carry and output carry. Of course, there is a wide variety of possible decimal adder circuits, dependent upon the code used to represent the decimal digits.

This combinational circuit adds up two decimal numbers when they are encoded with binary-coded decimal (BCD) form. Adding two decimal digits in BCD, together with a possible carry, the output sum cannot be greater than 9 + 9 + 1 = 19.

Applying two BCD digits to a 4-bit binary adder, the adder will form the sum in binary ranging from 0 to 19. These binary numbers are listed in Table below and are labeled by symbols K, Z8, Z4, Z2, Z1. K is the carry, and the subscripts under the letter z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Truth Table



Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Decimal/BCD Adder

- ✓ In examining the content of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed.
- ✓ When the binary sum is greater than 1001, we obtain a non- valid BCD representation. The addition of binary 0110 (6 in decimal) to the binary sum converts it to the correct BCD representation and also produces an output carry.
- ✓ It is obvious from the table that a correction is needed when the binary sum has an output carry k = 1.
- ✓ If k = 1 (Satisfies 16-19)
- ✓ If $Z_8 .Z_4$ = 1 (Satisfies 12-15)
- ✓ If Z_2 . Z_8= 1 (Satisfies 10 and 11)

Decimal/BCD Adder

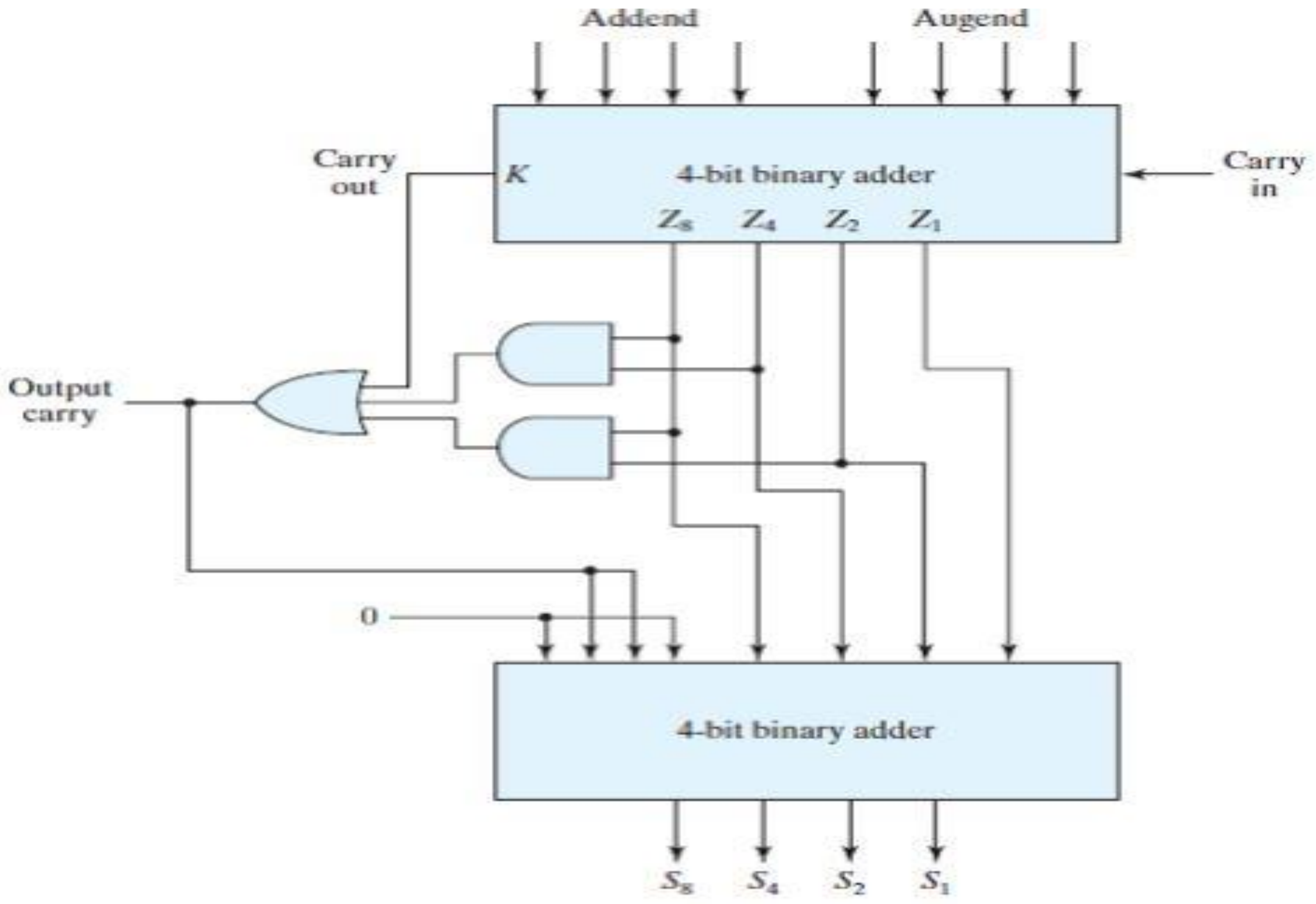


Fig: BCD adder

Magnitude Comparator

A magnitude digital Comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than, or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for A > B condition, one for A = B condition, and one for A < B condition.

1-Bit Magnitude Comparator:

A comparator used to compare two bits is called a single-bit comparator. It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 1-bit comparator is given below:

Magnitude Comparator

A	B	A<B	A>B	A=B
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

From the above truth table logical expressions for each output can be expressed as follows:

A>B: AB'

A<B: A'B

A=B: A’B' + AB

Now, draw the logical diagram

Magnitude Comparator

2-Bit Magnitude Comparator:

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below:

Use truth table to generate expression and draw the logical circuit.

After solving from the k-map , you will get following expression:

A>B:A1B1’ + A0B1’B0’ + A1A0B0’

A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’

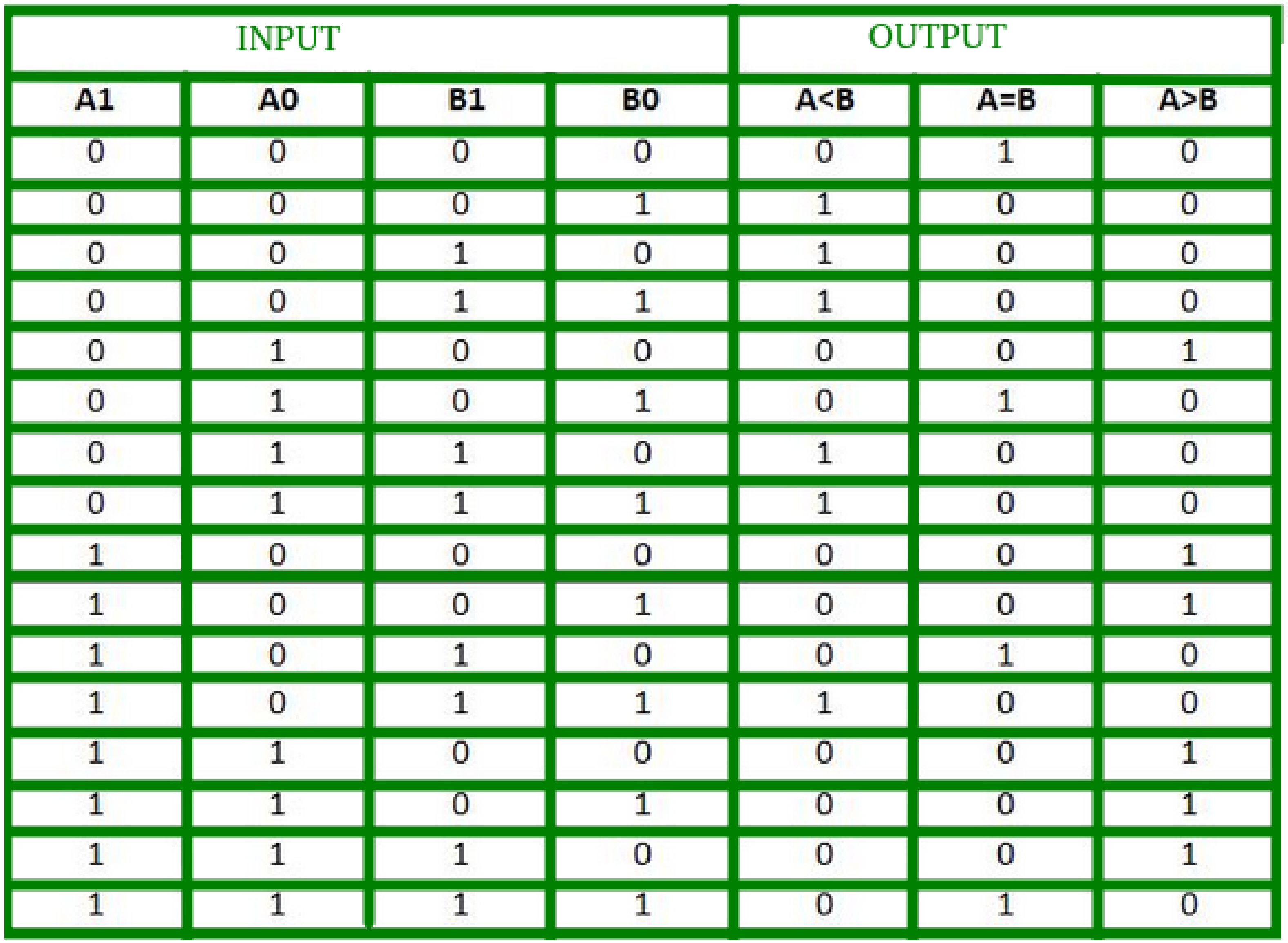
: A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)

: (A0B0 + A0’B0’) (A1B1 + A1’B1’)

: (A0 Ex-Nor B0) (A1 Ex-Nor B1)

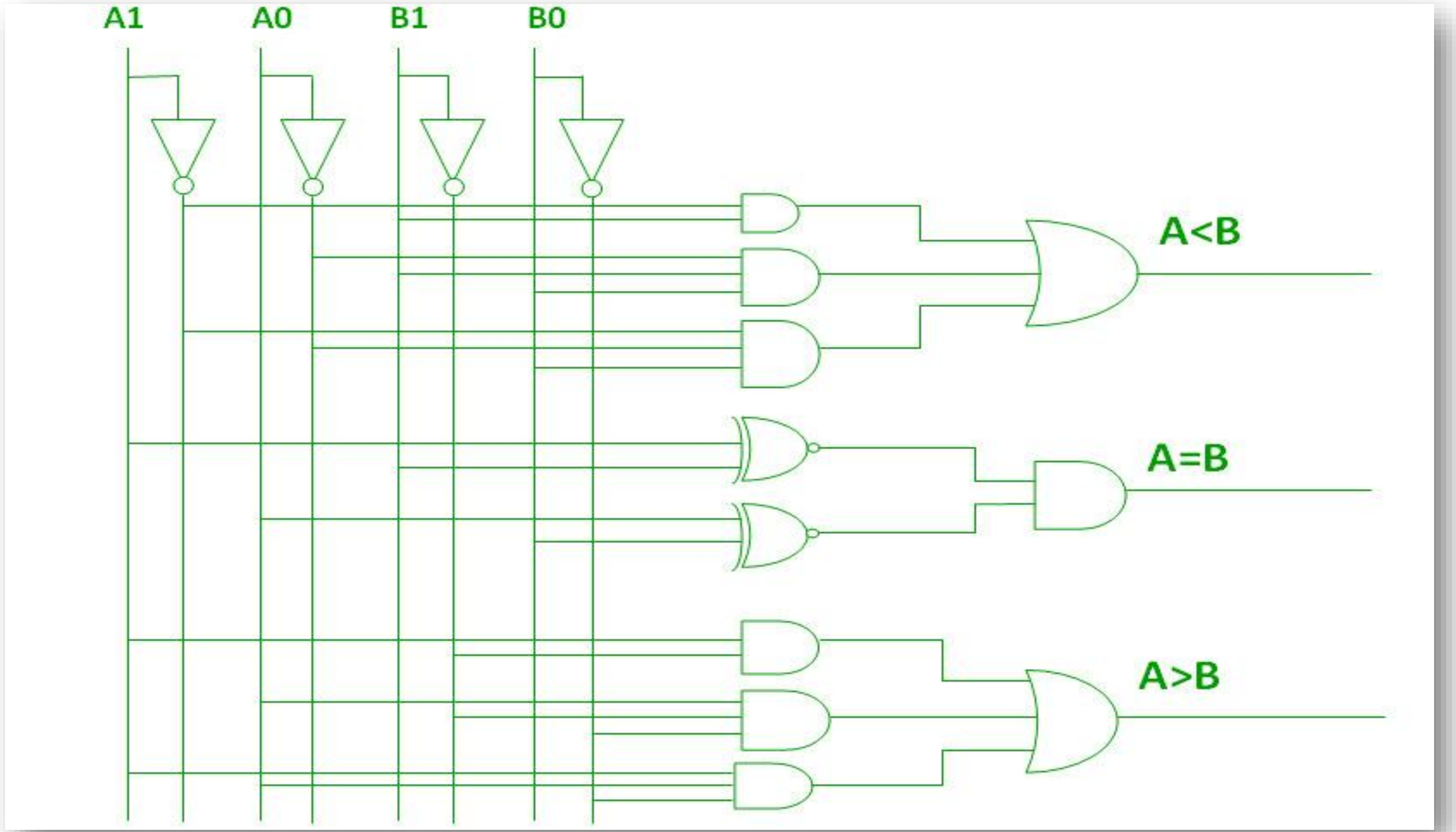
A<B:A1’B1 + A0’B1B0 + A1’A0’B0

Magnitude Comparator



INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Magnitude Comparator



Magnitude Comparator

4-Bit Magnitude Comparator:

A comparator used to compare two binary numbers each of four bits is called a 4-bit magnitude comparator. It consists of eight inputs each for two four-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.

