Digital Logic Design

UNIT-3

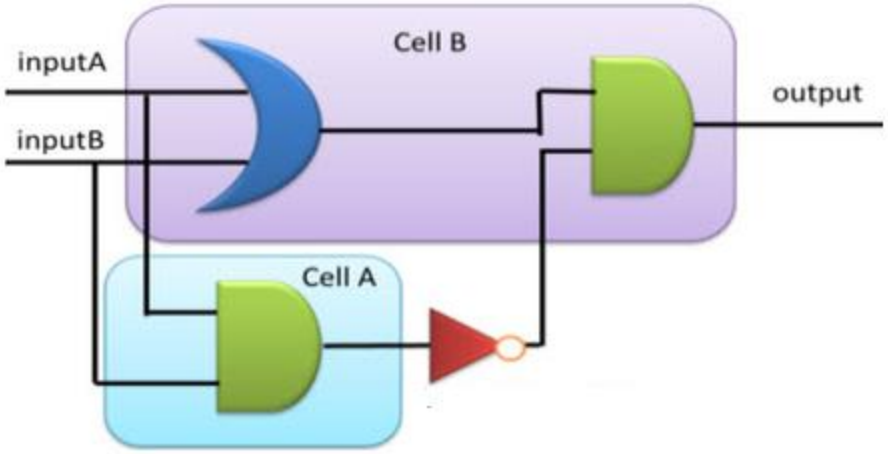
Combinational Circuits

CONTENTS

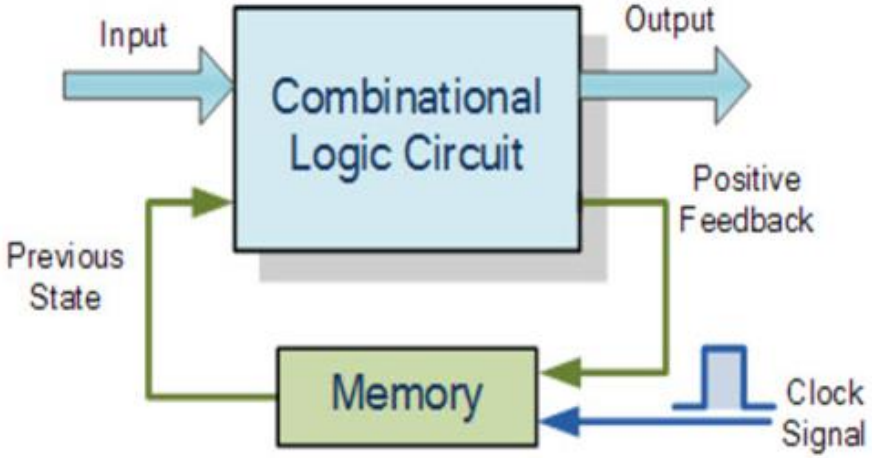
- Analysis of combinational circuits
- Design Procedure – Binary Adders, Subtractors, BCD Adder, multiplier, comparator, decoders, encoders, multiplexers, demultiplexers, code converters.
- Basic PLD's-ROM, PROM, PLA, PAL Realizations.

Digital Logic Circuits

- A digital logic circuit is defined as the one in which voltages are assumed to be having a finite number of distinct value. These are the basic circuits used in most of the digital electronic devices like computers, calculators, mobile phones.
- Digital logic circuits are often known as switching circuits, because in digital circuits the voltage levels are assumed to be switched from one value to another value instantaneously. These circuits are termed as logic circuits, as their operation obeys a definite set of logic rules.
- Types of digital logic circuits are *combinational logic circuits* and *sequential logic circuits*.



combinational logic circuit



sequential logic circuit

Analysis of combinational circuits

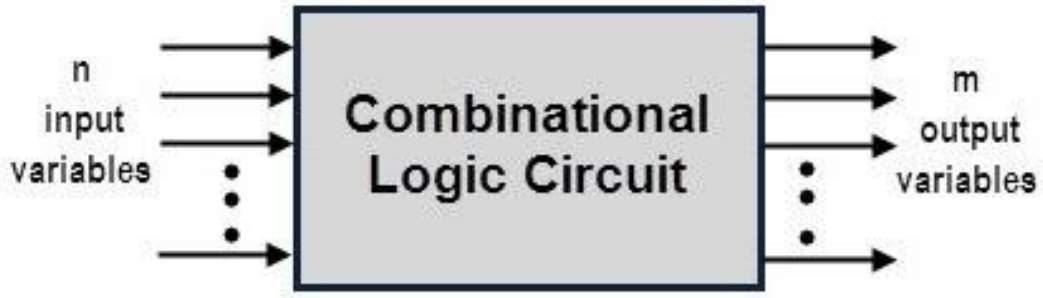
The analysis of combinational circuits can be done as:

- Definition and Block diagram of combinational circuits
- Types of combinational circuits
- Design procedure of combinational circuits

What are combinational logic circuits?

- Combinational circuits are a basic collection of logic gates.
- The output of a combinational circuit depends on its present inputs only. Combinational circuits perform a specific information processing operation fully specified logically by set of Boolean functions.
- A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and generate signals to the outputs.
- For 'n' input variables, there are '2^n' possible combinations of binary input variables. For each input combination, there is only one possible output combination.

Block diagram of a combinational logic circuit



Different types of combinational logic circuits

There are three main types of combinational logic circuits:

- Arithmetic and logical combinational circuits – Adders, Subtractors, Multipliers, Comparators.
- Data handling combinational circuits – Multiplexers, Demultiplexers, priority encoders, decoders.
- Code converting combinational circuits – Binary to Gray, Gray to Binary, Binary to Excess 3,seven-segment, etc.

Design Procedure

The design procedure for combinational logic circuits starts with the problem specification and comprises the following steps:

- Determine required number of inputs and outputs from the specifications. Assign letter symbols for input and output variables.
- Derive the truth table for each of the outputs based on their relationships to the input.
- Simplify the Boolean expression for each output. Use Karnaugh Maps or Boolean algebra.
- Draw a logic diagram that represents the simplified Boolean expression. Verify the design by analyzing or simulating the circuit.

Example: Is input greater than or equal to 5?

Specification: Design a circuit that has a 3-bit binary input and a single output (Z) specified as follows:

- Z = 0, when the input is less than 5_{10}
- Z = 1, otherwise

Step1: Determine the inputs and Outputs

- Label the inputs (3 bits) as A, B, C
  - A is the most significant bit
  - C is the least significant bit
- The output (1 bit) is Z
  - Z = 1 -> 101_2, 110_2, 111_2
  - Z = 0 -> other inputs

Step2: Derive the Truth Table

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

Step 3: Simplify the Boolean Expression

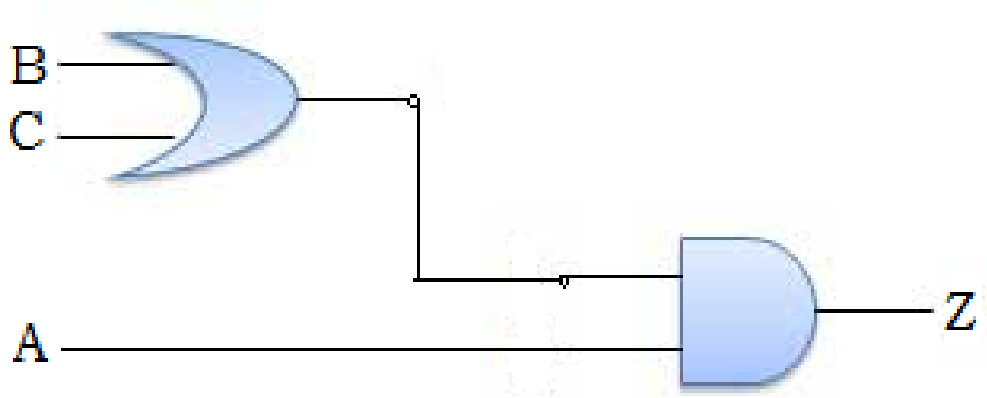
From the truth table, one of the following two methods to obtain the simplified Boolean expression.

- Use Karnaugh Map to minimize the logic or
- From the truth table, get the Canonical Sum of Products Boolean expression.
  Z = A * ~B * C + A * B * ~C + A * B * C
- Use Boolean Algebra to simplify the Boolean expression to: **Z = (B + C) * A**

Step 4: Draw the logic diagram

Draw a logic diagram that represents the simplified Boolean expression. Verify the design by analyzing or simulating the circuit.

Boolean Expression: Z = (B + C) * A



Arithmetic Circuits

- Combinational Arithmetic Circuits perform arithmetic functions like Addition, Subtraction and Multiplication i.e., the logic circuits which are used for performing the digital arithmetic operations such as addition, subtraction, multiplication and division are called ‘arithmetic circuits’.

- They are structured or array combinational circuits. For example, an n-bit adder is made up of a 1-dimensional array of 1-bit full adders.

Adders

- An **adder** is a device that will add together two bits and give the result as the output.
- There are two kinds of **adders** - **half adders** and **full adders**.
  - A **half adder** just adds two bits together and gives a two-bit output.
  - A **full adder** adds two inputs and a carried input from another **adder**, and also gives a two-bit output.

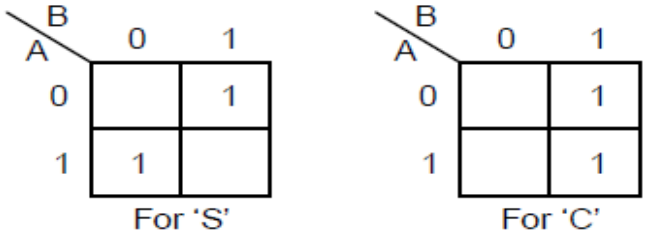
Half adder:

A	0	0	1	1
B	+0	+1	+0	+1
	0	1	1	0 (carry 1)

Truth Table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

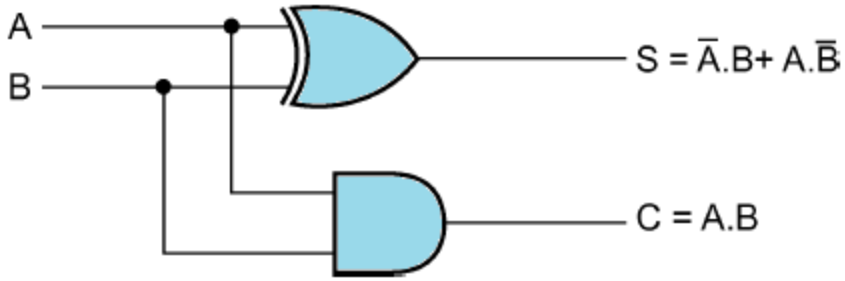
Logic Equations:



$S = AB' + A'B$

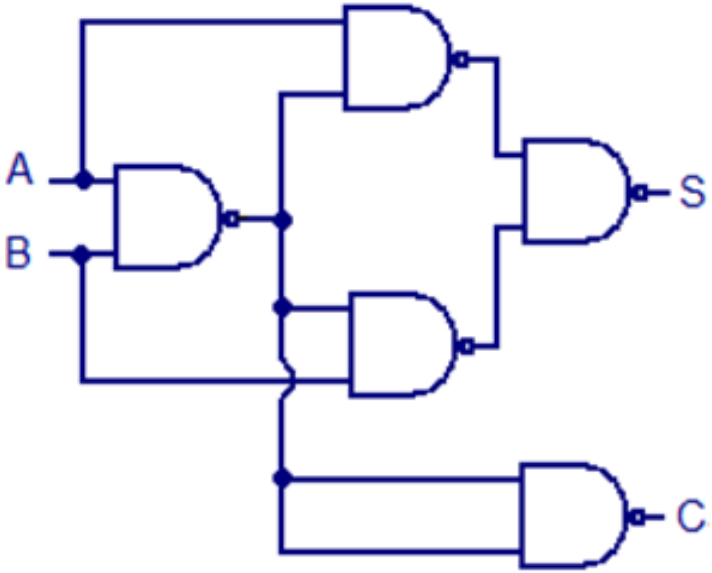
$= A \oplus B.$

$C = AB$

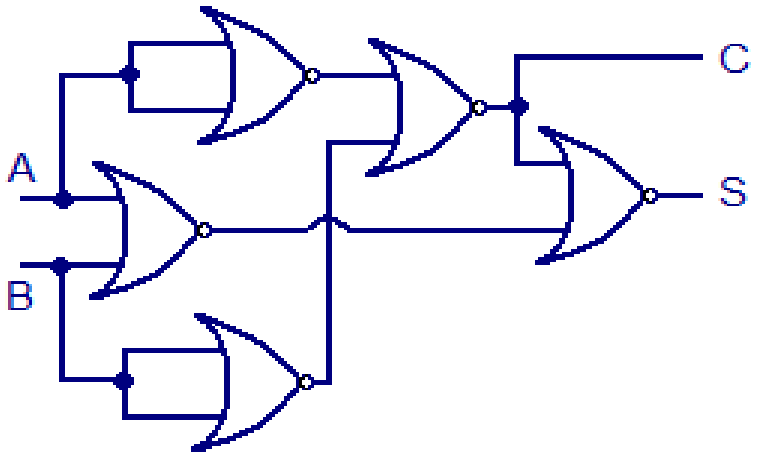


Logic Circuit:

Half adder using Universal gates



Half adder using NAND logic



Half adder using NOR logic

$$\text{SUM, } S = \overline{\overline{A.\overline{AB}} \; . \; \overline{B.\overline{AB}}}$$

$$\text{SUM, } S = A\bar{B} + \bar{A}B$$

$$\text{Carry, } C = AB = \overline{\overline{AB}}$$

$$\text{SUM, } S = \overline{\overline{A+B} + \overline{A+B}}$$

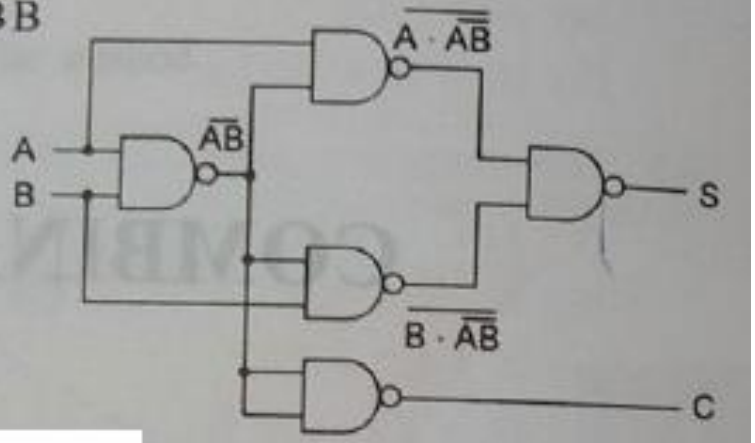
$$\text{SUM, } S = A\bar{B} + \bar{A}B$$

$$\text{Carry, } C = \overline{\overline{A}+\overline{B}}$$

NAND logic

$$S = A\overline{B} + \overline{A}B = A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B}$$
$$= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$$
$$= A \cdot \overline{AB} + B \cdot \overline{AB}$$
$$= \overline{A \cdot \overline{AB} \cdot B \cdot \overline{AB}}$$

$$C = AB = \overline{\overline{AB}}$$

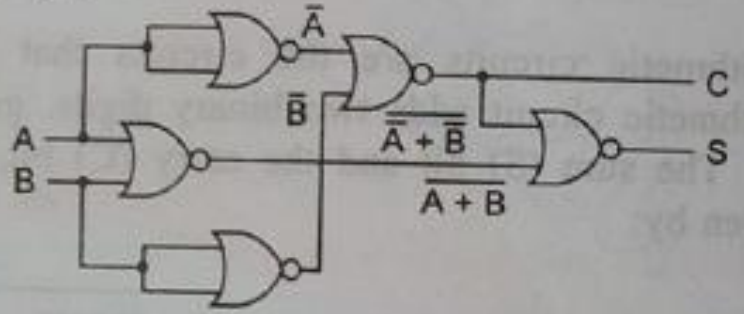


Half-adder using NAND logic.

NOR logic

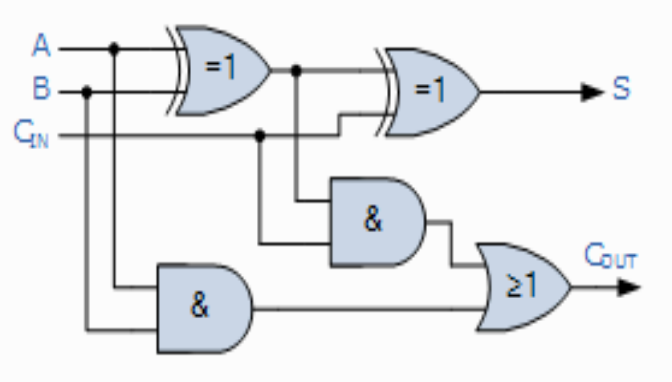
$$S = A\overline{B} + \overline{A}B = A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B}$$
$$= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$$
$$= (A + B)(\overline{A} + \overline{B})$$
$$= \overline{\overline{A + B} + \overline{\overline{A} \cdot \overline{B}}}$$

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



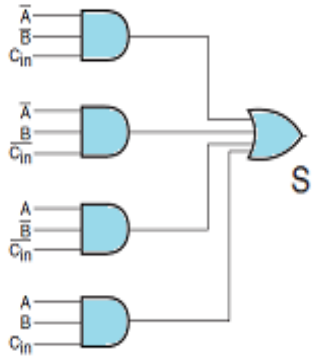
Half-adder using NOR logic.

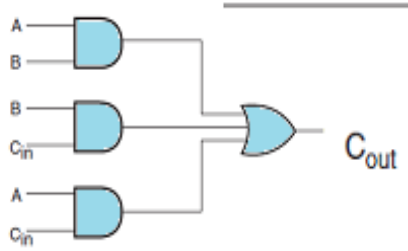
Full adder:



Logic Symbol

A	B	C_{in}	SUM (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



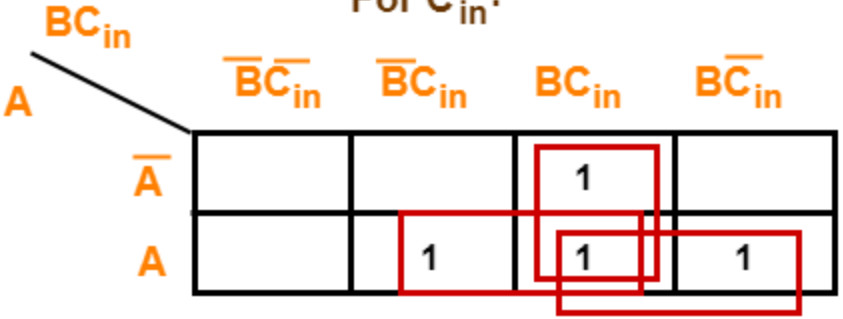


Logic Equation:

$$S = A \oplus B \oplus Ci$$

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

For C_{in}:

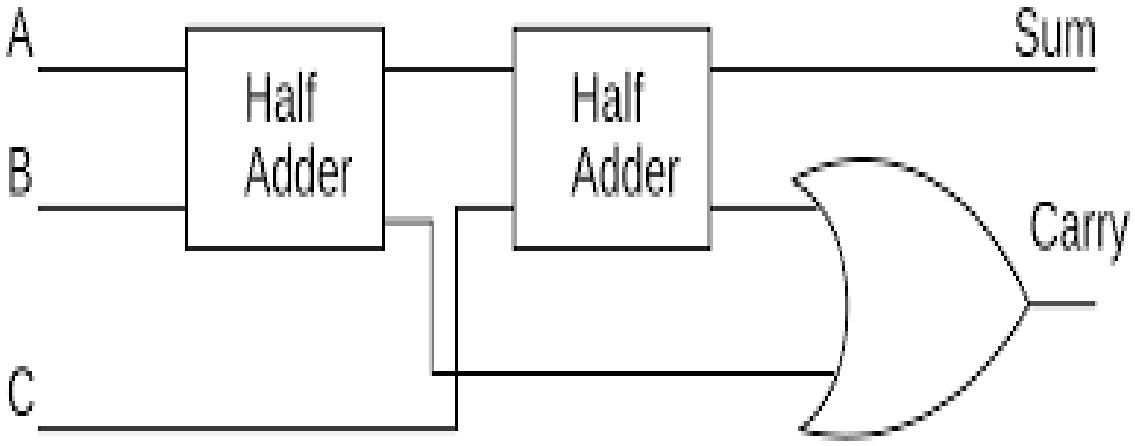


$$C_{out} = AB + BC_{in} + C_{in}A$$

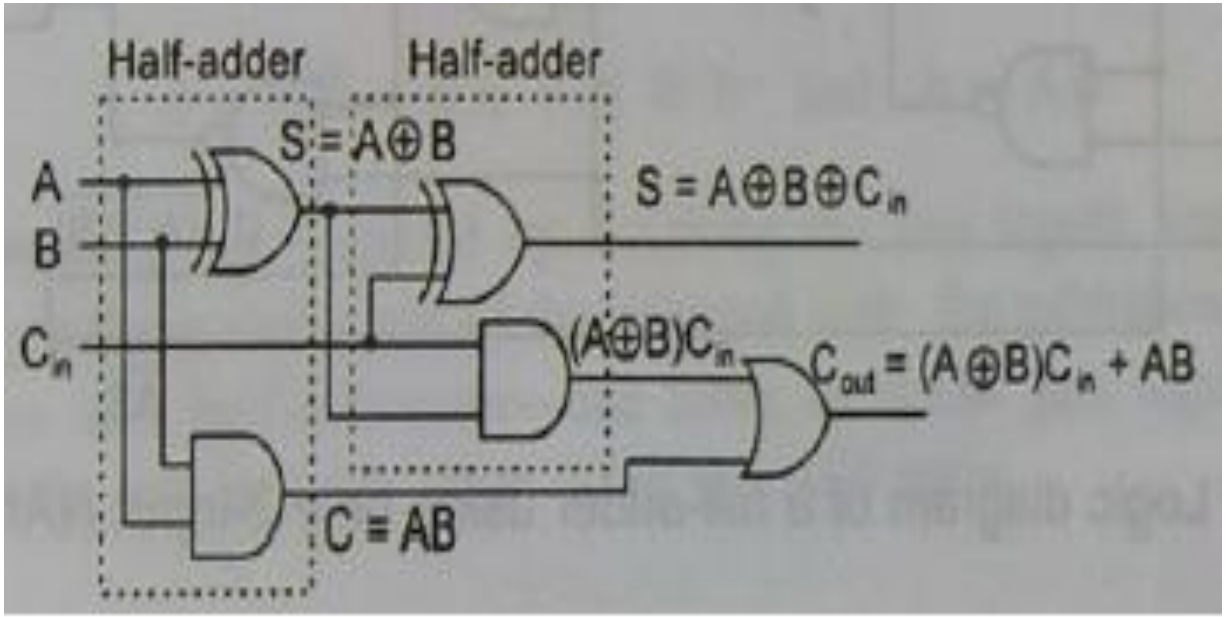
For S:

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

Full Adder using two Half Adders:



$$
\begin{aligned}
S &= ABC + AB'C' + A'BC' + A'B'C \\
&= ABC + A'B'C + AB'C' + A'BC' \\
&= C\,(AB + A'B') + C'\,(AB' + A'B) \\
&= C\,(AB' + A'B)' + C'\,(AB' + A'B) \\
&= (A \oplus B) \oplus C
\end{aligned}
$$



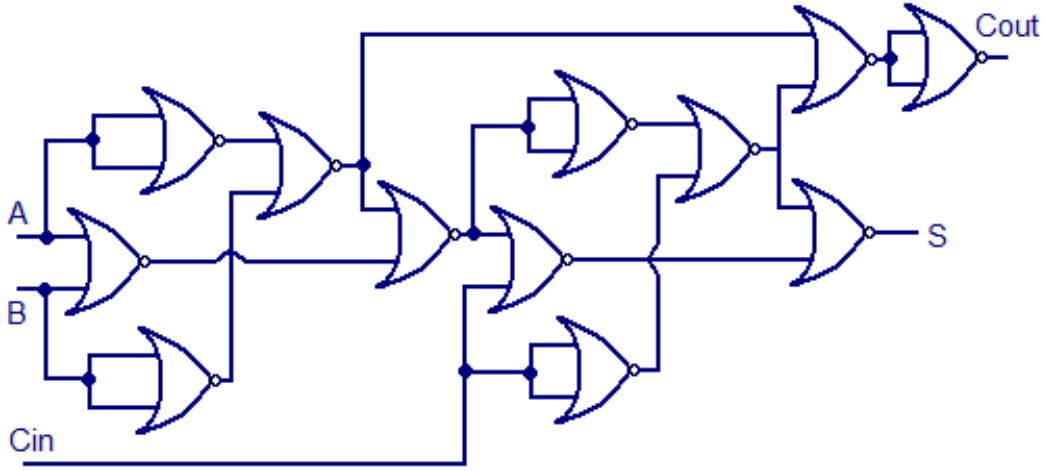
$$
\begin{aligned}
C_0 &= AB + AC + BC \\
&= AB + C\,(A + B) \\
&= AB + C\,(A + B)\,(A + A')\,(B + B') \\
&= AB + C\,[AB + AB' + A'B] \\
&= AB + ABC + C\,(AB' + A'B) \\
&= AB\,(1 + C) + C\,(A \oplus B) \\
&= AB + C\,(A \oplus B)
\end{aligned}
$$

Full adder using Universal gates

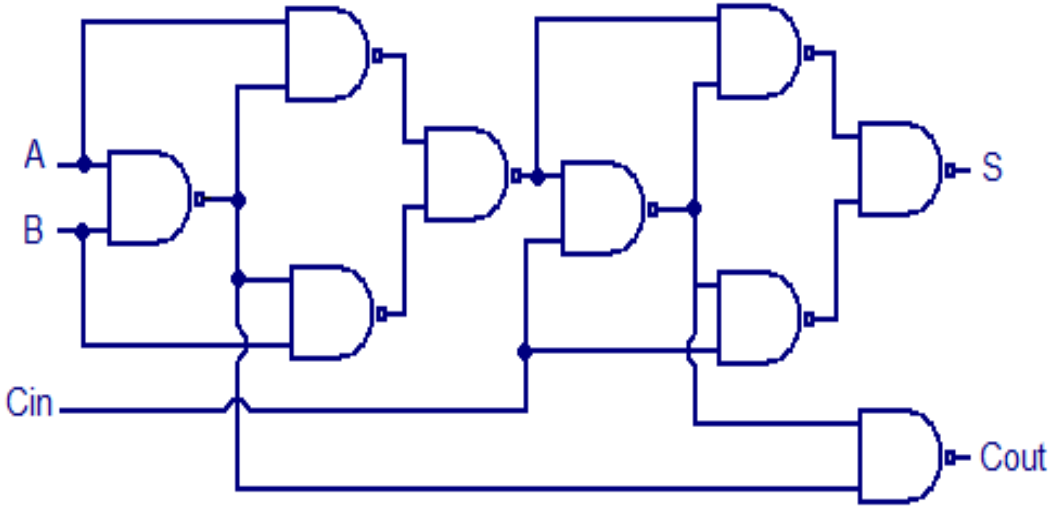
$$A \oplus B = X = \overline{\overline{(A + B)} + \overline{A} + \overline{B}}$$

$$S = A \oplus B \oplus C_{in} = X \oplus C_{in} = \overline{\overline{X + C_{in}} + \overline{\overline{X} + C_{in}}}$$

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



Full adder using NOR logic



Full adder using NAND logic

$$A \oplus B = \overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}} = X. \text{ Then}$$

$$S = A \oplus B \oplus C_{in} = \overline{X \cdot \overline{XC_{in}} \cdot C_{in} \cdot \overline{XC_{in}}} = X \oplus C_{in}$$

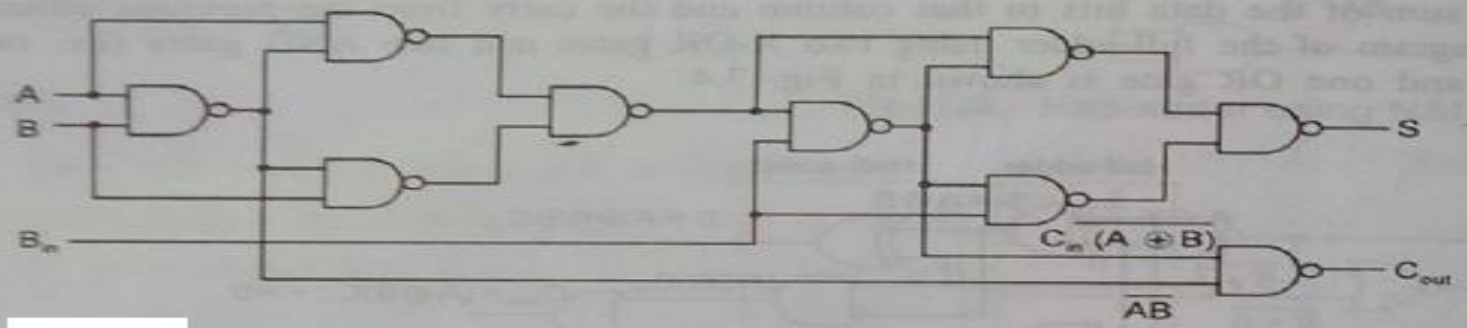
$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$

NAND logic

Let $A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}} = X$. Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{X \cdot \overline{XC_{in}}} \cdot C_{in} \cdot \overline{XC_{in}}} = X \oplus C_{in}$$

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



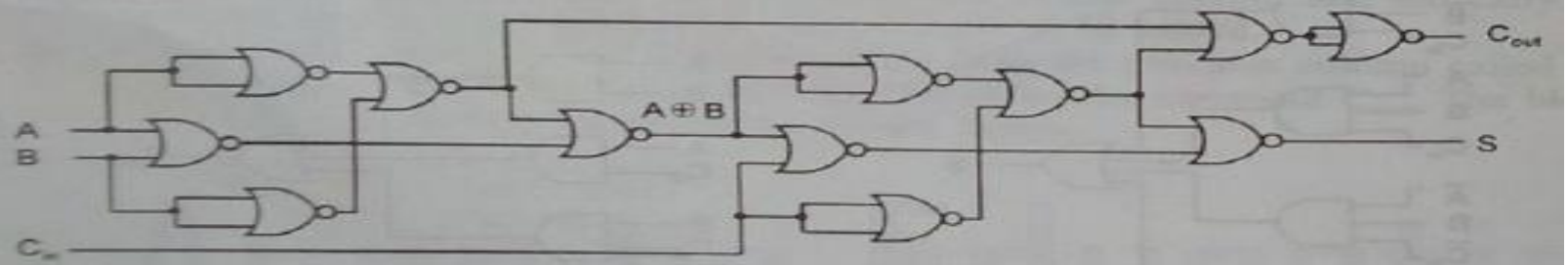
Logic diagram of a full-adder using only 2-input NAND gates.

