

CHAPTER 1

COMBINATIONAL LOGIC

1.1 Introduction to Combinational Circuits

Electronic circuits constructed using digital logic gates and devices and designed to operate on digital inputs and outputs are called **digital logic circuits**. The digital logic circuits can be broadly classified into combinational circuits and sequential circuits.

The combinational circuits are digital logic circuits without feedbacks from output to input. Therefore, the outputs of combinational circuit will depend only on present inputs.

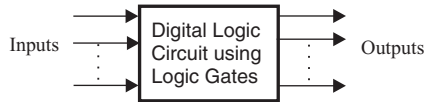


Fig. 1.1: Block diagram representation of a combinational circuit.

The sequential circuits are digital logic circuits with feedback from output to input, the sequential circuits are discussed in Chapter-2.

The combinational logic does not depend on feedback signals or previous output. The logical operations are performed using present inputs. The circuits that perform combinational logic operations are called **combinational logic circuits** and they are constructed using logic gates. The working of logic gates are governed by Boolean algebra and hence the design of combinational circuits requires a knowledge about Boolean algebra.

In combinational circuits the output at any time depends on input at that time. In combinational circuits there is no storage element and there is no feedback from output to input. Therefore, combinational circuits are designed for applications which do not require a record of previous outputs and for applications in which the present outputs do not depend on previous outputs. Some examples of combinational circuits are shown in Fig. 1.2.

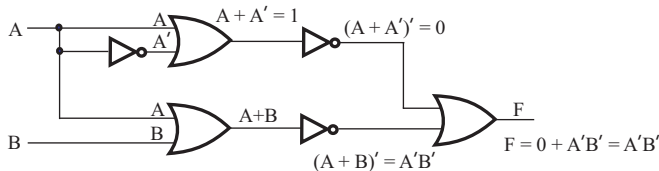


Fig. a.

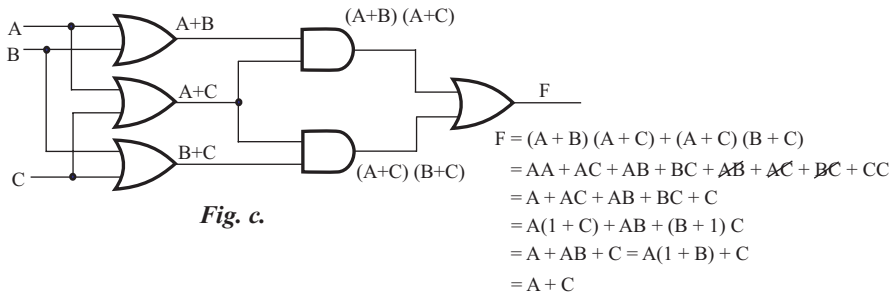
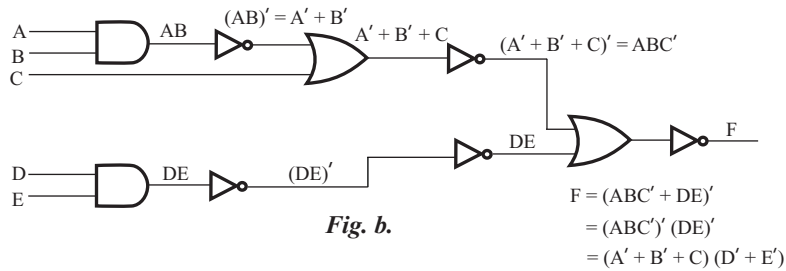


Fig. 1.2: Examples of combinational circuit.

1.1.1 Problem Formulation and Design of Combinational Circuits

The combinational circuits are designed to provide hardware based solution for logical problems. The design starts with problem specifications. The procedure to design a combinational circuit is given below:

1. Determine the required inputs and outputs from the problem specifications.
2. Assign a symbol to each input and output.
3. Derive the truth table.
4. Draw K-map for every output and form the prime implicants.
5. Determine the simplified Boolean function for every output from the prime implicants.
6. Implement the Boolean functions of all the outputs as a digital circuit using logic gates.

1.2 Boolean Algebra

George Boole developed Boolean algebra in **1854**.

Boolean algebra is an algebraic structure that includes a set of elements consisting of

- Binary operators, " + " and " . "
- Binary variables, x, y, z
- Binary elements, 0 and 1
- Boolean postulates and theorems

Let x and y be two boolean variables that can take all possible combination of binary value. The rules for binary operators " + " and " . " are listed in Table 1.1 using the two binary variables x and y.

Table 1.1: Rules for Binary Operators "+" and "."

x	y	$x \cdot y$	$x + y$	x	x'
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1		
1	1	1	1		

Operator Precedence while Evaluating Boolean Expression

A statement written with Boolean variables, constants and operators is called **Boolean expression**.

Examples of Boolean expression: $x + y \cdot z'$

$$(x \cdot y) + z' + (x + y \cdot z)'$$

In order to evaluate Boolean expression the operator precedence is as follows:

Step-1: Evaluate expression with in paranthesis, i.e, within ().

Step-2: Evaluate complement

Step-3: Evaluate "."

Step-4: Evaluate "+"

Note: To simplify Boolean expression dot operation is represented without operator.

$$x \cdot y \Rightarrow xy$$

$$y \cdot z \Rightarrow yz$$

$$x \cdot y \cdot z \Rightarrow xyz$$

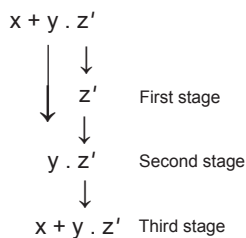
Example 1.1

Demonstrate the operator precedence in the evaluation of following Boolean expression.

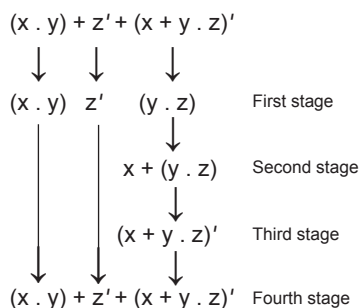
a) $x + y \cdot z'$ b) $(x \cdot y) + z' + (x + y \cdot z)'$

Solution

a) $x + y \cdot z'$



b) $(x \cdot y) + z' + (x + y \cdot z)'$



1.2.1 Duality

The **duality** principle of Boolean algebra states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operator and identity element are interchanged. Here, the operators are " + " and " . " and the identity elements are " 0 " and " 1 ".

Table 1.2: Basic Example of Duality

x	y	x + y	Change "+" to "."			x	y	x . y
0	0	0	Change 0 to 1 and 1 to 0	1	1	1	0	0
0	1	1		1	0	0	0	1
1	0	1		0	1	0	1	0
1	1	1		0	0	0	1	1

1.2.2 Postulates of Boolean Algebra

Postulates of Boolean algebra are developed by **E. V. Huntington** in 1904.

Boolean structure have to satisfy the following postulates.

1. a) The structure is closed with the respect to operator " + "
- b) The structure is closed with respect to operator " . "
2. a) The element 0 is an identity element with respect to operator " + "

$$\therefore x + 0 = x$$

$$0 + x = x$$
 where x is a Boolean variable that can take a value either 0 or 1.
- b) The element 1 is an identity element with respect to operator " . "

$$\therefore x . 1 = x$$

$$1 . x = x$$
 where x is Boolean variable that can take a value either 0 or 1.
3. a) The structure is commutative with respect to operator " + "

$$\therefore x + y = y + x$$
 where x and y are two Boolean variables.
- b) The structure is commutative with respect to operator " . "

$$\therefore x . y = y . x$$
4. a) The operator " . " is distributive over operator " + "

$$\therefore x . (y + z) = (x . y) + (x . z)$$
- b) The operator " + " is distributive over operator " . "

$$\therefore x + (y . z) = (x + y) . (x + z)$$
5. a) For every variable x there exists an element x' called complement of x

$$\therefore x + x' = 1 \quad \text{and} \quad x . x' = 0$$
6. There exists at least two variables x and y such that, $x \neq y$.

Distributive Law

Verification of Postulates

1. From the Table 1.1 it is obvious that results of "+" and "." operations are either 0 or 1 and so structure is closed with respect to operators "+" and "."

2. $x + 0 = x$, when $x = 0$, $x + 0 = 0 + 0 = 0 = x$

$x = 1$, $x + 0 = 1 + 0 = 1 = x$

$0 + x = x$, when $x = 0$, $0 + x = 0 + 0 = 0 = x$

$x = 1$, $0 + x = 0 + 1 = 1 = x$

3. $x \cdot 1 = x$, when $x = 0$, $x \cdot 1 = 0 \cdot 1 = 0 = x$

$x = 1$, $x \cdot 1 = 1 \cdot 1 = 1 = x$

$1 \cdot x = x$, when $x = 0$, $1 \cdot x = 1 \cdot 0 = 0 = x$

$x = 1$, $1 \cdot x = 1 \cdot 1 = 1 = x$

4. Let x, y, z be three Boolean variables that takes all possible combinations of binary value.

Construct a truth table for all possible combinations of x, y and z as shown in Table 1.3 to prove,

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

Table 1.3: Verification of (Distributive Law) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

x	y	z	y + z	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Similarly, Table 1.4 is constructed to prove,

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

Table 1.4: Verification of (Distributive Law) $x + (y \cdot z) = (x + y) \cdot (x + z)$

x	y	z	y · z	$x + (y \cdot z)$	$(x + y)$	$(x + z)$	$(x + y) \cdot (x + z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

5. $x + x' = 1$

when $x = 0$, $x' = 1$, $\therefore x + x' = 0 + 1 = 1$

when $x = 1$, $x' = 0$, $\therefore x + x' = 1 + 0 = 1$

$x \cdot x' = 0$

when $x = 0$, $x' = 1$, $\therefore x \cdot x' = 0 \cdot 1 = 0$

when $x = 1$, $x' = 0$, $\therefore x \cdot x' = 1 \cdot 0 = 0$

6. Boolean algebra has two elements 1 and 0, where $1 \neq 0$.

\therefore If x and y are two variables, then there is a possibility that

$x = 0$, $y = 1$, $\therefore x \neq y$

and $x = 1$, $y = 0$, $\therefore x \neq y$

1.2.3 Boolean Theorems

Theorem 1: $x + x = x$

$x \cdot x = x$

Theorem 2: $x + 1 = 1$

$x \cdot 0 = 0$

Theorem 3: $(x')' = x$
(Involution)

Theorem 4: $x + (y + z) = (x + y) + z$

(Associative) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Theorem 5: $(x + y)' = x' \cdot y'$

(DeMorgan's Law
or DeMorgan's
Theorem) $(x \cdot y)' = x' + y'$

Theorem 6: $x + (x \cdot y) = x$

(Absorption) $x \cdot (x + y) = x$
(y is absorbed)

Theorem 7: $x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$

(Consensus
Theorem) $(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$

Proof of Theorems**Theorem 1:**

$$x + x = x$$

$$\text{when, } x = 0, x + x = 0 + 0 = 0 = x$$

$$\text{when, } x = 1, x + x = 1 + 1 = 1 = x$$

$$x \cdot x = x$$

$$\text{when, } x = 0, x \cdot x = 0 \cdot 0 = 0 = x$$

$$\text{when, } x = 1, x \cdot x = 1 \cdot 1 = 1 = x$$

Theorem 2:

$$x + 1 = 1$$

$$\text{when, } x = 0, x + 1 = 0 + 1 = 1$$

$$\text{when, } x = 1, x + 1 = 1 + 1 = 1$$

$$x \cdot 0 = 0$$

$$\text{when, } x = 0, x \cdot 0 = 0 \cdot 0 = 0$$

$$\text{when, } x = 1, x \cdot 0 = 1 \cdot 0 = 0$$

Theorem 3: Involution

$$(x')' = x$$

$$\text{when, } x = 0, (x')' = (0')' = 1' = 0 = x$$

$$\text{when, } x = 1, (x')' = (1')' = 0' = 1 = x$$

Theorem 4: Associative

$$x + (y + z) = (x + y) + z$$

Construct a truth table for all possible combinations of x , y and z as shown in Table 1.5 to prove,

$$x + (y + z) = (x + y) + z$$

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

Construct a truth table for all possible combinations of x , y and z as shown in Table 1.6 to prove,

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

Table 1.5: Verification of $x + (y + z) = (x + y) + z$

x	y	z	y + z	x + (y + z)	x + y	(x + y) + z
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Table 1.6: Verification of $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

x	y	z	y · z	x · (y · z)	x · y	(x · y) · z
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

Theorem 5: DeMorgan's Theorem

$$(x + y)' = x' \cdot y'$$

Construct a truth table for all possible combinations of x and y as shown in Table 1.7 to prove,

$$(x + y)' = x' \cdot y'$$

Table 1.7: Verification of $(x + y)' = x' \cdot y'$

x	y	$x + y$	$(x + y)'$	x'	y'	$x' \cdot y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

$$(x \cdot y)' = x' + y'$$

Construct a truth table for all possible combinations of x and y as shown in Table 1.8 to prove,

$$(x \cdot y)' = x' + y'$$

Table 1.8: Verification of $(x \cdot y)' = x' + y'$

x	y	$x \cdot y$	$(x \cdot y)'$	x'	y'	$x' + y'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Theorem 6: Absorption

$$x + (x \cdot y) = x$$

Construct a truth table for all possible combinations of x and y as shown in Table 1.9 to prove,

$$x + (x \cdot y) = x$$

Table 1.9: Verification of $x + (x \cdot y) = x$

x	y	$x \cdot y$	$x + (x \cdot y)$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

$$x \cdot (x + y) = x$$

Construct a truth table for all possible combinations of x and y as shown in Table 1.10 to prove,

$$x \cdot (x + y) = x$$

Table 1.10: Verification of $x \cdot (x + y) = x$

x	y	$x + y$	$x \cdot (x + y)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Theorem 7: Consensus Theorem

$$x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$$

Construct a truth table for all possible combinations of x, y and z as shown in Table 1.11 to prove,

$$x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$$

Table 1.11: Verification of $x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$

x	y	z	x'	$x \cdot y$	$x' \cdot z$	$y \cdot z$	$x \cdot y + x' \cdot z + y \cdot z$	$x \cdot y + x' \cdot z$
0	0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1	1
0	1	0	1	0	0	0	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	1	1
1	1	1	0	1	0	1	1	1

$$(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$$

Construct a truth table for all possible combinations of x, y and z as shown in Table 1.12 to prove,

$$(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$$

Table 1.12: Verification of $(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$

x	y	z	x'	x + y	x' + z	y + z	$(x + y) \cdot (x' + z) \cdot (y + z)$	$(x + y) \cdot (x' + z)$
0	0	0	1	0	1	0	0	0
0	0	1	1	0	1	1	0	0
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	0	0	1	0	1	0	0
1	1	1	0	1	1	1	1	1

1.3 Binary Logic

Binary logic consists of binary variables and a set of logical operations. The binary variables are denoted by letters of alphabet, A, B, C,, x, y, z,

Each binary variable can take only two possible values 0 or 1. In positive logic, 0 is **low** and 1 is **high**. The basic logical operations are AND, OR and NOT.

AND operation is same as " \cdot " operation. In positive logic, the logical AND of two or more variables will be 1 if and only if the value of all the variables is 1.

OR operation is same as " $+$ " operation. In positive logic, the logical OR of two or more variables will be 1 if the value of any one of the variables is 1.

NOT operation is same as complement operation.

The results or outputs of logical operations of two or more variables for all possible combinations of the variables can be listed in a table called **truth table**. The truth tables of basic logical operations are shown in Table 1.13.

Table 1.13: Truth Tables of AND, OR and NOT Operation of Two Variables**AND operation**

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR operation

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT operation

x	x'
0	1
1	0

Positive and Negative Logic

Binary constants are 1 and 0. Physically in a digital circuit they represent two voltage levels.

The two voltage levels are called **high** and **low**.

The voltage level of **high** and **low** depends on technology used to fabricate the gates.

In TTL logic (Transistor Transistor Logic) **high** is +5 V and **low** is 0 V.

In CMOS (Complementary Symmetry MOSFET) **high** is +3.3 V and **low** is 0 V.

If, 1 represent **high** voltage and 0 represent **low** voltage then logic system is called **positive logic**.

If, 1 represent **low** voltage and 0 represent **high** voltage then the logic system is called **negative logic**.

Table 1.14: Positive and Negative Logic Levels

Logic Type (or Logic System)	Technology	
	TTL	CMOS
Positive Logic	1 = High = +5 V 0 = Low = 0 V	1 = High = + 3.3 V 0 = Low = 0 V
Negative Logic	0 = High = + 5 V 1 = Low = 0 V	0 = High = + 3.3 V 1 = Low = 0 V

1.4 Logic Gates

Logic gates are electronic devices or circuits that perform logical operations on one or more input logical variables and produce a binary output. The basic logic gates are AND, OR and NOT gates.

The logic gates have standard symbols as shown in Figs. 1.3 to 1.7.

AND Gate

AND gate is an electronic device that performs logical AND operation of two or more variables.

**Fig. 1.3:** Two input AND gate.**Fig. 1.4:** Three input AND gate.

OR Gate

OR gate is an electronic device that performs logical OR operation of two or more variables.

**Fig. 1.5:** Two input OR gate.**Fig. 1.6:** Three input OR gate.

Note: Theoretically, AND and OR gates can have any number of inputs.

NOT Gate (or inverter)

NOT gate is an electronic device that performs complement operation of a Boolean variable.

**Fig. 1.7:** NOT gate.

Other Logical Operations

With n variables it is possible to form 2^{2n} Boolean functions.

When, $n = 2$, $2^{2n} = 2^{2 \times 2} = 2^4 = 16$

Therefore, with two variables we can form 16 Boolean functions.

All the 16 possible function of two variables are listed in Table 1.15.

The output of 16 functions are 16 binary combinations of 4-bit binary.

Let, $F_0, F_1, F_2, \dots, F_{15}$ be 16 possible Boolean functions for two variables x, y .

Table 1.15: Two-Variable Boolean Functions

Functions	x	y	F	Functions	x	y	F
Null operation $F = F_0 = 0$	0	0	0	NOR operation $F = F_8 = (x + y)'$	0	0	1
	0	1	0		0	1	0
	1	0	0		1	0	0
	1	1	0		1	1	0
AND operation $F = F_1 = x \cdot y$	0	0	0	Exclusive NOR(XNOR) (or equivalence) $F = F_9 = x \odot y = (x \oplus y)'$ $= x \cdot y + x' \cdot y'$	0	0	1
	0	1	0		0	1	0
	1	0	0		1	0	0
	1	1	1		1	1	1
Inhibition $F = F_2 = x \cdot y'$ (x true, but y false)	0	0	0	Complement of y $F = F_{10} = y'$	0	0	1
	0	1	0		0	1	0
	1	0	1		1	0	1
	1	1	0		1	1	0
Transfer $F = F_3 = x$ (Transfer x)	0	0	0	Implication $F = F_{11} = x + y'$ (If y = 1, F = x If y = 0, F = y')	0	0	1
	0	1	0		0	1	0
	1	0	1		1	0	1
	1	1	1		1	1	1
Inhibition $F = F_4 = x' \cdot y$ (y true, but x false)	0	0	0	Complement of x $F = F_{12} = x'$	0	0	1
	0	1	1		0	1	1
	1	0	0		1	0	0
	1	1	0		1	1	0
Transfer $F = F_5 = y$ (Transfer y)	0	0	0	Implication $F = F_{13} = x' + y$ (If x = 1, F = y If x = 0, F = x')	0	0	1
	0	1	1		0	1	1
	1	0	0		1	0	0
	1	1	1		1	1	1
XOR operation $F = F_6 = x \oplus y$ $= x \cdot y' + x' \cdot y$	0	0	0	NAND operation $F = F_{14} = (x \cdot y)'$	0	0	1
	0	1	1		0	1	1
	1	0	1		1	0	1
	1	1	0		1	1	0
OR operation $F = F_7 = x + y$	0	0	0	Identity $F = F_{15} = 1$	0	0	1
	0	1	1		0	1	1
	1	0	1		1	0	1
	1	1	1		1	1	1

Summary of Logic Gates

The AND, OR and NOT are basic gates. Using these basic gates some more useful logical operations can be defined. They are NAND, NOR, Exclusive-OR (XOR) and Exclusive-NOR (XNOR).

Besides logic gates, buffers or drivers are also used in digital circuits to augment or increase current levels of signals where ever required.

The summary of popular logic gates used in digital electronics are listed in Table 1.16.

Table 1.16: Summary of Popular Logic Gates

(AU, Nov/Dec'22, 13 Marks)








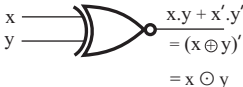
Gate	Symbol	Truth Table																				
AND Gate		<table><tr><th>x</th><th>y</th><th>$x \cdot y$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	$x \cdot y$	0	0	0	0	1	0	1	0	0	1	1	1					
x	y	$x \cdot y$																				
0	0	0																				
0	1	0																				
1	0	0																				
1	1	1																				
OR Gate		<table><tr><th>x</th><th>y</th><th>$x + y$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	$x + y$	0	0	0	0	1	1	1	0	1	1	1	1					
x	y	$x + y$																				
0	0	0																				
0	1	1																				
1	0	1																				
1	1	1																				
Inverter Gate (or NOT gate)		<table><tr><th>x</th><th>x'</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	x'	0	1	1	0														
x	x'																					
0	1																					
1	0																					
Buffer		<table><tr><th>x</th></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	x	0	1																	
x																						
0																						
1																						
NAND Gate		<table><tr><th>x</th><th>y</th><th>$x \cdot y$</th><th>$(x \cdot y)'$</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	x	y	$x \cdot y$	$(x \cdot y)'$	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0
x	y	$x \cdot y$	$(x \cdot y)'$																			
0	0	0	1																			
0	1	0	1																			
1	0	0	1																			
1	1	1	0																			

Table 1.16: Continued...

Gate	Symbol	Truth Table																									
NOR Gate		<table><tr><th>x</th><th>y</th><th>$x + y$</th><th>$(x + y)'$</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	x	y	$x + y$	$(x + y)'$	0	0	0	1	0	1	1	0	1	0	1	0	1	1	1	0					
x	y	$x + y$	$(x + y)'$																								
0	0	0	1																								
0	1	1	0																								
1	0	1	0																								
1	1	1	0																								
Exclusive-OR Gate (XOR)		<table><tr><th>x</th><th>y</th><th>$x \cdot y'$</th><th>$x' \cdot y$</th><th>$x \oplus y$</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	x	y	$x \cdot y'$	$x' \cdot y$	$x \oplus y$	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	1	1	0	0	0
x	y	$x \cdot y'$	$x' \cdot y$	$x \oplus y$																							
0	0	0	0	0																							
0	1	0	1	1																							
1	0	1	0	1																							
1	1	0	0	0																							
Exclusive-NOR Gate (XNOR) (or equivalence)		<table><tr><th>x</th><th>y</th><th>$x \oplus y$</th><th>$(x \oplus y)' = x \odot y$</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	x	y	$x \oplus y$	$(x \oplus y)' = x \odot y$	0	0	0	1	0	1	1	0	1	0	1	0	1	1	0	1					
x	y	$x \oplus y$	$(x \oplus y)' = x \odot y$																								
0	0	0	1																								
0	1	1	0																								
1	0	1	0																								
1	1	0	1																								

Note: Bubble at output of gate indicates NOT operation.

Example 1.2

Prove the following identity.

- $AB + A(B + C) + B(B + C) = B + AC$
- $AB + A'B + A'B' = A' + B$
- $AB'C + A'BC + ABC = AC + BC$

Solution

$$\begin{aligned}
 \text{a) } AB + A(B + C) + B(B + C) &= AB + AB + AC + BB + BC \\
 &= AB + AC + B + BC \\
 &= B(A + 1 + C) + AC = B \cdot 1 + AC \\
 &= B + AC
 \end{aligned}$$

$$x + x = x$$

Repeated terms are considered once. $xx = x$

$$1 + x = 1$$

$$\begin{aligned}
 \text{b) } AB + A'B + A'B' &= AB + A'(B + B') \\
 &= AB + A' \cdot 1 = AB + A' \\
 &= (A' + A)(A' + B) = 1 \cdot (A' + B) \\
 &= A' + B
 \end{aligned}$$

$$x + x' = 1$$

$$\begin{aligned}
 \text{c) } AB'C + A'BC + ABC &= ABC + AB'C + A'BC + ABC \\
 &= AC(B + B') + BC(A + A') \\
 &= AC + BC
 \end{aligned}$$

$$x + x' = 1$$

Example 1.3

Prove the following identity using Boolean algebra:

$$(A + B)(A + (AB)')C + A'(B + C') + A'B + ABC = C(A + B) + A'(B + C')$$

Solution

$$\text{L.H.S} = (A + B)(A + (AB)')C + A'(B + C') + A'B + ABC$$

$$\text{R.H.S} = C(A + B) + A'(B + C')$$

$$\text{L.H.S} = (A + B)(A + (AB)')C + A'(B + C') + A'B + ABC$$

$$= (A + B)(A + A' + B')C + A'B + A'C' + B(A' + AC)$$

$$= (A + B)(1 + B')C + A'B + A'C' + B((A' + A)(A' + C))$$

$$= (A + B)1.C + A'B + A'C' + B(1.(A' + C))$$

$$= AC + BC + A'B + A'C' + B(A' + C)$$

$$= AC + BC + A'B + A'C' + \cancel{A'B} + \cancel{BC}$$

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

$$= C(A + B) + A'(B + C')$$

$$= \text{RHS}$$

$$x + x' = 1$$

Using DeMorgan's theorem

Repeated terms are considered once.

Example 1.4

Prove the following:

$$\text{a) } A \oplus B = A' \oplus B' \quad \text{b) } (A \oplus B)' = A \oplus B' = A' \oplus B$$

Solution

$$\text{a) } A \oplus B = A' \oplus B'$$

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

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

$$x \oplus y = xy' + x'y$$

$$\text{b) } (A \oplus B)' = A \oplus B' = A' \oplus B$$

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

$$= AB + A'B' = (A \oplus B)'$$

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

$$= A'B' + AB = (A \oplus B)'$$

1.4.1 Universal Gates

NAND and NOR gates are called **universal gates**, because any Boolean function can be realized only using NAND gates or only using NOR gates.

The NAND gate is a combination of AND followed by NOT gate. The NOR gate is a combination of OR followed by NOT gate.

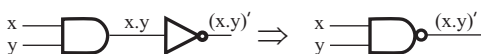


Fig. 1.8: NAND gate.

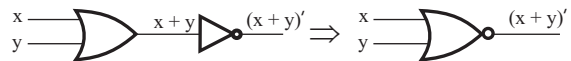


Fig. 1.9: NOR gate.

Table 1.17: Truth Table of NAND Gate

x	y	$(x \cdot y)'$
0	0	1
0	1	1
1	0	1
1	1	0

Table 1.18: Truth Table of NOR Gate

x	y	$(x + y)'$
0	0	1
0	1	0
1	0	0
1	1	0

The realization of basic gates using only NAND gates are shown in Table 1.19.

The realization of basic gates using only NOR gates are shown in Table 1.20.

Table 1.19: Realization of Basic Logic Gates using NAND Gates


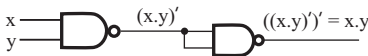
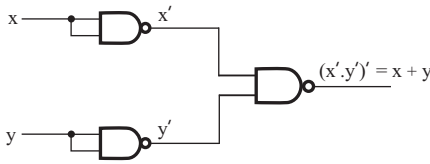
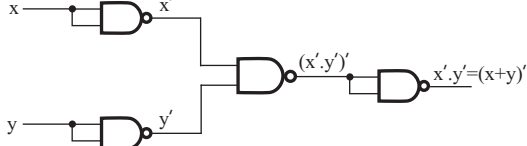
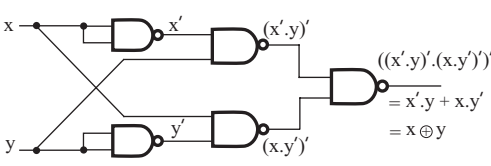

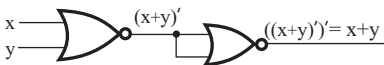
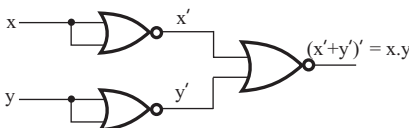
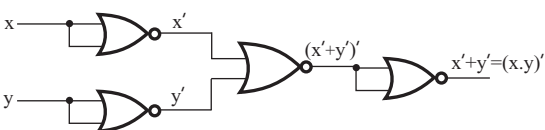
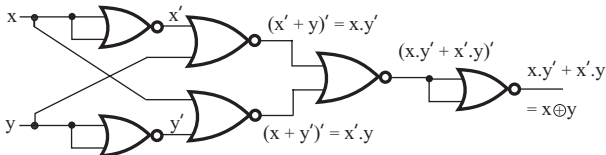
Gate	Logic Circuit using NAND	Truth Table																									
Inverter/NOT gate using NAND		<table><tr><th>x</th><th>x'</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	x'	0	1	1	0																			
x	x'																										
0	1																										
1	0																										
AND using NAND		<table><tr><th>x</th><th>y</th><th>x . y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	x . y	0	0	0	0	1	0	1	0	0	1	1	1										
x	y	x . y																									
0	0	0																									
0	1	0																									
1	0	0																									
1	1	1																									
OR using NAND		<table><tr><th>x</th><th>y</th><th>x + y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	x + y	0	0	0	0	1	1	1	0	1	1	1	1										
x	y	x + y																									
0	0	0																									
0	1	1																									
1	0	1																									
1	1	1																									
NOR using NAND		<table><tr><th>x</th><th>y</th><th>x + y</th><th>(x + y)'</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	x	y	x + y	(x + y)'	0	0	0	1	0	1	1	0	1	0	1	0	1	1	1	0					
x	y	x + y	(x + y)'																								
0	0	0	1																								
0	1	1	0																								
1	0	1	0																								
1	1	1	0																								
XOR using NAND		<table><tr><th>x</th><th>y</th><th>x . y'</th><th>x' . y</th><th>x ⊕ y</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table> <p style="text-align: center;">$x \oplus y = x y' + x' y$</p>	x	y	x . y'	x' . y	x ⊕ y	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	1	1	0	0	0
x	y	x . y'	x' . y	x ⊕ y																							
0	0	0	0	0																							
0	1	0	1	1																							
1	0	1	0	1																							
1	1	0	0	0																							

Table 1.20: Realization of Basic Logic Gates using NOR Gates

Gate	Logic Circuit using NOR	Truth Table																									
Inverter/NOT gate using NOR	 $(x + x)' = x'$	<table><tr><th>x</th><th>x'</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	x'	0	1	1	0																			
x	x'																										
0	1																										
1	0																										
OR using NOR	 $((x+y)')' = x+y$	<table><tr><th>x</th><th>y</th><th>x + y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	x + y	0	0	0	0	1	1	1	0	1	1	1	1										
x	y	x + y																									
0	0	0																									
0	1	1																									
1	0	1																									
1	1	1																									
AND using NOR	 $(x'+y')' = x.y$	<table><tr><th>x</th><th>y</th><th>x . y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	x . y	0	0	0	0	1	0	1	0	0	1	1	1										
x	y	x . y																									
0	0	0																									
0	1	0																									
1	0	0																									
1	1	1																									
NAND using NOR	 $x' + y' = (x.y)'$	<table><tr><th>x</th><th>y</th><th>x.y</th><th>(x.y)'</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	x	y	x.y	(x.y)'	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0					
x	y	x.y	(x.y)'																								
0	0	0	1																								
0	1	0	1																								
1	0	0	1																								
1	1	1	0																								
XOR using NOR	 $x \oplus y = x.y' + x'.y$	<table><tr><th>x</th><th>y</th><th>x . y'</th><th>x' . y</th><th>x ⊕ y</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table> $x \oplus y = x.y' + x'.y$	x	y	x . y'	x' . y	x ⊕ y	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	1	1	0	0	0
x	y	x . y'	x' . y	x ⊕ y																							
0	0	0	0	0																							
0	1	0	1	1																							
1	0	1	0	1																							
1	1	0	0	0																							

1.4.2 Positive and Negative Logic Gates

In positive logic, 1 is logic **high** and 0 is logic **low**. In negative logic, 0 is logic **high** and 1 is logic **low**. The logic gates discussed in Section 1.4 are positive logic gates which means that the logic levels of inputs and outputs are positive logic levels.

Logic gates can also be designed to work with negative logic levels. The AND and OR operation in positive and negative logic are given below:

In this book, only positive logic gates are used for analysis and design of digital logic circuit.

AND operation: AND operation of two or more variables is **high** only if all the variables are **high**.

Table 1.21: Truth Table of Positive Logic AND Gate

Inputs		Output
a	b	a . b
0 (low)	0 (low)	0 (low)
0 (low)	1 (high)	0 (low)
1 (high)	0 (low)	0 (low)
1 (high)	1 (high)	1 (high)

Table 1.22: Truth Table of Negative Logic AND Gate

Inputs		Output
a	b	a . b
0 (high)	0 (high)	0 (high)
0 (high)	1 (low)	1 (low)
1 (low)	0 (high)	1 (low)
1 (low)	1 (low)	1 (low)

OR operation: OR operation of two or more variables is **high** if one of the variable is **high**.

Table 1.23: Truth Table of Positive Logic OR Gate















Inputs		Output
a	b	a + b
0 (low)	0 (low)	0 (low)
0 (low)	1 (high)	1 (high)
1 (high)	0 (low)	1 (high)
1 (high)	1 (high)	1 (high)

Table 1.24: Truth Table of Negative Logic OR Gate

Inputs		Output
a	b	a + b
0 (high)	0 (high)	0 (high)
0 (high)	1 (low)	0 (high)
1 (low)	0 (high)	0 (high)
1 (low)	1 (low)	1 (low)

In order to differentiate negative logic level gates from the positive logic level gates, a bubble is added at every input and output. The bubble basically represent an inverter which is used to invert the logic levels. When a bubble is added to the output end that already has a bubble then it represents double time inversion and so that output end will not have a bubble in negative logic representation. The conversion of positive logic level gates to negative logic level gates are listed in Table 1.25.

Table 1.25: Summary of Positive and Negative Logic Gates

Gate	Positive Logic	Negative Logic
AND Gate		
OR Gate		
Inverter Gate		
Buffer		
NAND Gate		
NOR Gate		
XOR gate		

Using the output equations of logic gates listed in Table 1.25, the truth table of positive and negative logic gates are obtained as shown in Tables 1.26 to 1.29.

Table 1.26: Truth Table of AND Gate

a) Positive Logic

Inputs		Output
x	y	$z = x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

b) Negative Logic

Inputs		Complement of Inputs		Output
x	y	x'	y'	$z = (x' \cdot y')'$
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

Table 1.27: Truth Table of OR Gate

a) Positive Logic

Inputs		Output
x	y	$z = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

b) Negative Logic

Inputs		Complement of Inputs		Output
x	y	x'	y'	$z = (x' + y')'$
0	0	1	1	0
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

In negative logic AND gate,

$$z = (x' \cdot y')' \Rightarrow z = x + y$$

Using DeMorgan's theorem

Therefore, we can say that the negative logic AND gate is same as positive logic OR gate.

In negative logic OR gate,

$$z = (x' + y')' \Rightarrow z = x \cdot y$$

Using DeMorgan's theorem

Therefore, we can say that the negative logic OR gate is same as positive logic AND gate.

Table 1.28: Truth Table of NAND Gate

a) Positive Logic

Inputs		Output
x	y	$z = (x \cdot y)'$
0	0	1
0	1	1
1	0	1
1	1	0

b) Negative Logic

Inputs		Complement of Inputs	Output
x	y	x' y'	$z = x' \cdot y'$
0	0	1 1	1
0	1	1 0	0
1	0	0 1	0
1	1	0 0	0

Table 1.29: Truth Table of NOR Gate

a) Positive Logic

Inputs		Output
x	y	$z = (x + y)'$
0	0	1
0	1	0
1	0	0
1	1	0

b) Negative Logic

Inputs		Complement of Inputs	Output
x	y	x' y'	$z = x' + y'$
0	0	1 1	1
0	1	1 0	1
1	0	0 1	1
1	1	0 0	0

In negative logic NAND gate,

$$z = x' \cdot y' \Rightarrow z = (x + y)'$$

Using DeMorgan's theorem

Therefore, we can say that the negative logic NAND gate is same as positive logic NOR gate.

In negative logic NOR gate,

$$z = x' + y' \Rightarrow z = (x \cdot y)'$$

Using DeMorgan's theorem

Therefore, we can say that the negative logic NOR gate is same as positive logic NAND gate.

1.5 Boolean Functions

A **Boolean function** is described by a Boolean expression which consists of binary variables, binary constants 0 and 1 and logical operators, AND, OR and NOT. Binary variables are denoted by either lower case (a, b, c,, x, y, z) or upper case (A, B, C,, X, Y, Z) alphabets.

Some examples of Boolean functions are given below:

$$F = (x \cdot y) + z'$$

$$F_1 = (x \cdot z) + y'$$

$$F_2 = (x \cdot y \cdot z) + (y \cdot z')$$

Alternatively,

$$F(x, y, z) = (x \cdot y) + z'$$

$$F_1(x, y, z) = (x \cdot z) + y'$$

$$F_2(x, y, z) = (x \cdot y \cdot z) + (y \cdot z')$$

A Boolean function can be evaluated for all possible combinations of binary values of the variables of the function. The variables of a function are also known as **input variables** or **inputs**.

Let, n = Number of variables in a Boolean function

2^n = Number of combinations of binary values of n variables.

Truth table for a Boolean function can be constructed with 2^n combinations of n input variables.

Note: To simplify Boolean expression dot or AND operation is represented without operator.

$$x \cdot y \Rightarrow xy \qquad y \cdot z \Rightarrow yz \qquad x \cdot y \cdot z \Rightarrow xyz$$

Example 1.5

Construct the truth table for the following functions.

a) $F_1 = xy + xy' + y'z$ b) $F_2 = bc + a'c'$

Solution

a) $F_1 = xy + xy' + y'z$

Table 1: Truth Table of Function, F_1

Input Variables and Complement				Product Terms			Function Output
x	y	z	y'	xy	xy'	y'z	F ₁
0	0	0	1	0	0	0	0
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	0	1
1	1	1	0	1	0	0	1

b) $F_2 = bc + a'c'$

Table 1: Truth Table of Function, F_2

Input Variables and Complements					Product Terms		Function Output
a	b	c	a'	c'	bc	a'c'	F ₂
0	0	0	1	1	0	1	1
0	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0
1	1	1	0	0	1	0	1

Example 1.6

Realize 2-input XNOR gate using NOR gates.

Solution

The XNOR operation is given by the following Boolean equation.

$$x \odot y = xy + x'y'$$

The above equation can be implemented using NOR gate as shown in Fig. 1.

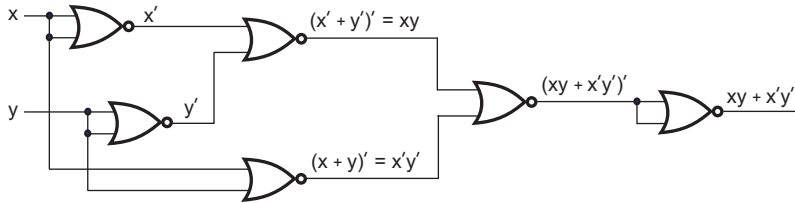


Fig. 1: XNOR operation using only NOR gates.

Example 1.7

Realize the 3-input gate using 2-input gates for the following gates:

- a) AND b) OR c) NAND d) NOR

Solution

- a) AND

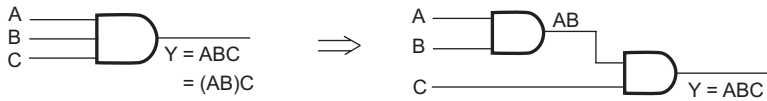


Fig. 1: 3-input AND gate using 2-input AND gate.

- b) OR

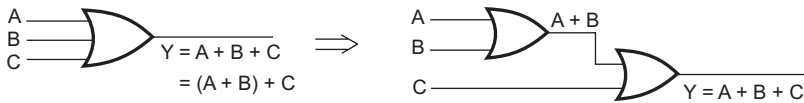


Fig. 2: 3-input OR gate using 2-input OR gate.

- c) NAND

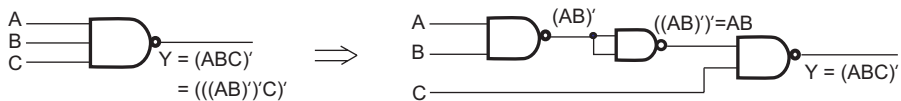


Fig. 3: 3-input NAND gate using 2-input NAND gate.

- d) NOR

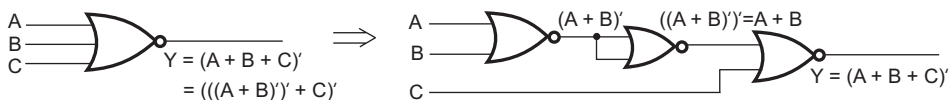


Fig. 4: 3-input NOR gate using 2-input NOR gate.

1.5.1 Implementation of Boolean Functions by Logic Gates

A Boolean function basically involves product of variables, sum of variables and complement of variables. Therefore, the straight forward implementation of Boolean function can be made by using basic gates AND, OR and NOT. The complement of a variable is obtained using NOT gate. The product term is realized using AND gate. The sum term is realized using OR gate.

Example 1.8

Implement the following functions using basic logic gates.

a) $F_1 = xy + xy' + y'z$ b) $F_2 = bc + a'c'$

Solution

a) $F_1 = xy + xy' + y'z$

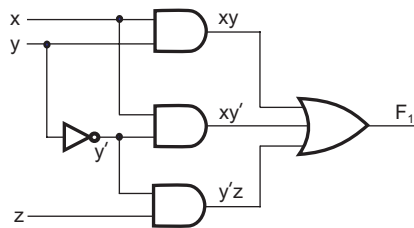


Fig. 1: Logic circuit for function, F_1 .

b) $F_2 = bc + a'c'$

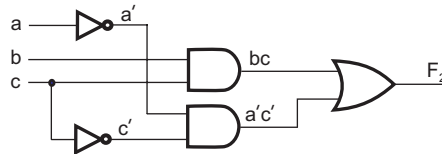


Fig. 2: Logic circuit for function, F_2 .

1.5.2 Minterms

Minterms are 2^n possible combinations of AND terms (or product terms) with n variables such that the logical AND of all the variables is 1.

The minterms can be formed with two, three, four, variables. The 2^n minterms are denoted as $m_0, m_1, m_2, m_3, \dots, m_q$ where $q = 2^n - 1$.

The 2^n combination of AND terms, for $n = 2$ and 3 are shown below. The AND terms are formed from 2^n combinations of n -bit binary. In AND terms a literal is primed if its value is 0 and unprimed if its value is 1. So that the AND of all literals is always 1.

Let, $n = 2$

Let x and y be the two variables to denote the two bits of binary.

When $n = 2$, $2^n = 2^2 = 4$

Therefore, four combinations of AND terms with two variables are possible as shown below:

$$\begin{array}{lll}
 00 & x'.y' & (0' \cdot 0' = 1) \quad x'y' = m_0 \\
 01 \Rightarrow & x'.y & (0' \cdot 1 = 1) \Rightarrow x'y = m_1 \\
 10 & x.y' & (1 \cdot 0' = 1) \quad xy' = m_2 \\
 11 & x.y & (1 \cdot 1 = 1) \quad xy = m_3
 \end{array}$$

Let, $n = 3$

Let x , y and z be the three variables to denote the three bits of binary.

When $n = 3$, $2^n = 2^3 = 8$

Therefore, eight combinations of AND terms with three variables are possible as shown below:

$$\begin{array}{lll}
 000 & x'.y'.z' & (0' \cdot 0' \cdot 0' = 1) \quad x'y'z' = m_0 \\
 001 \Rightarrow & x'.y'.z & (0' \cdot 0' \cdot 1 = 1) \Rightarrow x'y'z = m_1 \\
 010 & x'.y.z' & (0' \cdot 1 \cdot 0' = 1) \quad x'y z' = m_2 \\
 011 & x'.y.z & (0' \cdot 1 \cdot 1 = 1) \quad x'y z = m_3 \\
 100 & x.y'.z' & (1 \cdot 0' \cdot 0' = 1) \quad x y'z' = m_4 \\
 101 \Rightarrow & x.y'.z & (1 \cdot 0' \cdot 1 = 1) \Rightarrow x y'z = m_5 \\
 110 & x.y.z' & (1 \cdot 1 \cdot 0' = 1) \quad x y z' = m_6 \\
 111 & x.y.z & (1 \cdot 1 \cdot 1 = 1) \quad x y z = m_7
 \end{array}$$

Table 1.30: Two-Variable Minterms

x	y	Minterm	Notation
0	0	$x'.y'$	m_0
0	1	$x'.y$	m_1
1	0	$x.y'$	m_2
1	1	$x.y$	m_3

Table 1.31: Three-Variable Minterms

x	y	z	Minterm	Notation
0	0	0	$x'.y'.z'$	m_0
0	0	1	$x'.y'.z$	m_1
0	1	0	$x'.y.z'$	m_2
0	1	1	$x'.y.z$	m_3
1	0	0	$x.y'.z'$	m_4
1	0	1	$x.y'.z$	m_5
1	1	0	$x.y.z'$	m_6
1	1	1	$x.y.z$	m_7

Table 1.32: Four-Variable Minterms

a	b	c	d	Minterm	Notation
0	0	0	0	$a'.b'.c'.d'$	m_0
0	0	0	1	$a'.b'.c'.d$	m_1
0	0	1	0	$a'.b'.c.d'$	m_2
0	0	1	1	$a'.b'.c.d$	m_3
0	1	0	0	$a'.b.c'.d'$	m_4
0	1	0	1	$a'.b.c'.d$	m_5
0	1	1	0	$a'.b.c.d'$	m_6
0	1	1	1	$a'.b.c.d$	m_7
1	0	0	0	$a.b'.c'.d'$	m_8
1	0	0	1	$a.b'.c'.d$	m_9
1	0	1	0	$a.b'.c.d'$	m_{10}
1	0	1	1	$a.b'.c.d$	m_{11}
1	1	0	0	$a.b.c'.d'$	m_{12}
1	1	0	1	$a.b.c'.d$	m_{13}
1	1	1	0	$a.b.c.d'$	m_{14}
1	1	1	1	$a.b.c.d$	m_{15}

1.5.3 Maxterms

Maxterms are 2^n possible combinations of OR terms (or sum terms) with n variables such that the logical OR of all the variables is 0.

The maxterms can be formed with two, three, four, variables.

The 2^n maxterms are denoted as $M_0, M_1, M_2, M_3, \dots, M_q$ where $q = 2^n - 1$.

The 2^n combinations of OR terms, for $n = 2$ and 3 are shown below. The OR terms are formed from 2^n combinations of n -bit binary. In OR terms a literal is primed if its value is 1, and unprimed if its value is 0 so that the OR of all literals is always 0.

Let, $n = 2$

Let x and y be the two variables to denote two bits of binary.

When $n = 2$, $2^n = 2^2 = 4$

Therefore, four combinations of OR terms with two variables are possible as shown below:

$$0 \ 0 \quad x + y \quad (0 + 0 = 0) \quad x + y = M_0$$

$$0 \ 1 \quad x + y' \quad (0 + 1' = 0) \quad x + y' = M_1$$

$$1 \ 0 \Rightarrow x' + y \quad (1' + 0 = 0) \Rightarrow x' + y = M_2$$

$$1 \ 1 \quad x' + y' \quad (1' + 1' = 0) \quad x' + y' = M_3$$

Let, $n = 3$

Let x, y and z be the three variables to denote three bits of binary.

When $n = 3$, $2^n = 2^3 = 8$

Therefore, eight combinations of OR terms with three variables are possible as shown below:

$$0 \ 0 \ 0 \quad x + y + z \quad (0 + 0 + 0 = 0) \quad x + y + z = M_0$$

$$0 \ 0 \ 1 \quad x + y + z' \quad (0 + 0 + 1' = 0) \quad x + y + z' = M_1$$

$$0 \ 1 \ 0 \quad x + y' + z \quad (0 + 1' + 0 = 0) \quad x + y' + z = M_2$$

$$0 \ 1 \ 1 \Rightarrow x + y' + z' \quad (0 + 1' + 1' = 0) \Rightarrow x + y' + z' = M_3$$

$$1 \ 0 \ 0 \quad x' + y + z \quad (1' + 0 + 0 = 0) \quad x' + y + z = M_4$$

$$1 \ 0 \ 1 \quad x' + y + z' \quad (1' + 0 + 1' = 0) \quad x' + y + z' = M_5$$

$$1 \ 1 \ 0 \quad x' + y' + z \quad (1' + 1' + 0 = 0) \quad x' + y' + z = M_6$$

$$1 \ 1 \ 1 \quad x' + y' + z' \quad (1' + 1' + 1' = 0) \quad x' + y' + z' = M_7$$

Table 1.33: Two-Variable Maxterms

x	y	Maxterm	Notation
0	0	$x + y$	M_0
0	1	$x + y'$	M_1
1	0	$x' + y$	M_2
1	1	$x' + y'$	M_3

Table 1.34: Three-Variable Maxterms

x	y	z	Maxterm	Notation
0	0	0	$x + y + z$	M_0
0	0	1	$x + y + z'$	M_1
0	1	0	$x + y' + z$	M_2
0	1	1	$x + y' + z'$	M_3
1	0	0	$x' + y + z$	M_4
1	0	1	$x' + y + z'$	M_5
1	1	0	$x' + y' + z$	M_6
1	1	1	$x' + y' + z'$	M_7

Table 1.35: Four-Variable Maxterms

a	b	c	d	Maxterm	Notation
0	0	0	0	$a + b + c + d$	M_0
0	0	0	1	$a + b + c + d'$	M_1
0	0	1	0	$a + b + c' + d$	M_2
0	0	1	1	$a + b + c' + d'$	M_3
0	1	0	0	$a + b' + c + d$	M_4
0	1	0	1	$a + b' + c + d'$	M_5
0	1	1	0	$a + b' + c' + d$	M_6
0	1	1	1	$a + b' + c' + d'$	M_7
1	0	0	0	$a' + b + c + d$	M_8
1	0	0	1	$a' + b + c + d'$	M_9
1	0	1	0	$a' + b + c' + d$	M_{10}
1	0	1	1	$a' + b + c' + d'$	M_{11}
1	1	0	0	$a' + b' + c + d$	M_{12}
1	1	0	1	$a' + b' + c + d'$	M_{13}
1	1	1	0	$a' + b' + c' + d$	M_{14}
1	1	1	1	$a' + b' + c' + d'$	M_{15}

1.5.4 Standard Forms

There are three types of standard forms of expressing a Boolean function. They are,

1. Sum-of-products (SOP) form
2. Product-of-sums (POS) form
3. Canonical form

The **canonical form** is also expressed in SOP or POS form. The SOP or POS form is said to be canonical only if all the variables are present in every term of the SOP or POS form.

In SOP or POS form each term of the Boolean expression need not have all the variables of the function. Hence SOP and POS forms are also simplified forms of Boolean functions.

1.5.5 Sum-of-Products and Product-of-Sums Simplification

Sum-of-products (SOP) can be realized by AND operation of literals followed by OR operation of output of AND. Here, AND operation of literals are the products and OR operation of output of AND is the sum of product terms. **Product-of-sums (POS)** can be realized by OR operation of literals followed by AND operation of output of OR. Here, OR operation of literals are the sums and AND operation of output of OR is the product of sum terms. The realizations in SOP and POS forms are called **two-level realization of standard forms**.

The two-level realization of SOP form can be obtained by using AND gates to implement product terms followed by OR gate to get sum of product terms. The two-level realization of POS form can be obtained by using OR gates to implement sum terms followed by AND gate to get product of sum terms.

Since NAND and NOR gates are universal gates, the two-level realization of SOP form of a Boolean function can be obtained using only NAND gates. Similarly, two-level realization of POS form of a Boolean function can be obtained using only NOR gates.

Example 1.9

Draw the two-level realization of the following functions.

a) $F_1 = y' + xy + x'y'z'$ b) $F_2 = x(y' + z)(x' + y + z')$

Solution

a) $F_1 = y' + xy + x'y'z'$ - Sum-of-products

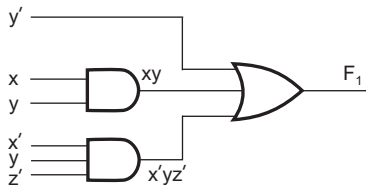


Fig. 1.

b) $F_2 = x(y' + z)(x' + y + z')$ - Product-of-sums

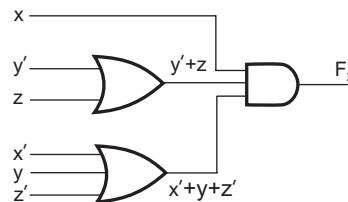


Fig. 2.

Example 1.10

Determine the complement of the following functions and show that SOP becomes POS and vice-versa.

a) $F_1 = y' + xy + x'y'z$ b) $F_2 = x(y' + z)(x' + y + z')$

Solution

a) $F_1 = y' + xy + x'y'z$ - Sum-of-products

⇓ Complement

$$\begin{aligned} F_1' &= (y' + xy + x'y'z)' \\ &= (y')'(xy)'(x'y'z)' \\ &= y(x' + y')(x + y' + z') \text{ - Product-of-sums} \end{aligned}$$

Using DeMorgan's theorem

$$(x + y)' = x'y'$$

$$(x \cdot y)' = x' + y'$$

b) $F_2 = x(y' + z)(x' + y + z')$ - Product-of-sums

⇓ Complement

$$\begin{aligned} F_2' &= (x(y' + z)(x' + y + z'))' \\ &= x' + (y' + z)' + (x' + y + z')' \\ &= x' + (yz') + (xy'z) \text{ - Sum-of-products} \end{aligned}$$

Using DeMorgan's theorem

$$(x + y)' = x'y'$$

$$(x \cdot y)' = x' + y'$$

1.5.6 Canonical Form

A Boolean function can be expressed algebraically from its truth table by forming a minterm for each combination of the variables that produces a 1 in the output and then taking the OR of all the minterms.

Similarly, a Boolean function can be expressed algebraically from its truth table by forming a maxterm for each combination of the variable that produces a 0 in the output and then taking the AND of all the maxterm. A Boolean function expressed as a sum of minterms or product of maxterms are said to be in canonical form.

The given function can be expressed in canonical form without using truth table. For sum of minterms insert sum of missing literal and its complement with "." operation and expand. Similarly for product of maxterms insert product of missing literal and its complement with "+" operation and expand.

1.5.7 Complement of a Function

If, F is a function then F' is complement of F . The complement of a function can be directly evaluated using Boolean theorems and postulates. Alternatively, the complement can also be obtained by using duality. In duality replace "+" by "." and "." by "+" and variables by its complement.

It can be observed that complement of sum-of-products will be product-of-sums and vice-versa

i.e, Product-of-sums $\xleftrightarrow{\text{Complement}}$ Sum-of-products

Note: This concept holds good for complement of canonical forms also.