In a 4-bit comparator the condition of A>B can be possible in the following four cases:

If A3 = 1 and B3 = 0

If A3 = B3 and A2 = 1 and B2 = 0

If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0

If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0

Similarly the condition for A<B can be possible in the following four cases:

If A3 = 0 and B3 = 1

If A3 = B3 and A2 = 0 and B2 = 1

If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1

If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

The condition of A=B is possible only when all the individual bits of one number exactly coincide with corresponding bits of another number.

4-Bit Magnitude Comparator

Use the reference of 1bit magnitude comparator. i.e

A>B: AB’

A<B: A’B

A=B: A’B’+AB= A x-nor B

$X_i = (A_i = B_i) = A_i$ xnor B_i

A_3 B_3	A_2 B_2	A_1 B_1	A_0 B_0	A > B	A<B	A=B
$A_3 > B_3$ $A_3 < B_3$	X X	X X	X X	1 0	0 1	0 0
$A_3 = B_3$ X_3	$A_2 > B_2$ $A_2 < B_2$			1 0	0 1	0 0
$A_3 = B_3$ X_3	$A_2 = B_2$ X_2	$A_1 > B_1$ $A_1 < B_1$		1 0	0 1	0 0
$A_3 = B_3$ X_3	$A_2 = B_2$ X_2	$A_1 = B_1$ X_1	$A_0 > B_0$ $A_0 < B_0$	1 0	0 1	0 0
X_3	X_2	X_1	X_0	0	0	1

4-Bit Magnitude Comparator

$A>B= A_3\ B_3'+X_3A_2B_2'+ X_3\ X_2A_1B_1'+ X_3\ X_2\ X_1\ A_0B_0'$

$A<B= A'_3\ B_3+X_3A'_2B_2+ X_3\ X_2A'_1B_1+ X_3\ X_2\ X_1\ A'_0B_0$

$(A=B) = X_3\ X_2\ X_1\ X_0$

$X_i= (A_i$ x-or $B_i)' = (A_iB_i'+ A'_iB_i)'$

So,

For X_3

$(A_3B_3'+A_3'\ B3)'$

Similarly make circuit for X_2 , X_1 and X_0

4-Bit Magnitude Comparator

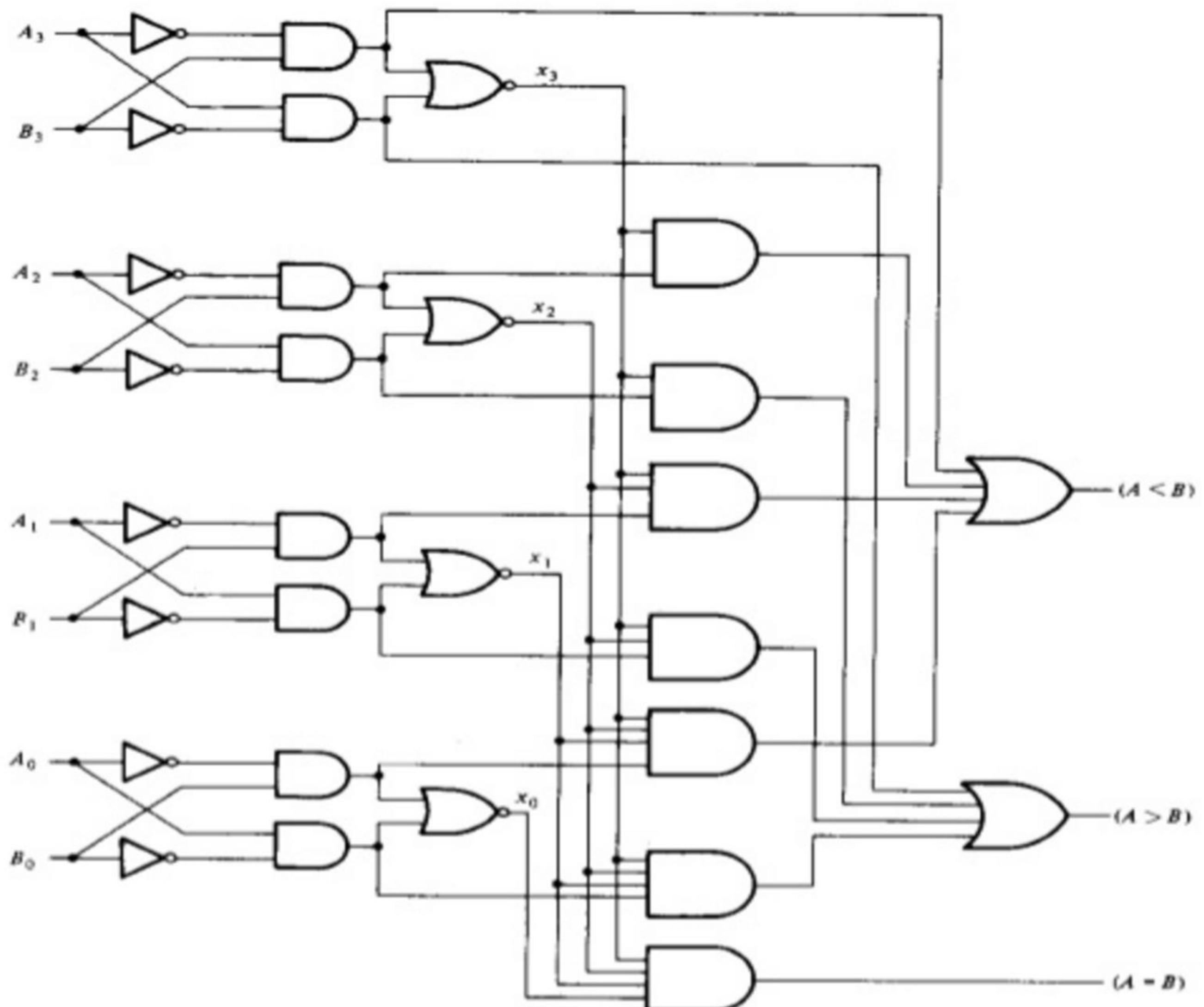


FIGURE 5-7
4-bit magnitude comparator

Encoders and Decoders

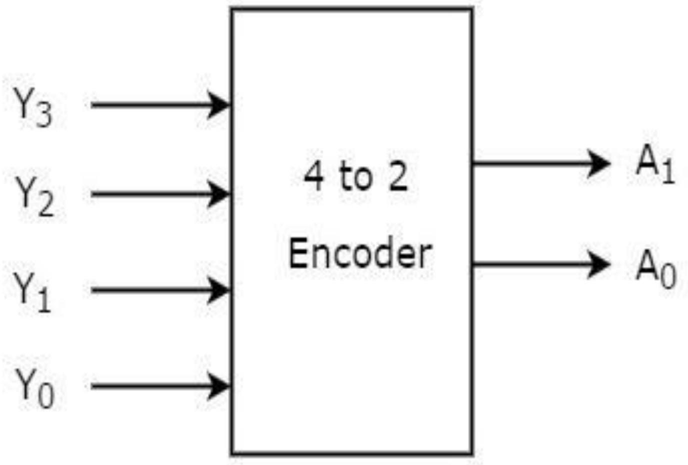
Binary code of N digits can be used to store 2^N distinct elements of coded information. This is what encoders and decoders are used for. Encoders convert 2^N lines of input into a code of N bits and Decoders decode the N bits into 2^N lines.

Encoders :

An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

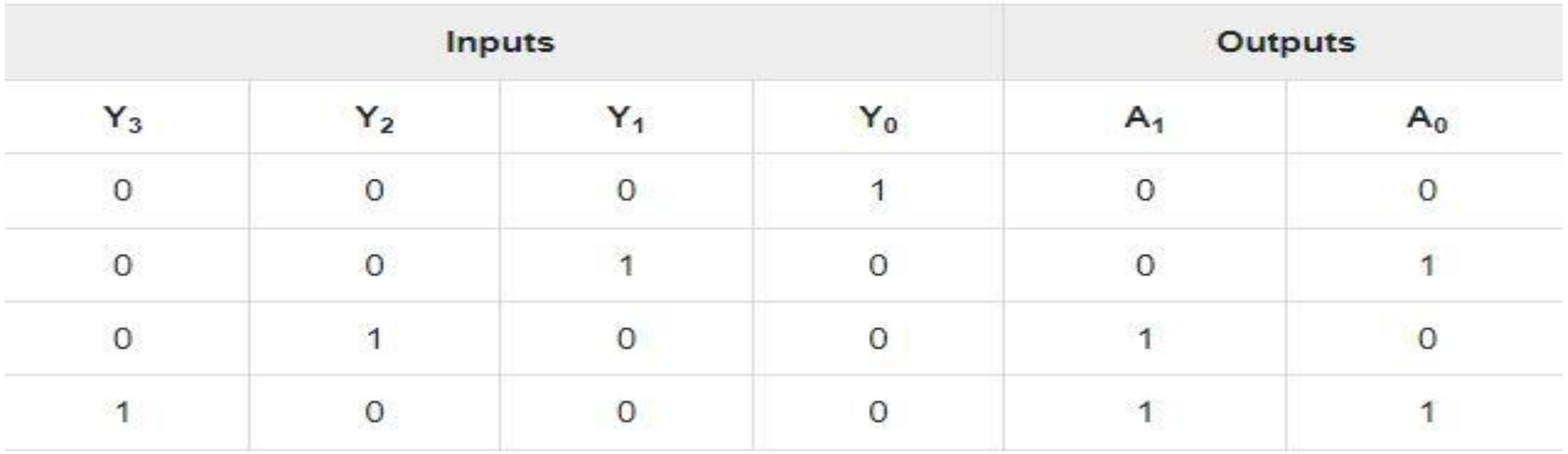
4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0. 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.

4 to 2 Encoder



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

$A1=Y3+Y_2$

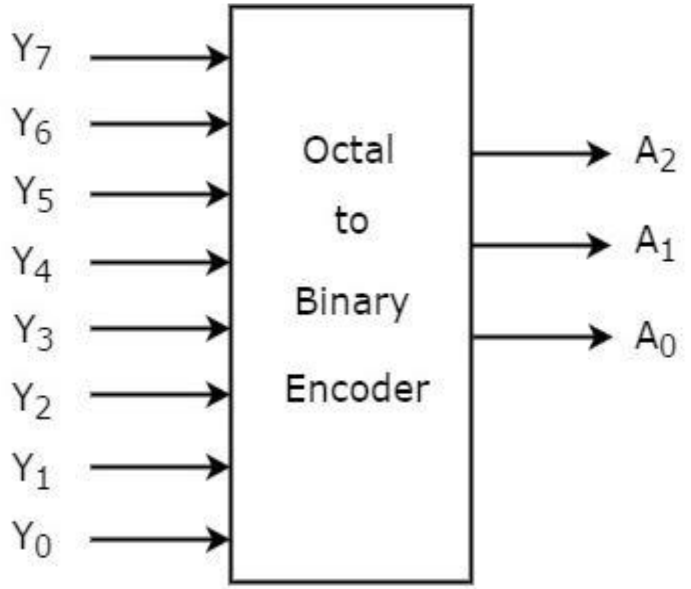
$A_0=Y_3+Y_1$

We can implement the above two Boolean functions by using two input OR gates.

Now , draw the circuit..

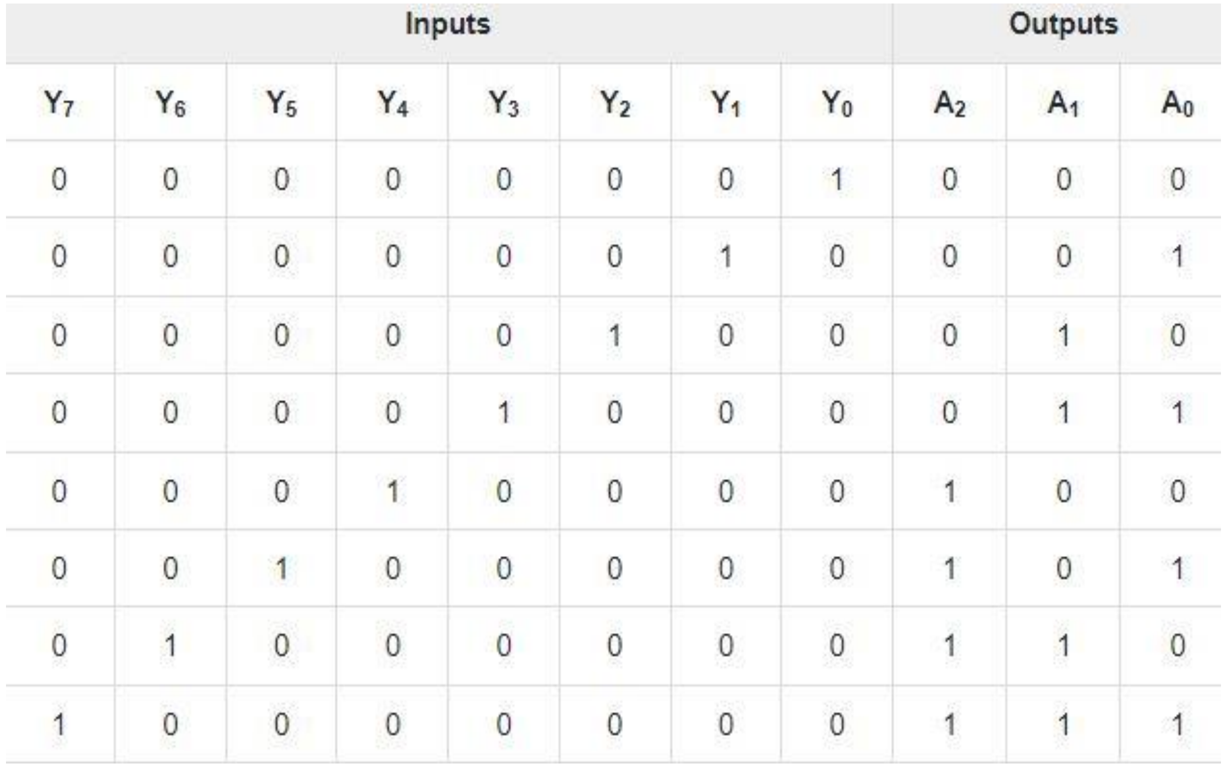
Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y7 to Y0 and three outputs A2, A1 & A0. 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.