NOR logic

Let $A \oplus B = X = \overline{\overline{(A + B)} + \overline{A} + \overline{B}}$

$$S = A \oplus B \oplus C_{in} = X \oplus C_{in} = \overline{\overline{X + C_{in}} + \overline{X} + \overline{C_{in}}}$$

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



Logic diagram of a full-adder using only 2-input NOR gates.

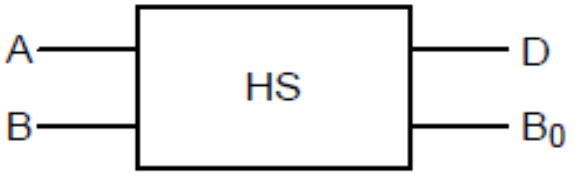
Subtractors

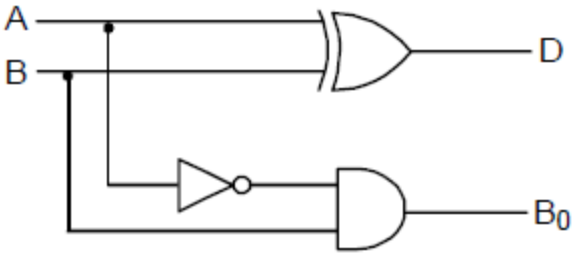
The logic circuits used for binary subtraction, are known as **binary subtractors**. There are two kinds of subtractors.

- A **half subtractor** is a combinational circuit which is used to perform the subtraction of two bits.
- A **full subtractor** a combinational circuit that performs the subtraction of three binary digits.

Half subtractor:

0 – 0 = 0
0 – 1 = 1, borrow 1
1 – 0 = 1
1 – 1 = 0

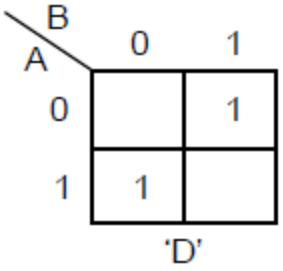




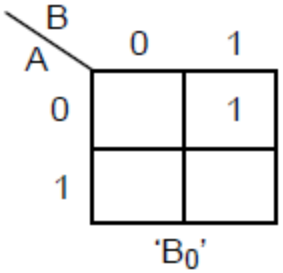
Truth Table:

Inputs		*Outputs*	
A	*B*	*D*	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logic Equations:

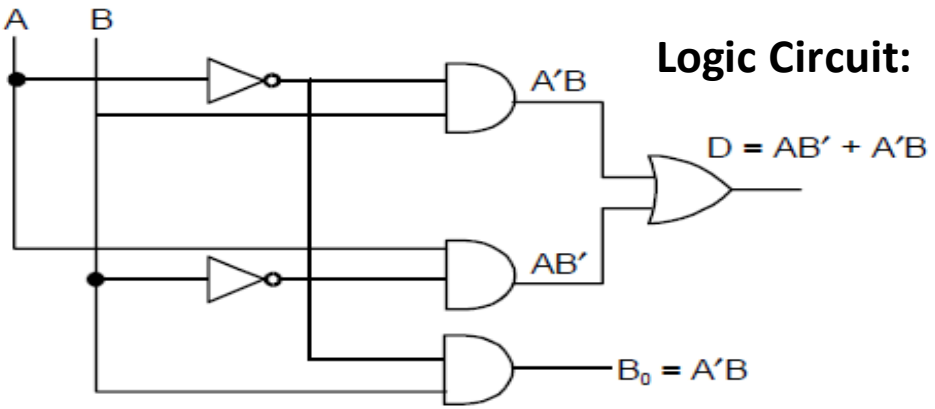


$D = AB' + A'B$
$= A \oplus B$



$B_0 = A'B$

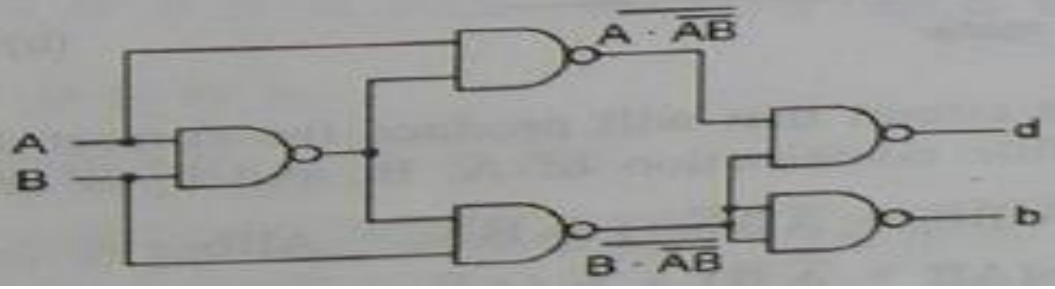
Logic Circuit:



Half subtractors using Universal gates

$$d = A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

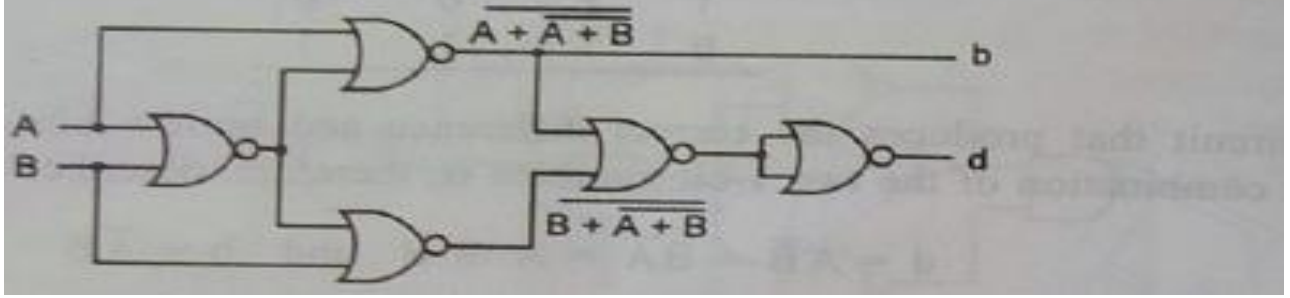
$$b = \overline{A}B = B(\overline{A} + \overline{B}) = B(\overline{AB}) = \overline{B \cdot \overline{AB}}$$



$$d = A \oplus B = A\overline{B} + \overline{A}B = A\overline{B} + B\overline{B} + \overline{A}B + A\overline{A}$$

$$= \overline{B}(A + B) + \overline{A}(A + B) = \overline{\overline{B} + \overline{A + B}} + \overline{\overline{A} + \overline{A + B}}$$

$$b = \overline{A}B = \overline{A}(A + B) = \overline{\overline{A}(A + B)} = \overline{A + (\overline{A + B})}$$



Full subtractor:

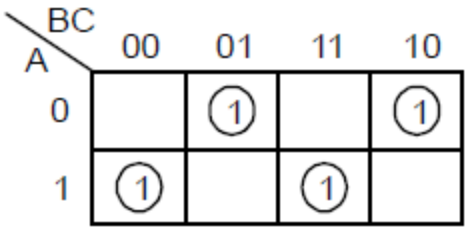
Logic Symbol

Truth Table

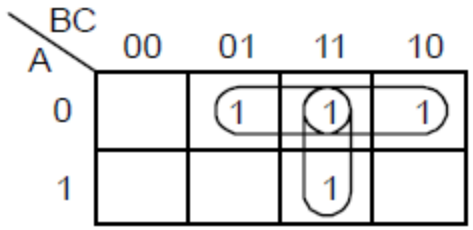
Inputs			*Output*	
A	*B*	*C*	*D*	B_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Logic Equation:

$$D = ABC + AB'C' + A'BC' + A'B'C$$
$$B_0 = A'B + A'C + BC$$



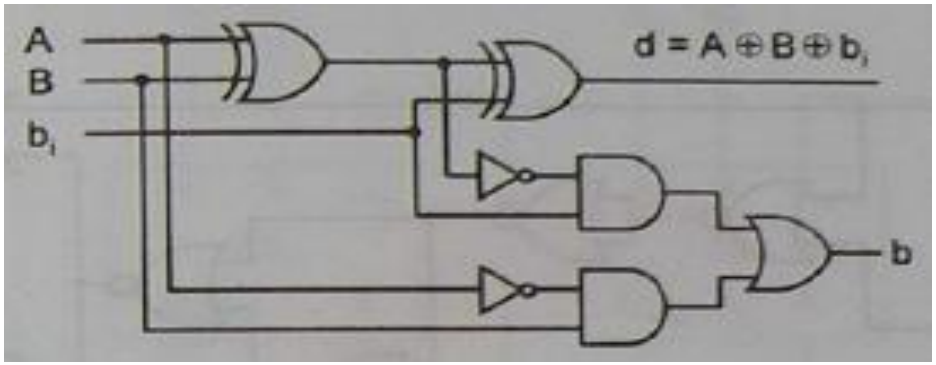
For 'D'



For 'B_0'

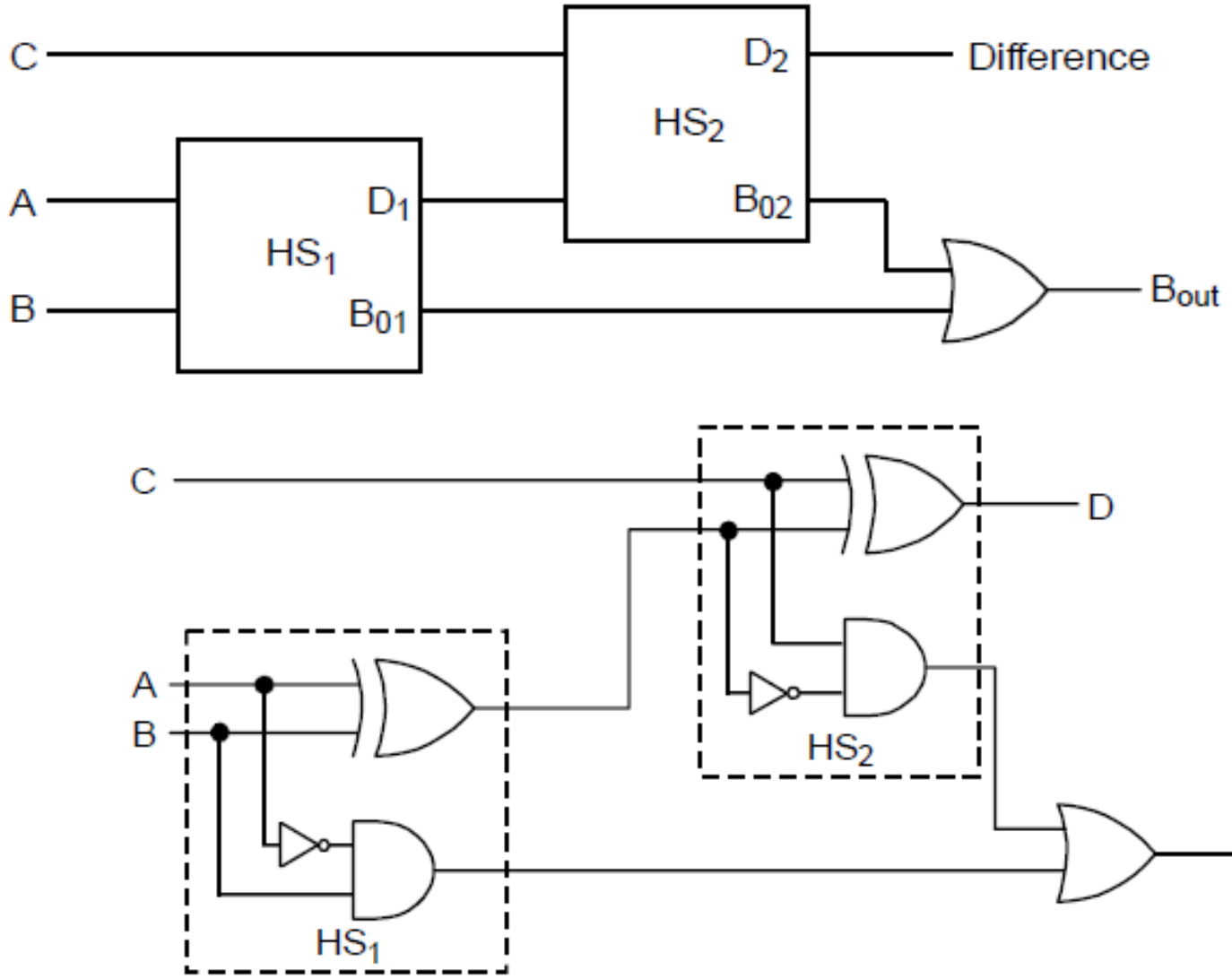
$$\begin{aligned} D &= ABC + AB'C' + A'BC' + A'B'C \\ &= ABC + A'B'C + AB'C' + A'BC' \\ &= C\,(AB + A'B') + C'\,(AB' + A'B) \\ &= C\,(AB' + A'B)' + C'\,(AB' + A'B) \\ &= C\,(A \oplus B)' + C'\,(A \oplus B) \\ &= (A \oplus B) \oplus C \end{aligned}$$

Logic diagram:

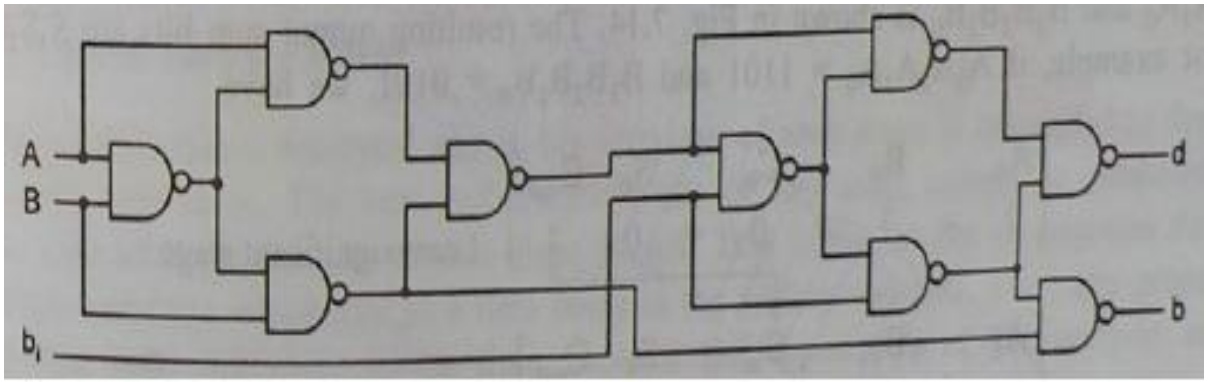


$$\begin{aligned} B_0 &= A'B + A'C + BC \\ &= A'B + C\,(A' + B) \\ &= A'B + C\,(A' + B)\,(A + A')\,(B + B') \\ &= A'B + C\,[A'B + AB + A'B'] \\ &= A'B + A'BC + C\,(AB + A'B') \\ &= A'B\,(C + 1) + C\,(A \oplus B)' \\ &= A'B + C\,(A \oplus B)' \end{aligned}$$

Full subtractors using two Half subtractors:



Full subtractor using Universal gates

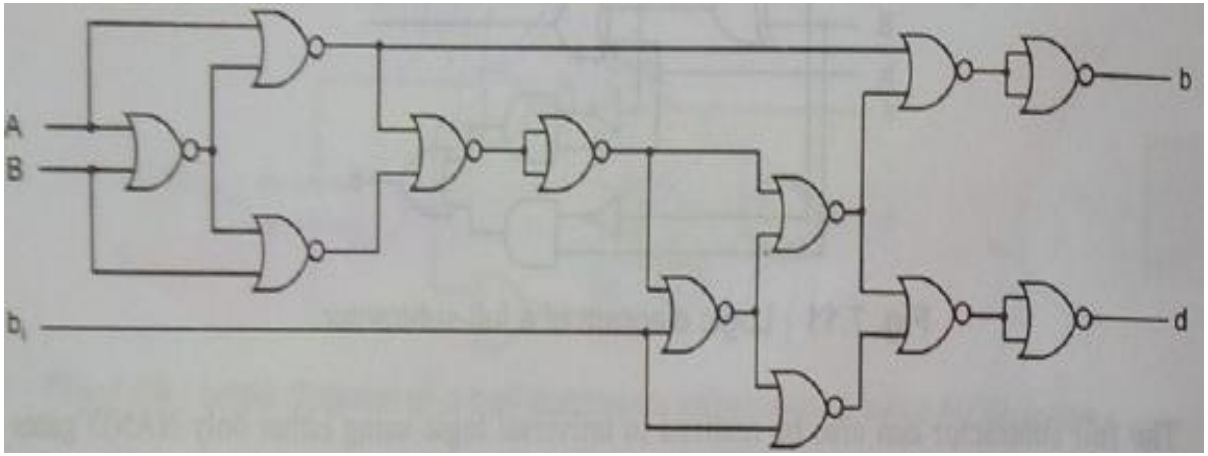


$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B)(A \oplus B)b_i} \cdot \overline{b_i(A \oplus B)b_i}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\bar{A}B + b_i(\overline{A \oplus B})}$$

$$= \overline{\overline{\bar{A}B} \cdot b_i\,(\overline{A \oplus B})} = \overline{B(\bar{A} + \bar{B}) \cdot b_i\,[\bar{b}_i + (\overline{A \oplus B})]}$$

$$= \overline{\overline{B \cdot \bar{A}B} \cdot b_i\,[\overline{b_i \cdot (A \oplus B)}]}$$



$$d = A \oplus B \oplus b_i$$

$$= \overline{(A \oplus B)b_i} + \overline{(A \oplus B)\bar{b}_i}$$

$$= \overline{[(A \oplus B) + (\overline{A \oplus B})\,\bar{b}_i]\,[b_i + (\overline{A \oplus B})\,\bar{b}_i]}$$

$$= \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}$$

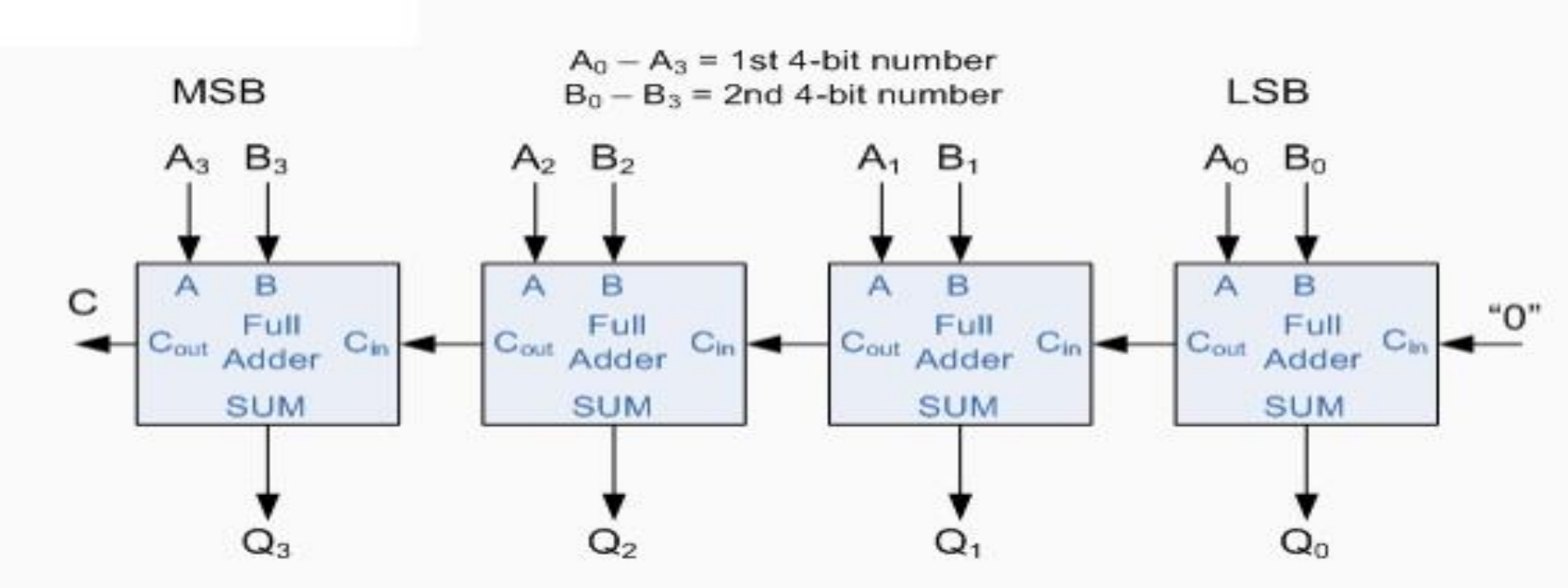
$$= (A \oplus B) + \overline{(A \oplus B) + b_i} + b_i + \overline{(A \oplus B) + b_i}$$

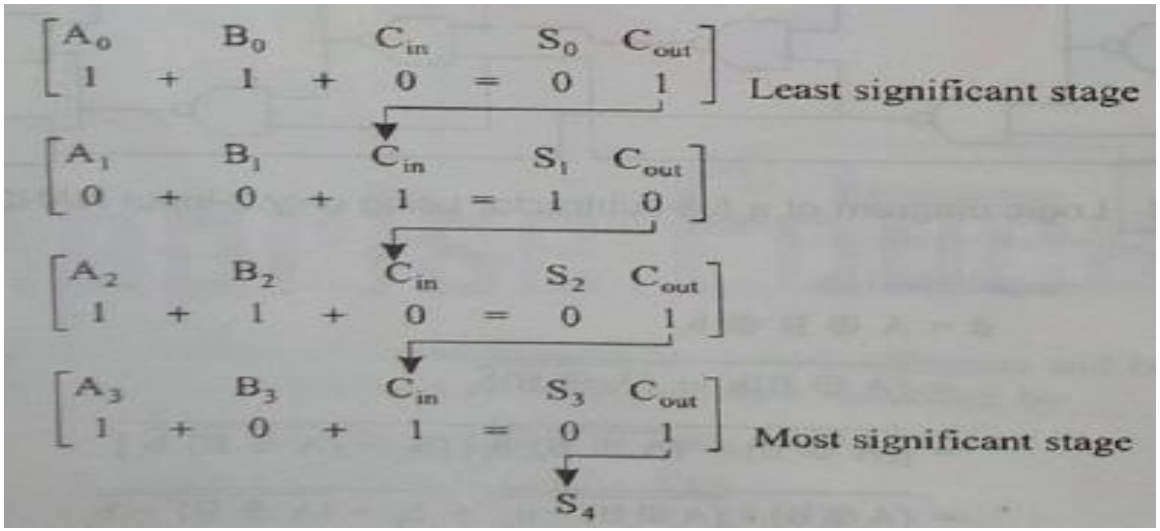
$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})\,[(A \oplus B) + b_i]$$

$$= \overline{A + (\overline{A + B})} + (A \oplus B) + (A \oplus B) + b_i$$

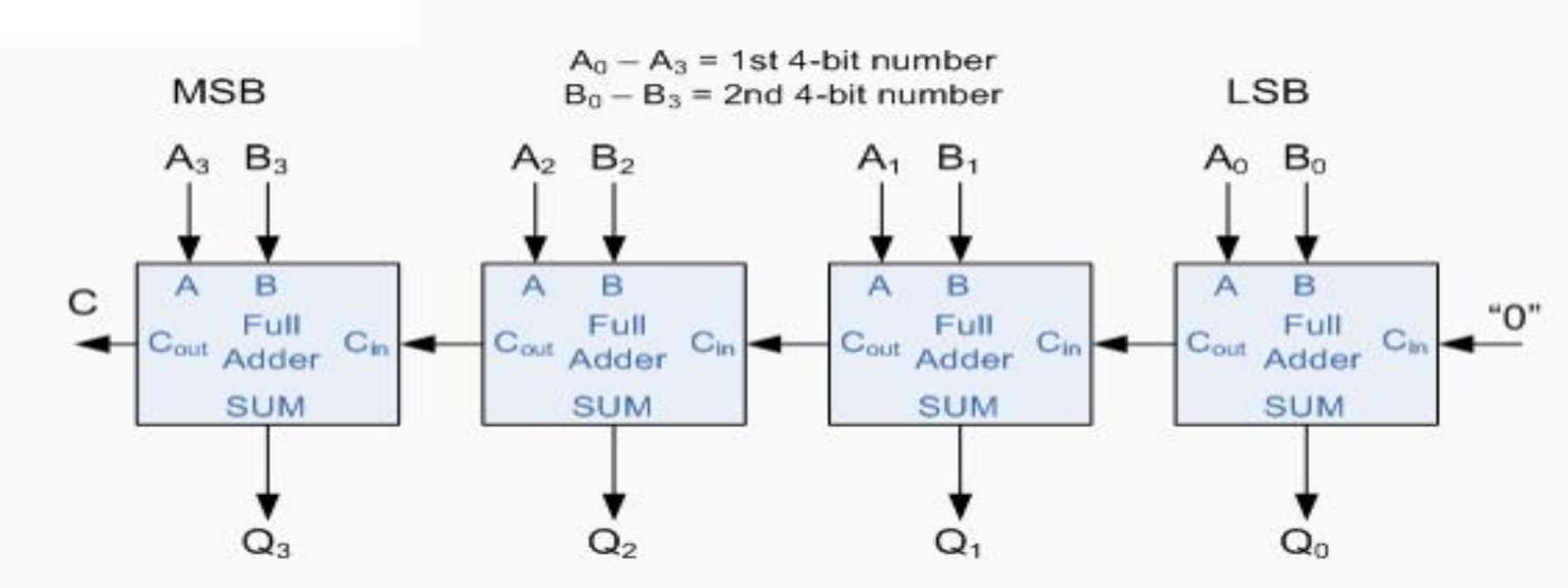
Binary Adder – 4 bit parallel adder

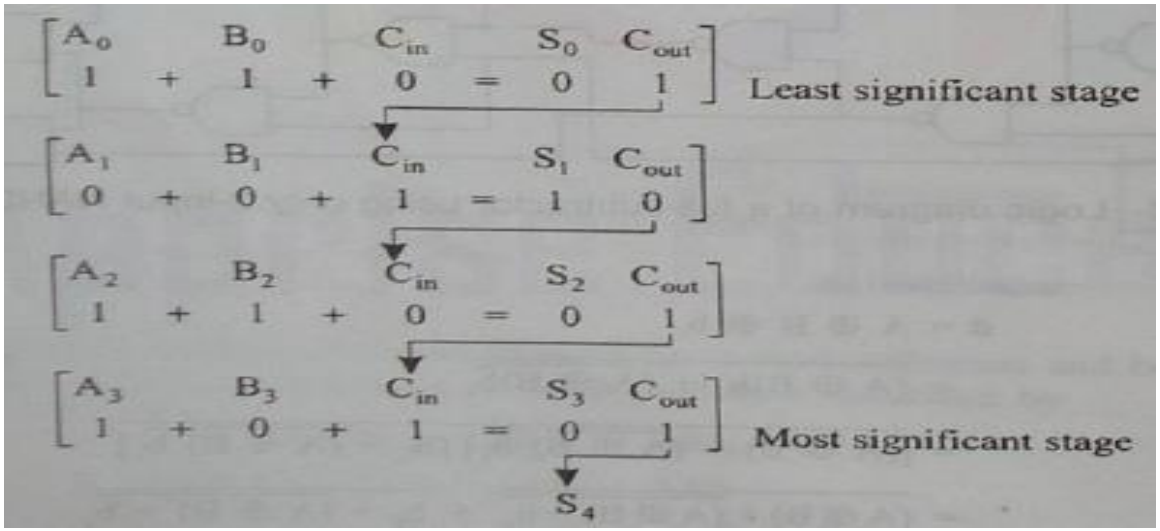




These full **adders** perform the addition of two **4-bit binary** numbers. The sum outputs are provided for each **bit** and the resultant carry (C4) is obtained from the fourth **bit**.

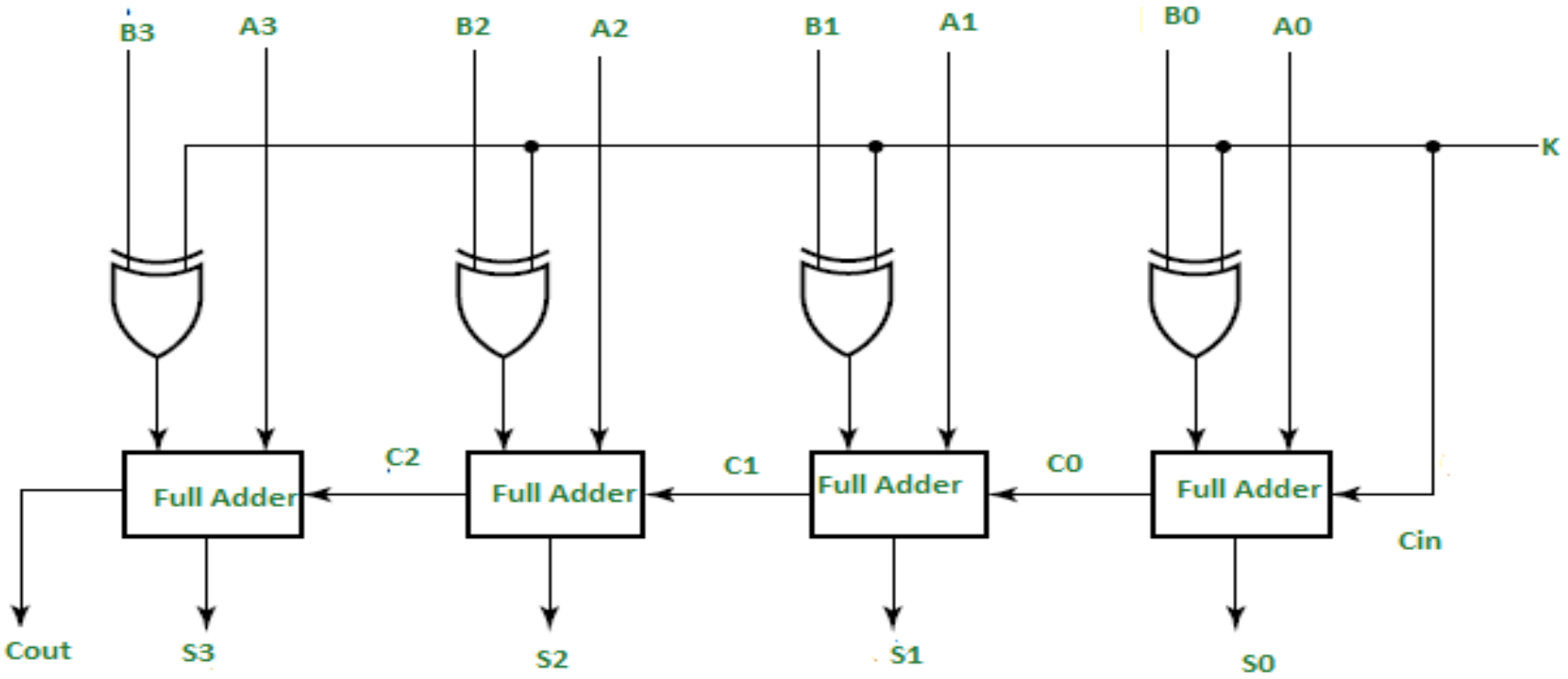
Binary Adder – 4 bit parallel adder





These full **adders** perform the addition of two **4-bit binary** numbers. The sum outputs are provided for each **bit** and the resultant carry (C4) is obtained from the fourth **bit**.