Example 1.11

Determine the complement of the following Boolean functions.

a) $F_1 = xy + x'z$ b) $F_2 = wx + yz$ c) $F_3 = xy' + x'y$

Solution

a) $F_1 = xy + x'z$

Case i: Direct Evaluation of Complement

$$\begin{aligned} F_1 &= xy + x'z \\ &\Downarrow \text{Complement} \\ F_1' &= (xy + x'z)' \\ &= (xy)' \cdot (x'z)' \\ &= (x' + y') \cdot ((x')' + z') \\ &= (x' + y') \cdot (x + z') \end{aligned}$$

Using DeMorgan's theorem

$$\begin{aligned} (x + y)' &= x'y' \\ (x \cdot y)' &= x' + y' \end{aligned}$$

Case ii: Complement using Duality

$$\begin{aligned} &x \cdot y + x' \cdot z \\ &\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \text{ Replace by dual elements} \\ &(x' + y') \cdot (x + z') \\ &\Downarrow \\ &(x' + y') \cdot (x + z') = F_1' \end{aligned}$$

b) $F_2 = wx + yz$

Case i: Direct Evaluation of Complement

$$\begin{aligned} F_2 &= wx + yz \\ &\Downarrow \text{Complement} \\ F_2' &= (wx + yz)' \\ &= (wx)' \cdot (yz)' \\ &= (w' + x') \cdot (y' + z') \end{aligned}$$

Using DeMorgan's theorem

$$\begin{aligned} (x + y)' &= x'y' \\ (x \cdot y)' &= x' + y' \end{aligned}$$

Case ii: Complement using Duality

$$\begin{aligned} &w \cdot x + y \cdot z \\ &\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\ &(w' + x') \cdot (y' + z') \\ &\Downarrow \text{ Replace by dual elements} \\ &(w' + x') \cdot (y' + z') = F_2' \end{aligned}$$

c) $F_3 = xy' + x'y$

Case i: Direct Evaluation of Complement

$$F_3 = xy' + x'y$$

⇓ Complement

$$\begin{aligned} F_3' &= (xy' + x'y)' \\ &= (xy')' \cdot (x'y)' \\ &= (x' + y) \cdot (x + y') \end{aligned}$$

Using DeMorgan's theorem
 $(x + y)' = x'y'$
 $(x \cdot y)' = x' + y'$

Case ii: Complement using Duality

$$\begin{aligned} &x \cdot y' + x' \cdot y \\ &\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \text{ Replace by dual elements} \\ &(x' + y) \cdot (x + y') \\ &\downarrow \\ &(x' + y) \cdot (x + y') = F_3' \end{aligned}$$

1.5.8 Implementation of Boolean Functions using Universal Gates

The NAND and NOR gates are called *universal gates*. When a Boolean function is expressed in SOP form, it can be realized using only NAND gates. When a Boolean function is expressed in POS form, it can be realized using only NOR gates.

Example 1.12

Express the function $F = x + y'z$, in sum of minterms and product of maxterms and verify the result by simplification using Boolean theorems and postulates.

Solution

Table 1: Truth Table

Input Variables			Minterm		Maxterm		Complement of y	Product Term	Function Output
x	y	z					y'	y'z	F
0	0	0	m ₀	x'y'z'	M ₀	x + y + z	1	0	0
0	0	1	m ₁	x'y'z	M ₁	x + y + z'	1	1	1
0	1	0	m ₂	x'y z'	M ₂	x + y' + z	0	0	0
0	1	1	m ₃	x'y z	M ₃	x + y' + z'	0	0	0
1	0	0	m ₄	x y'z'	M ₄	x' + y + z	1	0	1
1	0	1	m ₅	x y'z	M ₅	x' + y + z'	1	1	1
1	1	0	m ₆	x y z'	M ₆	x' + y' + z	0	0	1
1	1	1	m ₇	x y z	M ₇	x' + y' + z'	0	0	1

A function can be expressed as a sum of minterms for which the function output is 1. Here, F is 1, when m₁, m₄, m₅, m₆ and m₇ are inputs.

$$\begin{aligned} \therefore F &= m_1 + m_4 + m_5 + m_6 + m_7 \\ &= \sum m(1, 4, 5, 6, 7) = x'y'z + xy'z' + xy'z + xyz' + xyz \end{aligned}$$

A function can be expressed as a product of maxterms for which the function output is 0. Here, F is 0, when M_0 , M_2 and M_3 are inputs

$$\begin{aligned} F &= M_0 M_2 M_3 \\ &= \prod M(0, 2, 3) \\ &= (x + y + z)(x + y' + z)(x + y' + z') \end{aligned}$$

Proof:

Case i:

$$\begin{aligned} F &= m_1 + m_4 + m_5 + m_6 + m_7 \\ &= x'y'z + xy'z' + xy'z + xy'z' + xyz \\ &= x'y'z + xy'(z' + z) + xy(z' + z) \\ &= x'y'z + xy'.1 + xy.1 \\ &= x'y'z + xy' + xy \\ &= x'y'z + x(y' + y) \\ &= x'y'z + x.1 \\ &= x'y'z + x \\ &= (x' + x)(y'z + x) \\ &= 1.y'z + x \\ &= x + y'z \end{aligned}$$

$$x + x' = 1$$

$$x.1 = x$$

Case ii:

$$\begin{aligned} F &= M_0 M_2 M_3 \\ &= (x + y + z)(x + y' + z)(x + y' + z') \\ &= (x + z + y)(x + z + y')(x + y' + z') \\ &= (x + z + (yy'))(x + y' + z') \\ &= (x + z + 0)(x + y' + z') \\ &= x + (z(y' + z')) \\ &= x + zy' + zz' \\ &= x + y'z + 0 \\ &= x + y'z \end{aligned}$$

$$x.y' = 0$$

$$x + 0 = x$$

Example 1.13

Express the Boolean function $F(p, q, r) = (pq + r)(q + pr)$ as sum of minterms and product of maxterms.

Solution:

i) Sum of Minterms

$$\begin{aligned} F(p, q, r) &= (pq + r)(q + pr) \\ &= pqq + pqpr + rq + rpr \\ &= pq + pqr + rq + rp \\ &= pq(r + r') + pqr + rq(p + p') + rp(q + q') \end{aligned}$$

$$\begin{aligned}
 &= pqr + pqr' + \cancel{pqr} + \cancel{pqr} + p'qr + \cancel{pqr} + pqr' \\
 &= pqr + pqr' + p'qr + pqr' \\
 &= p'qr + pqr' + pqr' + pqr \\
 &= m_3 + m_5 + m_6 + m_7 = \sum m(3, 5, 6, 7)
 \end{aligned}$$

Repeated terms are considered once.

ii) Product of Maxterms

$$\begin{aligned}
 F(p, q, r) &= (pq + r)(q + pr) \\
 &= (p + r)(q + r)(q + p)(\cancel{q + r}) \\
 &= (p + r)(q + r)(q + p) \\
 &= (p + r + (qq'))((pp') + q + r)(p + q + (rr')) \\
 &= (p + r + q)(p + r + q')(p + q + r)(p + q + r') \\
 &= (p + q + r)(p + r + q')(p' + q + r)(p + q + r') \\
 &= (p + q + r)(p + q + r')(p + q' + r)(p' + q + r) \\
 &= M_0 M_1 M_2 M_4 = \prod M(0, 1, 2, 4)
 \end{aligned}$$

Missing literal in first term is q, second term is p and third term is r.

Example 1.14

Convert the following in the proper canonical form and write the decimal notation.

a) $R = F(x, y, z) = (x + y)(x' + z)$ into maxterm canonical form.

b) $Z = F(a, b, c) = ab + b'c + ac$ into minterm canonical form.

Solution

a) Maxterm canonical form (POS)

$$\begin{aligned}
 R = F(x, y, z) &= (x + y)(x' + z) \\
 &= (x + y + (zz'))(x' + z + (yy')) \\
 &= ((x + y + z)(x + y + z'))((x' + y + z)(x' + y' + z)) \\
 &\quad (M_0) \quad (M_1) \quad (M_4) \quad (M_6) \\
 &= M_0 M_1 M_4 M_6 = \prod M(0, 1, 4, 6)
 \end{aligned}$$

Missing literal in first term is z and second term is y.

b) Minterm canonical form (SOP)

$$\begin{aligned}
 Z = F(a, b, c) &= (ab + b'c + ac) \\
 &= ab(c + c') + b'c(a + a') + ac(b + b') \\
 &= abc + abc' + ab'c + a'b'c + \cancel{abc} + \cancel{ab'c} \\
 &= abc + abc' + ab'c + a'b'c \\
 &\quad (m_7) \quad (m_6) \quad (m_5) \quad (m_1) \\
 &= a'b'c + ab'c + abc' + abc \\
 &= m_1 + m_5 + m_6 + m_7 = \sum m(1, 5, 6, 7)
 \end{aligned}$$

Missing literal in first term is c, second term is a and third term is b.

Repeated terms are considered once.

Example 1.15

Express the following Boolean functions in canonical form and implement the Boolean functions using either only NAND or NOR gates.

a) $F_1 = xy + y'z$ b) $F_2 = (x + y)(y + z)$ c) $F_3 = x + y'z$

Solution

a) $F_1 = xy + y'z$

Case i: SOP

$$\begin{aligned} F_1 &= xy + y'z \\ &= xy(z + z') + (x + x')y'z \\ &= xyz + xyz' + x'y'z + x'y'z \\ &\quad (m_7) \quad (m_6) \quad (m_5) \quad (m_1) \\ &= x'y'z + x'y'z + xyz' + xyz \\ &= m_1 + m_5 + m_6 + m_7 \end{aligned}$$

Missing literal in first term is z and second term is x.

$$x + x' = 1$$

$$x \cdot 1 = x$$

The SOP form of F_1 is realized using only NAND gates as shown in Fig. 1.

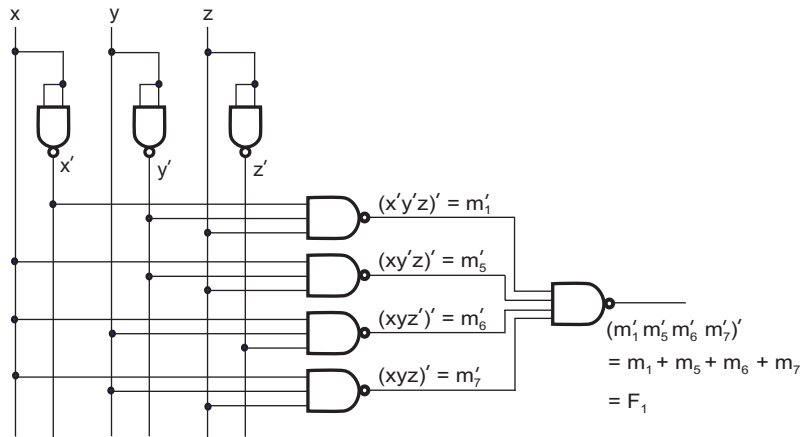


Fig. 1: Logic circuit of F_1 using only NAND gates.

Case ii: POS

$$\begin{aligned} F_1 &= xy + y'z \\ &= (x + y'z)(y + y'z) \\ &= (x + y')(x + z)(y + y')(y + z) \\ &= (x + y' + (zz'))(x + z + (yy'))((xx') + y + z) \\ &= (x + y' + z)(x + y' + z')(x + z + y)(x + z + y')(x + y + z)(x' + y + z) \\ &= (x + y' + z)(x + y' + z')(x + y + z) \cancel{(x + y' + z)} \cancel{(x + y + z)} (x' + y + z) \\ &\quad (M_2) \quad (M_3) \quad (M_0) \quad (M_2) \quad (M_0) \quad (M_4) \\ &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z) \\ &= M_0 M_2 M_3 M_4 \end{aligned}$$

$$\begin{aligned} x x' &= 0 \\ x + 0 &= x \end{aligned}$$

Missing literal in first term is z, second term is y and third term is x.

Repeated maxterms are considered only one time

The POS form of F_1 is realized using only NOR gates as shown in Fig. 2.

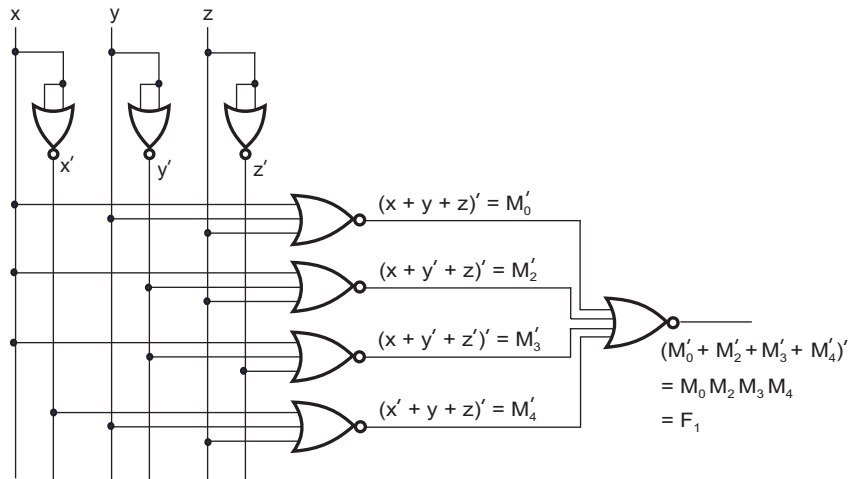


Fig. 2: Logic circuit of F_1 using only NOR gates.

b) $F_2 = (x + y)(y + z)$

Case i: SOP

$$\begin{aligned}
 F_2 &= (x + y)(y + z) = x(y + z) + y(y + z) \\
 &= xy + xz + yy + yz = xy + xz + y + yz \\
 &= xy(z + z') + xz(y + y') + (x + x')y(z + z') + (x + x')yz \\
 &= xyz + xyz' + xzy + xzy' + (xy + x'y)(z + z') + xyz + x'yz \\
 &= \underset{(m_7)}{xyz} + \underset{(m_6)}{xyz'} + \underset{(m_7)}{xzy} + \underset{(m_5)}{xzy'} + \underset{(m_7)}{xyz} + \underset{(m_6)}{xzy'} + \underset{(m_3)}{x'yz} + \underset{(m_2)}{x'yz'} + \underset{(m_7)}{xyz} + \underset{(m_3)}{x'yz'} \\
 &= x'y'z' + x'yz + xy'z + xyz' + xyz \\
 &= m_2 + m_3 + m_5 + m_6 + m_7
 \end{aligned}$$

$$yy = y$$

Insert missing literals

$$x + x' = 1$$

$$x \cdot 1 = x$$

Repeated minterms are considered only one time

The SOP form of F_2 is realized using only NAND gates as shown in Fig. 3.

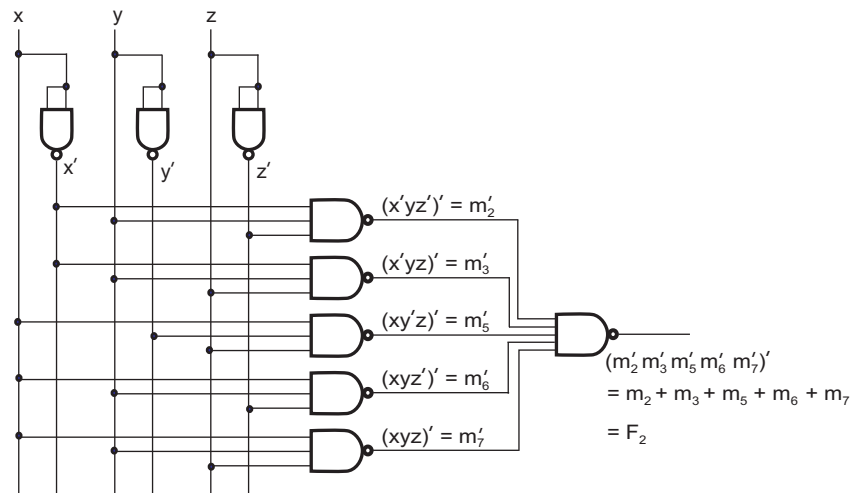


Fig. 3: Logic circuit of F_2 using only NAND gates.

Case ii: POS

$$F_2 = (x + y)(y + z)$$

$$= ((x + y) + (zz'))((xx') + (y + z))$$

$$= (x + y + z)(x + y + z')(x + y + z)(x' + y + z)$$

(M_0)
 (M_1)
 (M_0)
 (M_4)

$$= (x + y + z)(x + y + z')(x' + y + z)$$

$$= M_0 M_1 M_4$$

The POS form of F_2 is realized using only NOR gates as shown in Fig. 4.

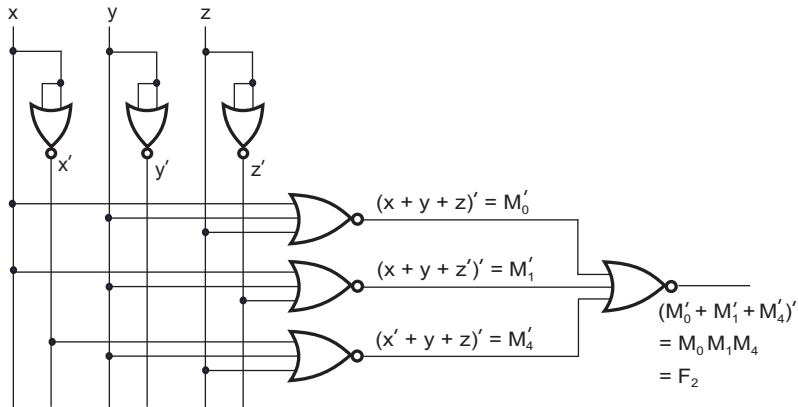


Fig. 4: Logic circuit of F_2 using only NOR gates.

Missing literal in first term is z and second term is x.

$$x x' = 0$$

$$x + 0 = x$$

Repeated maxterms are considered only one time

c) $F_3 = x + y'z$

Case i: SOP

$$F_3 = x + y'z$$

$$= x(y + y') + (x + x')y'z$$

$$= xy + xy' + xy'z + x'y'z$$

$$= xy(z + z') + xy'(z + z') + xy'z + x'y'z$$

$$= xy z + xy z' + xy' z + xy' z' + xy' z + x'y' z$$

(m_7)
 (m_6)
 (m_5)
 (m_4)
 (m_5)
 (m_1)

$$= x'y'z + xy'z' + xy'z + xy'z' + xy'z$$

$$= m_1 + m_4 + m_5 + m_6 + m_7$$

The SOP form of F_3 is realized using only NAND gates as shown in Fig. 5.

$$x \cdot 1 = x$$

$$x + x' = 1$$

Missing literal in first term is y and second term is x.

Missing literal in first term and second term is z.

Repeated minterms are considered only one time

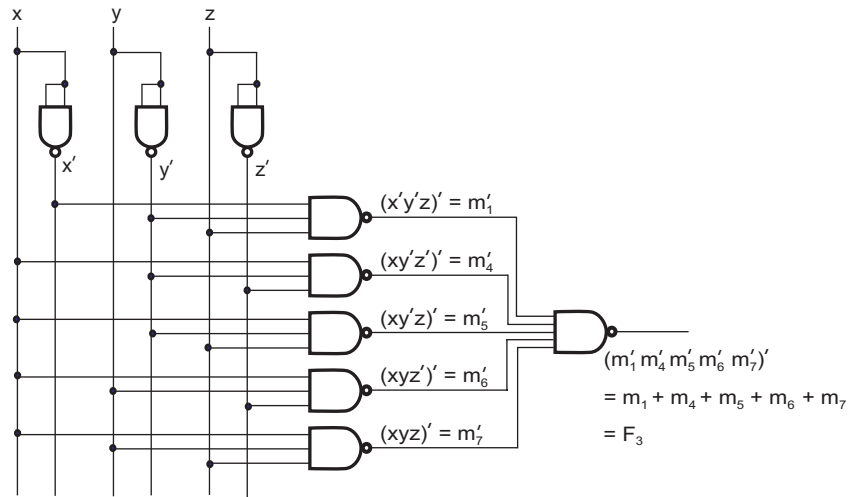


Fig. 5: Logic circuit of F_3 using only NAND gates.

Case ii: POS

$$\begin{aligned}
 F_3 &= x + y'z \\
 &= (x + y')(x + z) \\
 &= ((x + y') + (zz'))((x + z) + (yy')) \\
 &= (x + y' + z)(x + y' + z')(x + y + z)(\overline{x + y' + z}) \\
 &\quad (M_2) \quad (M_3) \quad (M_0) \quad (M_2) \\
 &= (x + y + z)(x + y' + z)(x + y' + z') \\
 &= M_0 M_2 M_3
 \end{aligned}$$

$x \cdot 1 = x$
$x + x' = 1$

Missing literal in first term is z and second term is y.

Repeated maxterms are considered only one time

The POS form of F_3 is realized using only NOR gates as shown in Fig. 6.

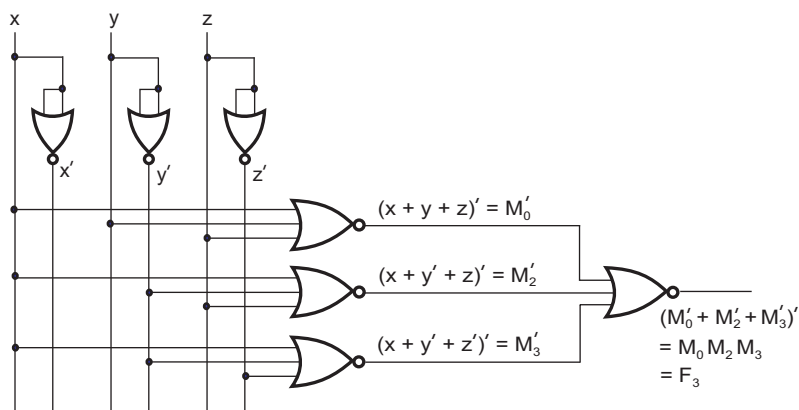


Fig. 6: Logic circuit of F_3 using only NOR gates.

Example 1.16

Express the following function in sum of minterms and product of maxterms

$$F(A, B, C, D) = A'B + BD + AC'$$

Solution**Table 1: Truth Table of F**

Input Variables				Minterm		Maxterm		Complement of A and C		Product Terms			Function Output
A	B	C	D					A'	C'	A'B	BD	AC'	
0	0	0	0	m ₀	A'B'C'D'	M ₀	A+B+C+D	1	1	0	0	0	0
0	0	0	1	m ₁	A'B'C'D	M ₁	A+B+C+D'	1	1	0	0	0	0
0	0	1	0	m ₂	A'B'CD'	M ₂	A+B+C'+D	1	0	0	0	0	0
0	0	1	1	m ₃	A'B'CD	M ₃	A+B+C'+D'	1	0	0	0	0	0
0	1	0	0	m ₄	A'BC'D'	M ₄	A+B'+C+D	1	1	1	0	0	1
0	1	0	1	m ₅	A'BC'D	M ₅	A+B'+C+D'	1	1	1	1	0	1
0	1	1	0	m ₆	A'BCD'	M ₆	A+B'+C'+D	1	0	1	0	0	1
0	1	1	1	m ₇	A'BCD	M ₇	A+B'+C'+D'	1	0	1	1	0	1
1	0	0	0	m ₈	AB'C'D'	M ₈	A'+B+C+D	0	1	0	0	1	1
1	0	0	1	m ₉	AB'C'D	M ₉	A'+B+C+D'	0	1	0	0	1	1
1	0	1	0	m ₁₀	AB'CD'	M ₁₀	A'+B+C'+D	0	0	0	0	0	0
1	0	1	1	m ₁₁	AB'CD	M ₁₁	A'+B+C'+D'	0	0	0	0	0	0
1	1	0	0	m ₁₂	ABC'D'	M ₁₂	A'+B'+C+D	0	1	0	0	1	1
1	1	0	1	m ₁₃	ABC'D	M ₁₃	A'+B'+C+D'	0	1	0	1	1	1
1	1	1	0	m ₁₄	ABCD'	M ₁₄	A'+B'+C'+D	0	0	0	0	0	0
1	1	1	1	m ₁₅	ABCD	M ₁₅	A'+B'+C'+D'	0	0	0	1	0	1

A function can be expressed as a sum of minterms for which the function output is 1. Here, F is 1, when m₄, m₅, m₆, m₇, m₈, m₉, m₁₂, m₁₃ and m₁₅ are inputs.

$$\therefore F = m_4 + m_5 + m_6 + m_7 + m_8 + m_9 + m_{12} + m_{13} + m_{15}$$

$$= A'BC'D' + A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'C'D + ABC'D' + ABC'D + ABCD$$

A function can be expressed as a product of maxterms for which the function output is 0. Here, F is 0, when M₀, M₁, M₂, M₃, M₁₀, M₁₁ and M₁₄ are inputs.

$$F = M_0 M_1 M_2 M_3 M_{10} M_{11} M_{14}$$

$$= (A+B+C+D)(A+B+C+D')(A+B+C'+D)(A+B+C'+D')(A'+B+C'+D)(A'+B+C'+D')(A'+B'+C'+D)$$

1.5.9 Simplification of Boolean Expressions or Functions

Each variable with in a term of a Boolean expression is called a *literal*. Example of 7 literals and 10 literals Boolean expression are shown below:

$$a' b + c d + a b d$$

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$\textcircled{1} \textcircled{2} \textcircled{3} \textcircled{4} \textcircled{5} \textcircled{6} \textcircled{7} \rightarrow 7 \text{ literals}$$

$$a b c + b' c + a c' + a b' c$$

$$\textcircled{1} \textcircled{2} \textcircled{3} \textcircled{4} \textcircled{5} \textcircled{6} \textcircled{7} \textcircled{8} \textcircled{9} \textcircled{10} \rightarrow 10 \text{ literals}$$

Reducing the number of literals in a Boolean expression or function will simplify the implementation of function by logic gates with minimum number of gates. The implementation of Boolean function using logic gates is also called MSI (Medium Scale Integration) Circuit.

The Boolean expression can be reduced or simplified using postulates and theorems of Boolean algebra but there is no standard procedure for simplification using theorems.

Alternatively, Boolean functions can be simplified using following two methods.

- Karnaugh map method
- Quine-McCluskey method or Table method

These two methods have standard procedure for simplification of Boolean functions.

Example 1.17

Reduce the number of literals in following expressions.

a) $xy + xy'$ b) $xy + x(wz + wz')$ c) $(A+B)'(A'+B)'$

Solution

a) $xy + xy'$

$$xy + xy' = x(y + y')$$

$$= x.1$$

$$= x$$

$$y + y' = 1$$

$$x.1 = x$$

The given expression $xy + xy'$ is 4 literal expression which is reduced to single literal.

b) $xy + x(wz + wz')$

$$xy + x(wz + wz') = x y + x(w(z + z'))$$

$$= x y + x(w.1)$$

$$= x y + x w$$

$$= x.(y + w)$$

$$z + z' = 1$$

$$w.1 = w$$

The given expression $xy + x(wz + wz')$ is 7 literals expression which is reduced to three literals.

c) $(A + B)' (A' + B')'$

$$(A + B)' (A' + B')' = (A + B)' \cdot ((A')' \cdot (B')')$$

$$= (A' \cdot B') \cdot (A \cdot B)$$

$$= A' \cdot B' \cdot A \cdot B$$

$$= A' \cdot A \cdot B \cdot B' = 0 \cdot 0 = 0$$

Using DeMorgan's law
 $(x + y)' = x' \cdot y'$

$$x \cdot x' = 0$$

The given expression is a 4 literal expression which is reduced to null expression.

1.6 Karnaugh Map (or K-map) Minimization

(AU, Nov/Dec'22, 3 Marks)

K-map is a pictorial form of truth table and used to simplify Boolean functions. Simplification of Boolean function using K-map is simple and straight forward. A K-map is a diagram made up of squares with each square representing one minterm. Alternatively, maxterm can be used to construct K-map in which each square represent a maxterm.

When minterms are considered for simplification the resultant Boolean function will be in sum-of-products form. When maxterms are considered for simplification the resultant Boolean function will be in product-of-sums form.

Note: The simplified Boolean function using K-map is not unique. Sometimes there may be multiple solutions.

For n variable Boolean function the K-map will have 2^n squares. Each square represent a minterm or maxterm. The literals of minterm are split and arranged as rows and columns. While arranging the literal/minterms only one change is allowed if we move from one row to next row or from one column to next column.

1.6.1 Analysis and Design Procedures of Combinational Circuits using K-map

1. Construct the truth table of Boolean function.
2. Draw K-map with 2^n squares where n is number of variables in Boolean function.
3. Enter the value of function or output in the squares as either 1 or 0.
4. The squares with entry 1 are the minterms that represent the function.
5. Each square with 1 is an **implicant**.
6. Combine adjacent 1's to form **prime implicants**.
7. A prime implicant with single 1 represent n literal product term.
8. A prime implicant with two 1's represent n – 1 literal product term.
9. A prime implicant with four 1's represent n – 2 literal product term and so on.
10. The literals represented by a prime implicant are the literals that present in all the squares of the prime implicant. (or the literals common to all the squares of the prime implicant).
11. While forming prime implicants any number of over lapping is allowed in horizontal and vertical directions. But the diagonal elements cannot be combined to form prime implicants.
12. While forming prime implicants the K-map can be folded in any direction to get adjacent 1's.
13. The simplified Boolean function is sum of all the product terms of prime implicant.

14. If all minterms are 1's then all squares of K-map will be filled by 1 and function value is 1.
15. The logic or MSI (Medium Scale Integration) circuit of the simplified Boolean function can be drawn using logic gates.

Table 1.36: Relationship between Number of Adjacent Squares and Literals in the Product Term

Number of Adjacent Squares in Prime Implicants	Number of Literals in a Product Term in an n-variable Map			
	n = 2	n = 3	n = 4	n = 5
1	2	3	4	5
2	1	2	3	4
4	0	1	2	3
8		0	1	2
16			0	1
32				0

1.6.2 Two-Variable K-map

Let, n = Number of variables in Boolean function

x, y = Two variables of Boolean function

Therefore, $n = 2$, for two-variable K-map

Here, $2^n = 2^2 = 4$

Therefore, two-variable, K-map will have 4 squares. The two-variable K-map and arrangements of literals and minterms are shown in Fig. 1.10.

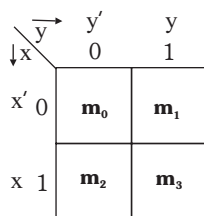


Fig. a: Arrangement of literals and minterms in 2-variable K-map.

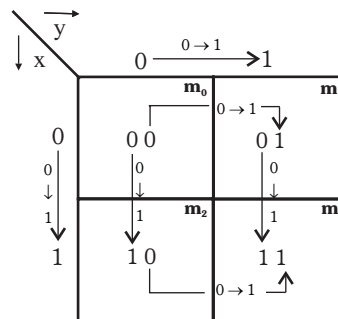
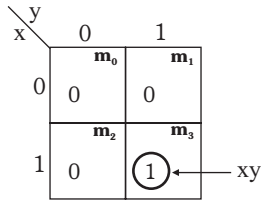
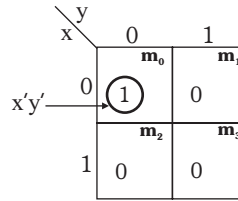
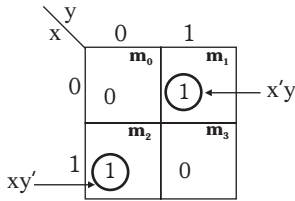
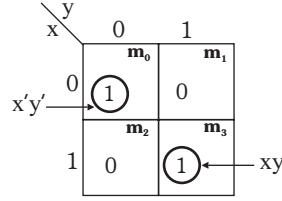


Fig. b: Change in bit values of literals and minterms in 2-variable K-map.

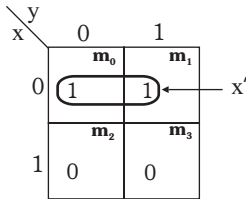
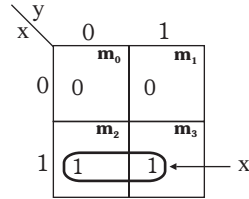
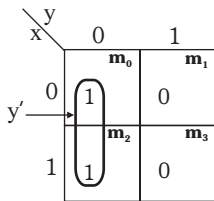
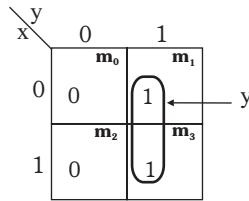
Fig. 1.10: Two-variable K-map.

Some examples of two-variable K-map with single prime implicant formed using single 1's is shown in Figs. 1.11 and 1.12. Since the prime implicants shown in Figs. 1.11 and 1.12 are formed using single 1's in two-variable K-map, each prime implicant represents a product term of two literals.

The K-maps in Fig. 1.11 have only one prime implicant and so the Boolean function, F is given by prime implicant of the K-map. The K-maps in Fig. 1.12 has two prime implicants and so the Boolean function, F is given by sum of the two prime implicants of the K-map.

**Fig. a:** K-map for, $F = xy$.**Fig. b:** K-map for, $F = x'y'$.**Fig. 1.11:** Two-variable K-map with single prime implicant formed using single 1's.**Fig. a:** K-map for, $F = x'y + xy'$.**Fig. b:** K-map for, $F = x'y' + xy$.**Fig. 1.12:** Two-variable K-map with two prime implicants formed using single 1's.

Some examples of two-variable K-map with single prime implicant formed using two adjacent 1's is shown in Fig. 1.13. Since the prime implicants shown in Fig. 1.13 are formed using two adjacent 1's in two-variable K-map, each prime implicant represents a single literal. Since the K-map have only one prime implicant the Boolean function, F is given by prime implicant of the K-map.

**Fig. a:** K-map for, $F = x'$.**Fig. b:** K-map for, $F = x$.**Fig. c:** K-map for, $F = y'$.**Fig. d:** K-map for, $F = y$.**Fig. 1.13:** Two-variable K-map with prime implicant formed using two adjacent 1's.

Some examples of two-variable K-map with two overlapping prime implicants formed using two adjacent 1's is shown in Fig. 1.14. Since the prime implicants shown in Fig. 1.14 are formed using two adjacent 1's in two-variable K-map, each prime implicant represents a single literal. Since the K-map has two prime implicant the Boolean function, F is given by sum of two prime implicants of the K-map.

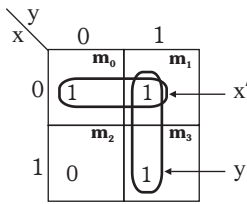
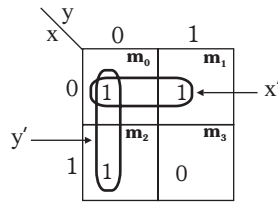
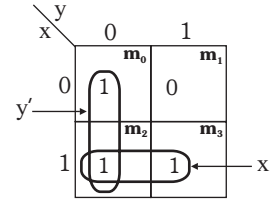
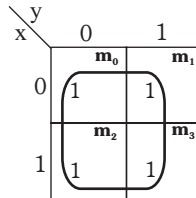
Fig. a: K-map for, $F = x' + y$.Fig. b: K-map for, $F = x' + y'$.Fig. c: K-map for, $F = x + y'$.

Fig. 1.14: Two-variable K-map with two overlapping prime implicants formed using two adjacent 1's.

When all the entries of K-map (or all minterms) are 1's then the Boolean function value is 1 ($F=1$) as shown in Fig. 1.15.

Fig. 1.15: Two-variable K-map with all minterms equal to 1 and $F=1$.

1.6.3 Three-Variable K-map

Let, n = Number of variables in Boolean function

x, y, z = Three variables of Boolean function

Therefore, $n = 3$, for three-variable K-map

Here, $2^n = 2^3 = 8$

Therefore, three-variable, K-map will have 8 squares. The three-variable K-map and arrangements of literals and minterms are shown in Fig. 1.16.

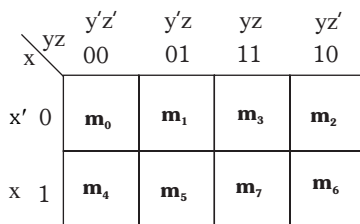


Fig. a: Arrangement of literals and minterms in 3-variable K-map.

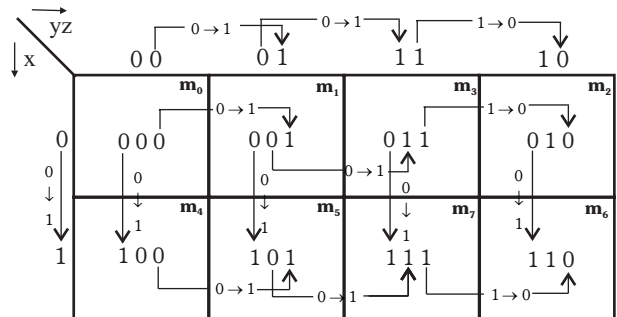


Fig. b: Change in bit values of literals and minterms in 3-variable K-map.

Some examples of three-variable K-map with prime implicant formed using single 1's are shown in Figs. 1.17 and 1.18. Since the prime implicants shown in Figs. 1.17 and 1.18 are formed using single 1's in three-variable K-map, each prime implicant represents a product term of three literals.

The K-maps in Fig. 1.17 have only one prime implicant and so the Boolean function, F is given by prime implicant of the K-map. The K-maps in Fig. 1.18 has two prime implicant and so the Boolean function, F is given by sum of the two prime implicants of the K-map.

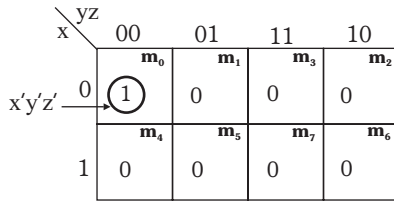


Fig. a: K-map for, $F = x'y'z'$.

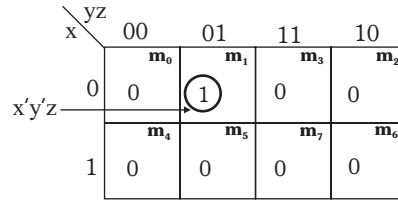


Fig. b: K-map for, $F = x'y'z$.

Fig. 1.17: Three-variable K-map with prime implicant formed using single 1's.

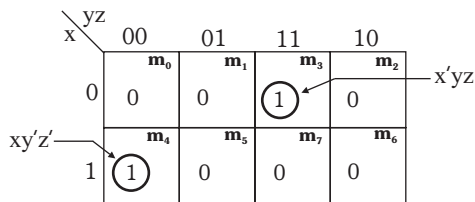


Fig. a: K-map for, $F = x'yz + xy'z'$.

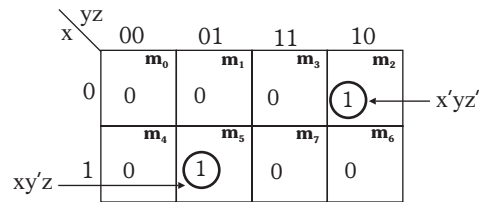


Fig. b: K-map for, $F = x'yz' + xy'z$.

Fig. 1.18: Three-variable K-map with two prime implicants formed using single 1's.

Some examples of three-variable K-map with two prime implicants formed using two adjacent 1's are shown in Fig. 1.19. Since the prime implicants shown in Fig. 1.19 are formed using two adjacent 1's in three-variable K-map, each prime implicant represents a product term of two literals. Since the K-map has two prime implicants the Boolean function, F is given by sum of the two prime implicants of the K-map.

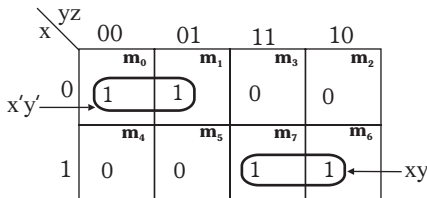


Fig. a: K-map for, $F = x'y' + xy$.

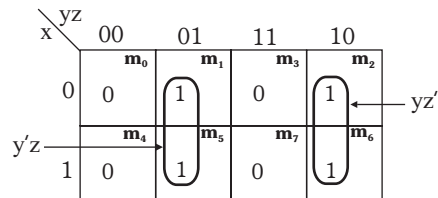


Fig. b: K-map for, $F = y'z + yz'$.

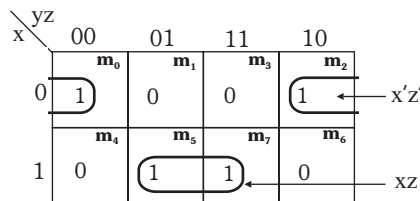


Fig. c: K-map for, $F = x'z' + xz$.

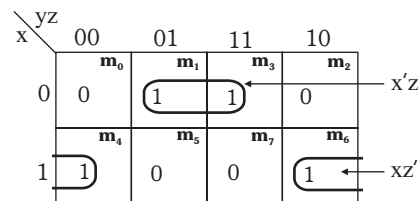


Fig. d: K-map for, $F = x'z + xz'$.

Fig. 1.19: Three-variable K-map with two prime implicants formed using two adjacent 1's.

Some examples of three-variable K-map with two overlapping prime implicants formed using two adjacent 1's and a prime implicant with single 1 are shown in Fig. 1.20. The prime implicants formed using two adjacent 1's in three-variable K-map, represents a product term of two literals and the prime implicant with single 1 represents a product term of three literal. Since the K-map has three prime implicant the Boolean function, F is given by sum of three prime implicants of the K-map.

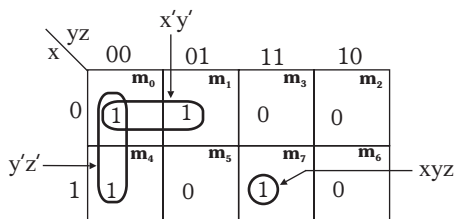


Fig. a: K-map for, $F = xyz + x'y' + y'z'$.

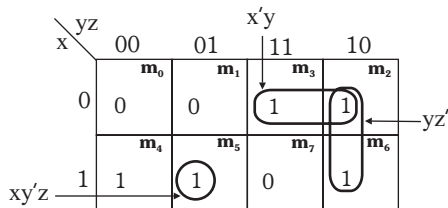


Fig. b: K-map for, $F = xy'z + yz' + x'y$.

Fig. 1.20: Three-variable K-map with prime implicants formed using single 1's and two overlapping two adjacent 1's.

Some examples of three-variable K-map with prime implicant formed using four adjacent 1's are shown in Fig. 1.21. Since the prime implicants shown in Fig. 1.21 are formed using four adjacent 1's in three-variable K-map, each prime implicant represents a single literal. Since the K-map have only one prime implicant the Boolean function, F is given by prime implicant of the K-map.

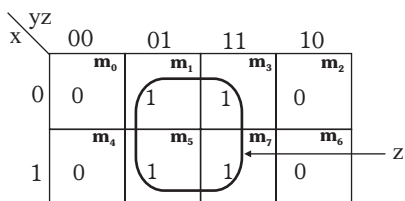


Fig. a: K-map for, $F = z$.

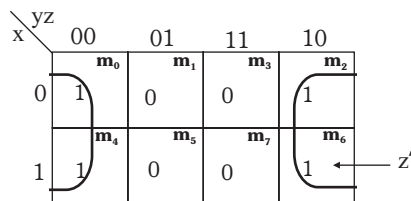


Fig. b: K-map for, $F = z'$.

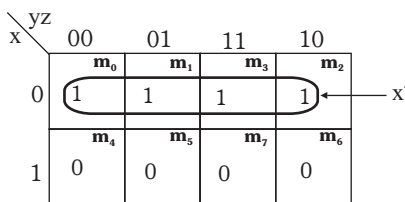


Fig. c: K-map for, $F = x'$.

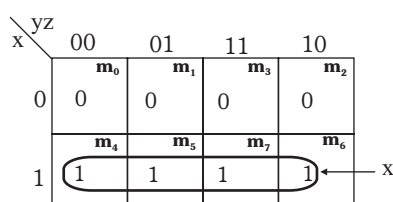


Fig. d: K-map for, $F = x$.

Fig. 1.21: Three-variable K-map with prime implicant formed using four adjacent 1's.

Some examples of three-variable K-map with two overlapping prime implicants formed using four adjacent 1's and two adjacent 1's are shown in Figs. 1.22 and 1.23. The prime implicants formed using two adjacent 1's in three-variable K-map, represents a product term of two literals and the prime implicant with four adjacent 1's represents a single literal. Since the K-map has two prime implicant the Boolean function, F is given by sum of two prime implicants of the K-map.

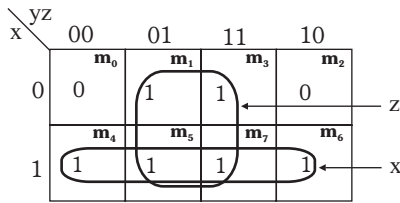
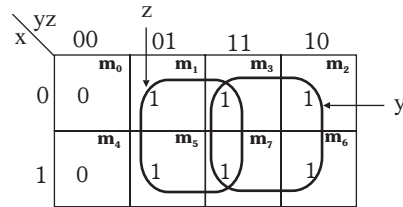
Fig. a: K-map for, $F = x + z$.Fig. b: K-map for, $F = y + z$.

Fig. 1.22: Three-variable K-map with two overlapping prime implicants formed using four adjacent 1's.

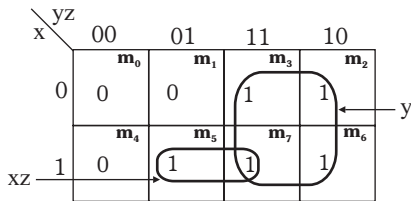
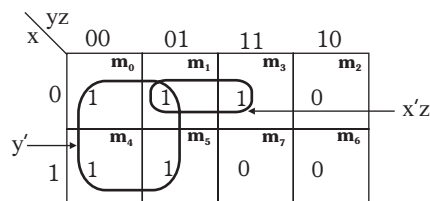
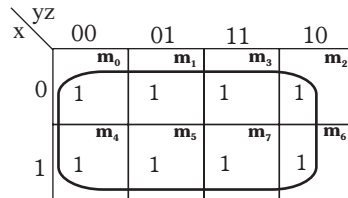
Fig. a: K-map for, $F = y + xz$.Fig. b: K-map for, $F = y' + x'z$.

Fig. 1.23: Three-variable K-map with overlapping prime implicants formed using two adjacent 1's and four adjacent 1's.

When all the entries of K-map (or all minterms) are 1's then the Boolean function value is 1 ($F = 1$) as shown in Fig. 1.24.

Fig. 1.24: Three-variable K-map with all minterms equal to 1 and $F=1$.

1.6.4 Four-Variable K-map

Let, n = Number of variables in Boolean function

w, x, y, z = Four variables of Boolean function

Therefore, $n = 4$, for four-variable K-map

Here, $2^n = 2^4 = 16$

Therefore, four-variable K-map will have 16 squares. The four-variable K-map and arrangements of literals and minterms are shown in Fig. 1.25.

	$y'z'$	$y'z$	yz	yz'
wx	00	01	11	10
$w'x'$ 00	m_0	m_1	m_3	m_2
$w'x'$ 01	m_4	m_5	m_7	m_6
wx 11	m_{12}	m_{13}	m_{15}	m_{14}
wx' 10	m_8	m_9	m_{11}	m_{10}

Fig. a: Arrangement of literals and minterms in four-variable K-map.

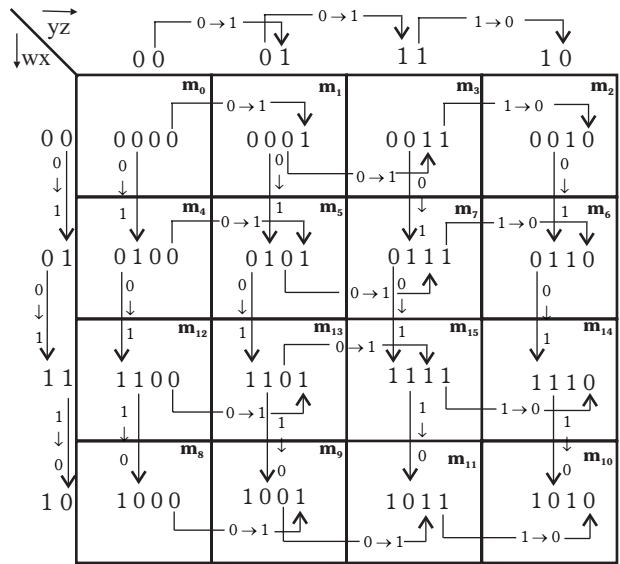


Fig. b: Change in bit values of literals and minterms in four-variable K-map.

Fig. 1.25: Four-variable K-map.

Some examples of four-variable K-map with prime implicants formed using four adjacent 1's, two adjacent 1's and single 1's are shown in Fig. 1.26. In four-variable K-map, the prime implicants formed using four adjacent 1's represents a product term of two literals, the prime implicants with two adjacent 1's represents a product term of three literals and the prime implicant with single 1 represents a product term of four literals. Boolean function, F is given by sum of prime implicants of the K-map.

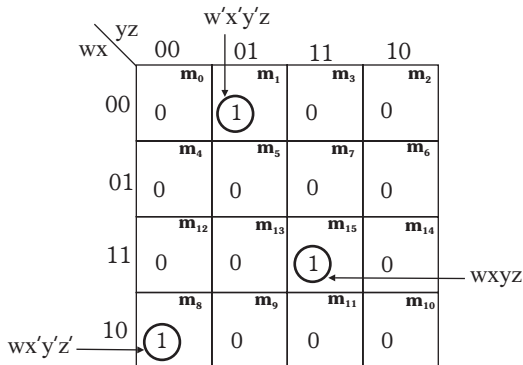


Fig. a: K-map for, $F = w'x'y'z + wxyz + wx'y'z$.

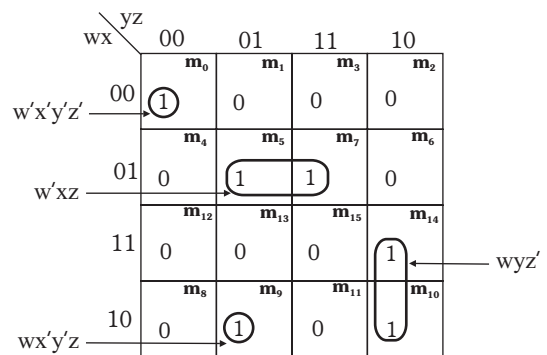


Fig. b: K-map for, $F = w'x'y'z + wx'y'z + w'xz + wyz'$.

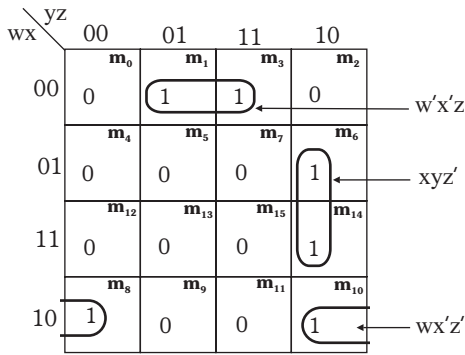


Fig. c: K-map for, $F = w'x'z + xyz' + wx'z$.

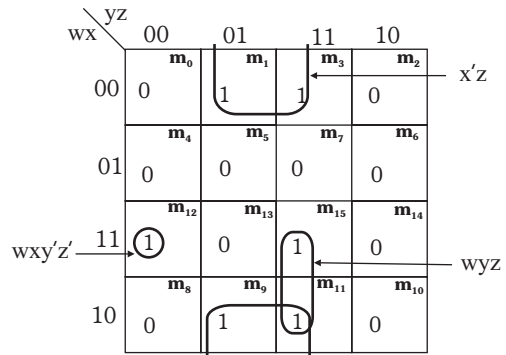


Fig. d: K-map for, $F = wxy'z' + wyz + x'z$.

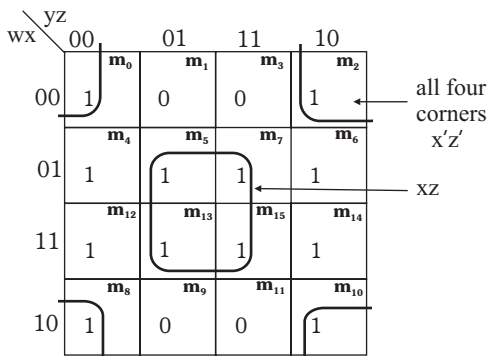


Fig. e: K-map for, $F = xz + x'z'$.

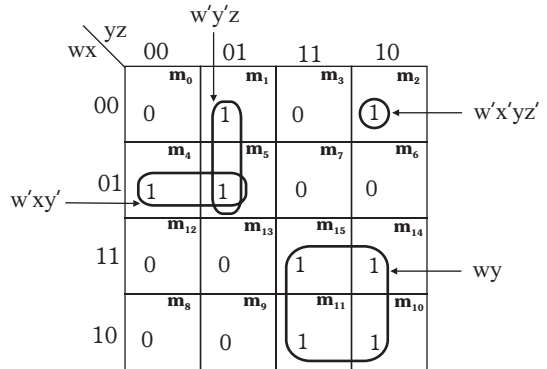


Fig. f: K-map for, $F = w'x'yz' + w'y'z + w'xy' + wy$.

Fig. 1.26: Four-variable K-maps.

Some examples of four-variable K-map with two overlapping prime implicants formed using eight adjacent 1's are shown in Fig. 1.27. Since the prime implicants shown in Fig. 1.27 are formed using eight adjacent 1's in four-variable K-map, each prime implicant represents a single literal. Since the K-map has two prime implicant the Boolean function, F is given by sum of two prime implicants of the K-map.

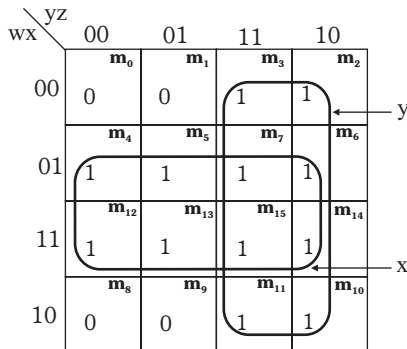


Fig. a: K-map for, $F = x + y$.

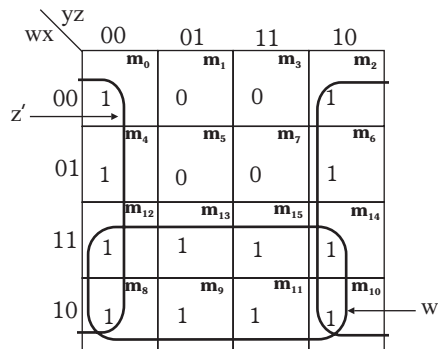


Fig. b: K-map for, $F = w + z'$.

Fig. 1.27: Four-variable K-map with overlapping prime implicants formed using eight adjacent 1's.

When all the entries of K-map (or all minterms) are 1's then the Boolean function value is 1 ($F = 1$) as shown in Fig. 1.28.

		yz			
wx		00	01	11	10
		m_0	m_1	m_3	m_2
	00	1	1	1	1
		m_4	m_5	m_7	m_6
	01	1	1	1	1
		m_{12}	m_{13}	m_{15}	m_{14}
	11	1	1	1	1
		m_8	m_9	m_{11}	m_{10}
	10	1	1	1	1

Fig. 1.28: Four-variable K-map with all minterms equal to 1 and $F = 1$.

1.6.5 Five-Variable K-map

Let A, B, C, D and E be the five-variables.

Now, $n = 5$ and $2^5 = 32$

Therefore, 32 minterms are possible as shown in Table 1.37.

Table 1.37: Truth Table

A	B	C	D	E	Minterm	A	B	C	D	E	Minterm
0	0	0	0	0	m_0	1	0	0	0	0	m_{16}
0	0	0	0	1	m_1	1	0	0	0	1	m_{17}
0	0	0	1	0	m_2	1	0	0	1	0	m_{18}
0	0	0	1	1	m_3	1	0	0	1	1	m_{19}
0	0	1	0	0	m_4	1	0	1	0	0	m_{20}
0	0	1	0	1	m_5	1	0	1	0	1	m_{21}
0	0	1	1	0	m_6	1	0	1	1	0	m_{22}
0	0	1	1	1	m_7	1	0	1	1	1	m_{23}
0	1	0	0	0	m_8	1	1	0	0	0	m_{24}
0	1	0	0	1	m_9	1	1	0	0	1	m_{25}
0	1	0	1	0	m_{10}	1	1	0	1	0	m_{26}
0	1	0	1	1	m_{11}	1	1	0	1	1	m_{27}
0	1	1	0	0	m_{12}	1	1	1	0	0	m_{28}
0	1	1	0	1	m_{13}	1	1	1	0	1	m_{29}
0	1	1	1	0	m_{14}	1	1	1	1	0	m_{30}
0	1	1	1	1	m_{15}	1	1	1	1	1	m_{31}

The 32 minterms can be arranged in 2 numbers of 4 variable K-map, one for $A = 0$ and other for $A = 1$ as shown in Figs. 1.29 and 1.30. These two K-maps can be placed one over the other as shown in Fig. 1.31. While forming prime implicants any two square that lie one over the other are considered adjacent.