Truth table



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:

A2=Y7+Y6+Y5+Y4

A1=Y7+Y6+Y3+Y2

A0=Y7+Y5+Y3+Y1

We can implement the above Boolean functions by using four input OR gates.

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 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’.

So, 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.

Priority encoder

A 4 to 2 priority encoder has four inputs Y3, Y2, Y1 & Y0 and two 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.

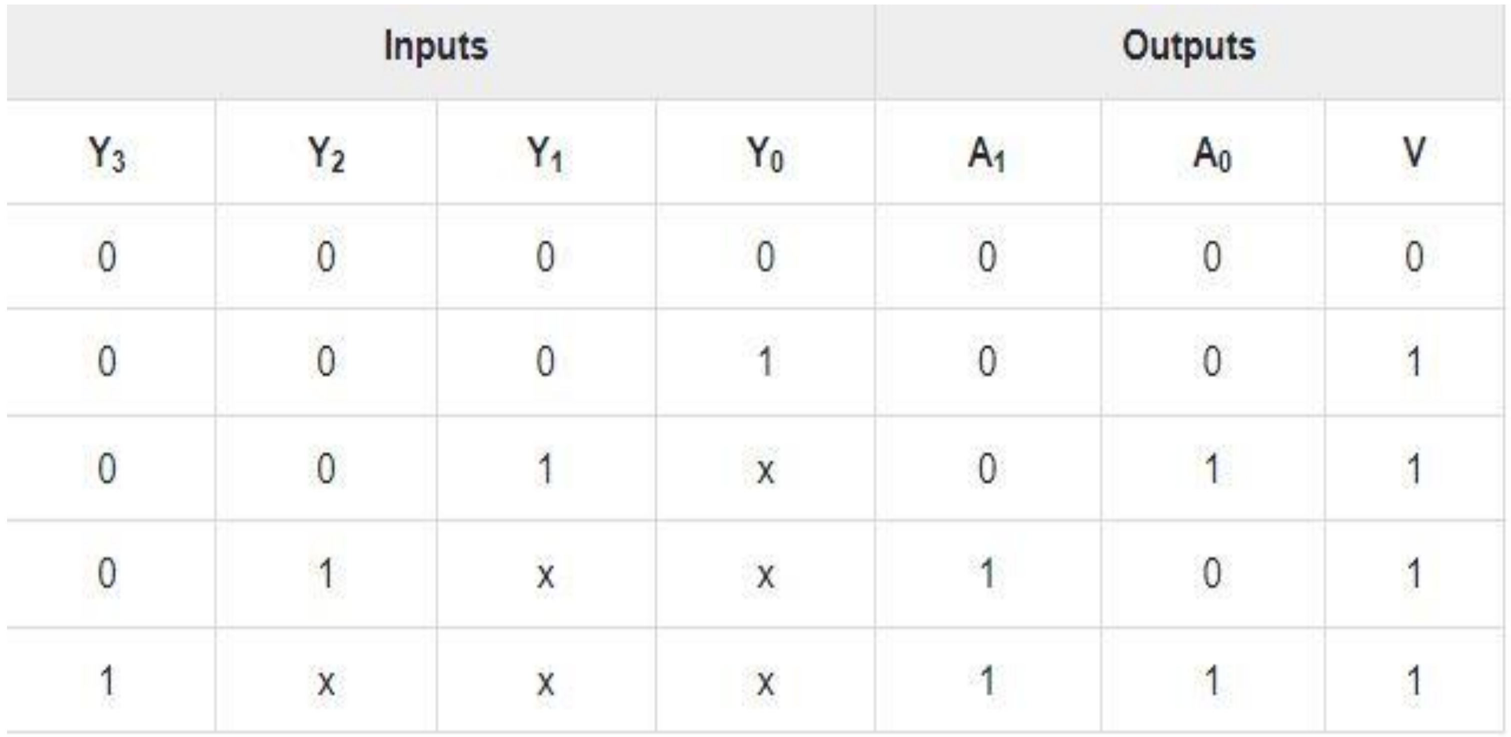
We considered one more output, V in order to know, whether the code available at outputs is valid or not.

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.

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

Priority encoder



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

Use 4 variable K-maps for getting simplified expressions for each output.

The simplified Boolean functions are

A1=Y3+Y2

A0=Y3+Y2′Y1

V=Y3+Y2+Y1+Y0

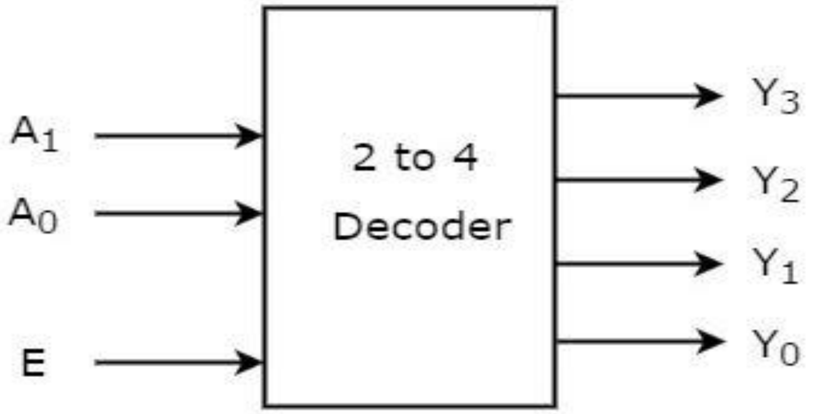
Now, Draw the logical diagram

Decoder

Decoder is a combinational circuit that has ‘n’ input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of ‘n’ input variables lines, when it is enabled.

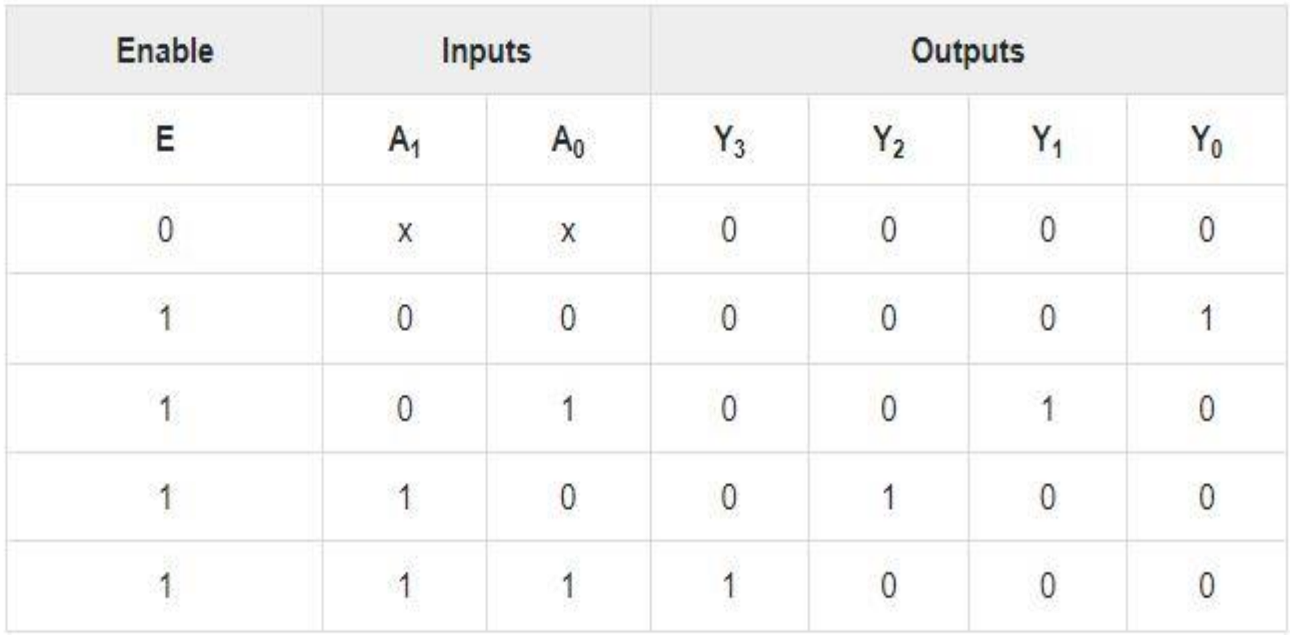
2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A1 & A0 and four outputs Y3, Y2, Y1 & Y0. The block diagram of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’. The Truth table of 2 to 4 decoder is shown below

2:4 Decoder



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

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

$Y3=E.A1.A0$

$Y2=E.A1.A0'$

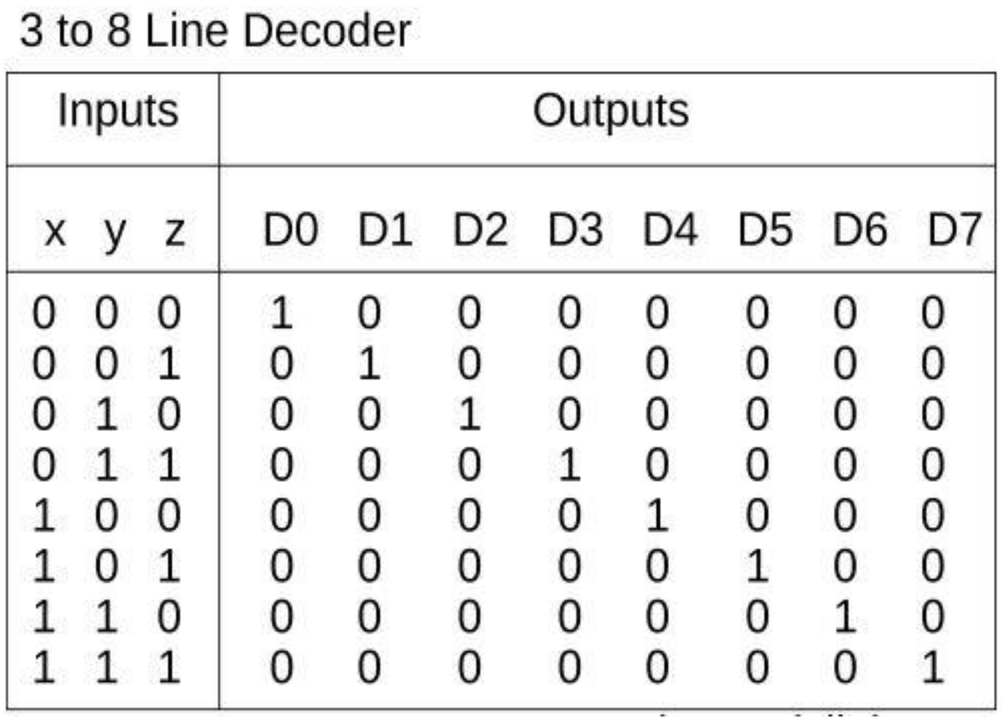
$Y1=E.A1'.A0$

$Y0=E.A1'.A0'$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters.

3:8 Decoder

The number of available inputs are 3 and outputs are 8.Let us represent the inputs and outputs by symbol letters. Let us represent the inputs by x, y, and z; and the outputs by D0, D1, D2, … D7.



3 to 8 Line Decoder

Inputs			Outputs							
x	y	z	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

3:8 Decoder

Now functions.