Binary Adder-Subtractor



The circuit consists of 4 full adders since we are performing operation on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation being carried out is addition or subtraction.

The first full adder has control line directly as its input(input carry C0), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the exor of B0 and K (S in fig But do not confuse it with Sum-S).

The two outputs produced are Sum/Difference (S0) and Carry (C1). If the value of K (Control line) is 1, the output of B0(exor) **K=B0'**(Complement B0). Thus the operation would be A+(B0'). Now 2's complement subtraction for two numbers A and B is given by A+B'. This suggests that when K=1, the operation being performed on the four bit numbers is subtraction.

Similarly If the Value of K=0, B0 (exor) **K=B0**. The operation is A+B which is simple binary addition. This suggests that When K=0, the operation being performed on the four bit numbers is addition. Then C0 is serially passed to the second full adder as one of it's outputs.

The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders. Then the third input is the B1, B2, B3 EXORed with K to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0.

Example: Lets take two 3 bit numbers A=010 and B=011 and input them in the full adder with both values of control lines.

For K=0
B0(exor)K=B0 and C0=K=0

Thus from first full adder
= A0+B0
= 0+1
= 1,
S0=1
C1=0

Similarly,
S1=0 with C2=1
S2=1 and C2=0

Thus, A = 010 =2 B = 011 = 3
Sum = 0101 = 5

For K=1 B0(exor)K=B0' and C0=K=1

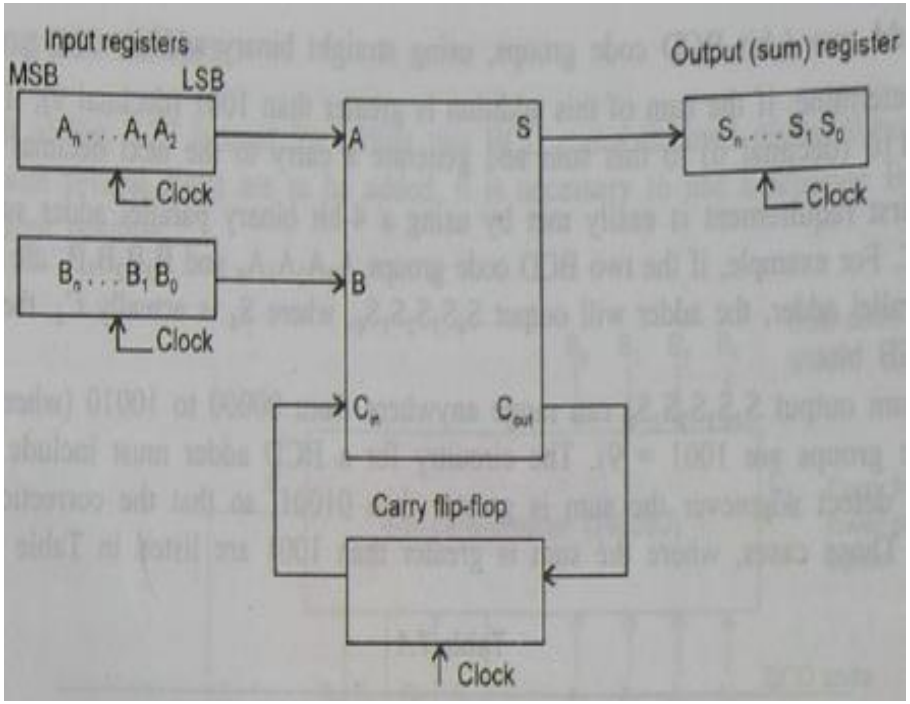
Thus S0=1 and C1=0

Similarly S1=1 and C2=0
S3=1 and c3=1

Thus, A = 010 = 2 B = 011 = 3
Sum(Difference) = 1111 = -1

Serial Binary Adder

Serial binary adder is a combinational logic circuit that performs the addition of two binary numbers in serial form. Serial binary adder performs bit by bit addition. Two shift registers are used to store the binary numbers that are to be added. A single full adder is used to add one pair of bits at a time along with the carry. The carry output from the full adder is applied to a D flip-flop. After that output is used as carry for next significant bits. The sum bit from the output of the full adder can be transferred into a third shift register.



Working Process:

Following is the procedure of addition using serial binary adder:

Step-1: The two shift registers A and B are used to store the numbers to be added.

Step-2: A single full adder is used to add one pair of bits at a time along with the carry.

Step-3: The contents of the shift registers shift from left to right and their output starting from **a** and **b** are fed into a single full adder along with the output of the carry flip-flop upon application of each clock pulse.

Step-4: The sum output of the full adder is fed to the most significant bit of the sum register.

Step-5: The content of sum register is also shifted to right when clock pulse is applied.

Step-6: After applying four clock pulse the addition of two registers (A & B) contents are stored in sum register.

BCD Adder

BCD stand for binary coded decimal. Consider two 4-bit numbers A and B. The value of A and B can varies from 0(0000 in binary) to 9(1001 in binary) because we are considering decimal numbers. The output will varies from 0 to 18, if we are not considering the carry from the previous sum. But if we are considering the carry, then the maximum value of output will be 19 (i.e. 9+9+1 = 19). When we are simply adding A and B, then we get the binary sum. Here, to get the output in BCD form, we will use BCD Adder.

Example 1:
Input : A = 0111 B = 1000
Output : Y = 1 0101
Explanation: We are adding A(=7) and B(=8).
The value of binary sum will be 1111(=15).
But, the BCD sum will be **1 0101**, where 1 is 0001 in binary and 5 is 0101 in binary.

Example 2:
Input : A = 0101 B = 1001
Output : Y = 1 0100
Explanation: We are adding A(=5) and B(=9).
The value of binary sum will be 1110(=14).
But, the BCD sum will be **1 0100**, where 1 is 0001 in binary and 4 is 0100 in binary.

Note – If the sum of two number is less than or equal to 9, then the value of BCD sum and binary sum will be same otherwise they will differ by 6(0110 in binary).

Now, lets move to the table and find out the logic when we are going to add “0110”.
We are adding “0110” (=6) only to the second half of the table.

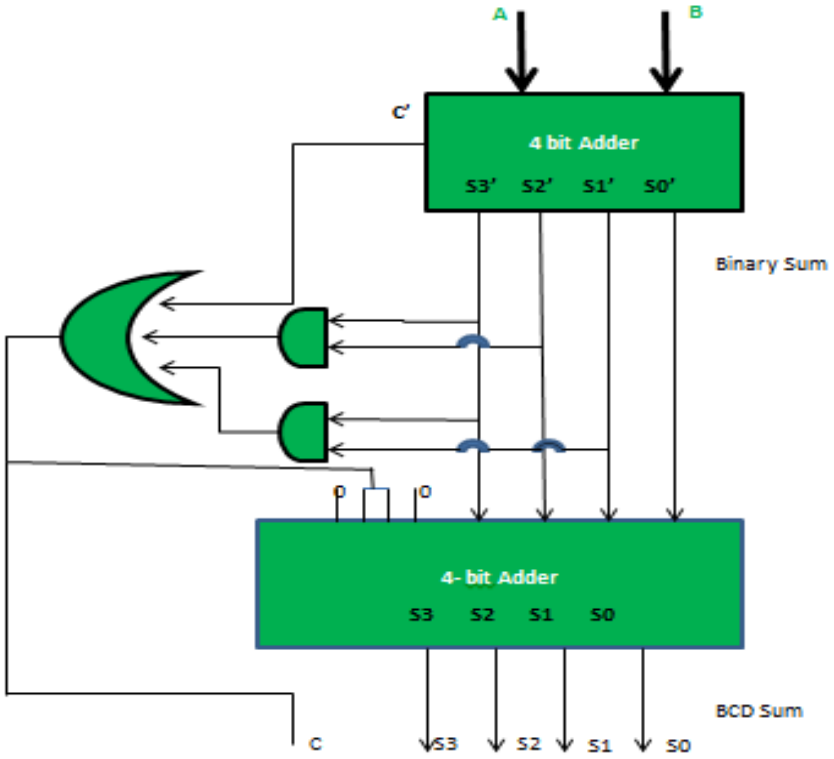
The conditions are:

If C’ = 1 (Satisfies 16-19)

If S3’.S2’ = 1 (Satisfies 12-15)

If S3’.S1’ = 1 (Satisfies 10 and 11)

So, our logic is **C' + S3'.S2' + S3'.S1' = 1**



Decimal	Binary Sum					BCD Sum				
	C'	S3'	S2'	S1'	S0'	C	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

Binary Multiplier

The multiplicand bits are B1 and B0, the multiplier bits are A1 and A0, and the product is C3,C2,C1,C0.

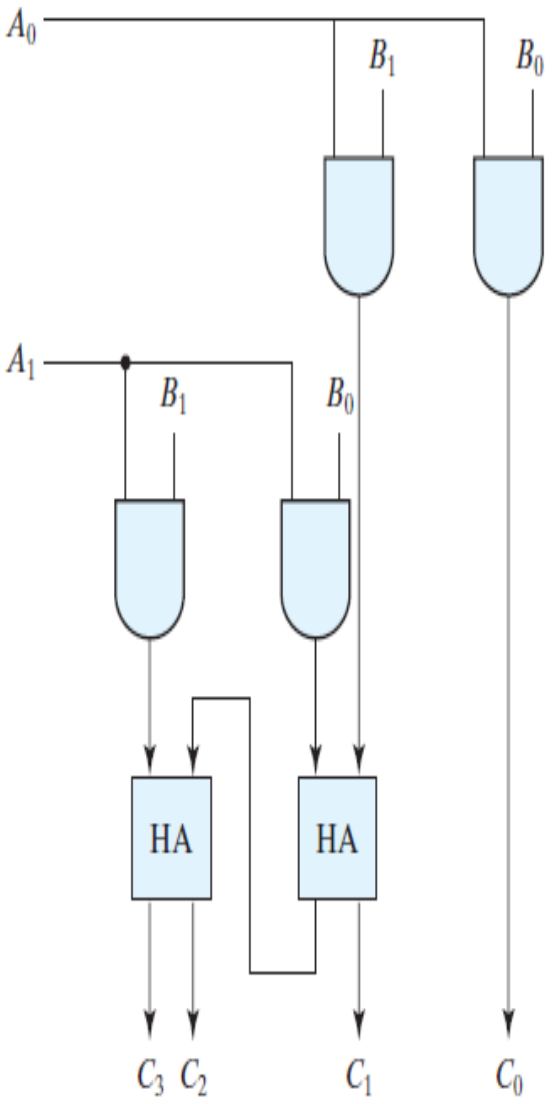
The first partial product is formed by multiplying B1 B0 by A0. The multiplication of two bits such as A0 and B0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation. Therefore, the partial product can be implemented with AND gates as shown in the diagram. The second partial product is formed by multiplying B1B0 by A1 and shifting one position to the left.

The two partial products are added with two half-adder (HA) circuits. Usually, there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

Example: Two-bit by two-bit binary multiplier

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

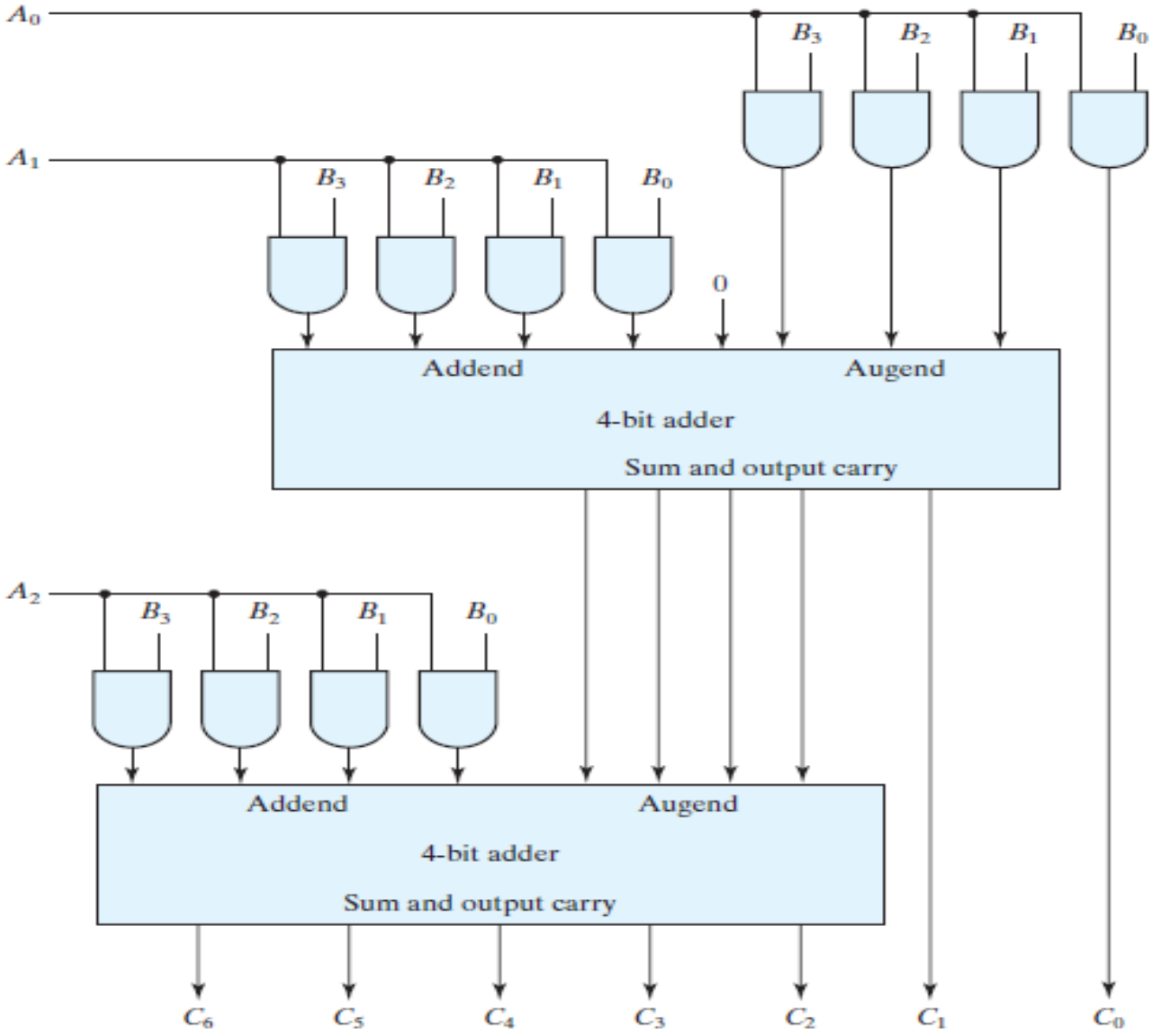


Example: Four-bit by three-bit binary multiplier

Consider a multiplier circuit that multiplies a binary number represented by four bits by a number represented by three bits.

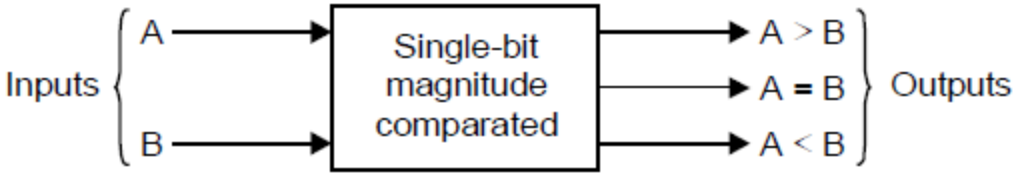
Let the multiplicand be represented by *B3 B2 B1 B0 and the multiplier by A2 A1 A0.*

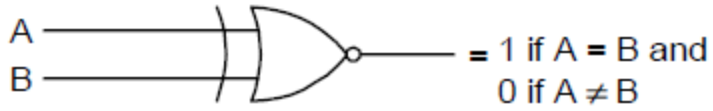
Since K = 4 and J = 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits.

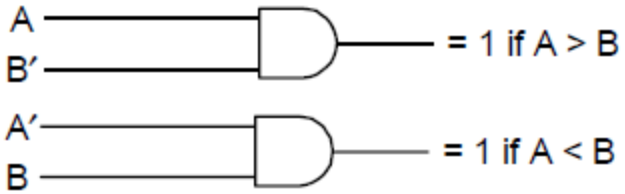


Magnitude Comparator

A magnitude comparator is a combinational circuit designed primarily to compare the relative magnitude of the two binary numbers A and B. Naturally, the result of this comparison is specified by three binary variables that indicate, whether A > B, A = B or A < B.







EX-OR gate is considered as the basic comparator circuit.

To implement the combinational circuit of a magnitude comparator the properties of Ex-NOR gate and AND gate can be used. An EX-NOR gate with two inputs A and B. If A = B then the output of Ex-NOR gate is equal to 1 otherwise 0.

In AND gates, one with A and B' as inputs and another with A' and B as their inputs. The AND gate output is 1 if A > B (i.e. A = 1 and B = 0) and 0 if A < B (i.e. A = 0 and B = 1). Similarly the AND gate output is 1 if A < B (i.e. A = 0 and B =1) and 0 if A > B (i.e. A = 1 and B = 0).

Example: Single bit magnitude comparator

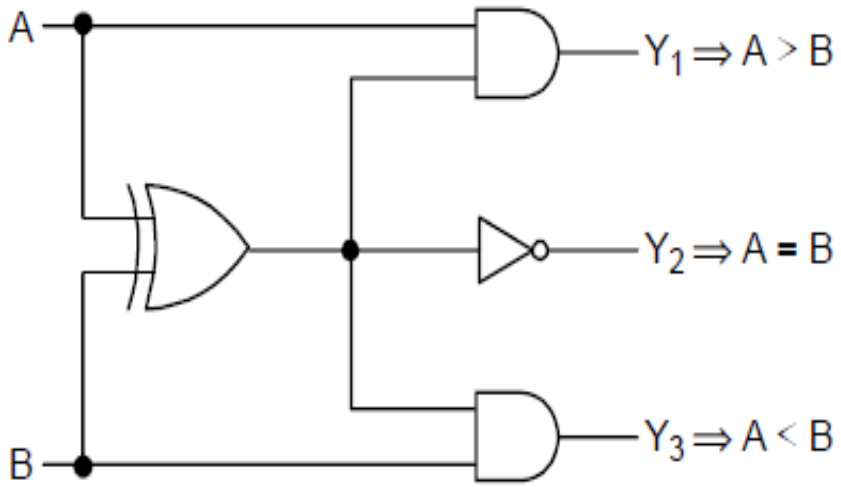
If the EX-NOR gate and two AND gates are combined as shown in figure below, *the circuit* with function as single bit magnitude comparator. For EX-NOR implementation, EX-OR followed by an inverter is used.

Inputs		*Output*		
A	*B*	Y_1	Y_2	Y_3
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Y_1 is high when A > B.

Y_2 is high when A = B

Y_3 is high when A < B.



Example: 4-bit Magnitude Comparator

Consider two numbers A and B, with four digits each.

$$A = A_3\ A_2\ A_1\ A_0$$
$$B = B_3\ B_2\ B_1\ B_0.$$

(*a*) The two numbers are equal if all pairs of significant digits are equal i.e. if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B$ and $A_0 = B_0$. We have seen that equality relation is generaed by EX-NOR gate. Thus

$x_i = A_i \cdot B_i = A_i\ B_i + A_i'\ B_i'$, $i = 0, 1, 2, 3.$

Where x_i represents the equality of two numbers

$x_i = 1$, if $A = B$.

$x_i = 0$, otherwise.

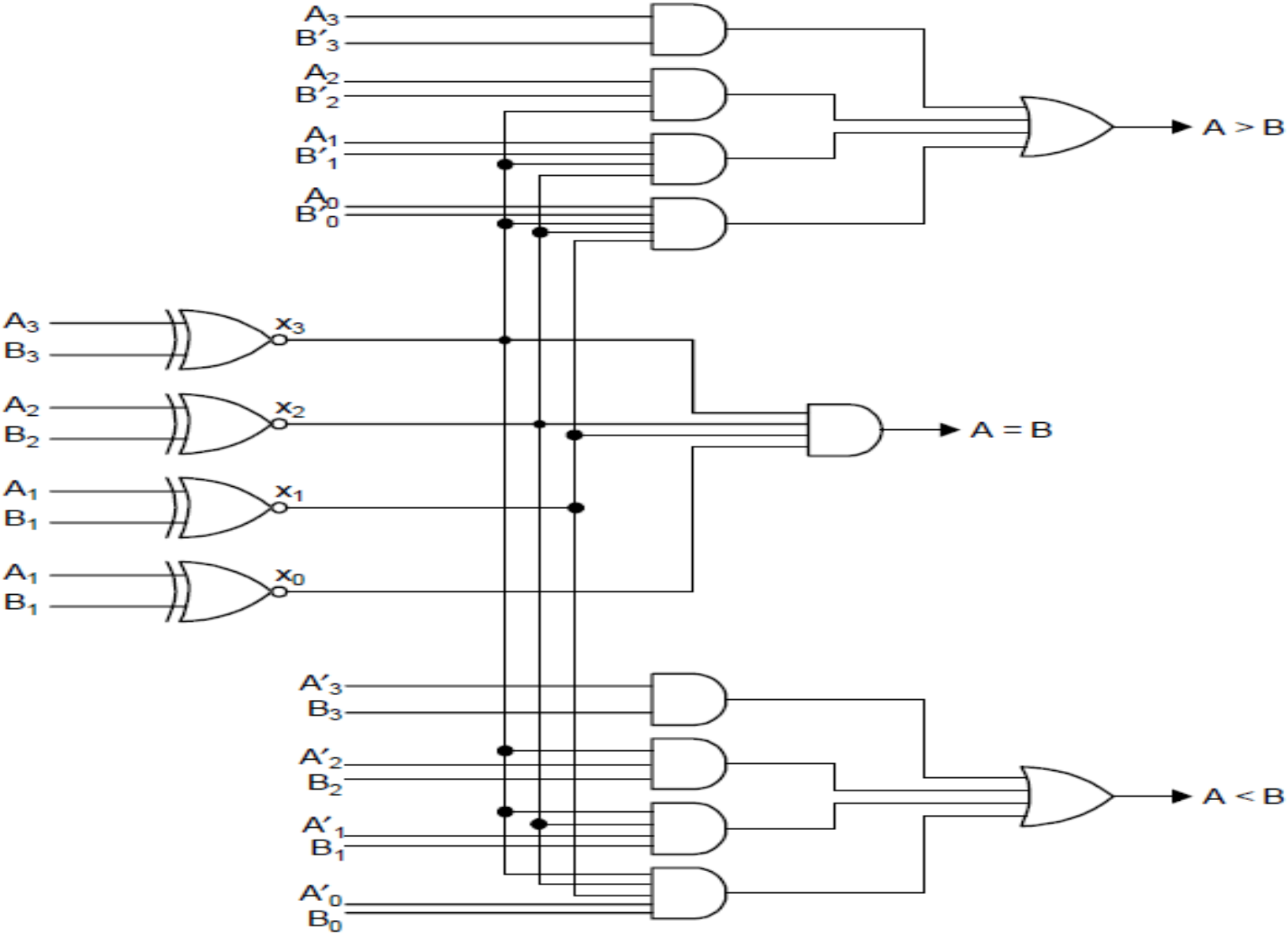
If follows an AND operation of all variables.

Therefore, $(A = B) = x_3\ x_2\ x_1\ x_0 = 1$ only if all pairs are equal.

(*b*) To determine if $A > B$ or $A < B$, we check the relative megnitude of pairs of significant digits starting from MSB. If the two digits are equal, we compare the next lower significant pair of digits. The comparison follows until a pair of unequal digits are reached. If the corresponding digit of A is 1 and that of B is 0, we say that $A > B$. If the corresponding digit A is 0 and that of B is $1 \Rightarrow A < B$.

This discussion can be expressed logically as :

$$(A > B) = A_3\ B_3' + x_3\ A_2\ B_2' + x_3\ x_2\ A_1\ B_1' + x_3\ x_2\ x_1\ A_0\ B_0'$$
$$(A < B) = A_3'\ B_3 + x_3\ A_2'\ B_2 + x_3\ x_2\ A_1'\ B_1 + x_3\ x_2\ x_1\ A_0'\ B_0.$$

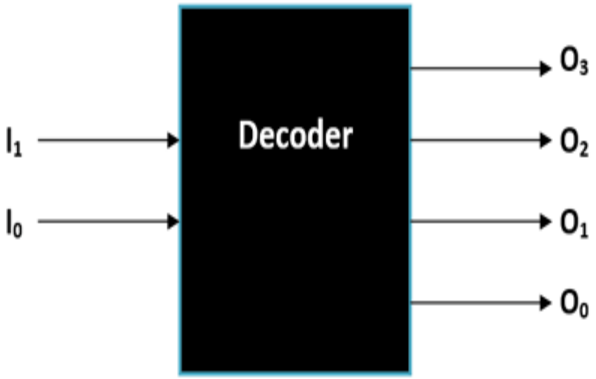


Logical implementation of or 4-bit magnitude comparator

Decoders

- A decoder has
  - N inputs
  - 2^N outputs
- A decoder selects one of 2^N outputs by decoding the binary value on the N inputs.
- The decoder generates all of the minterms of the N input variables.
  - Exactly one output will be active for each combination of the inputs.

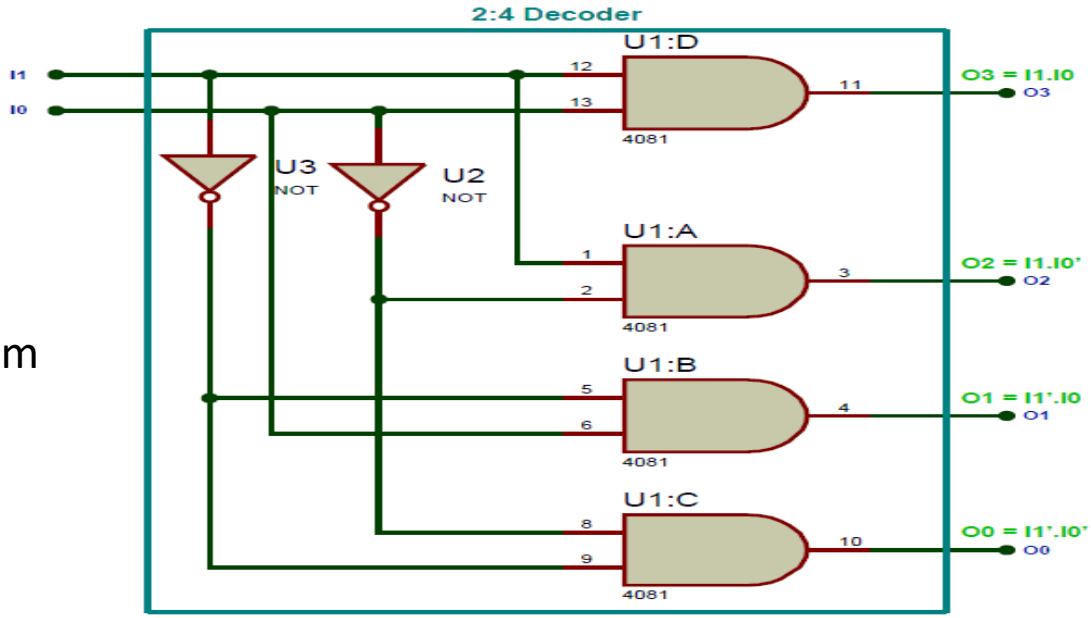
2x4 Decoder:



Functional Table of 2x4 Decoder

Input		Output			
I1	I0	O3	O2	O1	O0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Logic Diagram

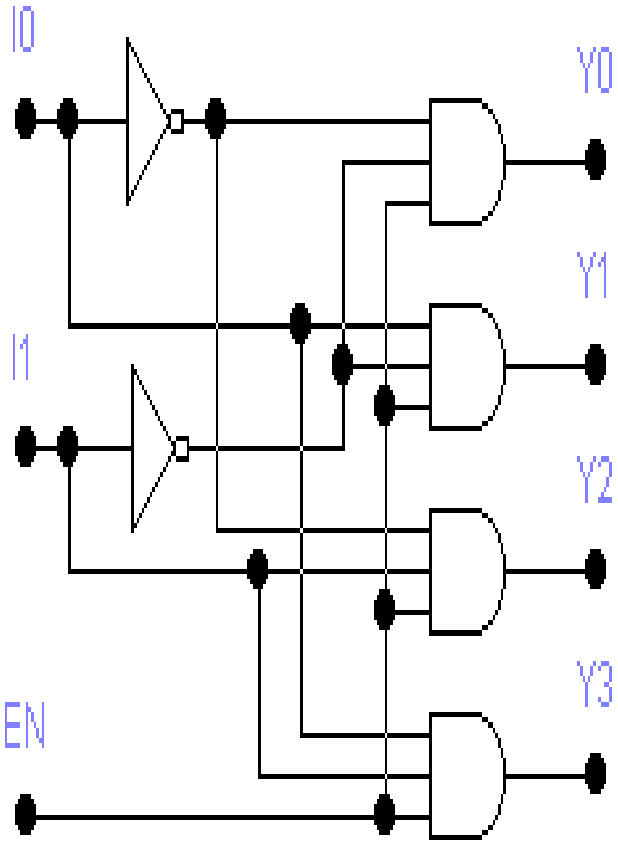


2x4 decoder with enable input:

Functional Table

Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
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