Apart from this, the guidelines for forming prime implicants of 4-variable K-map can be applied individually to Figs. 1.29 and 1.30 for the adjacent squares that do not lie one over the other in Fig. 1.31. Some examples of formation of prime implicants in 5-variable K-map are shown in Figs. 1.32 and 1.33.

$A = 0$

DE \ BC	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Fig. 1.29: 4-variable K-map for $A = 0$.

$A = 1$

DE \ BC	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

Fig. 1.30: 4-variable K-map for $A = 1$.

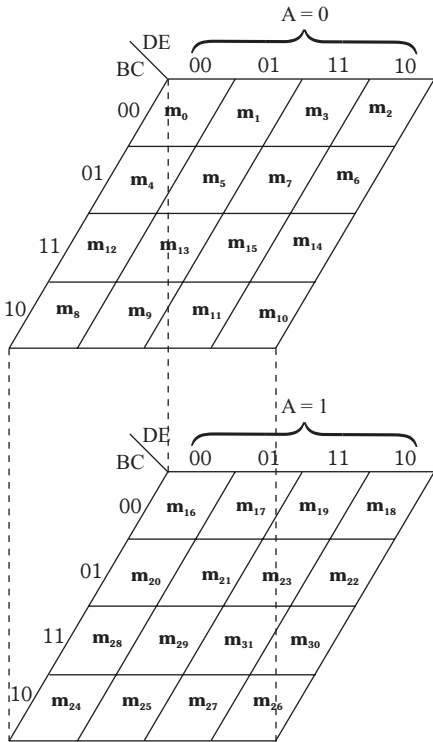


Fig. 1.31: 5-variable K-map.

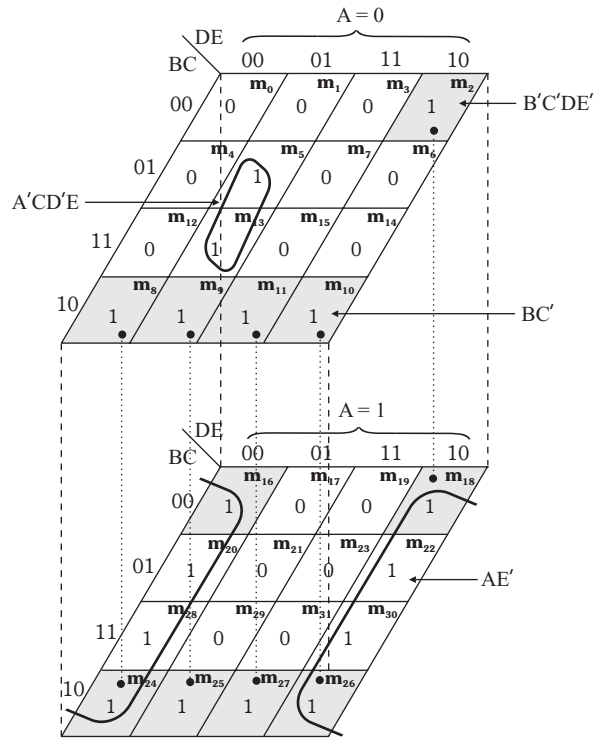


Fig. 1.32: K-map for $F = B'C'DE' + A'CD'E + AE' + BC'$.

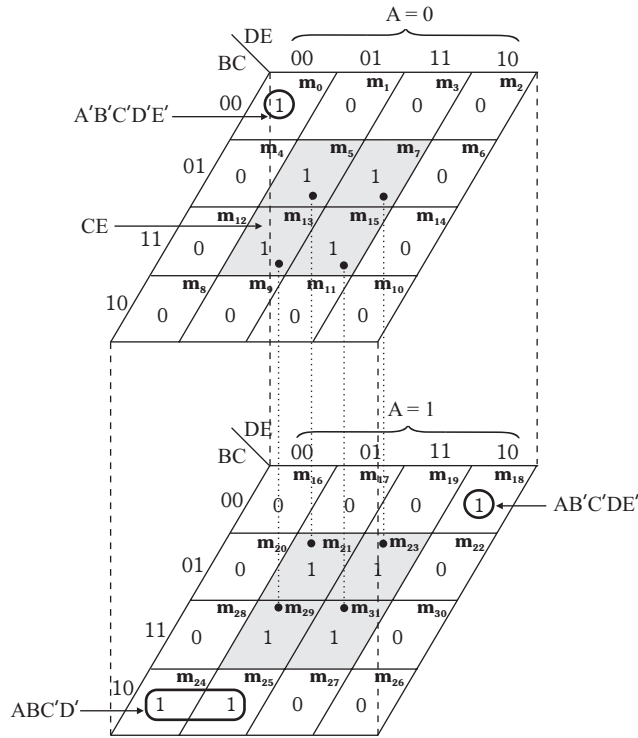


Fig. 1.33: K-map for $F = A'B'C'D'E' + CE + AB'C'DE' + ABC'D'$.

Example 1.18

Design a combinational circuit for the following specification.

The output is 1 when the binary value of the inputs for a 3 variables function is less than 3. The output is 0 otherwise.

Solution

The output is 1, when the input is 0, 1 or 2. The output will be 0 if the input is greater than or equal to 3. Let us consider 3-bit binary inputs which can take values from 0 to 7. Let x , y and z be input variables and F be function output. The truth table is shown in Table 1 and the three-variable K-map is shown in Fig. 1.

Table 1: Truth Table

Inputs			Minterm	Output
x	y	z		F
0	0	0	m_0	1
0	0	1	m_1	1
0	1	0	m_2	1
0	1	1	m_3	0
1	0	0	m_4	0
1	0	1	m_5	0
1	1	0	m_6	0
1	1	1	m_7	0

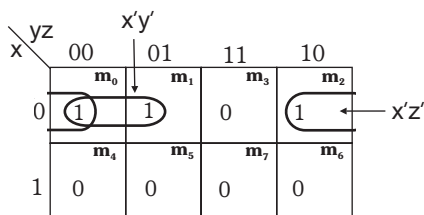


Fig. 1: K-map for F .

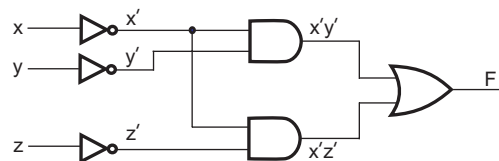


Fig. 2: Logic circuit of F .

In the K-map of Fig. 1, two overlapping prime implicants are formed using two adjacent 1's and the corresponding product terms are $x'y'$ and $x'z'$. Therefore, the Boolean function, F is sum of these two prime implicants as shown below. The combinational logic circuit is drawn using logic gates as shown in Fig. 2.

$$F = x'y' + x'z'$$

Example 1.19

Design a combinational circuit with three inputs x , y and z and the 3 outputs, A , B and C . When the binary input is 0, 1, 2 or 3 the binary output is one greater than the input. When the binary input is 4, 5, 6 or 7, the binary output is one less than the input.

Solution

The truth table of the given problem is developed as shown in Table 1. Using the truth table the K-maps are constructed as shown in Figs. 1 to 3.

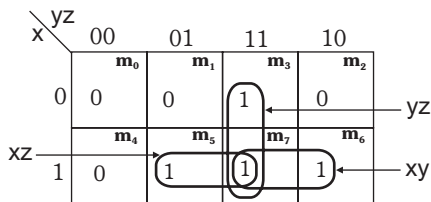


Fig. 1: K-map for A.

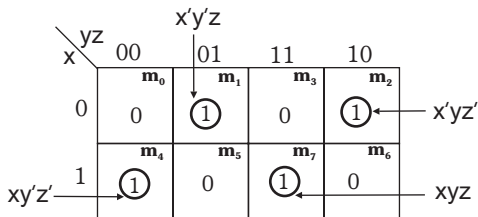


Fig. 2: K-map for B.

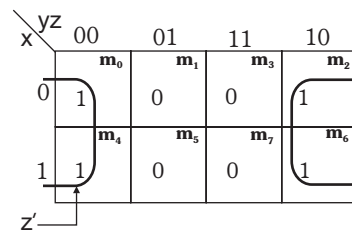


Fig. 3: K-map for C.

From the K-map for A, B and C the following Boolean equations are obtained and using these equations the logic circuit is drawn in Fig. 4.

$$A = xz + xy + yz$$

$$B = xy'z' + xyz + x'y'z + x'yz'$$

$$C = z'$$

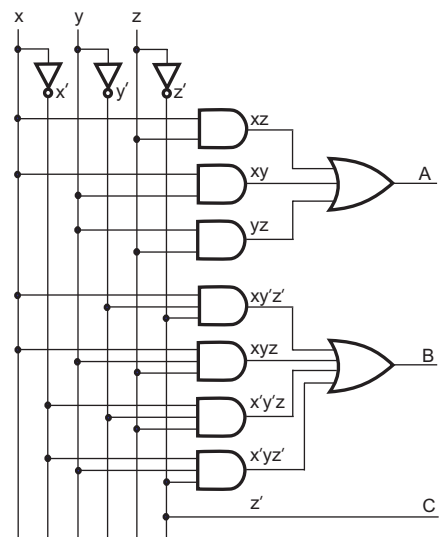


Fig. 4: Logic circuit of A, B and C.

Example 1.20

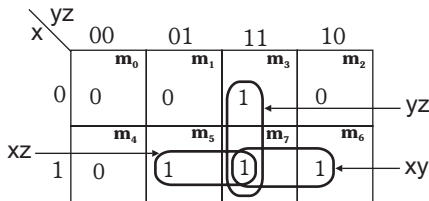
Design a 3-input majority circuit, in which the output is equal to 1 if the input variables have more 1's than 0's, otherwise, output is 0.

Solution

The truth table of majority circuit described in the problem is shown in Table 1 and the three variable K-map constructed using Table 1 is shown in Fig. 1.

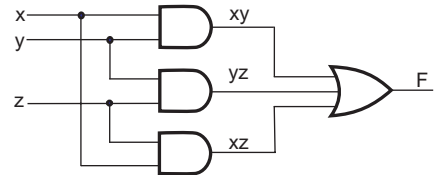
Table 1: Truth Table

Inputs			Minterm	Output
x	y	z		
0	0	0	m_0	0
0	0	1	m_1	0
0	1	0	m_2	0
0	1	1	m_3	1
1	0	0	m_4	0
1	0	1	m_5	1
1	1	0	m_6	1
1	1	1	m_7	1

**Fig. 1: K-map for F.**

In the K-map of Fig. 1, three overlapping prime implicants are formed using two adjacent 1's and the corresponding product terms are xy , yz and xz . Therefore, the Boolean function, F is sum of these three prime implicants as shown below. The combinational logic circuit is drawn using logic gates as shown in Fig. 2.

$$F = xy + yz + xz$$

**Fig. 2: Logic circuit of F.****Example 1.21**

Assume a 3-input AND gate with output F and a 3-input OR gate with output G . Show the signals of the outputs F and G as functions of the three inputs A , B and C . Use all 8 possible combinations of inputs ABC .

Solution

It's given that,

$$F = ABC \quad ; \quad G = A + B + C$$

The truth table for F and G for all possible combinations of A , B and C is developed as shown in Table 1.

Using the truth table the K-map for F and G are constructed as shown in Figs. 1 and 2.

Table 1: Truth Table for 3-input AND and OR gate

Inputs			Minterm	Outputs	
A	B	C		F	G
0	0	0	m_0	0	0
0	0	1	m_1	0	1
0	1	0	m_2	0	1
0	1	1	m_3	0	1
1	0	0	m_4	0	1
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

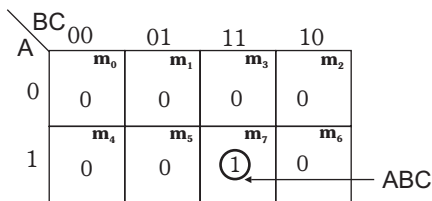


Fig. 1: K-map for F.

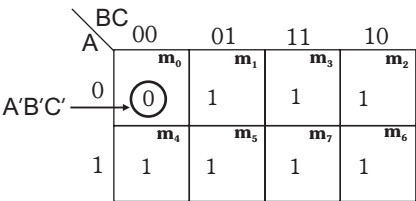


Fig. 2: K-map for G.

From the K-map we get the following Boolean equations.

$$F = ABC$$

$$G = (A'B'C')' = A + B + C$$

Using DeMorgan's law

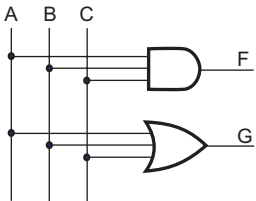


Fig. 3: Logic circuit of F and G.

Example 1.22

Simplify and implement the logic function $F(A, B, C) = \sum m(0, 1, 4, 5, 7)$ using logic gates.

Solution

The truth table of logic function is shown in Table 1 and the three variable K-map is constructed using Table 1 as shown in Fig. 1. The function output, F is 1 for minterms m_0, m_1, m_4, m_5 and m_7 .

Table 1: Truth Table

Inputs			Minterm	Output
A	B	C		F
0	0	0	m_0	1
0	0	1	m_1	1
0	1	0	m_2	0
0	1	1	m_3	0
1	0	0	m_4	1
1	0	1	m_5	1
1	1	0	m_6	0
1	1	1	m_7	1

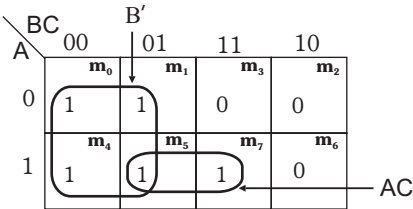


Fig. 1: K-map for F.

From the K-map we get the Boolean equation of the function as,

$$F = AC + B'$$

The logic circuit is drawn using the above Boolean function as shown in Fig. 2.

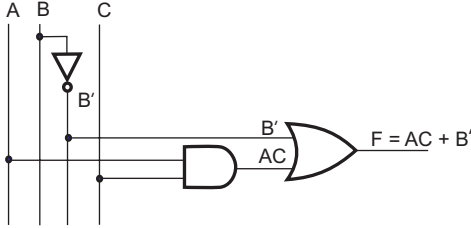


Fig. 2: Logic circuit of F.

Example 1.23

Simplify the following Boolean functions, using three variable K-maps:

a) $F(x, y, z) = \sum m(0, 2, 4, 5)$

b) $F(x, y, z) = \sum m(0, 2, 4, 5, 6)$

c) $F(x, y, z) = \sum m(0, 1, 2, 3, 5)$

Solution

a) $F(x, y, z) = \sum m(0, 2, 4, 5)$

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Fig. 1. The function is defined as sum of minterms m_0, m_2, m_4 and m_5 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, two prime implicants each with two adjacent 1's can be formed and the corresponding product terms are xy' and $x'z'$. The simplified Boolean function is given by sum of these two product terms.

$$\therefore F = xy' + x'z'$$

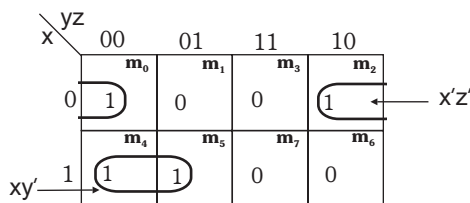


Fig. 1.

b) $F(x, y, z) = \sum m(0, 2, 4, 5, 6)$

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Fig. 2. The function is defined as sum of minterms m_0, m_2, m_4, m_5 and m_6 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 2, two prime implicants one with four adjacent 1's and another with two adjacent 1's can be formed and the corresponding product terms are z' and xy' . The simplified Boolean function is given by sum of these two product terms.

$$\therefore F = z' + xy'$$

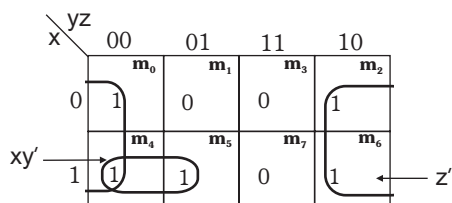


Fig. 2.

c) $F(x, y, z) = \sum m(0, 1, 2, 3, 5)$

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Fig. 3. The function is defined as sum of minterms m_0, m_1, m_2, m_3 and m_5 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 3, two prime implicants one with four adjacent 1's and another with two adjacent 1's can be formed and the corresponding product terms are x' and $y'z$. The simplified Boolean function is given by sum of these two product terms.

$$\therefore F = x' + y'z$$

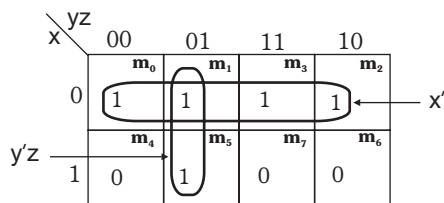


Fig. 3.

Example 1.24

Simplify the following Boolean function, using three-variable K-maps:

$$F = xy + x'y'z' + x'yz'$$

Solution

The truth table of the given function is formed as shown in Table 1 and the minterms for which function output is "1" are determined and using these minterms the function F_1 can be expressed as,

$$F_1 = \sum (m_0, m_2, m_6, m_7)$$

Table 1: Truth Table for Functions F_1

Input Variables			Minterm	Product Terms			Function Output
x	y	z		xy	x'y'z'	x'yz'	
0	0	0	m_0	0	1	0	1
0	0	1	m_1	0	0	0	0
0	1	0	m_2	0	0	1	1
0	1	1	m_3	0	0	0	0
1	0	0	m_4	0	0	0	0
1	0	1	m_5	0	0	0	0
1	1	0	m_6	1	0	0	1
1	1	1	m_7	1	0	0	1

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Fig. 1. The function is defined as sum of minterms m_0, m_2, m_6 and m_7 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, two prime implicants each with two adjacent 1's can be formed and the corresponding product terms are xy and $x'z'$. The simplified Boolean function is given by sum of these two product terms.

$$\therefore F_1 = xy + x'z'$$

x \ yz	yz			
	00	01	11	10
0	m_0 1	m_1 0	m_3 0	m_2 1 ← $x'z'$
1	m_4 0	m_5 0	m_7 1	m_6 1 ← xy

Fig. 1: K-map for function F_1 .**Example 1.25**

(AU, Nov/Dec'22, 10 Marks)

Simplify the following Boolean functions, using Karnaugh maps.

a) $F_1(A, B, C, D) = \sum m(3, 7, 11, 13, 14, 15)$

b) $F_2(w, x, y, z) = \sum m(2, 3, 12, 13, 14, 15)$

Solution

a) $F_1(A, B, C, D) = \sum m(3, 7, 11, 13, 14, 15)$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms $m_3, m_7, m_{11}, m_{13}, m_{14}$ and m_{15} and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, three prime implicants one with four adjacent 1's and the other two with two adjacent 1's can be formed and the corresponding product terms are CD, ABC and ABD. The simplified Boolean function is given by sum of these three product terms.

$$\therefore F_1 = CD + ABC + ABD$$

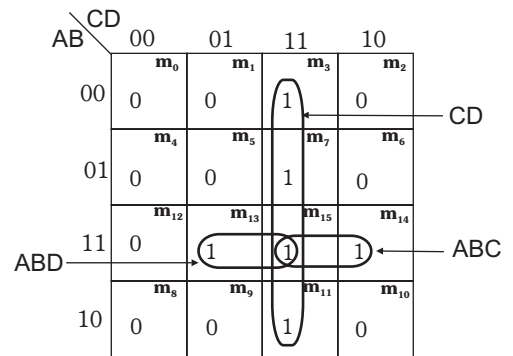


Fig. 1: K-map for F_1 .

$$b) F_2(w, x, y, z) = \sum m(2, 3, 12, 13, 14, 15)$$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 2. The function is defined as sum of minterms $m_2, m_3, m_{12}, m_{13}, m_{14}$ and m_{15} and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 2, two prime implicants one with four adjacent 1's and the other with two adjacent 1's can be formed and the corresponding product terms are wx and w'x'y. The simplified Boolean function is given by sum of these two product terms.

$$\therefore F_2 = wx + w'x'y$$

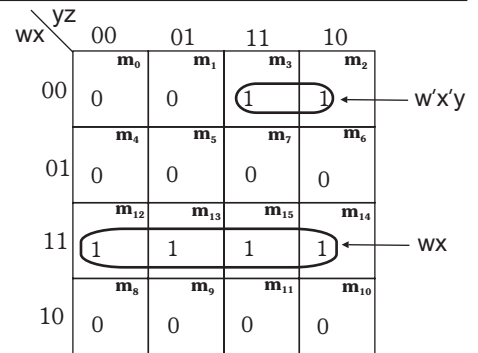


Fig. 2: K-map for F_2 .

Example 1.26

Minimize $F(A, B, C, D) = \sum m(1, 3, 4, 6, 8, 11, 15) + d(0, 5, 7)$ using K-map and draw MSI circuit for the output.

Solution

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms $m_1, m_3, m_4, m_6, m_8, m_{11}$ and m_{15} and so a "1" is filled in the corresponding squares in K-map the minterms m_0, m_5, m_7 and so a "x" is filled in corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, three prime implicants one with four adjacent 1's and the other two with two adjacent 1's can be formed and the corresponding product terms are CD, A'B and A'D and B'C'D'. The simplified Boolean function is given by sum of these four product terms.

$$\therefore F = CD + A'B + B'C'D' + A'D$$

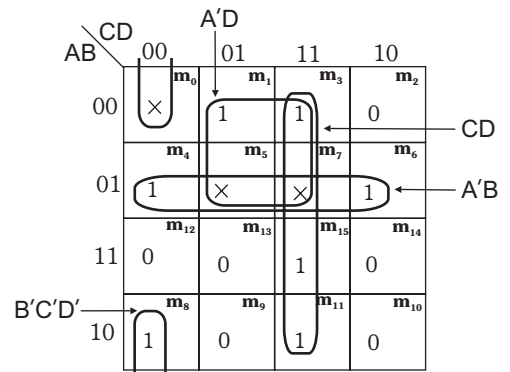
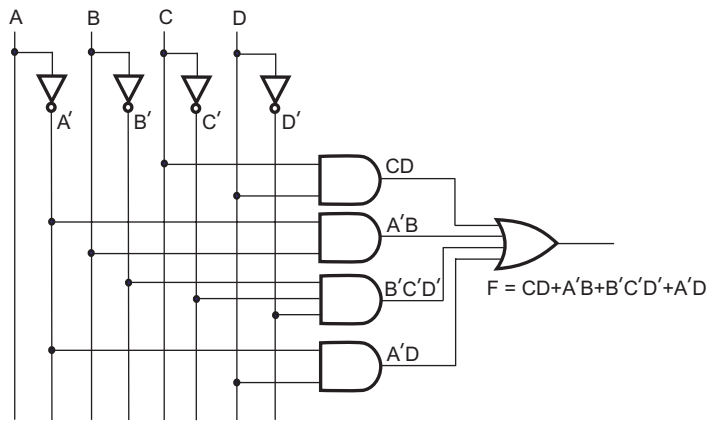


Fig. 1: K-map for F .

The MSI circuit is drawn using the above Boolean function as shown in Fig. 2.

Fig. 2: MSI circuit of F .**Example 1.27**

Simplify the following Boolean functions using K-maps and draw MSI circuits.

$$F = w'z + xz + x'y + wx'z$$

Solution

a) $F_1 = w'z + xz + x'y + wx'z$

The truth table of the given function is formed as shown in Table 1 and the minterms for which function output is "1" are determined and using these minterms the function F_1 can be expressed as,

$$F_1 = \sum (m_1, m_2, m_3, m_5, m_7, m_9, m_{10}, m_{11}, m_{13}, m_{15})$$

Table 1: Truth Table for Function F_1

Input Variables				Minterm	Complement of Inputs		Product Terms				Function Output
w	x	y	z		w'	x'	w'z	xz	x'y	wx'z	
0	0	0	0	m_0	1	1	0	0	0	0	0
0	0	0	1	m_1	1	1	1	0	0	0	1
0	0	1	0	m_2	1	1	0	0	1	0	1
0	0	1	1	m_3	1	1	1	0	1	0	1
0	1	0	0	m_4	1	0	0	0	0	0	0
0	1	0	1	m_5	1	0	1	1	0	0	1
0	1	1	0	m_6	1	0	0	0	0	0	0
0	1	1	1	m_7	1	0	1	1	0	0	1
1	0	0	0	m_8	0	1	0	0	0	0	0
1	0	0	1	m_9	0	1	0	0	0	1	1
1	0	1	0	m_{10}	0	1	0	0	1	0	1
1	0	1	1	m_{11}	0	1	0	0	1	1	1
1	1	0	0	m_{12}	0	0	0	0	0	0	0
1	1	0	1	m_{13}	0	0	0	1	0	0	1
1	1	1	0	m_{14}	0	0	0	0	0	0	0
1	1	1	1	m_{15}	0	0	0	1	0	0	1

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms $m_1, m_2, m_3, m_5, m_7, m_9, m_{10}, m_{11}, m_{13}$ and m_{15} and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, two prime implicants one with eight adjacent 1's and the other with four adjacent 1's can be formed and the corresponding product terms are z and $x'y$. The simplified Boolean function is given by sum of these two product terms.

$$\therefore F_1 = z + x'y$$

The MSI circuit is drawn using the above Boolean function as shown in Fig. 2.

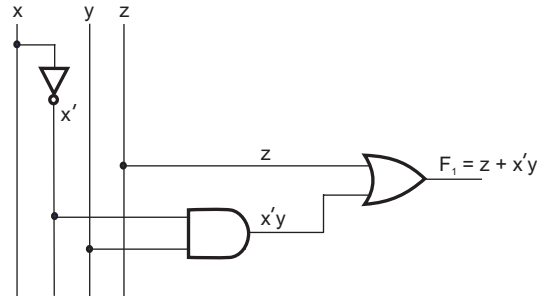


Fig. 2: MSI circuit of F_1 .

yz \ wx	00	01	11	10
00	m_0 0	m_1 1	m_3 1	m_2 1
01	m_4 0	m_5 1	m_7 1	m_6 0
11	m_{12} 0	m_{13} 1	m_{15} 1	m_{14} 0
10	m_8 0	m_9 1	m_{11} 1	m_{10} 1

Fig. 1: K-map for function F_1 .

1.6.6 Realization of Product-of-Sums Form using K-map

In truth table the function output value with "0" represent maxterms. Therefore in K-map if squares with 0's are considered for forming prime implicants and the complement of the resultant sum-of-products form will give product-of-sums form.

Two-Variable K-map

An example of two-variable K-map with two prime implicants formed using single 0's are shown in Fig. 1.34. Here, each prime implicant represents a product term of two literals. Now, the POS form of Boolean function is given by complement of sum of these two product terms as shown below:

$$F = (x'y + xy')' = (x'y)'(xy')'$$

$$= (x + y')(x' + y)$$

Using DeMorgan's Theorem

y \ x	0	1
0	m_0 1	m_1 0
1	m_2 0	m_3 1

Fig. 1.34: K-map for $F = (x'y + xy')'$.

Three-Variable K-map

An example of three-variable K-map with two prime implicants formed using single 0's are shown in Fig. 1.35. Here, each prime implicant represents a product term of three literals. Now, the POS form of Boolean function is given by complement of sum of these two product terms as shown below:

$$F = (x'yz + xy'z')'$$

Using DeMorgan's Theorem

$$= (x'yz)'(xy'z')' = (x + y' + z')(x' + y + z)$$

yz \ x	00	01	11	10
0	m_0 1	m_1 1	m_3 0	m_2 1
1	m_4 0	m_5 1	m_7 1	m_6 1

Fig. 1.35: K-map for $F = (x'yz + xy'z')'$.

An example of three-variable K-map with two prime implicants formed using two adjacent 0's are shown in Fig. 1.36. Here, each prime implicant represents a product term of two literals. Now, the POS form of Boolean function is given by complement of sum of these two product terms as shown below:

$$\begin{aligned} F &= (y z' + y' z)' \\ &= (y z')' (y' z)' \quad \text{Using DeMorgan's Theorem} \\ &= (y' + z) (y + z') \end{aligned}$$

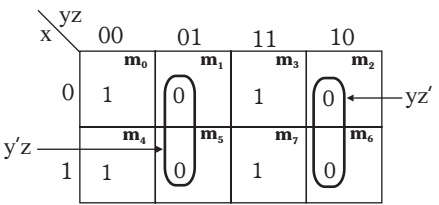


Fig. 1.36: K-map for $F = (yz' + y'z)'$.

Four-Variable K-map

An example of four-variable K-map with three prime implicants one formed using four adjacent 0's and the other two formed using two adjacent 0's are shown in Fig. 1.37. Here, the prime implicant with four adjacent 0's represent a product term of two literals and each prime implicant with two adjacent 0's represent a product term of three literals. Now, the POS form of Boolean function is given by complement of sum of these three product terms as shown below:

$$\begin{aligned} F &= (w z + x' y' z' + x y z')' \\ &= (w z)' (x' y' z')' (x y z')' \\ &= (w' + z') (x + y + z) (x' + y' + z) \end{aligned}$$

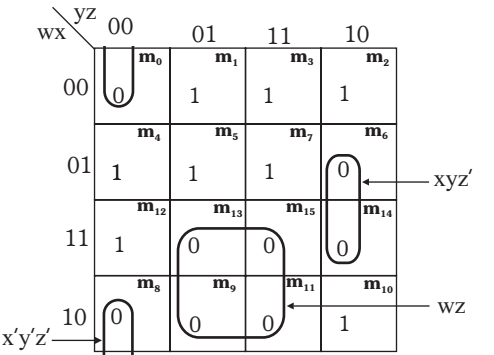


Fig. 1.37: K-map for $F = (wz + x'y'z' + xyz')'$.

Using DeMorgan's Theorem

1.6.7 Completely and Incompletely Specified Functions

(K-map with Don't-Care Conditions)

In n-variable Boolean function there are 2^n possible combinations of inputs and so there will be 2^n outputs. But in certain functions all the 2^n possible outputs are not defined (or not used).

When the outputs are defined for all the possible 2^n combinations of inputs then the function is called **completely specified function**. When the outputs are not defined for some of the possible combinations of inputs then the function is called **incompletely specified function**.

Consider the truth table shown in Table 1.38 and the corresponding K-map shown in Fig. 1.38. In Table 1.38 the undefined function outputs are called **don't-care conditions** and denoted by \times .

Table 1.38: Truth Table with Don't-cares

Inputs x y z	Minterm	Output F
0 0 0	m_0	0
0 0 1	m_1	1
0 1 0	m_2	\times
0 1 1	m_3	\times
1 0 0	m_4	0
1 0 1	m_5	1
1 1 0	m_6	1
1 1 1	m_7	0

In simplification of Boolean functions using K-maps, don't-care conditions can be considered as either 0 or 1. It is not necessary that don't-care conditions to be included in prime implicants, but if inclusion of a don't-care condition leads to minimization of a literal then, don't-care can be included in the prime implicants as shown in Fig. 1.38.

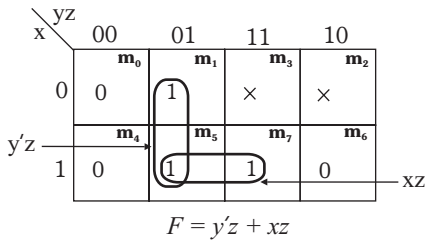


Fig. a: Formation of prime implicants without including don't-care condition.

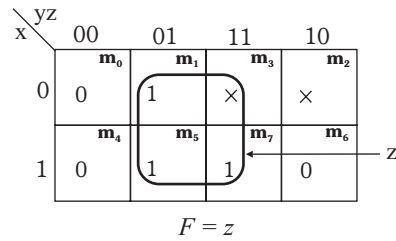


Fig. b: Formation of prime implicants including don't-care condition.

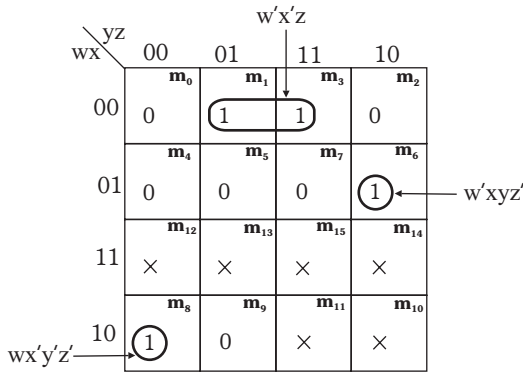
Fig. 1.38: K-map with don't-care conditions.

Another example of truth table with don't-cares is shown in Table 1.39 which is truth table of binary to 5421 BCD code conversion. K-maps for each output a_1 , a_2 , a_4 and a_5 are shown in Figs. 1.39 to 1.42. For each output two K-maps are drawn in order to show that the literals are reduced in Boolean expression of the outputs when don't-cares are included in formation of prime implicants.

Table 1.39: Truth Table of BCD

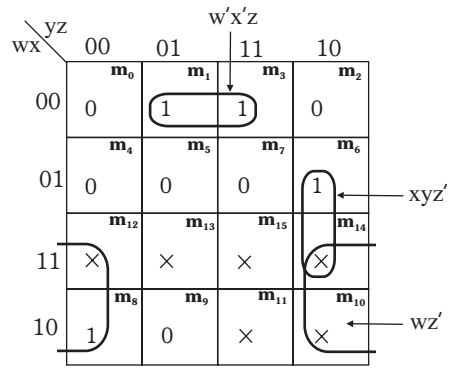
Decimal	Minterm	Input Binary w x y z	Output 5421 BCD $a_5 a_4 a_2 a_1$
0	m_0	0 0 0 0	0 0 0 0
1	m_1	0 0 0 1	0 0 0 1
2	m_2	0 0 1 0	0 0 1 0
3	m_3	0 0 1 1	0 0 1 1
4	m_4	0 1 0 0	0 1 0 0
5	m_5	0 1 0 1	1 0 0 0
6	m_6	0 1 1 0	1 0 0 1
7	m_7	0 1 1 1	1 0 1 0
8	m_8	1 0 0 0	1 0 1 1
9	m_9	1 0 0 1	1 1 0 0
-	m_{10}	1 0 1 0	x x x x
-	m_{11}	1 0 1 1	x x x x
-	m_{12}	1 1 0 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

6 don't-care conditions



$$a_1 = wx'y'z' + w'xyz' + w'x'z$$

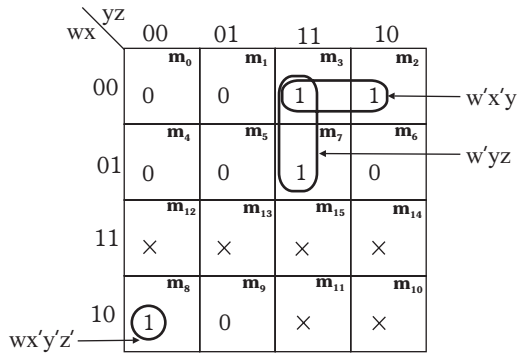
Fig. a: Prime implicants formed without including don't-cares.



$$a_1 = wz' + xyz' + w'x'z$$

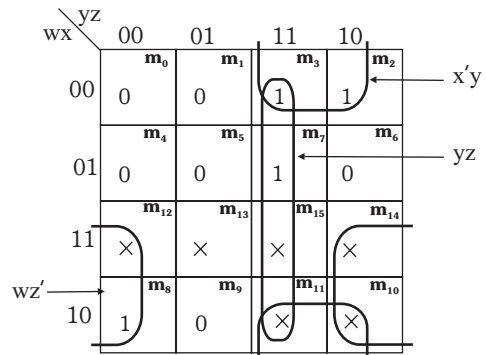
Fig. b: Prime implicants formed by including don't-cares.

Fig. 1.39: K-map for a_1 .



$$a_2 = wx'y'z' + w'yz + w'x'y$$

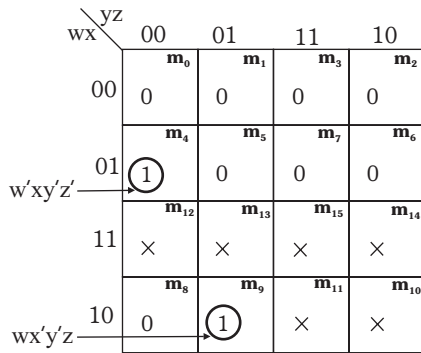
Fig. a: Prime implicants formed without including don't-cares.



$$a_2 = yz + x'y + wz'$$

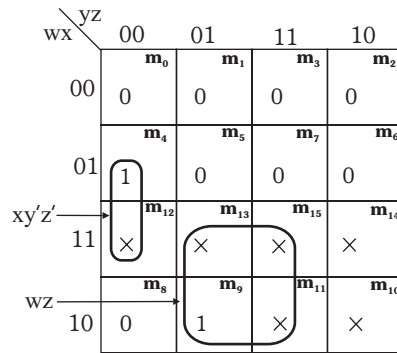
Fig. b: Prime implicants formed by including don't-cares.

Fig. 1.40: K-map for a_2 .



$$a_4 = w'xy'z' + wx'y'z'$$

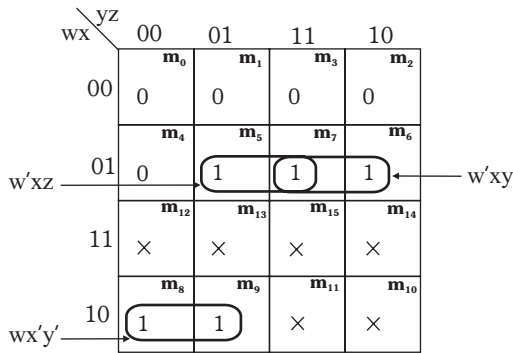
Fig. a: Prime implicants formed without including don't-cares.



$$a_4 = xy'z' + wz'$$

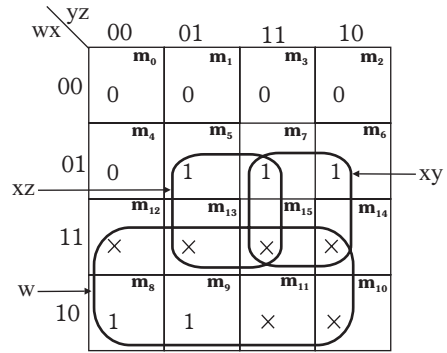
Fig. b: Prime implicants formed by including don't-cares.

Fig. 1.41: K-map for a_4 .



$$a_5 = wx'y' + w'xz + w'xy$$

Fig. a: Prime implicants formed without including don't-cares.



$$a_5 = w + xz + xy$$

Fig. b: Prime implicants formed by including don't-cares.

Fig. 1.42: K-map for a_5 .

Boolean function for a_1 , a_2 , a_4 and a_5 when don't-cares are not considered for formation of prime implicants

$$a_1 = wx'y'z' + w'xyz' + w'x'z$$

$$a_2 = wx'y'z' + w'yz + w'x'y$$

$$a_4 = w'xy'z' + wx'y'z$$

$$a_5 = wx'y' + w'xz + w'xy$$

Boolean function for a_1 , a_2 , a_4 and a_5 when don't-cares are included for formation of prime implicants

$$a_1 = wz' + xyz' + w'x'z$$

$$a_2 = yz + x'y + wz'$$

$$a_4 = xy'z' + wz$$

$$a_5 = w + xz + xy$$

Example 1.28

Simplify the following Boolean functions.

a) $F_1(A, B, C, D) = \prod M(1, 3, 5, 7, 13, 15)$

b) $F_2(A, B, C, D) = \prod M(1, 3, 6, 9, 11, 12, 14)$

Solution

a) $F_1(A, B, C, D) = \prod M(1, 3, 5, 7, 13, 15)$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as product of maxterms $M_1, M_3, M_5, M_7, M_{13}$ and M_{15} and so a "0" is filled in the corresponding squares in K-map and the remaining squares are filled with "1".

In the K-map of Fig. 1, two prime implicants each with four adjacent 0's can be formed and the corresponding product terms are $A'D$ and BD . The simplified Boolean function in POS form is given by complement of sum of these two product terms.

$$F_1 = (A'D + BD)'$$

$$= (A'D)'(BD)'$$

Using DeMorgan's Theorem

$$= (A + D')(B' + D')$$

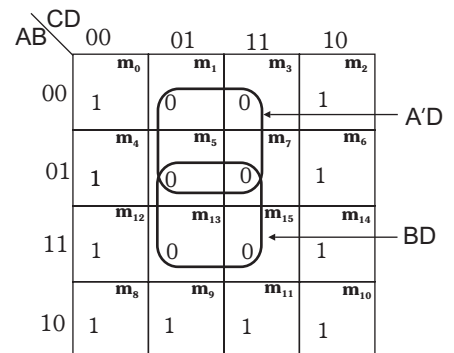


Fig. 1: K-map for F_1 .

b) $F_2(A, B, C, D) = \prod M(1, 3, 6, 9, 11, 12, 14)$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 2. The function is defined as product of maxterms $M_1, M_3, M_6, M_9, M_{11}, M_{12}$ and M_{14} and so a "0" is filled in the corresponding squares in K-map and the remaining squares are filled with "1".

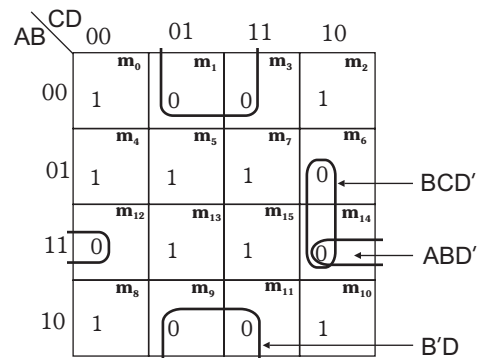
In the K-map of Fig. 2, three prime implicants one with four adjacent 0's and the other two each with two adjacent 0's can be formed and the corresponding product terms are $B'D$, BCD' and ABD' . The simplified Boolean function in POS form is given by complement of sum of these three product terms.

$$F_2 = (B'D + BCD' + ABD')'$$

$$= (B'D)' (BCD')' (ABD')'$$

Using DeMorgan's Theorem

$$= (B + D') (B' + C' + D) (A' + B' + D)$$

Fig. 2: K-map for F_2 .**Example 1.29**

(AU, Nov/Dec'23, 15 Marks)

Simplify the following function to sum-of-products and product-of-sums.

$$F = x'z' + y'z' + yz' + xy$$

Solution

The truth table of the given function is formed as shown in Table 1 and the minterms for which function output is "1" are determined and using these minterms the function F can be expressed as,

$$F = \sum (m_0, m_2, m_4, m_6, m_7)$$

Table 1: Truth Table

Input Variables			Minterm	Complement of Inputs			Product Terms				Function Output
x	y	z		x'	y'	z'	x'z'	y'z'	yz'	xy	F
0	0	0	m_0	1	1	1	1	1	0	0	1
0	0	1	m_1	1	1	0	0	0	0	0	0
0	1	0	m_2	1	0	1	1	0	1	0	1
0	1	1	m_3	1	0	0	0	0	0	0	0
1	0	0	m_4	0	1	1	0	1	0	0	1
1	0	1	m_5	0	1	0	0	0	0	0	0
1	1	0	m_6	0	0	1	0	0	1	1	1
1	1	1	m_7	0	0	0	0	0	0	1	1

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Fig. 1. The function is defined as sum of minterms m_0, m_2, m_4, m_6 and m_7 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

For SOP form of Boolean function, the prime implicants are formed by considering the squares filled with 1's. In the K-map of Fig. 1, two prime implicants one with four adjacent 1's and the other with two adjacent 1's can be formed and the corresponding product terms are z' and xy . The simplified Boolean function in SOP form is given by sum of these two product terms.

For POS form of Boolean function, the prime implicants are formed by considering the squares filled with 0's. In the K-map of Fig. 2, two prime implicants each with two adjacent 0's can be formed and the corresponding product terms are $x'z$ and $y'z$. The simplified Boolean function in POS form is given by complement of sum of these two product terms.

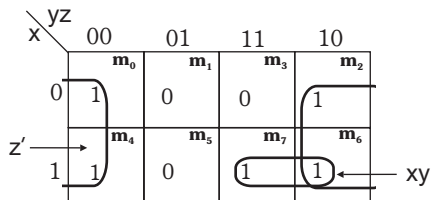


Fig. 1: K-map for SOP (Sum-of-Products).

$$F = z' + xy$$

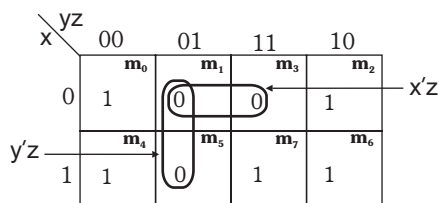


Fig. 2: K-map for POS (Product-of-Sums).

$$F = (x'z + y'z)'$$

$$= (x'z)'(y'z)'$$

$$= (x + z')(y + z')$$

Using DeMorgan's Theorem

Example 1.30

Simplify the following Boolean functions, together with the don't-care conditions and then express the simplified function in sum-of-products.

a) $F_1(A, B, C, D) = \sum m(0, 6, 8, 13, 14)$

b) $F_2(A, B, C, D) = \sum m(5, 6, 7, 12, 14, 15)$

$d_1(A, B, C, D) = \sum d(2, 4, 10)$

$d_2(A, B, C, D) = \sum d(1, 3, 9, 11)$

Solution

a) $F_1(A, B, C, D) = \sum m(0, 6, 8, 13, 14)$

$d_2(A, B, C, D) = \sum d(2, 4, 10)$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms m_0, m_6, m_8, m_{13} and m_{14} and so a "1" is filled in the corresponding squares in K-map. The function also has don't-care outputs for the minterms m_2, m_4 and m_{10} and so a "x" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 1, by considering the don't-care condition in m_2 and m_{10} as "1", three prime implicants can be formed in which two has four adjacent 1's and one have single 1 and the corresponding product terms are $B'D'$, CD' and $ABC'D$. The simplified Boolean function is given by sum of these three product terms.

$$\therefore F_1 = B'D' + CD' + ABC'D$$

Note: Don't-care in m_2 and m_{10} are considered as 1.

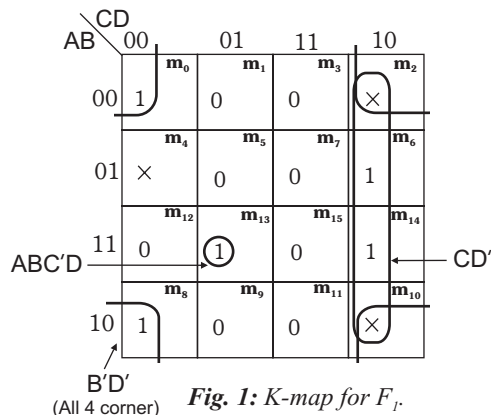


Fig. 1: K-map for F_1 .

$$b) F_2(A, B, C, D) = \sum m(5, 6, 7, 12, 14, 15)$$

$$d_2(A, B, C, D) = \sum d(1, 3, 9, 11)$$

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 2. The function is defined as sum of minterms $m_5, m_6, m_7, m_{12}, m_{14}$ and m_{15} and so a "1" is filled in the corresponding squares in K-map. The function also has don't-care outputs for the minterms m_1, m_3, m_9 and m_{11} and so a "x" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

In the K-map of Fig. 2, by considering the don't-care condition in m_1 and m_3 as "1", three prime implicants can be formed in which each of the two prime implicants has four adjacent 1's and one prime implicant has two adjacent 1's and the corresponding product terms are $A'D$, BC and ABD' . The simplified Boolean function is given by sum of these three product terms.

$$\therefore F_2 = A'D + BC + ABD'$$

Note: Don't-care in m_1 and m_3 are considered as 1.

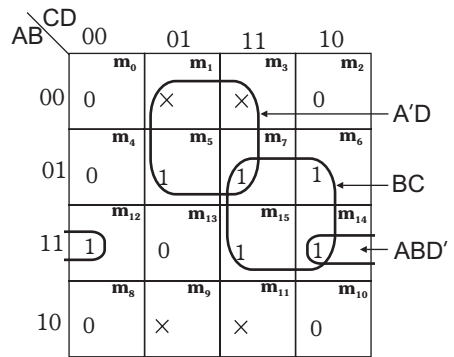


Fig. 2: K-map for F_2 .

Example 1.31

Simplify the following Boolean function with don't-care in POS form.

$$F(A, B, C, D) = \sum m(0, 2, 6, 12, 13, 14)$$

$$d(A, B, C, D) = \sum d(1, 4, 10)$$

Solution

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms $m_0, m_2, m_6, m_{12}, m_{13}$ and m_{14} and so a "1" is filled in the corresponding squares in K-map. The function also has don't-care outputs for the minterms m_1, m_4 and m_{10} and so a "x" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

For POS form of Boolean function, the prime implicants are formed by considering 0's. In the K-map of Fig. 1, by considering the don't-care condition in m_1 and m_{10} as "0", three prime implicants can be formed in which all three has four adjacent 0's and the corresponding product terms are $A'D$, AB' and CD . The simplified Boolean function in POS form is given by complement of sum of these three product terms.

$$F = (A'D + AB' + CD)'$$

$$= (A'D)' (AB')' (CD)'$$

$$= (A + D') (A' + B) (C' + D')$$

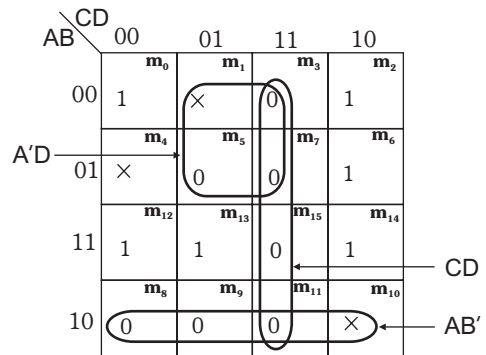


Fig. 1: K-map for F .

Using DeMorgan's Theorem

Example 1.32

A combinational circuit has 3 inputs A, B, C and output F. F is true for the following input combinations.

A is false, B is true.

A is false, C is true.

A and B and C are false.

A and B and C are true.

- Write the truth table for F. Use the convention True = 1 and false = 0.
- Write the simplified expression for F in SOP form.
- Write the simplified expression for F in POS form.

Solution**a) Truth Table for F**

A is false, B is true $\longrightarrow A'B$

A is false, C is true $\longrightarrow A'C$

A and B and C are false $\longrightarrow A'B'C'$

A and B and C are true $\longrightarrow ABC$

The Boolean equation for F is given by sum of all the above product terms.

$$\begin{aligned}
 F &= A'B + A'C + A'B'C' + ABC \\
 &= A'B(C + C') + A'C(B + B') + A'B'C' + ABC \\
 &= A'BC + A'BC' + A'B\bar{C} + A'B'C + A'B'C' + ABC \\
 &= A'BC + A'BC' + A'B'C + A'B'C' + ABC \\
 &\quad (m_3) \quad (m_2) \quad (m_1) \quad (m_0) \quad (m_7) \\
 &= \sum (m_0, m_1, m_2, m_3, m_7)
 \end{aligned}$$

$$x + x' = 1$$

$$x \cdot 1 = x$$

Table 1: Truth Table of Function F

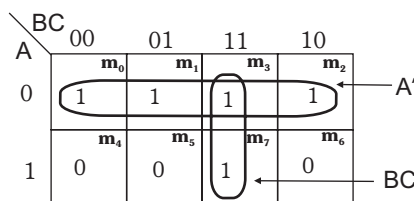
Input Variables			Minterm	Function Output
A	B	C		
0	0	0	m_0	1
0	0	1	m_1	1
0	1	0	m_2	1
0	1	1	m_3	1
1	0	0	m_4	0
1	0	1	m_5	0
1	1	0	m_6	0
1	1	1	m_7	1

The truth table for the above function is shown in Table 1. The function output is 1 for the minterms m_0, m_1, m_2, m_3 and m_7 and 0 for minterms m_4, m_5 and m_6 .

The given function has three variables and so a 3-variable K-map with 8 squares ($2^3 = 8$) is drawn as shown in Figs. 1 and 2. The function is defined as sum of minterms m_0, m_1, m_2, m_3 and m_7 and so a "1" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

b) SOP form

For SOP form of Boolean function, the prime implicants are formed by considering the squares filled with 1's. In the K-map of Fig. 1, one prime implicant with four adjacent 1's and another with two adjacent 1's can be formed and the corresponding product terms are A' and BC . The simplified Boolean function in SOP form is given by sum of these two product terms.

**Fig. 1: K-map for SOP (Sum-of-Products).**

From the K-map the simplified Boolean equation for F in SOP form is given by,

$$F = A' + BC$$

C) POS form

For POS form of Boolean function, the prime implicants are formed by considering the squares filled with 0's. In the K-map of Fig. 2, two prime implicants each with two adjacent 0's can be formed and the corresponding product terms are AC' and AB' . The simplified Boolean function in POS form is given by complement of sum of these two product terms.

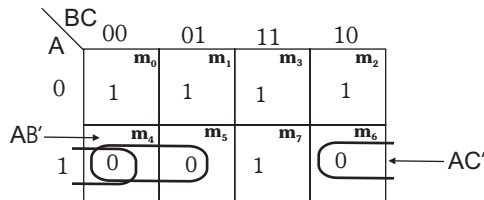


Fig. 2: K-map for POS (Product-of-Sum).

From the K-map the simplified Boolean equation for F in POS form is given by,

$$\begin{aligned} F &= (AC' + AB')' \\ &= (AC')'(AB')' \\ &= (A' + C)(A' + B) \end{aligned}$$

Example 1.33

Use Karnaugh Map method to simplify the following Boolean function

$$F(A, B, C, D) = \sum m(2, 4, 6, 10, 12) + \sum d(0, 8, 9, 13)$$

- Implement the Boolean function, F using only NAND gates.
- Implement the Boolean function, F using only NOR gates.

Solution

a) To Implement the Boolean function, F using only NAND gates

The given function has four variables and so a 4-variable K-map with 16 squares ($2^4 = 16$) is drawn as shown in Fig. 1. The function is defined as sum of minterms m_2, m_4, m_6, m_{10} and m_{12} and so a "1" is filled in the corresponding squares in K-map. The function also has don't-care outputs for the minterms m_0, m_8, m_9 and m_{13} and so a "x" is filled in the corresponding squares in K-map and the remaining squares are filled with "0".

For NAND implementation, the Boolean function should be formed in SOP form. In the K-map of Fig. 1, by considering the don't-care condition in m_0 and m_8 as "1", three prime implicants can be formed with four adjacent 1's and the corresponding product terms are $A'D'$, $C'D'$ and $B'D'$. The simplified Boolean function is given by sum of these three product terms.

$$\therefore F = A'D' + C'D' + B'D'$$

Using the above Boolean function the logic circuit using only NAND gates is drawn as shown in Fig. 2.

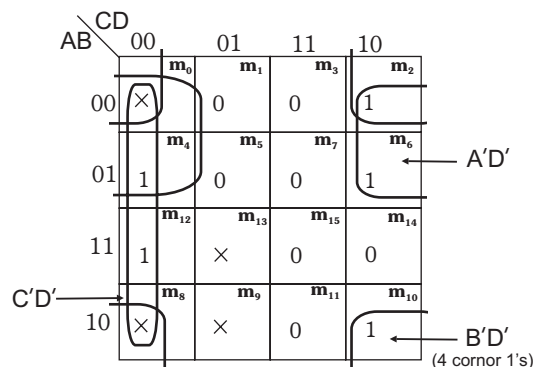


Fig. 1: K-map for F in SOP form.