D_0 =x'y'z'

D_1=x'y'z

D_2=x'yz'

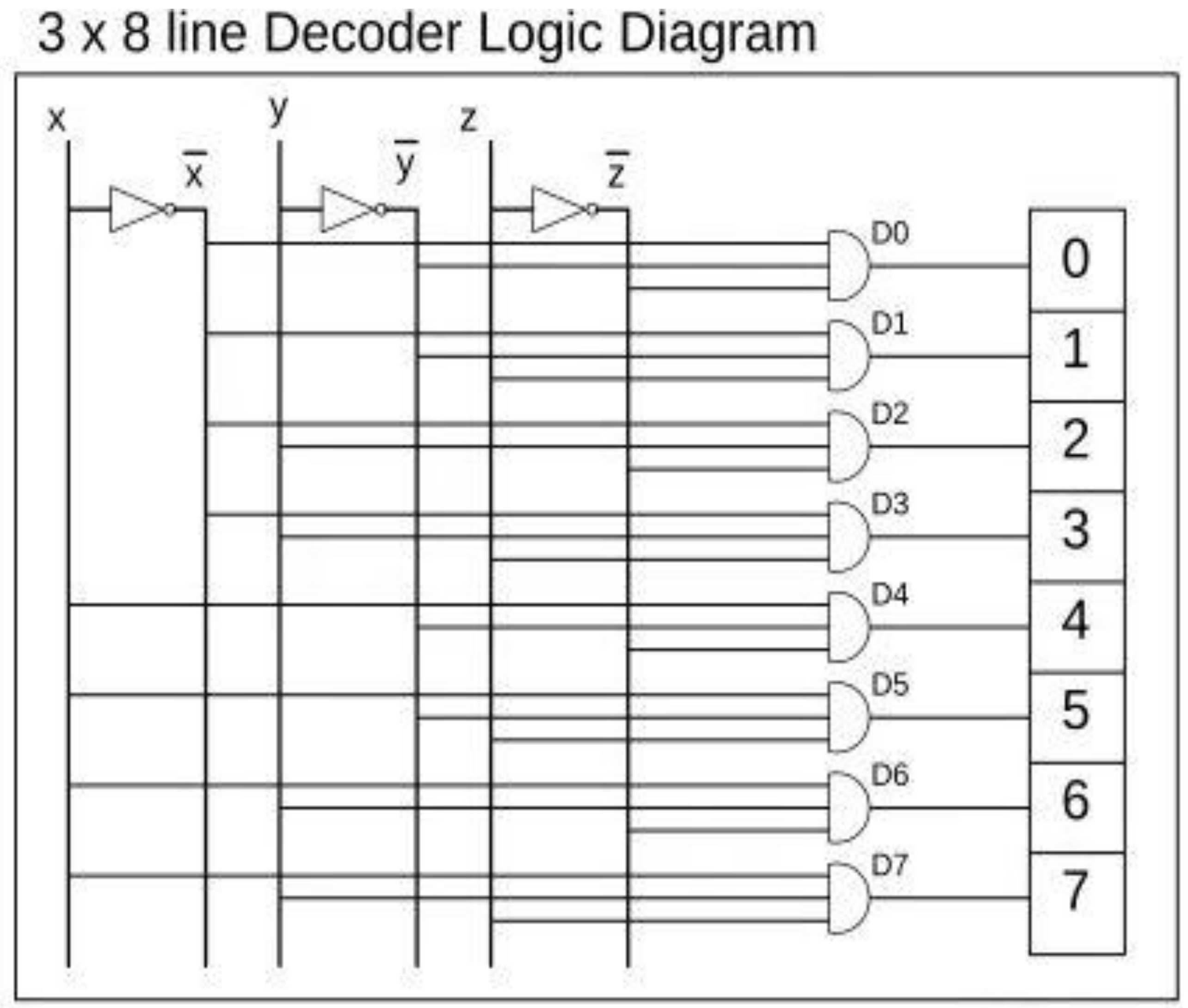
D_3=x'yz

D_4=xy'z'

D_5=xy'z

D_6=xyz'

D_7=xyz



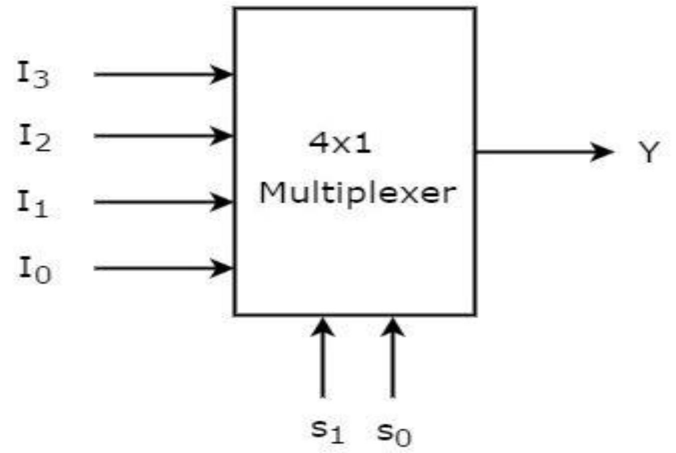
Multiplexer

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.

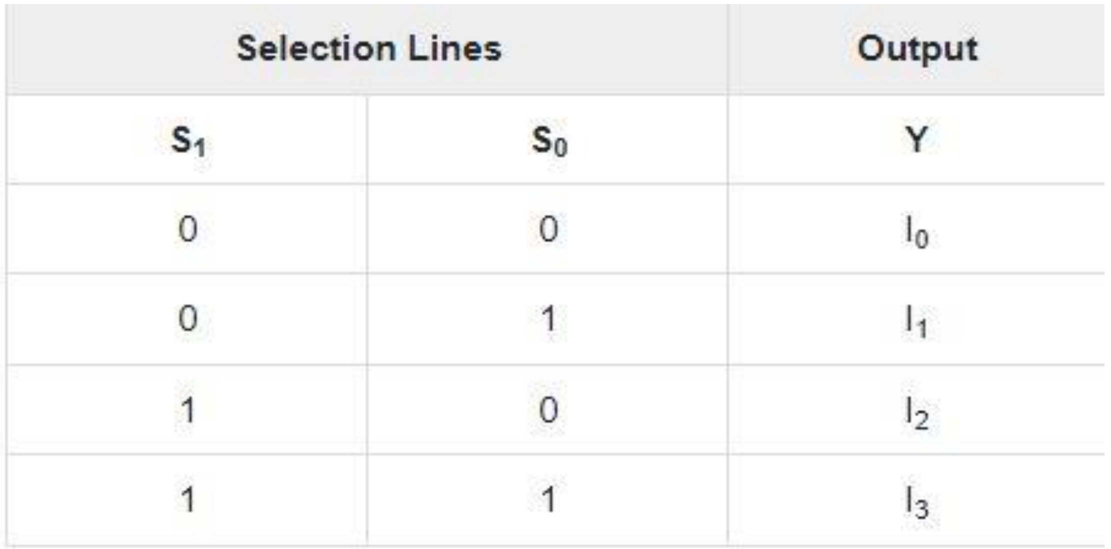
4x1 Multiplexer

4x1 Multiplexer has four data inputs I3, I2, I1 & I0, two selection lines s1 & s0 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.



Multiplexer

One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. Truth table of 4x1 Multiplexer is shown below.



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

From Truth table, we can directly write the Boolean function for output, Y as

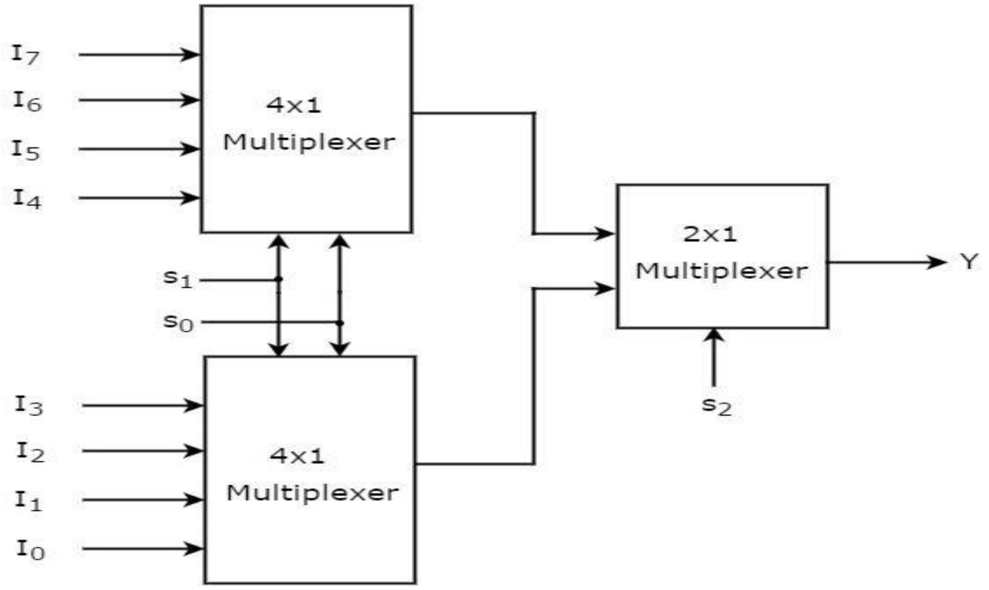
$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$

We can implement this Boolean function using Inverters, AND gates & OR gate.

8x1 Multiplexer

implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two 4x1 Multiplexers in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output



Exercise

1. Design the circuit diagram of 8X1 multiplexer.
2. Design 4X16 decoder using 3X8 decoder.

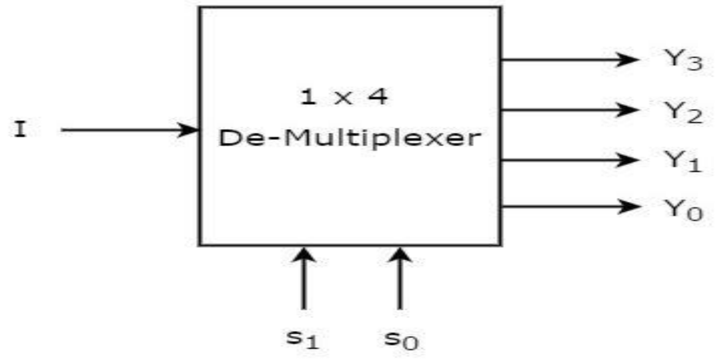
De-Multiplexer

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, ‘n’ selection lines and maximum of 2n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are ‘n’ selection lines, there will be 2n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as De-Mux.

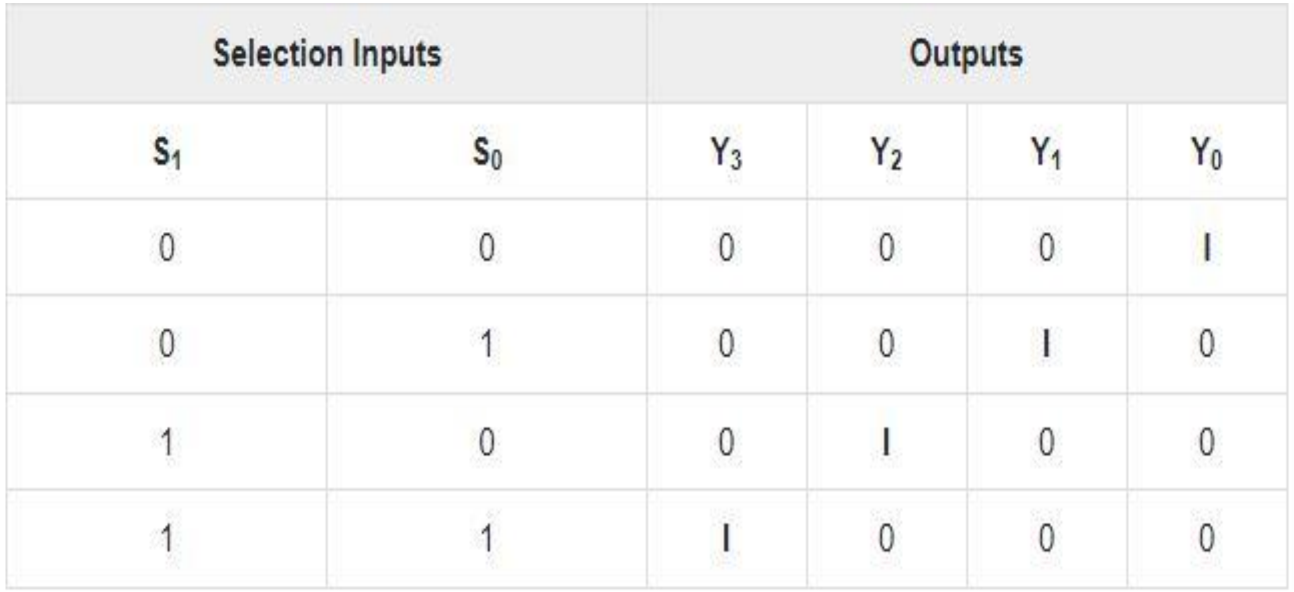
1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s1 & s0 and four outputs Y3, Y2, Y1 &Y0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.



De-Multiplexer

The single input ‘I’ will be connected to one of the four outputs, Y3 to Y0 based on the values of selection lines s1 & s0. The Truth table of 1x4 De-Multiplexer is shown below.



Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

$Y3=s1s0I$

$Y2=s1s0'I$

$Y1=s1's0I$

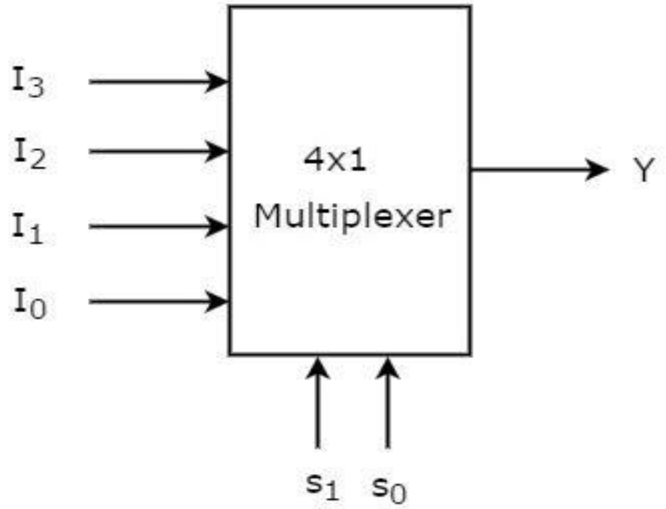
$Y0=s1's0'I$

Implementation of SOP using multiplexer

Q. Implement F=∑(1,3,5,6) using multiplexer.

Here, we need three variable to represent the above function. Let three variable be A,B,C.

Lets use 4x1 multiplexer to represent above function.



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

S_0 and S_1 can be any out of A,B,C . Let S_1 =A and S_0 =B

So 3 variable k-map is ,

Implementation of SOP using multiplexer

A \ BC	00	01	11	10
0	0	1	1	
1	0	1	0	1

Truth table of 4X1 multiplexer is,

S_1 (B)	S_0 (C)	Y
0	0	I_0 =0
0	1	I_1 =1
1	0	I_2 =A
1	1	I_3 =A’

Now , draw the multiplexer based on the above truth table.