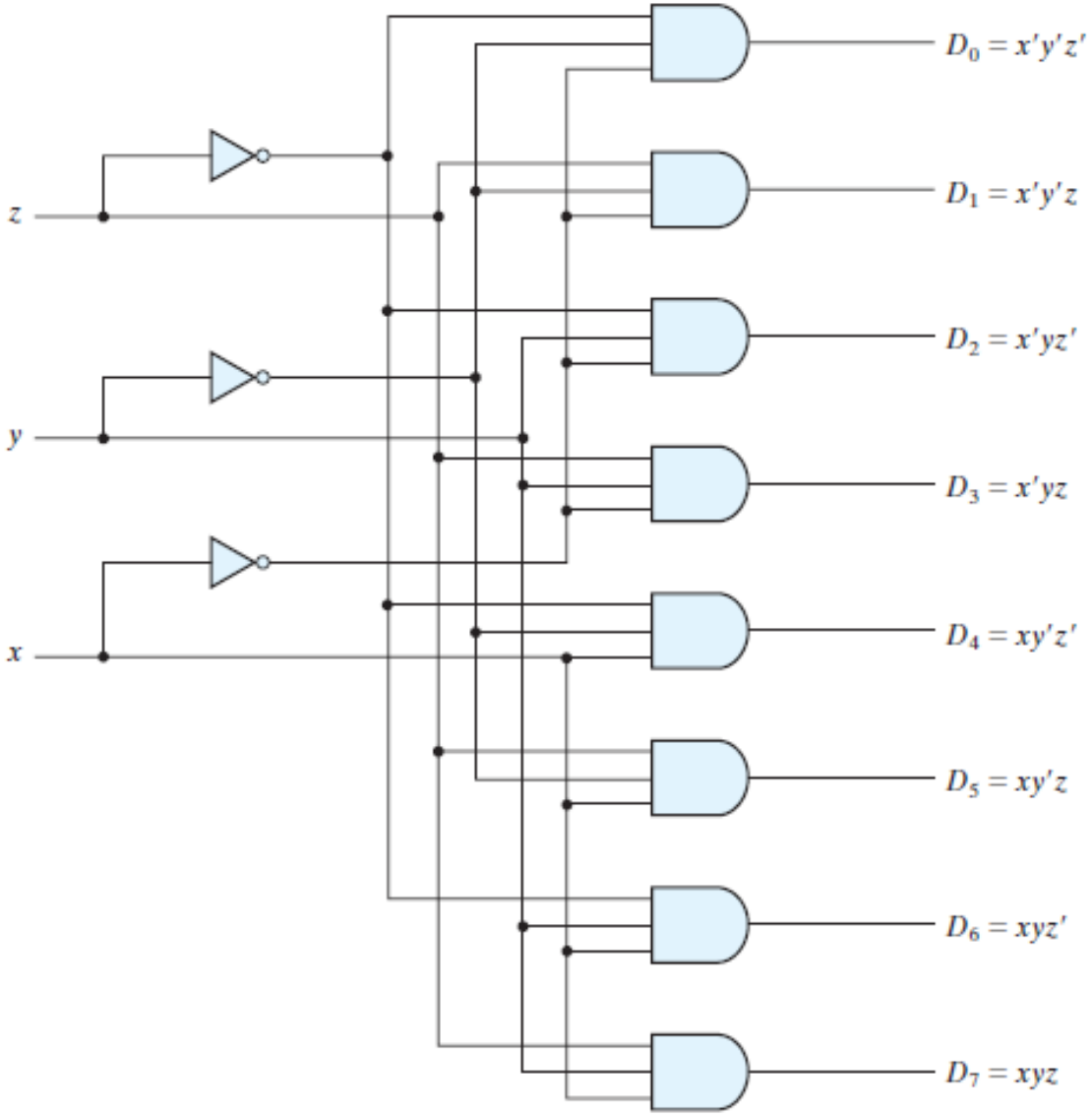
Logic Diagram



3x8 decoder

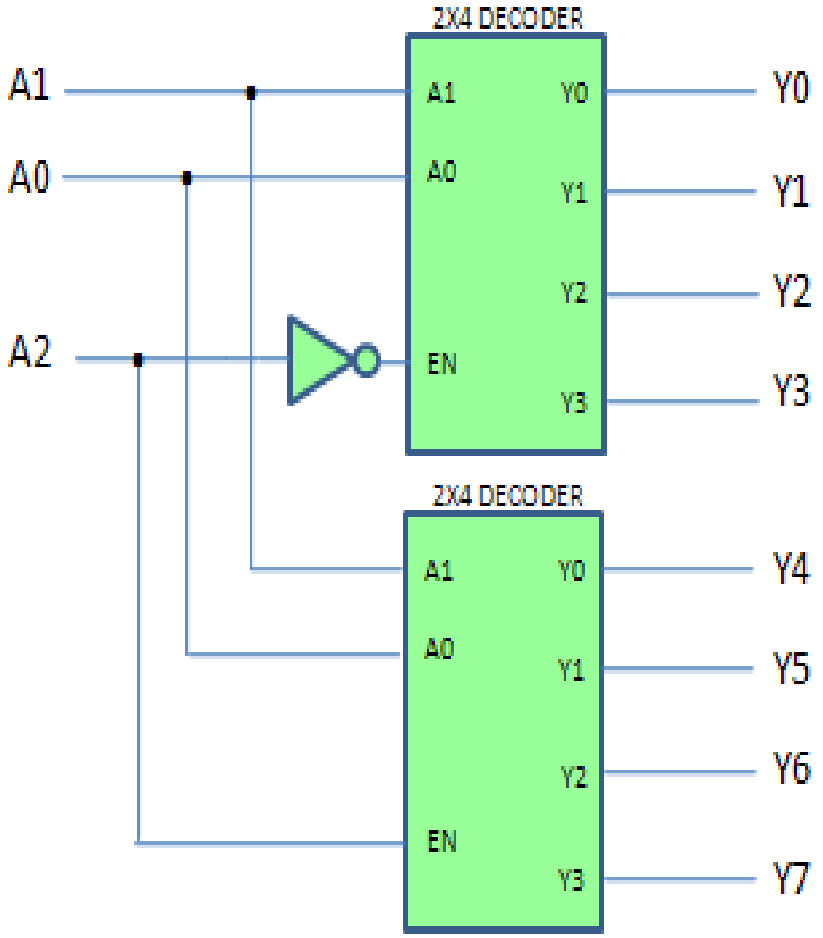
Inputs			outputs							
X	Y	Z	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Three-to-eight-line decoder



3x8 decoder using two 2x4 decoders

- The inputs A0 and A1 is connected as parallel inputs for both the decoders and then the Enable pin of the Second Decoder is made to act as A2 (third input).
- The Inverted signal of A2 is given to the Enable pin of first decoder to get the outputs Y0 to Y3. Here the outputs Y0 to Y3 is referred as Lower four minterms and the outputs Y4 to Y7 is referred as higher four minterms.
- The lower order minterms are obtained from the first decoder and the higher order minterms are obtained from the second decoder.

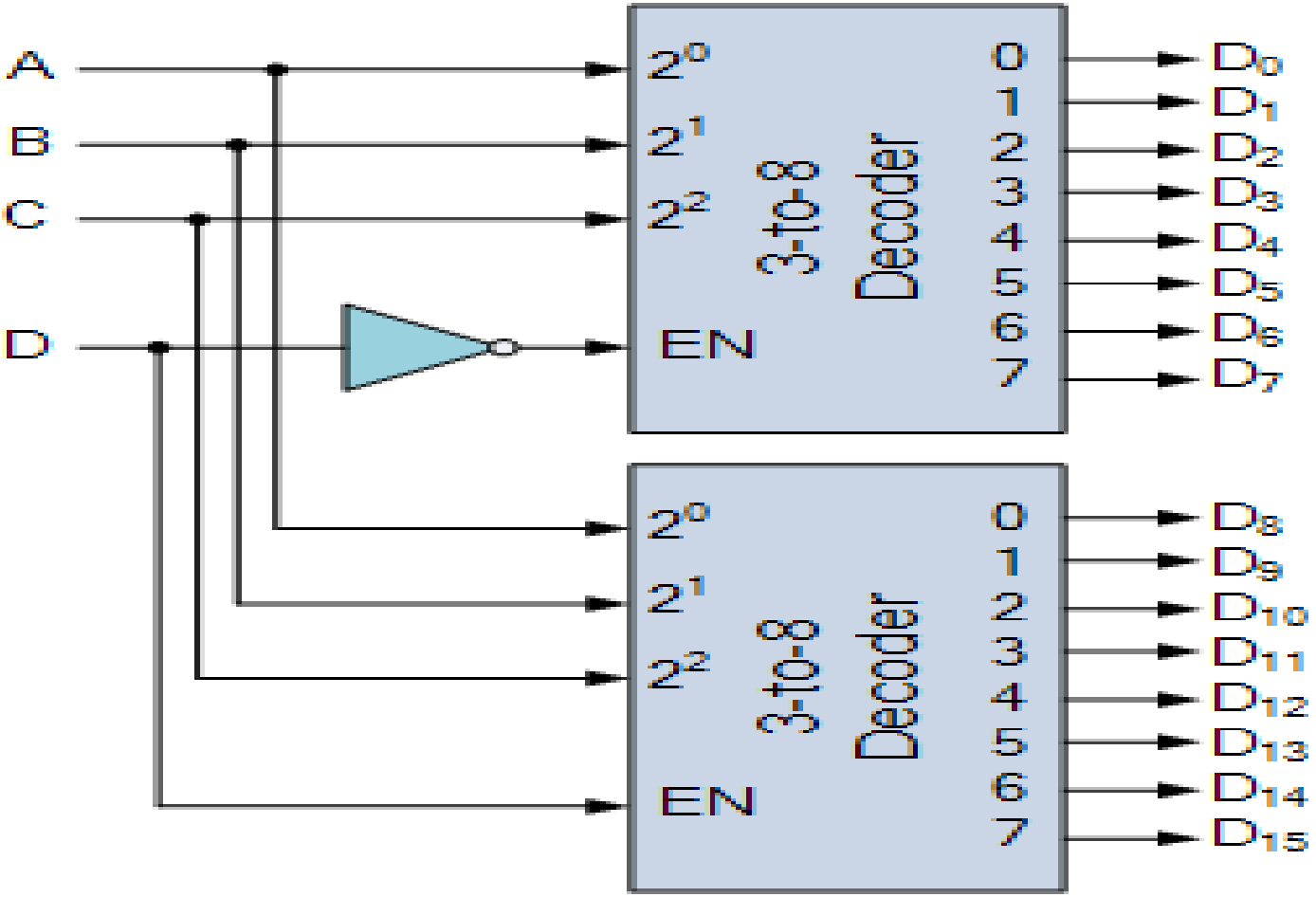


Logic Diagram

4x16 decoder

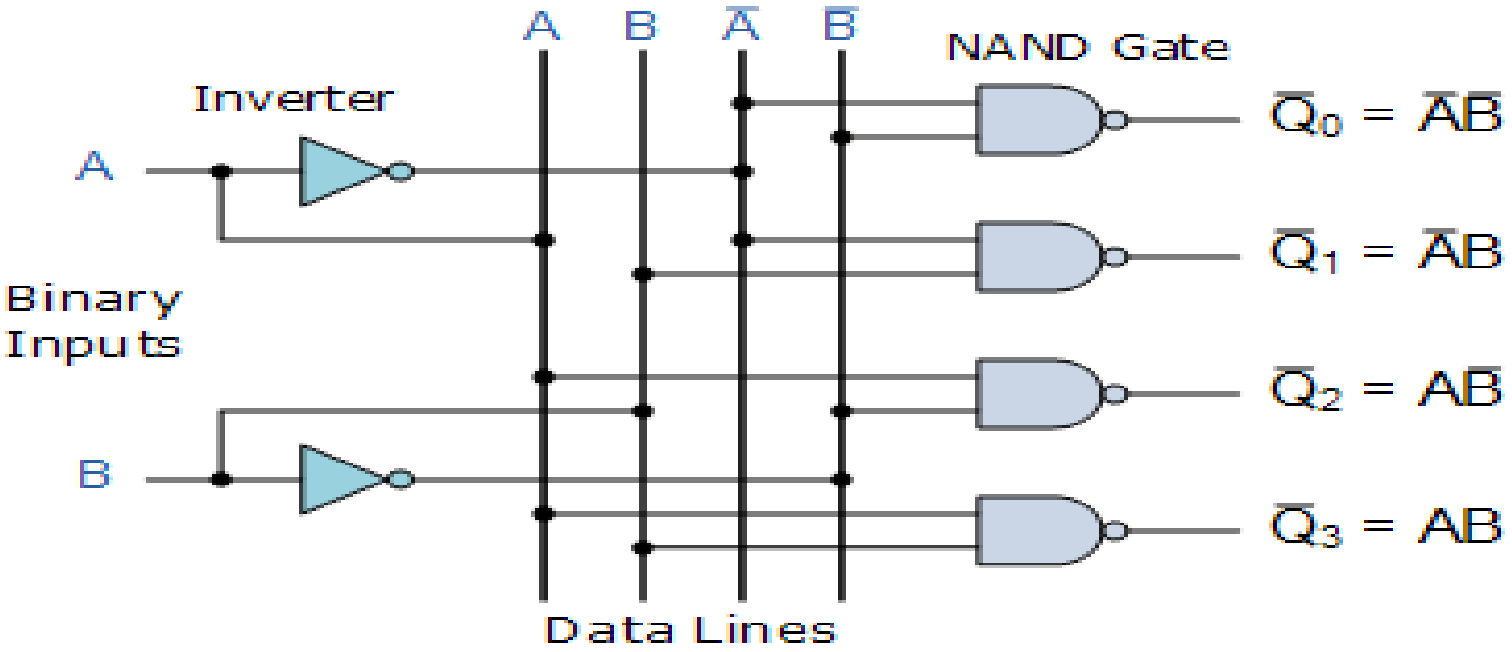
Inputs				outputs							
D	C	B	A	Y15	Y14	..	..	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	1	0	0	0
..											
0	1	1	1								
1	0	0	0								
1	0	0	1								
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

4x16 decoder using 3x8 decoder



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

2x4 decoder with active low output



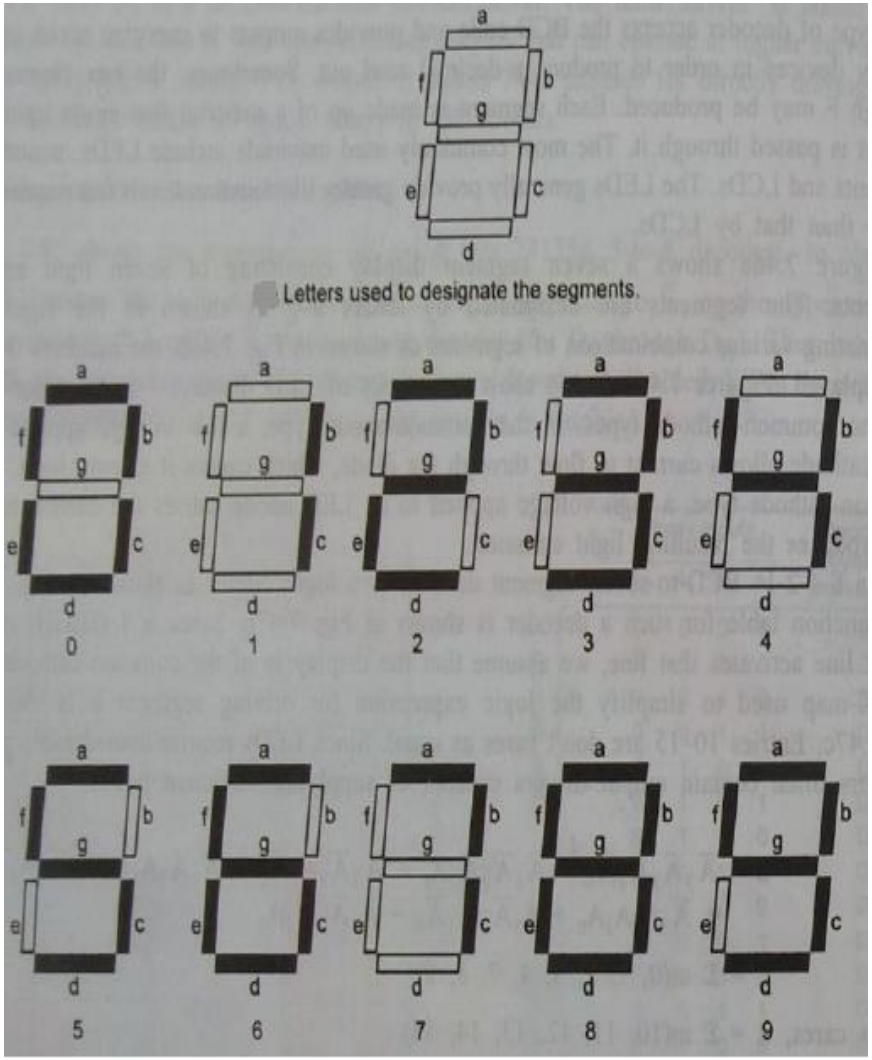
Truth Table

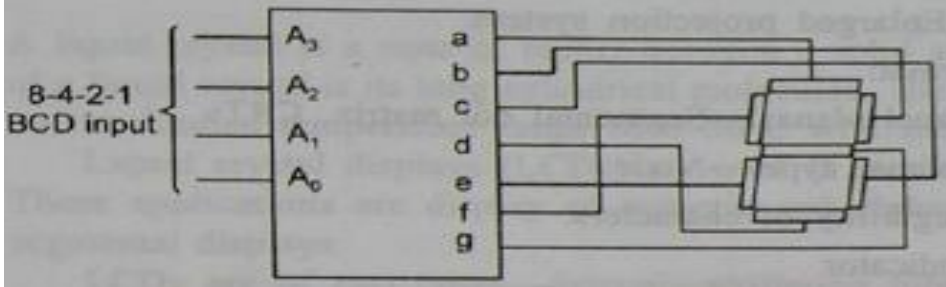
A	B	Q_0	Q_1	Q_2	Q_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

BCD to Seven Segment decoder

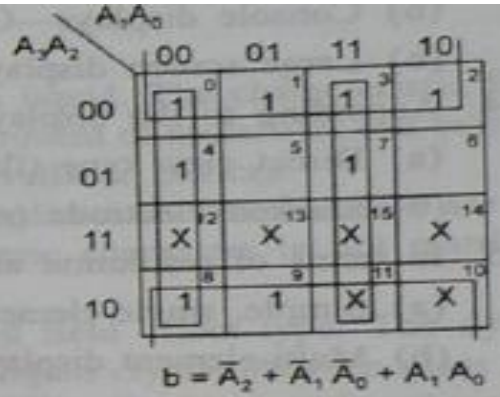
Display devices are used to provide display of numbers, alphabets and symbols in response to electrical input and are called as Electronic Display Systems. This display device accept input in the form of BCD number and display the particular number on the display.

Binary Inputs	Decoder Outputs	7 Segment Display Outputs
D C B A	a b c d e f g	
0 0 0 0	1 1 1 1 1 1 0	0
0 0 0 1	0 1 1 0 0 0 0	1
0 0 1 0	1 1 0 1 1 0 1	2
0 0 1 1	1 1 1 1 0 0 1	3
0 1 0 0	0 1 1 0 0 1 1	4
0 1 0 1	1 0 1 1 0 1 1	5
0 1 1 0	1 0 1 1 1 1 1	6
0 1 1 1	1 1 1 0 0 0 0	7
1 0 0 0	1 1 1 1 1 1 1	8
1 0 0 1	1 1 1 1 0 1 1	9





(a) Logic circuit



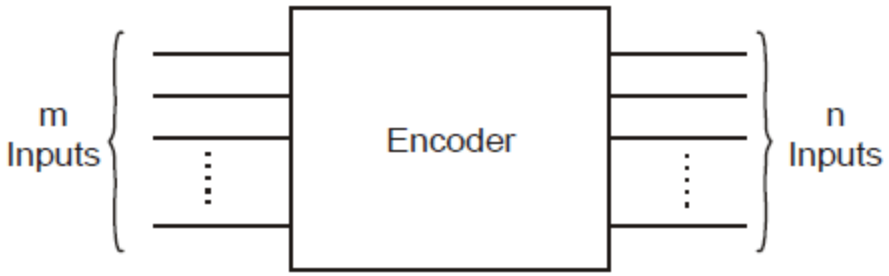
(c) K-map to derive simplified expression for driving segment (b)

Decimal digit	8-4-2-1 BCD				Seven segment code						
	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

(b) Function table

The K-map method can be used to derive the logic expression of the decimal numbers for display.

Encoders



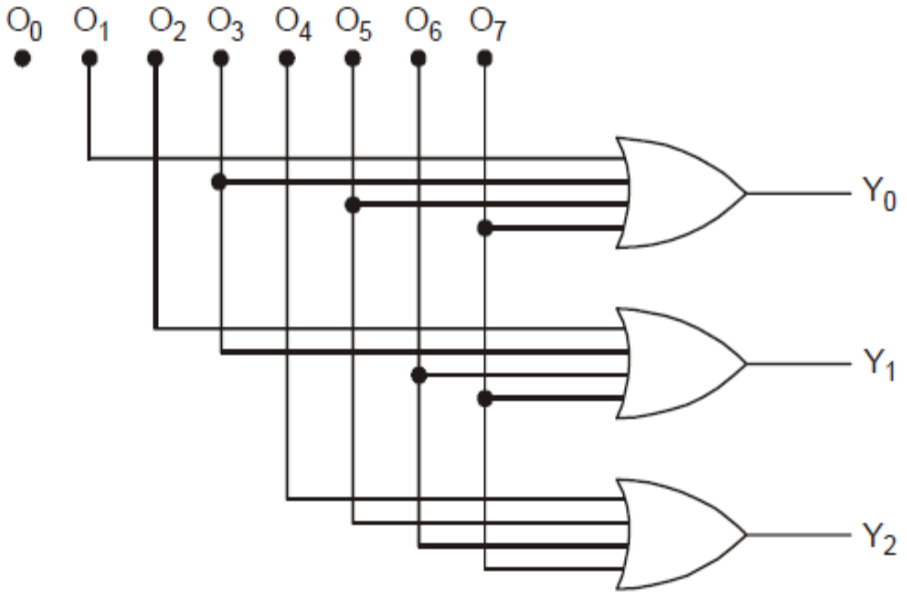
- An encoder has
  - 2^N inputs
  - N outputs
- An encoder outputs the binary value of the selected (or active) input.
- An encoder performs the inverse operation of a decoder.
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output *z* is equal to 1 when the input octal digit is 1, 3, 5,or 7. Output *y* is 1 for octal digits 2, 3, 6, or 7, and output *x* is 1 for digits 4, 5, 6, or 7.

Octal to Binary Encoder

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output Y0 is equal to 1 when the input octal digit is 1, 3, 5,or 7. Output Y1 is 1 for octal digits 2, 3, 6, or 7, and output Y2 is 1 for digits 4, 5, 6, or 7.

Inputs								Outputs		
O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$Y_0 = O_1 + O_3 + O_5 + O_7$$
$$Y_1 = O_2 + O_3 + O_6 + O_7$$
$$Y_2 = O_4 + O_5 + O_6 + O_7$$



Octal to Binary Encoder

- Issues
  - What if more than one input is active?
  - (111 000 00)
  - What if no inputs are active?
  - (000 000 00)

That is the octal to binary encoder has two limitations:
1. Only one input can be active at any given time. If two or more inputs are equal to 1 at the same time, the O/P is undefined. For example if O_2 and O_5 are active simultaneously, the o/p of encoder will be 111 that is equal to binary 7. This does not represent binary 2 or 5.
2. The output with all 0's is generated when all inputs are '0', and is also true when O_0 = '1'.

The first problem is taken care by a circuit, called as 'priority encoder'. It establishes a priority to ensure that only one input is active (High) at a given time.
The second problem is taken care by an extra line in the encoder output, called 'valid output indicator' that specifies the condition that none of the inputs are active.

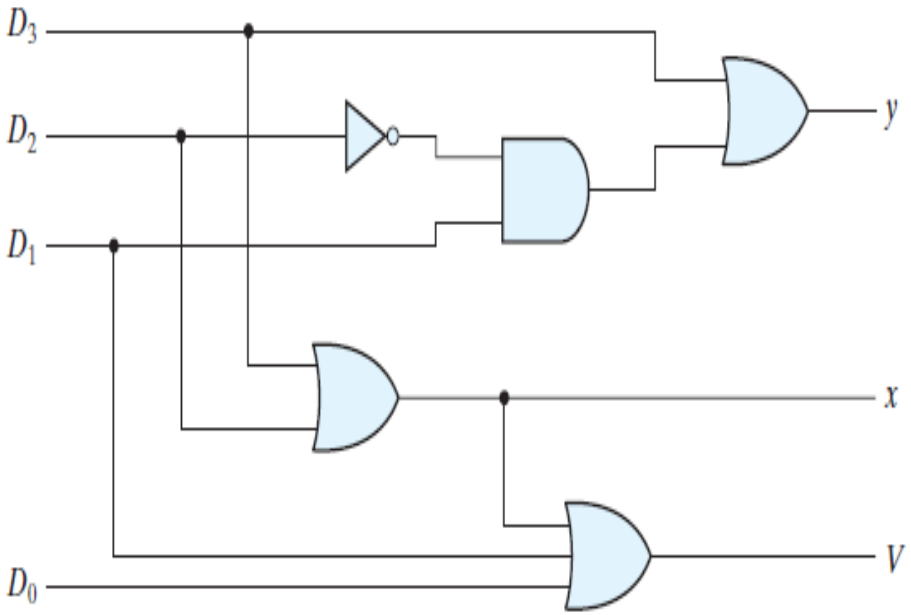
Priority Encoders

- If more than one input is active, the higher-order input has priority over the lower-order input.
  - The higher value is encoded on the output
- A valid indicator, v, is included to indicate whether or not the output is valid.
  - Output is invalid when no inputs are active
    - v = 0
  - Output is valid when at least one input is active
    - v = 1

Inputs				Outputs		
D3	D2	D1	D0	X	Y	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

Boolean Expressions

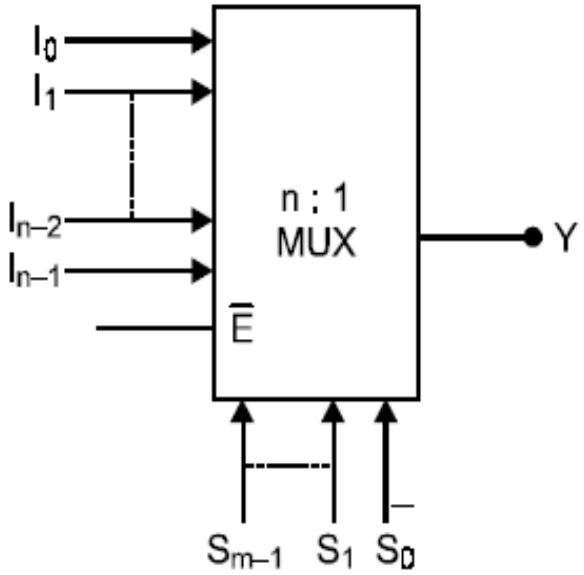
$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$



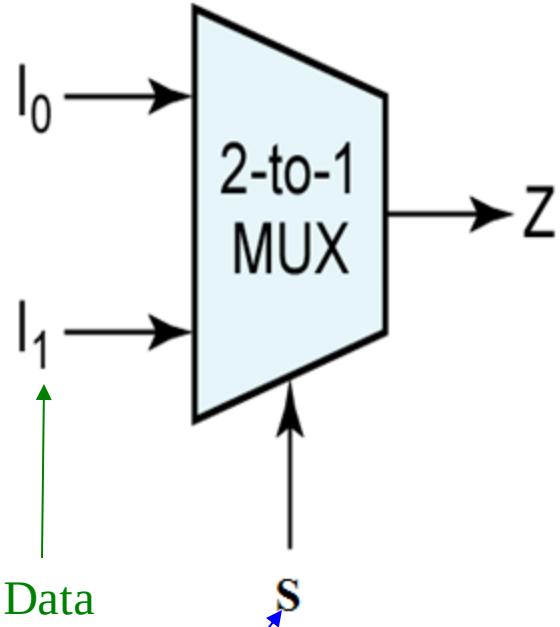
Logic Diagram

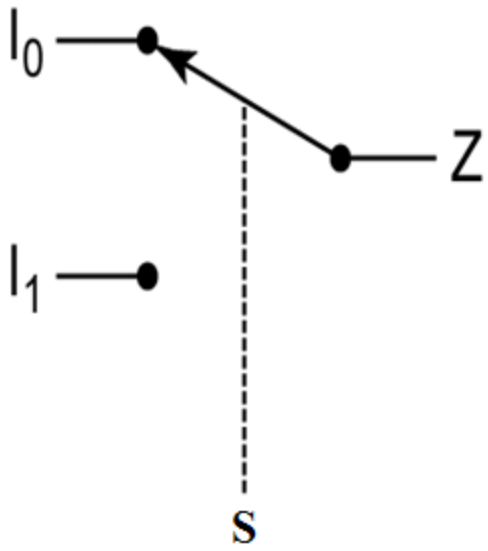
Multiplexers

- A multiplexer has
  - N control inputs(Select lines)
  - 2^N data inputs
  - 1 output
- A multiplexer routes (or connects) the selected data input to the output.
  - The value of the control inputs determines the data input that is selected.



2X1 Multiplexer

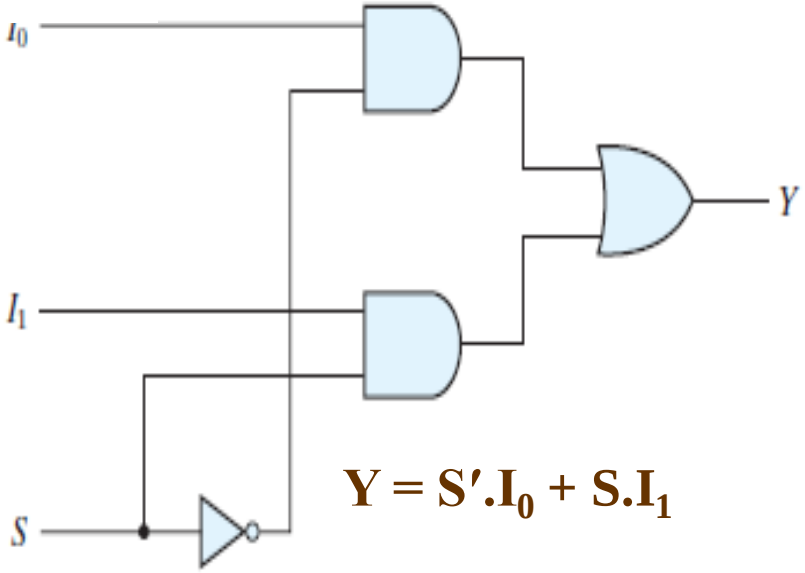




S (Select Line)	F
0	I_0
1	I_1

$$Z = S'.I_0 + S.I_1$$

Logic Diagram and Truth table



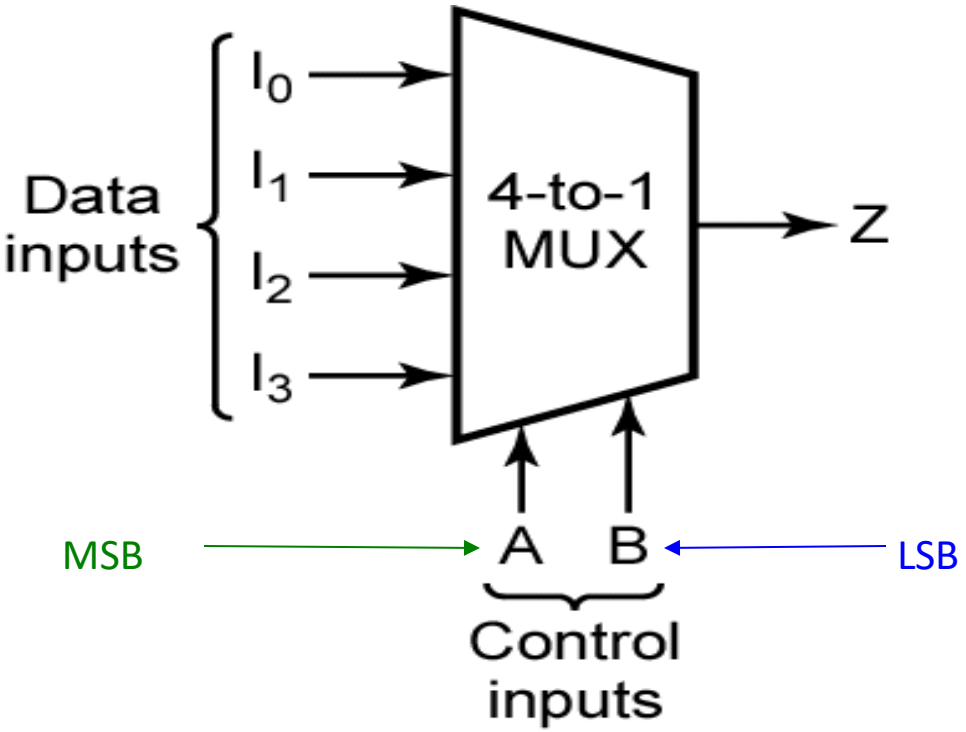
$$Y = S'.I_0 + S.I_1$$

A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination. The circuit has two data input lines, one output line, and one selection line S . When S = 0, the upper AND gate is enabled and I_0 has a path to the output. When S = 1, the lower AND gate is enabled and I_1 has a path to the output.