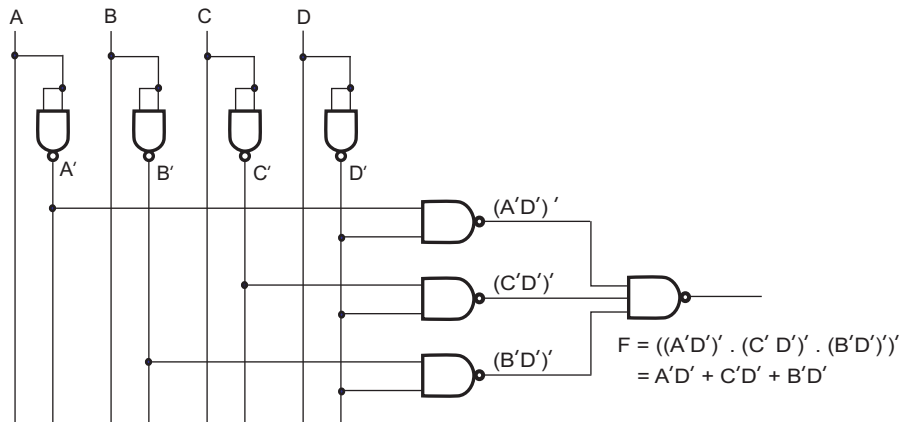


Fig. 2: Logic circuit of F using only NAND gates.

b) To Implement the Boolean function, F using only NOR gates

For NOR implementation, the Boolean function should be formed in POS form. For POS form of Boolean function, the prime implicants are formed by considering 0's. In the K-map of Fig. 3, by considering the don't-care condition in m_9 and m_{13} as "0" two prime implicants can be formed in which one prime implicant has eight adjacent 0's and one prime implicants has two adjacent 0's and the corresponding product terms are D and ABC . The simplified Boolean function in POS form is given by complement of sum of these two product terms.

$$\begin{aligned}
 F &= (D + ABC)' \\
 &= D'(ABC)' \\
 &= D'(A' + B' + C')
 \end{aligned}$$

CD \ AB	00	01	11	10
	m_0	m_1	m_3	m_2
00	×	0	0	1
01	m_4 1	0	0	1
11	m_{12} 1	×	0	0
10	×	×	0	1
m_8		m_9	m_{11}	m_{10}

← D (points to row 01 and 11)
 ← ABC (points to column 11 and 10)

Fig. 3: K-map for F in POS form.

Using the above POS form of Boolean function, the logic circuit using only NOR gates is drawn as shown in Fig. 4.

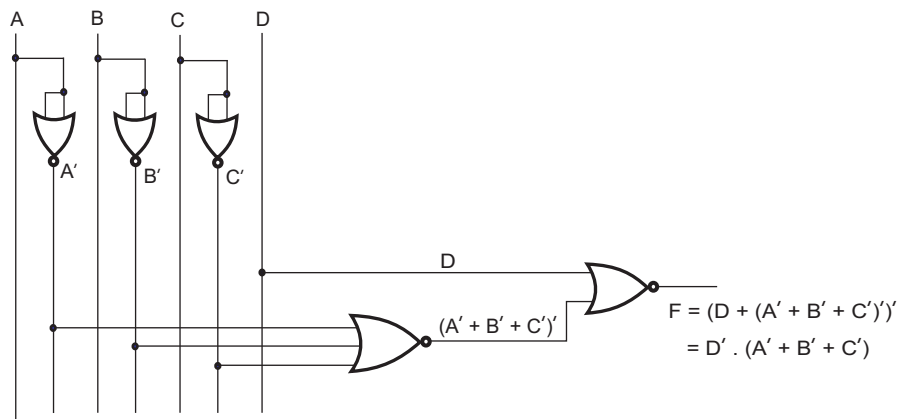


Fig. 4: Logic circuit of F using only NOR gates.

Example 1.34

Find the minimum sum of products (MSOP) representation for,

$$F(A, B, C, D, E) = \sum m(1, 4, 6, 10, 20, 22, 24, 26) + \sum d(0, 11, 16, 27)$$

using K-map. Draw the logical circuit of the minimal expression using only NAND gates.

Solution

The given function has five variables and so two 4-variable K-map each with 16 squares ($2^4 = 16$) are drawn as shown in Figs. 1 and 2. The K-map of Fig. 1 is drawn for minterms m_0 to m_{15} in which $A = 0$. The K-map of Fig. 2 is drawn for minterms m_{16} to m_{31} in which $A = 1$. The function is defined as sum of minterms $m_1, m_4, m_6, m_{10}, m_{20}, m_{22}, m_{24}$ and m_{26} and so a "1" is filled in the corresponding squares in K-maps. The function also has don't-care outputs for the minterms m_0, m_{11}, m_{16} and m_{27} and so a "x" is filled in the corresponding squares in K-maps and the remaining squares are filled with "0".

For NAND realization the Boolean function should be formed in SOP form. In order to find adjacent squares the K-maps of Figs. 1 and 2 are placed one over the other as shown in Fig. 3. In the K-map of Fig. 3, by considering the don't-care condition in m_0, m_{11}, m_{16} and m_{27} as "1", four prime implicants can be formed in which two has four adjacent 1's and the other two has two adjacent 1's and the corresponding product terms are $B'CE'$, $BC'D$, $A'B'C'D'$ and $ABC'E'$. The simplified Boolean function is given by sum of these four product terms.

$$\therefore F = B'CE' + BC'D + A'B'C'D' + ABC'E'$$

Using the above Boolean function the logic circuit using only NAND gate is drawn as shown in Fig. 4.

		A = 0			
		DE			
BC		00	01	11	10
		m_0	m_1	m_3	m_2
00		x	1	0	0
01		m_4	m_5	m_7	m_6
		1	0	0	1
11		m_{12}	m_{13}	m_{15}	m_{14}
		0	0	0	0
10		m_8	m_9	m_{11}	m_{10}
		0	0	x	1

Fig. 1: 4-variable K-map for $A = 0$.

		A = 1			
		DE			
BC		00	01	11	10
		m_{16}	m_{17}	m_{19}	m_{18}
00		x	0	0	0
01		m_{20}	m_{21}	m_{23}	m_{22}
		1	0	0	1
11		m_{28}	m_{29}	m_{31}	m_{30}
		0	0	0	0
10		m_{24}	m_{25}	m_{27}	m_{26}
		1	0	x	1

Fig. 2: 4-variable K-map for $A = 1$.

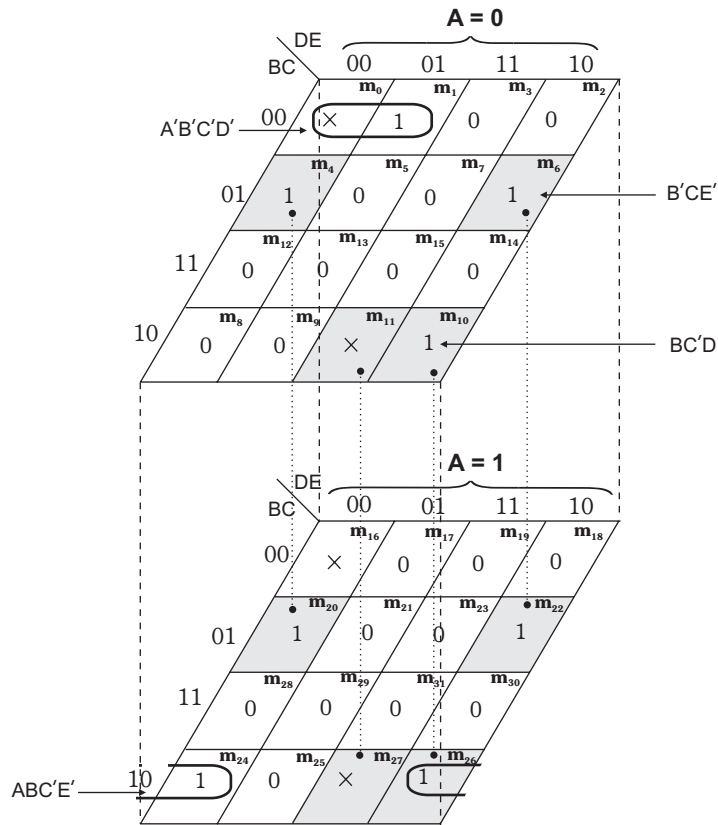
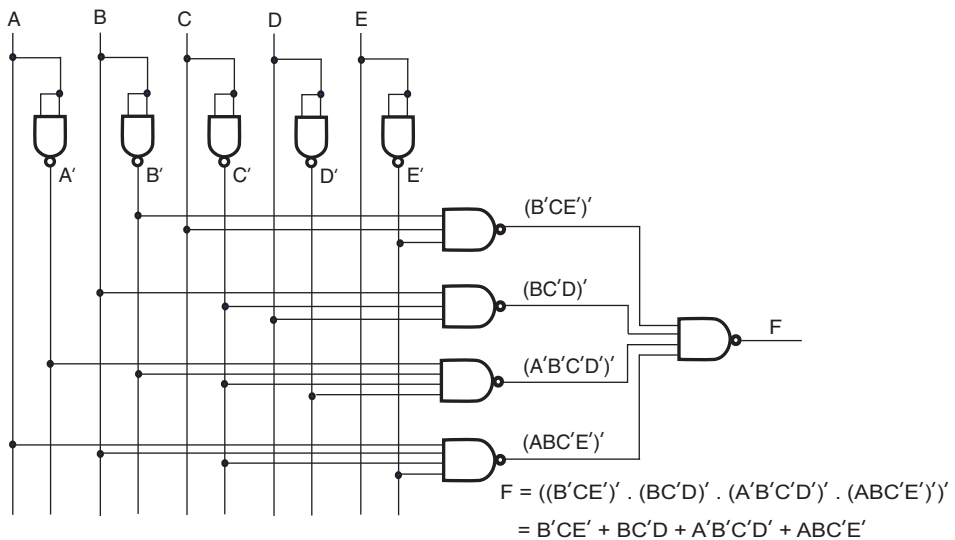


Fig. 3: 5-variable K-map.

Fig. 4: Logic circuit of function, F using only NAND gates.

Example 1.35

Find the minimum sum products of sum (MPOS) representation for,

$$F(A, B, C, D, E) = \prod M(0, 5, 7, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27) + \sum d(23, 31)$$

using K-map. Draw the logical circuit of the minimal expression using only NOR gates.

Solution

The given function has five variables and so two 4-variable K-map each with 16 squares ($2^4 = 16$) are drawn as shown in Figs. 1 and 2. The K-map of Fig. 1 is drawn for minterms m_0 to m_{15} in which $A = 0$. The K-map of Fig. 2 is drawn for minterms m_{16} to m_{31} in which $A = 1$. The function is defined as sum of minterms $m_0, m_5, m_7, m_8, m_9, m_{10}, m_{11}, m_{16}, m_{20}, m_{24}, m_{25}, m_{26}$ and m_{27} and so a "0" is filled in the corresponding squares in K-maps. The function also has don't-care outputs for the minterms m_{23} and m_{31} and so a "x" is filled in the corresponding squares in K-maps and the remaining squares are filled with "0".

For NOR realization the Boolean function should be formed in POS form. In order to find adjacent squares the K-maps of Figs. 1 and 2 are placed one over the other as shown in Fig. 3. In the K-map of Fig. 3, by considering the don't-care condition in m_{23} and m_{31} as "1", four prime implicants can be formed in which two has four adjacent 1's and the other two has two adjacent 1's and the corresponding product terms are $(B + C + D + E)$, $(A + B + C' + E')$, $(B' + C)$ and $(A' + B + D + E)$. The simplified Boolean function is given by sum of these four product terms.

$$\therefore F = (B + C + D + E) (A + B + C' + E') (B' + C) (A' + B + D + E)$$

Using the above Boolean function the logic circuit using only NAND gate is drawn as shown in Fig. 4.

		A = 0			
		DE			
BC		00	01	11	10
		m_0	m_1	m_3	m_2
00		0	1	1	1
01		m_4	m_5	m_7	m_6
		1	0	0	1
11		m_{12}	m_{13}	m_{15}	m_{14}
		1	1	1	1
10		m_8	m_9	m_{11}	m_{10}
		0	0	0	0

Fig. 1: 4-variable K-map for $A = 0$.

		A = 1			
		DE			
BC		00	01	11	10
		m_{16}	m_{17}	m_{19}	m_{18}
00		0	1	1	1
01		m_{20}	m_{21}	m_{23}	m_{22}
		0	1	x	1
11		m_{28}	m_{29}	m_{31}	m_{30}
		1	1	x	1
10		m_{24}	m_{25}	m_{27}	m_{26}
		0	0	0	0

Fig. 2: 4-variable K-map for $A = 1$.

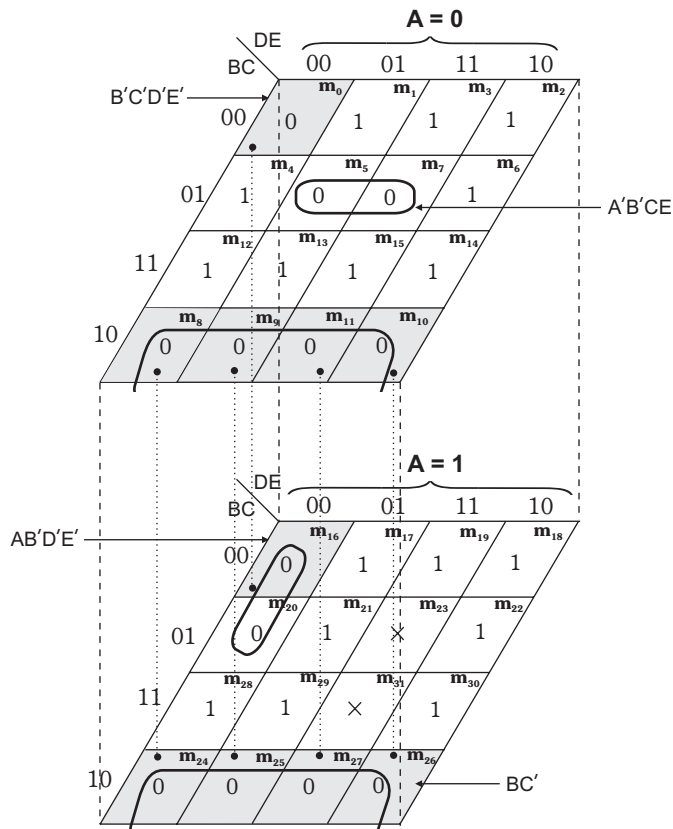
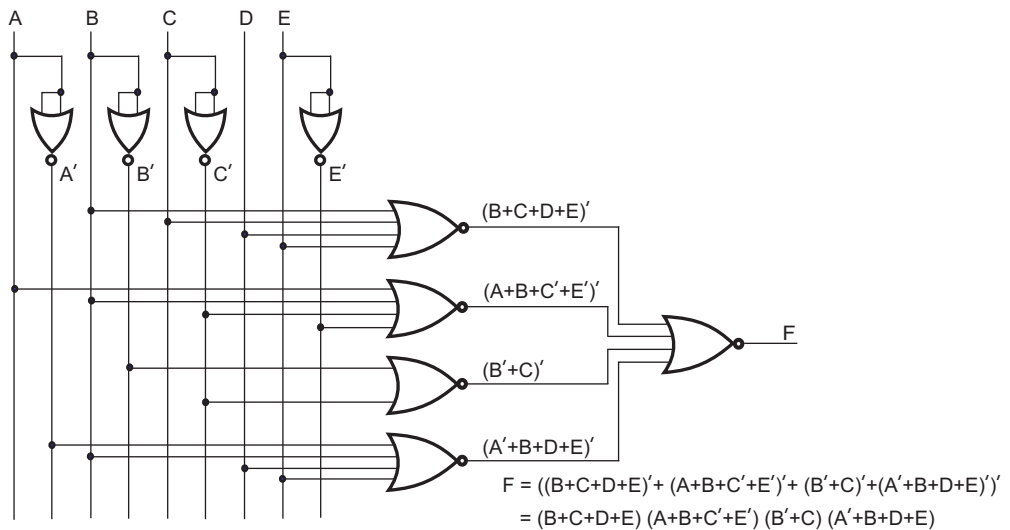


Fig. 3: 5-variable K-map.

Fig. 4: Logic circuit of function, F using only NOR gates.

1.7 Binary Arithmetic

Binary arithmetic is similar to decimal arithmetic of only two numbers 0 and 1. But in binary arithmetic, the result should be in binary. The rules for binary addition and subtraction are listed in Table 1.40.

Table 1.40: Rules of Binary Arithmetic

Binary Addition	Binary Subtraction
$0 + 0 = 0$	$0 - 0 = 0$
$0 + 1 = 1$	$0 - 1 = 1 \quad 1$
$1 + 0 = 1$	$1 - 0 = 1$
$1 + 1 = 1 \quad 0$	$1 - 1 = 0$
$\uparrow \quad \uparrow$ Carry Sum	$\uparrow \quad \uparrow$ Borrow Difference

1.7.1 Binary Addition

(AU, Nov/Dec'22, 2 Marks)

Addition of two binary numbers is performed by bit-by-bit addition starting from LSB (Least Significant Bit). The carry generated in one bit addition is considered in next bit addition. If carry is generated in the addition of MSB (Most Significant Bit) then it is considered as MSB of sum.

Example 1.36

Perform the following binary addition.

a) $10110_2 + 111_2$ b) $1010_2 + 11011_2$

Solution

a) $10110_2 + 111_2$

Addend:	1 0 1 1 0	← Carry
Augend: (+)	<div style="border-top: 1px solid black; padding-top: 2px;"> 1 1 1 0 1 </div>	
	<div style="border-top: 1px solid black; padding-top: 2px;"> 1 1 1 1 0 1 </div>	

$\uparrow \quad \uparrow \quad \uparrow$
 Carry

$\therefore 10110_2 + 111_2 = 11101_2$

b) $1010_2 + 11011_2$

Addend:	1 0 1 0	← Carry
Augend: (+)	<div style="border-top: 1px solid black; padding-top: 2px;"> 1 1 0 1 1 </div>	
	<div style="border-top: 1px solid black; padding-top: 2px;"> 1 1 0 1 0 1 </div>	

$\uparrow \quad \uparrow \quad \uparrow$
 Carry

$\therefore 1010_2 + 11011_2 = 100101_2$

1.7.2 Binary Subtraction

Subtraction of two binary numbers is performed by bit-by-bit subtraction starting from LSB (Least Significant Bit). When a subtraction of two bits need borrow then a 1 is borrowed from next available higher order bit and the borrowed bit is made 0.

When a larger magnitude number has to be subtracted from smaller magnitude number then the subtraction is performed by subtracting smaller magnitude number from larger magnitude number and a minus sign is added in the result.

Note: Binary number system and complement number system are presented in Appendices 6 and 7.

Example 1.37

Perform the following binary subtraction.

a) $10110_2 - 111_2$ b) $11000_2 - 1010_2$ c) $1011_2 - 11001_2$

Solution

a) $10110_2 - 111_2$

$$\begin{array}{r} \text{Minuend:} \quad 1 \ 0 \ 1 \ \textcircled{1} \ 0 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 1 \ 1 \\ \hline \quad \quad \quad 1 \end{array} \Rightarrow$$

$$\therefore 10110_2 - 111_2 = 1111_2$$

$$\begin{array}{r} \text{Minuend:} \quad 1 \ 0 \ \textcircled{1} \ 0 \ 10 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 1 \ 1 \\ \hline \quad \quad \quad 1 \ 1 \end{array} \Downarrow$$

$$\begin{array}{r} \text{Minuend:} \quad \textcircled{1} \ 0 \ 0 \ 10 \ 10 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \ 1 \end{array}$$

b) $11000_2 - 1010_2$

$$\begin{array}{r} \text{Minuend:} \quad 1 \ \textcircled{1} \ 0 \ 0 \ 0 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 0 \ 1 \ 0 \\ \hline \quad \quad \quad 1 \ 1 \ 0 \end{array} \Rightarrow$$

$$\therefore 11000_2 - 1010_2 = 1110_2$$

$$\begin{array}{r} \text{Minuend:} \quad \textcircled{1} \ 0 \ 1 \ 10 \ 0 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 0 \ 1 \ 0 \\ \hline 0 \ 1 \ 1 \ 1 \ 0 \end{array}$$

c) $1011_2 - 11001_2$

Here, the subtrahend is greater than the minuend. Hence, the minuend and subtrahend are interchanged and then the subtraction is performed and the result is considered as negative.

$$\begin{array}{r} \text{Minuend:} \quad 1 \ \textcircled{1} \ 0 \ 0 \ 1 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 0 \ 1 \ 1 \\ \hline \quad \quad \quad 1 \ 1 \ 0 \end{array} \Rightarrow$$

$$\therefore 1011_2 - 11001_2 = -1110_2$$

$$\begin{array}{r} \text{Minuend:} \quad \textcircled{1} \ 0 \ 1 \ 10 \ 1 \\ \text{Subtrahend: } (-) \quad \quad 1 \ 0 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \ 0 \end{array}$$

1.7.3 One's and Two's Complement Addition

Addition involves addition of two positive or negative numbers or addition of a positive and negative number. Usually, the binary numbers in complement form will have a fixed bit size which leads to a fixed range of positive and negative number representation for a specified bit size.

The addition of two positive numbers is same as unsigned binary addition except the handling of final carry. In one's complement addition the final carry is added to LSB (Least significant Bit) of sum but in two's complement addition the final carry is discarded. When negative numbers has to be added they are converted to complement form and then added. The final carry is handled similar to addition of positive numbers. But the sum can be positive or negative.

When addition involves negative numbers,

if MSB of sum is 0, then result is positive.

if MSB of sum is 1, then result is negative, take respective complement of sum and put "-" sign.

Example 1.38

Perform the following addition by one's and two's complement addition.

a) $01101_2 + (-01010_2)$ b) $00110_2 + (-01110_2)$ c) $(-01000_2) + (-01100_2)$ d) $00111_2 + 0010_2$

Solution

a) $01101_2 + (-01010_2)$

One's complement addition

$$\begin{array}{r} \text{Addend: } 01101_2 \Rightarrow \begin{array}{ccc} 1 & 1 & 1 \\ \hline & 0 & 1 & 1 & 0 & 1 \end{array} \leftarrow \text{Carry} \\ \text{Augend: } -01010_2 \xRightarrow{1's \text{ complement}} (+) \begin{array}{r} 10101 \\ \hline \end{array} \\ \begin{array}{r} \boxed{1} 00010 \\ \hline \text{Add carry } \rightarrow 1 \\ \hline 00011 \end{array} \end{array}$$

1's complement of 01010

01010

Comp \Downarrow

10101

Two's complement addition

$$\begin{array}{r} \text{Addend: } 01101_2 \Rightarrow \begin{array}{ccc} 1 & 1 & 1 \\ \hline & 0 & 1 & 1 & 0 & 1 \end{array} \leftarrow \text{Carry} \\ \text{Augend: } -01010_2 \xRightarrow{2's \text{ complement}} (+) \begin{array}{r} 10110 \\ \hline \end{array} \\ \begin{array}{r} \boxed{1} 00011 \\ \hline \text{Discard carry} \end{array} \end{array}$$

2's complement of 01010

01010

Comp \Downarrow

10101

+1

\hline

10110

$$\therefore 01101_2 + (-01010_2) = 00011_2$$

b) $00110_2 + (-01110_2)$

One's complement addition

$$\begin{array}{r} \text{Addend: } 00110_2 \Rightarrow 00110 \\ \text{Augend: } -01110_2 \xRightarrow{1's \text{ complement}} (+) \begin{array}{r} 10001 \\ \hline \end{array} \\ \hline 10111 \text{ (Sum)} \end{array}$$

Since MSB of sum is 1, the result is negative. Hence take 1's complement of 10111 (sum) and put "-" sign for sum.

$$10111 \xRightarrow{\text{complement}} 01000$$

$$\therefore \text{Sum} = -01000_2$$

1's complement of 01110

01110

Comp \Downarrow

10001

Two's complement addition

$$\begin{array}{r} \text{Addend: } 00110_2 \Rightarrow \begin{array}{ccc} 1 & 1 \\ \hline & 0 & 0 & 1 & 1 & 0 \end{array} \leftarrow \text{Carry} \\ \text{Augend: } -01110_2 \xRightarrow{2's \text{ complement}} (+) \begin{array}{r} 10010 \\ \hline \end{array} \\ \hline 11000 \text{ (Sum)} \end{array}$$

Since MSB of sum is 1, the result is negative. Hence take 2's complement of 11000 (sum) and put "-" sign for sum.

$$11000 \xRightarrow{\text{complement}} 00111$$

$$\begin{array}{r} \hline +1 \\ \hline 01000 \end{array}$$

$$\therefore \text{Sum} = -01000_2$$

2's complement of 01110

01110

Comp \Downarrow

10001

+1

\hline

10010

$$\therefore 00110_2 + (-01110_2) = -01000_2$$

c) $(-01000_2) + (-00100_2)$

One's complement addition

$$\begin{array}{r} \text{Addend: } -01000_2 \xRightarrow{1's \text{ complement}} \begin{array}{ccc} 1 & 1 & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 1 & 1 \end{array} \leftarrow \text{Carry} \\ \text{Augend: } -00100_2 \xRightarrow{1's \text{ complement}} (+) \begin{array}{r} 11011 \\ \hline \end{array} \\ \begin{array}{r} \boxed{1} 10010 \\ \hline \text{Add carry } \rightarrow 1 \\ \hline 10011 \end{array} \text{ (Sum)} \end{array}$$

Two's complement addition

$$\begin{array}{r} \text{Addend: } -01000_2 \xRightarrow{2's \text{ complement}} \begin{array}{ccc} 1 & 1 \\ \hline & 1 & 1 & 0 & 0 & 0 \end{array} \leftarrow \text{Carry} \\ \text{Augend: } -00100_2 \xRightarrow{2's \text{ complement}} (+) \begin{array}{r} 11100 \\ \hline \end{array} \\ \begin{array}{r} \boxed{1} 10100 \text{ (Sum)} \\ \hline \text{Discard carry} \end{array} \end{array}$$

1's complement of 01000	1's complement of 00100
01000	00100
Comp ↓	Comp ↓
10111	11011

Since MSB of sum is 1, the result is negative. Hence take 1's complement of 10011 (sum) and put "-" sign for sum.

$$10011 \xrightarrow{\text{complement}} 01100$$

$$\therefore \text{Sum} = -01100_2$$

2's complement of 01000	2's complement of 00100
01000	00100
Comp ↓	Comp ↓
10111	11011
+1	+1
<u>11000</u>	<u>11100</u>

Since MSB of sum is 1, the result is negative. Hence take 2's complement of 10100 (sum) and put "-" sign for sum.

$$10100 \xrightarrow{\text{complement}} 01011$$

$$\begin{array}{r} +1 \\ \hline 01100 \end{array}$$

$$\therefore \text{Sum} = -01100_2$$

$$\therefore -01000_2 + (-00100_2) = -01100_2$$

d) $00111_2 + 00101_2$

One's and two's complement addition

$$\begin{array}{r} 111 \leftarrow \text{Carry} \\ \text{Addend : } 00111 \\ \text{Augend : (+) } 00101 \\ \hline 01100 \end{array}$$

$$\therefore 00111_2 + 00101_2 = 01100_2$$

Errors in Complement Addition

In addition of two positive numbers the sum should not exceed maximum value of positive number representation for the bit size of the number and if it exceeds then the error is called overflow error. In addition of two negative numbers the sum should not go below the minimum value of negative number representation for the bit size of the number and if it goes below the minimum value then the error is called underflow error. In order to prevent overflow and underflow errors, the binary numbers should be sign extended to larger bit size before addition.

Example 1.39

Perform addition of two's complement numbers:

- a) 0110 and 0010 b) 1011 and 1010

Solution

a) 0110 and 0010

$$\begin{array}{r} 11 \leftarrow \text{Carry} \\ \text{Addend: } 0110 \\ \text{Augend: (+) } 0010 \\ \hline 1000 \text{ (Sum)} \end{array}$$

The most significant bit in the given numbers is 0 and hence they are positive numbers. But in sum the most significant bit is 1 which indicates a negative sum. In order to prevent the overflow error, the given numbers are sign extended to 5-bit numbers and addition is performed as shown ahead:

$$\begin{array}{rcl}
 0110 & \xrightarrow{\text{Sign extend to 5 bits}} & 00110 \\
 0010 & \xrightarrow{\text{Sign extend to 5 bits}} & 00010 \\
 \hline
 \therefore 0110_2 + 0010_2 = 1000_2
 \end{array}
 \quad
 \begin{array}{r}
 \text{Addend: } \begin{array}{r} 11 \\ 00110 \end{array} \leftarrow \text{Carry} \\
 \text{Augend: } (+) \begin{array}{r} 00010 \\ \hline 01000 \end{array} \text{ (Sum)}
 \end{array}$$

b) 1011 and 1010

$$\begin{array}{r}
 \text{Addend: } \begin{array}{r} 1 \quad 1 \\ 1011 \end{array} \leftarrow \text{Carry} \\
 \text{Augend: } (+) \begin{array}{r} 1010 \\ \hline 1 \quad 0101 \end{array} \text{ (Sum)} \\
 \uparrow \\
 \text{Discard carry}
 \end{array}$$

The most significant bit in the given numbers is 1 and hence they are negative numbers. But in sum the most significant bit is 0 which indicates a positive sum. In order to prevent the underflow error, the given numbers are sign extended to 5-bit numbers and addition is performed as shown ahead:

$$\begin{array}{rcl}
 1011 & \xrightarrow{\text{Sign extend to 5 bits}} & 11011 \\
 1010 & \xrightarrow{\text{Sign extend to 5 bits}} & 11010 \\
 \hline
 \therefore 1011_2 + 1010_2 = 10101_2
 \end{array}
 \quad
 \begin{array}{r}
 \text{Addend: } \begin{array}{r} 1 \quad 1 \quad 1 \\ 11011 \end{array} \leftarrow \text{Carry} \\
 \text{Augend: } (+) \begin{array}{r} 11010 \\ \hline 1 \quad 10101 \end{array} \text{ (Sum)} \\
 \uparrow \\
 \text{Discard carry}
 \end{array}$$

1.7.4 One's and Two's Complement Subtraction

In one's and two's complement subtraction, the subtrahend is represented in complement form and added to minuend. In one's complement subtraction if there is a final carry then it is added to LSB (Least Significant Bit) and in two's complement subtraction the final carry is discarded.

The sum can be positive or negative. Therefore, in complement subtraction,

if MSB of sum is 0, then result is positive.

if MSB of sum is 1, then result is negative, take respective complement of sum and put "-" sign.

Example 1.40

Perform the following subtraction by one's and two's complement subtraction.

a) $11101_2 - 10001_2$

b) $10001_2 - 11101_2$

Solution

a) $11101_2 - 10001_2$

$$\begin{array}{rcl}
 \text{Minuend: } & 11101_2 & \Rightarrow \begin{array}{r} 1 \quad 1 \quad 1 \\ 11101 \end{array} \leftarrow \text{Carry} \\
 \text{Subtrahend: } & -10001_2 & \xrightarrow{1's \text{ complement}} (+) \begin{array}{r} 01110 \\ \hline 1 \quad 01011 \end{array} \\
 & & \uparrow \\
 & & \text{Add carry} \\
 & & \hline
 & & 01100
 \end{array}$$

1's complement of 10001

10001

Comp \Downarrow

01110

$$\begin{array}{rcl}
 \text{Minuend: } & 11101_2 & \Rightarrow \begin{array}{r} 1 \quad 1 \quad 1 \quad 1 \quad 1 \\ 11101 \end{array} \leftarrow \text{Carry} \\
 \text{Subtrahend: } & -10001_2 & \xrightarrow{2's \text{ complement}} (+) \begin{array}{r} 01111 \\ \hline 1 \quad 01100 \end{array} \\
 & & \uparrow \\
 & & \text{Discard carry}
 \end{array}$$

2's complement of 10001

10001

Comp \Downarrow

01110

+1

\hline

01111

$$\therefore 11101_2 - 10001_2 = 01100_2$$

b) $10001_2 - 11101_2$ **One's complement subtraction**

$$\begin{array}{rcl}
 \text{Minuend:} & 10001_2 & \Rightarrow 10001 \\
 \text{Subtrahend: } -11101_2 & \xRightarrow{\text{1's complement}} & (+) 00010 \\
 \hline
 & & 10011 \text{ (Sum)} \\
 & \uparrow & \\
 & \text{No carry} &
 \end{array}$$

1's complement of 11101

$$\begin{array}{r}
 11101 \\
 \text{Comp} \downarrow \\
 00010
 \end{array}$$

1's complement of 10011

$$\begin{array}{r}
 10011 \\
 \text{Comp} \downarrow \\
 01100
 \end{array}$$

Since MSB of sum is 1, the result is negative. Hence take 1's complement of 10011 (sum) and put "-" sign for sum.
 $\therefore \text{Sum} = -01100_2$

Two's complement subtraction

$$\begin{array}{rcl}
 \text{Minuend:} & 10001_2 & \Rightarrow 10001 \quad \begin{array}{l} 11 \\ \leftarrow \text{Carry} \end{array} \\
 \text{Subtrahend: } -11101_2 & \xRightarrow{\text{2's complement}} & (+) 00011 \\
 \hline
 & & 10100 \text{ (Sum)} \\
 & \uparrow & \\
 & \text{No carry} &
 \end{array}$$

2's complement of 11101

$$\begin{array}{r}
 11101 \\
 \text{Comp} \downarrow \\
 00010 \\
 +1 \\
 \hline
 00011
 \end{array}$$

2's complement of 10100

$$\begin{array}{r}
 10100 \\
 \text{Comp} \downarrow \\
 01011 \\
 +1 \\
 \hline
 01100
 \end{array}$$

Since MSB of sum is 1, the result is negative. Hence take 2's complement of 10100 (sum) and put "-" sign for sum.
 $\therefore \text{Sum} = -01100_2$

$$\therefore 10001_2 - 11101_2 = -01100_2$$

Example 1.41

Perform one's complement and two's complement subtraction of fractional numbers 1101.101_2 and 1000.001_2 and compare the results with actual subtraction.

Solution**Case i: One's complement subtraction**

$$\begin{array}{rcl}
 \text{Minuend:} & 1101.101_2 & \xRightarrow{\text{Carry} \rightarrow 1 \ 1111} 1101.101 \\
 \text{Subtrahend: } -1000.001_2 & \xRightarrow{\text{1's complement}} & (+) 0111.110 \\
 \hline
 & & 10101.011 \\
 & \text{Add carry} \rightarrow +1 & \\
 \hline
 & & 0101.100
 \end{array}$$

1's complement of 1000.001

$$\begin{array}{r}
 1000.001 \\
 \text{Comp} \downarrow \\
 0111.110
 \end{array}$$

Case ii: Two's complement subtraction

$$\begin{array}{rcl}
 \text{Minuend:} & 1101.101_2 & \xRightarrow{\text{Carry} \rightarrow 1 \ 1111 \ 11} 1101.101 \\
 \text{Subtrahend: } -1000.001_2 & \xRightarrow{\text{2's complement}} & (+) 0111.111 \\
 \hline
 & & 10101.100 \\
 & \text{Discard carry} & \\
 \hline
 & & 0101.100
 \end{array}$$

2's complement of 1000.001

$$\begin{array}{r}
 1000.001 \\
 \text{Comp} \downarrow \\
 0111.110 \\
 +1 \\
 \hline
 0111.111
 \end{array}$$

$$\therefore 1101.101_2 - 1000.001_2 = 0101.100_2$$

Case iii: Actual Subtraction

$$\begin{array}{r}
 1101.101 \\
 - 1000.001 \\
 \hline
 0101.100
 \end{array}$$

Note: The results are same in direct subtraction, one's and two's complement subtraction.

1.7.5 Adders

Adders have been developed to perform arithmetic addition operation on binary numbers. Half adder can perform addition of two 1-bit binary number. The possible additions of two 1-bit binary numbers are shown below:

$0 + 0 =$	0		0 0
$0 + 1 =$	1	<u>2-bit standard representation</u> →	0 1
$1 + 0 =$	1		0 1
$1 + 1 =$	1 0		1 0
	↓	↓	↓
	Sum		Sum
	↓	↓	↓
	Carry		Carry

In the above additions, it can be observed that result is either 1-bit or 2-bit. The 1-bit result is called **sum**. In the 2-bit result, the first bit is called **sum** and the second bit is called **carry**. Hence, half adder is designed to generate 2-bit standard output.

In addition of n-bit binary numbers, the addition is performed bit-by-bit. In this the carry generated in an addition should be considered in next addition to get correct result and so full adder is developed which can add three binary input bits in which one of the bits is carry generated in previous addition. Therefore, n-bit addition can be performed by using n full adders in parallel.

1.7.6 Half Adder

(AU, Apr/May'23, 2 Marks)

Half adder (HA) is a combinational circuit that performs arithmetic sum of two binary bits. The outputs are sum and carry. The truth table of half adder is shown in Table 1.41. The block diagram and symbolic representation of half adder are shown in Figs. 1.43 and 1.46 respectively.

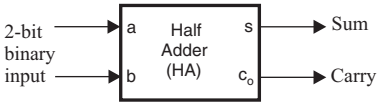


Fig. 1.43: Block diagram of half adder.

Table 1.41: Truth Table of Half Adder

Inputs		Minterm	Outputs	
a	b		s	c _o
0	0	m ₀	0	0
0	1	m ₁	1	0
1	0	m ₂	1	0
1	1	m ₃	0	1

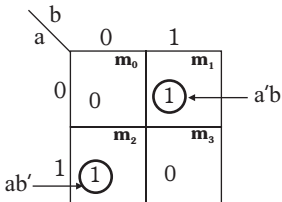


Fig. a: K-map for s.

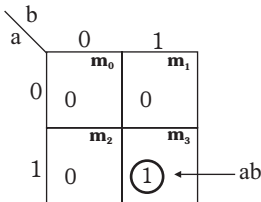


Fig. b: K-map for c_o.

Fig. 1.44: K-map for design of half adder.

Using truth table (Table 1.41) the K-maps for designing half adder are drawn as shown in Fig. 1.44. From the K-map we get the following Boolean equations and using these equations the logic circuit of half adder is drawn as shown in Fig. 1.45.

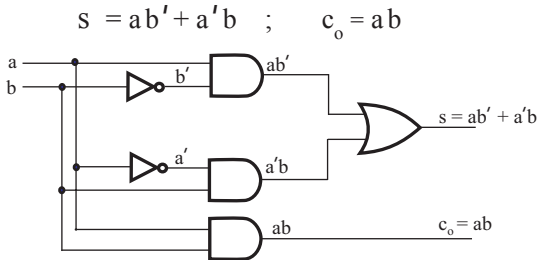


Fig.1.45: Logic circuit of half adder.

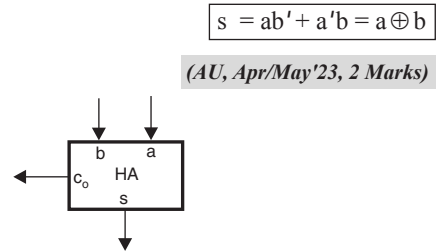


Fig. 1.46: Symbolic representation of half adder.

Alternatively, the sum can be realized from XOR gate. If we compare the sum and output of XOR gate we observe that both are same. The logic circuit of half adder using XOR gate is shown in Fig. 1.47.

Table 1.42: Truth Table of XOR Gate

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

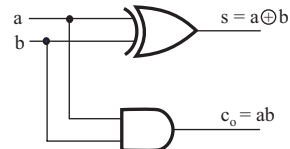


Fig. 1.47: Half adder using XOR gate.

1.7.7 Full Adder

(AU, Nov/Dec'23, 6 Marks)

(AU, Apr/May'23, 2 Marks)

Full adder (FA) is a combinational circuit that perform arithmetic sum of three binary bits in which one of the bit is carry generated in previous addition. Hence, full adder will be useful in n-bit addition to add carry of an addition in the next bit addition.

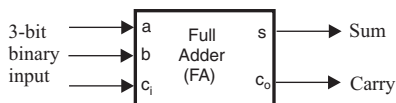


Fig. 1.48: Block diagram of full adder.

The outputs are sum and carry. The truth table of full adder is shown in Table 1.43. The block diagram and symbolic representation of full adder are shown in Figs. 1.48 and 1.53 respectively.

Table 1.43: Truth Table of Full Adder

Inputs				Outputs	
a	b	c_i		s	c_o
0	0	0	m_0	0	0
0	0	1	m_1	1	0
0	1	0	m_2	1	0
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

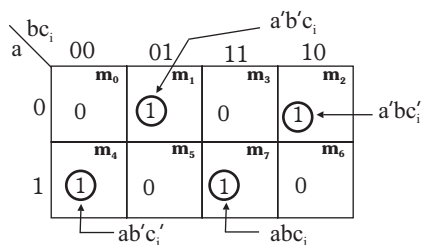


Fig. 1.49: K-map for s.

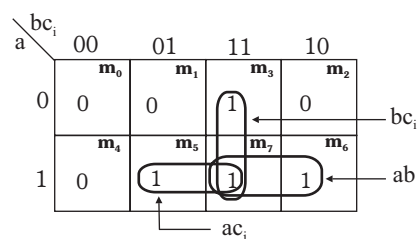


Fig. 1.50: K-map for c_o .

Using truth table (Table 1.43) the K-maps for designing full adder are drawn as shown in Figs. 1.49 and 1.50. From the K-map we get the following Boolean equations and using these equations the logic circuit of full adder is drawn as shown in Fig. 1.51.


Note: With reference to Example 1.26, sum, s can be further simplified to XOR of a , b and c_i .

$\therefore s = a \oplus b \oplus c_i$

a

b

c_i



$s = a \oplus b \oplus c_i$

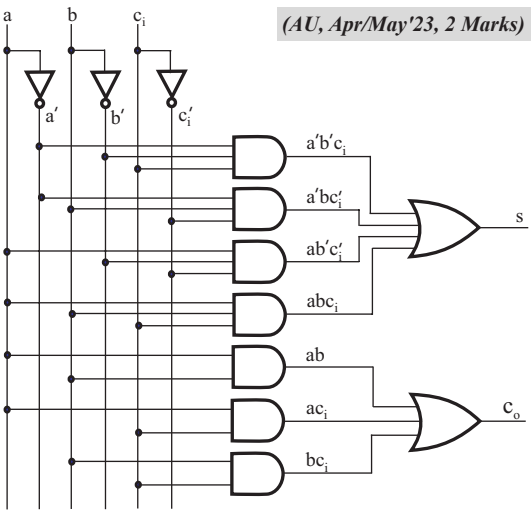


Fig. 1.51: Logic circuit of full adder.

Alternatively, full adder can be realized using two half adders as shown in Fig. 1.52. The first half adder is used to add two binary input bits and the second half adder is used to add sum of first half adder and third binary input.

(AU, Apr/May'23, 7 Marks)

The carry is obtained by logical OR of the carry of both the half adders.

The combinational circuit of Fig. 1.52 can be verified using the truth table (Table 1.44).

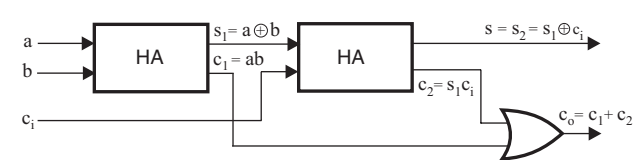


Fig. 1.52: Full adder using two half adders.

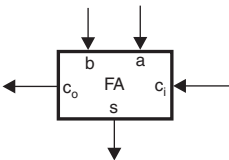


Fig. 1.53: Symbolic representation of full adder.

Table 1.44: Truth Table to Verify Logic Circuit of Fig. 1.52

Inputs			s_1	$s = s_2$	c_1	c_2	c_o
a	b	c_i	$a \oplus b$	$s_1 \oplus c_i$	ab	$s_1 c_i$	$c_1 + c_2$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	1	0	0	1	1
1	0	0	1	1	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	0	1	0	1
1	1	1	0	1	1	0	1

The following equations are obtained from design of half adder in Section 1.7.6, for the sum and carry of the logic circuit of Fig. 1.52.

$$\begin{array}{l|l} s_1 = a \oplus b & s = s_2 = s_1 \oplus c_i \\ c_1 = ab & c_2 = s_1 c_i \end{array}$$

Note: It can be observed that s and c_o of Table 1.44 is same as s and c_o of full adder.

Example 1.42

Design a half adder using NOR gates only.

Solution

The half adder has two inputs a , b and two outputs s , c_o (Refer Fig. 1.46). The truth table of half adder along with minterms is shown in Table 1.

Using truth table (Table 1) the K-maps for designing half adder are drawn as shown in Figs. 1 and 2.

Table 1: Truth Table of Half Adder

Inputs		Minterm	Outputs	
a	b		s	c_o
0	0	m_0	0	0
0	1	m_1	1	0
1	0	m_2	1	0
1	1	m_3	0	1

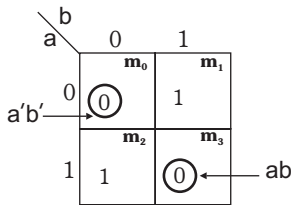


Fig. 1: K-map for s .

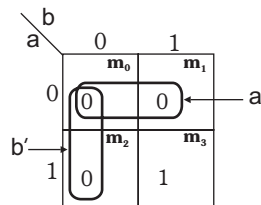


Fig. 2: K-map for c_o .

From the K-maps we get the following Boolean equations,

$$s = (a'b' + ab)' = (a'b')' \cdot (ab)' = (a+b) \cdot (a'+b')$$

$$c_o = (a' + b')'$$

Using the above Boolean equations the logic circuit of half adder is drawn using only NOR gates as shown in Fig. 3.

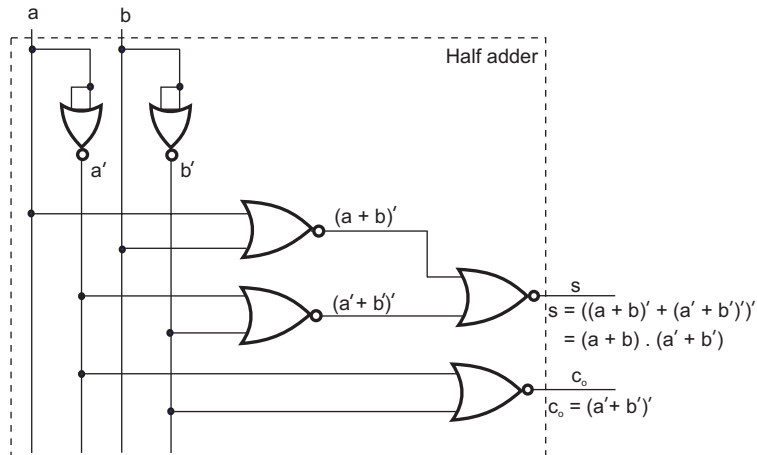


Fig. 3: Logic circuit of half adder using NOR gates.

Example 1.43

Design a full adder using NAND gates only.

Solution

The full adder has three inputs a , b , c_i and two outputs s , c_o (Refer Fig. 1.53). The truth table of full adder along with minterms is shown in Table 1.

Using truth table (Table 1) the K-maps for designing full adder are drawn as shown in Figs. 1 and 2.

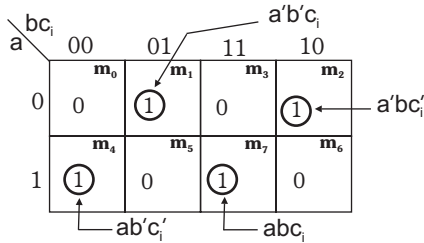


Fig. 1: K-map for s .

From the K-maps we get the following Boolean equations,

$$s = a'b'c_i + a'bc_i' + ab'c_i + abc_i$$

$$c_o = ab + ac_i + bc_i$$

Using the above Boolean equations the logic circuit of full adder using only NAND gates is drawn as shown in Fig. 3.

Table 1: Truth Table of Full Adder

Inputs			Outputs	
a	b	c_i	Sum s	Carry c_o
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

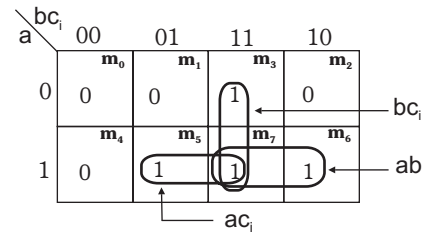


Fig. 2: K-map for c_o .

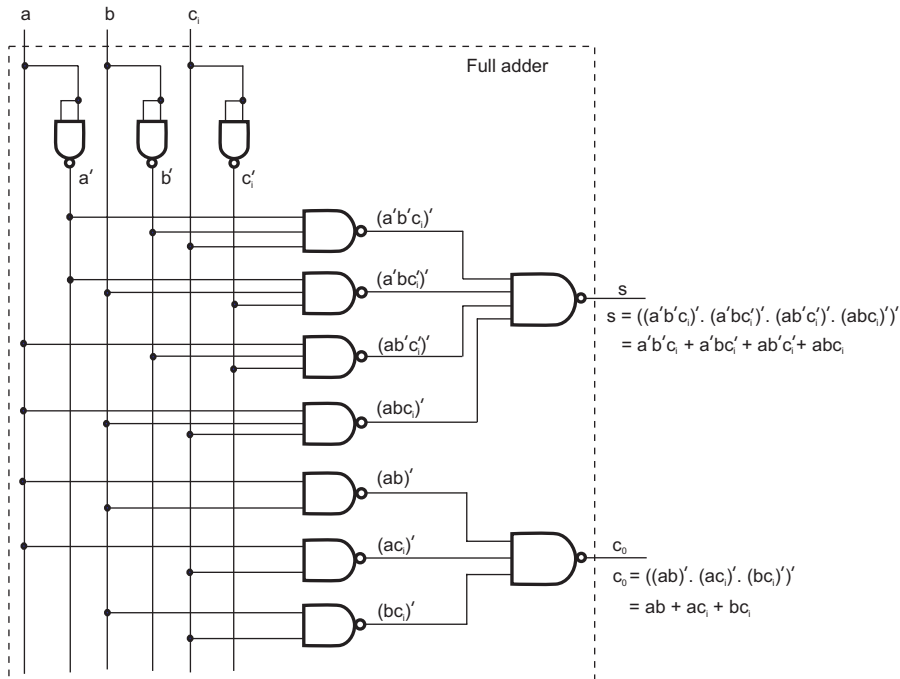


Fig. 3: Logic circuit of full adder using NAND gates.

1.7.8 Binary Parallel Adder

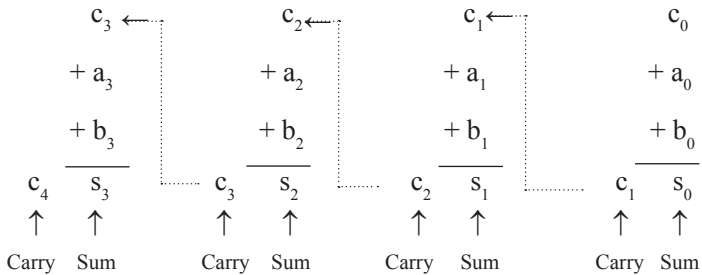
A **binary parallel adder** performs arithmetic sum of two n -bit binary numbers. A full adder can add two 1-bit binary along with previous carry. Hence, in order to add two n -bit binary numbers, n full adders are required. Each full adder will add one bit of binary numbers.

Consider two, 4-bit binary numbers A and B shown below:

Binary number-1: $a_3 a_2 a_1 a_0$ (A)

Binary number-2: $b_3 b_2 b_1 b_0$ (B)

The bit-by-bit addition is performed as follows.



In the above addition the initial carry $c_0 = 0$. The carry generated in an addition is considered in the next addition. A 4-bit binary adder to perform above addition is constructed using 4 full adders as shown in Fig. 1.54. The symbolic representations of 4-bit binary adders are shown in Fig. 1.55.

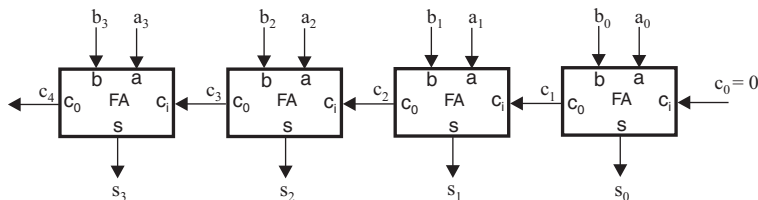


Fig. 1.54: 4-bit binary adder using full adders.

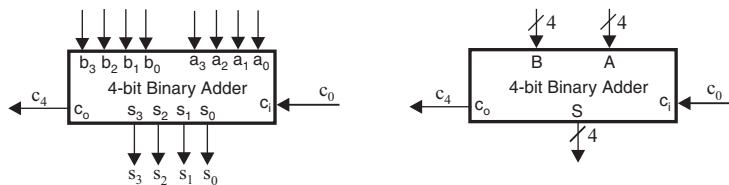


Fig. a.

Fig. b.

Fig. 1.55: Symbolic representation of 4-bit binary adder.

The 4-bit binary adder is available as a standard IC with number 7483. The pin configuration of 7483 is shown in Fig. 1.56.

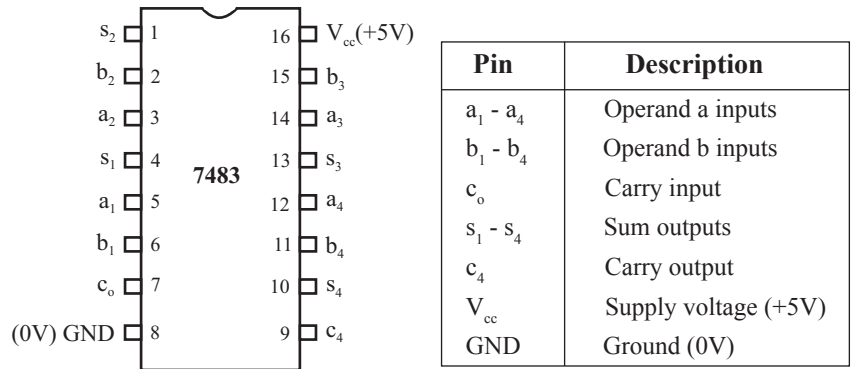


Fig. 1.56: Pin configuration of 4-bit binary adder IC 7483.

The 4-bit binary adders can be connected in parallel to perform addition of higher bit size binary. Two 4-bit adders can be connected as shown in Fig. 1.57 to add 8-bit binary numbers. Three 4-bit adders can be connected as shown in Fig. 1.58 to add 12-bit binary numbers and so on.

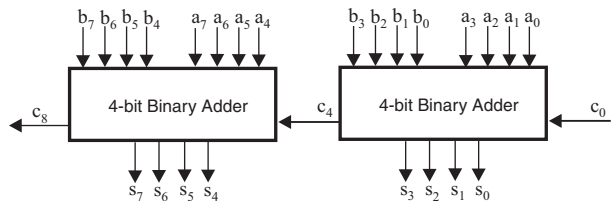


Fig. 1.57: 8-bit binary adder using 4-bit binary adders.

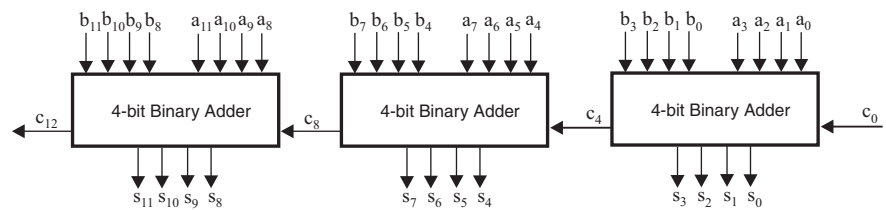


Fig. 1.58: 12-bit binary adder using 4-bit binary adders.

Alternatively, 8-bit binary adder can be constructed by connecting 8 full adders in parallel as shown in Fig. 1.59. In general, n-bit binary adder can be constructed by connecting n full adders in parallel as shown in Fig. 1.60.

Note: n-bit binary adder is called parallel adder because it add all bits of data simultaneously.

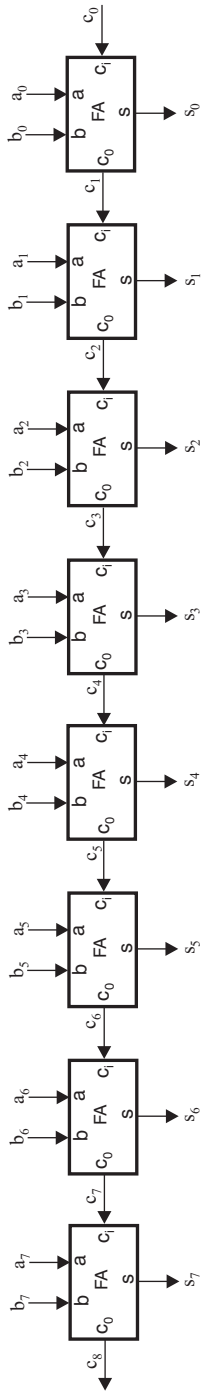


Fig. 1.59: 8-bit binary adder using full adders.