Q. Implement F=∑(1,4,5,7,9,12,13) using multiplexer.

Decoder problem

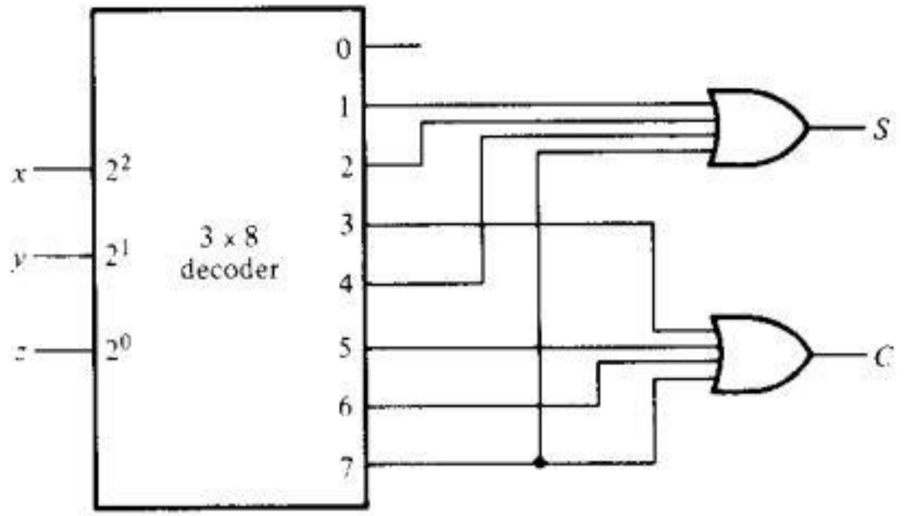
Q. Implement the full adder circuit using decoder and OR gate.

From the truth table of the full adder, we obtain the functions for this combinational circuit in sum of minterms:

$S(x,y,z)=\sum(1,2,4,7)$

$C(x,y,z)=\sum(3,5,6,7)$

Since there are three inputs and total of eight minterms, we need a 3 to 8 line decoder. The implementation is shown below.



Decoder problem

Q. Using a decoder and external gates, design the combinational circuit defined by the following three Boolean functions:

F1 = x′y′z + xz′

F2 = x’yz′ + xy′

F3 = xyz′ + xy

Convert the above function in sum of minterms:

F_1=x’y’z’+x(y+y’)z’

=x’y’z’+xyz’+xy’z’=∑(0,6,4)

F_2=x’yz’+xy’= xyz’+xy’(z+z’)

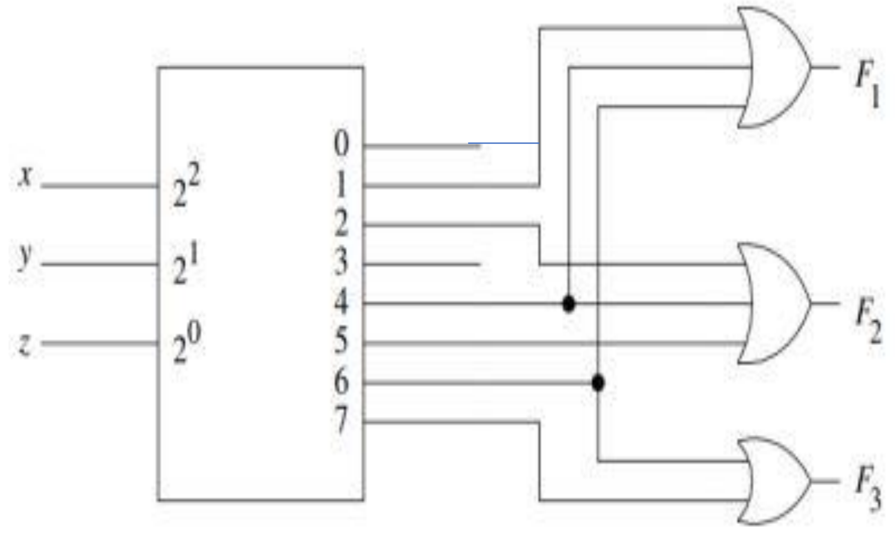
=xyz’+xy’z+xy’z’

=∑(6,5,4)

F3 = xyz′ + xy=xyz’+xy(z+z’)

=xyz’+xyz+xyz’

=∑(6,7)



Logic Diagram

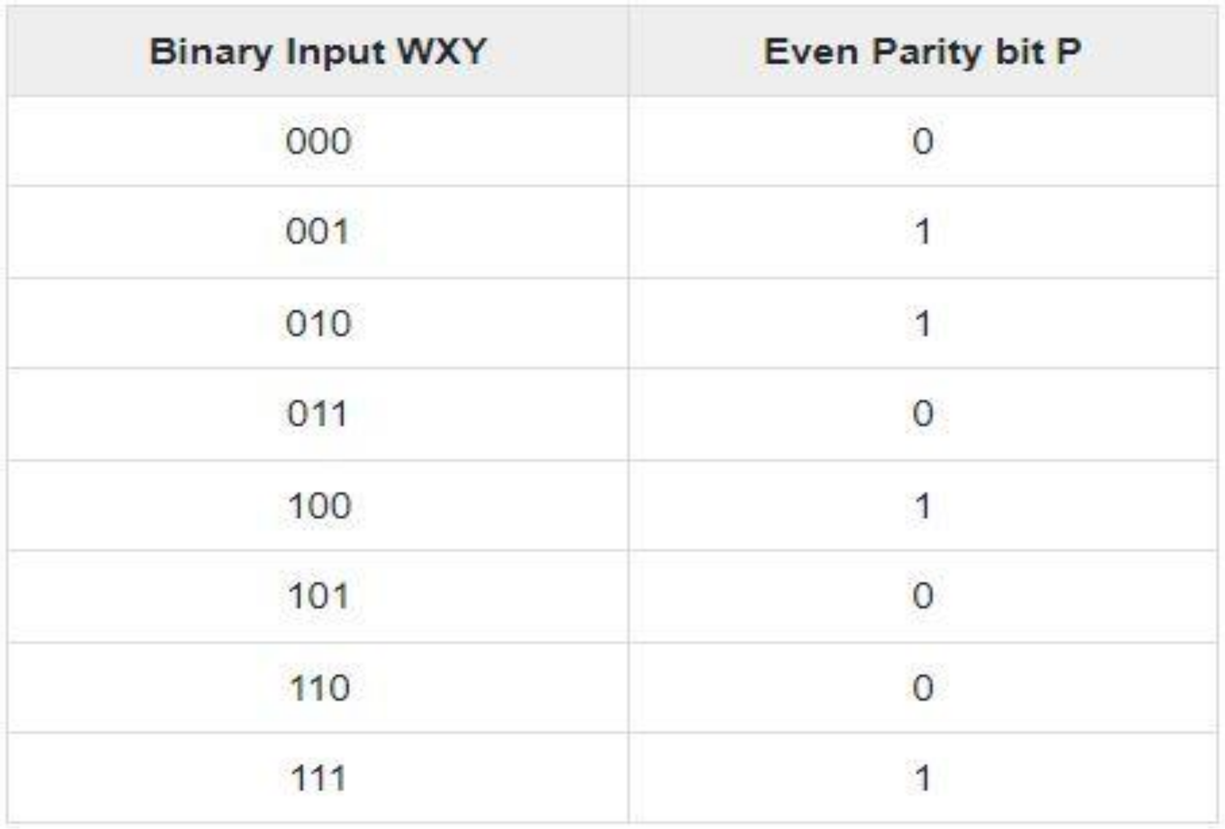
Parity Bit Generator(Imp)

There are two types of parity bit generators based on the type of parity bit being generated. Even parity generator generates an even parity bit. Similarly, odd parity generator generates an odd parity bit.

Even Parity Generator:

Now, let us implement an even parity generator for a 3-bit binary input, WXY. It generates an even parity bit, P. If odd number of ones present in the input, then even parity bit, P should be ‘1’ so that the resultant word contains even number of ones. For other combinations of input, even parity bit, P should be ‘0’. The following table shows the Truth table of even parity generator.

Even Parity Generator



Binary Input WXY	Even Parity bit P
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

From the above Truth table, we can write the Boolean function for even parity bit as

$P=W'X'Y+W'XY'+WX'Y'+WXY$

$=W'(X'Y+XY')+W(X'Y'+XY)= W'(X\oplus Y)+W(X\oplus Y)'=W\oplus X\oplus Y$

Odd Parity Generator

It generates an odd parity bit, P. If even number of ones present in the input, then odd parity bit, P should be ‘1’ so that the resultant word contains odd number of ones. For other combinations of input, odd parity bit, P should be ‘0’.

Draw the truth table and circuit diagram based on the Boolean expression.

??

Parity Check

It is a logic circuit that checks for possible errors in the transmission. This circuit can be an even parity checker or odd parity checker depending on the type of parity generated at the transmission end. When this circuit is used as even parity checker, the number of input bits must always be even.

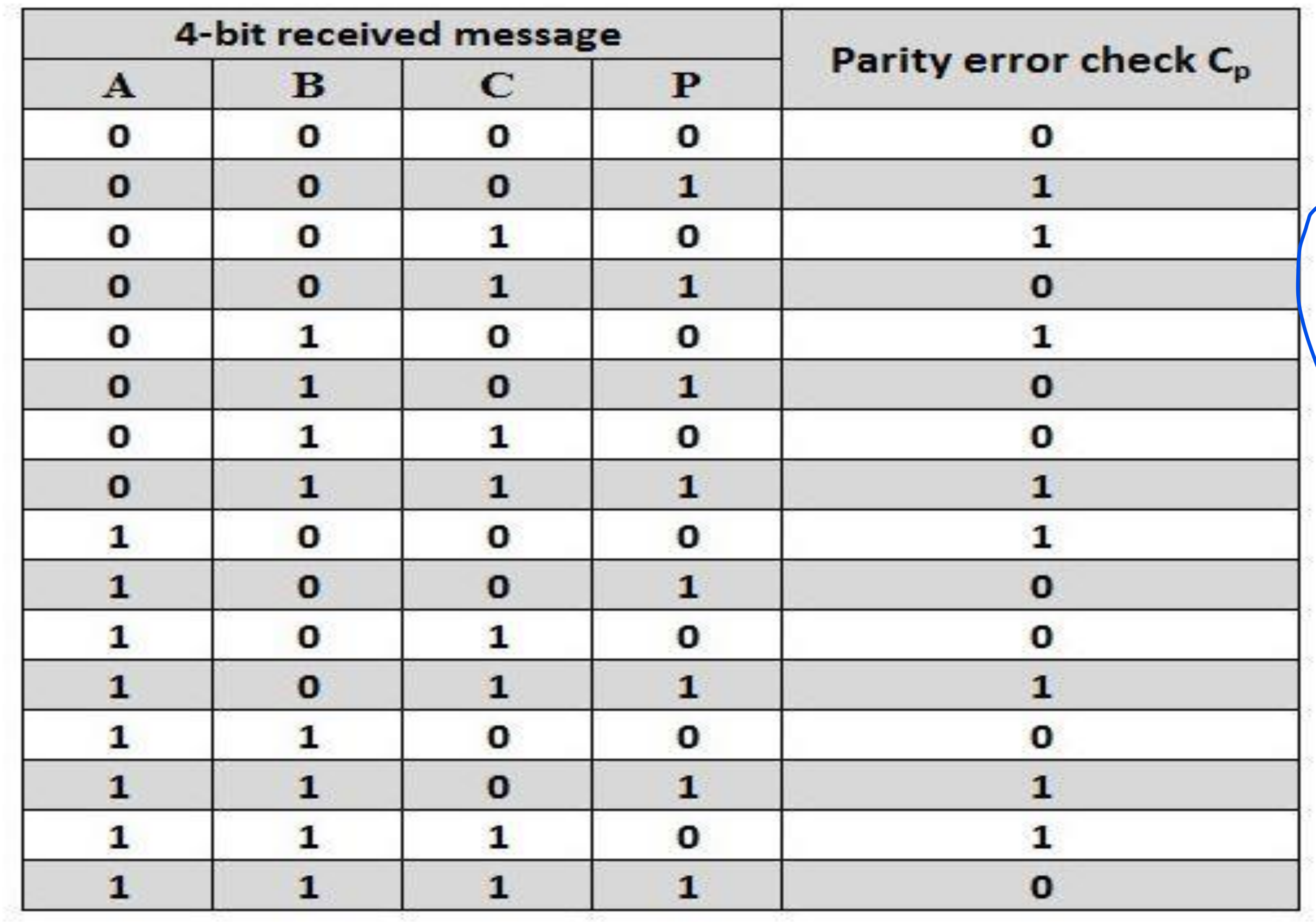
Even Parity Checker

Consider that three input message along with even parity bit is generated at the transmitting end. These 4 bits are applied as input to the parity checker circuit, which checks the possibility of error on the data. Since the data is transmitted with even parity, four bits received at circuit must have an even number of 1s.

If any error occurs, the received message consists of odd number of 1s. The output of the parity checker is denoted by PEC (Parity Error Check).

The below table shows the truth table for the Even Parity Checker in which PEC = 1 if the error occurs, i.e., the four bits received have odd number of 1s and PEC = 0 if no error occurs, i.e., if the 4-bit message has even number of 1s.

Even Parity Checker



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Logic Diagram also

Now draw k map to obtain Boolean expression then draw logical circuit for even parity checker.