The multiplexer acts like an electronic switch that selects one of two sources.

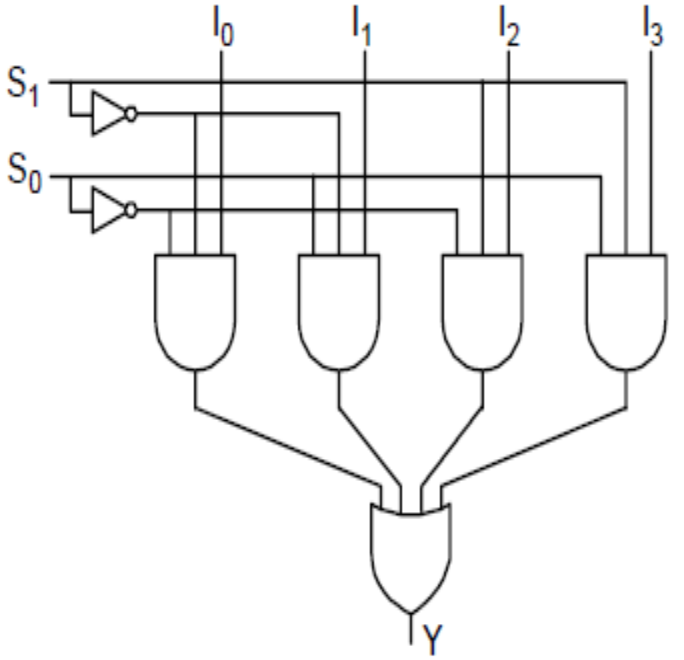
4X1 Multiplexer

Select Line		O/P
A (S1)	B (S0)	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

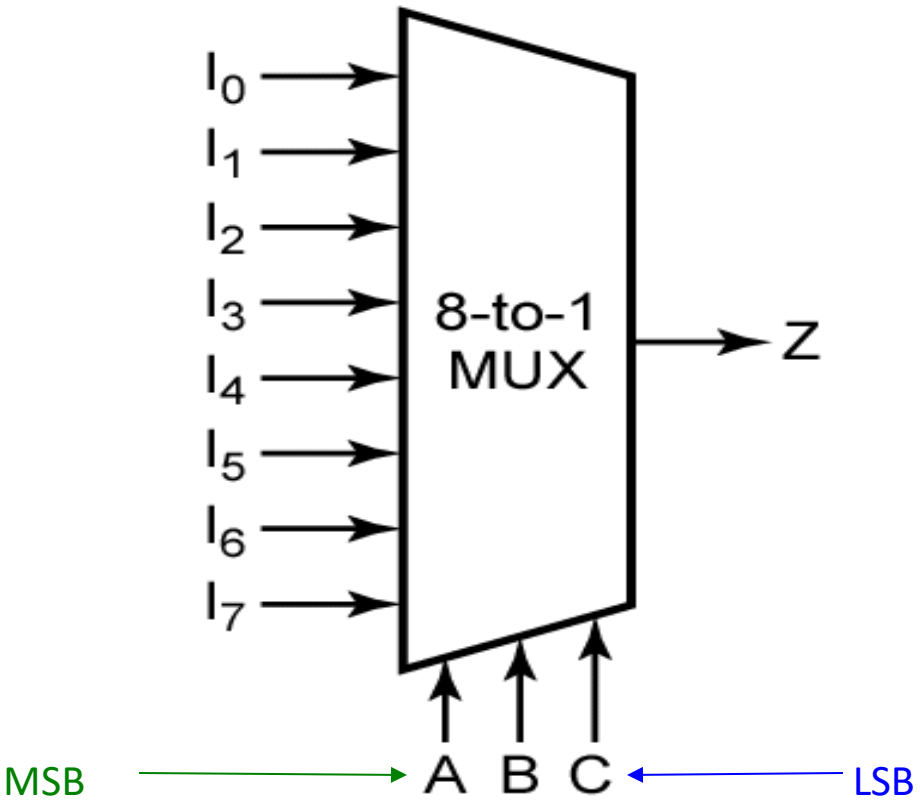


$$Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3$$

$$Y = I_0.\overline{S}_1.\overline{S}_0 + I_1.\overline{S}_1.S_0 + I_2.S_1.\overline{S}_0 + I_3.S_1.S_0$$

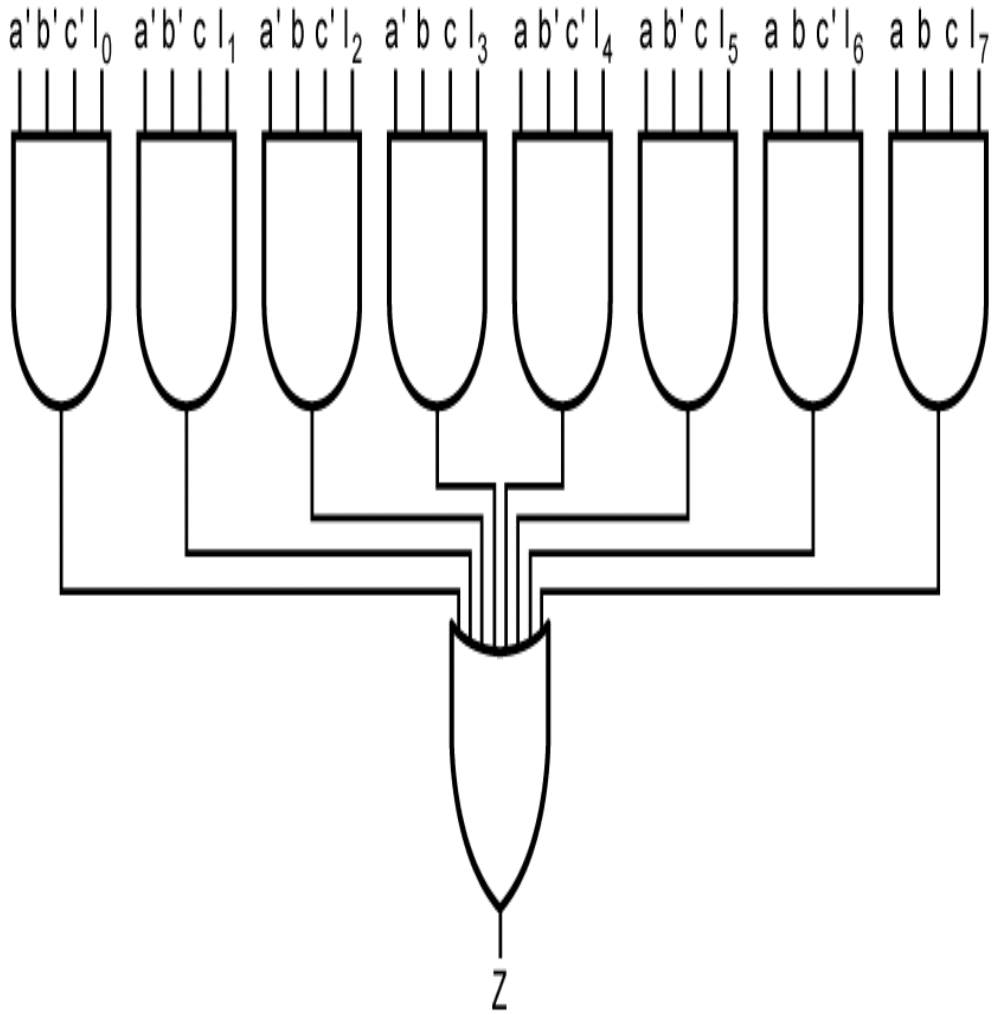


8X1 Multiplexer

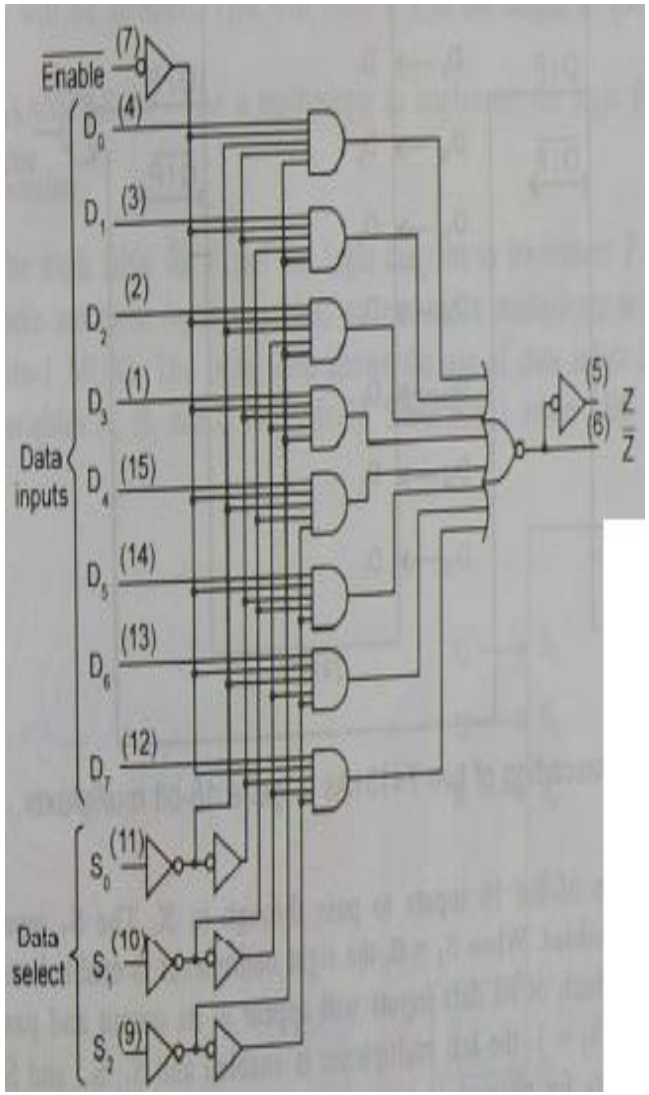


Select Line			O/P
A (S2)	B (S1)	C (S0)	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

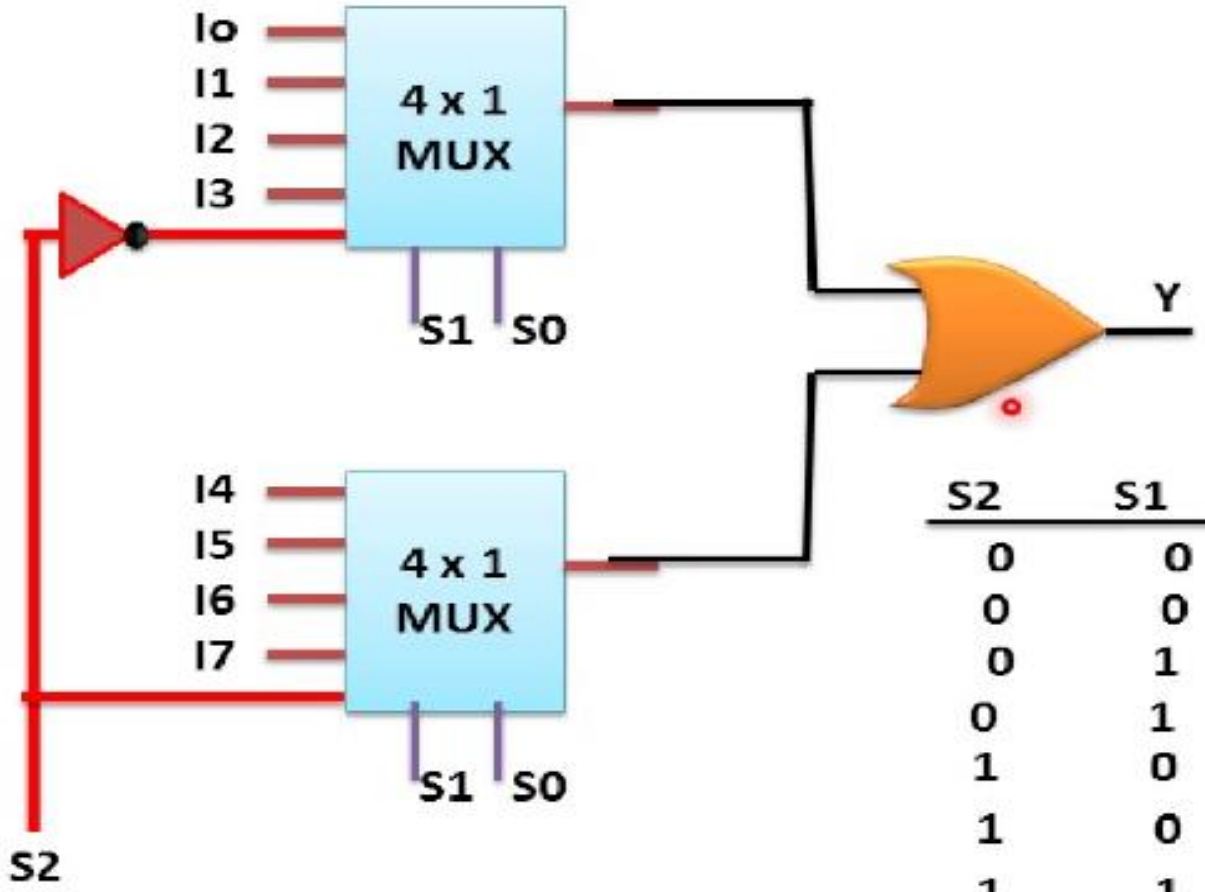
$$Z = A'.B'.C'.I_0 + A'.B'.C.I_1 + A'.B.C'.I_2 + A'.B.C.I_3 + A.B'.C'.I_0 + A.B'.C.I_1 + A'.B.C'.I_2 + A.B.C.I_3$$



Logic Diagram of 8x1 Multiplexer

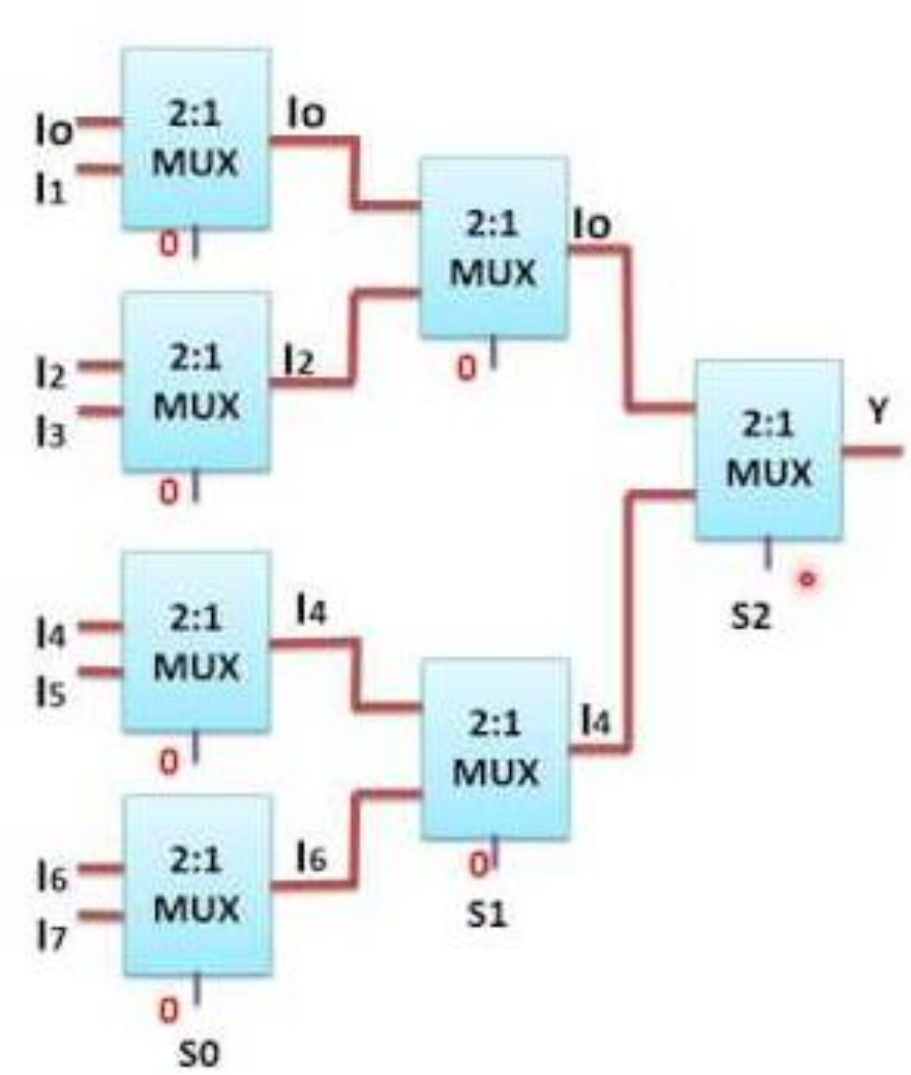


8X1 MUX using two 4X1 MUXs



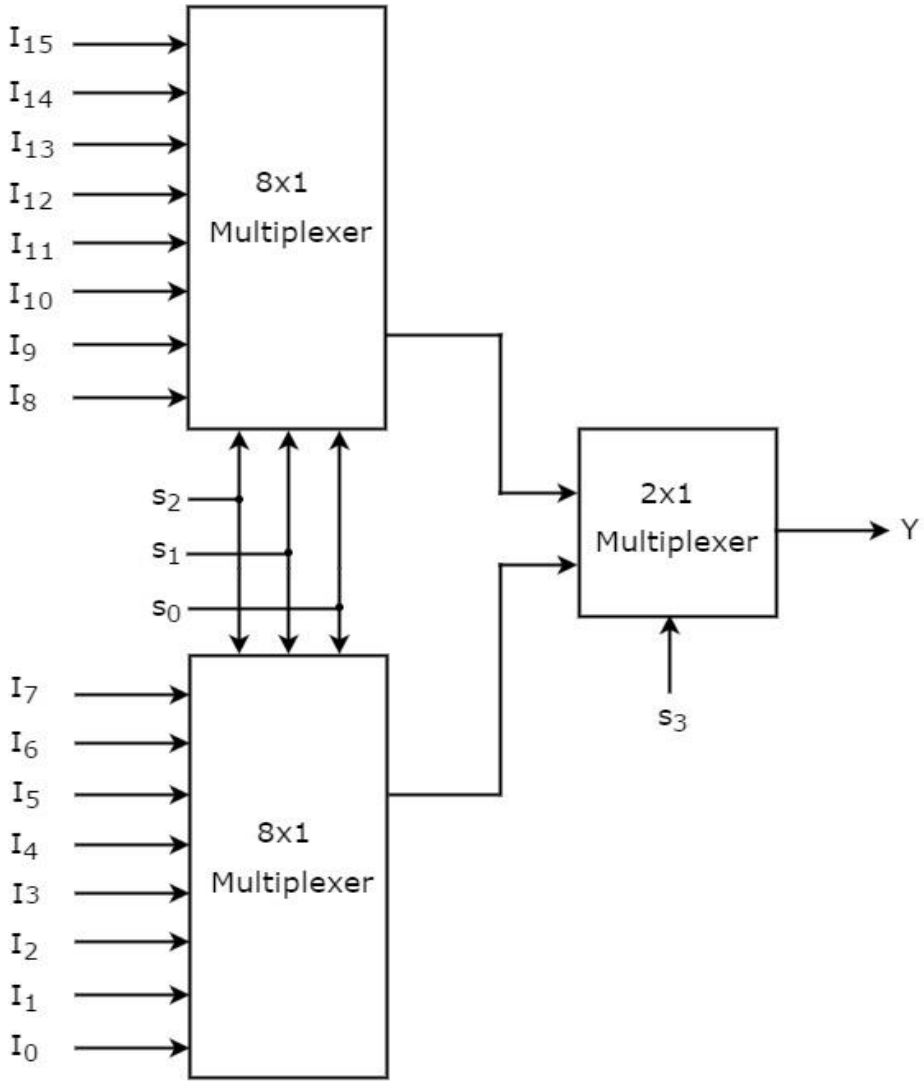
S2	S1	S0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

8X1 MUX using Seven 2X1 MUXs : This consists of 8 inputs and 3 select lines

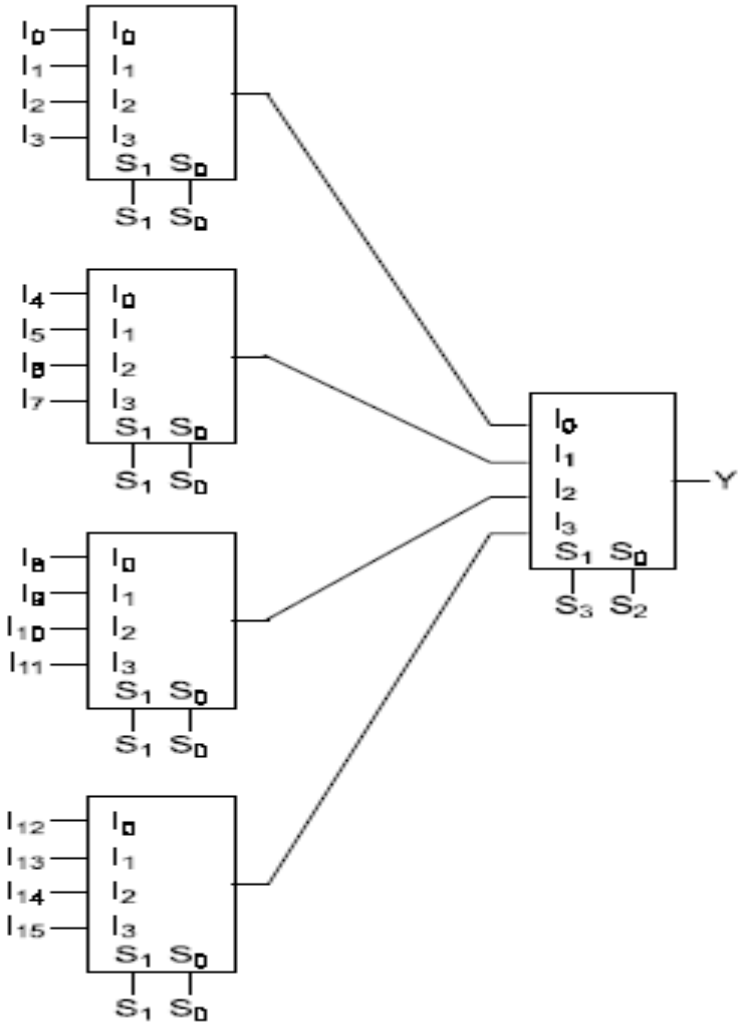


S2	S1	S0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

16x1 MULTIPLEXER USING Two 8X1 MUX

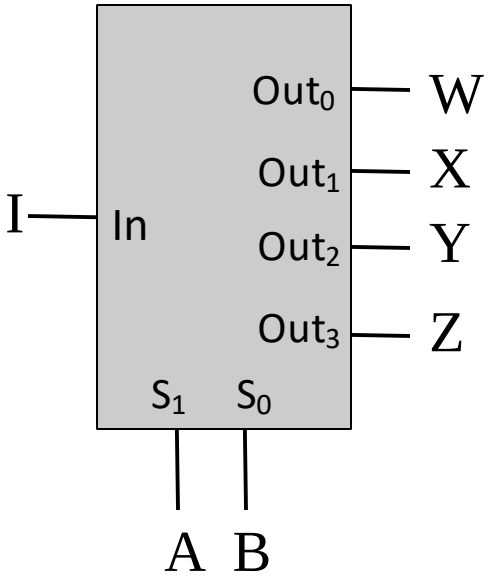


16x1 MULTIPLEXER USING Five 4X1 MUX



Demultiplexers

- A demultiplexer has
  - N control inputs
  - 1 data input
  - 2^N outputs
- A demultiplexer routes (or connects) the data input to the selected output.
  - The value of the control inputs determines the output that is selected.
- A demultiplexer performs the opposite function of a multiplexer.



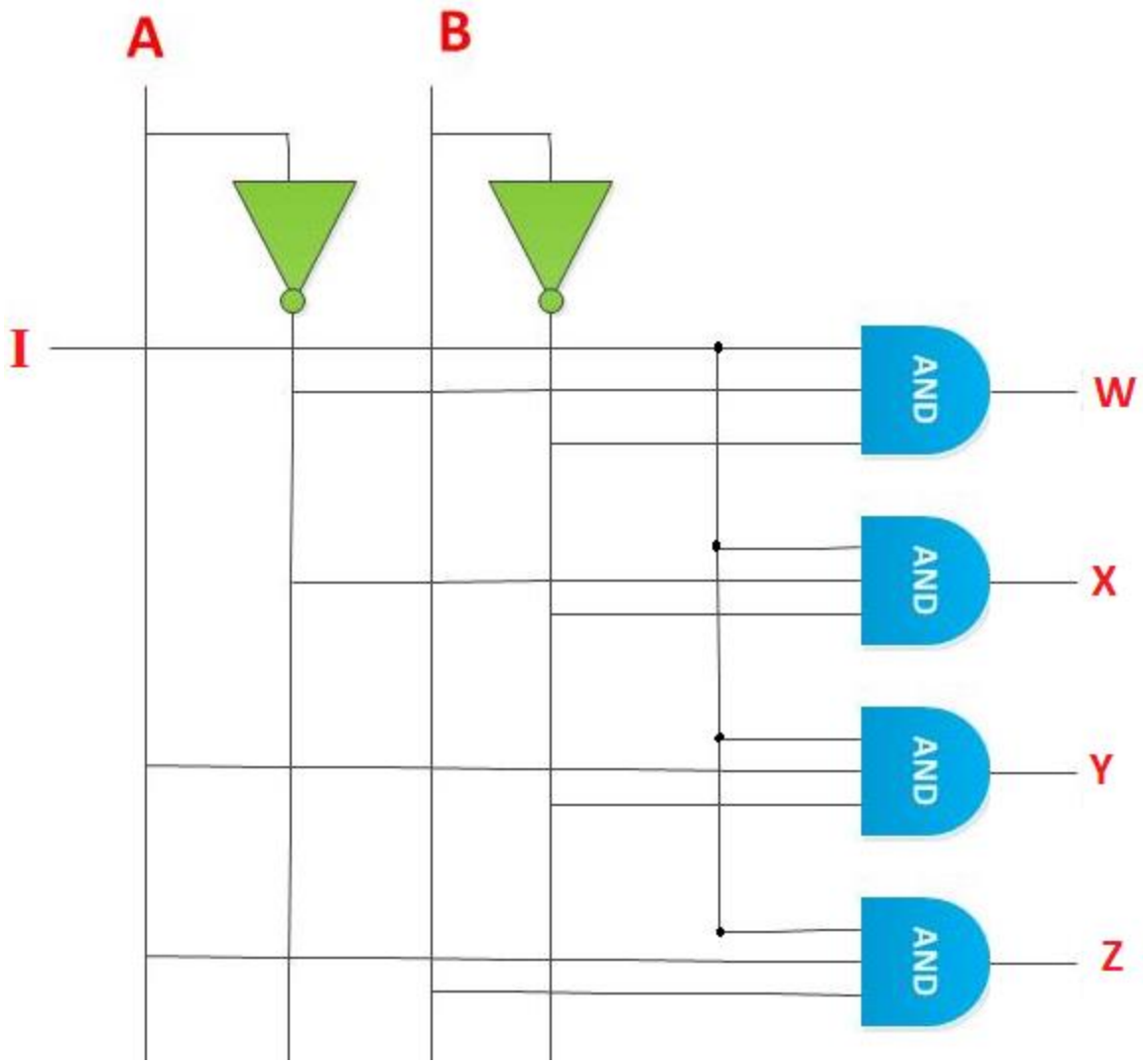
W = A'.B'.I

X = A.B'.I

Y = A'.B.I

Z = A.B.I

Select Line		Outputs			
A (S1)	B (S0)	W	X	Y	Z
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I



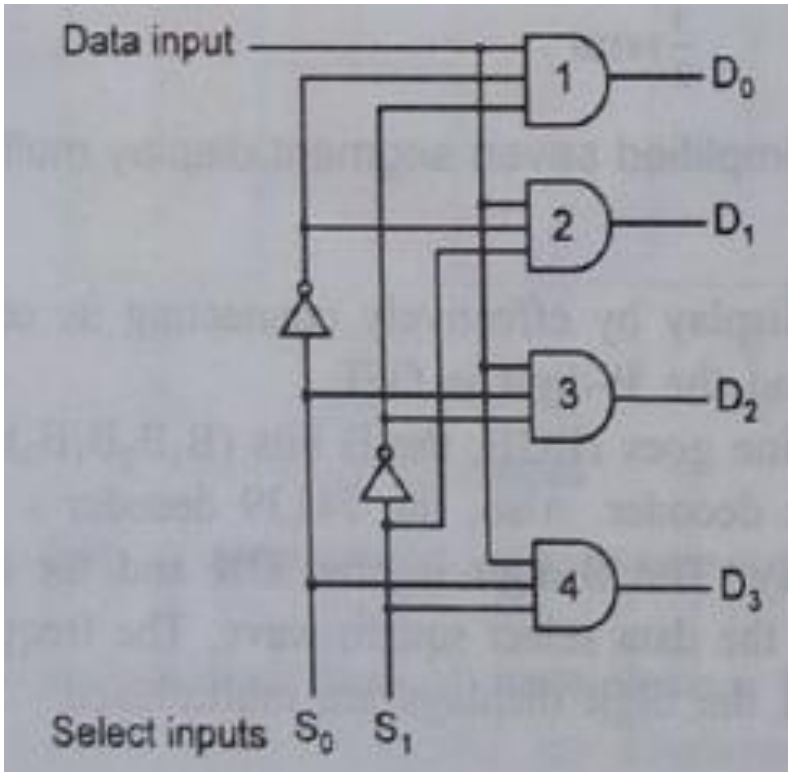
W = A'.B'.I

X = A.B'.I

Y = A'.B.I

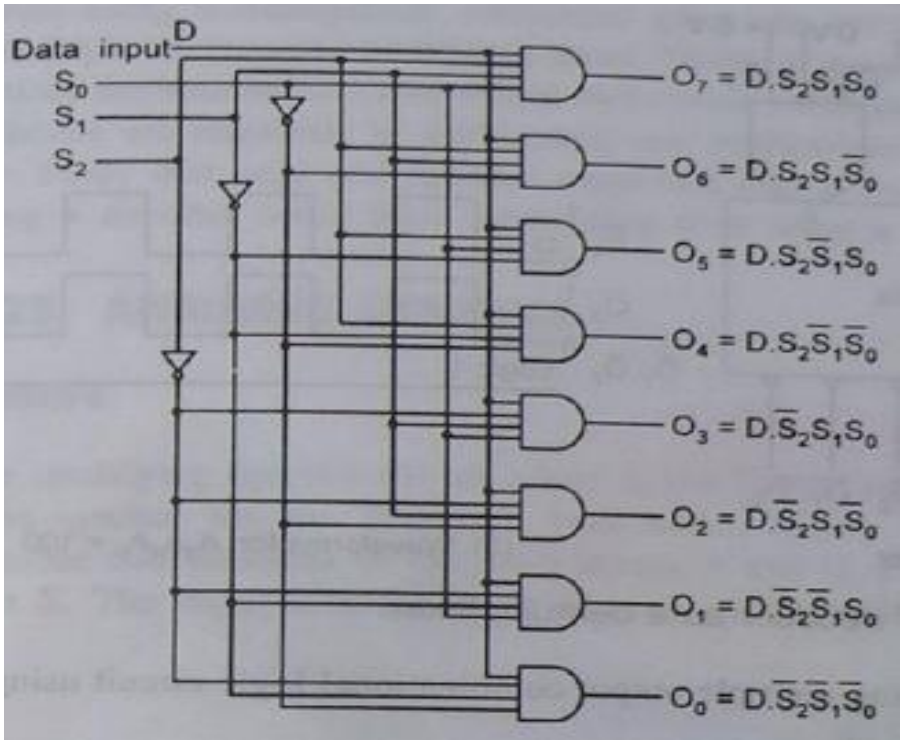
Z = A.B.I

1-line to 4-line Demultiplexer



Input	Select Lines	Output Lines
I	S_1 S_0	D_0 D_1 D_2 D_3
I	0 0	1 0 0 0
I	0 1	0 1 0 0
I	1 0	0 0 1 0
I	1 1	0 0 0 1

1-line to 8-line Demultiplexer



Data Inputs	Select Inputs			Outputs							
	S2	S1	S0	Q0.7	Q0.6	Q0.5	Q0.4	Q0.3	Q0.2	Q0.1	Q0.0
Di	0	0	0	0	0	0	0	0	0	0	Di
Di	0	0	1	0	0	0	0	0	0	Di	0
Di	0	1	0	0	0	0	0	0	Di	0	0
Di	0	1	1	0	0	0	0	Di	0	0	0
Di	1	0	0	0	0	0	Di	0	0	0	0
Di	1	0	1	0	0	Di	0	0	0	0	0
Di	1	1	0	0	Di	0	0	0	0	0	0
Di	1	1	1	Di	0	0	0	0	0	0	0

Designing logic circuits using multiplexers

Using an *n*-input Multiplexer:

- Use an *n*-input multiplexer to realize a logic circuit for a function with *n* minterms.
  - $m = 2^n$, where m = # of variables in the function
- Each minterm of the function can be mapped to an input of the multiplexer.
- For each row in the truth table, for the function, where the output is 1, set the corresponding input of the multiplexer to 1.
  - That is, for each minterm in the minterm expansion of the function, set the corresponding input of the multiplexer to 1.
- Set the remaining inputs of the multiplexer to 0.