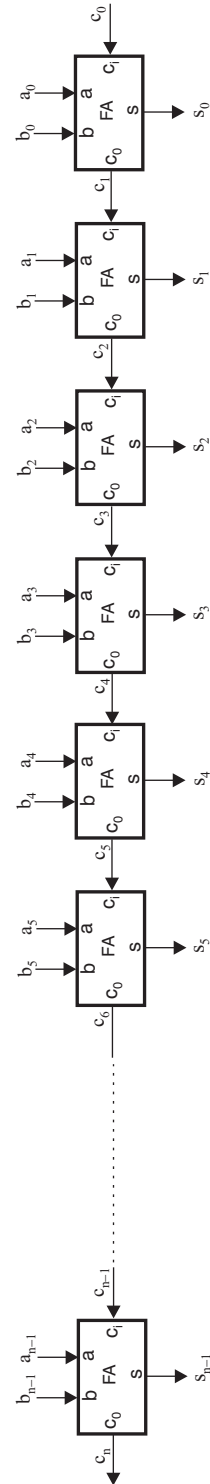


Fig. 1.60: n -bit binary adder using full adders.

1.7.9 BCD Adder (Decimal Adder)

In decimal addition we can use binary or BCD to represent decimal numbers. In both the representation when binary adders are used to perform addition the result or sum will be in binary. But in BCD addition we need result/sum in BCD. Hence we need additional logic circuit in binary adders to convert binary result to BCD.

The decimal digits represented in BCD are,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Consider the following example of decimal addition.

$$\begin{array}{r}
 \begin{array}{c} 1 \\ 4 \\ + 5 \\ \hline 9 \end{array} \begin{array}{c} 2 \\ 3 \\ 6 \end{array} \begin{array}{c} 9 \\ 9 \\ 9 \end{array} \begin{array}{c} 8 \\ 7 \\ 5 \end{array} \\
 \begin{array}{c} \leftarrow \text{Carry} \\ \text{Maximum possible sum} = 1 + 9 + 9 = 19
 \end{array}
 \end{array}$$

From the above example it is evident that maximum possible sum of two BCD digits is,

$$1 + 9 + 9 = 19$$

↑

Previous carry

Therefore in BCD addition the sum of any two digits will be in the range 0 to 19_{10} . When sum is in the range 0 to 9_{10} no correction is needed and it can be represented by 4 digit BCD. But for sum in the range 10_{10} to 19_{10} a correction 6_{10} is added to sum to get the result in BCD. (Refer Section 1.8.1 for examples of BCD addition.)

The **BCD adder** is a binary adder with additional logic circuit to perform addition of correction 6_{10} when the sum of a BCD digit exceeds 9_{10} .

A truth table (Table 1.45) is formed to show the range of binary sum and correction needed to convert binary sum to BCD sum. In Table 1.45, the binary sum is denoted as $z_8 z_4 z_2 z_1$ and BCD sum as $s_8 s_4 s_2 s_1$. Here, k is carry in binary sum and c is carry in BCD sum.

From the truth table (Table 1.45) we can observe that the correction by adding $+6_{10}$ (0110_2) can be achieved using following three conditions.

1. When binary sum is 10_{10} and 11_{10} correction 6 can be added if both z_2 and z_8 are 1.
2. When binary sum is in the range 12_{10} to 15_{10} correction 6 can be added if both z_4 and z_8 are 1.
3. When binary sum is in the range 16_{10} to 19_{10} correction 6 can be added if carry k in binary sum is 1.

Therefore, during BCD addition, first binary addition is performed using 4-bit binary adder then additional logic circuit is used to generate correction 6 for the conditions mentioned above and another 4-bit binary adder is used to add correction 6 to binary sum so that the binary sum is converted to BCD sum.

Note: BCD representation of decimal numbers are presented in Appendix 9.

Table 1.45: Truth Table for Conversion of Binary to BCD Sum

Decimal Sum	Binary Addition		Correction	BCD Addition	
	Carry k	Binary Sum $z_8 z_4 z_2 z_1$	Binary Sum + 6_{10} (0110)	Carry c	BCD Sum $s_8 s_4 s_2 s_1$
0	0	0 0 0 0	No Correction	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	01010 + 0110	1	0 0 0 0
11	0	1 0 1 1	01011 + 0110	1	0 0 0 1
12	0	1 1 0 0	01100 + 0110	1	0 0 1 0
13	0	1 1 0 1	01101 + 0110	1	0 0 1 1
14	0	1 1 1 0	01110 + 0110	1	0 1 0 0
15	0	1 1 1 1	01111 + 0110	1	0 1 0 1
16	1	0 0 0 0	10000 + 0110	1	0 1 1 0
17	1	0 0 0 1	10001 + 0110	1	0 1 1 1
18	1	0 0 1 0	10010 + 0110	1	1 0 0 0
19	1	0 0 1 1	10011 + 0110	1	1 0 0 1

The logic circuit to perform 1-digit BCD addition using two 4-bit binary adders is shown in Fig. 1.61. The symbolic representation of BCD adder is shown in Fig. 1.62.

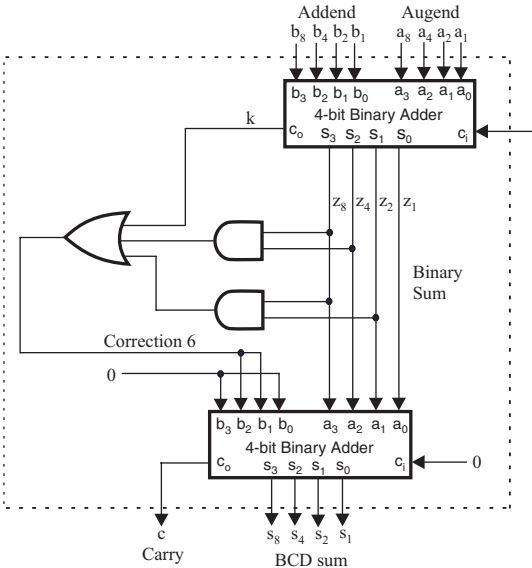


Fig. 1.61: BCD adder using binary adders.

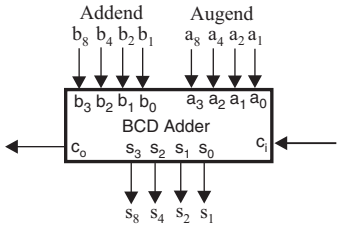


Fig. a.

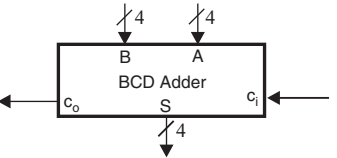


Fig. b.

Fig. 1.62: Symbolic representation of BCD adder.

Fig. 1.64: Pin configuration of 4-bit BCD adder IC 74583.

1.7.10 Subtractors

Subtractors have been developed to perform arithmetic subtraction operation on binary numbers. Half subtractor can perform subtraction of two 1-bit binary number. The possible subtractions of two 1-bit binary numbers are shown below:

$0 - 0 = 0$		0 0
$0 - 1 = 1\ 1$	2-bit standard representation →	1 1
$1 - 0 = 1$		0 1
$1 - 1 = 0$		0 0
	└─→ Difference	└─→ Difference
	└─→ Borrow	└─→ Borrow

In the above subtractions, it can be observed that result is either 1-bit or 2-bit. The 1-bit result is called **difference**. In the 2-bit result, the first bit is called **difference** and the second bit is called **borrow**. Hence, half subtractor is designed to generate 2-bit standard output.

In subtraction of n-bit binary numbers, the subtraction is performed bit-by-bit. In this the borrow generated in a subtraction should be considered in next subtraction to get correct result and so full subtractor is developed which can subtract three binary input bits in which one of the bits is borrow generated in previous subtraction. Therefore, n-bit subtraction can be performed by using n full subtractors in parallel.

In practice, subtraction is performed in two's complement method in which the subtrahend is converted to two's complement form and added with minuend. Therefore, adders are used to perform subtraction as an addition of positive and negative number. The addition of positive number and two's complement of negative number will give the result of subtraction.

1.7.11 Half Subtractor

Half subtractor is a combination circuit that performs arithmetic subtraction of two binary bits.

The outputs are difference and borrow. The truth table of half subtractor is shown in Table 1.46. The block diagram of half subtractor is shown in Fig. 1.65.

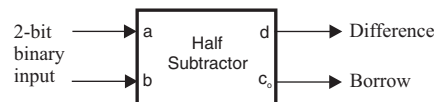


Fig. 1.65: Block diagram of half subtractor.

Table 1.46: Truth Table of Half Subtractor

Inputs		Minterm	Outputs	
a	b		d	c ₀
0	0	m ₀	0	0
0	1	m ₁	1	1
1	0	m ₂	1	0
1	1	m ₃	0	0

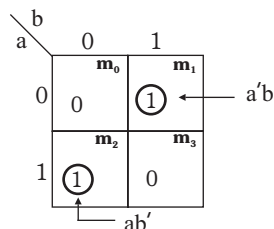


Fig. a: K-map for d.

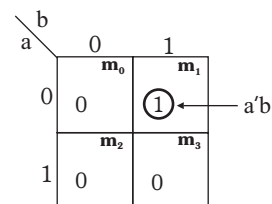


Fig. b: K-map for c₀.

Fig. 1.66: K-map for design of half subtractor.

Using truth table (Table 1.46) the K-maps for designing half subtractor are drawn as shown in Fig. 1.66.

From the K-map we get the following Boolean equations,

$$d = ab' + a'b$$

$$c_o = a'b$$

$$d = ab' + a'b = a \oplus b$$

Using the above Boolean equations the logic circuit of half subtractor is drawn as shown in Fig. 1.67.

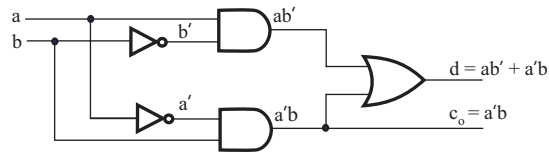


Fig. 1.67: Logic circuit of half subtractor.

Alternatively, the difference can be realized from XOR gate. If we compare the difference and output of XOR gate, we observe that both are same. The logic circuit of half subtractor using XOR gate is shown in Fig. 1.68.

Table 1.47: Truth Table of XOR Gate

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

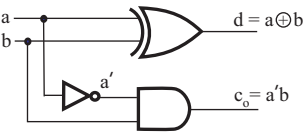


Fig. 1.68: Half subtractor using XOR gate.

1.7.12 Full Subtractor

(AU, Nov/Dec'23, 7 Marks)

Full subtractor is a combinational circuit that perform arithmetic subtraction of three binary bits in which one of the bit is borrow generated in previous subtraction.

Hence, full subtractor will be useful in *n*-bit subtraction to subtract borrow of a subtraction in next bit subtraction. The outputs are difference and borrow. The truth table of full subtractor is shown in Table 1.48. The block diagram of full subtractor is shown in Fig. 1.69.

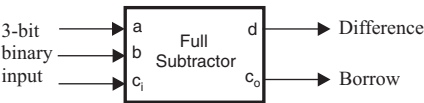


Fig. 1.69: Block diagram of full subtractor.

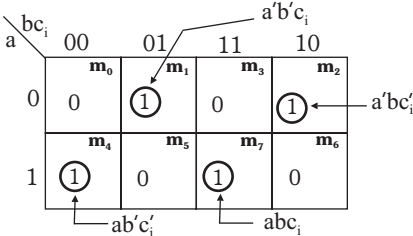


Fig. a: K-map for *d*.

Table 1.48: Truth Table of Full Subtractor

Inputs			Minterm	Outputs	
a	b	<i>c_i</i>		d	<i>c_o</i>
0	0	0	<i>m₀</i>	0	0
0	0	1	<i>m₁</i>	1	1
0	1	0	<i>m₂</i>	1	1
0	1	1	<i>m₃</i>	0	1
1	0	0	<i>m₄</i>	1	0
1	0	1	<i>m₅</i>	0	0
1	1	0	<i>m₆</i>	0	0
1	1	1	<i>m₇</i>	1	1

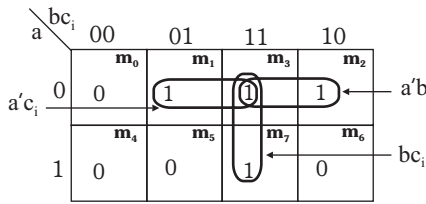
Fig. b: K-map for c_o .

Fig. 1.70: K-map for design of full subtractor.

Using truth table (Table 1.48) the K-maps for designing full subtractor are drawn as shown in Fig. 1.70.

From the K-maps we get the following Boolean equations,

$$d = a'b'c_i + a'bc_i' + ab'c_i' + abc_i$$

$$c_o = a'b + a'c_i + bc_i$$

Using the above Boolean equations the logic circuit of full subtractor is drawn as shown in Fig. 1.71.

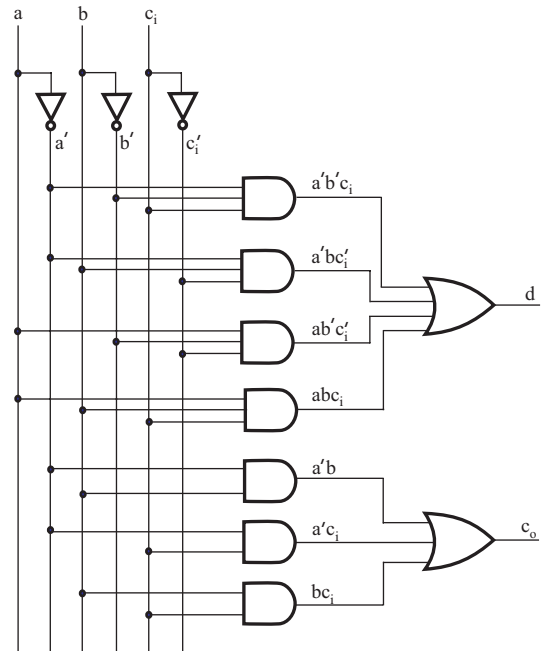


Fig. 1.71: Logic circuit of full subtractor.

Example 1.44

Design a half subtractor using NAND gates only.

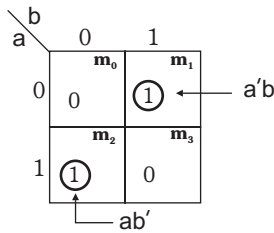
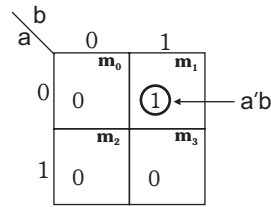
Solution

The half subtractor has two inputs a, b and two outputs d, c_o (Refer Fig. 1.67). The truth table of half subtractor along with minterms is shown in Table 1.

Table 1: Truth Table of Half Subtractor

Inputs		Minterm	Outputs	
a	b		d	c_o
0	0	m_0	0	0
0	1	m_1	1	1
1	0	m_2	1	0
1	1	m_3	0	0

Using truth table (Table 1) the K-maps for designing half subtractor are drawn as shown in Figs. 1 and 2.

Fig. 1: K-map for d .Fig. 2: K-map for c_o .

From the K-maps we get the following Boolean equations,

$$d = ab' + a'b \quad ; \quad c_o = a'b$$

Using the above Boolean equations the logic circuit of half subtractor using only NAND gates is drawn as shown in Fig. 3.

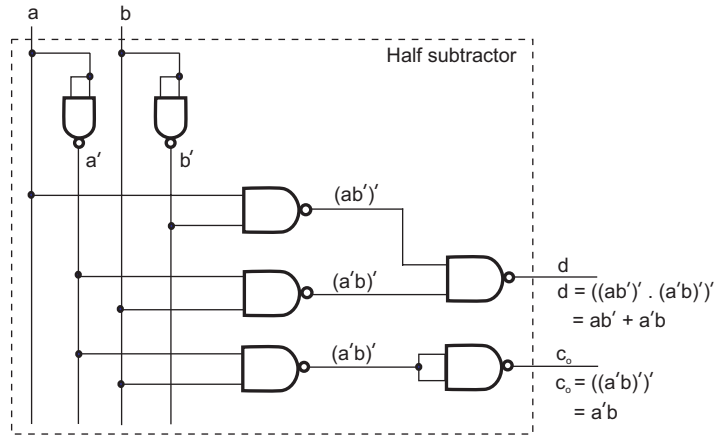


Fig. 3: Logic circuit of half subtractor using NAND gates.

Example 1.45

Design a full subtractor using NAND gates.

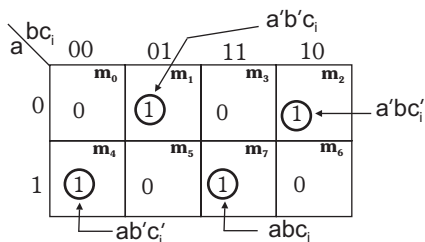
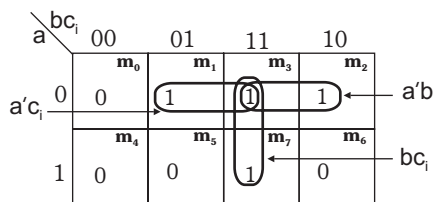
Solution

The full subtractor has three inputs a, b, c_i and two outputs d, c_o (Refer Fig. 1.71). The truth table of full subtractor along with minterms is shown in Table 1.

Table 1: Truth Table of Full Subtractor

Inputs			Minterm	Outputs	
a	b	c		d	c_o
0	0	0	m_0	0	0
0	0	1	m_1	1	1
0	1	0	m_2	1	1
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	0
1	1	0	m_6	0	0
1	1	1	m_7	1	1

Using truth table (Table 1) the K-maps for designing full subtractor are drawn as shown in Figs. 1 and 2.

Fig. 1: K-map for d .Fig. 2: K-map for c_o .

From the K-maps we get the following Boolean equations,

$$d = a'b'c_i + a'bc_i + ab'c_i + abc_i \quad ; \quad c_o = a'b + a'c_i + bc_i$$

Using the above Boolean equations the logic circuit of full subtractor using only NAND gates is drawn as shown in Fig. 3.

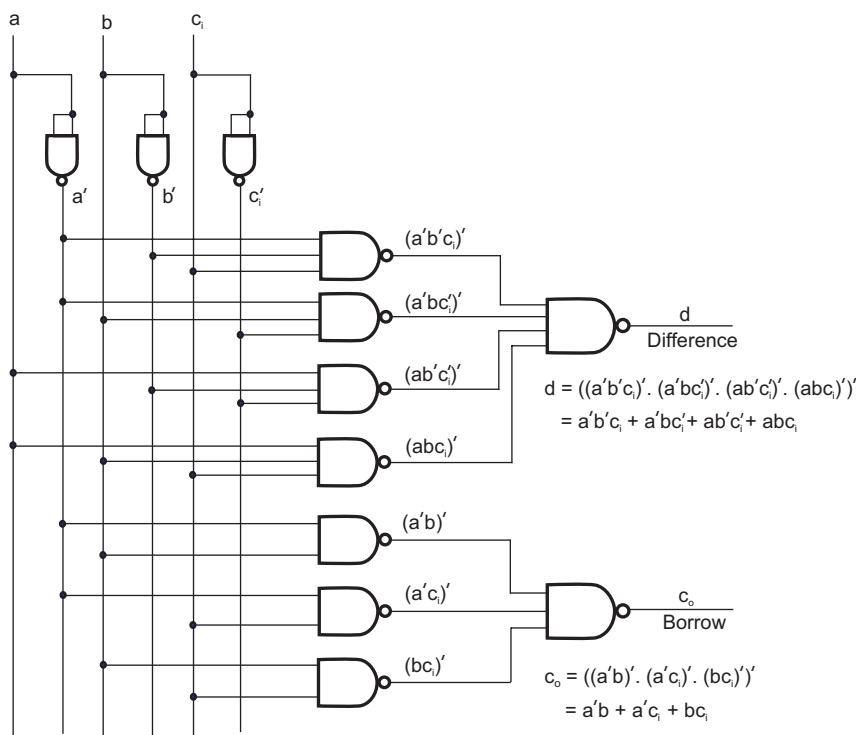


Fig. 3: Logic circuit of full subtractor using NAND gates.

1.7.13 Binary Parallel Adder/Subtractor

Practically subtraction is performed only in 2's complement form in which the subtraction is considered as addition of positive and negative numbers and the negative number is represented in 2's complement form. Therefore, subtractor circuit is not of much use and subtraction can be performed using adder itself.

Therefore, subtraction of two BCD digit A and B is considered as,

$$A - B = A + (-B)$$

↑ 2's complement of B

The complement of binary can be easily obtained using XOR gate. Consider the truth table of XOR gate (Table 1.49).

Table 1.49: Truth Table of XOR Gate

x	y	$x \oplus y = y$
0	0	0
0	1	1

x	y	$x \oplus y = y'$
1	0	1
1	1	0

With reference to Table 1.49 we can say that,

If $x = 0$, then $x \oplus y = y$

If $x = 1$, then $x \oplus y = y'$

Therefore in XOR gate, input x can be used as control or selection and input y can be data.

If $x = 0$, the data will be output as such.

If $x = 1$, the output will be complement of data.

The n -bit binary adder using full adder can be modified to perform both addition and subtraction as shown in Fig. 1.72.

One input can be given directly to full adders, another input can be given through XOR gate to full adders, so that second data can be sent to full adders with or without complement.

The control input M is used to perform complement operation. When $M = 1$, the output of XOR gate is complement of the input.

Consider addition and subtraction of two, 4-bit binary numbers.

Binary number-1: $a_3 a_2 a_1 a_0$ (data-A)

Binary number-2: $b_3 b_2 b_1 b_0$ (data-B)

Let us perform, $A + B$ and $A - B = A + (-B)$

For addition, B is input to full adder without complement as shown in Fig. 1.73.

\therefore For addition, $M = c_0 = 0$

For subtraction, B is input to full adder after complement and adding 1 for 2's complement can be performed by making $c_0 = 1$ for subtraction as shown in Fig. 1.74.

\therefore For subtraction, $M = c_0 = 1$

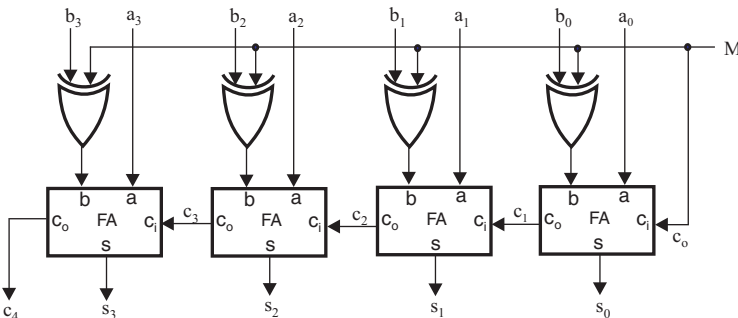


Fig. a: Binary adder/subtractor using full adders.

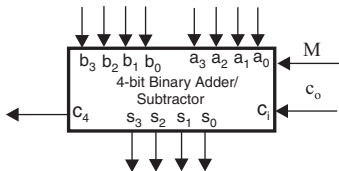


Fig. b: Symbolic representation.

Fig. 1.72: 4-bit binary adder/subtractor.

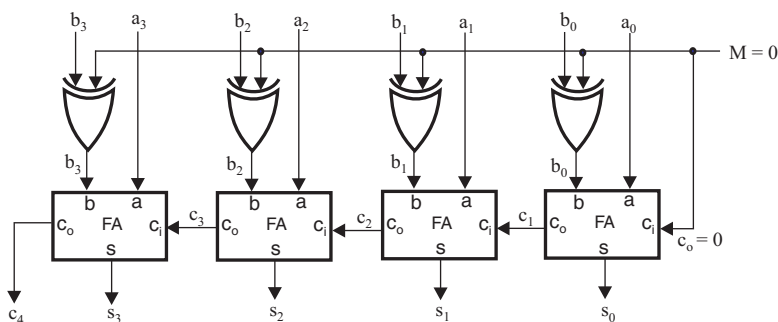


Fig. 1.73: Binary addition using adder/subtractor.

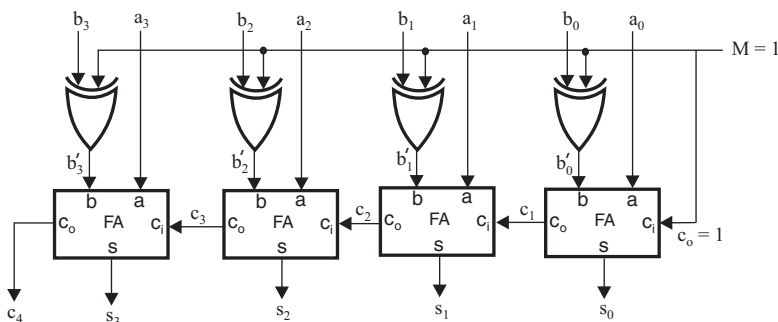


Fig. 1.74: Binary subtraction using adder/subtractor.

Note: Sum is difference and carry is borrow.

1.8 Magnitude Comparator

Magnitude comparator is a combinational circuit used to compare two binary numbers and determine whether they are equal or unequal and if unequal then it can make a decision on larger or smaller magnitude.

1.8.1 4-bit Magnitude Comparator

(AU, Apr/May'23, 7 Marks)

Consider two 4-bit binary numbers A and B.

Let, $A = a_3 a_2 a_1 a_0$ and $B = b_3 b_2 b_1 b_0$

On comparing the two numbers using magnitude comparator it is possible to determine any one of the following condition.

1. $A = B$
2. $A < B$
3. $A > B$

The 4-bit magnitude comparator is available as a standard IC with number 7485. The pin configuration of 7485 is shown in Fig. 1.75.

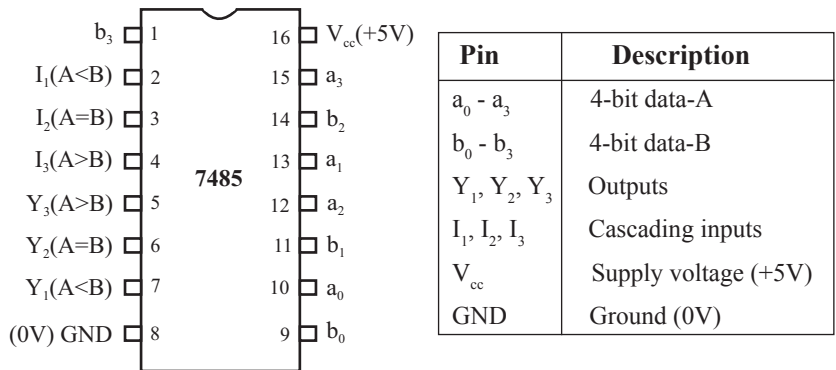


Fig. 1.75: Pin configuration of 4-bit magnitude comparator IC 7485.

Checking for A = B

For, $A = B$, every digit of A and B should be equal.

$$\begin{array}{l|l} \text{i.e, } a_3 = b_3 & a_1 = b_1 \\ a_2 = b_2 & a_0 = b_0 \end{array}$$

The equality can be ensured by the following Boolean function.

$$\text{Let, } x_3 = (a'_3 b_3 + a_3 b'_3)' ; \text{ if } x_3 = 1 \text{ then } a_3 = b_3$$

$$x_2 = (a'_2 b_2 + a_2 b'_2)' ; \text{ if } x_2 = 1 \text{ then } a_2 = b_2$$

$$x_1 = (a'_1 b_1 + a_1 b'_1)' ; \text{ if } x_1 = 1 \text{ then } a_1 = b_1$$

$$x_0 = (a'_0 b_0 + a_0 b'_0)' ; \text{ if } x_0 = 1 \text{ then } a_0 = b_0$$

Now the combined Boolean equation for $A = B$, is,

$$x_3 x_2 x_1 x_0 = 1$$

$$\text{Let, } F_1 = x_3 x_2 x_1 x_0$$

$$\text{Now, if } A = B, \text{ then } F_1 = 1$$

$$\text{and, if } A \neq B, \text{ then } F_1 = 0$$

Every digit of the binary numbers can take two possible values 0 and 1. Therefore, the value of $x_3 = 1$, when the digits a_3 and b_3 take either, both 0 or both 1. This can be verified in Table 1.50. Similarly the equality of other bits when they take either, both 0 or both 1 are verified in Tables 1.50 to 1.53.

Table 1.50: Verification of x_3 for $a_3 = b_3$

$a_3 \ b_3$	$a'_3 \ b'_3$	$a'_3 b_3$	$a_3 b'_3$	$a'_3 b_3 + a_3 b'_3$	$x_3 = (a'_3 b_3 + a_3 b'_3)'$
0 0	1 1	0	0	0	1
0 1	1 0	1	0	1	0
1 0	0 1	0	1	1	0
1 1	0 0	0	0	0	1

Alternatively

$$x_3 = (a_3 \oplus b_3)'$$

$$x_2 = (a_2 \oplus b_2)'$$

$$x_1 = (a_1 \oplus b_1)'$$

$$x_0 = (a_0 \oplus b_0)'$$

Table 1.51: Verification of x_2 for $a_2 = b_2$

$a_2 \ b_2$	$a'_2 \ b'_2$	$a'_2 b_2$	$a_2 b'_2$	$a'_2 b_2 + a_2 b'_2$	$x_2 = (a'_2 b_2 + a_2 b'_2)'$
0 0	1 1	0	0	0	1
0 1	1 0	1	0	1	0
1 0	0 1	0	1	1	0
1 1	0 0	0	0	0	1

Table 1.52: Verification of x_1 for $a_1 = b_1$

$a_1 \ b_1$	$a'_1 \ b'_1$	$a'_1 b_1$	$a_1 b'_1$	$a'_1 b_1 + a_1 b'_1$	$x_1 = (a'_1 b_1 + a_1 b'_1)'$
0 0	1 1	0	0	0	1
0 1	1 0	1	0	1	0
1 0	0 1	0	1	1	0
1 1	0 0	0	0	0	1

Table 1.53: Verification of x_0 for $a_0 = b_0$

$a_0 \ b_0$	$a'_0 \ b'_0$	$a'_0 b_0$	$a_0 b'_0$	$a'_0 b_0 + a_0 b'_0$	$x_0 = (a'_0 b_0 + a_0 b'_0)'$
0 0	1 1	0	0	0	1
0 1	1 0	1	0	1	0
1 0	0 1	0	1	1	0
1 1	0 0	0	0	0	1

The logic circuit to check only equality of two 4-bit binary number is shown in Fig. 1.76.

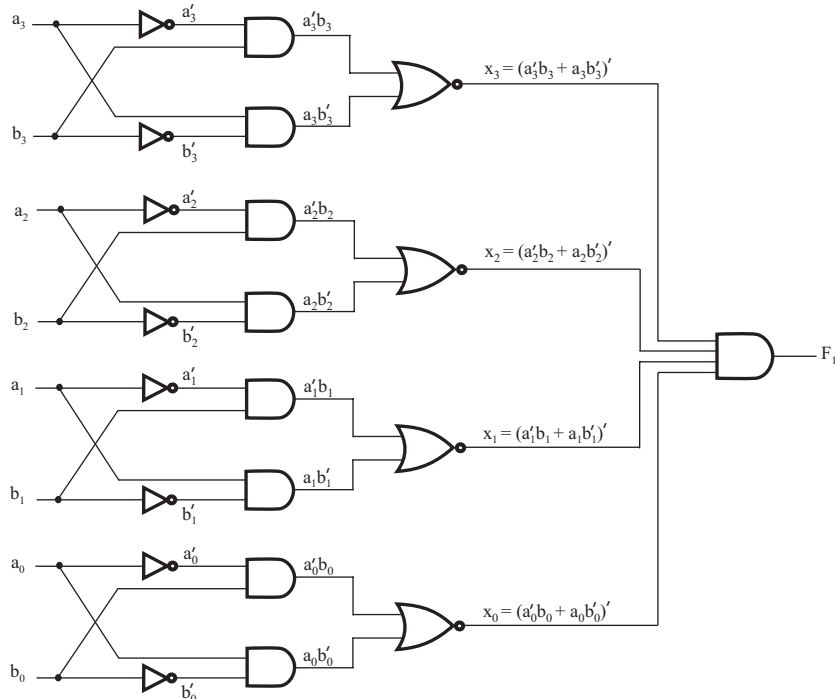


Fig. 1.76: Logic circuit to check equality of two 4-bit binary data.

Checking for $A > B$

In order to determine $A > B$, check bit-by-bit starting from most significant bit. If most significant bits of A and B are not equal and if most significant bit of A is greater than most significant bit of B then $A > B$.

If most significant bits are equal then proceed to next most significant bit and so on. For comparison of 4-digit binary numbers, 4 stages of comparison is required.

Stage-1: Comparing a_3 and b_3

First check for $a_3 > b_3$. From row-3 of Table 1.50, we can say that, if $a_3 b'_3 = 1$ and $x_3 = 0$ then $a_3 > b_3$ and declare $A > B$ and skip other stages of comparisons. Otherwise check for $a_3 = b_3$. From rows-1 and 4 of Table 1.50 we can say that if $a_3 b'_3 = 0$ and $x_3 = 1$ then $a_3 = b_3$ and so proceed to next stage.

<p>If $a_3 b'_3 = 1$ and $x_3 = 0$ then $a_3 > b_3$; Declare $A > B$</p>	<p>If $a_3 b'_3 = 0$ and $x_3 = 1$ then $a_3 = b_3$; Check next most significant bits</p>
---	---

Stage-2: Comparing a_2 and b_2

Check for $a_2 > b_2$. From row-3 of Table 1.51, we can say that, if $a_2 b'_2 = 1$ and $x_2 = 0$ then $a_2 > b_2$ and declare $A > B$ and skip other stages of comparisons. Otherwise check for $a_2 = b_2$. From rows-1 and 4 of Table 1.51 we can say that if $a_2 b'_2 = 0$ and $x_2 = 1$ then $a_2 = b_2$ and so proceed to next stage.

<p>If $a_2 b'_2 = 1$ and $x_2 = 0$ then $a_2 > b_2$; Declare $A > B$</p>	<p>If $a_2 b'_2 = 0$ and $x_2 = 1$ then $a_2 = b_2$; Check next most significant bits</p>
---	---

Stage-3: Comparing a_1 and b_1

Check for $a_1 > b_1$. From row-3 of Table 1.52, we can say that, if $a_1 b'_1 = 1$ and $x_1 = 0$ then $a_1 > b_1$ and declare $A > B$ and skip other stages of comparisons. Otherwise check for $a_1 = b_1$. From rows-1 and 4 of Table 1.52 we can say that if $a_1 b'_1 = 0$ and $x_1 = 1$ then $a_1 = b_1$ and so proceed to next stage.

<p>If $a_1 b'_1 = 1$ and $x_1 = 0$ then $a_1 > b_1$; Declare $A > B$</p>	<p>If $a_1 b'_1 = 0$ and $x_1 = 1$ then $a_1 = b_1$; Check next most significant bits</p>
---	---

Stage-4: Comparing a_0 and b_0

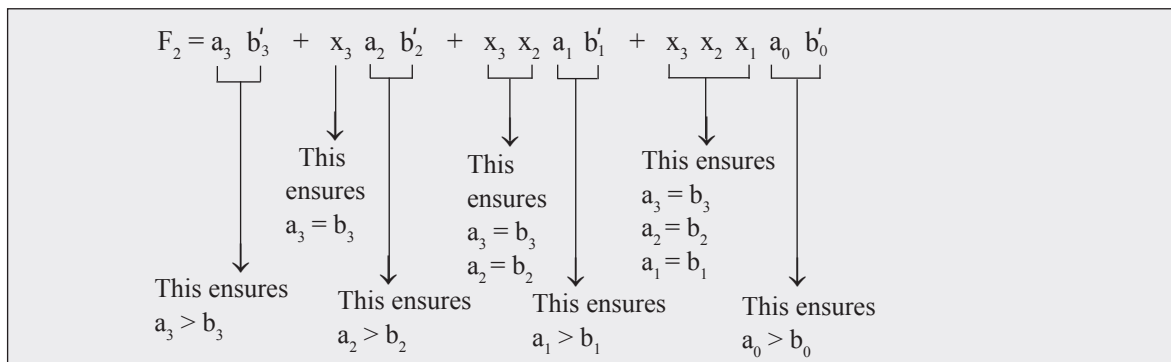
Check for $a_0 > b_0$. From row-3 of Table 1.53, we can say that, if $a_0 b'_0 = 1$ and $x_0 = 0$ then $a_0 > b_0$ and declare $A > B$. Otherwise check for $a_0 = b_0$. From rows-1 and 4 of Table 1.53 we can say that if $a_0 b'_0 = 0$ and $x_0 = 1$ then $a_0 = b_0$ and so declare $A = B$.

If $a_0 b'_0 = 1$ and $x_0 = 0$ then $a_0 > b_0$; Declare $A > B$	If $a_0 b'_0 = 0$ and $x_0 = 1$ then $a_0 = b_0$; Declare $A = B$
--	--

Conclusion

Now the combined Boolean equation for $A > B$ is,

$$F_2 = a_3 b'_3 + x_3 a_2 b'_2 + x_3 x_2 a_1 b'_1 + x_3 x_2 x_1 a_0 b'_0$$



Note: In any stage of comparison if greater than or equality conditions are not satisfied then obviously $A < B$.

Check for $A < B$

In order to determine $A < B$, check bit-by-bit starting from most significant bit. If most significant bits of A and B are not equal and if most significant bit of A is less than most significant bit of B then $A < B$.

If most significant bits are equal then proceed to next most significant bit and so on. For comparison of 4-digit binary numbers, 4 stages of comparison is required.

Stage-1: Comparing a_3 and b_3

First check for $a_3 < b_3$. From row-2 of Table 1.50, we can say that, if $a'_3 b_3 = 1$ and $x_3 = 0$ then $a_3 < b_3$ and declare $A < B$ and skip other stages of comparisons. Otherwise check for $a_3 = b_3$. From rows-1 and 4 of Table 1.50 we can say that if $a'_3 b_3 = 0$ and $x_3 = 1$ then $a_3 = b_3$ and so proceed to next stage.

If $a'_3 b_3 = 1$ and $x_3 = 0$ then $a_3 < b_3$; Declare $A < B$	If $a'_3 b_3 = 0$ and $x_3 = 1$ then $a_3 = b_3$; Check next most significant bits
--	---

Stage-2: Comparing a_2 and b_2

Check for $a_2 < b_2$. From row-2 of Table 1.51, we can say that, if $a'_2 b_2 = 1$ and $x_2 = 0$ then $a_2 < b_2$ and declare $A < B$ and skip other stages of comparisons. Otherwise check for $a_2 = b_2$. From rows-1 and 4 of Table 1.51 we can say that if $a'_2 b_2 = 0$ and $x_2 = 1$ then $a_2 = b_2$ and so proceed to next stage.

If $a'_2 b_2 = 1$ and $x_2 = 0$ then $a_2 < b_2$; Declare $A < B$	If $a'_2 b_2 = 0$ and $x_2 = 1$ then $a_2 = b_2$; Check next most significant bits
--	---

Stage-3: Comparing a_1 and b_1

Check for $a_1 < b_1$. From row-2 of Table 1.52, we can say that, if $a'_1 b_1 = 1$ and $x_1 = 0$ then $a_1 < b_1$ and declare $A < B$ and skip other stages of comparisons. Otherwise check for $a_1 = b_1$. From rows-1 and 4 of Table 1.52 we can say that if $a'_1 b_1 = 0$ and $x_1 = 1$ then $a_1 = b_1$ and so proceed to next stage.

If $a'_1 b_1 = 1$ and $x_1 = 0$ then $a_1 < b_1$; Declare $A < B$	If $a'_1 b_1 = 0$ and $x_1 = 1$ then $a_1 = b_1$; Check next most significant bits
--	---

Stage-4: Comparing a_0 and b_0

Check for $a_0 < b_0$. From row-2 of Table 1.53, we can say that, if $a'_0 b_0 = 1$ and $x_0 = 0$ then $a_0 < b_0$ and declare $A < B$. Otherwise check for $a_0 = b_0$. From rows-1 and 4 of Table 5.45 we can say that if $a'_0 b_0 = 0$ and $x_0 = 1$ then $a_0 = b_0$ and so declare $A = B$.

If $a'_0 b_0 = 1$ and $x_0 = 0$ then $a_0 < b_0$; Declare $A < B$	If $a'_0 b_0 = 0$ and $x_0 = 1$ then $a_0 = b_0$; Declare $A = B$
--	--

Conclusion

Now the combined Boolean equation for $A < B$ is,

$$F_3 = a'_3 b_3 + x_3 a'_2 b_2 + x_3 x_2 a'_1 b_1 + x_3 x_2 x_1 a'_0 b_0$$

$a'_3 b_3$	$x_3 a'_2 b_2$	$x_3 x_2 a'_1 b_1$	$x_3 x_2 x_1 a'_0 b_0$
↓	↓	↓	↓
	This ensures $a_3 = b_3$	This ensures $a_3 = b_3$ $a_2 = b_2$	This ensures $a_3 = b_3$ $a_2 = b_2$ $a_1 = b_1$
This ensures $a_3 < b_3$	This ensures $a_2 < b_2$	This ensures $a_1 < b_1$	This ensures $a_0 < b_0$

Note: In any stage of comparison if less than or equality conditions are not satisfied then obviously $A > B$.

Logic Circuit to Check $A = B$ or $A > B$ or $A < B$

The following three Boolean equations are developed for the comparison of two 4-bit binary numbers.

$$F_1 = x_3 x_2 x_1 x_0 \quad ; \quad F_2 = a_3 b_3' + x_3 a_2 b_2' + x_3 x_2 a_1 b_1' + x_3 x_2 x_1 a_0 b_0'$$

$$F_3 = a_3' b_3 + x_3 a_2' b_2 + x_3 x_2 a_1' b_1 + x_3 x_2 x_1 a_0' b_0$$

$$\text{where, } x_3 = (a_3 \oplus b_3)' = (a_3 b_3 + a_3 b_3')' \quad ; \quad x_2 = (a_2 \oplus b_2)' = (a_2 b_2 + a_2 b_2')'$$

$$x_1 = (a_1 \oplus b_1)' = (a_1 b_1 + a_1 b_1')' \quad ; \quad x_0 = (a_0 \oplus b_0)' = (a_0 b_0 + a_0 b_0')'$$

Using the above equations the logic circuit to compare two 4-bit binary numbers is drawn as shown in Fig. 1.77. The circuit has three outputs, which can be interpreted as follows:

If $F_1 = 1$, then $A = B$; If $F_2 = 1$, then $A > B$; If $F_3 = 1$, then $A < B$

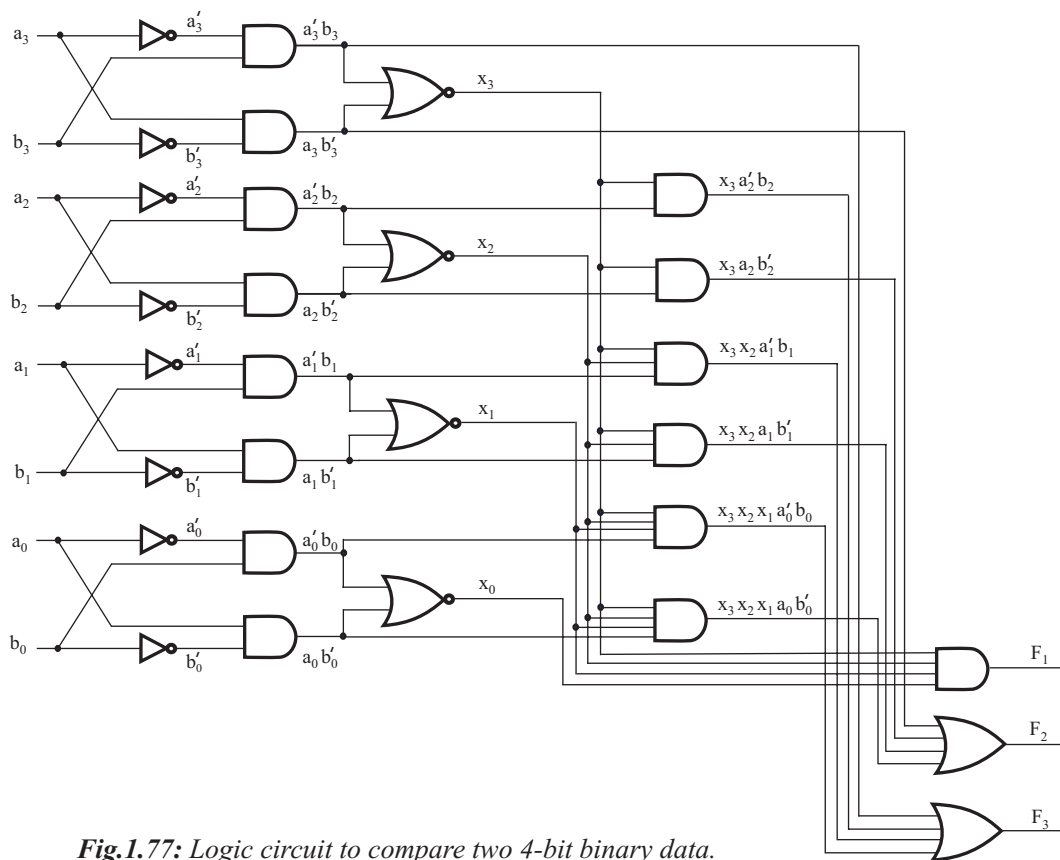


Fig.1.77: Logic circuit to compare two 4-bit binary data.

1.8.2 3-bit Magnitude Comparator

Consider two 3-bit binary numbers A and B.

$$\text{Let, } A = a_2 a_1 a_0 \quad \text{and} \quad B = b_2 b_1 b_0$$

The 3-bit magnitude comparator is similar to 4-bit comparator if the MSB (Most Significant Bit) in the 4-bit comparator is neglected. Therefore, the comparison of a_2 and b_2 , a_1 and b_1 , and a_0 and b_0 are same as that discussed in 4-bit comparator. The reduced logic circuit for 3-bit comparator can be obtained from Fig. 1.77 as shown in Fig. 1.78.

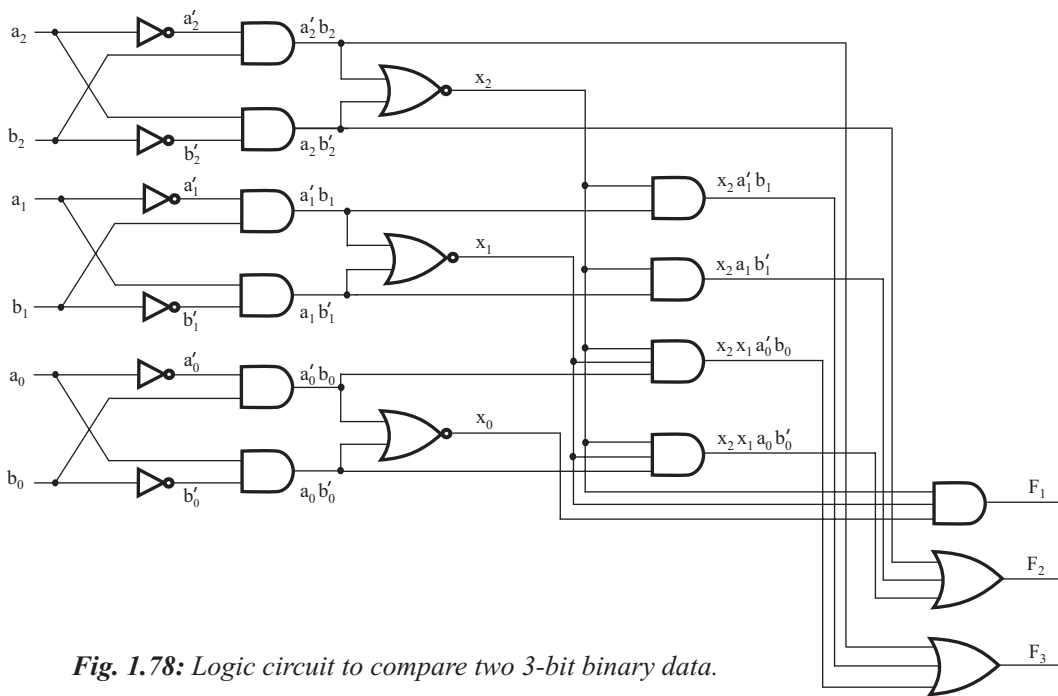


Fig. 1.78: Logic circuit to compare two 3-bit binary data.

1.8.3 2-bit Magnitude Comparator

(AU, Apr/May'23, 2 Marks)

Consider two 2-bit binary numbers A and B.

Let, $A = a_1a_0$ and $B = b_1b_0$

The 2-bit magnitude comparator is similar to 4-bit comparator if upper two bits in the 4-bit comparator is neglected. Therefore, the comparison of a_1 and b_1 , and a_0 and b_0 are same as that discussed in 4-bit comparator. The reduced logic circuit for 2-bit comparator can be obtained from Fig. 1.77 as shown in Fig. 1.79.

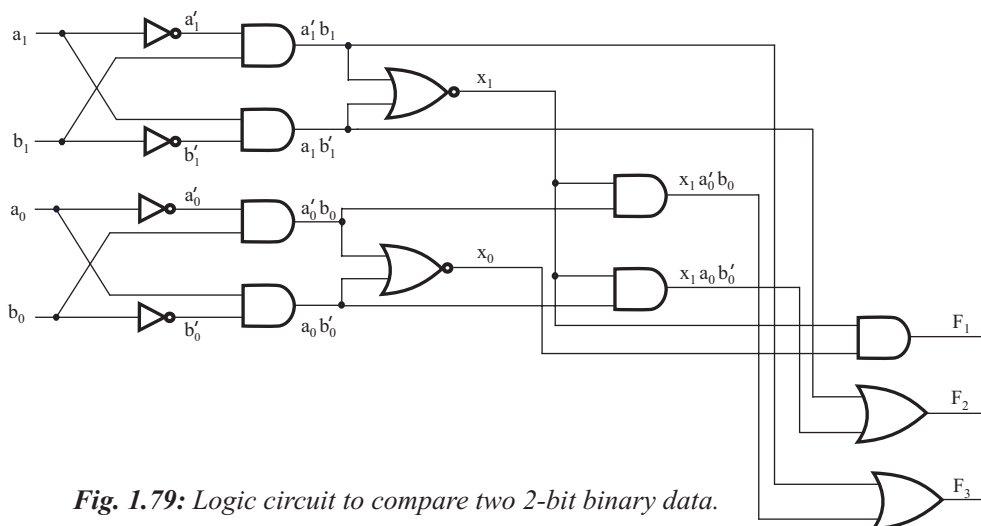


Fig. 1.79: Logic circuit to compare two 2-bit binary data.

1.9 Decoders

A **decoder** is a combinational logic device that decodes one of the 2^n binary information depending on n -bit binary input. The n -bit binary information is decoded into 2^n binary information.

In other words, for an n -bit binary input one of the 2^n output is activated. The activated output may be logic **high** or logic **low**. In active **high** or logic **high** decoder, one of the output is **high** for an n -bit input and all other outputs are **low**. In active **low** or logic **low** decoder, one of the output is **low** for an n -bit input and all other outputs are **high**.

In general, a decoder is referred to as n -to- 2^n decoder and the decoder outputs may be logic **high** or **low**.

For example, consider a 3-to- 2^3 (3-to-8) decoder. Let the 8 outputs of decoder be $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6$ and Y_7 . In a 3-bit binary input, the possible minterms are $m_0, m_1, m_2, m_3, m_4, m_5, m_6$ and m_7 .

In logic **high** 3-to-8 decoder,

When input is m_0 then output Y_0 is active **high**,

When input is m_1 then output Y_1 is active **high**,

and so on.

Therefore, we can say that output decodes the input binary by asserting the output as **high**.

Similarly, in a 3-bit binary input, the possible maxterms are $M_0, M_1, M_2, M_3, M_4, M_5, M_6$ and M_7 .

In logic **low** 3-to-8 decoder,

When input is M_0 then output Y_0 is active **low**,

When input is M_1 then output Y_1 is active **low**,

and so on.

Therefore, we can say that output decodes the input binary by asserting the output as **low**.

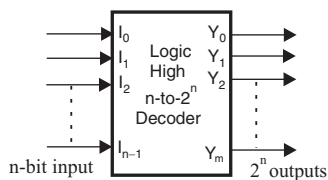


Fig. a: Logic high decoder.

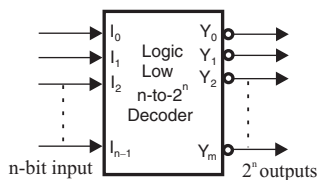


Fig. b: Logic low decoder.

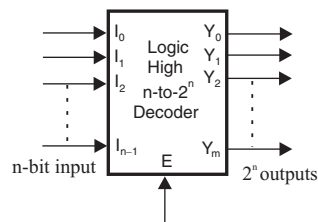


Fig. c: Logic high decoder with high enable.

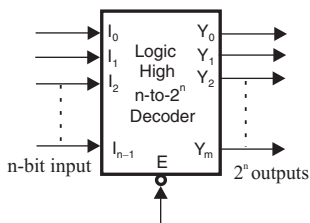


Fig. d: Logic high decoder with low enable.

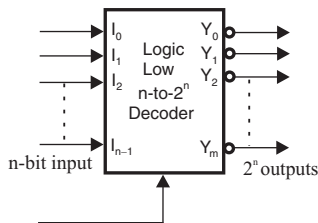


Fig. e: Logic low decoder with high enable.

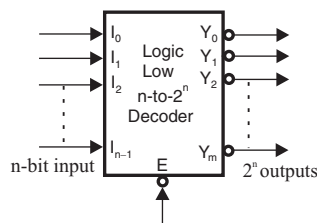


Fig. f: Logic low decoder with low enable.

Fig. 1.80: n -to- 2^n decoder.

Most of the decoders are provided with an enable signal. The input of decoder is recognized only when enable signal is active. The enable can be active **high** or **low**. In decoder with active **high** enable, the input is recognized only when enable is tied to **high**. In decoder with active **low** enable, the input is recognized only when enable is tied to **low**.

Therefore, six types of n -to- 2^n decoders are available as shown in Fig. 1.80. A bubble in the input/output line indicates that the corresponding input/output is active **low**.

1.9.1 Logic High 2-to-4 Decoder

A 2-to-4 (2 -to- 2^2) decoder can generate 4 decoded outputs from the two bit input.

The **logic high 2-to-4 decoder** will generate 4 unique outputs in which only one will be **high** at any one time and all other output will be zero. The block diagram representation of logic **high** 2-to-4 decoder is shown in Fig. 1.81.

Table 1.54: Truth Table of Logic High 2-to-4 Decoder

Inputs		Minterm	Outputs			
I_1	I_0		Y_0	Y_1	Y_2	Y_3
0	0	m_0	1	0	0	0
0	1	m_1	0	1	0	0
1	0	m_2	0	0	1	0
1	1	m_3	0	0	0	1

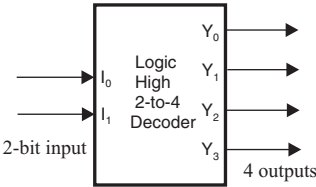


Fig. 1.81: Logic high 2-to-4 decoder.

The truth table of logic **high** 2-to-4 decoder is shown in Table 1.54 and using this truth table, the K-maps for the decoder design are drawn as shown in Fig. 1.82.

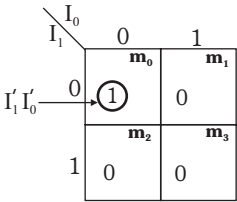


Fig. a: K-map for Y_0 .