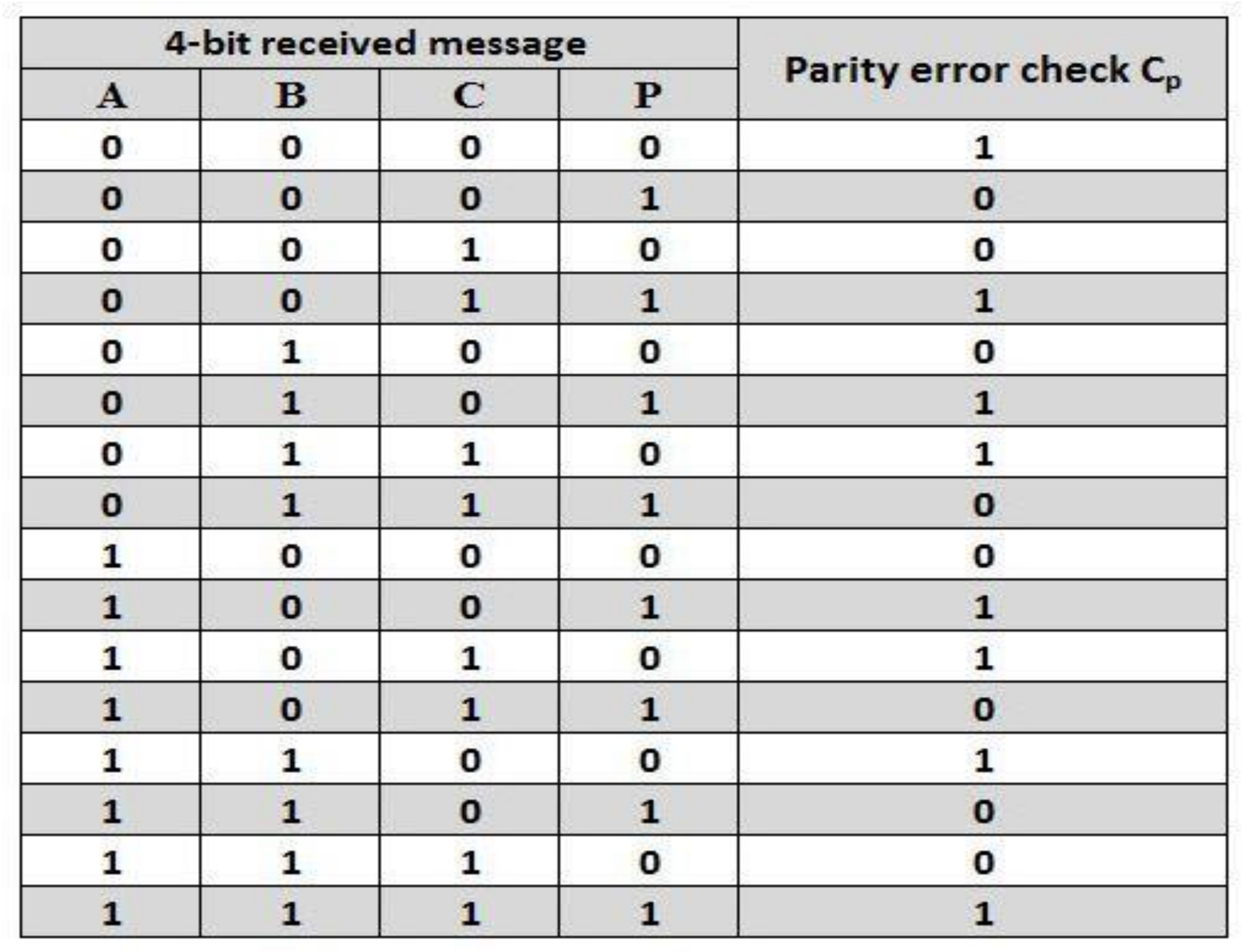
Odd Parity Checker

Consider that a three bit message along with odd parity bit is transmitted at the transmitting end. Odd parity checker circuit receives these 4 bits and checks whether any error are present in the data.

If the total number of 1s in the data is odd, then it indicates no error, whereas if the total number of 1s is even then it indicates the error since the data is transmitted with odd parity at transmitting end.

The below figure shows the truth table for odd parity generator where PEC =1 if the 4-bit message received consists of even number of 1s (hence the error occurred) and PEC= 0 if the message contains odd number of 1s (that means no error).

Odd Parity Checker



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

After simplification, the final expression for the PEC is obtained as
PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR P)

Read Only Memory (ROM)

- A read-only memory (ROM) is a device that includes both the decoder and the OR gates within a single IC package. The connections between the outputs of the decoder and the inputs of the OR gates can be specified for each particular configuration by "programming" the ROM.
- A ROM is essentially a memory (or storage) device in which a fixed set of binary information is stored.
- The binary information must first be specified by the user and is then embedded in the unit to form the required interconnection pattern. ROM's come with special internal links that can be fused or broken. The desired interconnection for a particular application requires that certain links be fused to form the required circuit paths. Once a pattern is established for a ROM, it remain fixed even when power is turned off and then on again.
- A ROM consists of n input lines and m output lines.
- Each bit combination of input variables is called an address.
- Each bit combination that comes out of the output lines is called a word. The number of bits per word is equal to the number of output lines m.
- A ROM with n input lines has 2^n distinct addresses, so there are 2^n distinct words which are said to be stored in the unit.
- Internally, the ROM is a combinational circuit with AND gates connected as a decoder and a number of OR gates equal to the number of outputs in the unit.

Read Only Memory (ROM)

Consider a 32 x 4 ROM. The unit consists of 32 words of 8 bits each. This means that there are eight output lines and that there are 32, distinct words stored in the unit, each of which may be applied to the output lines. The particular word selected that is presently available on the output lines is determined from the five input lines. There are only five inputs in a 32 x 8 ROM because $2^5 = 32$, and with five variables, we can specify 32 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 00000, word number O is selected and it appears on the output lines. if the input address is 11111, word number 31 is selected and applied to the output lines. In between, there are 30 other addresses that can select the other 30 words.

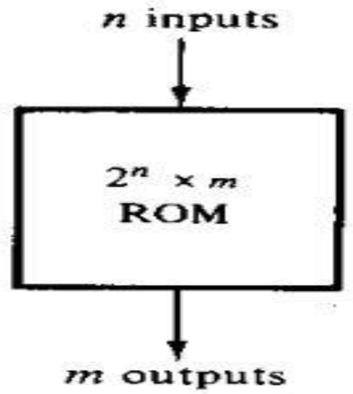


FIGURE 5-21
ROM block diagram

32 x 4 ROM

The five input variables are decoded into 32 lines. Each output of the decoder represents one of the minterms of a function of five variables. Each one of the 32 addresses selects one and only one output from the decoder. The address is a 5-bit number applied to the inputs, and the selected minterm out of the decoder is the one marked with the equivalent decimal number. The 32 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 32 inputs and each input goes through a fuse that can be blown as desired.

32 x 4 ROM

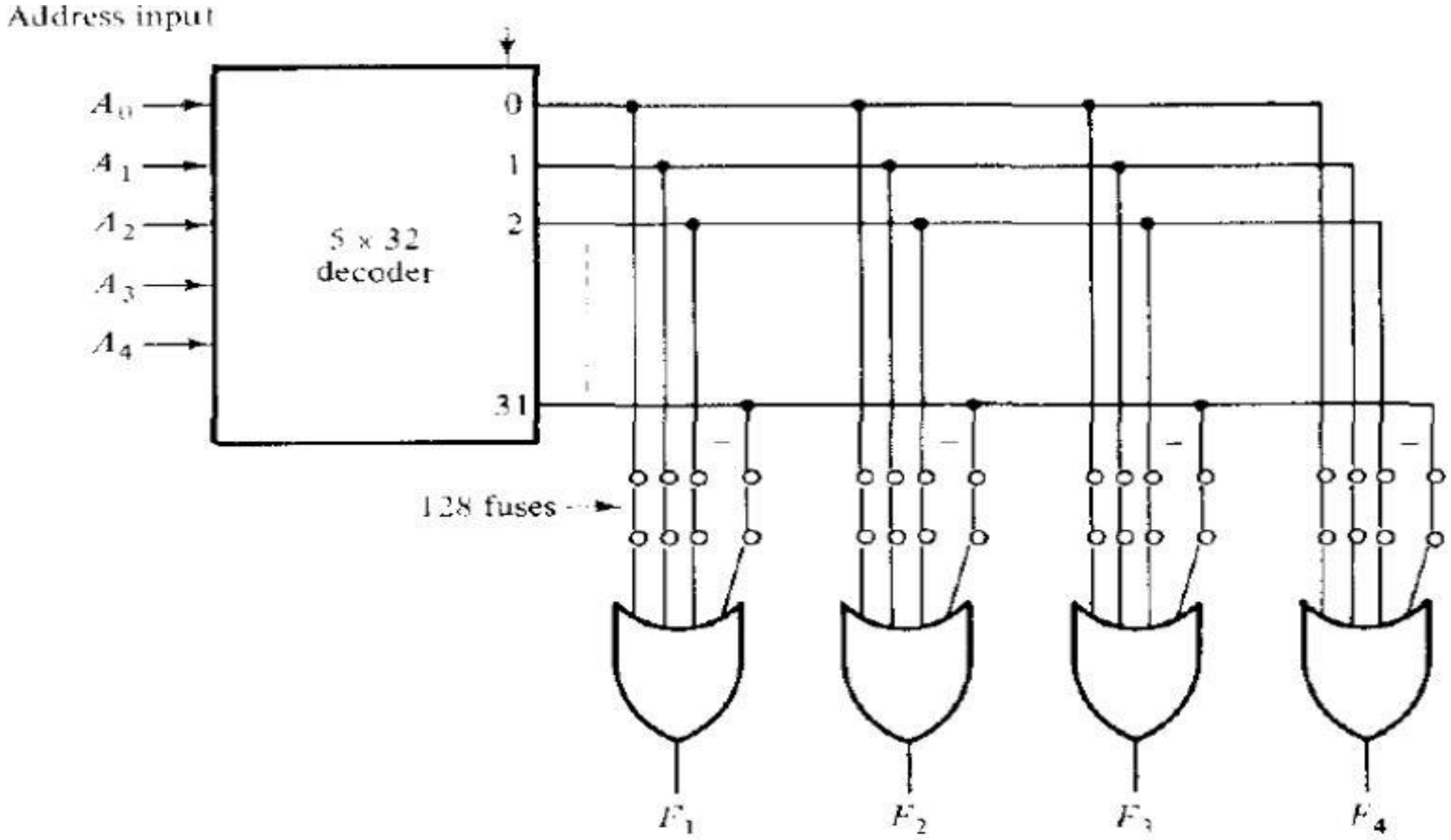


Fig: logic construction of a 32 x 4 ROM

Combinational Logic Implementation

Q. Design ROM for the following output function.

A1	A2	F1	F2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

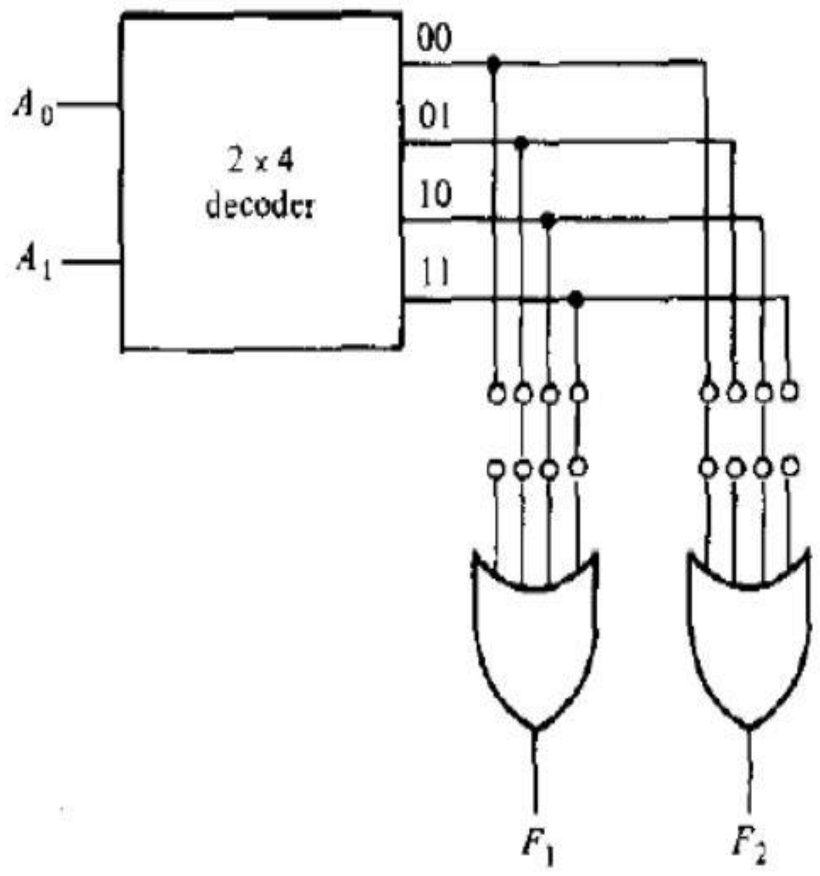
Truth table specifies a combinational circuit with 2 inputs and 2 outputs. The Boolean function can be represented in SOP:

$F1(A1,A0)=\sum(1,2,3)$

$F1(A1,A0)=\sum(0,2)$

Combinational Logic Implementation

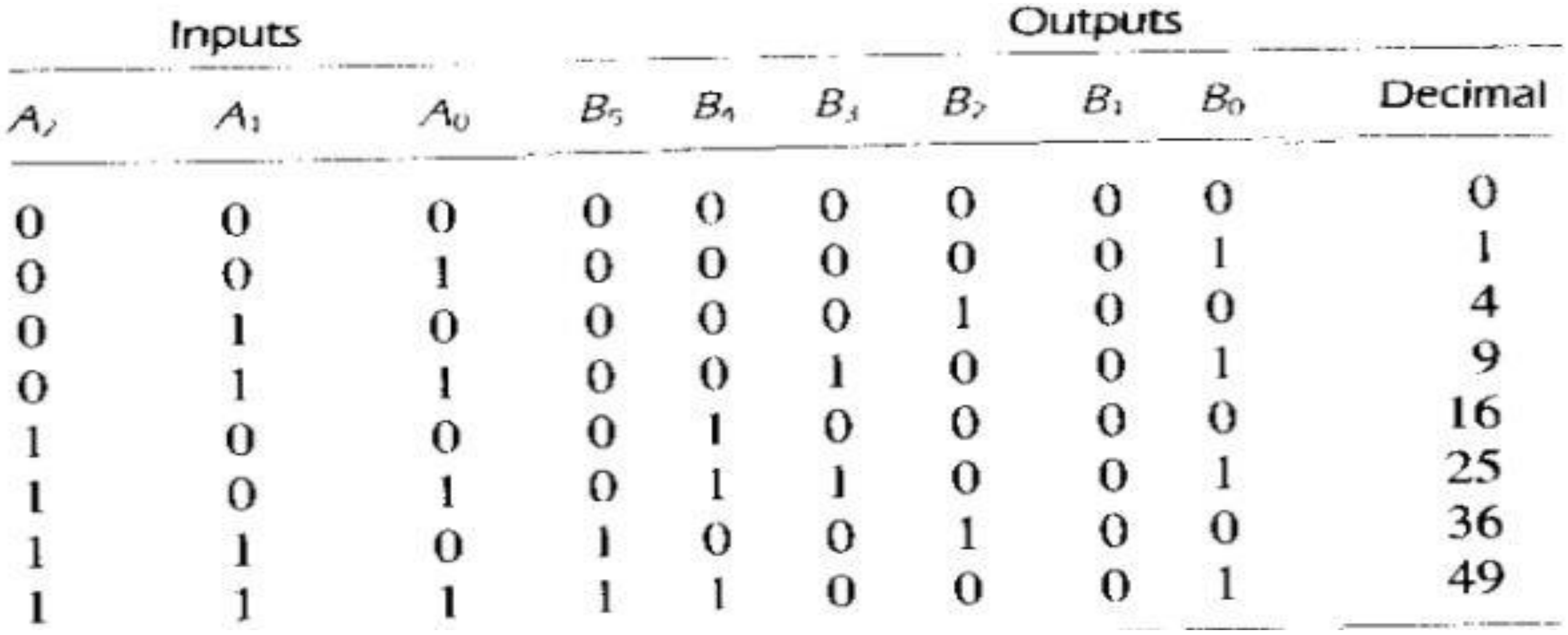
Diagram shows the internal construction of a 4X2 ROM. It is now necessary to determine which of the eight available fuses must be blown and which should be left intact. This can be easily done from the output functions listed in the truth table. Those minterms that specify an output of 0 should not have a path to the output through the OR gate. Thus, for this particular case, the truth table shows three 0's, and their corresponding fuses to the OR gates must be blown.



Combinational Logic Implementation

Q. 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.

The first step is to derive the truth table for the combinational circuit. In most cases, this is all that is needed. In some cases, we can fit a smaller truth table for the ROM by using certain properties in the truth table of the combinational circuit.



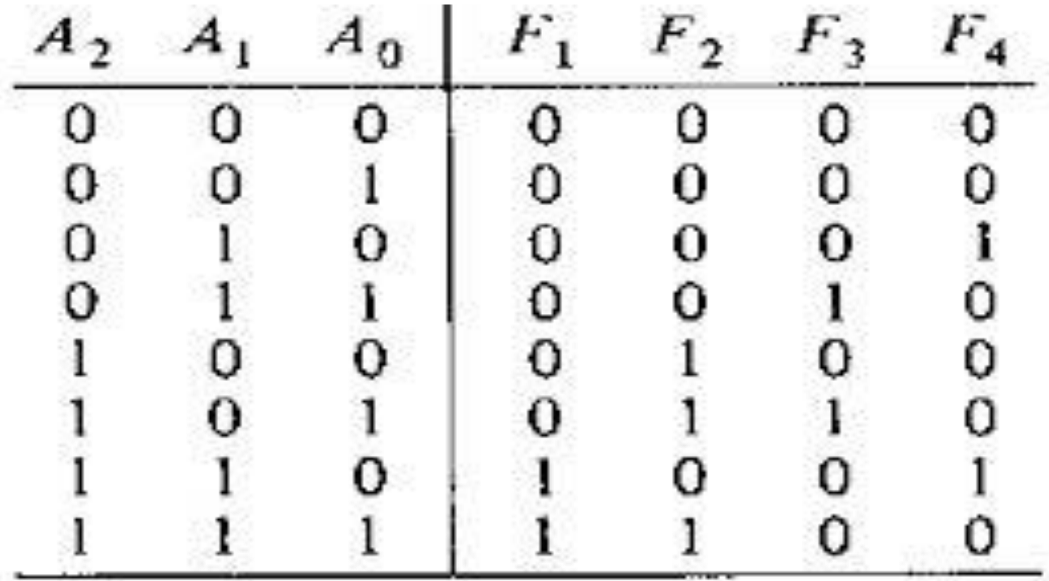
Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Combinational Logic Implementation

We note that output B_0 is always equal to input A_0 so there is no need to generate B_0 with a ROM since it is equal to an input variable. Moreover, output B_1 is always 0, so this output is always known. We actually need to generate only four outputs with the ROM; the other two are easily obtained.

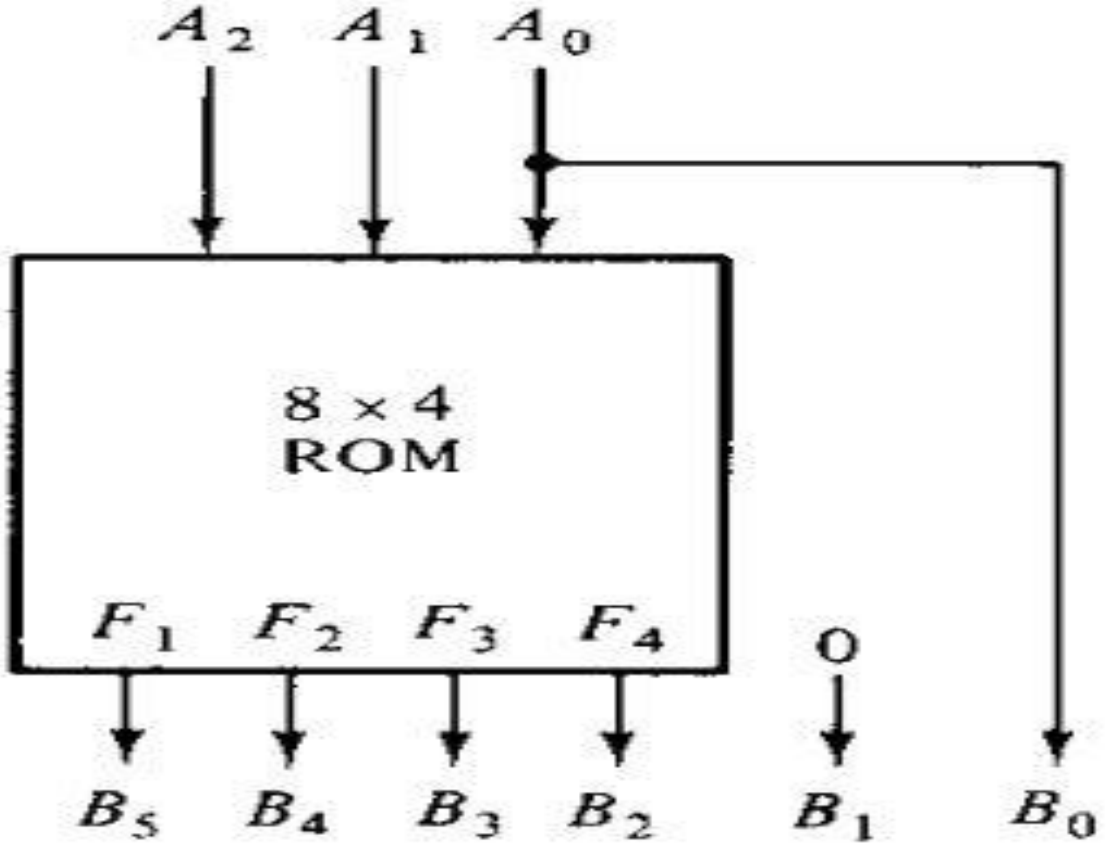
The minimum-size ROM needed must have three inputs and four outputs. Three inputs specify eight words, so the ROM size must be 8 x 4

So functional truth table becomes,



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

Combinational Logic Implementation



(a) Block diagram

Types of ROM

Mask 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

- Once the memory is programmed during the manufacturing process, the user cannot alter the programs.

PROM (Programmable ROM)

- A blank chip which can be programmed only once using a special device called programmer.

- Once it’s programmed its content cannot be modified or erased.

EPROM (Erasable Programmable ROM)

- Can be programmed multiple times.

- Its content can be erased by using UV (ultra violet) light.

- Exposure to the UV light will erase all contents.

EEPROM (Electrically Erasable Programmable ROM)

- Similar to EPROM but its contents can be electrically erased and re-written without having to remove it from the computer.

Programmable Logic Array (PLA)

A combinational circuit may occasionally have don't-care conditions. When implemented with a ROM, a don't care condition becomes an address input that will never occur. The words at the don't-care addresses need not be programmed and may be left in their original state (all 0's or all 1's). The result is that not all the bit patterns available in the ROM are used, which may be considered a waste of available equipment.

Programmable Logic Array or PLA is LSI component that can be used in economically as an alternative to ROM where number of don’t-care conditions is excessive.

ROM	PLA
ROM generates all the minterms as an output of decoder.	PLA does not provide full decoding of the variables
Uses decoder	Uses NAND gates
The size of the ROM is specified by the number of inputs (n) and the number of outputs (m)	The size of the PLA is specified by the number of inputs (n), the number of product terms(k), and the number of outputs (m)
No. of programmed fuses = $2^n * m$	No. of programmed fuses = $2n * k + k * m + m$

Block Diagram of PLA

A block diagram of the PLA is shown in Fig. below. It consists of n inputs, m outputs, k product terms, and m sum terms. The product terms constitute a group of k AND gates and the sum terms constitute a group of m OR gates. Fuses are inserted between all n inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates.

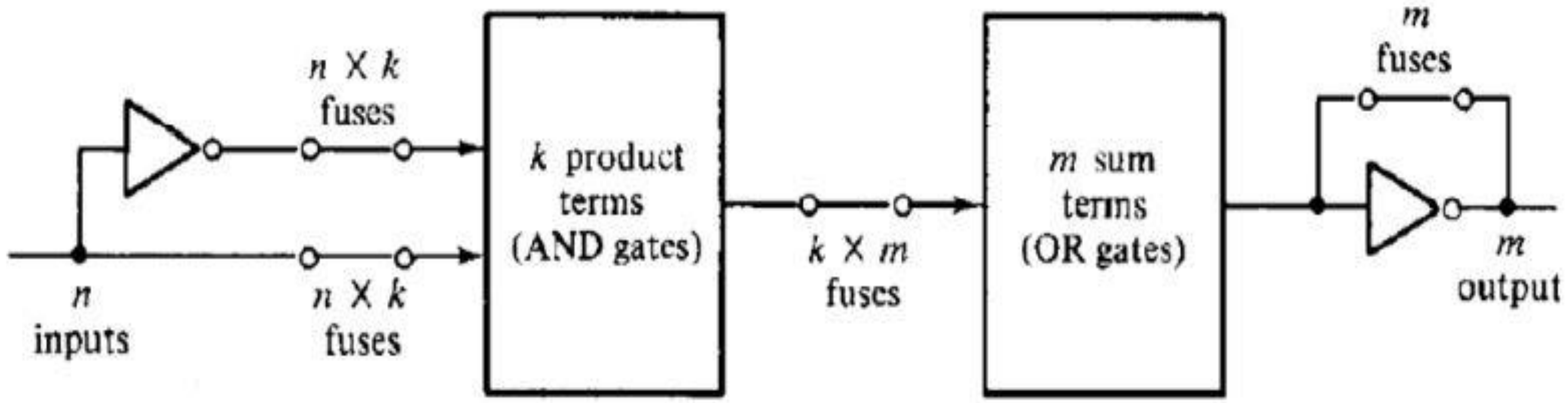


Fig: PLA block diagram

PLA program table and Boolean function Implementation

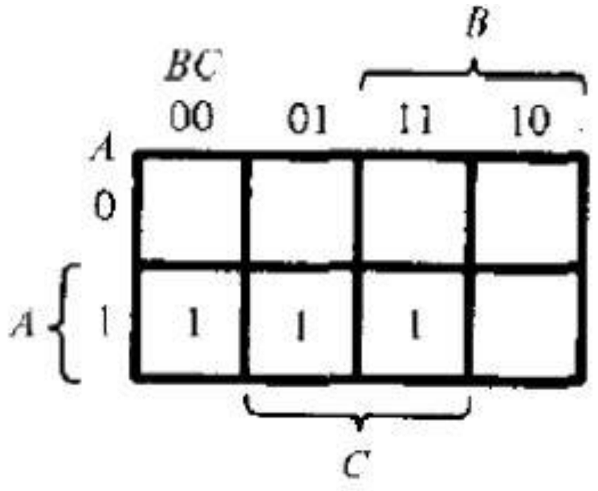
The use of a PLA must be considered for combinational circuits that have a large number of inputs and outputs. It is superior to a ROM for circuits that have a large number of don't-care conditions. Let me explain the example to demonstrate how PLA is programmed.