Example: *Implement the following function with multiplexer.*

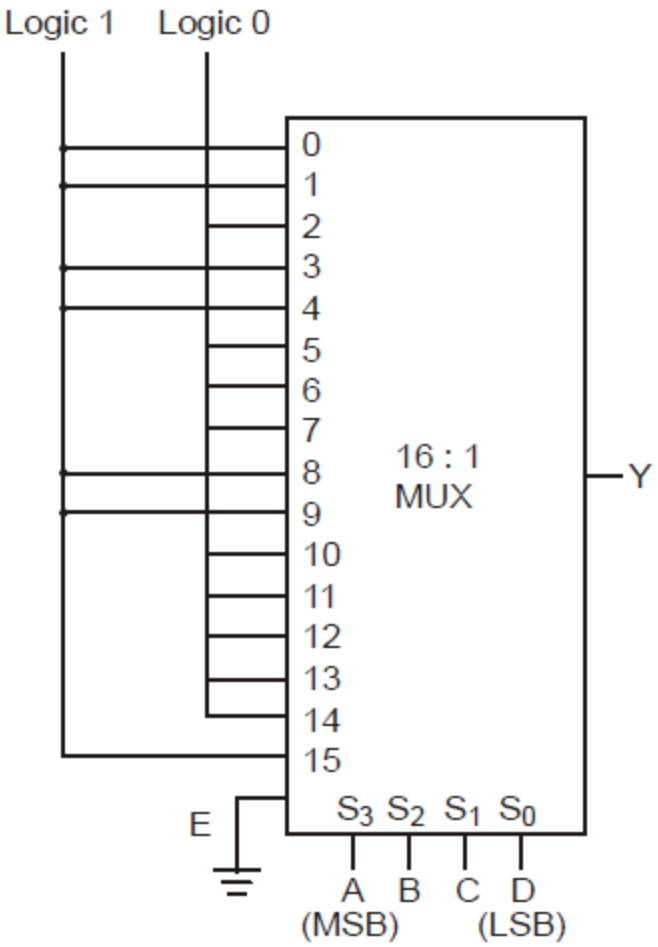
Y = F (A, B, C, D) = Σm (0, 1, 3, 4, 8, 9, 15)

Solution. STEP 1 : The input lines corresponding to each minterms (decimal number) are to be connected to logic 1.

Therefore input lines 0, 1, 3, 4, 8, 9, 15 have to be connected to logic 1.

STEP 2 : All other input lines except 0, 1, 3, 4, 8, 9, 15 are to be connected to logic 0.

STEP 3 : The control inputs A, B, C, D are to be applied to select inputs.



Using an *(n / 2)*-input Multiplexer:

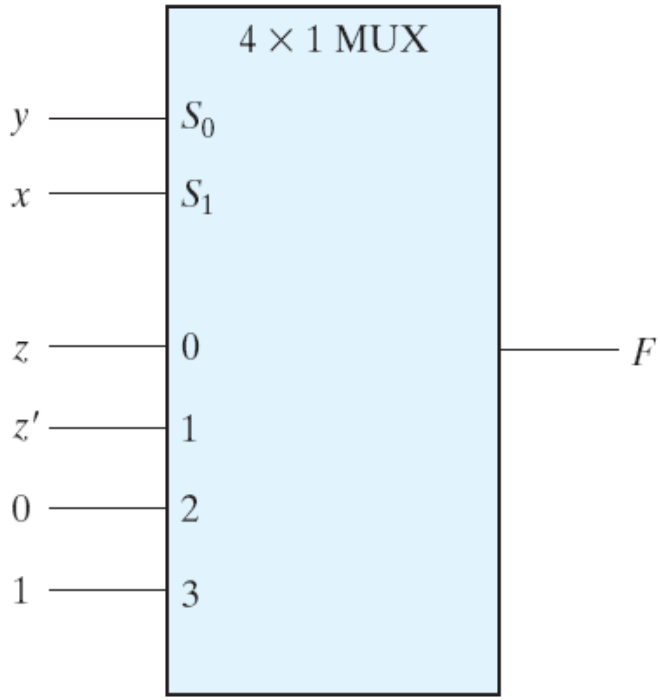
- Use an *(n / 2)*-input multiplexer to realize a logic circuit for a function with *n* minterms.
  - $m = 2^n$, where m = # of variables in the function
- Group the rows of the truth table, for the function, into *(n / 2)* pairs of rows.
  - Each pair of rows represents a product term of *(m – 1)* variables.
  - Each pair of rows can be mapped to a multiplexer input.
- Determine the logical function of each pair of rows in terms of the m^{th} variable.
  - If the m^{th} variable, for example, is x, then the possible values are x, x', 0, and 1.

Using an *(n / 2)*-input Mux

Example: F(x,y,z) = Σm(1, 2, 6, 7)

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

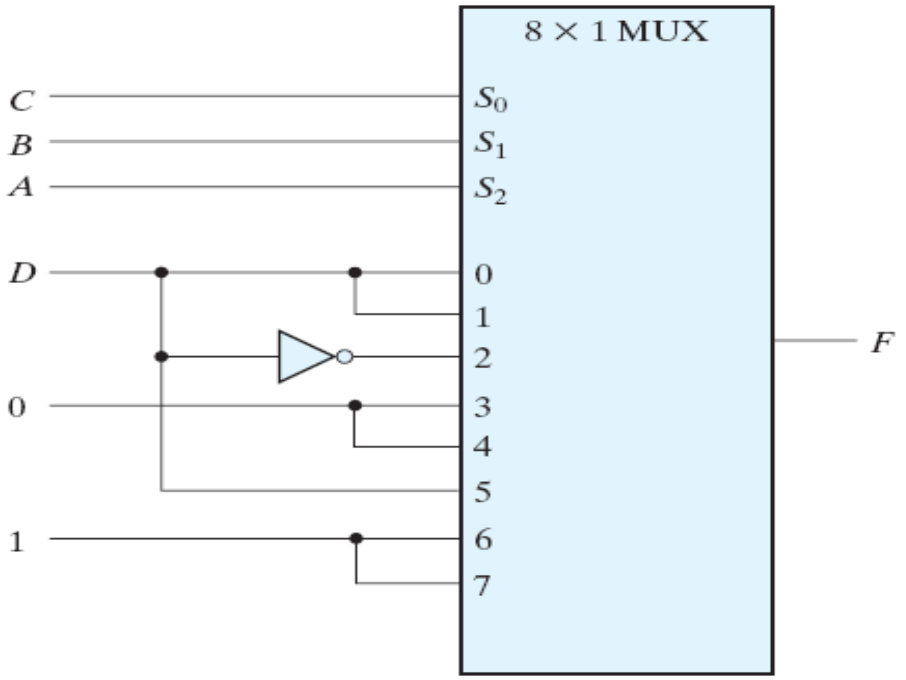


(b) Multiplexer implementation

Using an *(n / 2)*-input Mux

Example: F(A,B,C,D) = Σm(1,3,4,11,12,13,14,15)

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Implement a full adder with a decoder and two OR-gates.

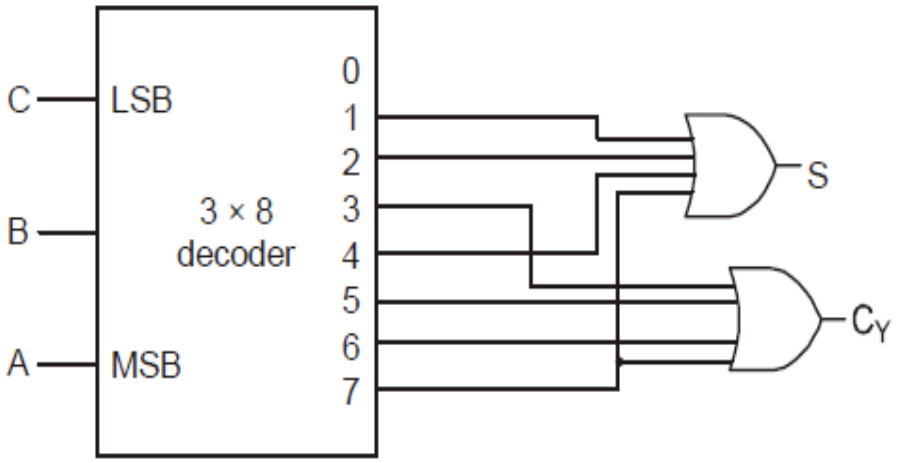
Solution. From the previous example we note that expression for summation is given by

$$S\ (A, B, C) = \Sigma m\ (1, 2, 4, 7)$$

and expression for carry is given by

$$C_F\ (A, B, C) = \Sigma m\ (3, 5, 6, 7)$$

The combinational logic of full adder can be implemented with due help of 3-line to 8-line decoder/1:8 demultiplexer as shown in Fig.



Full adder implementation using 3 × 8 decoder

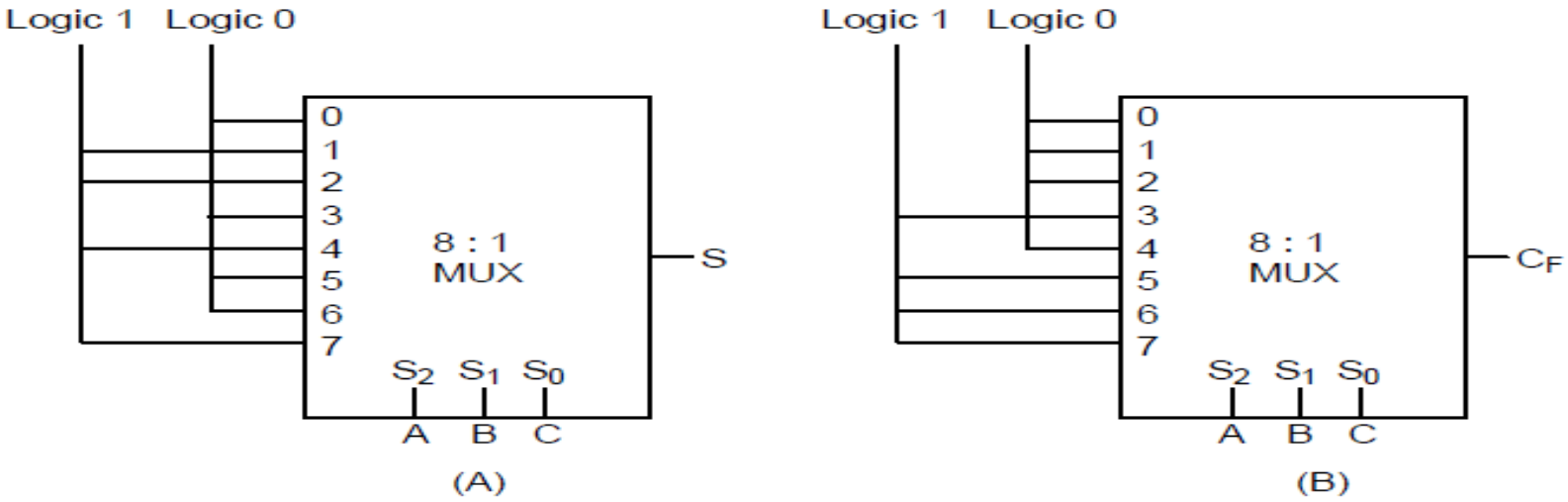
Design a full adder using 8:1 multiplexer.

Solution. The truth table of a full adder is given as

A	B	C	S	C_F
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

$$S\ (A,\ B,\ C) = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC = \Sigma m\ (1,\ 2,\ 4,\ 7)$$

$$C_F\ (A,\ B,\ C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = \Sigma m\ (3,\ 5,\ 6,\ 7)$$



Full adder implementation using 8:1 multiplexer

Implement the following function with a 4×1 multiplexer.

$$Y = F(A, B, C) = \Sigma m\ (1, 3, 5, 6)$$

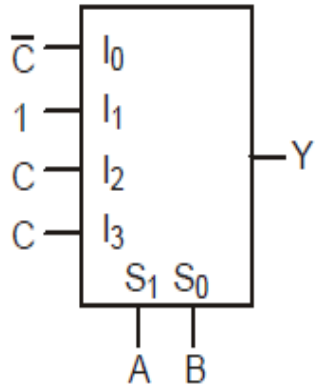
Solution. Given

$$Y = F(A, B, C) = \Sigma m\ (1, 3, 5, 6)$$

$$= \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$$

We use the A and B variables as data select inputs. We can use the above equation to construct the table shown in Fig. The residues are what is "left over" in each minterm when the "address" variables are taken away. To implement this circuit, we connect I_0, I_1 and I_2 to C and I_3 to $\overline{C}$

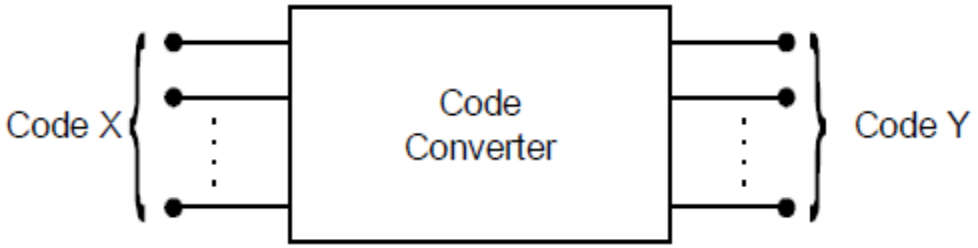
Input	*"Address"*	*Other variables (residues)*
I_0	$\overline{A}\,\overline{B}$	C
I_1	$\overline{A}B$	C
I_2	$A\overline{B}$	C
I_3	AB	$\overline{C}$



A 4-line to 1-line MUX implementation of a function of 3 variables

Code Converters

Coding was defined as the use of groups of bits to represent items of information that are multi-valued. Assigning each item of information a unique combination of bits makes a transformation of the original information. This we recognize as information being processed into another form. Moreover, we have seen that there are many coding schemes exist. Different digital systems may use different coding schemes. It is sometimes necessary to use the output of one system as the input to other.
Therefore a sort of code conversion is necessary between the two systems to make them compatible for the same information.

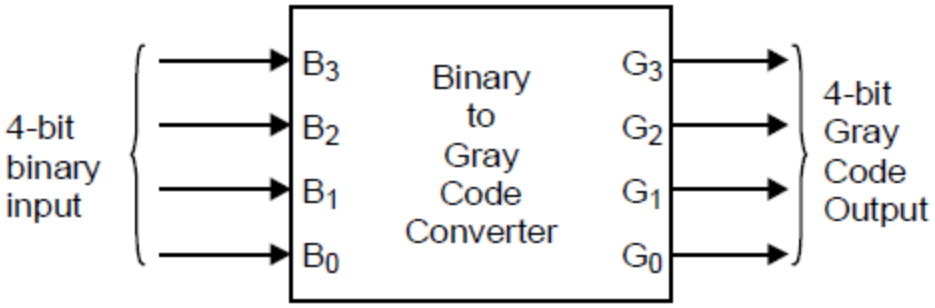


Code converter

A code converter is a combinational logic circuit that changes data presented in one type of binary code to another type of binary code.' A general block diagram of a code converter is shown in Fig.

Example: 4-bit Binary to Gray code conversion

It has four inputs (B3 B2 B1 B0) representing 4-bit binary numbers and four outputs (G3 G2 G1 G0) representing 4-bit gray code.



Now all the gray outputs distantly solved with respect to binary inputs From the truth table; the logic expressions for the gray code outputs can be written as

$$G_3 = \Sigma\,(8, 9, 10, 11, 12, 13, 14, 15)$$
$$G_2 = \Sigma\,(4, 5, 6, 7, 8, 9, 10, 11)$$
$$G_1 = \Sigma\,(2, 3, 4, 5, 10, 11, 12, 13)$$
$$G_0 = \Sigma\,(1, 2, 5, 6, 9, 10, 13, 14).$$

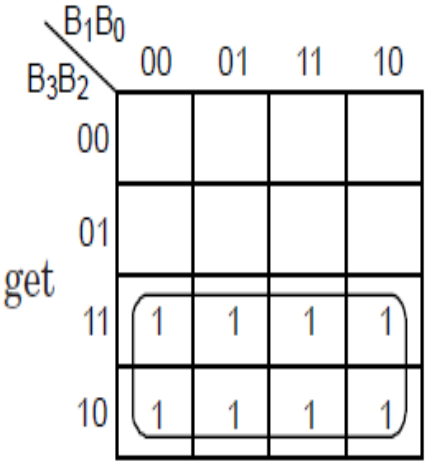
Binary Inputs				*Gray code Outputs*			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

The above expressions can be simplified using K-map

Map for G_3:

From the octet, we get

$G_3 = B_3$

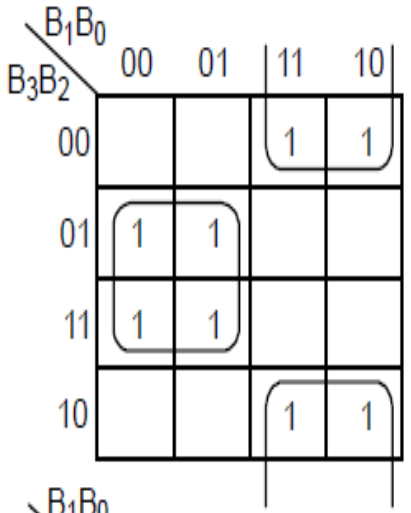


Map for G_1:

From the two quads, we get

$G_1 = B_2 B_1' + B_2' B_1$

$= B_2 \oplus B_1$

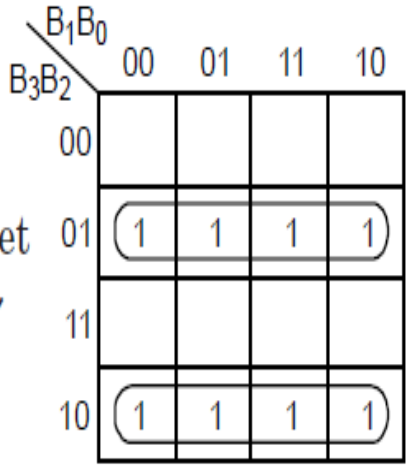


Map for G_2:

From the two quads, we get

$G_2 = B_3' B_2 + B_3 B_2'$

$= B_3 \oplus B_2.$

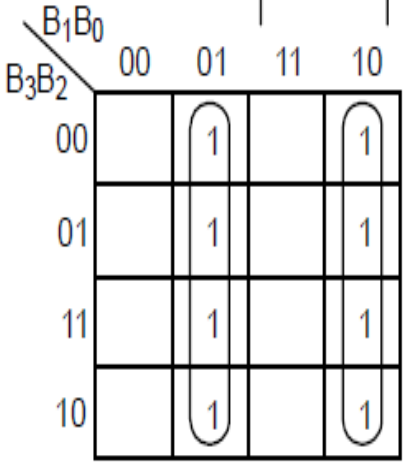


Map for G_0:

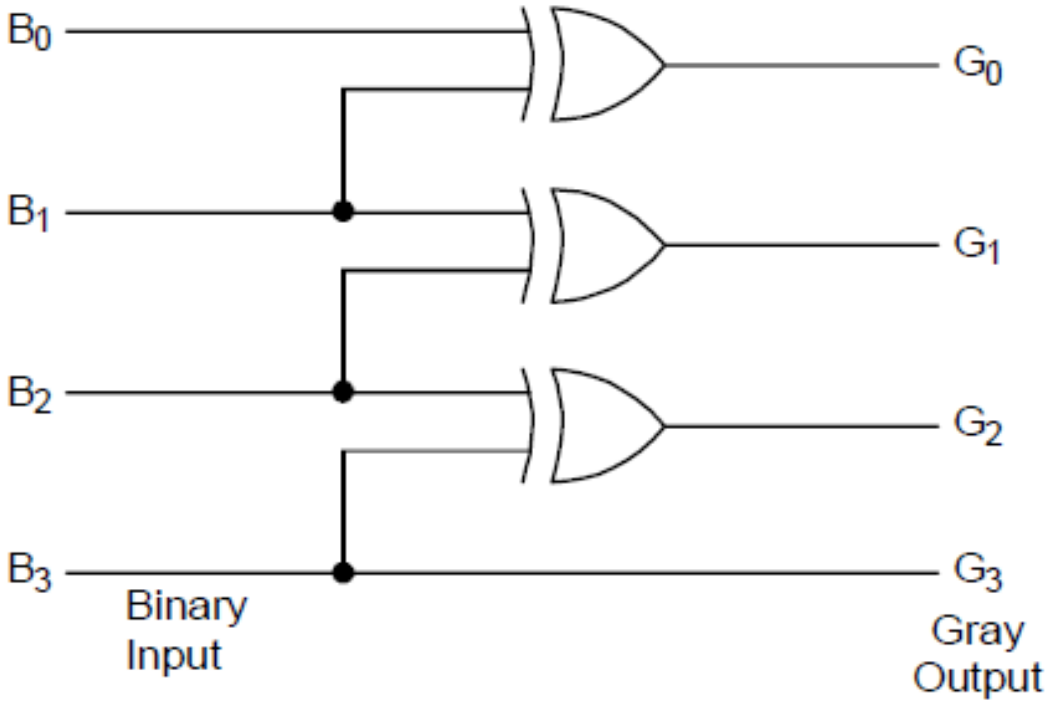
From the two quads, we get

$G_0 = B_1' B_0 + B_1 B_0'$

$= B_1 \oplus B_0.$

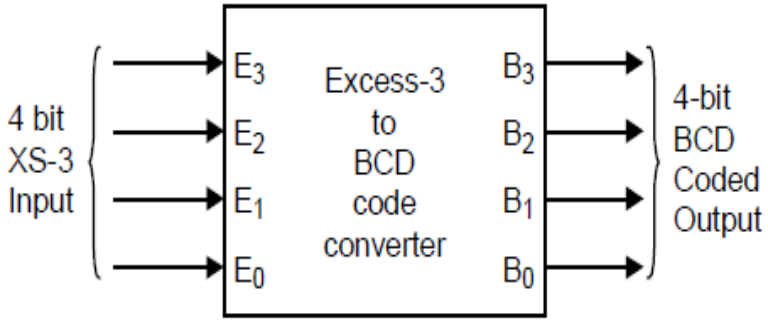


Now the above expressions can be implemented using X-OR gates to yield the desired code converter circuit shown in Fig.



Example: XS-3 to BCD code converter

The block diagram of an XS-3 to BCD code converter is shown in Fig. It has four inputs (E3, E2, E1, E0) representing 4 bit XS-3 number and four outputs (B3B2 B1 B0) representing 4-bit BCD code.



XS-3 codes are obtained from BCD code by adding 3 to each coded number. Moreover 4 binary variables may have 16 combinations, but only 10 are listed. The six not listed are don't care-combinations. Since they will never occur, we are at liberty to assign to the output variable either a 1 or a 0, whichever gives a simpler circuit. In this particular example, the unused i/o combinations are listed below the truth table.

Min Terms	*Excess-3 Inputs*				*BCD Outputs*				*Decimal Equivalent*
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0	
m_3	0	0	1	1	0	0	0	0	0
m_4	0	1	0	0	0	0	0	1	1
m_5	0	1	0	1	0	0	1	0	2
m_6	0	1	1	0	0	0	1	1	3
m_7	0	1	1	1	0	1	0	0	4
m_8	1	0	0	0	0	1	0	1	5
m_9	1	0	0	1	0	1	1	0	6
m_{10}	1	0	1	0	0	1	1	1	7
m_{11}	1	0	1	1	1	0	0	0	8
m_{12}	1	1	0	0	1	0	0	1	9
	Unused I/Ps				Outputs				
m_0	0	0	0	0	x	x	x	x	
m_1	0	0	0	1	x	x	x	x	
m_2	0	0	1	0	x	x	x	x	
m_{13}	1	1	0	1	x	x	x	x	
m_{14}	1	1	1	0	x	x	x	x	
m_{15}	1	1	1	1	x	x	x	x	

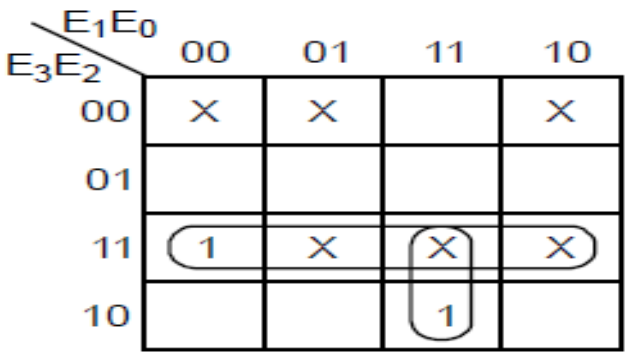
$B_3 = \Sigma\ (m_{11}, m_{12}),\ d\ (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$

$B_2 = \Sigma\ (m_7, m_8, m_9, m_{10}),\ d\ (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$

$B_1 = \Sigma\ (m_5, m_6, m_9, m_{10}),\ d\ (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$

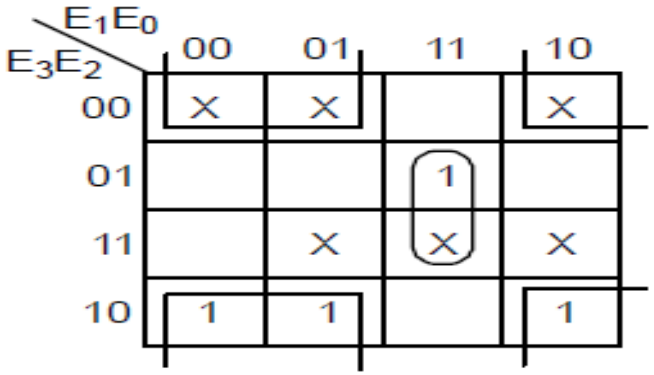
$B_0 = \Sigma\ (m_4, m_6, m_8, m_{10}, m_{12}),\ d\ (m_0, m_1, m_2, m_{13}, m_{14}, m_{15}).$

Map for B_3



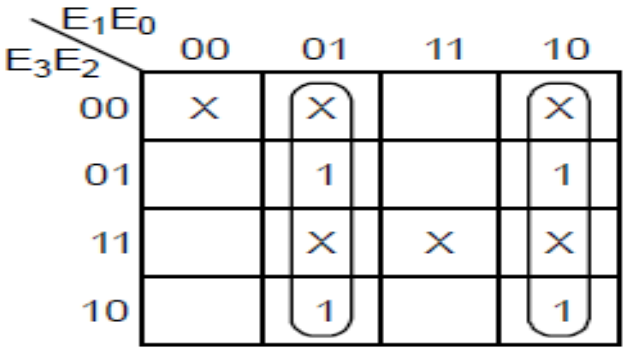
$$B_3 = E_3 E_2 + E_3 E_1 E_0$$

Map for B_2

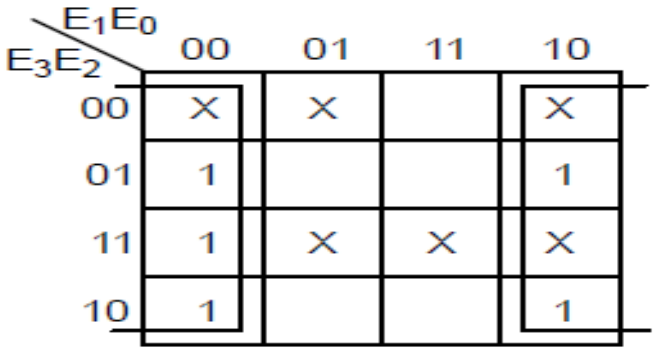


$$B_2 = E_2' E_0' + E_2 E_1 E_0 + E_2' E_1'$$

Map for B_1



Map for B_0



$$B_0 = E_0'$$

$$B_1 = E_1' E_0 + E_1 E_0'$$
$$= E_1 \oplus E_0$$
$$B_3 = E_3 E_2 + E_3 E_1 E_0$$
$$B_2 = E_2' E_0 + E_2 E_1 E_0 + E_2' E_1'$$
$$B_1 = E_1 \oplus E_0$$
$$B_0 = E_0'$$

The expressions for BCD outputs (B3 B2 B1 B0) can be implemented for terms of inputs (E3 E2 E1 E0) to form a XS-3 to BCD code converter circuit. The implementation is left as an exercise.