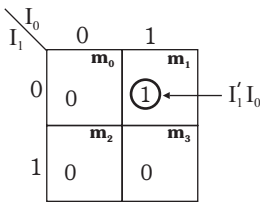


Fig b: K-map for Y_1 .

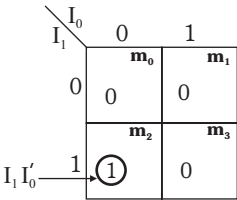


Fig. c: K-map for Y_2 .

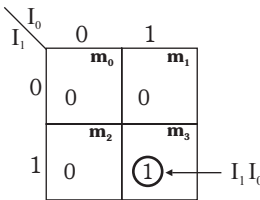


Fig. d: K-map for Y_3 .

Fig. 1.82: K-map for logic high 2-to-4 decoder.

From the K-map for Y_0 , Y_1 , Y_2 and Y_3 the following Boolean equations are obtained and using these equations the logic circuit for logic **high** 2-to-4 decoder is drawn as shown in Fig. 1.83.

$$Y_0 = I_1' I_0'$$

$$Y_1 = I_1' I_0$$

$$Y_2 = I_1 I_0'$$

$$Y_3 = I_1 I_0$$

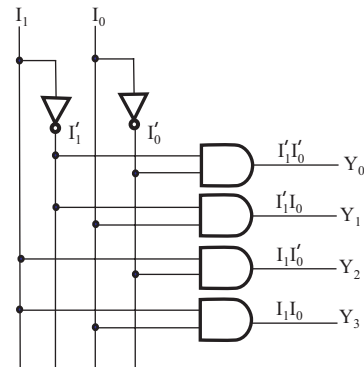


Fig. 1.83: Logic circuit for logic high 2-to-4 decoder.

Note: Alternatively, the output equations of logic **high** 2-to-4 decoder can be directly written from the truth table which are minterms for which output is **high**.

1.9.2 Logic Low 2-to-4 Decoder

The **logic low 2-to-4 decoder** will generate 4 unique outputs in which only one of the output will be **low** at any one time. The output of logic **low** decoder will be complement to that of logic **high** decoder.

The block diagram representation of logic **low** 2-to-4 decoder is shown in Fig. 1.84. The bubble at the outputs indicate that the outputs are active **low**. The dual 2-to-4 decoder is available as a standard IC with number 74139. The pin configuration of 74139 is shown in Fig. 1.85.

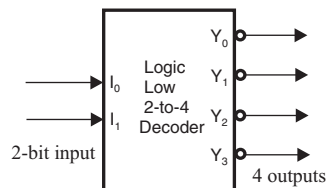


Fig. 1.84: Logic low 2-to-4 decoder.

The truth table of logic **low** 2-to-4 decoder is shown in Table 1.55 and using this truth table, the K-maps for the design of decoder are drawn as shown in Fig. 1.86.

Table 1.55: Truth Table of Logic Low 2-to-4 Decoder

Inputs		Minterm	Outputs			
I_1	I_0		Y_0	Y_1	Y_2	Y_3
0	0	m_0	0	1	1	1
0	1	m_1	1	0	1	1
1	0	m_2	1	1	0	1
1	1	m_3	1	1	1	0

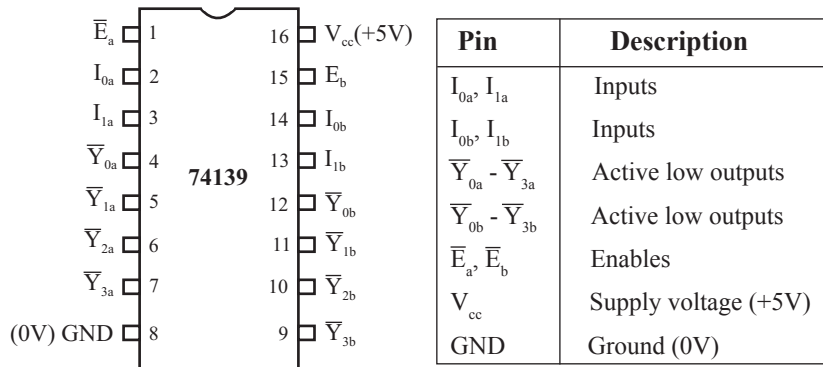


Fig. 1.85: Pin configuration of dual 2-to-4 decoder IC 74139.

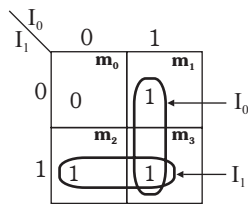


Fig. a: K-map for Y_0 .

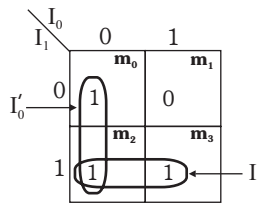


Fig. b: K-map for Y_1 .

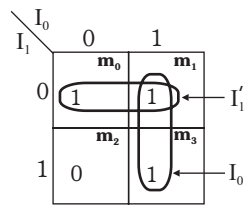


Fig. c: K-map for Y_2 .

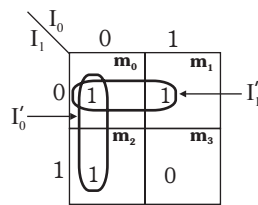


Fig. d: K-map for Y_3 .

Fig. 1.86: K-map for logic low 2-to-4 decoder.

From the K-map for Y_0 , Y_1 , Y_2 and Y_3 the following Boolean equations are obtained and using these equations the logic circuit for logic **low** 2-to-4 decoder is drawn in Fig. 1.87.

$$Y_0 = I_1 + I_0$$

$$Y_1 = I_1 + I'_0$$

$$Y_2 = I'_1 + I_0$$

$$Y_3 = I'_1 + I'_0$$

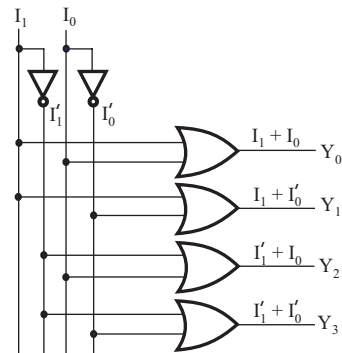


Fig. 1.87: Logic circuit for logic low 2-to-4 decoder.

Note: Alternatively, the output equations of logic **low** 2-to-4 decoder can be directly written from the truth table which are maxterms for which output is **low**.

1.9.3 Logic High 2-to-4 Decoder with Logic Low Enable

The block diagram representation of logic **high** 2-to-4 decoder with logic **low** enable is shown in Fig. 1.88. When enable input is made **high** then inputs are not recognized and so all outputs are zero.

When enable input is made **low** then inputs are recognized and any one of the outputs is set to 1 depending on input.

The truth table of decoder for both active and inactive enable is shown in Table 1.56. Using this truth table, 3-variable K-maps are drawn for 4 outputs of decoder as shown in Fig. 1.89.

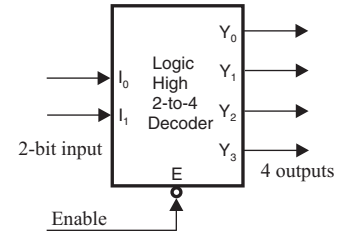


Fig. 1.88: Logic high 2-to-4 decoder with logic low enable.

Table 1.56: Truth Table of Logic High 2-to-4 Decoder with Logic Low Enable

Inputs			Outputs				Comment
E	I ₁	I ₀	Y ₀	Y ₁	Y ₂	Y ₃	
1	x	x	0	0	0	0	Output when enable is active
0	0	0	1	0	0	0	
0	0	1	0	1	0	0	
0	1	0	0	0	1	0	
0	1	1	0	0	0	1	Output when enable is inactive
1	0	0	0	0	0	0	
1	0	1	0	0	0	0	
1	1	0	0	0	0	0	
1	1	1	0	0	0	0	

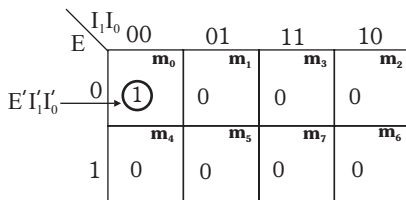


Fig. a: K-map for Y_0 .

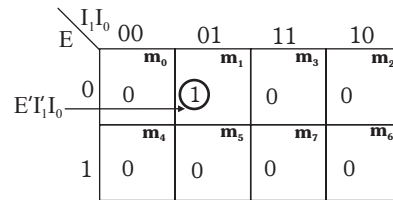


Fig. b: K-map for Y_1 .

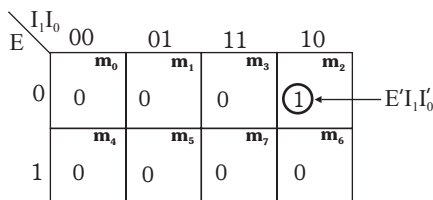


Fig. c: K-map for Y_2 .

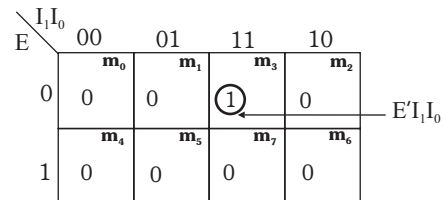


Fig. d: K-map for Y_3 .

Fig. 1.89: K-map for logic high 2-to-4 decoder with logic low enable.

From the K-map for Y_0 , Y_1 , Y_2 and Y_3 the following Boolean equations are obtained and using these equations the logic circuit of decoder is drawn in Fig. 1.91.

$$Y_0 = E' I_1' I_0'$$

$$Y_1 = E' I_1' I_0$$

$$Y_2 = E' I_1 I_0'$$

$$Y_3 = E' I_1 I_0$$

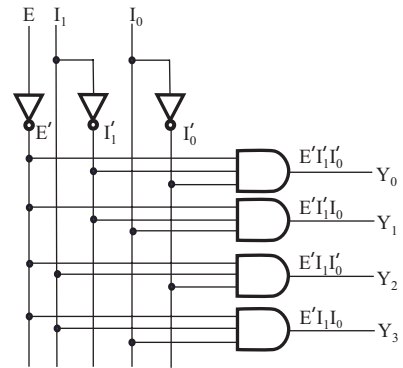


Fig. 1.90: Logic circuit for 2-to-4 logic high decoder with logic low enable.

Using the above equations it is possible to realize the decoder using only NAND gates as shown in Fig. 1.91.

Note: Alternatively, the output equations of logic **high** 2-to-4 decoder can be directly written from the truth table which are minterms for which output is **high**.

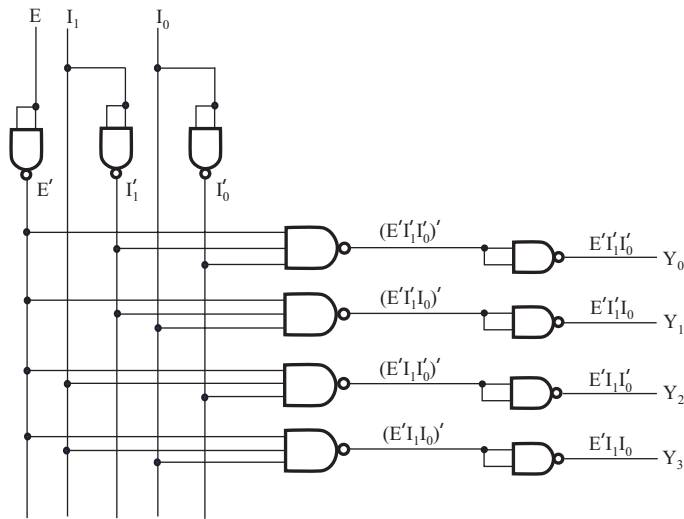


Fig. 1.91: NAND realization of logic high 2-to-4 decoder with logic low enable.

The product terms in the Boolean equations of decoder can be converted to sum terms using DeMorgan's theorem as shown below:

$$Y_0 = E' I_1' I_0'$$

$$Y_1 = E' I_1' I_0$$

$$Y_2 = E' I_1 I_0'$$

$$Y_3 = E' I_1 I_0$$

Using
DeMorgan's
Theorem
→

$$Y_0 = (E + I_1 + I_0)'$$

$$Y_1 = (E + I_1 + I_0')'$$

$$Y_2 = (E + I_1' + I_0)'$$

$$Y_3 = (E + I_1' + I_0')'$$

Using the above equations, the NOR realization of decoder is obtained as shown in Fig. 1.92.

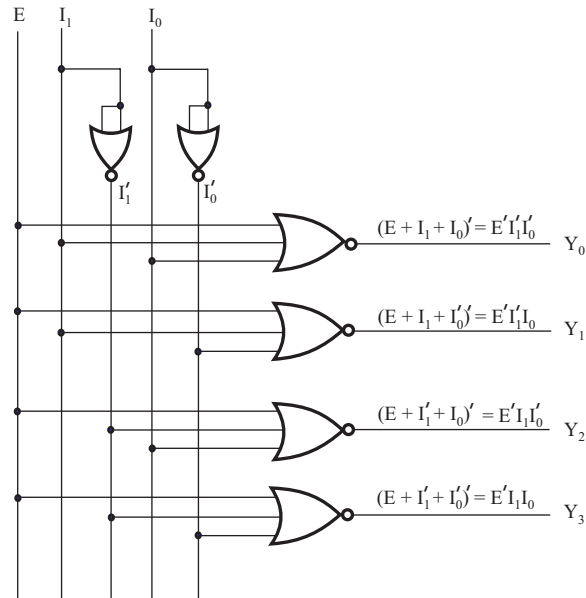


Fig. 1.92: NOR realization of logic high 2-to-4 decoder with logic low enable.

1.9.4 Logic High 3-to-8 Decoder

(AU, Nov/Dec'22, 15 Marks)

A **3-to-8 (3-to-2³) decoder** can generate 8 decoded outputs from the three bit inputs.

The **logic high 3-to-8 decoder** will generate 8 unique outputs in which only one will be **high** at any one time and all other outputs will be **low**.

The block diagram representation of logic **high** 3-to-8 decoder is shown in Fig. 1.93. The truth table of logic **high** 3-to-8 decoder is shown in Table 1.57.

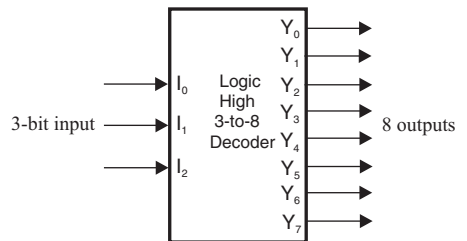


Fig. 1.93: Logic high 3-to-8 decoder.

Table 1.57: Truth Table of Logic High 3-to-8 Decoder

Inputs			Minterm	Outputs							
I ₂	I ₁	I ₀		Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	0	0	m ₀	1	0	0	0	0	0	0	0
0	0	1	m ₁	0	1	0	0	0	0	0	0
0	1	0	m ₂	0	0	1	0	0	0	0	0
0	1	1	m ₃	0	0	0	1	0	0	0	0
1	0	0	m ₄	0	0	0	0	1	0	0	0
1	0	1	m ₅	0	0	0	0	0	1	0	0
1	1	0	m ₆	0	0	0	0	0	0	1	0
1	1	1	m ₇	0	0	0	0	0	0	0	1

From the truth table we can directly write Boolean equations for outputs of logic **high** 3-to-8 decoder which are the minterms for which output is **high** as shown below:

$$\begin{array}{ll}
 Y_0 = I_2' I_1' I_0' & Y_4 = I_2 I_1' I_0' \\
 Y_1 = I_2' I_1' I_0 & Y_5 = I_2 I_1' I_0 \\
 Y_2 = I_2' I_1 I_0' & Y_6 = I_2 I_1 I_0' \\
 Y_3 = I_2' I_1 I_0 & Y_7 = I_2 I_1 I_0
 \end{array}$$

The logic **high** 3-to-8 decoder is available as a standard IC with number 74237. The pin configuration of 74237 is shown in Fig. 1.94.

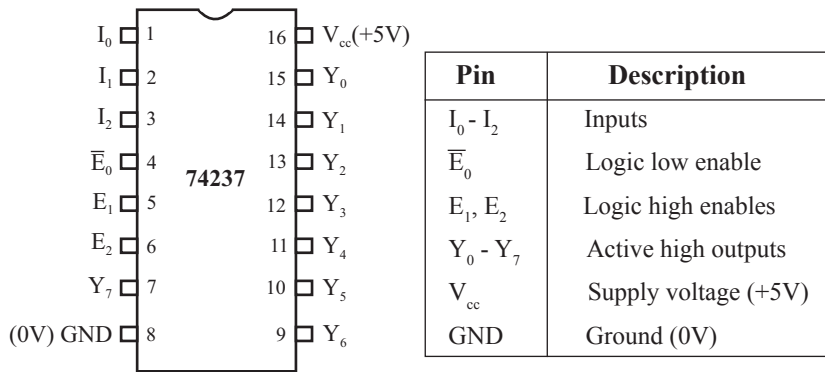


Fig. 1.94: Pin configuration of logic high 3-to-8 decoder IC 74237.

The product terms in the Boolean equations of logic **high** 3-to-8 decoder can be converted to sum terms as shown below:

$$\begin{array}{ll}
 Y_0 = I_2' I_1' I_0' & Y_0 = (I_2 + I_1 + I_0)' \\
 Y_1 = I_2' I_1' I_0 & Y_1 = (I_2 + I_1 + I_0')' \\
 Y_2 = I_2' I_1 I_0' & Y_2 = (I_2 + I_1' + I_0)' \\
 Y_3 = I_2' I_1 I_0 & Y_3 = (I_2 + I_1' + I_0')' \\
 Y_4 = I_2 I_1' I_0' & Y_4 = (I_2' + I_1 + I_0)' \\
 Y_5 = I_2 I_1' I_0 & Y_5 = (I_2' + I_1 + I_0')' \\
 Y_6 = I_2 I_1 I_0' & Y_6 = (I_2' + I_1' + I_0)' \\
 Y_7 = I_2 I_1 I_0 & Y_7 = (I_2' + I_1' + I_0')'
 \end{array}$$

Using DeMorgan's Theorem \longrightarrow

(Used for AND Realization)

(Used for NOR Realization)

Using the above Boolean equations the logic circuit of logic **high** 3-to-8 decoder using AND gates is drawn as shown in Fig. 1.95. It is possible to realize the logic **high** 3-to-8 decoder using only NOR gates as shown in Fig. 1.96.

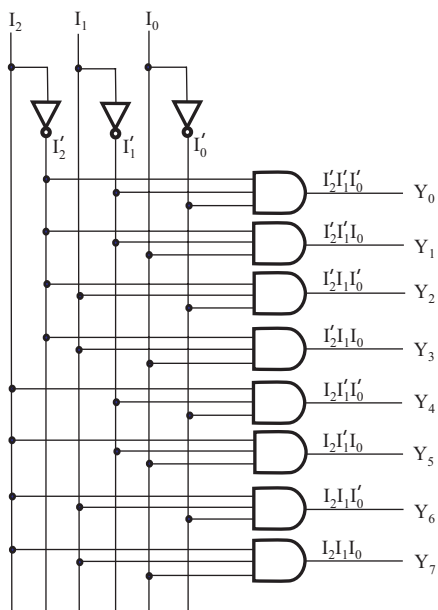


Fig. 1.95: AND realization of logic high of 3-to-8 decoder.

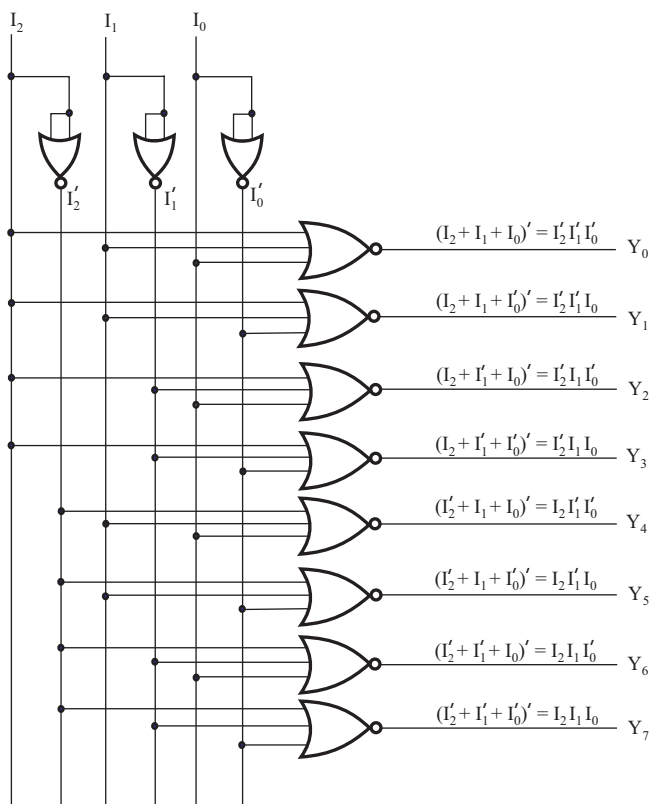


Fig. 1.96: NOR realization of logic high 3-to-8 decoder.

1.9.5 Logic Low 3-to-8 Decoder

The **logic low 3-to-8 decoder** will generate 8 unique outputs in which only one will be **low** at any one time and all other outputs will be **high**.

The block diagram representation of logic **low** 3-to-8 decoder is shown in Fig. 1.97. The truth table of logic **low** 3-to-8 decoder is shown in Table 1.58.

From the truth table we can directly write Boolean equations for outputs of logic **low** 3-to-8 decoder which are the maxterms for which output is **low** as shown below.

$$Y_0 = I_2 + I_1 + I_0$$

$$Y_1 = I_2 + I_1 + I'_0$$

$$Y_2 = I_2 + I'_1 + I_0$$

$$Y_3 = I_2 + I'_1 + I'_0$$

$$Y_4 = I'_2 + I_1 + I_0$$

$$Y_5 = I'_2 + I_1 + I'_0$$

$$Y_6 = I'_2 + I'_1 + I_0$$

$$Y_7 = I'_2 + I'_1 + I'_0$$

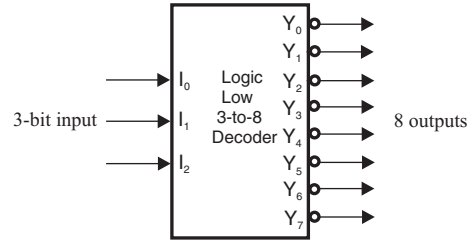


Fig. 1.97: Logic low 3-to-8 decoder.

Table 1.58: Truth Table of Logic Low 3-to-8 Decoder

Inputs			Maxterm	Outputs							
I ₂	I ₁	I ₀		Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	0	0	M ₀	0	1	1	1	1	1	1	1
0	0	1	M ₁	1	0	1	1	1	1	1	1
0	1	0	M ₂	1	1	0	1	1	1	1	1
0	1	1	M ₃	1	1	1	0	1	1	1	1
1	0	0	M ₄	1	1	1	1	0	1	1	1
1	0	1	M ₅	1	1	1	1	1	0	1	1
1	1	0	M ₆	1	1	1	1	1	1	0	1
1	1	1	M ₇	1	1	1	1	1	1	1	0

The logic **low** 3-to-8 decoder is available as a standard IC with number 74138. The pin configuration of 74138 is shown in Fig. 1.98.

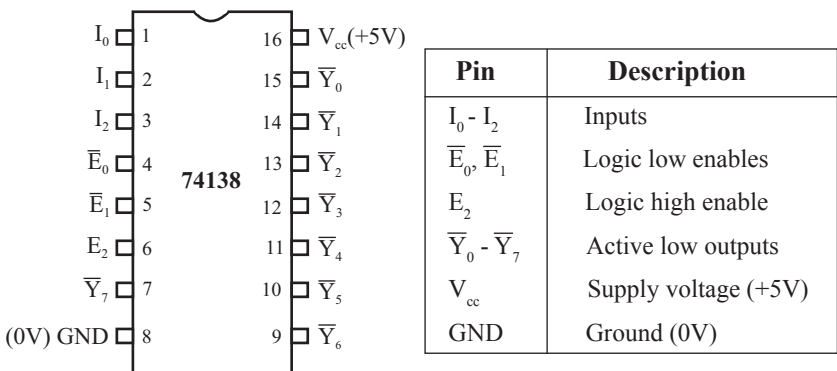


Fig. 1.98: Pin configuration of logic low 3-to-8 decoder IC 74138.

The sum terms in the Boolean equations of 3-to-8 logic **low** decoder can be converted to product terms as shown below:

$$Y_0 = I_2 + I_1 + I_0$$

$$Y_1 = I_2 + I_1 + I'_0$$

$$Y_2 = I_2 + I'_1 + I_0$$

$$Y_3 = I_2 + I'_1 + I'_0$$

$$Y_4 = I'_2 + I_1 + I_0$$

$$Y_5 = I'_2 + I_1 + I'_0$$

$$Y_6 = I'_2 + I'_1 + I_0$$

$$Y_7 = I'_2 + I'_1 + I'_0$$

(Used for OR
Realization)

Using
DeMorgan's
Theorem
→

$$Y_0 = (I'_2 I'_1 I'_0)'$$

$$Y_1 = (I'_2 I'_1 I_0)'$$

$$Y_2 = (I'_2 I_1 I'_0)'$$

$$Y_3 = (I'_2 I_1 I_0)'$$

$$Y_4 = (I_2 I'_1 I'_0)'$$

$$Y_5 = (I_2 I'_1 I_0)'$$

$$Y_6 = (I_2 I_1 I'_0)'$$

$$Y_7 = (I_2 I_1 I_0)'$$

(Used for NAND
Realization)

Using the above Boolean equations the logic circuit of logic **low** 3-to-8 decoder using OR gates is drawn as shown in Fig. 1.99. It is possible to realize the logic **low** 3-to-8 decoder using only NAND gates as shown in Fig. 1.100.

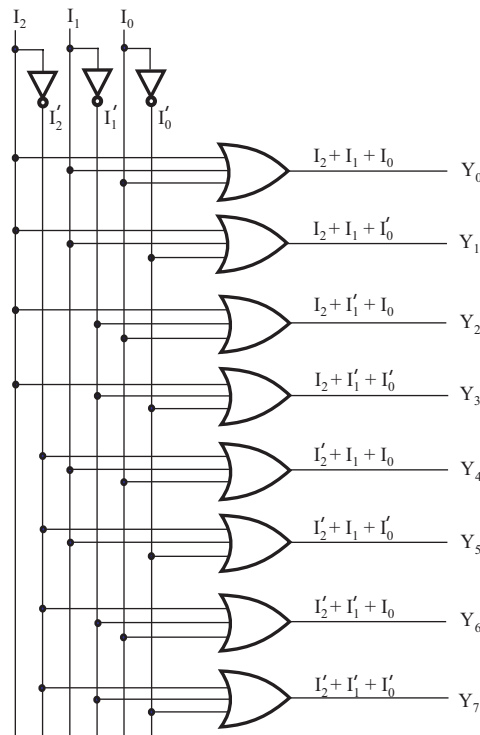


Fig. 1.99: OR realization of logic low 3-to-8 decoder.

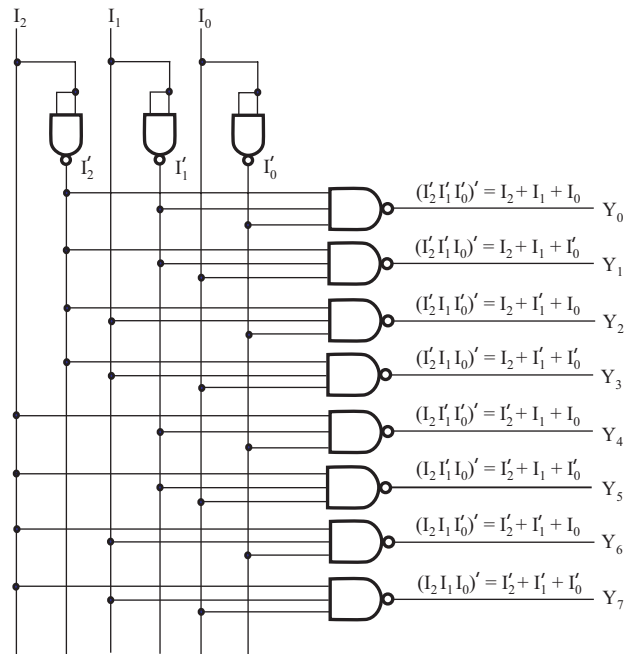


Fig. 1.100: NAND realization of logic low 3-to-8 decoder.

Example 1.46

Design a half adder using logic **high** decoder.

Solution

The half adder has two inputs a , b and two outputs s , c_o (Refer Fig. 1.46). The truth table of half adder along with minterms is shown in Table 1.

From the truth table of half adder we can write the following SOP form of Boolean equations for sum and carry of half adder.

$$\text{Sum, } s = m_1 + m_2$$

$$\text{Carry, } c_o = m_3$$

Table 1: Truth Table of Half Adder

Inputs		Minterm	Outputs	
a	b		s	c_o
0	0	m_0	0	0
0	1	m_1	1	0
1	0	m_2	1	0
1	1	m_3	0	1

Since the half adder has two inputs, a 2-to-4 decoder can be selected to implement the half adder. The truth table of logic **high** 2-to-4 decoder is shown in Table 2.

On taking into account the minterms for which sum and carry are 1, the following Boolean equations can be obtained for sum and carry using the logic **high** decoder output.

$$s = Y_1 + Y_2$$

$$c_o = Y_3$$

Using the above Boolean equations the logic circuit of half adder using logic **high** 2-to-4 decoder and 2-input OR gate is drawn as shown in Fig. 1.

Table 2: Truth Table of Logic High 2-to-4 Decoder

Inputs		Minterm	Decoder Outputs			
a	b		Y_0	Y_1	Y_2	Y_3
0	0	m_0	1	0	0	0
0	1	m_1	0	1	0	0
1	0	m_2	0	0	1	0
1	1	m_3	0	0	0	1

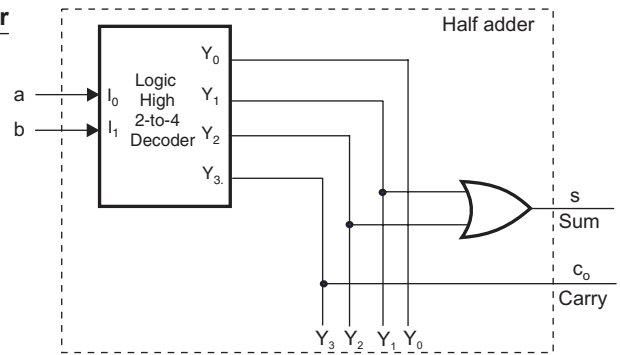


Fig. 1: Half adder using logic high 2-to-4 decoder.

Example 1.47

Design a half adder using a logic **low** decoder and NOR gates.

Solution

The half adder has two inputs a, b and two outputs s, c_0 (Refer Fig. 1.46). The truth table of half adder along with maxterms is shown in Table 1.

From the truth table of half adder we can write the following POS form of Boolean equations for sum and carry of half adder.

$$\text{Sum, } s = M_0 M_3$$

$$\text{Carry, } c_0 = M_0 M_1 M_2$$

Since the half adder has two inputs, a 2-to-4 decoder can be selected to implement the half adder. The truth table of logic **low** 2-to-4 decoder is shown in Table 2.

On taking into account the maxterm for which sum and carry are 0, the following Boolean equations can be obtained for sum and carry using the logic **low** decoder output.

$$s = Y_0 Y_3 = (Y_0 + Y_3)'$$

$$c_0 = Y_0 Y_1 Y_2 = (Y_0 + Y_1 + Y_2)'$$

Using the above Boolean equations the logic circuit of half adder using logic **low** 2-to-4 decoder and NOR gates is drawn as shown in Fig. 1.

Table 1: Truth Table of Half Adder

Inputs		Maxterm	Outputs	
a	b		s	c_0
0	0	M_0	0	0
0	1	M_1	1	0
1	0	M_2	1	0
1	1	M_3	0	1

Table 2: Truth Table of Logic Low 2-to-4 Decoder

Inputs		Maxterm	Decoder Outputs			
a	b		Y_0	Y_1	Y_2	Y_3
0	0	M_0	0	1	1	1
0	1	M_1	1	0	1	1
1	0	M_2	1	1	0	1
1	1	M_3	1	1	1	0

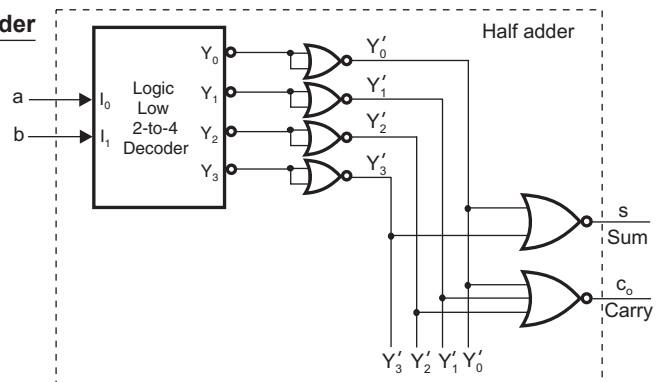


Fig. 1: Half adder using logic low 2-to-4 decoder.

Example 1.48

Design a full adder using logic **high** decoder.

Solution

The full adder has three inputs a , b , c_i and two outputs s , c_o (Refer Fig. 1.53). The truth table of full adder along with minterms is shown in Table 1.

From the truth table of full adder we can write the following SOP form of Boolean equations for sum and carry of full adder.

$$\text{Sum, } s = m_1 + m_2 + m_4 + m_7 ; \quad \text{Carry, } c_o = m_3 + m_5 + m_6 + m_7$$

Since the full adder has three inputs, a 3-to-8 decoder can be selected to implement the full adder. The truth table of logic **high** 3-to-8 decoder is shown in Table 2. On taking into account the minterms for which sum and carry are 1, the following Boolean equations can be obtained for sum and carry using the logic **high** decoder output.

$$s = Y_1 + Y_2 + Y_4 + Y_7 ; \quad c_o = Y_3 + Y_5 + Y_6 + Y_7$$

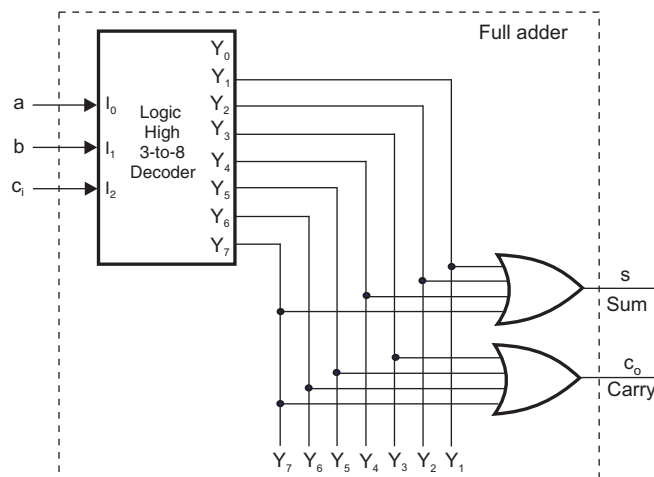
Using the above Boolean equations the logic circuit of full adder using logic **high** 3-to-8 decoder and 4-input OR gate is drawn as shown in Fig. 1.

Table 2: Truth Table of Logic High 3-to-8 Decoder

Inputs			Minterm	Decoder Outputs							
a	b	c_i		Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	m_0	1	0	0	0	0	0	0	0
0	0	1	m_1	0	1	0	0	0	0	0	0
0	1	0	m_2	0	0	1	0	0	0	0	0
0	1	1	m_3	0	0	0	1	0	0	0	0
1	0	0	m_4	0	0	0	0	1	0	0	0
1	0	1	m_5	0	0	0	0	0	1	0	0
1	1	0	m_6	0	0	0	0	0	0	1	0
1	1	1	m_7	0	0	0	0	0	0	0	1

Table 1: Truth Table of Full Adder

Inputs			Minterm	Outputs	
a	b	c_i		Sum s	Carry c_o
0	0	0	m_0	0	0
0	0	1	m_1	1	0
0	1	0	m_2	1	0
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

**Fig. 1: Full adder using logic high 3-to-8 decoder.**

Example 1.49

Design a half subtractor using logic **high** decoder.

Solution

The half subtractor has two inputs a , b and two outputs s , c_o (Refer Fig. 1.67). The truth table of half subtractor adder along with minterms is shown in Table 1.

From the truth table of half subtractor we can write the following SOP form of Boolean equations for difference and borrow of half subtractor.

$$\text{Difference, } d = m_1 + m_2 \quad ; \quad \text{Borrow, } c_o = m_1$$

Since the full subtractor has two inputs, a 2-to-4 decoder can be selected to implement the half subtractor. The truth table of logic **high** 2-to-4 decoder is shown in Table 2.

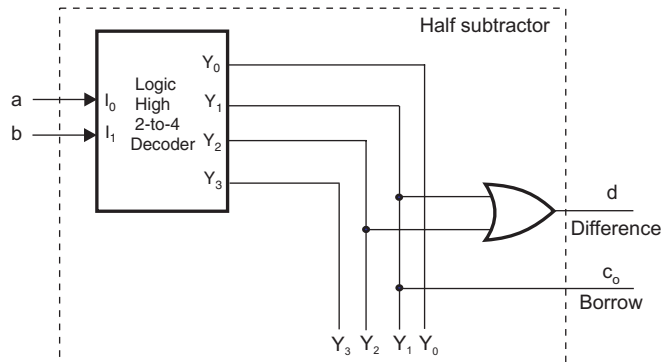
On taking into account the minterms for which difference and borrow are 1, the following Boolean equations can be obtained for difference and borrow using the logic **high** decoder output.

$$d = Y_1 + Y_2 \quad ; \quad c_o = Y_1$$

Using the above Boolean equations the logic circuit of half subtractor using logic **high** 2-to-4 decoder and 2-input OR gate is drawn as shown in Fig. 1.

Table 2: Truth Table of Logic High 2-to-4 Decoder

Inputs		Minterm	Decoder Outputs			
a	b		Y_0	Y_1	Y_2	Y_3
0	0	m_0	1	0	0	0
0	1	m_1	0	1	0	0
1	0	m_2	0	0	1	0
1	1	m_3	0	0	0	1

**Fig. 1:** Half subtractor using logic high 2-to-4 decoder.**Example 1.50**

Design a half subtractor using a logic **low** decoder and NOR gates.

Solution

The half subtractor has two inputs a , b and two outputs d , c_o (Refer Fig. 1.67). The truth table of half subtractor along with maxterms is shown in Table 1.

From the truth table of half subtractor we can write the following POS form of Boolean equations for difference and borrow of half subtractor.

$$\text{Difference, } d = M_0 M_3$$

$$\text{Borrow, } c_o = M_0 M_2 M_3$$

Table 1: Truth Table of Half Subtractor

Inputs		Maxterm	Outputs	
a	b		d	c_o
0	0	M_0	0	0
0	1	M_1	1	1
1	0	M_2	1	0
1	1	M_3	0	0

Since the half subtractor has two inputs, a 2-to-4 decoder can be selected to implement the half subtractor. The truth table of logic **low** 2-to-4 decoder is shown in Table 2.

On taking into account the maxterm for which difference and borrow are 0, the following Boolean equations can be obtained for difference and borrow using the logic **low** decoder output.

$$s = Y_0 Y_3 = (Y_0 + Y_3)' \quad ; \quad c_0 = Y_0 Y_2 Y_3 = (Y_0 + Y_2 + Y_3)'$$

Using the above Boolean equations the logic circuit of half subtractor using logic **low** 2-to-4 decoder and NOR gates is drawn as shown in Fig. 1.

Table 2: Truth Table of Logic Low 2-to-4 Decoder

Inputs		Maxterm	Decoder Outputs			
a	b		Y_0	Y_1	Y_2	Y_3
0	0	M_0	0	1	1	1
0	1	M_1	1	0	1	1
1	0	M_2	1	1	0	1
1	1	M_3	1	1	1	0

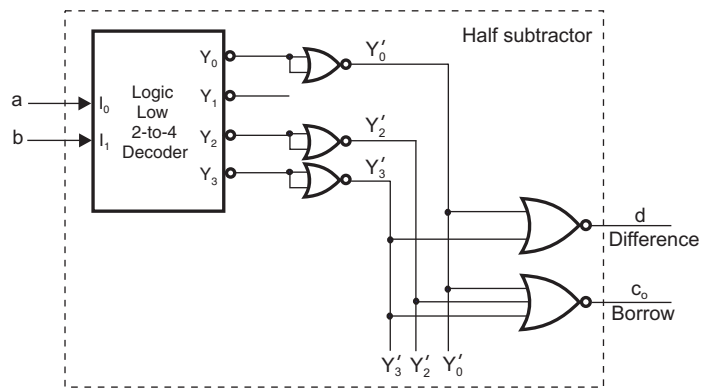


Fig. 1: Half subtractor using logic low 2-to-4 decoder.

1.9.6 Combinational Circuit Design using Decoder

The logic **high** decoder generate output 1 for only one particular minterm as input. Therefore, a Boolean function expressed as a sum of minterms can be obtained by logical OR of the outputs of decoder for which the minterms of function generates logic **high** or 1.

The complement of logic **low** decoder output will be same as output of logic **high** decoder. Hence, the complement of output of logic **low** decoder can be used to realize SOP form of Boolean function.

Similarly, the logic **low** decoder generates output 0 for only one particular maxterm as input. Therefore, a Boolean function expressed as a product of maxterms can be obtained by logical AND of the outputs of decoder for which the maxterms of function generates logic **low** or 0.

The complement of logic **high** decoder output will be same as output of logic **low** decoder. Hence the complement of output of logic **high** decoder can be used to realize POS form of Boolean function.

Example 1.51

Realize the following functions,

$$F_1 = \sum m(0, 1, 3, 6)$$

$$F_2 = \prod M(0, 2, 4, 7)$$

(a) Using logic **high** 3-to-8 decoder and external OR gates only

(b) Using logic **high** 3-to-8 decoder and external NOR gates only.

Solution

Given that, $F_1 = \sum m(0, 1, 3, 6)$; $F_2 = \prod M(0, 2, 4, 7) = \sum m(1, 3, 5, 6)$

Since the given functions has minterms m_0 to m_7 , the given functions are 3-variable functions and so can be realized using a 3-to-8 decoder. The 3-to-8 decoder has three inputs and the minterms are formed from all possible combination of inputs as shown in Table 1. Here, x, y and z are input variables of the functions.

The truth table of F_1 and F_2 is shown in Table 2.

Table 1: Truth Table of Logic High 3-to-8 Decoder

Inputs			Minterm	Outputs							
x	y	z		Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	0	0	m ₀	1	0	0	0	0	0	0	0
0	0	1	m ₁	0	1	0	0	0	0	0	0
0	1	0	m ₂	0	0	1	0	0	0	0	0
0	1	1	m ₃	0	0	0	1	0	0	0	0
1	0	0	m ₄	0	0	0	0	1	0	0	0
1	0	1	m ₅	0	0	0	0	0	1	0	0
1	1	0	m ₆	0	0	0	0	0	0	1	0
1	1	1	m ₇	0	0	0	0	0	0	0	1

Table 2: Truth Table of F_1 and F_2

Minterm	F_1	F_2
$m_0 = 000$	1	0
$m_1 = 001$	1	1
$m_2 = 010$	0	0
$m_3 = 011$	1	1
$m_4 = 100$	0	0
$m_5 = 101$	0	1
$m_6 = 110$	1	1
$m_7 = 111$	0	0

(a) Using Logic High 3-to-8 Decoder and External OR Gates Only

In function F_1 , the output is "1" for minterms 0, 1, 3 and 6. In logic **high** decoder the output Y_n is "1" when the input corresponds to minterm m_n and so the outputs Y_0 , Y_1 , Y_3 and Y_6 are logically ORed for OR implementation.

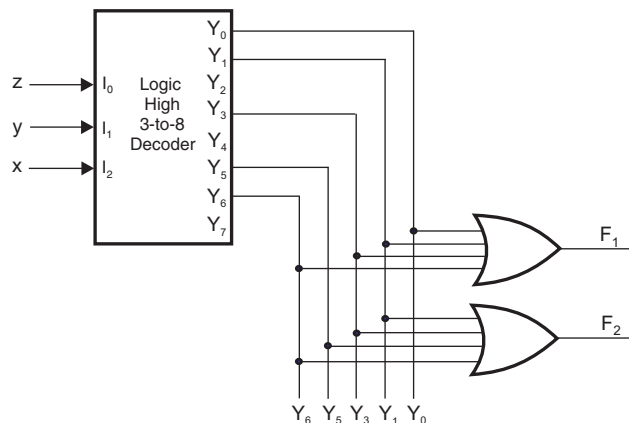
Similarly, for function, F_2 the decoder outputs Y_1 , Y_3 , Y_5 and Y_6 are logically ORed for OR implementation.

The Boolean equations of functions F_1 and F_2 in terms of decoder output are given below:

$$F_1 = Y_0 + Y_1 + Y_3 + Y_6 \quad \text{.....(1)}$$

$$F_2 = Y_1 + Y_3 + Y_5 + Y_6 \quad \text{.....(2)}$$

The equations (1) and (2) are used to implement functions F_1 and F_2 using logic **high** 3-to-8 decoder and external OR gates as shown in Fig. 1.

**Fig. 1: OR implementation of F_1 and F_2 .**

(b) Using Logic High 3-to-8 Decoder and External NOR Gates Only

The equations (1) and (2) can be modified for NOR implementation as shown below:

$$\therefore F_1 = Y_0 + Y_1 + Y_3 + Y_6 = \left((Y_0 + Y_1 + Y_3 + Y_6)' \right)' = (F_1')' \quad \text{.....(3)}$$

$$F_2 = Y_1 + Y_3 + Y_5 + Y_6 = \left((Y_1 + Y_3 + Y_5 + Y_6)' \right)' = (F_2')' \quad \text{.....(4)}$$

The equations (3) and (4) are used to implement functions F_1 and F_2 using logic **high** 3-to-8 decoder and external NOR gates as shown in Fig. 2.

In NOR implementation the inverters are constructed using NOR gates and the OR gate is constructed using NOR gate followed by an inverter as shown in Fig. 2.

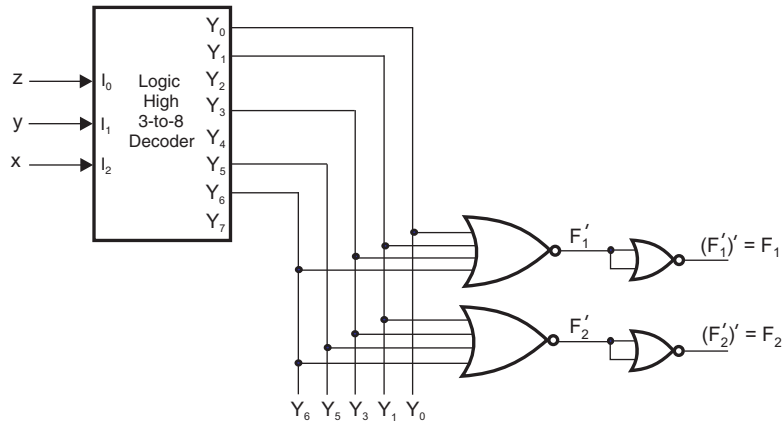


Fig. 2: NOR implementation of F_1 and F_2 .

Example 1.52

Using decoder and external gates, design the combination circuit defined by the following Boolean functions:

i) $F_1 = (y + x')z$ ii) $F_2 = yz + x'y + y'z$ iii) $F_3 = (x + y')z$

Solution

The truth table of Boolean functions F_1 , F_2 and F_3 are shown in Table 1. Here, x , y and z are input variables of the functions.

Table 1: Truth Table for F_1 , F_2 and F_3

x	y	z	Minterm	Maxterm	x'	y'	z'	y + x'	F ₁	yz	x'y	y'z	F ₂	x + y'	F ₃
0	0	0	m ₀	M ₀	1	1	1	1	0	0	0	0	0	1	0
0	0	1	m ₁	M ₁	1	1	0	1	1	0	0	1	1	1	1
0	1	0	m ₂	M ₂	1	0	1	1	0	0	1	0	1	0	0
0	1	1	m ₃	M ₃	1	0	0	1	1	1	1	0	1	0	0
1	0	0	m ₄	M ₄	0	1	1	0	0	0	0	0	0	1	0
1	0	1	m ₅	M ₅	0	1	0	0	0	0	0	1	1	1	1
1	1	0	m ₆	M ₆	0	0	1	1	0	0	0	0	0	1	0
1	1	1	m ₇	M ₇	0	0	0	1	1	1	0	0	1	1	1

From the truth table the following Boolean equations in SOP form are obtained. The function is given by sum of minterms for which output is 1.

$$F_1 = m_1 + m_3 + m_7 \quad ; \quad F_2 = m_1 + m_2 + m_3 + m_5 + m_7 \quad ; \quad F_3 = m_1 + m_5 + m_7$$

On taking into account the minterms for which function outputs are 1 the following Boolean equations are obtained using the logic **high** decoder output.

$$F_1 = Y_1 + Y_3 + Y_7 \quad ; \quad F_2 = Y_1 + Y_2 + Y_3 + Y_5 + Y_7 \quad ; \quad F_3 = Y_1 + Y_5 + Y_7$$

Using the above Boolean equations the logic circuit for the given functions using logic **high** 3-to-8 decoder and OR gate is drawn as shown in Fig. 1.

From the truth table the following Boolean equations in POS form are obtained. The function is given by product of maxterms for which function output is 0.

$$F_1 = M_0 M_2 M_4 M_5 M_6 \quad ; \quad F_2 = M_0 M_4 M_6 \quad ; \quad F_3 = M_0 M_2 M_3 M_4 M_6$$

On taking into account the maxterms for which the functions outputs are 0, the following Boolean equations are obtained using logic **low** decoder output.

$$F_1 = Y_0 Y_2 Y_4 Y_5 Y_6 \quad ; \quad F_2 = Y_0 Y_4 Y_6 \quad ; \quad F_3 = Y_0 Y_2 Y_3 Y_4 Y_6$$

Using the above Boolean equations the logic circuit for the given functions using logic **low** 3-to-8 decoder and AND gate is drawn as shown in Fig. 2.

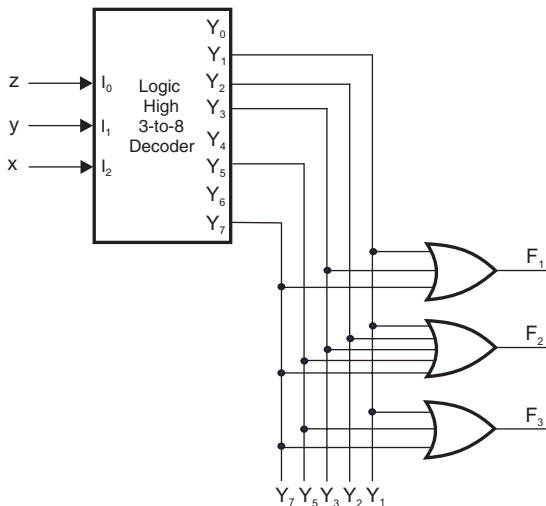


Fig. 1: Logic circuit of F_1, F_2, F_3 using logic high decoder and OR gate.

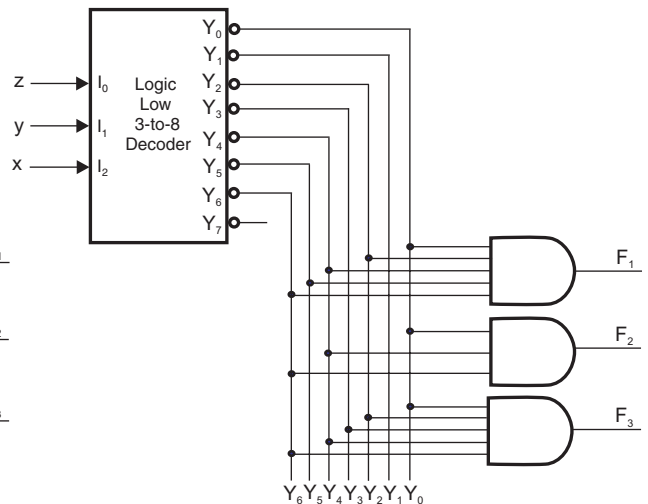


Fig. 2: Logic circuit of F_1, F_2, F_3 using logic low decoder and AND gate.

Example 1.53

Realize the following functions,

$$F_1 = \sum m(0, 1, 3, 6)$$

$$F_2 = \prod M(0, 2, 4, 5)$$

(a) Using logic **low** 3-to-8 decoder and external AND gates only

(b) Using logic **low** 3-to-8 decoder and external NAND gates only.

Solution

$$\text{Given that, } F_1 = \sum m(0, 1, 3, 6) = \prod M(2, 4, 5, 7)$$

$$F_2 = \prod M(0, 2, 4, 5)$$

Since the given function has maxterms M_0 to M_7 the given functions are 3-variable functions and so can be realized using a 3-to-8 decoder. The 3-to-8 decoder has three inputs and the maxterms are formed from all possible combination of inputs as shown in Table 1. Here, x , y and z are input variables of the functions.

The truth table of F_1 and F_2 is shown in Table 2.

Table 1: Truth Table of Logic Low 3-to-8 Decoder

Inputs			Maxterm	Outputs							
x	y	z		Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	0	0	M ₀	0	1	1	1	1	1	1	1
0	0	1	M ₁	1	0	1	1	1	1	1	1
0	1	0	M ₂	1	1	0	1	1	1	1	1
0	1	1	M ₃	1	1	1	0	1	1	1	1
1	0	0	M ₄	1	1	1	1	0	1	1	1
1	0	1	M ₅	1	1	1	1	1	0	1	1
1	1	0	M ₆	1	1	1	1	1	1	0	1
1	1	1	M ₇	1	1	1	1	1	1	1	0

Table 2: Truth Table of F_1 and F_2

Maxterm	F_1	F_2
$M_0 = 0 + 0 + 0$	1	0
$M_1 = 0 + 0 + 1'$	1	1
$M_2 = 0 + 1' + 0$	0	0
$M_3 = 0 + 1' + 1'$	1	1
$M_4 = 1' + 0 + 0$	0	0
$M_5 = 1' + 0 + 1'$	0	0
$M_6 = 1' + 1' + 0$	1	1
$M_7 = 1' + 1' + 1'$	0	1

(a) Using Logic Low 3-to-8 Decoder and External AND Gates Only

In function, F_1 the output is "0" for maxterms 2, 4, 5 and 7. In logic low decoder the output Y_n is "0" when the input corresponds to maxterm M_n and so the outputs Y_2 , Y_4 , Y_5 and Y_7 are logically ANDed in order to realize F_1 .

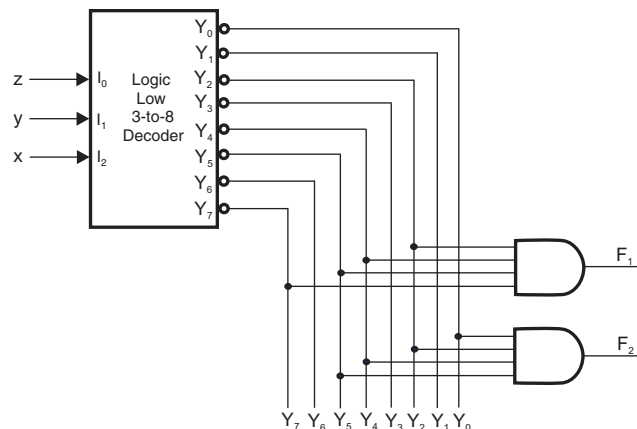
Similarly, for function, F_2 the decoder outputs Y_0 , Y_2 , Y_4 and Y_5 are logically ANDed in order to realize F_2 .