Consider a truth table of the combinational circuit:

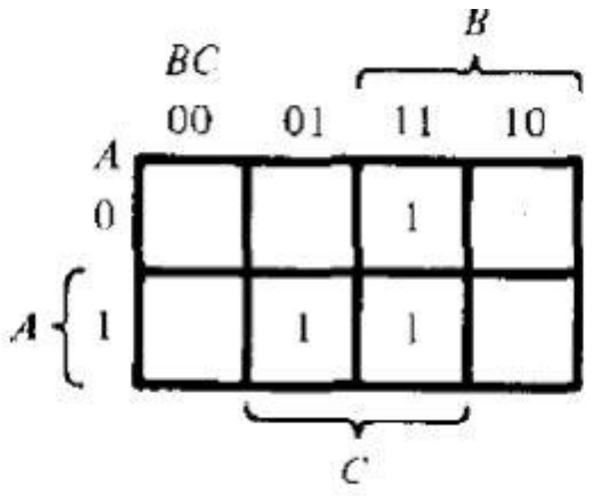
A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

PLA program table and Boolean function Implementation

PLA implements the functions in their sum of products form Each product term in the expression requires an AND gate. It is necessary to simplify the function to a minimum number of product terms in order to minimize the number of AND gates used. The simplified functions in sum of products are obtained from the following maps.



$F_1 = AB' + AC$



$F_2 = AC + BC$

There are three distinct product terms in this combinational circuit: AB’, AC and BC. The circuit has three inputs and two outputs

PLA program table and Boolean function Implementation

The PLA can be drawn to implement this combinational circuit

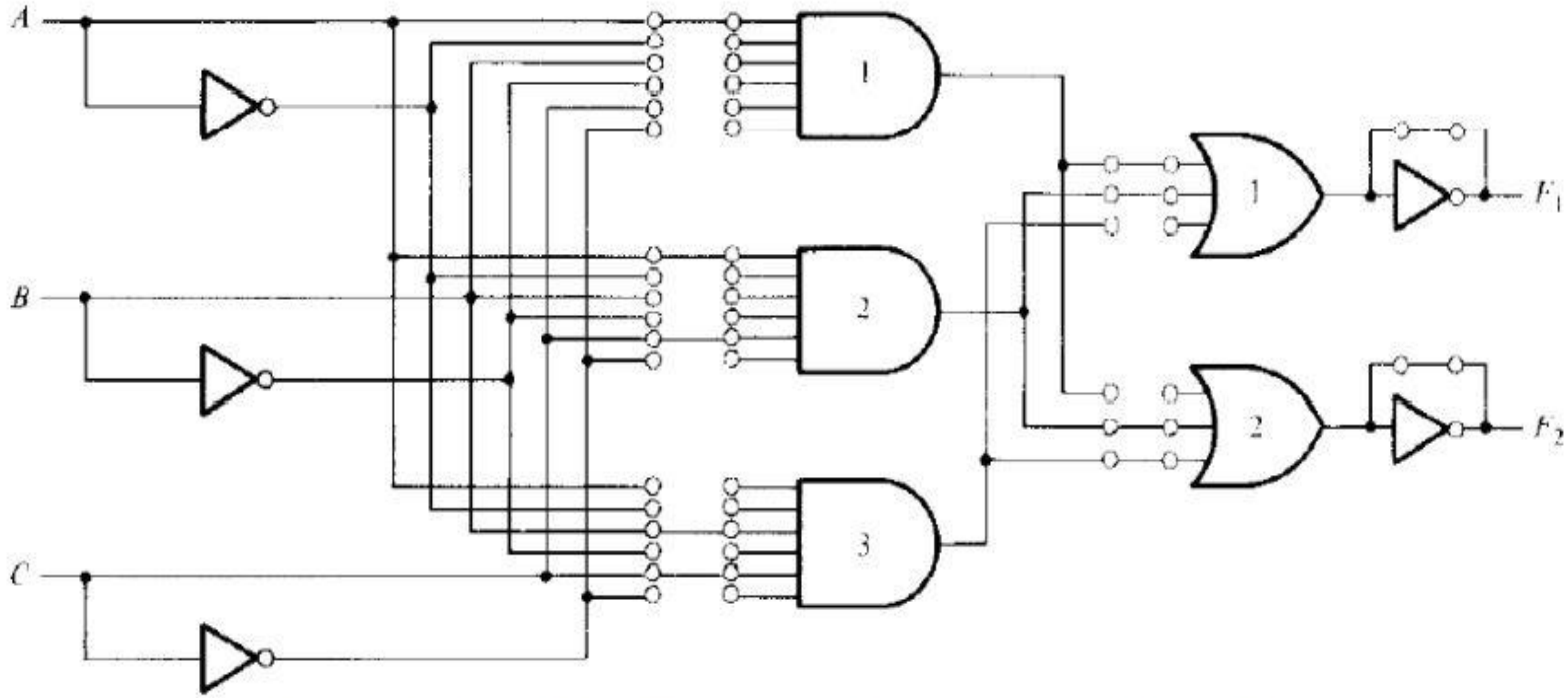


Fig: PLA with 3 inputs, 3 product terms, and 2 outputs

PLA program table

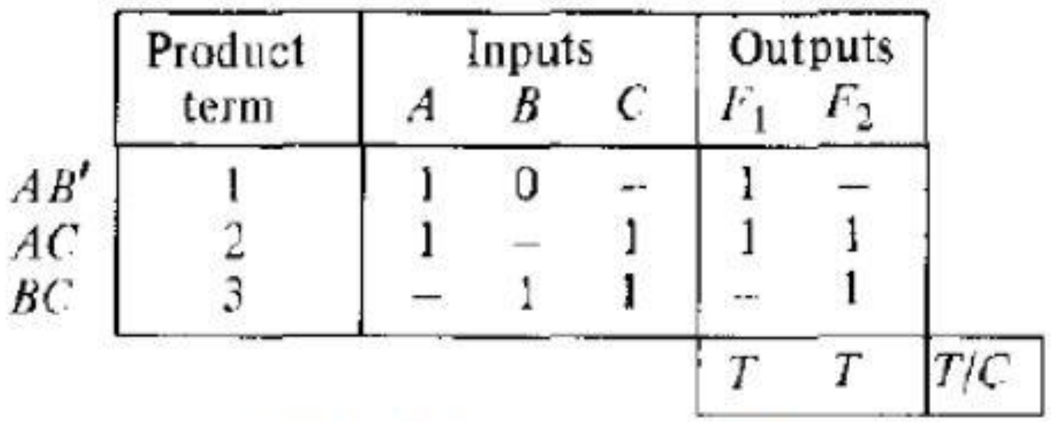
A typical PLA program table consists of three columns.

First column: lists the product terms numerically.

Second column: specifies the required paths between inputs and AND gates.

Third column: specifies the paths between the AND gates and the OR gates.

Under each output variable, we write a T (for true) if the output inverter is to be bypassed, and C (for complement) if the function is to be complemented with the output inverter.

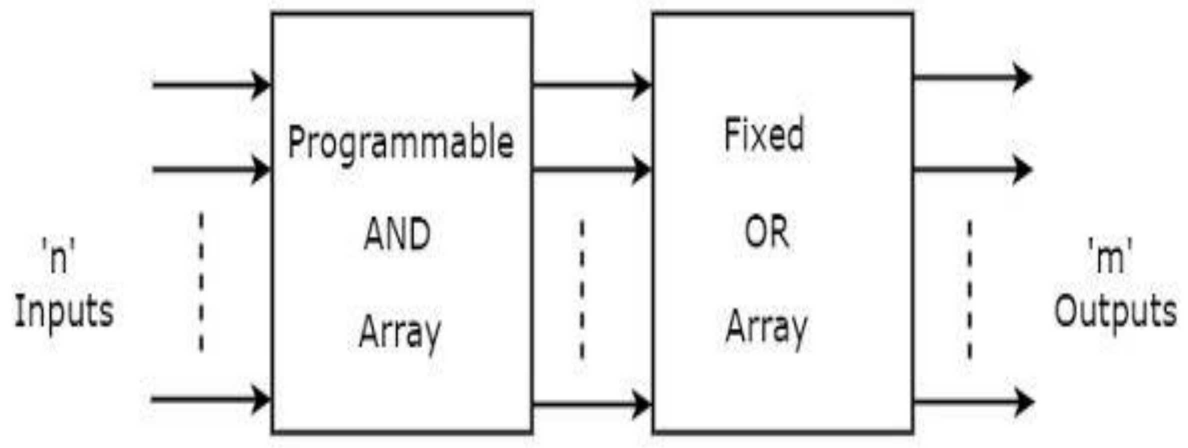


	Product term	Inputs			Outputs		
		A	B	C	F_1	F_2	
AB'	1	1	0	–	1	–	
AC	2	1	–	1	1	1	
BC	3	–	1	1	–	1	
					T	T	T/C

Table: PLA program table

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

Q. Implement the following Boolean function using PAL

A=XY+XZ′

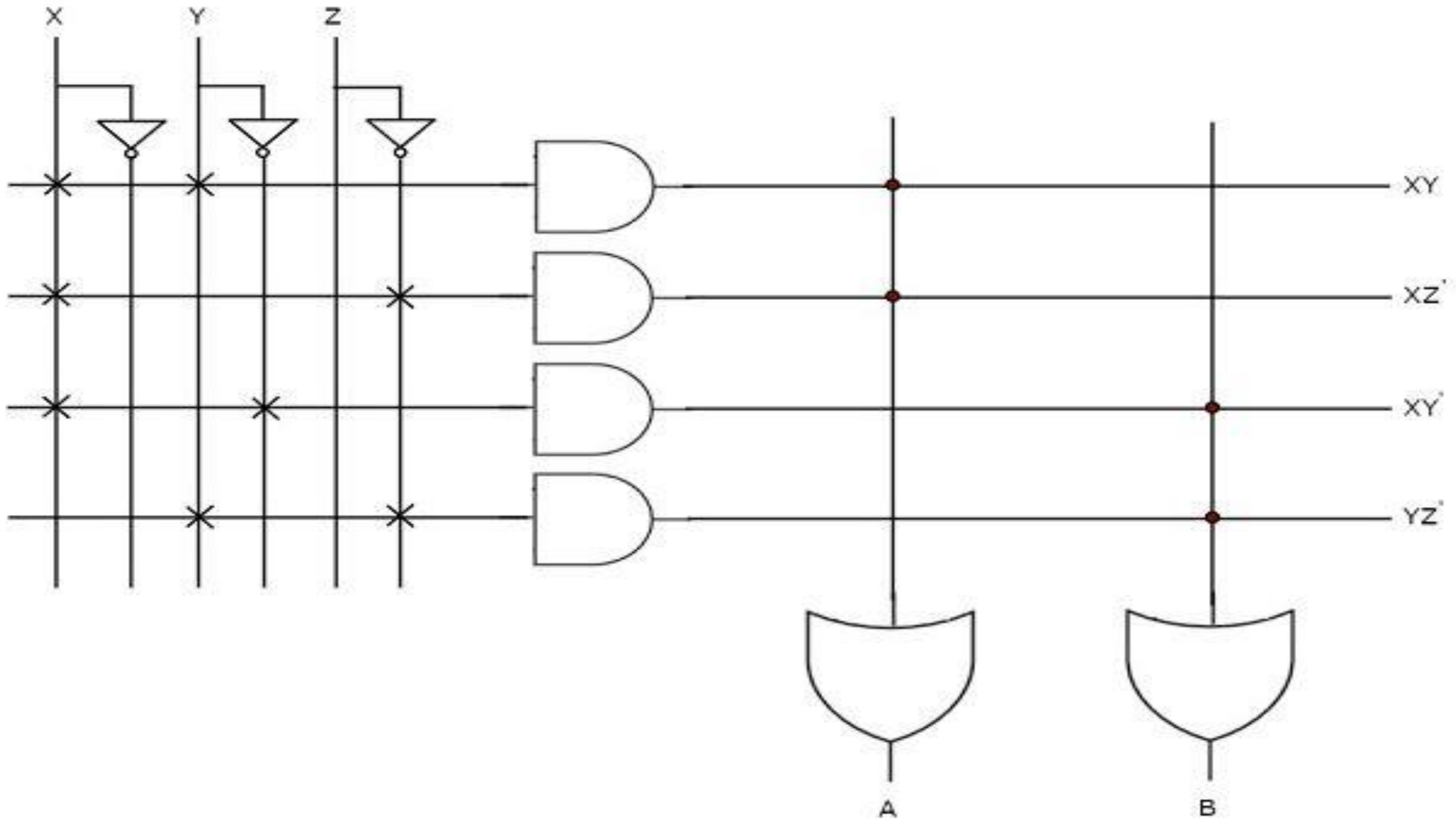
B=XY′+YZ′

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. The corresponding PAL is shown in the following figure.

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. The symbol ‘X’ is used for programmable connections.

Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each OR gate. So that the OR gates produce the respective Boolean functions. The symbol ‘.’ is used for fixed connections.

Example



Difference Between PLA and PAL

PLA	PAL
PLA speed is lower than PAL.	While PAL’s speed is higher than PLA.
The complexity of PLA is high.	While PAL’s complexity is less.
The cost of PLA is also high.	While the cost of PAL is low.
Programmable Logic Array is less available.	While Programmable Array Logic is more available than Programmable Logic Array.
It is less used than PAL.	While it is more used than PLA.

Assignment

Q. Design a combinational circuit that generates 9’s complement of a BCD number. [TU 2077]

Q. Design a combinational circuit that generates 10’s complement of a BCD number.

Q. Implement the following function using PLA and PAL

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

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

Q. Implement the following function using Decoder, Multiplexer, PLA and PAL[very important]

$F=\sum(1,2,8,12,13)$