Design of an SOP Circuit to Detect the Decimal Numbers 0, 2, 4, 6, and 8 in a 4-bit 5211 BCD Code Input

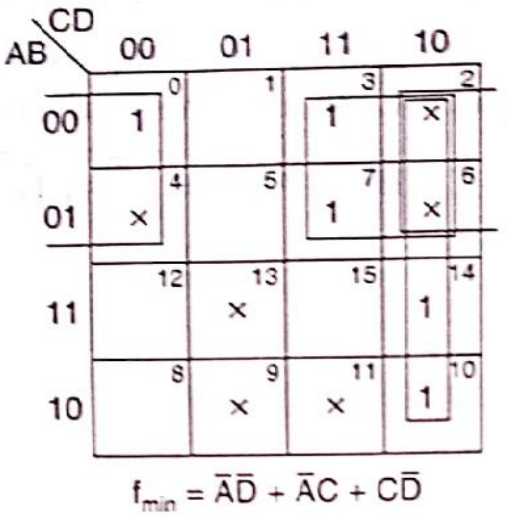
The input to the SOP circuit is a 5211 BCD code. Let it be ABCD. It is a 4-bit input. Therefore, there are 16 possible combinations of inputs, out of which only the 10 combinations shown in the truth table of Figure 4.39a are used to code the decimal digits in 5211 code. The remaining 6 combinations 0010, 0100, 0110, 1001, 1011, and 1101 are invalid. So, the corresponding outputs are don't cares (i.e. minterms 2, 4, 6, 9, 11, and 13 are don't cares). Looking at the truth table of the SOP circuit shown in Figure we observe that the output is 1 for the input combinations corresponding to minterms 0, 3, 7, 10, 14 (i.e. corresponding to 5211 code of decimal numbers 0, 2, 4, 6, and 8). So the problem may be stated as $F = \Sigma\, m(0, 3, 7, 10, 14) + d(2, 4, 6, 9, 11, 13)$

The K-map, its minimization, the minimal expression obtained from it and the realization of the minimal expression in NAND logic are shown in Figure

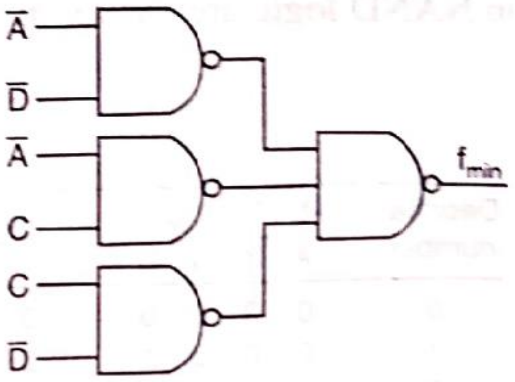
$$f_{min} = \bar{A}\bar{D} + \bar{A}C + C\bar{D} = \overline{\overline{\bar{A}\bar{D}} \cdot \overline{\bar{A}C} \cdot \overline{C\bar{D}}}$$

Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

Truth table



K-map



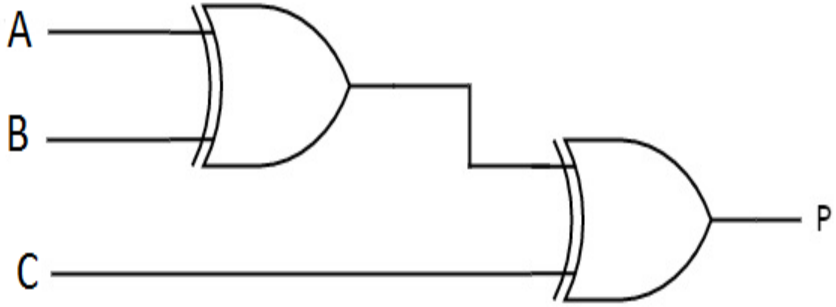
Logic diagram

PARITY BIT GENERATOR

Even parity generator

3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Functional Table

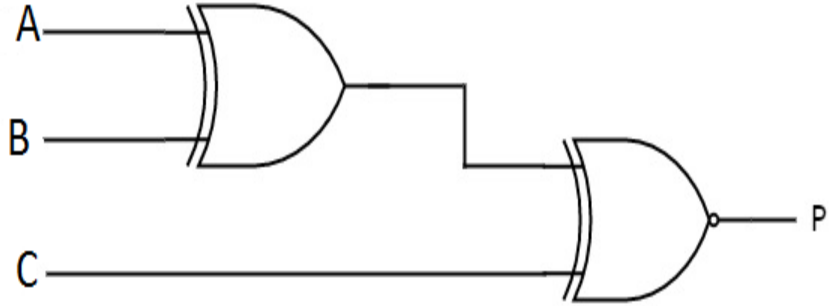


Logic Diagram

Odd parity generator

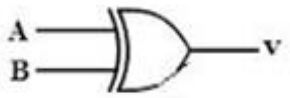
3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Functional Table

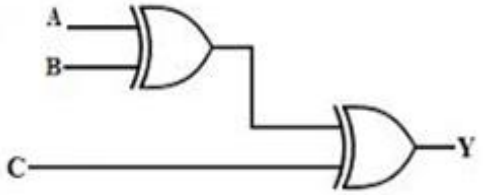


Logic Diagram

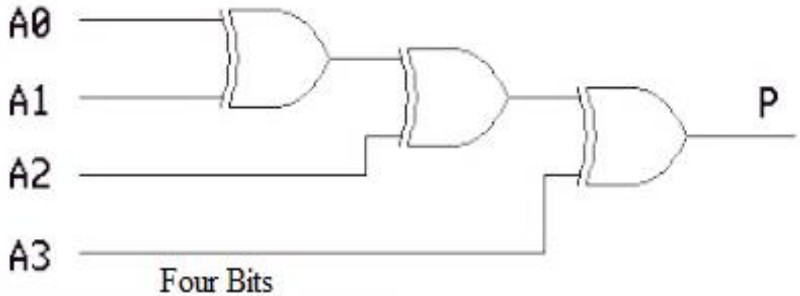
This can be drawn by finding Boolean Expression through 3 variable K-Map



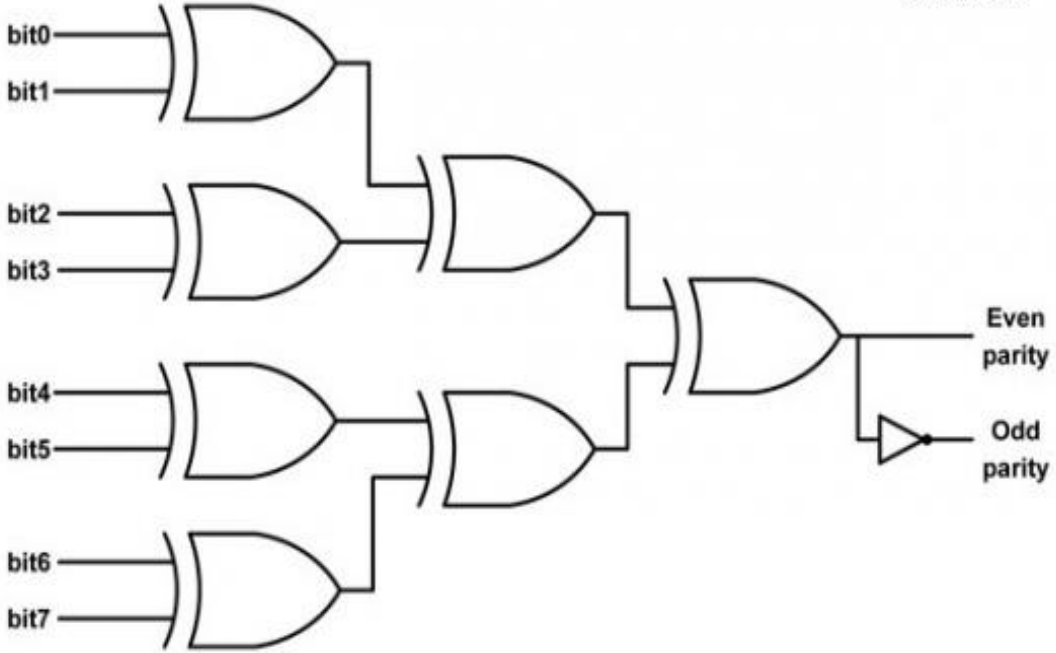
Two Bits



Three Bits

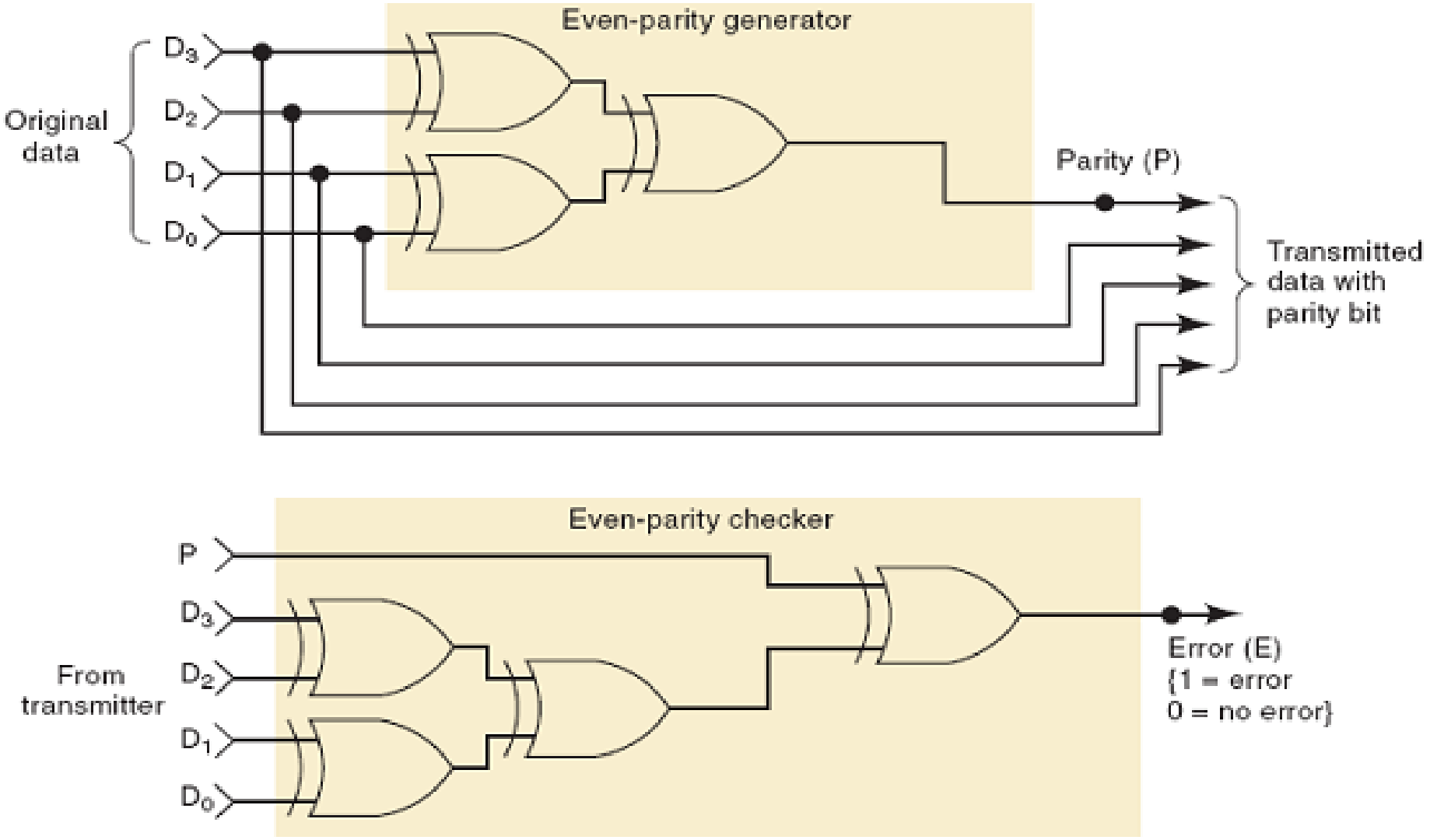


Four Bits



INPUT	EVEN PARITY	ODD PARITY
01010101	0	1
11010101	1	0

PARITY BIT GENERATOR/CHECKER

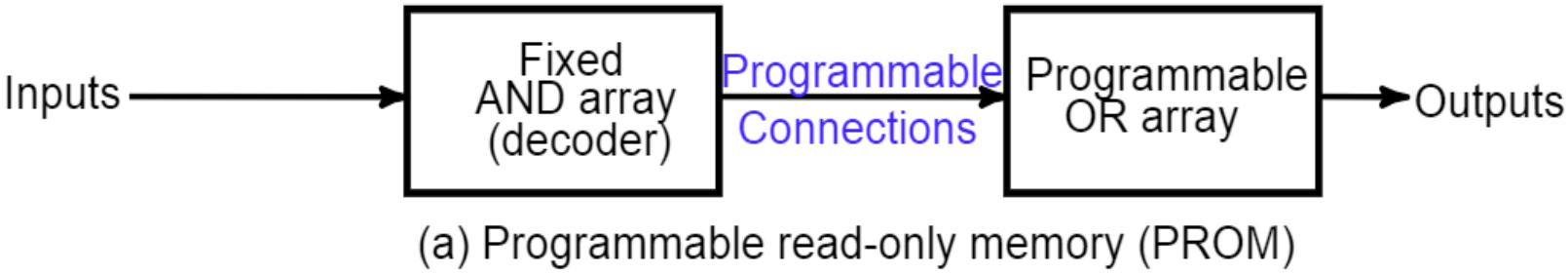


PROGRAMMABLE LOGIC DEVICES(PLD's)

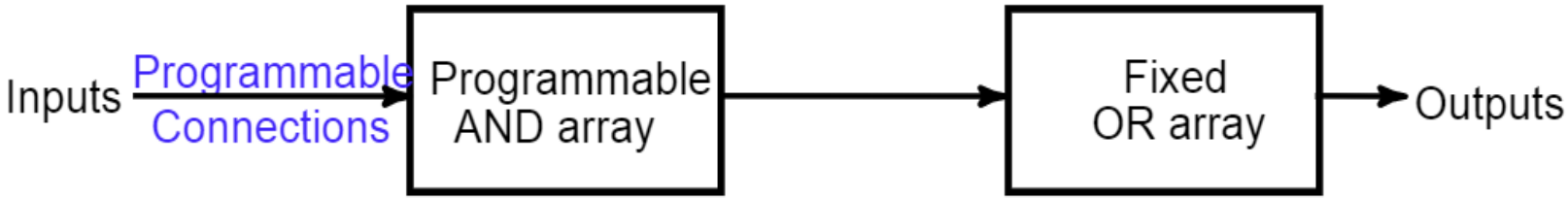
- Contains an array of AND gates and OR gates

- Advantages:
  - Low cost
  - Design a larger circuit
  - Reprogramming (Modify the design)

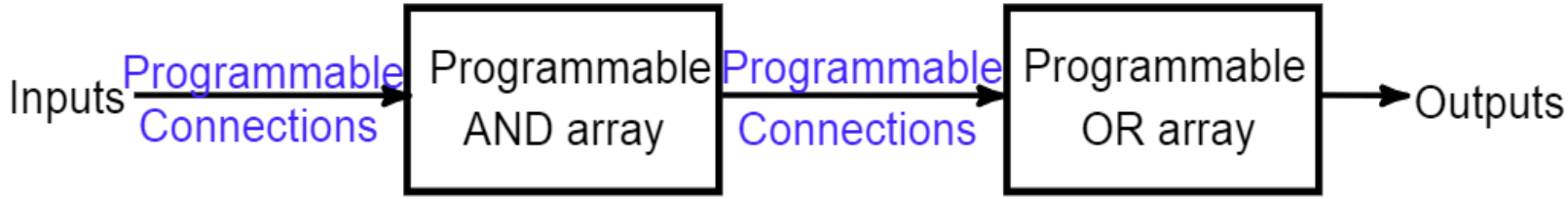
TYPES OF PLD's:



(a) Programmable read-only memory (PROM)



(b) Programmable array logic (PAL) device



(c) Programmable logic array (PLA) device

PROM

- Contains an fixed AND array & Programmable OR array gates.

- AND gate will generate 2^n product terms by using 2^n AND gates having n inputs each by using $nx2^n$ decoder. So, this decoder generates 'n' **min terms**.

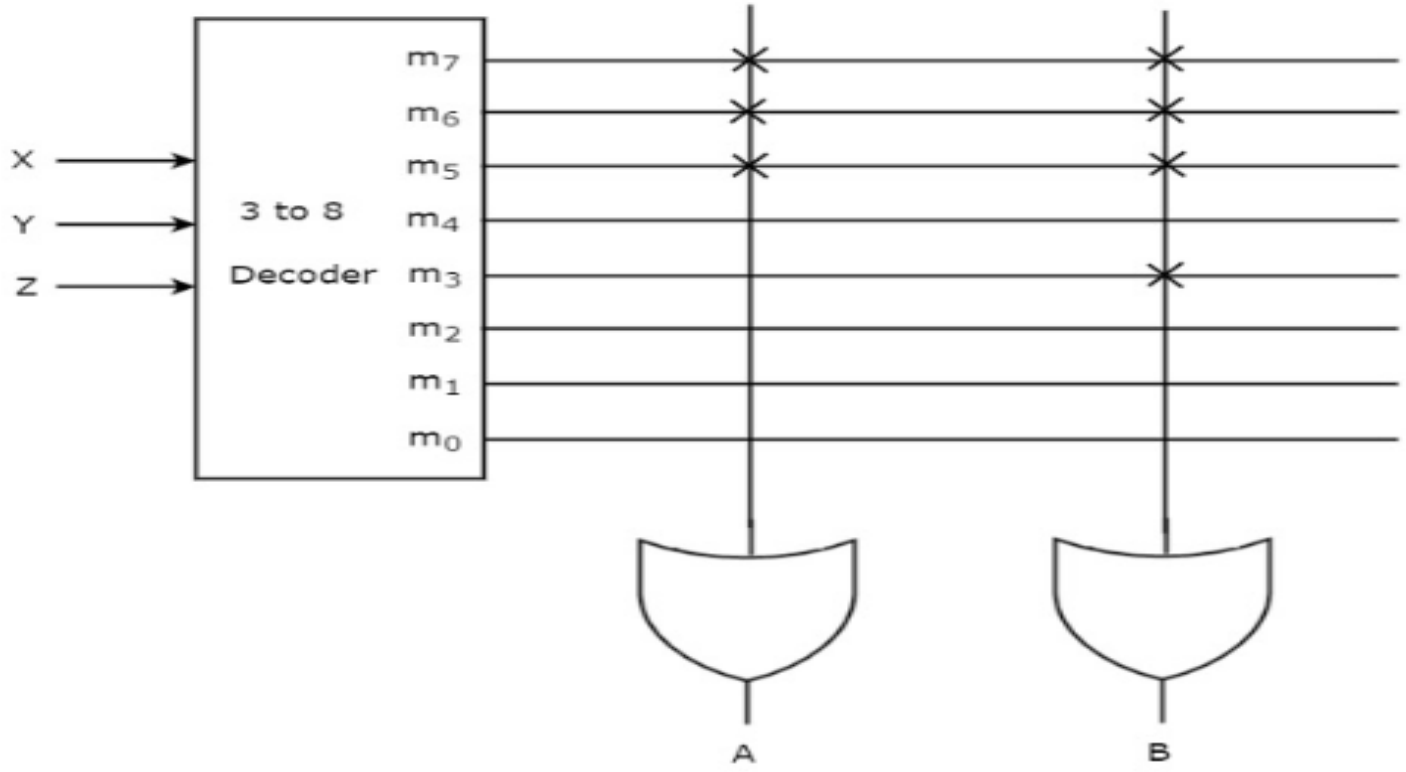
- OR GATE: we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of **sum of min terms**

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. The corresponding **PROM** is shown in the following figure.



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.

PAL

- Contains an Programmable AND array & Fixed OR array gates.

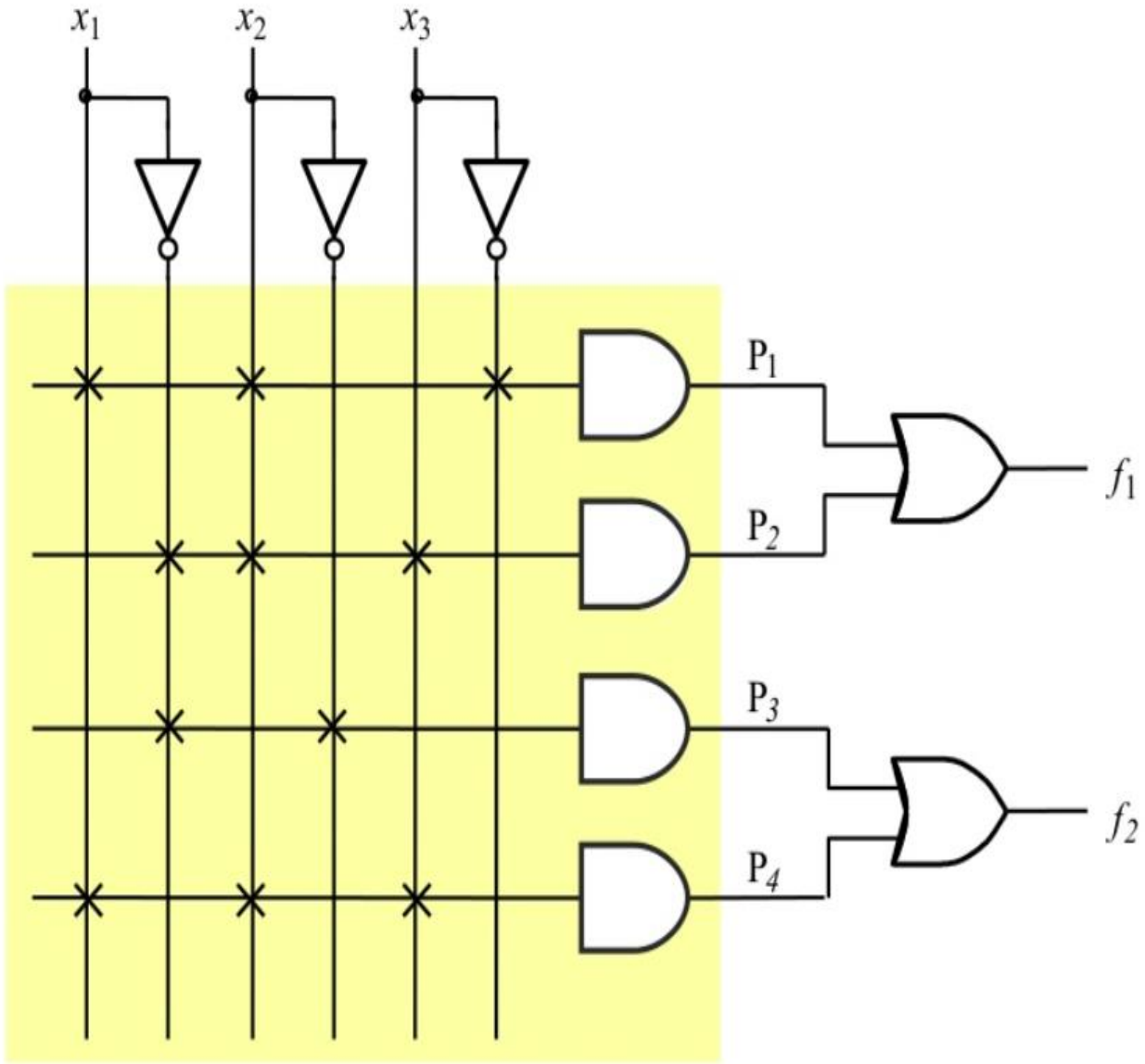
- Each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

- OR GATE: 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**.

$f_1 = x_1x_2x_3' + x_1'x_2x_3$

$f_2 = x_1'x_2' + x_1x_2x_3$

P1,P2,P3,P4 are four product terms



PAL PROGRAMMING TABLE

Product Terms	Inputs			Outputs	
	X_1	X_2	X_3	F_1	F_2
$P_1(X_1X_2X_3')$	1	1	0		
$P_2(X_1'X_2X_3)$	0	1	1		
$P_3(X_1'X_2')$	0	0	-		
$P_4(X_1X_2X_3)$	1	1	1		

$$f_1 = x_1x_2x_3' + x_1'x_2x_3$$

$$f_2 = x_1'x_2' + x_1x_2x_3$$

P1,P2,P3,P4 are four product terms

PAL PROGRAMMING TABLE

Product Terms	Inputs			Outputs	
	X_1	X_2	X_3	F_1	F_2
$P_1(X_1X_2X_3')$	1	1	0	1	-
$P_2(X_1'X_2X_3)$	0	1	1	1	-
$P_3(X_1X_2')$	0	0	-	-	1
$P_4(X_1X_2X_3)$	1	1	1	-	1

$f_1 = x_1x_2x_3'+x_1'x_2x_3$

$f_2 = x_1'x_2'+x_1x_2x_3$

P1,P2,P3,P4 are four product terms

PLA

- Contains an Programmable AND array & Programmable OR array gates.

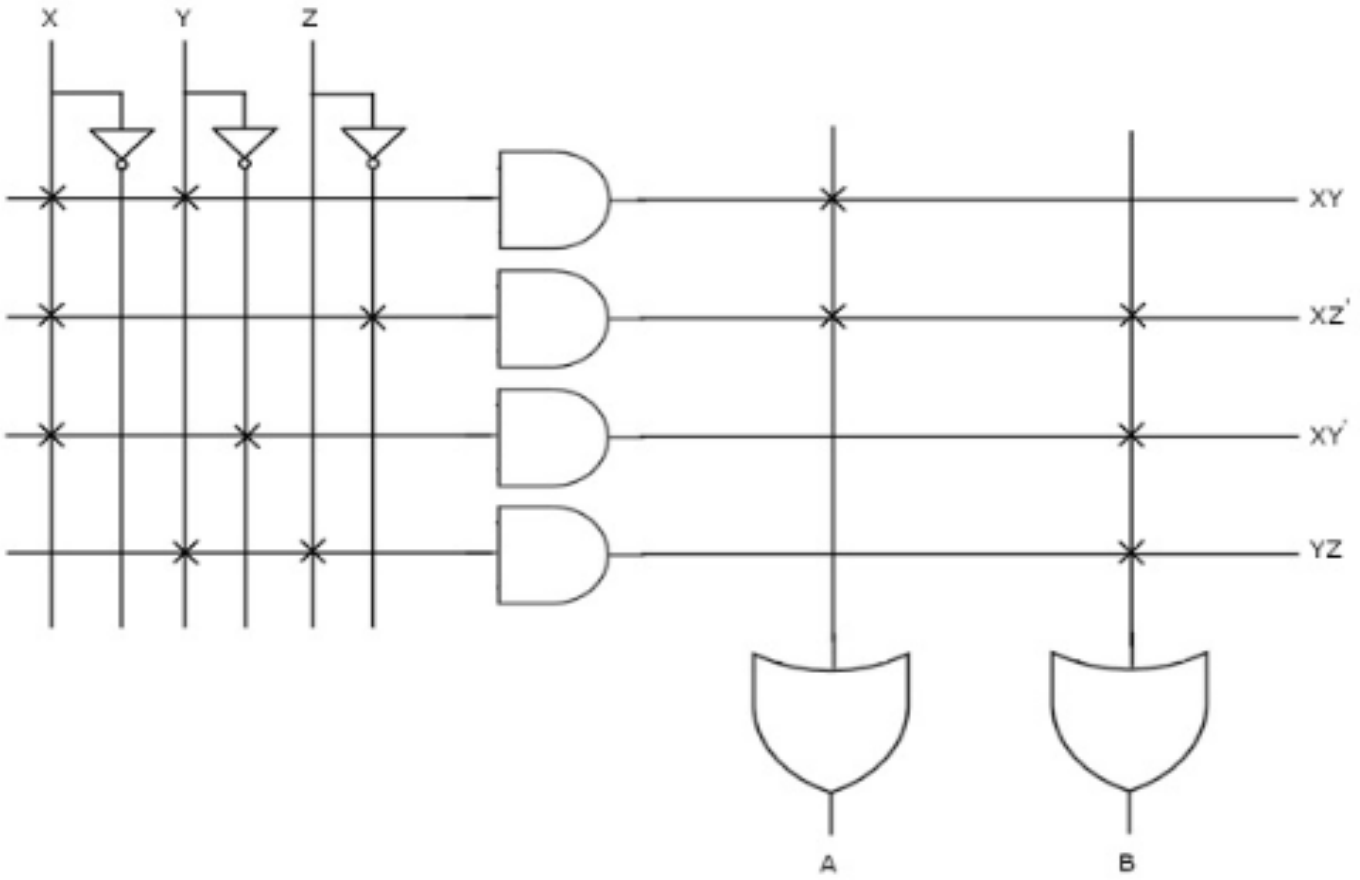
- Each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

- OR GATE: 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 PLA will be in the form of **sum of min terms**

Let us implement the following Boolean functions using PLA.

$$A = XY + XZ'$$

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



PLA PROGRAMMING TABLE

Product Terms	Inputs			Outputs	
	X	Y	Z	A	B
$P_1(XY)$	1	1	-	1	-
$P_2(XZ')$	1	-	0	1	1
$P_3(XY')$	1	0	-	-	1
$P_4(YZ)$	-	1	1	-	1

$$A = XY + XZ'$$

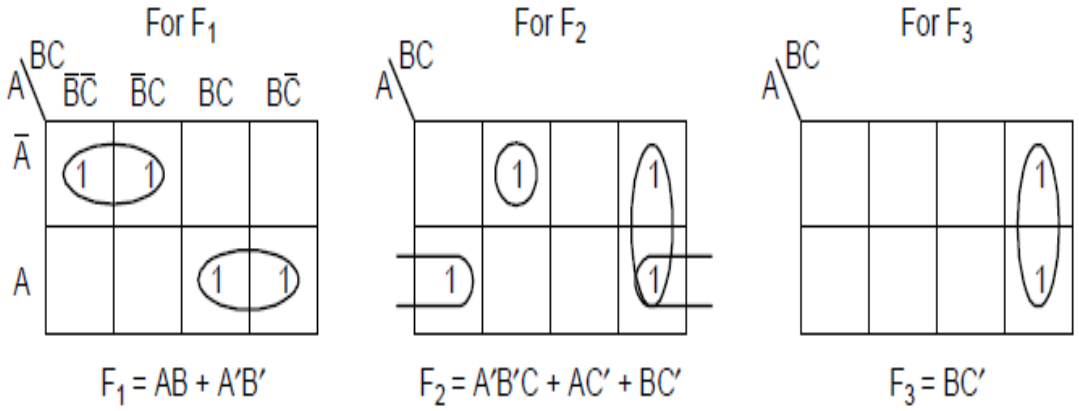
$$B = XY' + YZ + XZ'$$

Implement the following function using PLA and draw its Programming Table

$F_1(A, B, C) = \sum(0,1,6,7)$
$F_2(A, B, C) = \sum(1,2,4,6)$
$F_3(A, B, C) = \sum(2,6)$