The Boolean equations of functions F_1 and F_2 in terms of logic **low** decoder output are given below:

$$F_1 = \prod M(2, 4, 5, 7) = Y_2 Y_4 Y_5 Y_7 \quad \text{.....(1)}$$

$$F_2 = \prod M(0, 2, 4, 5) = Y_0 Y_2 Y_4 Y_5 \quad \text{.....(2)}$$

The equations (1) and (2) are used to implement functions F_1 and F_2 using logic **low** 3-to-8 decoder and external AND gate as shown in Fig. 1.

**Fig. 1: AND implementation of F_1 and F_2 .**

(b) Using Logic Low 3-to-8 Decoder and External NAND Gates Only

The equations (1) and (2) can be modified for NAND implementation as shown below:

$$F_1 = \prod M(2, 4, 5, 7) = Y_2 Y_4 Y_5 Y_7 = ((Y_2 Y_4 Y_5 Y_7)')' = (F_1')' \quad \text{.....(3)}$$

$$F_2 = \prod M(0, 2, 4, 5) = Y_0 Y_2 Y_4 Y_5 = ((Y_0 Y_2 Y_4 Y_5)')' = (F_2')' \quad \text{.....(4)}$$

The equations (3) and (4) are used to implement functions F_1 and F_2 using logic **low** 3-to-8 decoder and external NAND gate as shown in Fig. 2.

In NAND implementation the inverters are constructed using NAND gates and AND gate is constructed using NAND gate followed by an inverter as shown in Fig. 2.

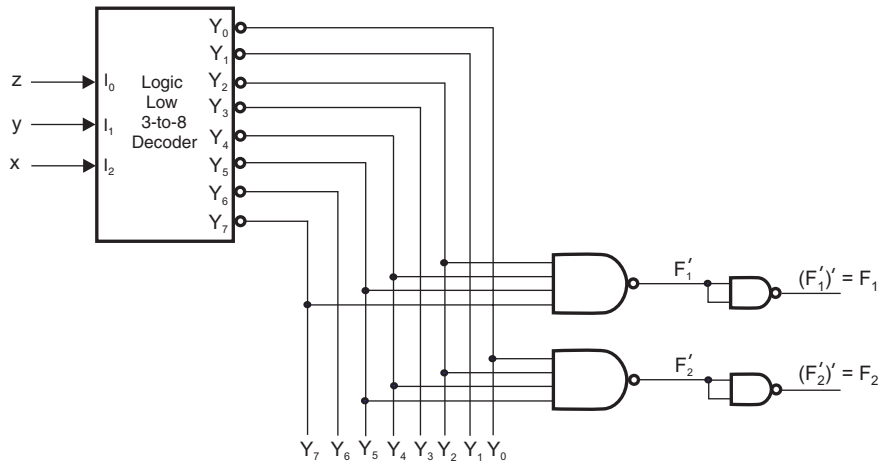


Fig. 2: NAND implementation of F_1 and F_2 .

1.10 Encoders

An **encoder** is a combinational circuit that performs the inverse operation of a decoder.

In general, an encoder has 2^n input lines and n output lines. When one of 2^n inputs is activated, it generates a unique n -bit output. The block diagram of 2^n -to- n encoder is shown in Fig. 1.101.

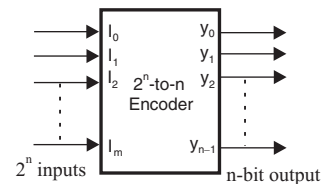


Fig. 1.101: 2^n -to- n encoder.

$$m = 2^n - 1$$

1.10.1 Logic High 4-to-2 Encoder

The **4-to-2 encoder** has 4 inputs and 2 outputs. When any one of the inputs is asserted **high** then a corresponding 2-bit binary value is generated in the output.

Let, I_0, I_1, I_2, I_3 = Inputs

Y_0, Y_1 = Outputs

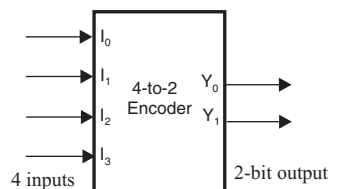


Fig. 1.102: 4-to-2 encoder.

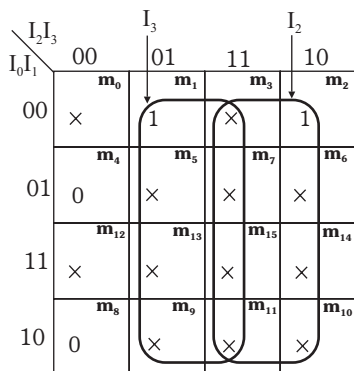
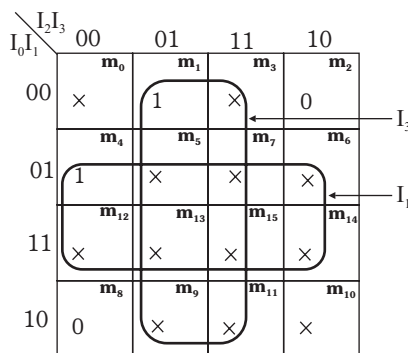
The block diagram of 4-to-2 encoder is shown in Fig. 1.102 and the truth table is shown in Table 1.59. Three different types of encoder design is presented here.

Table 1.59: Truth Table of 4-to-2 Logic High Encoder

Inputs				Minterm	Outputs	
I_0	I_1	I_2	I_3		Y_1	Y_0
1	0	0	0	m_8	0	0
0	1	0	0	m_4	0	1
0	0	1	0	m_2	1	0
0	0	0	1	m_1	1	1

Design 1: 4-to-2 Encoder Design using K-map

Since the 4-to-2 encoder has 4 inputs the possible minterms are $m_0, m_1, m_2, \dots, m_{15}$. But only 4 minterms are required for valid outputs as shown in truth table (Table 1.59). The outputs for remaining minterms are don't-care outputs.

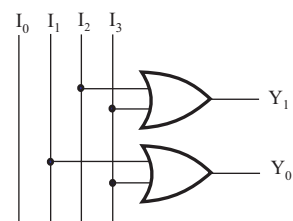
**Fig. a: K-map for Y_1 .****Fig. b: K-map for Y_0 .****Fig. 1.103: 4-to-2 encoder design using K-map.**

Using Table 1.59, the K-maps for Y_0 and Y_1 are constructed as shown in Fig. 1.103. From the K-map we get the following Boolean equations for encoder outputs.

$$Y_1 = I_2 + I_3$$

$$Y_0 = I_1 + I_3$$

Using the above Boolean equations the logic circuit of 4-to-2 encoder is drawn as shown in Fig. 1.104.

**Fig. 1.104: 4-to-2 encoder.**

Note: Since we use minimum literals I_0 is not used in generating output.

Design 2: 4-to-2 Encoder Design using Minterms without Minimization

In this design a SOP form of Boolean equation can be directly formed for each output by considering the minterms for which outputs are 1.

The truth table of 4-to-2 encoder is shown in Table 1.60. In this table the minterms for which outputs are 1 are identified and using these minterms the following SOP form of Boolean equations are obtained for outputs.

$$Y_1 = m_2 + m_1 = I'_0 I'_1 I_2 I'_3 + I'_0 I'_1 I'_2 I_3$$

$$Y_0 = m_4 + m_1 = I'_0 I_1 I_2 I'_3 + I'_0 I'_1 I'_2 I_3$$

Using the above Boolean equations the logic circuit of 4-to-2 encoder is drawn as shown in Fig. 1.105.

Table 1.60: Truth Table of 4-to-2 Encoder

I_0	I_1	I_2	I_3	Minterm	Y_1	Y_0
1	0	0	0	m_8	0	0
0	1	0	0	m_4	0	1
0	0	1	0	m_2	1	0
0	0	0	1	m_1	1	1

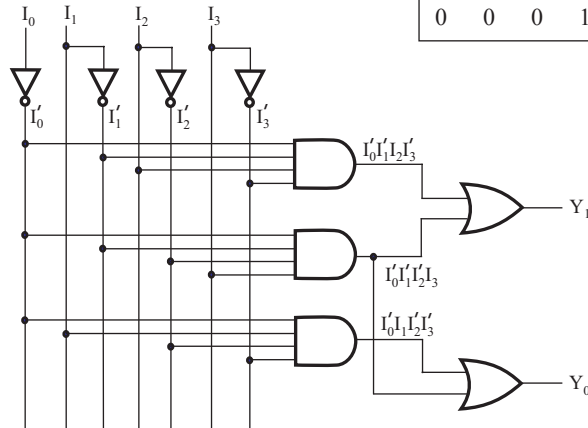


Fig. 1.105: 4-to-2 encoder design using minterms without minimization.

Design 3: 4-to-2 Encoder Design using OR Gate

From the truth table (Table 1.59) we can make following observations.

$$Y_1 \text{ is 1, when } I_2 = 1 \text{ or } I_3 = 1 ; \therefore Y_0 = I_2 + I_3$$

$$Y_0 \text{ is 1, when } I_1 = 1 \text{ or } I_3 = 1 ; \therefore Y_1 = I_1 + I_3$$

Using the above Boolean equations the logic circuit of 4-to-2 encoder is drawn as shown in Fig. 1.106.

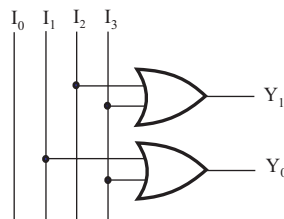


Fig. 1.106: 4-to-2 encoder design using OR gate.

1.10.2 Logic High 8-to-3 Encoder

The **8-to-3 encoder** has eight inputs and 3 outputs. When any one of the inputs is asserted **high** then a corresponding 3-bit binary value is generated in the output.

Let, I_0, I_1, \dots, I_7 = Inputs

Y_0, Y_1, Y_2 = Outputs

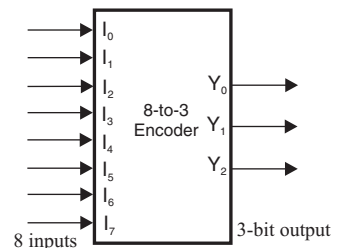


Fig. 1.107: 8-to-3 encoder.

The block diagram of 8-to-3 encoder is shown in Fig. 1.107 and the truth table is shown in Table 1.61. Two different types of 8-to-3 encoder design is presented here.

Since the 8-to-3 encoder has 8 inputs the possible minterms are $m_0, m_1, m_2, \dots, m_{255}$. But only 8 minterms are required for valid outputs as shown in Table 1.61. The outputs for remaining minterms are don't-care outputs.

Table 1.61: Truth Table of 8-to-3 Logic High Encoder

Inputs								Minterms	Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7		Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	m_{128}	0	0	0
0	1	0	0	0	0	0	0	m_{64}	0	0	1
0	0	1	0	0	0	0	0	m_{32}	0	1	0
0	0	0	1	0	0	0	0	m_{16}	0	1	1
0	0	0	0	1	0	0	0	m_8	1	0	0
0	0	0	0	0	1	0	0	m_4	1	0	1
0	0	0	0	0	0	1	0	m_2	1	1	0
0	0	0	0	0	0	0	1	m_1	1	1	1

Design 1: 8-to-3 Encoder Design using OR gates

From the truth table (Table 1.61) we can make following observations.

Y_0 is 1 when $I_1 = 1, I_3 = 1, I_5 = 1, I_7 = 1$; $\therefore Y_0 = I_1 + I_3 + I_5 + I_7$

Y_1 is 1 when $I_2 = 1, I_3 = 1, I_6 = 1, I_7 = 1$; $\therefore Y_1 = I_2 + I_3 + I_6 + I_7$

Y_2 is 1 when $I_4 = 1, I_5 = 1, I_6 = 1, I_7 = 1$; $\therefore Y_2 = I_4 + I_5 + I_6 + I_7$

Using the above Boolean equations the logic circuit of 8-to-3 encoder is drawn using OR gates as shown in Fig. 1.108.

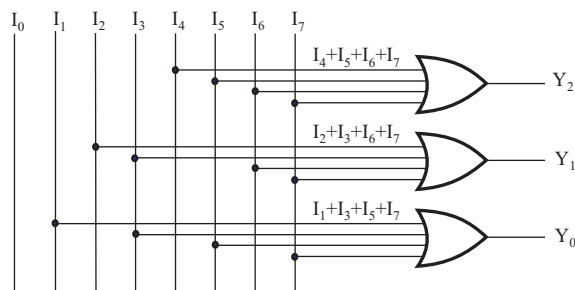


Fig. 1.108: 8-to-3 encoder using OR gates.

1.11 Priority Encoders

(AU, Apr/May'23, 6 Marks)

A **priority encoder** is an encoder circuit that includes a priority function in order to recognize only one input when multiple inputs are asserted **high**.

The operation of the priority encoder is such that if two or more inputs are asserted **high** (or equal to 1) then the output corresponds to the input having highest priority.

1.11.1 Logic High 4-to-2 Priority Encoder

The 4-to-2 encoder has 4 inputs and 2 outputs.

Let, $I_0, I_1, I_2, I_3 =$ Inputs

$Y_0, Y_1 =$ Outputs

Let the order of priority from highest to lowest be,

I_3 - Highest

\vdots

I_2 \vdots

I_1 \vdots

I_0 - Lowest

Now, the above priority works as follows:

If I_3 and I_2 are set **high** then output will correspond to only I_3 .

If I_2 and I_1 are set **high** then output will correspond to only I_2 .

If I_1 and I_0 are set **high** then output will correspond to only I_1 .

To implement the above priority we have to write a priority function.

The priority functions for the above example are,

$$H_3 = I_3 \quad ; \text{ } I_3 \text{ is recognized}$$

$$H_2 = I_2 I_3' \quad ; \text{ } I_2 \text{ is recognized only if } I_3 = 0$$

$$H_1 = I_1 I_2' I_3' \quad ; \text{ } I_1 \text{ is recognized only if } I_2 = I_3 = 0$$

$$H_0 = I_0 I_1' I_2' I_3' \quad ; \text{ } I_0 \text{ is recognized only if } I_1 = I_2 = I_3 = 0$$

The truth table of priority encoder with above priority functions is shown in Table 1.62.

Table 1.62: Truth Table of 4-to-2 Priority Encoder

Inputs				Outputs	
I_0	I_1	I_2	I_3	Y_1	Y_0
1	0	0	0	0	0
×	1	0	0	0	1
×	×	1	0	1	0
×	×	×	1	1	1

From the truth table (Table 1.62) we can note that Y_1 is 1 when I_2 and I_3 are 1.

$$\therefore Y_1 = H_2 + H_3 = I_2 I_3' + I_3$$

From the truth table (Table 1.62) we can note that Y_0 is 1 when I_1 and I_3 are 1.

$$\therefore Y_0 = H_1 + H_3 = I_1 I_2' I_3' + I_3$$

Therefore, the Boolean equations of 4-to-2 priority encoder with priority order from I_3 to I_0 are,

$$Y_1 = I_2 I_3' + I_3$$

$$Y_0 = I_1 I_2' I_3' + I_3$$

The above equations are verified in Table 1.63 for same possible combinations of inputs.

Table 1.63: Verification of 4-to-2 Priority Encoder

I_0	I_1	I_2	I_3	I_2'	I_3'	$I_2 I_3'$	$I_1 I_2' I_3'$	Y_1	Y_0	Comment
1	0	0	0	1	1	0	0	0	0	Only one input is asserted high and so the output corresponds to the input asserted high.
0	1	0	0	1	1	0	1	0	1	
0	0	1	0	0	1	1	0	1	0	
0	0	0	1	1	0	0	0	1	1	
0	0	1	1	0	0	0	0	1	1	2 and 3 asserted high ; output corresponds to 3
0	1	1	0	0	1	1	0	1	0	1 and 2 asserted high ; output corresponds to 2
1	1	0	0	1	1	0	1	0	1	0 and 1 asserted high ; output corresponds to 1

Usually a priority encoder will have a valid bit v , to indicate that one or more input is set **high**. The valid bit is zero if all inputs are zero. The truth table of 4-to-2 priority encoder with valid bit v is shown in Table 1.64. Since, value of valid bit is 1 if any one input is 1, the equation of valid bit is given by sum of all inputs as shown below:

$$\therefore v = I_0 + I_1 + I_2 + I_3$$

Table 1.64: Truth Table of 4-to-2 Priority Encoder with Valid Bit v

Inputs				Outputs		
I_0	I_1	I_2	I_3	Y_1	Y_0	v
0	0	0	0	×	×	0
1	0	0	0	0	0	1
×	1	0	0	0	1	1
×	×	1	0	1	0	1
×	×	×	1	1	1	1

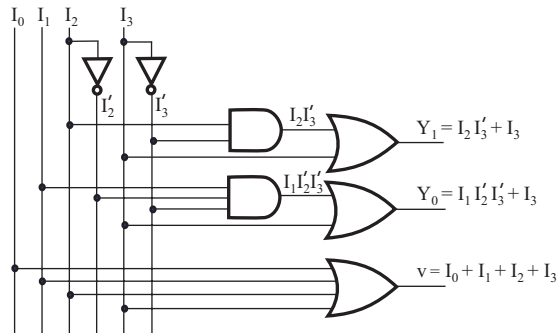


Fig. 1.109: 4-to-2 priority encoder with I_3 having highest priority.

The logic circuit of 4-to-2 priority encoder with priority order I_3 to I_0 is drawn as shown in Fig. 1.109 using the Boolean equations of outputs and valid bit.

Alternatively, the priority encoder can be designed using K-maps.

1.11.2 Logic High 4-to-2 Priority Encoder Design using K-maps

The don't-care inputs are expanded with all possible combination of inputs and listed in Table 1.65.

Table 1.65: Truth Table of Priority Encoder with all Possible Combination of Inputs

Inputs				Minterm	Outputs		
I_0	I_1	I_2	I_3		Y_1	Y_0	v
0	0	0	0	m_0	×	×	0
1	0	0	0	m_8	0	0	1
×	1	0	0	m_4, m_{12}	0	1	1
×	×	1	0	m_2, m_6 m_{10}, m_{14}	1	0	1
×	×	×	1	m_1, m_3 m_5, m_7 m_9, m_{11} m_{13}, m_{15}	1	1	1

Inputs				Minterm	Outputs		
I_0	I_1	I_2	I_3		Y_1	Y_0	v
0	0	0	0	m_0	×	×	0
1	0	0	0	m_8	0	0	1
0	1	0	0	m_4	0	1	1
1	1	0	0	m_{12}	0	1	1
0	0	1	0	m_2	1	0	1
0	1	1	0	m_6	1	0	1
1	0	1	0	m_{10}	1	0	1
1	1	1	0	m_{14}	1	0	1
0	0	0	1	m_1	1	1	1
0	0	1	1	m_3	1	1	1
0	1	0	1	m_5	1	1	1
0	1	1	1	m_7	1	1	1
1	0	0	1	m_9	1	1	1
1	0	1	1	m_{11}	1	1	1
1	1	0	1	m_{13}	1	1	1
1	1	1	1	m_{15}	1	1	1

The K-map for Y_0 , Y_1 and v are constructed using the truth table (Table 1.65) as shown in Fig. 1.110.

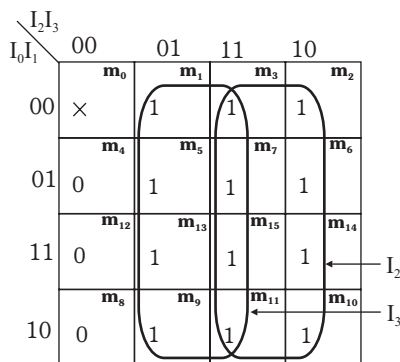


Fig. a: K-map for Y_1 .

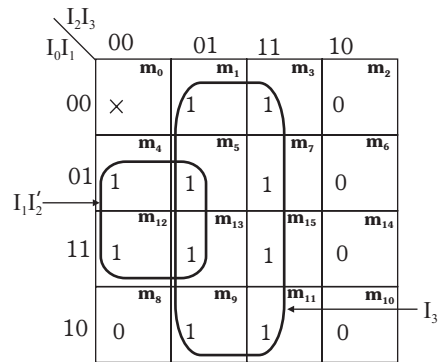


Fig. b: K-map for Y_0 .

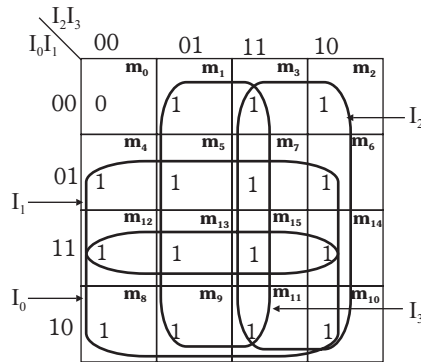


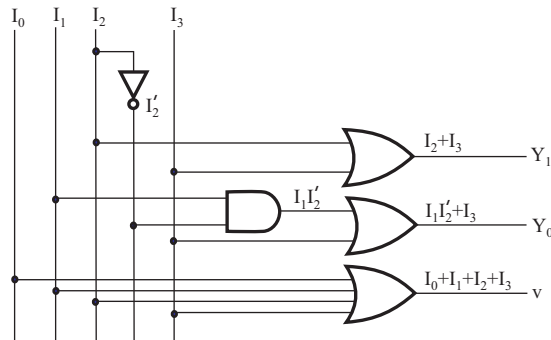
Fig. c: K-map for v.

Fig. 1.110: K-map for design of 4-to-2 encoder.

From the K-map we get the following Boolean equations.

$$Y_1 = I_2 + I_3 \quad ; \quad Y_0 = I_1 I_2' + I_3 \quad ; \quad v = I_0 + I_1 + I_2 + I_3$$

Using the above Boolean equations the logic circuit for 4-to-2 priority encoder is drawn as shown in Fig. 1.111.

Fig. 1.111: Logic circuit for 4-to-2 priority encoder with I_3 having highest priority.

1.11.3 Logic High 8-to-3 Priority Encoder

The 8-to-3 priority encoder has 8 inputs and 3 outputs.

Let, I_0, I_1, \dots, I_7 = Inputs

Y_0, Y_1, Y_2 = Outputs

Let the order of priority be,

I_7 - highest

I_6 - .

I_5 - .

I_4 - .

I_3 - .

I_2 - .

I_1 - .

I_0 - lowest

The priority functions for the above priority order are,

$$\begin{aligned}
 H_7 &= I_7 && ; I_7 \text{ is recognized} \\
 H_6 &= I_6 I_7' && ; I_5 \text{ is recognized only if } I_7 \text{ is } 0 \\
 H_5 &= I_5 I_6' I_7' && ; I_5 \text{ is recognized only if } I_6 \text{ and } I_7 \text{ are } 0 \\
 H_4 &= I_4 I_5' I_6' I_7' && ; I_4 \text{ is recognized only if } I_5, I_6 \text{ and } I_7 \text{ are } 0 \\
 H_3 &= I_3 I_4' I_5' I_6' I_7' && ; I_3 \text{ is recognized only if } I_4, I_5, I_6 \text{ and } I_7 \text{ are } 0 \\
 H_2 &= I_2 I_3' I_4' I_5' I_6' I_7' && ; I_2 \text{ is recognized only if } I_3, I_4, I_5, I_6 \text{ and } I_7 \text{ are } 0 \\
 H_1 &= I_1 I_2' I_3' I_4' I_5' I_6' I_7' && ; I_1 \text{ is recognized only if } I_2, I_3, I_4, I_5, I_6 \text{ and } I_7 \text{ are } 0 \\
 H_0 &= I_0 I_1' I_2' I_3' I_4' I_5' I_6' I_7' && ; I_0 \text{ is recognized only if } I_1, I_2, I_3, I_4, I_5, I_6 \text{ and } I_7 \text{ are } 0
 \end{aligned}$$

The truth table of priority encoder with above priority functions is shown in Table 1.66.

From the truth table we can note that Y_2 is 1 when, I_4, I_5, I_6 and I_7 are 1.

$$\begin{aligned}
 \therefore Y_2 &= H_4 + H_5 + H_6 + H_7 \\
 &= I_4 I_5' I_6' I_7' + I_5 I_6' I_7' + I_6 I_7' + I_7
 \end{aligned}$$

From the truth table we can note that Y_1 is 1 when I_2, I_3, I_6 and I_7 are 1.

$$\begin{aligned}
 \therefore Y_1 &= H_2 + H_3 + H_6 + H_7 \\
 &= I_2 I_3' I_4' I_5' I_6' I_7' + I_3 I_4' I_5' I_6' I_7' + I_6 I_7' + I_7
 \end{aligned}$$

From the truth table we can note that Y_0 is 1 when I_1, I_3, I_5 and I_7 are 1

$$\begin{aligned}
 \therefore Y_0 &= H_1 + H_3 + H_5 + H_7 \\
 &= I_1 I_2' I_3' I_4' I_5' I_6' I_7' + I_3 I_4' I_5' I_6' I_7' + I_5 I_6' I_7' + I_7
 \end{aligned}$$

The valid bit v equation is,

$$v = I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7$$

Using the above Boolean equations the logic circuit for 8-to-3 priority encoder is drawn as shown in Fig. 1.112.

Table 1.66: Truth Table of 8-to-3 Priority Encoder

Inputs								Outputs			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0	v
0	0	0	0	0	0	0	0	×	×	×	0
1	0	0	0	0	0	0	0	0	0	0	1
×	1	0	0	0	0	0	0	0	0	1	1
×	×	1	0	0	0	0	0	0	1	0	1
×	×	×	1	0	0	0	0	0	1	1	1
×	×	×	×	1	0	0	0	1	0	0	1
×	×	×	×	×	1	0	0	1	0	1	1
×	×	×	×	×	×	1	0	1	1	0	1
×	×	×	×	×	×	×	1	1	1	1	1

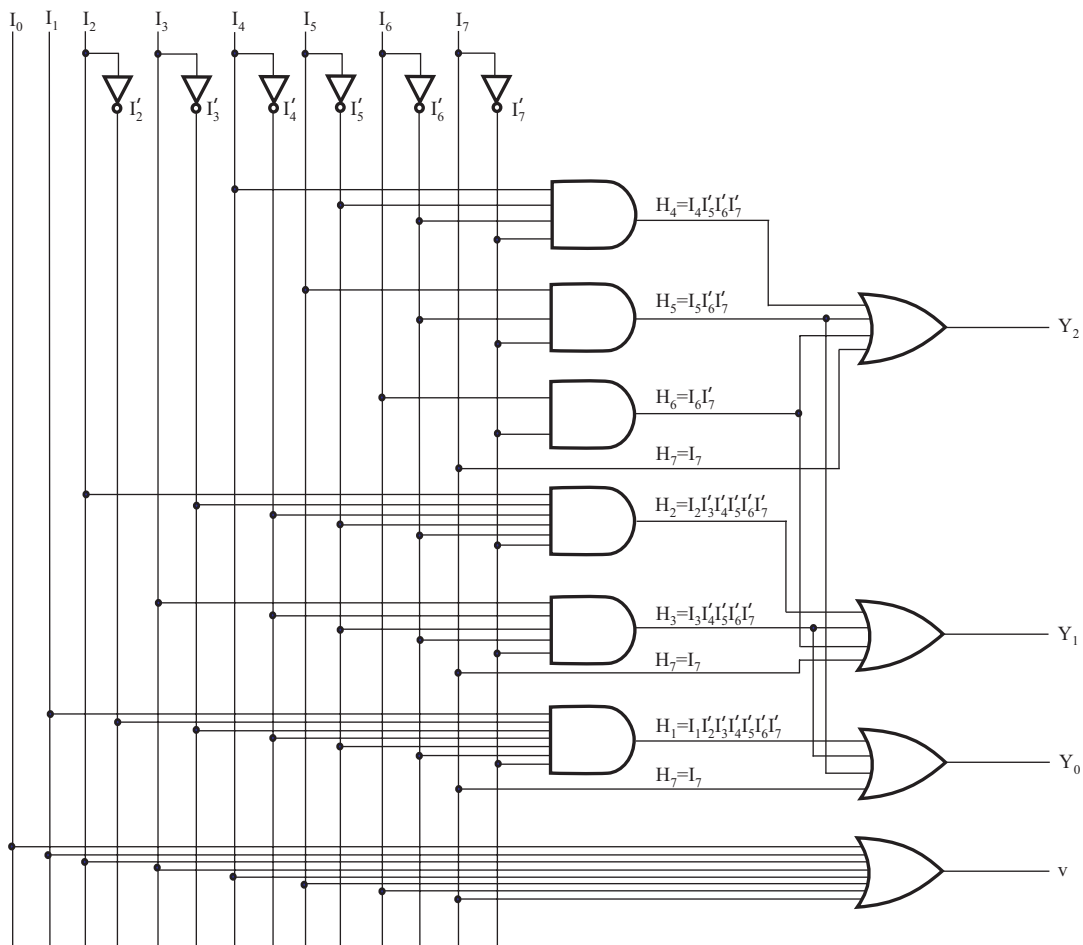


Fig. 1.112: 8-to-3 priority encoder with I_7 having highest priority.

The 8-to-3 priority encoder is available as a standard IC with number 74148. The pin configuration of 74148 is shown in Fig. 1.113.

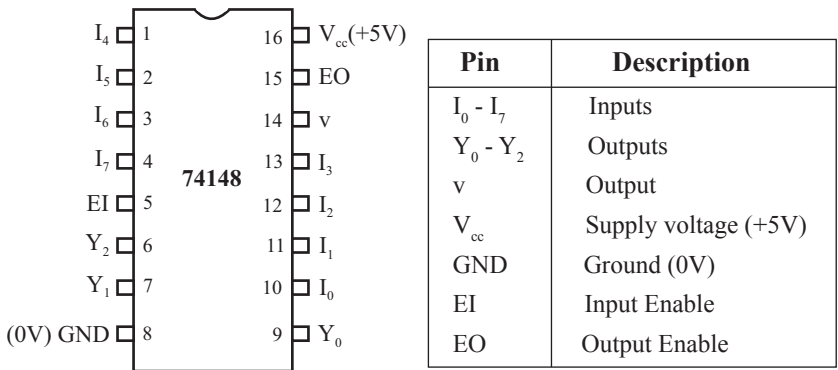


Fig. 1.113: Pin configuration of 8-to-3 priority encoder IC 74148.

Example 1.54

Design a 4-to-2 priority encoder with input I_0 having the highest priority and input I_3 the lowest priority.

Solution

Let, I_0, I_1, I_2, I_3 = Inputs

Y_0, Y_1, v = Outputs

The truth table of priority encoder is shown in Table 1.

Table 1: Truth Table of 4-to-2 Priority Encoder with I_0 as Highest Priority

Inputs				Outputs		
I_0	I_1	I_2	I_3	Y_1	Y_0	v
0	0	0	0	x	x	0
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1

Given that the order of priority highest to lowest are,

I_0 - highest

I_1 -

I_2 -

I_3 - lowest

The priority functions are,

$H_0 = I_0$; I_0 is recognized

$H_1 = I_1 I_0'$; I_1 is recognized only if I_0 is 0

$H_2 = I_2 I_1' I_0'$; I_2 is recognized only if I_1 and I_0 are 0

$H_3 = I_3 I_2' I_1' I_0'$; I_3 is recognized only if I_2, I_1 and I_0 are 0

From the truth table we can note that Y_1 is 1 when, I_2 and I_3 are 1.

$$\therefore Y_1 = H_2 + H_3 = I_2 I_1' I_0' + I_3 I_2' I_1' I_0'$$

From the truth table we can note that Y_0 is 1 when I_1 and I_3 are 1.

$$\begin{aligned} \therefore Y_0 &= H_1 + H_3 \\ &= I_1 I_0' + I_3 I_2' I_1' I_0' \end{aligned}$$

The equation for valid bit, v is,

$$v = I_0 + I_1 + I_2 + I_3$$

The logic circuit of 4-to-2 priority encoder with priority order I_0 and I_3 is drawn as shown in Fig. 1, using Boolean equations of output and valid bit, v .

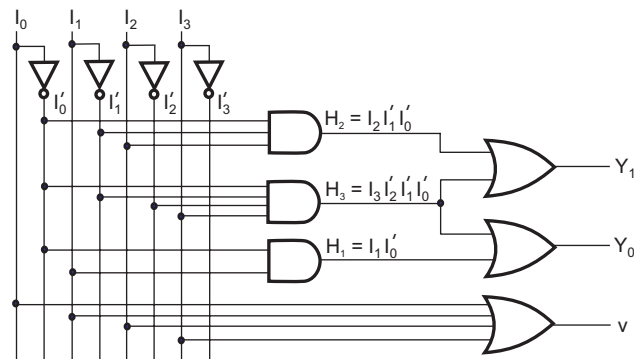


Fig. 1: Logic circuit of 4-to-2 priority encoder with I_0 having highest priority.

The output Boolean equation are verified for some possible combinations of inputs in Table 2.

Table 2: Verification of Output Equations

Inputs				Product Terms										Outputs		Comments	
I ₀	I ₁	I ₂	I ₃	I' ₀	I' ₁	I' ₂	I ₂	I' ₁	I' ₀	I ₃	I' ₂	I' ₁	I' ₀	I ₁	I' ₀		Y ₁
1	0	0	0	0	1	1	0			0				0	0	0	Only one input is asserted high and so the output corresponds to the input asserted high
0	1	0	0	1	0	1	0			0				1	0	1	
0	0	1	0	1	1	0	1			0				0	1	0	
0	0	0	1	1	1	1	0			1				0	1	1	
1	1	0	0	0	0	1	0			0				0	0	0	0 and 1 are asserted high ; output corresponds to 0
0	1	1	0	1	0	0	0			0				1	0	1	1 and 2 are asserted high ; output corresponds to 1
0	0	1	1	1	1	0	1			0				0	1	0	2 and 3 are asserted high ; output corresponds to 2

Example 1.55

(AU, Nov/Dec'23, 13 Marks)

Explain binary to octal decoder and octal to binary encoder with the help of circuit diagram.

Solution

Binary to Octal Decoder

A binary to octal decoder, accept a 3-bit binary input and activates one of the eight outputs corresponding to the octal digit.

The truth table of binary to octal decoder is shown in Table 1 and the Boolean equations for octal outputs are obtained from minterms for which the output is 1.

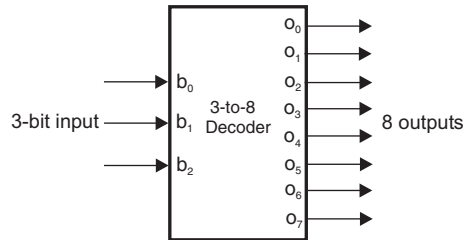


Fig. 1: 3-to-8 decoder.

Table 1: Truth Table

Inputs $b_2 \ b_1 \ b_0$	Minterm	Outputs							
		o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7
0 0 0	m_0	1	0	0	0	0	0	0	0
0 0 1	m_1	0	1	0	0	0	0	0	0
0 1 0	m_2	0	0	1	0	0	0	0	0
0 1 1	m_3	0	0	0	1	0	0	0	0
1 0 0	m_4	0	0	0	0	1	0	0	0
1 0 1	m_5	0	0	0	0	0	1	0	0
1 1 0	m_6	0	0	0	0	0	0	1	0
1 1 1	m_7	0	0	0	0	0	0	0	1

$$o_0 = b_2' b_1' b_0'$$

$$o_1 = b_2' b_1' b_0$$

$$o_2 = b_2' b_1 b_0'$$

$$o_3 = b_2' b_1 b_0$$

$$o_4 = b_2 b_1' b_0'$$

$$o_5 = b_2 b_1' b_0$$

$$o_6 = b_2 b_1 b_0'$$

$$o_7 = b_2 b_1 b_0$$

The logic circuit of binary to octal decoder is shown in Fig. 2.

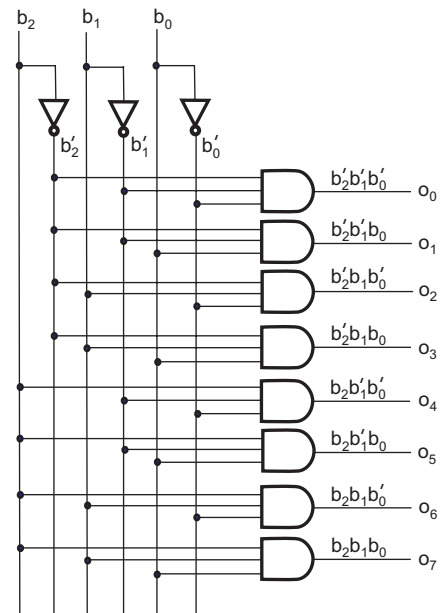


Fig. 2: Logic circuit of binary to octal decoder.

Octal to Binary Encoder

An octal to binary encoder is a digital circuit that converts one of 8 inputs corresponding to 8 octal numbers to its corresponding binary output (3-bit binary number).

The truth table of octal to binary encoder is shown in Table 2 and the Boolean equations for binary outputs are obtained by logical OR of the inputs for which the output is 1.

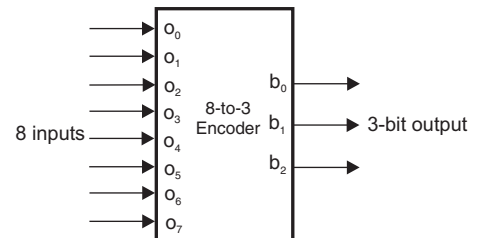


Fig. 3: 8-to-3 encoder.

Table 2: Truth Table

Inputs								Minterms	Outputs		
o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7		b_2	b_1	b_0
1	0	0	0	0	0	0	0	m_{128}	0	0	0
0	1	0	0	0	0	0	0	m_{64}	0	0	1
0	0	1	0	0	0	0	0	m_{32}	0	1	0
0	0	0	1	0	0	0	0	m_{16}	0	1	1
0	0	0	0	1	0	0	0	m_8	1	0	0
0	0	0	0	0	1	0	0	m_4	1	0	1
0	0	0	0	0	0	1	0	m_2	1	1	0
0	0	0	0	0	0	0	1	m_1	1	1	1

$$b_0 = o_4 + o_5 + o_6 + o_7$$

$$b_1 = o_2 + o_3 + o_6 + o_7$$

$$b_2 = o_1 + o_3 + o_5 + o_7$$

The logic circuit of octal to binary encoder is shown in Fig. 4.

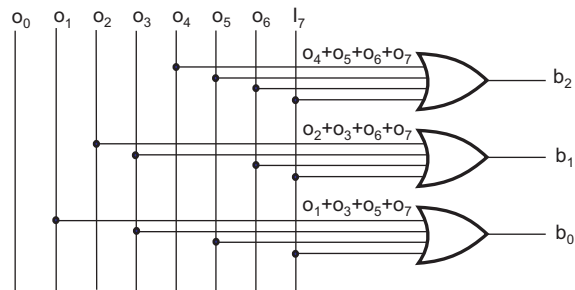


Fig. 4: Logic circuit of octal to binary encoder.

1.12 Multiplexer

(AU, Nov/Dec'22, 2 Marks)

A **multiplexer** is a combinational circuit that selects one binary information from many inputs. For this reason a multiplexer is otherwise called a **data selector**. In general, a multiplexer will have 2^n inputs and one output and so it allows only one input to output at any one time. In order to select one of the 2^n inputs the multiplexer will have n selection lines (or address). In short multiplexer is denoted as MUX.

The block diagram and symbolic representation of $2^n : 1$ multiplexer with n selection lines to select one of the 2^n inputs are shown in Fig. 1.114.

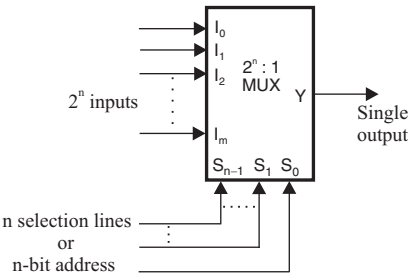


Fig. a: Block diagram representation.

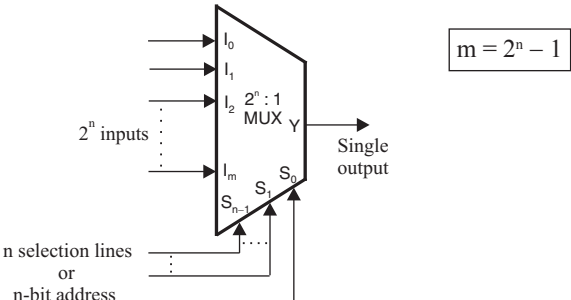


Fig. b: Standard symbol.

Fig. 1.114: Multiplexer.

2:1 Multiplexer

The **2:1 multiplexer** is used to select one of the two inputs using 1-bit address or selection input. Since $2 = 2^1$, for 2:1 multiplexer one selection line is required to select one of the two inputs.

Let, I_0, I_1 = Inputs

S_0 = Selection input/Address

Y = Output

In 2:1 multiplexer,

When, $S_0 = 0$, I_0 is passed to output, $\therefore Y = I_0$

When, $S_0 = 1$, I_1 is passed to output, $\therefore Y = I_1$

The above selection logic are listed as truth table in Table 1.67.

Table 1.67: Truth Table of 2:1 Multiplexer

Selection Input S_0	Output Y
0	I_0
1	I_1

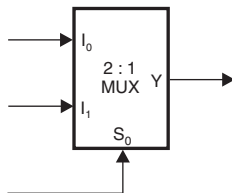


Fig. a: Block diagram representation.

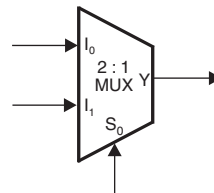


Fig. b: Standard symbol.

Fig. 1.115: 2:1 multiplexer.

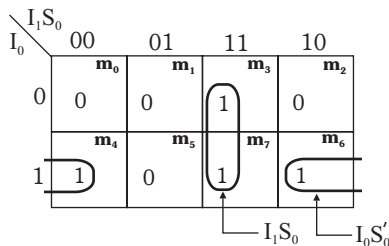


Fig. 1.116: K-map for design of 2:1 multiplexer.

Table 1.68: Truth Table of 2:1 Multiplexer for all Combination of Inputs

Inputs		Selection Input	Minterm	Output Y
I_0	I_1	S_0		
0	0	0	m_0	$Y = I_0 = 0$
0	0	1	m_1	$Y = I_1 = 0$
0	1	0	m_2	$Y = I_0 = 0$
0	1	1	m_3	$Y = I_1 = 1$
1	0	0	m_4	$Y = I_0 = 1$
1	0	1	m_5	$Y = I_1 = 0$
1	1	0	m_6	$Y = I_0 = 1$
1	1	1	m_7	$Y = I_1 = 1$

The truth table of 2:1 multiplexer for all combination of inputs is shown in Table 1.68. The K-map for the design of 2:1 multiplexer is shown in Fig. 1.116 and from the K-map we get the following Boolean equation.

$$Y = I_0 S_0' + I_1 S_0$$

The logic circuit for 2:1 multiplexer is drawn using the above equation as shown in Fig. 1.117.

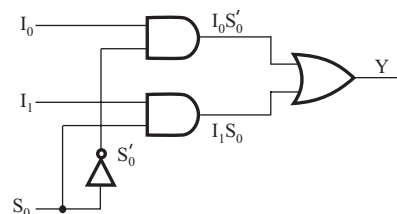


Fig. 1.117: Logic circuit of 2:1 multiplexer.

4:1 Multiplexer

The **4:1 multiplexer** is used to select one of the four inputs using 2-bit address or selection input.

Since $4 = 2^2$, for 4:1 multiplexer two selection lines are required to select one of the four inputs.

- Let, $I_0, I_1, I_2, I_3 =$ Inputs
- $S_0, S_1 =$ Selection inputs/Address
- $Y =$ Output

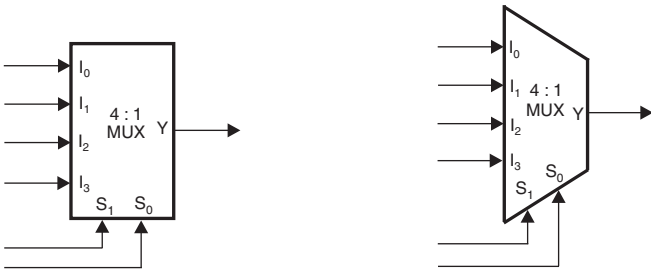
In 4:1 multiplexer,

- When, $S_1 = 0, S_0 = 0, I_0$ is passed to output, $\therefore Y = I_0$
- When, $S_1 = 0, S_0 = 1, I_1$ is passed to output, $\therefore Y = I_1$
- When, $S_1 = 1, S_0 = 0, I_2$ is passed to output, $\therefore Y = I_2$
- When, $S_1 = 1, S_0 = 1, I_3$ is passed to output, $\therefore Y = I_3$

The above selection logic are listed as truth table in Table 1.69.

The block diagram and symbolic representation of 4:1 multiplexer with two selection lines to select one of the four inputs are shown in Fig. 1.118.

Table 1.69: Truth Table of 4:1 Multiplexer



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

Fig. a: Block diagram representation. Fig. b: Standard symbol.

Fig. 1.118: 4:1 multiplexer.

The **dual 4:1 multiplexer** is available as a standard IC with number 7453. The pin configuration of 7453 is shown in Fig. 1.119.

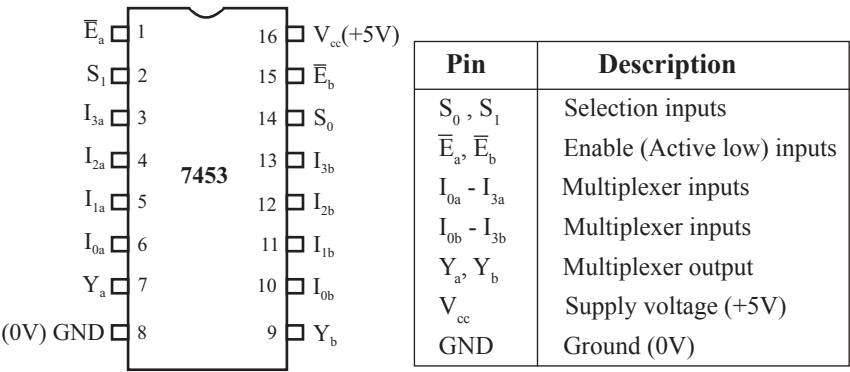


Fig. 1.119: Pin configuration of dual 4:1 multiplexer IC 7453.

The logic circuit of 4:1 multiplexer can be realized using AND and OR gates as shown in Fig. 1.120. Table 1.70 can be used to check the logic circuit of 4:1 multiplexer.

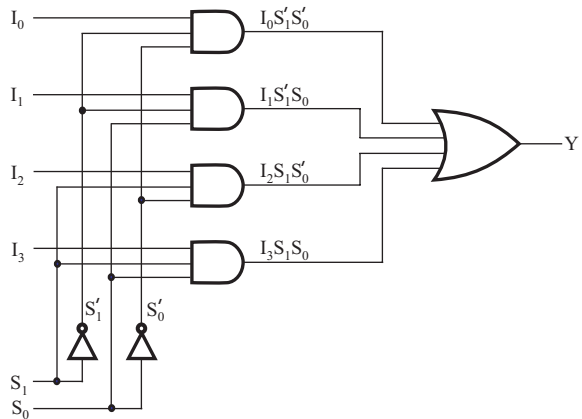


Fig. 1.120: Logic circuit of 4:1 multiplexer.

Table 1.70: Verification of Output of 4:1 MUX shown in Fig. 1.120

S_1	S_0	S'_1	S'_0	$I_0 S'_1 S'_0$	$I_1 S'_1 S_0$	$I_2 S_1 S'_0$	$I_3 S_1 S_0$	Y
0	0	1	1	I_0	0	0	0	I_0
0	1	1	0	0	I_1	0	0	I_1
1	0	0	1	0	0	I_2	0	I_2
1	1	0	0	0	0	0	I_3	I_3

8:1 Multiplexer

The **8:1 multiplexer** is used to select one of the eight inputs using 3-bit address or selection input.

Since $8 = 2^3$, for 8:1 multiplexer 3 selection lines are required to select one of the eight inputs. The input selection logic are listed in truth table shown in Table 1.71. The block diagram and symbolic representation of 8:1 multiplexer with three selection lines to select one of the eight inputs are shown in Fig. 1.121.

Let, $I_0, I_1, I_2, \dots, I_7$ = Inputs
 S_0, S_1, S_2 = Selection inputs/Address
Y = Output

Table 1.71: Truth Table of 8:1 Multiplexer

Selection Inputs			Output Y
S_2	S_1	S_0	
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

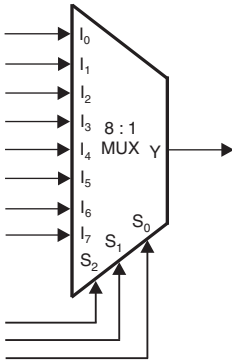
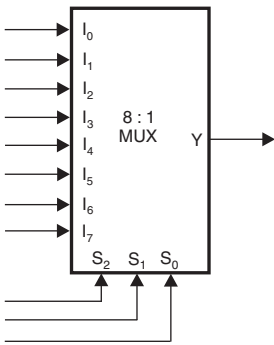


Fig. a: Block diagram representation. Fig. b: Standard symbol.

Fig. 1.121: 8:1 multiplexer.

The logic circuit of 8:1 multiplexer can be realized using AND and OR gates as shown in Fig. 1.122. Table 1.72 can be used to verify the logic circuit of 8:1 multiplexer.

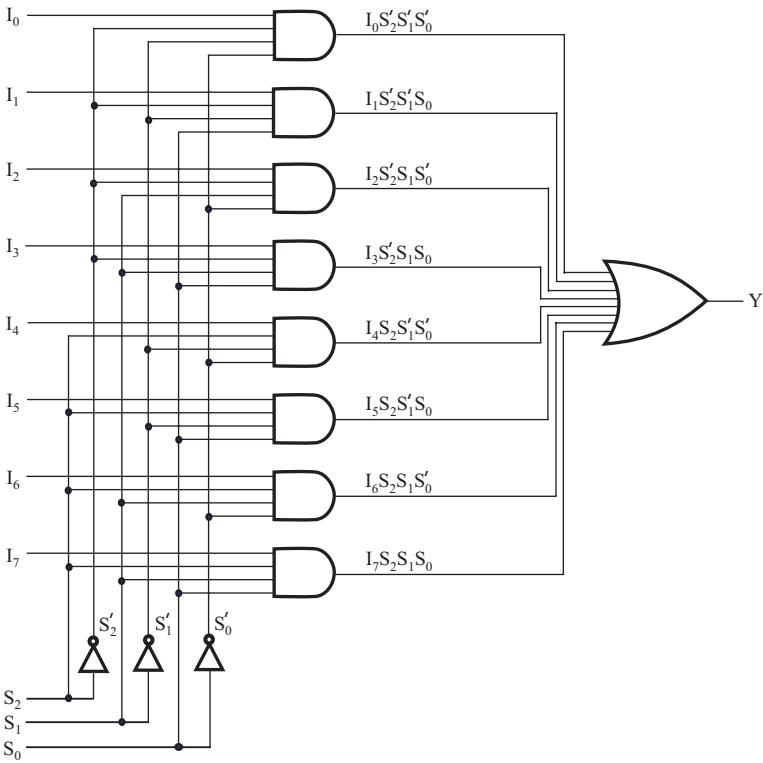


Fig. 1.122: Logic circuit of 8:1 multiplexer.

Table 1.72: Verification of Output of 8:1 Multiplexer of Fig. 1.122

Selection Inputs $S_2 S_1 S_0$	Complement of Selection Inputs $S'_2 S'_1 S'_0$	Product Terms								Output Y
		$I_0 S'_2 S'_1 S'_0$	$I_1 S'_2 S'_1 S_0$	$I_2 S'_2 S_1 S'_0$	$I_3 S'_2 S_1 S_0$	$I_4 S_2 S'_1 S'_0$	$I_5 S_2 S'_1 S_0$	$I_6 S_2 S_1 S'_0$	$I_7 S_2 S_1 S_0$	
0 0 0	1 1 1	I_0	0	0	0	0	0	0	0	I_0
0 0 1	1 1 0	0	I_1	0	0	0	0	0	0	I_1
0 1 0	1 0 1	0	0	I_2	0	0	0	0	0	I_2
0 1 1	1 0 0	0	0	0	I_3	0	0	0	0	I_3
1 0 0	0 1 1	0	0	0	0	I_4	0	0	0	I_4
1 0 1	0 1 0	0	0	0	0	0	I_5	0	0	I_5
1 1 0	0 0 1	0	0	0	0	0	0	I_6	0	I_6
1 1 1	0 0 0	0	0	0	0	0	0	0	I_7	I_7

The 8:1 multiplexer is available as a standard IC with number 74151. The pin configuration of 74151 is shown in Fig. 1.123.

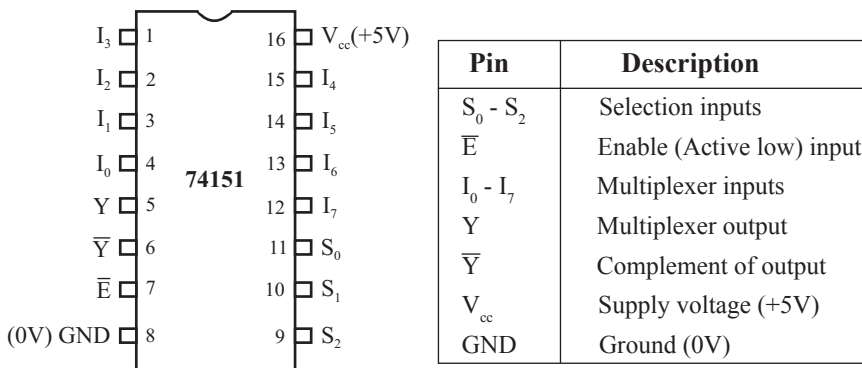


Fig. 1.123: Pin configuration of 8:1 multiplexer IC 74151.

Example 1.56

Design a full adder using multiplexer.

- a) Use 4:1 MUX b) Use 8:1 MUX

Solution

a) Full Adder using 4:1 MUX

The full adder has three inputs a , b , c_i and two outputs s , c_o (Refer Fig. 1.53). The truth table of full adder along with minterms is shown in Table 1.

The full adder has two outputs and hence two separate multiplexers are needed for outputs. Since the full adder has 3-bit input, let us select $2^2:1$ (4:1) multiplexers. The full adder using 4:1 multiplexer is shown in Fig. 1. The inputs a and b of full adder are connected to selection inputs of both the multiplexers. The input c_i of full adder and its complement are used as some of the inputs of multiplexer. The other inputs of multiplexers are permanently tied to 0 or 1. The working of full adder using 4:1 multiplexer is verified in Table 2.

Table 1: Truth Table of Full Adder

Inputs			Minterm	Outputs	
a	b	c_i		Sum s	Carry c_o
0	0	0	m_0	0	0
0	0	1	m_1	1	0
0	1	0	m_2	1	0
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

Table 2: Verification of Logic Circuit of s , c_o in Fig. 1

Inputs			Selection Inputs		Full adder Outputs		MUX Outputs	
a	b	c_i	S_1	S_0	s	c_o	$Y_1 = s$	$Y_2 = c_o$
0	0	0	0	0	0	0	$Y_1 = I_{01} = c_i$	$Y_2 = I_{02} = 0$
0	0	1	0	0	1	0		
0	1	0	0	1	1	0	$Y_1 = I_{11} = c_i'$	$Y_2 = I_{12} = c_i$
0	1	1	0	1	0	1		
1	0	0	1	0	1	0	$Y_1 = I_{21} = c_i'$	$Y_2 = I_{22} = c_i$
1	0	1	1	0	0	1		
1	1	0	1	1	0	1	$Y_1 = I_{31} = c_i$	$Y_3 = I_{32} = 1$
1	1	1	1	1	1	1		

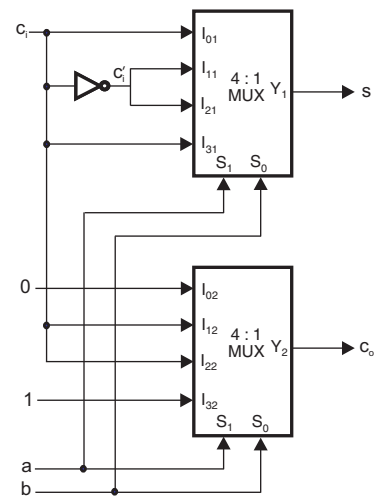


Fig. 1: Full adder using two 4:1 MUX.

b) Full Adder using 8:1 MUX

In 8:1 MUX if selection input S_2 is tied to 0 permanently then it will ignore the last four inputs and behave as 4:1 MUX. Now adder design using 8:1 MUX is same as that of design using 4:1 MUX. The logic circuit of full adder using two 8:1 MUX is shown in Fig. 2.