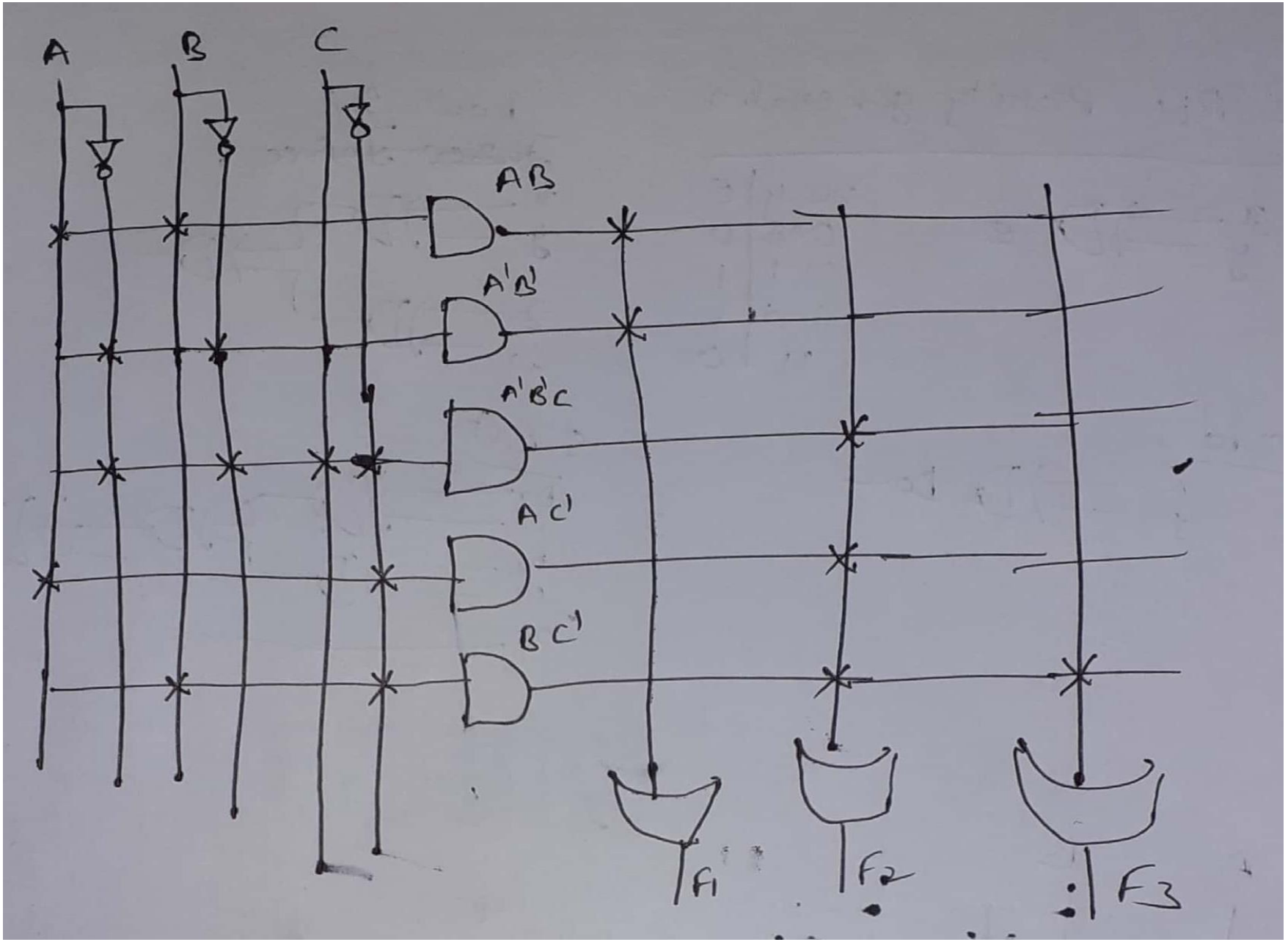
Solution:

Simplify functions using K-map



Product term	Inputs			Outputs		
	A	B	C	F_3	F_2	F_1
1	0	0	1	-	1	-
2	1	-	0	-	1	-
3	-	1	0	1	1	-
4	0	0	-	-	-	1
5	1	1	-	-	-	1

PLA Programming Table


A
B
C
AB
A'B'
A'BC
AC'
BC'
F1
F2
F3

Implement the following functions using PAL.

$F_1(A, B, C) = \sum(0,1,6,7)$
$F_2(A, B, C) = \sum(1,2,4,6)$
$F_3(A, B, C) = \sum(2,6)$

Simplify functions using K-map

For F_1

A \ BC	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	1	1		
A			1	1

$F_1 = AB + A'B'$

For F_2

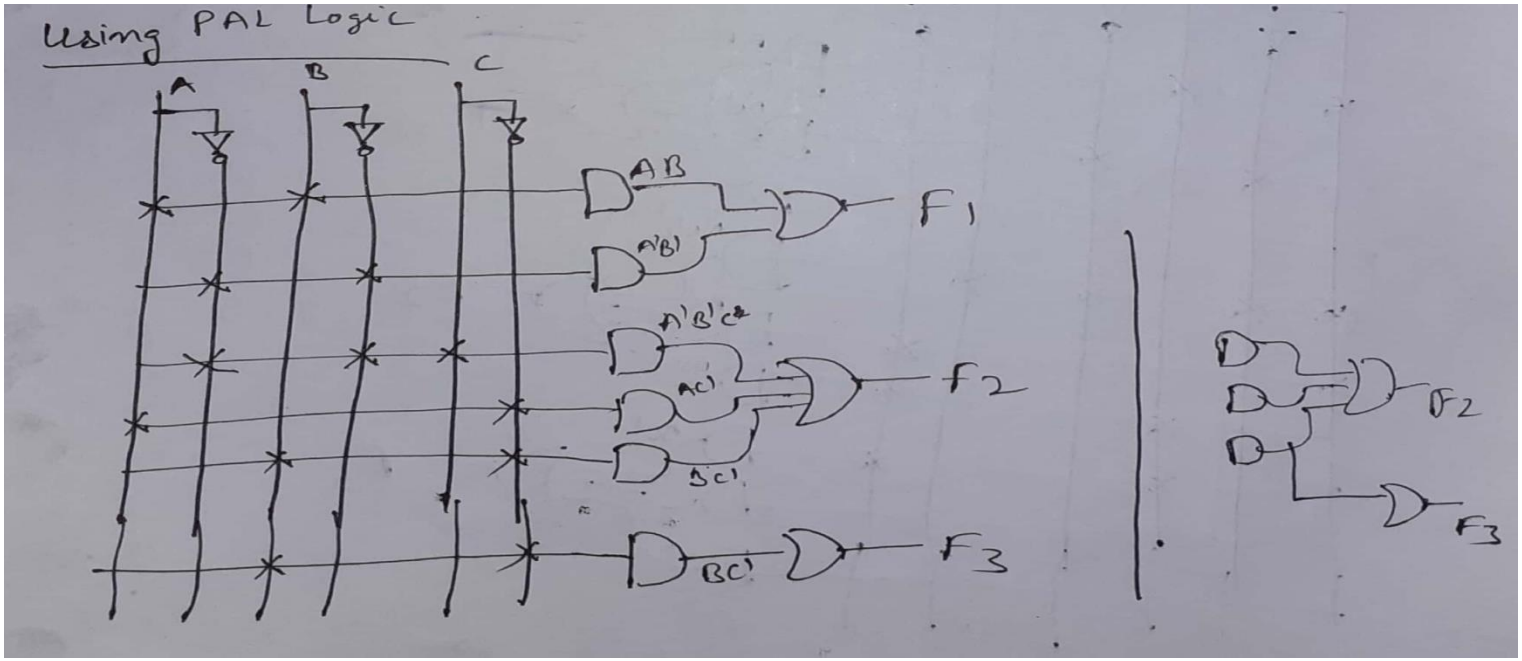
A \ BC				
		1		1
	1			1

$F_2 = A'B'C + AC' + BC'$

For F_3

A \ BC				
				1
				1

$F_3 = BC'$

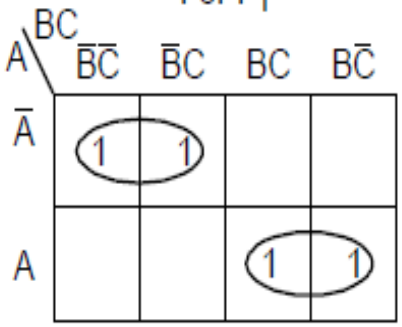


Implement the following functions using 3-input, 3 product terms and 3 output PAL.

$F_1(A, B, C) = \sum(0,1,6,7)$
$F_2(A, B, C) = \sum(1,2,4,6)$
$F_3(A, B, C) = \sum(2,6)$

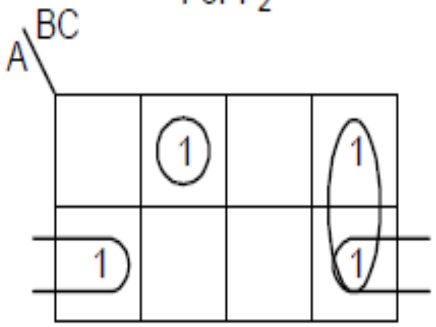
Simplify functions using K-map

For F_1



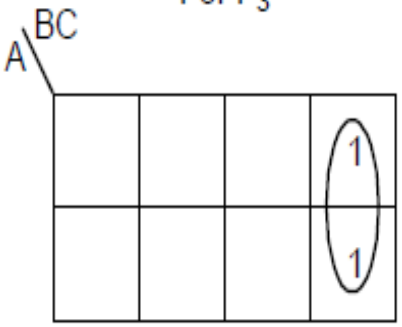
$F_1 = AB + A'B'$

For F_2

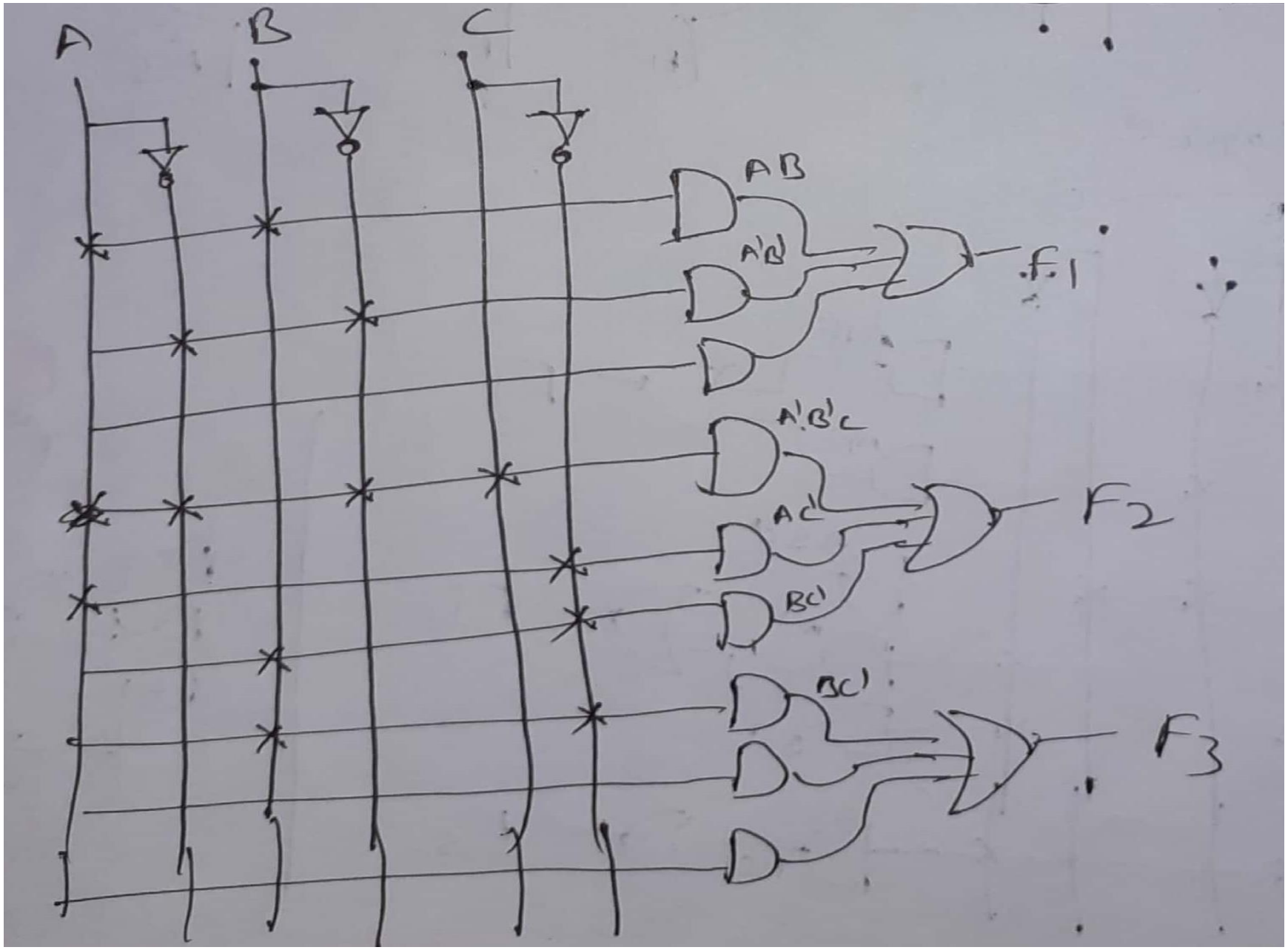


$F_2 = A'B'C + AC' + BC'$

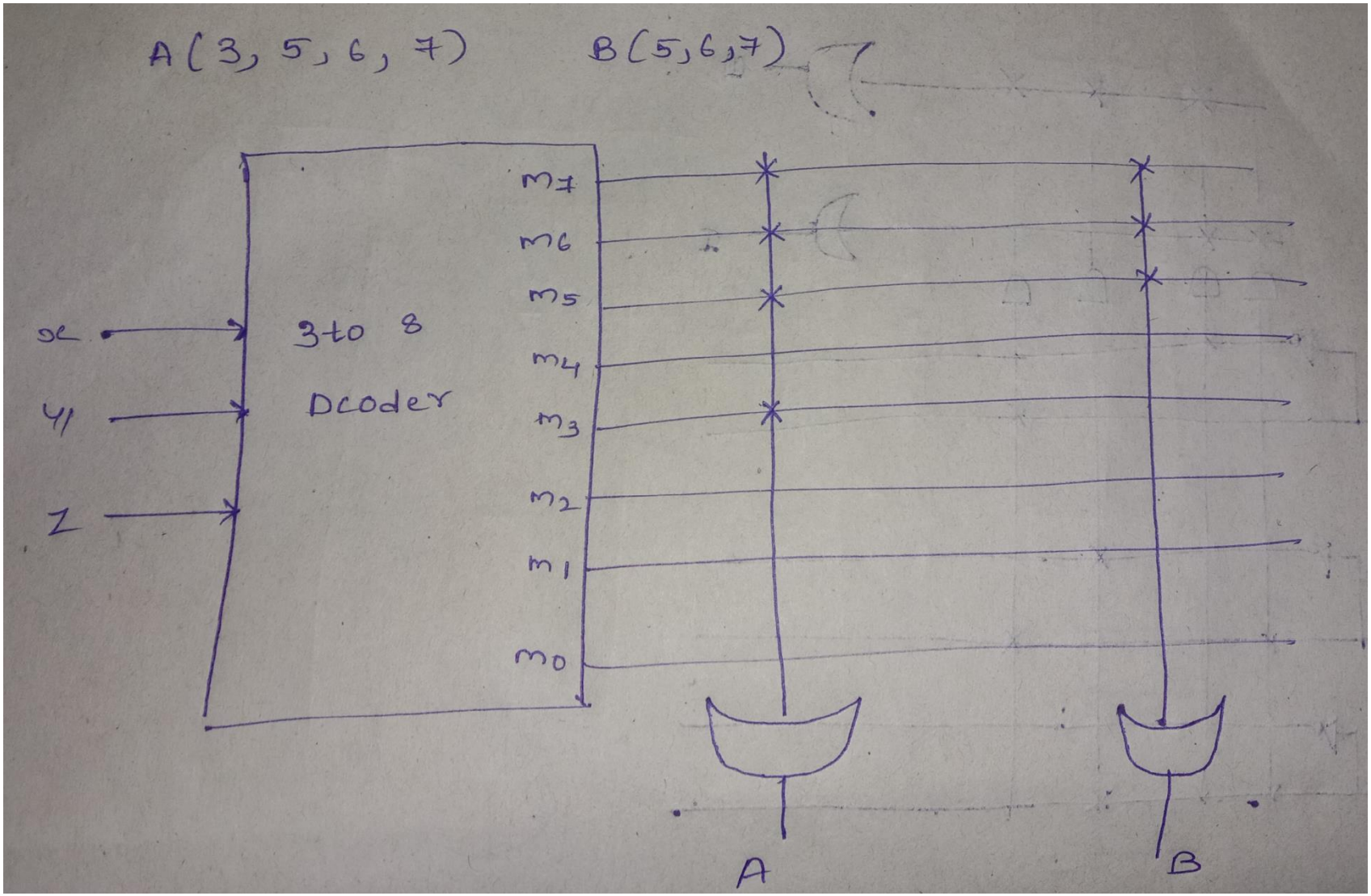
For F_3



$F_3 = BC'$


A
B
C
AB
A'B'
f1
A'B'C
AC'
F2
BC'
BC'
F3

Implement the following functions using PROM.



PLA

A(3,5,6,7)

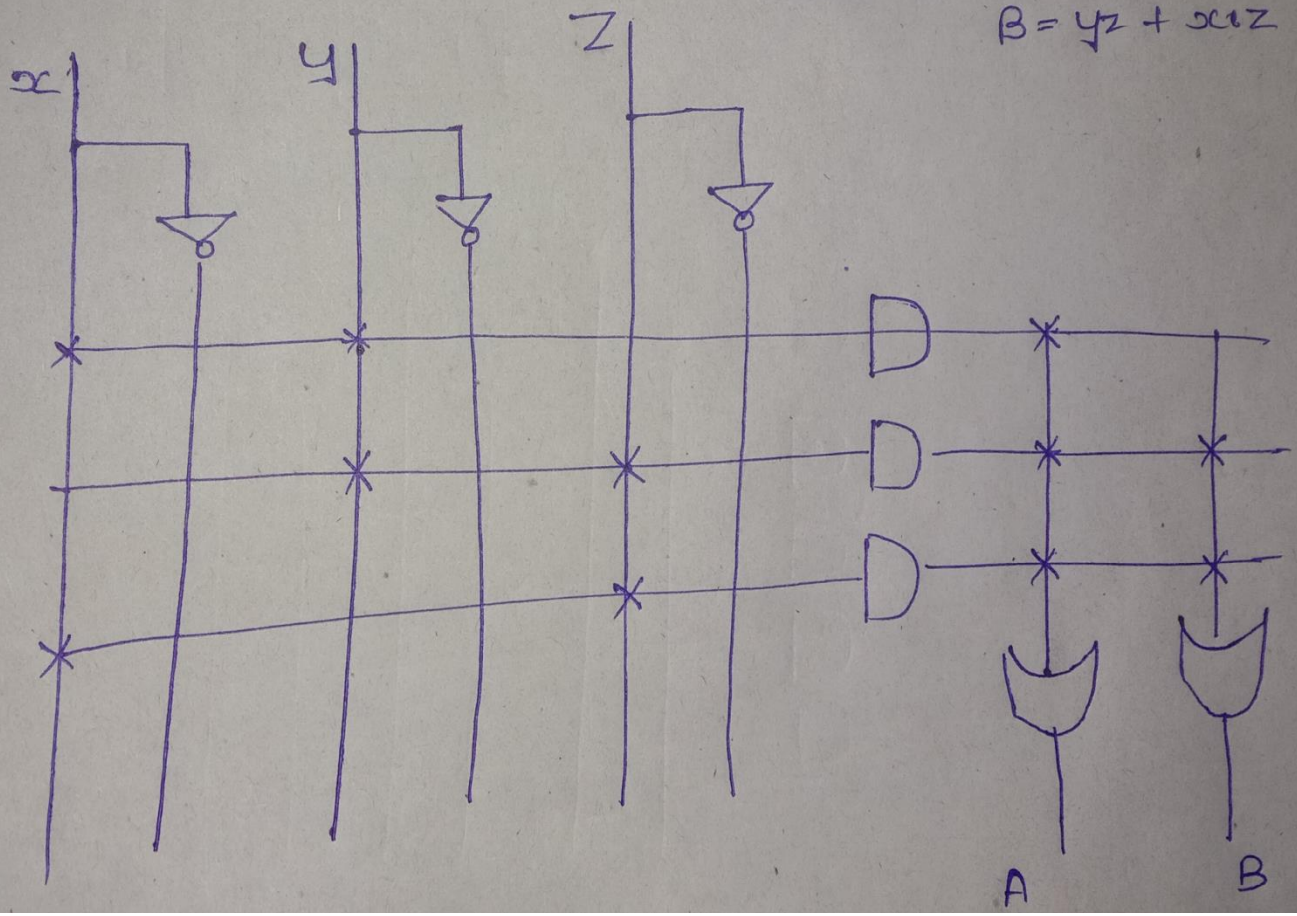
z \ xy	00	01	11	10
0			1	
1		1	1	1

$A = xy + yz + xz$

B(5,6,7)

z \ xy	00	01	11	10
0				
1		1	1	1

$B = yz + xz$



PLA

A(3,5,6,7)

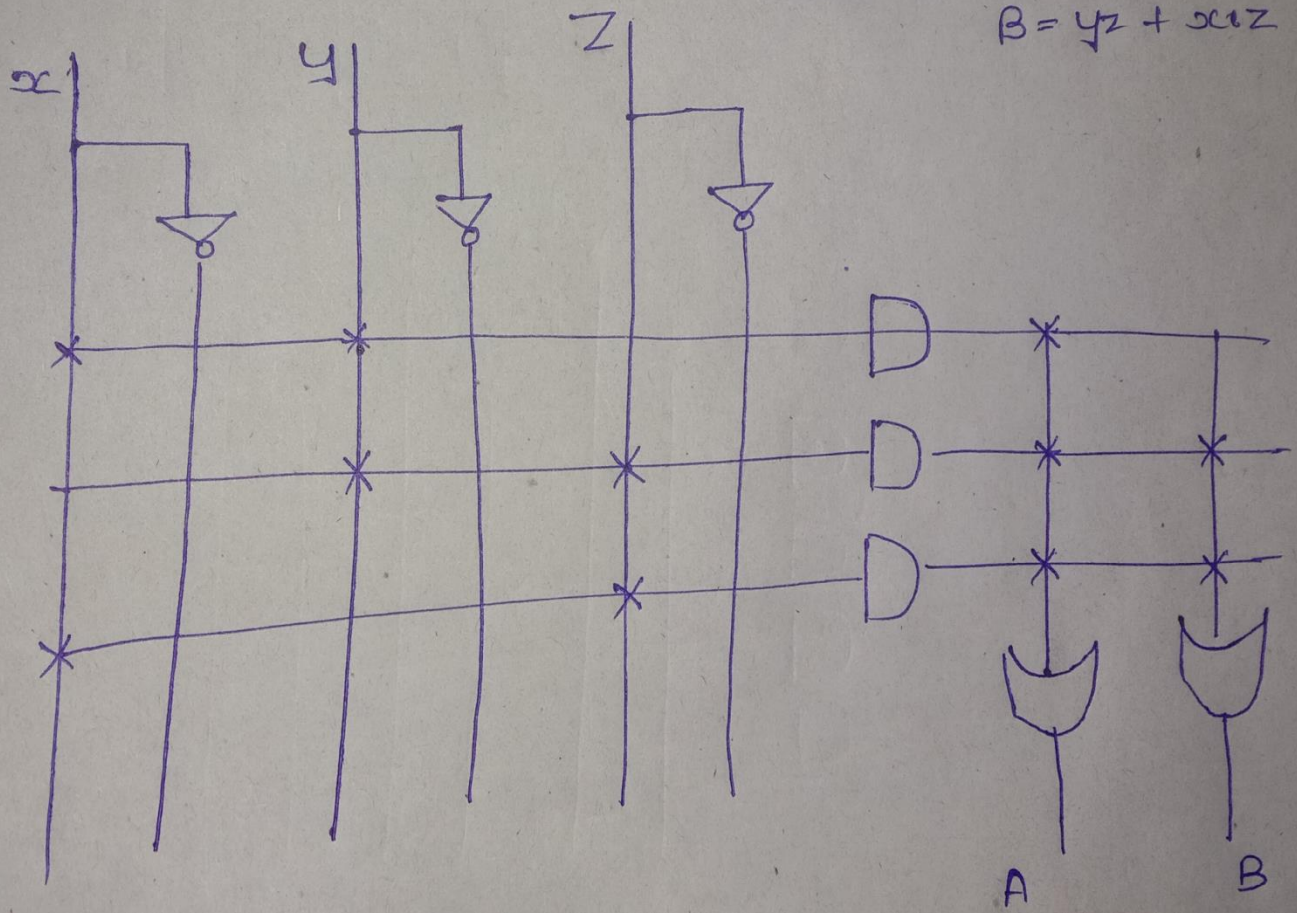
z \ xy	00	01	11	10
0			1	
1		1	1	1

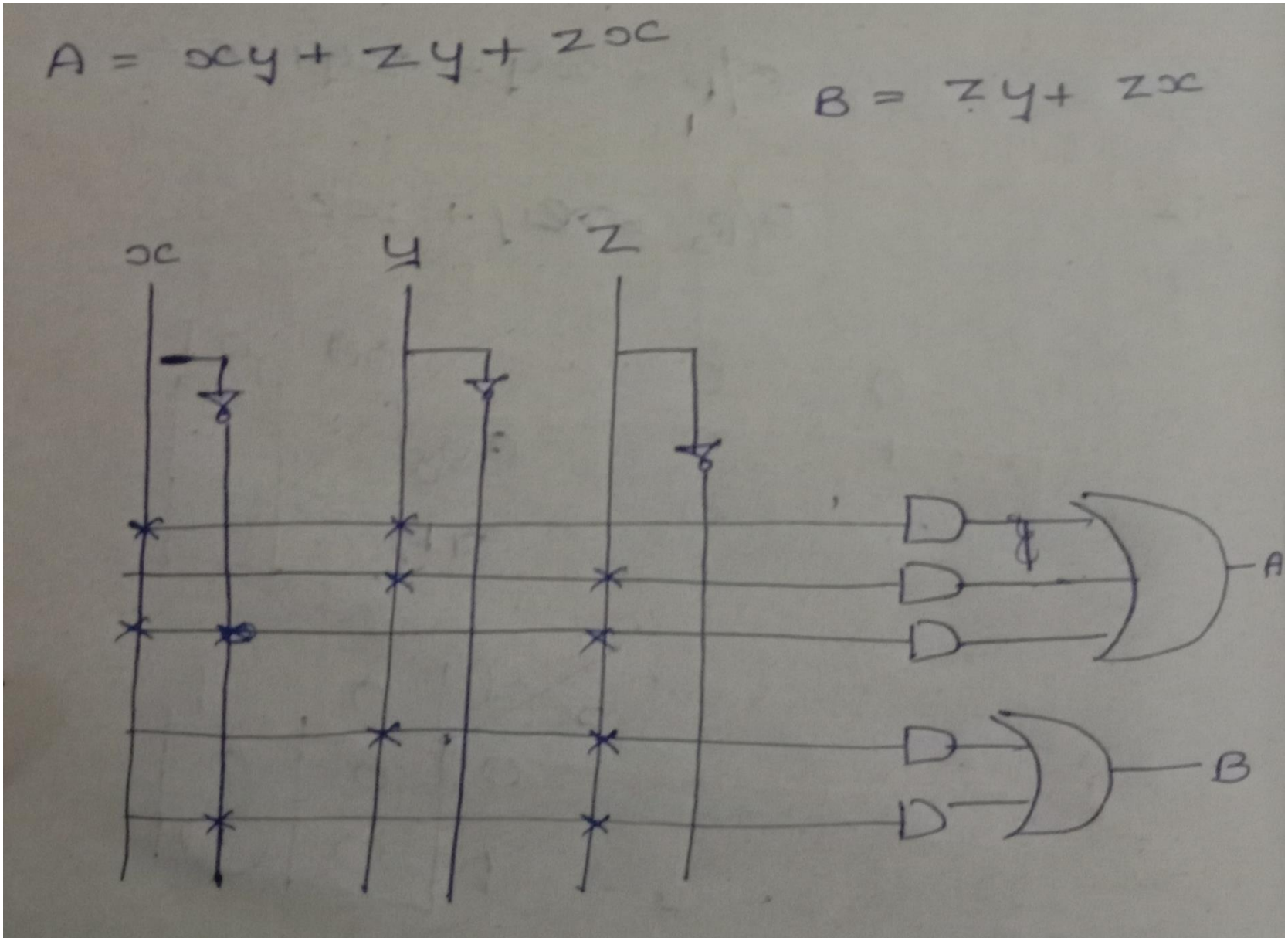
$A = xy + yz + xz$

B(5,6,7)

z \ xy	00	01	11	10
0				
1		1	1	1

$B = yz + xz$




A = xy + zy + zx
B = zy + zx
x
y
z
A
B

References

- https://www.elprocus.com/different-types-of-digital-logic-circuits/
- https://technobyte.org/sequential-combinational-logic-circuits-types/
- Kumar, A. Anand. *Switching Theory and Logic Design*. PHI Learning Pvt. Ltd., 2014.
- http://electronics-course.com/combinational-logic-design
- https://electronicscoach.com/half-adder.html
- https://www.gatevidyalay.com/tag/full-adder-using-nand-gates/
- A.K. Singh, *Foundation Of Switching Theory And Logic Design*, New Age International (P) Limited, 2007.
- https://www.geeksforgeeks.org/digital-electronics-logic-design-tutorials/
- M Morris Mano, Michael D. Ciletti. *Digital design: with an introduction to the verilog HDL*, Pearson Publishing, 2013.