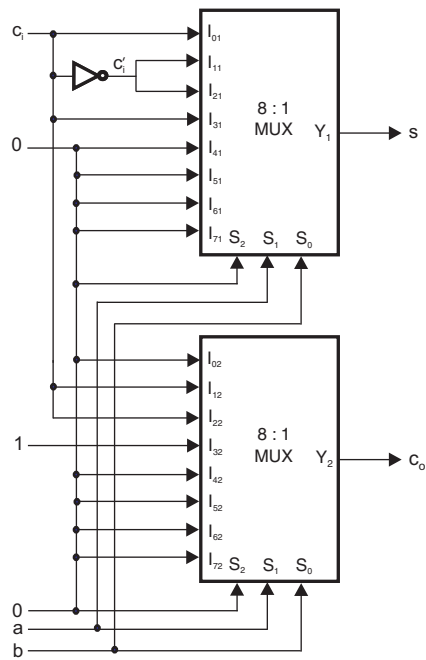


Fig. 2: Full adder using two 8:1 MUX.

1.12.1 Combinational Circuit Design using Multiplexer

In multiplexer, the possible binary values of selection inputs are same as minterms. Therefore, the logic circuit of Boolean functions in SOP form can be realized using multiplexer. Since a multiplexer has a single output, a separate multiplexer is required for every output of Boolean function.

There are different methods of implementing combinational circuits using multiplexer.

Method-1

The n -variable Boolean function can be implemented directly by $2^n:1$ multiplexer. The n inputs of Boolean function are connected to n selection inputs. The inputs $I_0, I_1, I_2, \dots, I_{2^n-1}$ are tied to either 0 or 1 from the knowledge of output in the truth table of Boolean function. If a minterm m_p generates 1 in output of Boolean function then input I_p of multiplexer is tied to 1. If a minterm m_q generates 0 in output of Boolean function then input I_q of multiplexer is tied to 0.

Method-2

The n -variable Boolean function can be implemented by $2^{n-1}:1$ multiplexer. A three variable Boolean function can be realized using $2:1$ multiplexer, a four variable Boolean function can be realized using $8:1$ multiplexer and so on.

In order to implement n -variable Boolean function the first $n-1$ inputs/variables of the function are applied to selection lines.

Now the function output can be represented by the remaining one variable, complement of this one variable, 0 and 1. The function output in terms of all this four (remaining one variable and its complement, 0 and 1) are applied as input $I_0, \dots, I_{2^{n-1}}$ of multiplexer.

Method-3

The n -variable Boolean function can be implemented using smaller size multiplexers than method-1 and 2. For example, a four variable Boolean function can be implemented using a $4:1$ multiplexer (Refer Example 1.58). In this method some of the inputs are connected to selection inputs of multiplexer and the remaining function inputs are applied to inputs of multiplexer through additional logic gates that can be designed by using K-maps.

Example 1.57

Implement the Boolean function $F(A, B, C) = \sum m(1, 3, 5, 6)$.

i) using $4:1$ multiplexer ii) using $8:1$ multiplexer.

Solution

Case i: Implementation using $4:1$ Multiplexer

The truth table of the given function is formed as shown in Table 1. The $4:1$ multiplexer will have two selection inputs S_0 and S_1 . Let us connect the inputs A and B to selection inputs S_1 and S_0 as shown in Fig. 1.

The selection inputs take four different values and so Table 1 is divided into 4 groups based on selection inputs and a separate table is drawn for each group as shown in Tables 2 to 5.

Table 1: Truth Table of F

Inputs			Minterm	Output F	Comments
A	B	C			
0	0	0	m_0	0	When, $A = B = C = 0$, $F = 0$
0	0	1	m_1	1	When, $A = B = 0$, $C = 1$, $F = 1$
0	1	0	m_2	0	When, $A = C = 0$, $B = 1$, $F = 0$
0	1	1	m_3	1	When, $A = 0$, $B = C = 1$, $F = 1$
1	0	0	m_4	0	When, $A = 1$, $B = C = 0$, $F = 0$
1	0	1	m_5	1	When, $A = C = 1$, $B = 0$, $F = 1$
1	1	0	m_6	1	When, $A = B = 1$, $C = 0$, $F = 1$
1	1	1	m_7	0	When, $A = B = C = 1$, $F = 0$

Truth Table of 4:1

Multiplexer

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

Table 2:

Inputs			Selection Inputs		F = C
A	B	C	S_1	S_0	
0	0	0	0	0	0
0	0	1	0	0	1

Table 3:

Inputs			Selection Inputs		F = C
A	B	C	S_1	S_0	
0	1	0	0	1	0
0	1	1	0	1	1

Table 4:

Inputs			Selection Inputs		F = C
A	B	C	S_1	S_0	
1	0	0	1	0	0
1	0	1	1	0	1

Table 5:

Inputs			Selection Inputs		F = C'
A	B	C	S_1	S_0	
1	1	0	1	1	1
1	1	1	1	1	0

The input I_0 , I_1 , I_2 and I_3 of multiplexer are obtained from Tables 2 to 5.

From Tables 2, 3 and 4 it can be observed that function output is C when selection inputs are 00, 01 and 10 and so I_0 , I_1 and I_2 are tied to C as shown in Fig. 1. From Table 5 it can be observed that function output is complement of C when selection input is 11 and so C is inverted to get C' and connected to I_3 as shown in Fig. 1. The working of 4:1 MUX is verified in Table 6.

Table 6: Verification of Logic Circuit of F in Fig. 1

A	B	C	S_1	S_0	F	Y = F
0	0	0	0	0	0	$Y = I_0 = C = 0$
0	0	1	0	0	1	$Y = I_0 = C = 1$
0	1	0	0	1	0	$Y = I_1 = C = 0$
0	1	1	0	1	1	$Y = I_1 = C = 1$
1	0	0	1	0	0	$Y = I_2 = C = 0$
1	0	1	1	0	1	$Y = I_2 = C = 1$
1	1	0	1	1	1	$Y = I_3 = C' = 1$
1	1	1	1	1	0	$Y = I_3 = C' = 0$

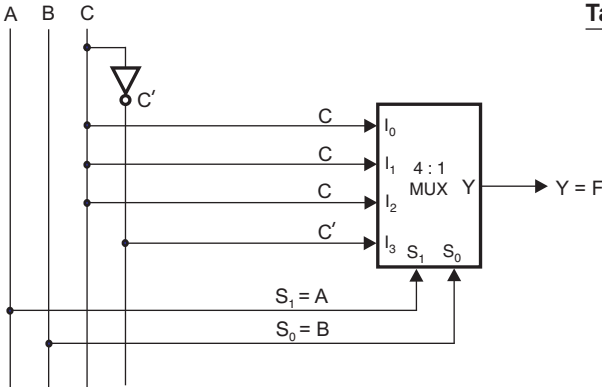


Fig. 1: Implementation of Boolean function using 4:1 MUX.

Case ii: Implementation using 8:1 Multiplexer

The function F is 3-variable function and so a $2^3:1$ multiplexer (8:1 multiplexer) can be used to implement the function as shown in Fig. 2. The function inputs A , B and C are connected to selection inputs S_2 , S_1 and S_0 . The function output F is 1 for minterms m_1 , m_3 , m_5 and m_6 and so the MUX inputs I_1 , I_3 , I_5 and I_6 are permanently tied to 1. The function output F is 0 for minterms m_0 , m_2 , m_4 and m_7 and so the MUX inputs I_0 , I_2 , I_4 and I_7 are permanently tied to 0. The working of 8:1 MUX is verified in Table 8.

Table 7: Truth Table of F

Inputs			Minterm	Output F	Comments
A	B	C			
0	0	0	m_0	0	When, $A = B = C = 0$, $F = 0$
0	0	1	m_1	1	When, $A = B = 0$, $C = 1$, $F = 1$
0	1	0	m_2	0	When, $A = C = 0$, $B = 1$, $F = 0$
0	1	1	m_3	1	When, $A = 0$, $B = C = 1$, $F = 1$
1	0	0	m_4	0	When, $A = 1$, $B = C = 0$, $F = 0$
1	0	1	m_5	1	When, $A = C = 1$, $B = 0$, $F = 0$
1	1	0	m_6	1	When, $A = B = 1$, $C = 0$, $F = 1$
1	1	1	m_7	0	When, $A = B = C = 1$, $F = 0$

Selection Inputs			Output Y
S_2	S_1	S_0	
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

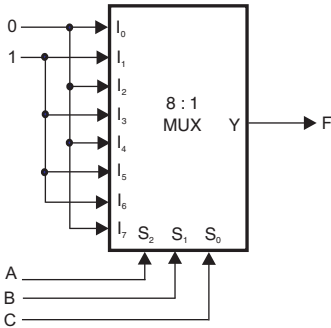


Fig. 2: Realization of F using 8:1 MUX.

Table 8: Verification of Logic Circuit of F in Fig. 2

A	B	C	S_2	S_1	S_0	F	$Y = F$
0	0	0	0	0	0	0	$Y = I_0 = 0$
0	0	1	0	0	1	1	$Y = I_1 = 1$
0	1	0	0	1	0	0	$Y = I_2 = 0$
0	1	1	0	1	1	1	$Y = I_3 = 1$
1	0	0	1	0	0	0	$Y = I_4 = 0$
1	0	1	1	0	1	1	$Y = I_5 = 1$
1	1	0	1	1	0	1	$Y = I_6 = 1$
1	1	1	1	1	1	0	$Y = I_7 = 0$

Example 1.58

Design two input XOR gate using 4:1 MUX.

Solution



Fig. 1: Symbol of 2-input XOR gate.

$$\therefore x \oplus y = x y' + x' y$$

Table 1: Truth Table of 2-input XOR gate

x	y	$x \cdot y'$	$x' \cdot y$	$F = x \oplus y$
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Implementation using 4:1 Multiplexer

The truth table of the given function is formed as shown in Table 2. The 4:1 multiplexer will have two selection inputs S_0 and S_1 . Let us connect the inputs x and y to selection inputs S_1 and S_0 as shown in Fig. 2.

The selection inputs take two different values and so Table 2 is divided into 2 groups based on selection inputs and a separate table is drawn for each group as shown in Tables 3 and 4.

Table 2: Truth Table of F

Inputs		Minterm	Output	Comments
x	y		F	
0	0	m_0	0	When, $x = y = 0$, $F = 0$
0	1	m_1	1	When, $x = 0$, $y = 1$, $F = 1$
1	0	m_2	1	When, $x = 1$, $y = 0$, $F = 1$
1	1	m_3	0	When, $x = y = 1$, $F = 0$

Truth Table of 4:1 Multiplexer		
Selection Input		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 3:

Inputs	Selection Inputs		F = y
x y	S_1	S_0	
0 0	0	0	0
0 1	0	1	1

Table 4:

Inputs	Selection Inputs		F = y'
x y	S_1	S_0	
1 0	1	0	1
1 1	1	1	0

The input I_0 , I_1 , I_2 and I_3 of multiplexer are obtained from Tables 3 and 4.

From Table 3, it can be observed that function output is y when selection inputs are 00 and 01 and so I_0 and I_1 are tied to y as shown in Fig. 2. From Table 4, it can be observed that function output is complement of y when selection input is 10 and 11 and so y is inverted to get y' and connected to I_2 and I_3 as shown in Fig. 2. The working of 4:1 MUX is verified in Table 5.

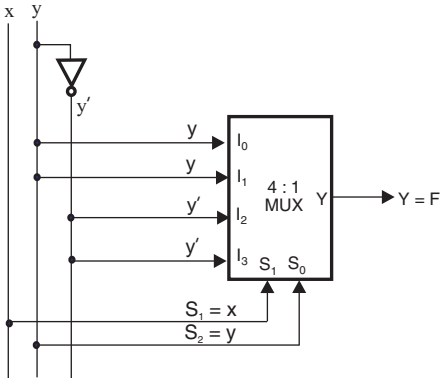


Table 5: Verification of Logic Circuit of F in Fig. 2

x	y	S_1	S_0	F	Y = F
0	0	0	0	0	$Y = I_0 = y = 0$
0	1	0	1	1	$Y = I_1 = y = 1$
1	0	1	0	1	$Y = I_2 = y' = 1$
1	1	1	1	0	$Y = I_3 = y' = 0$

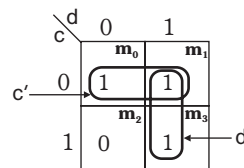
Fig. 2: Implementation of Boolean function using 4:1 MUX.

Implement Boolean function $F(a, b, c, d) = \sum m(4, 5, 7, 8, 10, 12, 15)$ using 4:1 MUX and external gates:

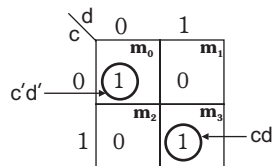
- a) a, b are connected to select lines S_1 and S_0 respectively.
b) c, d are connected to select lines S_1 and S_0 respectively.

a) a, b are connected to select lines S_1 and S_0

Inputs				Minterm	Output	Inputs				Minterm	Output
a	b	c	d			a	b	c	d		
0	0	0	0	m_0	0	1	0	0	0	m_8	1
0	0	0	1	m_1	0	1	0	0	1	m_9	0
0	0	1	0	m_2	0	1	0	1	0	m_{10}	1
0	0	1	1	m_3	0	1	0	1	1	m_{11}	0
0	1	0	0	m_4	1	1	1	0	0	m_{12}	1
0	1	0	1	m_5	1	1	1	0	1	m_{13}	0
0	1	1	0	m_6	0	1	1	1	0	m_{14}	0
0	1	1	1	m_7	1	1	1	1	1	m_{15}	1



$$F = c' + d$$



$$F = cd + c'd'$$

Fig: K-map for Table 5.

Inputs				Selection Inputs		F = 0
a	b	c	d	S ₁	S ₀	
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0

Inputs				Selection Inputs		c'	$c' + d$	$F = c' + d$
a	b	c	d	S_1	S_0			
0	1	0	0	0	1	1	1	1
0	1	0	1	0	1	1	1	1
0	1	1	0	0	1	0	0	0
0	1	1	1	0	1	0	1	1

Inputs				Selection Inputs		d'	F = d'
a	b	c	d	S ₁	S ₀		
1	0	0	0	1	0	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	0	1	1	1	0	0	0

Inputs				Selection Inputs		cd	c'd'	F = cd + c'd'
a	b	c	d	S ₁	S ₀			
1	1	0	0	1	1	0	1	1
1	1	0	1	1	1	0	0	0
1	1	1	0	1	1	0	0	0
1	1	1	1	1	1	1	0	1

Table 6: Verification

a	b	c	d	S ₁	S ₀	F	Y = F
0	0	0	0	0	0	0	$I_0 = 0$
0	0	0	1	0	0	0	$I_0 = 0$
0	0	1	0	0	0	0	$I_0 = 0$
0	0	1	1	0	0	0	$I_0 = 0$
0	1	0	0	0	1	1	$I_1 = c' + d = 1$
0	1	0	1	0	1	1	$I_1 = c' + d = 1$
0	1	1	0	0	1	0	$I_1 = c' + d = 0$
0	1	1	1	0	1	1	$I_1 = c' + d = 1$
1	0	0	0	1	0	1	$I_2 = d' = 1$
1	0	0	1	1	0	0	$I_2 = d' = 0$
1	0	1	0	1	0	1	$I_2 = d' = 1$
1	0	1	1	1	0	0	$I_2 = d' = 0$
1	1	0	0	1	1	1	$I_3 = cd + c'd' = 1$
1	1	0	1	1	1	0	$I_3 = cd + c'd' = 0$
1	1	1	0	1	1	0	$I_3 = cd + c'd' = 0$
1	1	1	1	1	1	1	$I_3 = cd + c'd' = 1$

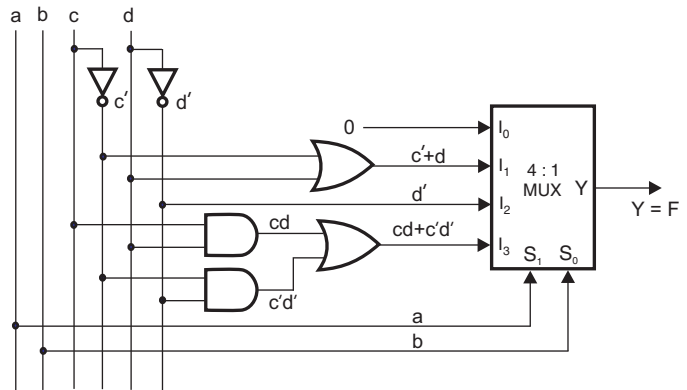


Fig. 1: Implementation of Boolean function using 4:1 MUX by taking a and b as selection inputs..

b) c, d are connected to select lines S₁ and S₀

Table 7:

Inputs				Selection Inputs		F
a	b	c	d	S ₁	S ₀	
0	0	0	0	0	0	$I_0 = 0$
0	0	0	1	0	1	$I_1 = 0$
0	0	1	0	1	0	$I_2 = 0$
0	0	1	1	1	1	$I_3 = 0$

Table 8:

Inputs				Selection Inputs		F
a	b	c	d	S ₁	S ₀	
0	1	0	0	0	0	$I_0 = 1$
0	1	0	1	0	1	$I_1 = 1$
0	1	1	0	1	0	$I_2 = 0$
0	1	1	1	1	1	$I_3 = 1$

Table 9:

Inputs				Selection Inputs		F
a	b	c	d	S ₁	S ₀	
1	0	0	0	0	0	$I_0 = 1$
1	0	0	1	0	1	$I_1 = 0$
1	0	1	0	1	0	$I_2 = 1$
1	0	1	1	1	1	$I_3 = 0$

Table 10:

Inputs				Selection Inputs		F
a	b	c	d	S ₁	S ₀	
1	1	0	0	0	0	$I_0 = 1$
1	1	0	1	0	1	$I_1 = 0$
1	1	1	0	1	0	$I_2 = 0$
1	1	1	1	1	1	$I_3 = 1$

In order to find Boolean equations for 4:1 MUX inputs I_0 , I_1 , I_2 and I_3 the Tables 11 to 14 are formed as shown ahead and K-maps are constructed to determine the Boolean equations for I_0 , I_1 , I_2 and I_3 .

Table 11:

a	b	$F = I_0$
0	0	0
0	1	1
1	0	1
1	1	1

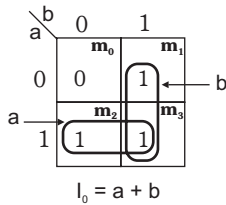
Fig 2: K-map for I_0 .

Table 12:

a	b	$F = I_1$
0	0	0
0	1	1
1	0	0
1	1	0

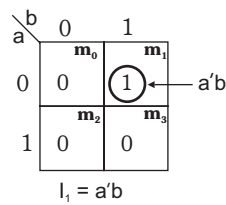
Fig 3: K-map for I_1 .

Table 13:

a	b	$F = I_2$
0	0	0
0	1	0
1	0	1
1	1	0

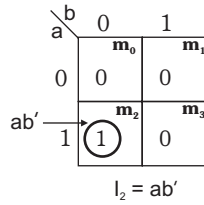
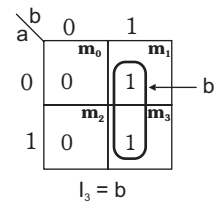
Fig 4: K-map for I_2 .

Table 14:

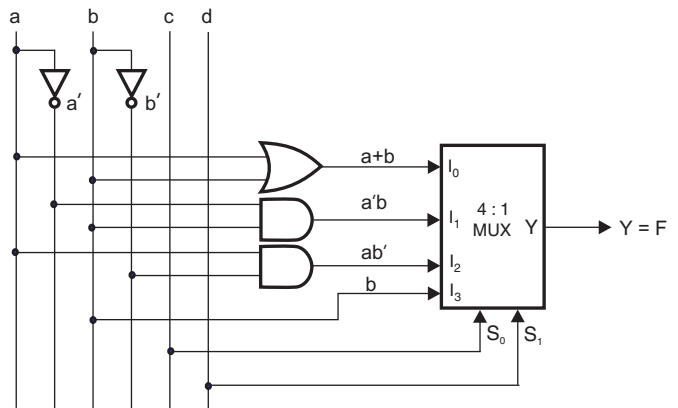
a	b	$F = I_3$
0	0	0
0	1	1
1	0	0
1	1	1

Fig 5: K-map for I_3 .

Using the above Boolean equations the given Boolean function is implemented using 4:1 MUX by taking c and d as selection inputs as shown in Fig. 6.

Table 11: Verification

a	b	c	d	S_1	S_0	F	$Y = F$
0	0	0	0	0	0	0	$I_0 = a + b = 0$
0	0	0	1	0	1	0	$I_1 = a'b = 0$
0	0	1	0	1	0	0	$I_2 = ab' = 0$
0	0	1	1	1	1	0	$I_3 = b = 0$
0	1	0	0	0	0	1	$I_0 = a + b = 1$
0	1	0	1	0	1	1	$I_1 = a'b = 1$
0	1	1	0	1	0	0	$I_2 = ab' = 0$
0	1	1	1	1	1	1	$I_3 = b = 1$
1	0	0	0	1	0	1	$I_0 = a + b = 1$
1	0	0	1	1	0	0	$I_1 = a'b = 0$
1	0	1	0	1	0	1	$I_2 = ab' = 1$
1	0	1	1	1	0	0	$I_3 = b = 0$
1	1	0	0	1	1	1	$I_3 = a + b = 1$
1	1	0	1	1	1	0	$I_3 = a'b = 0$
1	1	1	0	1	1	0	$I_3 = ab' = 0$
1	1	1	1	1	1	1	$I_3 = b = 1$

Fig. 6: Implementation of Boolean function using 4:1 MUX by taking c and d as selection inputs..**Example 1.60**

(AU, Apr/May'23, 6, 15 Marks)

Realize $F(A, B, C, D) = \sum m(0, 1, 3, 4, 8, 9, 15)$ using 8:1 MUX by selecting A as an input.

Solution

It is given that function output F is 1 when the inputs are equal to minterms $m_0, m_1, m_3, m_4, m_8, m_9$ and m_{15} . For all other inputs the function output F is 0.

The given function is analyzed using Table 1 and observed that the function output can be expressed in terms of A, A', 0 and 1. Therefore, the given 4-variable function can be realized using $2^{4-1}:1$ multiplexer (8:1 multiplexer) by taking B, C and D as selection inputs as shown in Fig. 1.

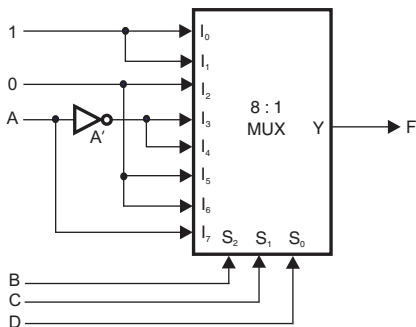
Table 1: Truth Table of F

Inputs				Minterm	Output F	Output in terms of A, A', 0, 1	Comments
A	B	C	D				
0	0	0	0	m_0	1	$F = 1$	When, B = C = D = 0
1	0	0	0	m_8	1	$F = 1$	Output, F = 1
0	0	0	1	m_1	1	$F = 1$	When, B = C = 0, D = 1
1	0	0	1	m_9	1	$F = 1$	Output, F = 1
0	0	1	0	m_2	0	$F = 0$	When, B = D = 0, C = 1
1	0	1	0	m_{10}	0	$F = 0$	Output, F = 0
0	0	1	1	m_3	1	$F = A'$	When, B = 0, C = D = 1
1	0	1	1	m_{11}	0	$F = A'$	Output, F = A'
0	1	0	0	m_4	1	$F = A'$	When B = 1, C = D = 0
1	1	0	0	m_{12}	0	$F = A'$	Output, F = A'
0	1	0	1	m_5	0	$F = 0$	When, B = D = 1, C = 0
1	1	0	1	m_{13}	0	$F = 0$	Output, F = 0
0	1	1	0	m_6	0	$F = 0$	When, B = C = 1, D = 0
1	1	1	0	m_{14}	0	$F = 0$	Output, F = 0
0	1	1	1	m_7	0	$F = A$	When, B = C = D = 1
1	1	1	1	m_{15}	1	$F = A$	Output, F = A

The implementation of function, F using 8:1 MUX is shown in Fig. 1. The variables B, C and D are used as selection inputs S_2 , S_1 and S_0 of 8:1 MUX. In 8:1 multiplexer of Fig. 1, the inputs I_2 , I_5 and I_6 are permanently tied to 0 and the inputs I_0 and I_1 are tied to 1. The input I_7 is tied to A and the inputs I_3 and I_4 are tied to A'. The working of 8:1 multiplexer is verified in Table 2.

Table 2: Verification of Logic Circuit of F in Fig. 1

Inputs				Selection Inputs			Function Output	MUX Output
A	B	C	D	S_2	S_1	S_0	F	$Y = F$
0	0	0	0	0	0	0	1	$Y = I_0 = 1$
1	0	0	0	0	0	0	1	$Y = I_0 = 1$
0	0	0	1	0	0	1	1	$Y = I_1 = 1$
1	0	0	1	0	0	1	1	$Y = I_1 = 1$
0	0	1	0	0	1	0	0	$Y = I_2 = 0$
1	0	1	0	0	1	0	0	$Y = I_2 = 0$
0	0	1	1	0	1	1	1	$Y = I_3 = A' = 1$
1	0	1	1	0	1	1	0	$Y = I_3 = A' = 0$
0	1	0	0	1	0	0	1	$Y = I_4 = A' = 1$
1	1	0	0	1	0	0	0	$Y = I_4 = A' = 0$
0	1	0	1	1	0	1	0	$Y = I_5 = 0$
1	1	0	1	1	0	1	0	$Y = I_5 = 0$
0	1	1	0	1	1	0	0	$Y = I_6 = 0$
1	1	1	0	1	1	0	0	$Y = I_6 = 0$
0	1	1	1	1	1	1	0	$Y = I_7 = A = 0$
1	1	1	1	1	1	1	1	$Y = I_7 = A = 1$

**Fig. 1: Realization of F using 8:1 MUX.**

Example 1.61

Implement the following Boolean function

$$F(w, x, y, z) = \sum m(2, 3, 5, 6, 11, 14, 15)$$

i) using multiplexer, ii) using decoder, iii) using multiplexer and decoder

Solution**Case i: Implementation using Multiplexer**

It is given that function output F is 1 when the inputs are equal to minterms $m_2, m_3, m_5, m_6, m_{11}, m_{14}$ and m_{15} . For all other inputs the function output F is 0.

The given function is analyzed using Table 1 and observed that the function output can be expressed in terms of $z, z', 0$ and 1 . Therefore, the given 4-variable function can be realized using $2^{4-1}:1$ multiplexer (8:1 multiplexer).

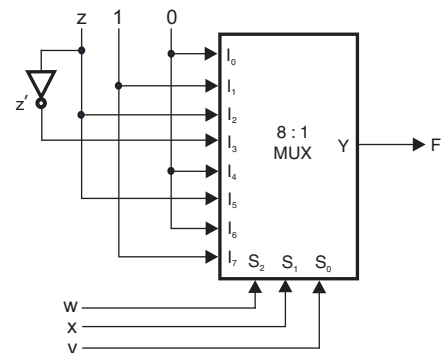
Table 1: Truth Table of F

Inputs				Minterm	Output F	Output in terms of $z, z', 0, 1$	Comments
w	x	y	z				
0	0	0	0	m_0	0	$F = 0$	When, $w = x = y = 0$ Output, $F = 0$
0	0	0	1	m_1	0	$F = 0$	
0	0	1	0	m_2	1	$F = 1$	When, $w = x = 0, y = 1$ Output, $F = 1$
0	0	1	1	m_3	1	$F = 1$	
0	1	0	0	m_4	0	$F = z$	When, $w = y = 0, x = 1$ Output, $F = z$
0	1	0	1	m_5	1	$F = z$	
0	1	1	0	m_6	1	$F = z'$	When, $w = 0, x = y = 1$ Output, $F = z'$
0	1	1	1	m_7	0	$F = z'$	
1	0	0	0	m_8	0	$F = 0$	When $w = 1, x = y = 0$ Output, $F = 0$
1	0	0	1	m_9	0	$F = 0$	
1	0	1	0	m_{10}	0	$F = z$	When, $w = y = 1, x = 0$ Output, $F = z$
1	0	1	1	m_{11}	1	$F = z$	
1	1	0	0	m_{12}	0	$F = 0$	When, $w = x = 1, y = 0$ Output, $F = 0$
1	1	0	1	m_{13}	0	$F = 0$	
1	1	1	0	m_{14}	1	$F = 1$	When, $w = x = y = 1$ Output, $F = 1$
1	1	1	1	m_{15}	1	$F = 1$	

The implementation of function, F using 8:1 MUX is shown in Fig. 1. The variables w, x and y are used as selection inputs S_2, S_1 and S_0 of 8:1 MUX. In 8:1 MUX of Fig. 1, the inputs I_0, I_4 and I_5 are permanently tied to 0 and the inputs I_1 and I_7 are tied to 1. The inputs I_2 and I_6 are tied to z and the input I_3 is tied to z' . The working of 8:1 MUX is verified in Table 2.

Table 2: Verification of Logic Circuit of F in Fig. 1

w	x	y	z	S_2	S_1	S_0	F	Y = F
0	0	0	0	0	0	0	0	$Y = I_0 = 0$
0	0	0	1	0	0	0	0	$Y = I_0 = 0$
0	0	1	0	0	0	1	1	$Y = I_1 = 1$
0	0	1	1	0	0	1	1	$Y = I_1 = 1$
0	1	0	0	0	1	0	0	$Y = I_2 = z = 0$
0	1	0	1	0	1	0	1	$Y = I_2 = z = 1$
0	1	1	0	0	1	1	1	$Y = I_3 = z' = 1$
0	1	1	1	0	1	1	0	$Y = I_3 = z' = 0$
1	0	0	0	1	0	0	0	$Y = I_4 = 0$
1	0	0	1	1	0	0	0	$Y = I_4 = 0$
1	0	1	0	1	0	1	0	$Y = I_5 = z = 0$
1	0	1	1	1	0	1	1	$Y = I_5 = z = 1$
1	1	0	0	1	1	0	0	$Y = I_6 = 0$
1	1	0	1	1	1	0	0	$Y = I_6 = 0$
1	1	1	0	1	1	1	1	$Y = I_7 = 1$
1	1	1	1	1	1	1	1	$Y = I_7 = 1$

**Fig. 1: Implementation of F using multiplexer.**

Case ii: Implementation using Decoder

From Table 1 it can be observed that when $w = 0$, the inputs x , y and z can take 8 combinations of binary. Similarly, when $w = 1$, the inputs x , y and z can take 8 combinations of binary. Hence, two 3-to-8 decoders can be used to implement the given function. One decoder is enabled by w' and other by w .

The inputs of both the decoders are connected to x , y and z . The output of decoder enabled by w' will be same as function output when inputs are minterms m_0 to m_7 . The output of decoder enabled by w will be same as function output when inputs are minterms m_8 to m_{15} . The required outputs are logically ORed to get function output as shown in Fig. 2.

On taking into account the minterms for which function outputs are 1, the following Boolean equations are obtained using the logic **high** decoder output.

$$F_1 = Y_{21} + Y_{31} + Y_{51} + Y_{61} \quad ; \quad F_2 = Y_{32} + Y_{62} + Y_{72}$$

$$F = F_1 + F_2$$

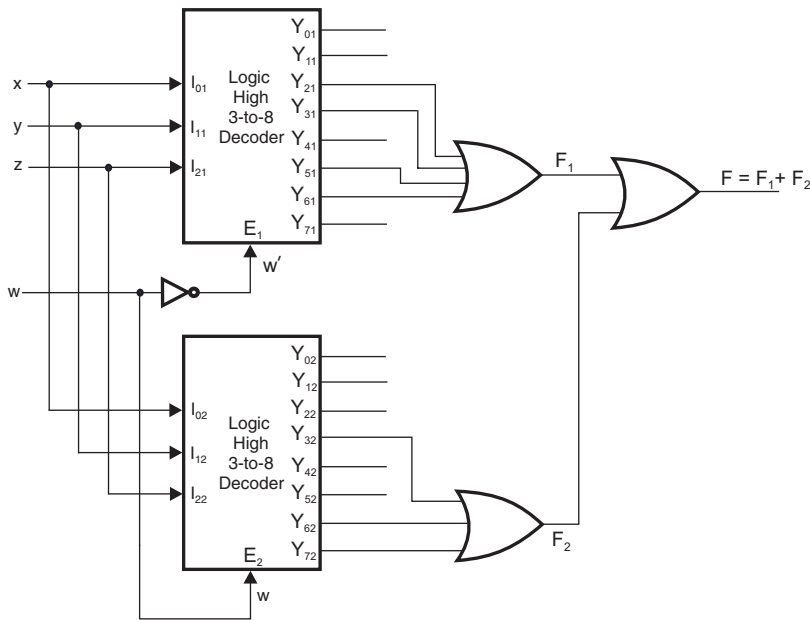


Fig. 2: Implementation of F using decoder.

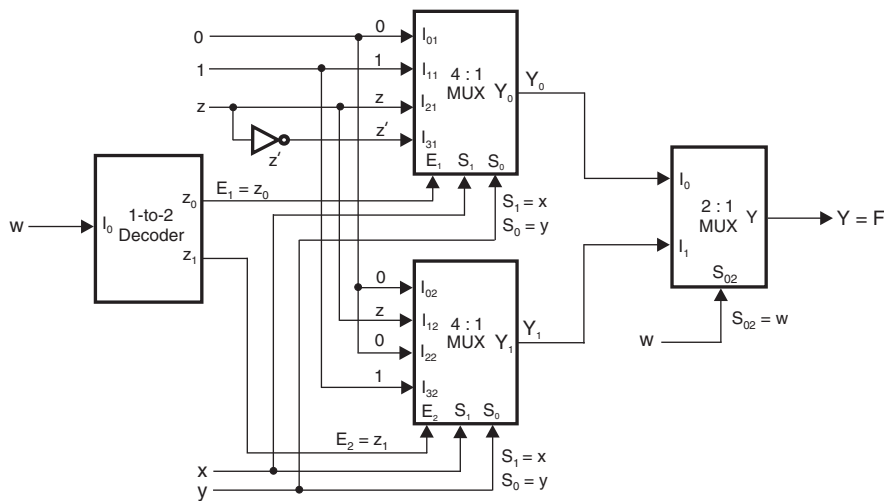
Case iii: Implementation using Multiplexer and Decoder

In order to implement the given 4-variable Boolean function, one 1-to-2 decoder, two 4:1 MUX and one 2:1 MUX are used as shown in Fig. 3. A 4:1 MUX has 4 inputs, 2 selection inputs, 1 enable input and 1 output. The enable inputs are generated using 1-to-2 decoder. The outputs of decoder are used as enable inputs for 4:1 MUX.

The inputs x and y are used as selection inputs S_1 and S_0 of 4:1 MUX.

The output of 4:1 MUX are again connected as inputs of 2:1 MUX. The input w is used as selection input S_{02} of 2:1 MUX.

In the first 4:1 MUX of Fig. 3, the input I_{01} is tied to 0 and I_{11} is tied to 1. The input I_{21} is tied to z and I_{31} is tied to z' . In the second 4:1 MUX of Fig. 3, the inputs I_{02} and I_{22} are tied to 0, the input I_{32} is tied to 1 and the input I_{12} is tied to 1. The working of the circuit of Fig. 3, is verified in Table 3.

Fig. 3: Implementation of F using decoder and multiplexer.Table 3: Verification of Logic Circuit of F in Fig. 3

Inputs				z'	Selection Inputs of 4:1 MUX		Selection Input of 2:1 MUX $S_{02} = w$	Function Output F	2:1 MUX Output $Y = F$	
w	x	y	z		$S_1 = x$	$S_0 = y$				
0	0	0	0	1	0	0	0	0	$Y = I_0 = Y_0$	$Y_0 = I_{01} = 0$
0	0	0	1	0	0	0	0	0		$Y_0 = I_{11} = 1$
0	0	1	0	1	0	1	0	1		$Y_0 = I_{21} = z$
0	1	0	0	1	1	0	0	0		$Y_0 = I_{31} = z'$
0	1	1	0	1	1	1	0	1		
0	1	1	1	0	1	1	0	0	$Y = I_1 = Y_1$	$Y_1 = I_{02} = 0$
1	0	0	0	1	0	0	1	0		$Y_1 = I_{12} = z$
1	0	0	1	0	0	0	1	0		$Y_1 = I_{22} = 0$
1	0	1	0	1	0	1	1	0		$Y_1 = I_{23} = 1$
1	0	1	1	0	0	1	1	1		
1	1	0	0	1	1	0	1	0	$Y = I_1 = Y_1$	$Y_1 = I_{02} = 0$
1	1	0	1	0	1	0	1	0		$Y_1 = I_{12} = z$
1	1	1	0	1	1	1	1	1		$Y_1 = I_{22} = 0$
1	1	1	1	0	1	1	1	1		$Y_1 = I_{23} = 1$
1	1	1	1	0	1	1	1	1		

1.13 Demultiplexer

A demultiplexer is a combinational circuit that send one binary information to many outputs. For this reason a demultiplexer is otherwise called a **data distributor**.

A demultiplexer will perform the reverse operation of multiplexer. Therefore, a demultiplexer is a combinational circuit which can transmit a binary input to any one of 2^n output lines using n selection lines or n -bit address. The demultiplexer is shortly called **demux**.

The block diagram and symbolic representation of $1:2^n$ demultiplexer with n selection inputs is shown in Fig. 1.124.

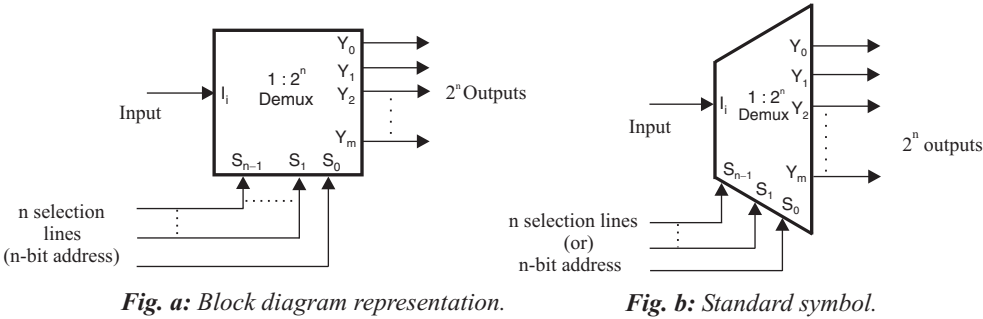


Fig. 1.124: $1:2^n$ demultiplexer.

1:2 Demux

The **1:2 demux** will transmit one binary input to any one of the two outputs depending on the value of selection input.

Let, I_i = Input
 S_0 = Selection input/Address
 Y_0, Y_1 = Outputs

In 1:2 demultiplexer,

When, $S_0 = 0$, I_i is passed to Y_0 , $\therefore Y_0 = I_i$
 When, $S_0 = 1$, I_i is passed to Y_1 , $\therefore Y_1 = I_i$

The above selection logic is listed as truth table in Table 1.73. The block diagram and symbolic representation of 1:2 demux with selection input S_0 are shown in Fig. 1.125.

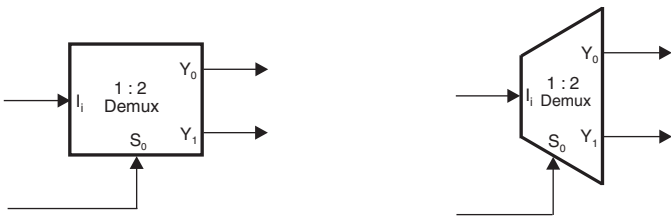


Table 1.73: Truth Table of 1:2 Demultiplexer

S_0	Y_0	Y_1
0	I_i	0
1	0	I_i

Fig. a: Block diagram representation. **Fig. b: Standard symbol.**
Fig. 1.125: 1:2 demultiplexer.

The truth table of 1:2 demux for all possible combination of inputs is shown in Table 1.74. The K-maps for design of 1:2 demux are shown in Fig. 1.126 and from the K-map we get the following Boolean equations,

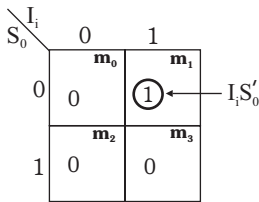
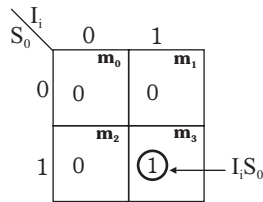
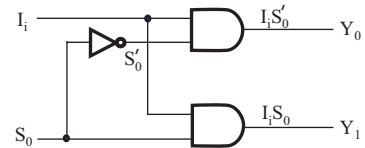
$$Y_0 = I_i S_0'$$

$$Y_1 = I_i S_0$$

The logic circuit of 1:2 demux is drawn using the above Boolean equations as shown in Fig. 1.127.

Table 1.74: Truth Table of 1:2 Demux for all Combination of Inputs

S_0	I_i	Minterm	Y_0	Y_1
0	0	m_0	$I_i = 0$	0
0	1	m_1	$I_i = 1$	0
1	0	m_2	0	$I_i = 0$
1	1	m_3	0	$I_i = 1$

**Fig. a:** K-map for Y_0 .**Fig. b:** K-map for Y_1 .**Fig. 1.126:** K-map for design of 1:2 demux.**Fig. 1.127:** Logic circuit of 1:2 demux.

1:4 Demux

The **1:4 demultiplexer** will transmit one binary input to any one of the four outputs depending on the value of selection input.

Let, I_i = Input

S_0, S_1 = Selection inputs

Y_0, Y_1, Y_2, Y_3 = Outputs

In 1:4 demultiplexer,

When, $S_1 = 0$ and $S_0 = 0$, I_i is passed to Y_0 , $\therefore Y_0 = I_i$

When, $S_1 = 0$ and $S_0 = 1$, I_i is passed to Y_1 , $\therefore Y_1 = I_i$

When, $S_1 = 1$ and $S_0 = 0$, I_i is passed to Y_2 , $\therefore Y_2 = I_i$

When, $S_1 = 1$ and $S_0 = 1$, I_i is passed to Y_3 , $\therefore Y_3 = I_i$

The above selection logic is listed as truth table in Table 1.75. The block diagram and symbolic representation of 1:4 demux with selection inputs S_0 and S_1 are shown in Fig. 1.128.

Table 1.75: Truth Table of 1:4 Demux

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I_i	0	0	0
0	1	0	I_i	0	0
1	0	0	0	I_i	0
1	1	0	0	0	I_i

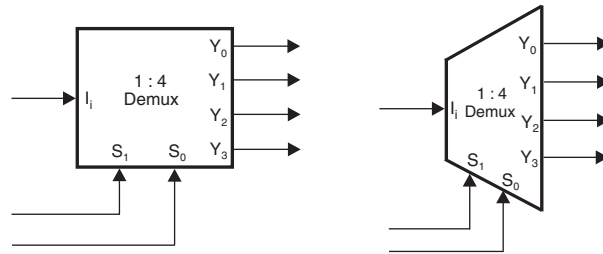


Fig. a: Block diagram representation. **Fig. b:** Standard symbol.

Fig. 1.128: 1:4 demultiplexer.

The truth table of 1:4 demultiplexer for all possible combination of inputs is shown in Table 1.76.

Table 1.76: Truth Table of 1:8 Demux for all Combination of Inputs

S_1	S_0	I_i	Minterm	Y_0	Y_1	Y_2	Y_3
0	0	0	m_0	$I_i = 0$	0	0	0
0	0	1	m_1	$I_i = 1$	0	0	0
0	1	0	m_2	0	$I_i = 0$	0	0
0	1	1	m_3	0	$I_i = 1$	0	0
1	0	0	m_4	0	0	$I_i = 0$	0
1	0	1	m_5	0	0	$I_i = 1$	0
1	1	0	m_6	0	0	0	$I_i = 0$
1	1	1	m_7	0	0	0	$I_i = 1$

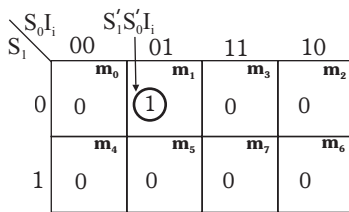


Fig. a: K-map for Y_0 .

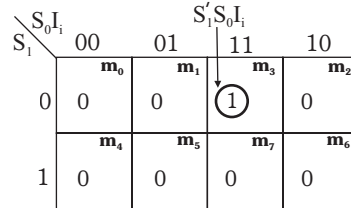


Fig. b: K-map for Y_1 .

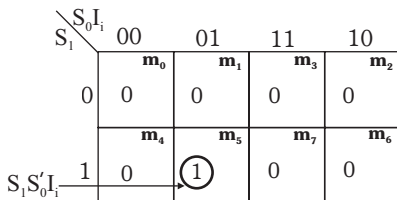


Fig. c: K-map for Y_2 .

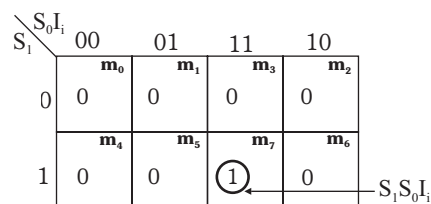


Fig. d: K-map for Y_3 .

Fig. 1.129: K-map for design of 1:4 demux.

The K-maps for design of 1:4 demultiplexer are drawn in Fig. 1.129 and from the K-map we get the following Boolean equations,

$$\begin{array}{l|l} Y_0 = I_i S_1' S_0' & Y_2 = I_i S_1 S_0' \\ Y_1 = I_i S_1' S_0 & Y_3 = I_i S_1 S_0 \end{array}$$

The logic circuit of 1:4 demux is drawn using the above Boolean equations as shown in Fig. 1.130.

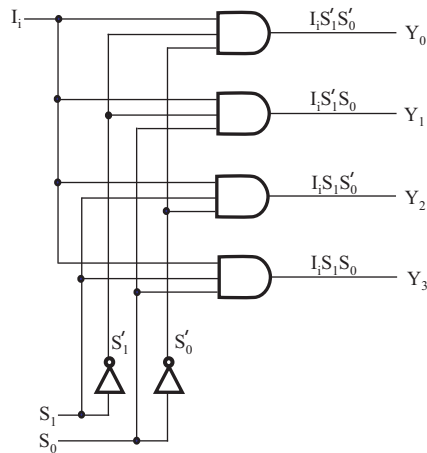


Fig. 1.130: Logic circuit of 1:4 demux.

The dual 1:4 demux is available as a standard IC with number 74155. The pin configuration of 74155 is shown in Fig. 1.131.

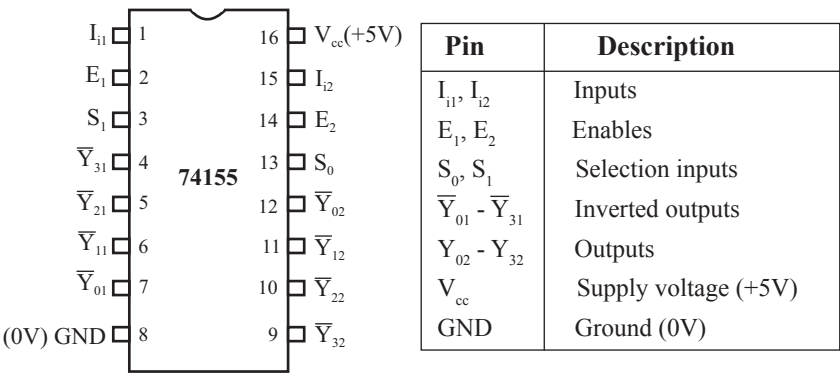


Fig. 1.131: Pin configuration of dual 1:4 demultiplexer IC 74155.

1:8 Demux

The **1:8 demultiplexer** will transmit one binary input to any one of the eight outputs depending on the value of selection inputs.

Let, I_i = Input

S_0, S_1, S_2 = Selection inputs

$Y_0, Y_1, Y_2, \dots, Y_7$ = Outputs

The logic to pass the input to output in 1:8 demux is listed in Table 1.77.

Table 1.77: Truth Table of 1:8 Demux

[illegible]

The block diagram and symbolic representation of 1:8 demux is shown in Fig. 1.132.

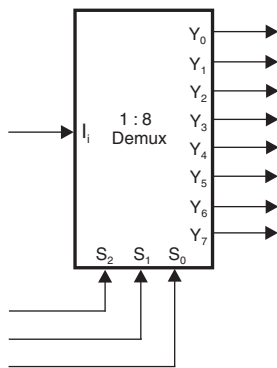


Fig. a: Block diagram representation.

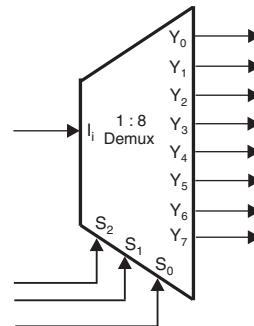


Fig. b: Standard symbol.

Fig. 1.132: 1:8 demultiplexer.

Table 1.78: Verification of Output of 1:8 Demux of Fig. 1.132

[illegible]

The logic circuit of 1:8 demux can be realized using AND gate as shown in Fig. 1.133 and Table 1.78 can be used to verify the logic circuit of Fig. 1.133.

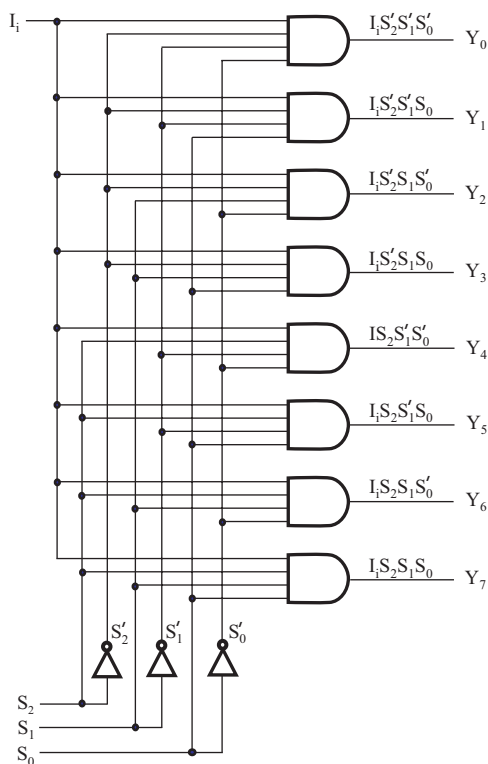


Fig. 1.133: Logic circuit of 1:8 demux.

1.13.1 Demultiplexers and Decoders

The demultiplexer can be made to work as logic high decoder if the select inputs of demultiplexer are made inputs of decoder and data of input of demultiplexer is permanently tied to 1.

On the other hand, the decoder can be made to work as demultiplexer if the input of logic high decoder is made as select inputs of demultiplexer and enable input of decoder is made as data input of demultiplexer.

The above concepts can be directly verified or observed from truth tables of decoder and demultiplexer. Many of the IC manufactures name the decoder and demultiplexer ICs as decoder/demultiplexer.

1.14 Summary of Important Concepts

1. Electronic circuits constructed using digital logic gates and devices and designed to operate on digital inputs and outputs are called digital logic circuits.
2. The digital logic circuits can be broadly classified into combinational circuits and sequential circuits.
3. The combinational circuits are digital logic circuits without feedbacks from output to input.
4. The sequential circuits are digital logic circuits with feedback from output to input.
5. George Boole developed Boolean algebra in 1854.

6. Boolean algebra is an algebraic structure that includes a set of elements consisting of binary operators "+" and ".", binary variables x, y, z, \dots , binary elements 0 and 1 and Boolean postulates and theorems.
7. A statement written with Boolean variables, constants and operators is called Boolean expression.
8. Postulates of Boolean algebra are developed by E.V.Huntington in 1904.
9. In positive logic, the logical AND of two or more variables will be 1 if and only if the value of all the variables are 1.
10. In positive logic, the logical OR of two or more variables will be 1 if the value of one of the variables is 1.
11. NOT operation is same as complement operation.
12. Logic gates are electronic devices or circuits that perform logical operations on one or more input logical variables and produce a binary output.
13. NAND and NOR gates are called universal gates, because any Boolean function can be realized only using NAND gates or only using NOR gates.
14. A Boolean function is described by a Boolean expression which consists of binary variables, binary constants 0 and 1 and logical operators AND, OR and NOT.
15. Minterms are 2^n possible combinations of AND terms with n variables such that the logical AND of all the variables is 1.
16. Maxterms are 2^n possible combinations of OR terms with n variables such that the logical OR of all the variables is 0.
17. The three types of standard forms of expressing a Boolean function are Sum-of-Products (SOP) form, Product-of-Sums (POS) form and Canonical form.
18. The SOP or POS form is said to be canonical only if all the variables are present in every term of the SOP or POS form.
19. The realizations in SOP and POS forms are called two level realization of standard forms.
20. Each variable within a term of a Boolean expression is called a literal.
21. The complement of a function can be obtained by using duality by replacing "+" by "." and "." by "+" and variables by its complement.
22. K-map is a pictorial form of truth table and used to simplify Boolean functions.
23. A K-map is a diagram made up of squares with each square representing a minterm.
24. When minterms are considered for simplification using K-map the resultant Boolean function will be in sum-of-products form.
25. When maxterms are considered for simplification using K-map the resultant Boolean function will be in product-of-sums form.
26. For n -variable Boolean function the K-map will have 2^n squares.
27. In a K-map, combine adjacent 1's to form prime implicants.
28. While forming prime implicants any number of over lapping is allowed in horizontal and vertical directions.
29. When squares with 1's are considered for forming prime implicants, the simplified Boolean function is sum of all the product terms of prime implicant.
30. If all minterms are 1's then all squares of K-map will be filled by 1 and the function value is 1.

31. The undefined function outputs are called don't-care conditions and denoted by \times .
32. In simplification of Boolean functions using K-maps, don't-care conditions can be considered as either 0 or 1.
33. Binary arithmetic is similar to decimal arithmetic of only two numbers 0 and 1 with results in binary.
34. Adders have been developed to perform arithmetic addition operation on binary numbers.
35. Half adder can perform addition of two 1-bit binary number.
36. Full adder is a combinational circuit that perform the arithmetic sum of three binary bits.
37. The outputs of half and full adders are sum and carry.
38. A binary parallel adder performs arithmetic sum of two n-bit binary numbers.
39. In decimal addition we can use binary or BCD to represent decimal numbers.
40. In BCD addition the sum of any two digits will be in the range 0 to 19_{10} .
41. Subtractors have been developed to perform arithmetic subtraction operation on binary numbers.
42. Half subtractor can perform subtraction of two 1-bit binary number.
43. Full subtractor is a combinational circuit that perform arithmetic subtraction of three binary bits.
44. The outputs of half and full subtractors are difference and borrow.
45. Magnitude comparator is a combinational circuit used to compare two binary numbers.
46. A decoder is a combinational logic device that decodes n-bit binary input to one of the 2^n binary information.
47. An encoder is a combinational circuit that will generate a unique n-bit output, for each of the 2^n inputs.
48. A priority encoder is an encoder circuit that includes a priority function in order to recognize only one input when multiple inputs are asserted **high**.
49. A multiplexer is a combinational circuit which can select one of the 2^n inputs as output using n selection lines.
50. Multiplexer of smaller size can be connected in cascade to expand the size of multiplexer.
51. A demultiplexer will perform the reverse operation of multiplexer.
52. A demultiplexer is a combinational circuit which can transmit a binary input to any one of 2^n output lines using n selection lines or n-bit address.
53. Demultiplexers of smaller size can be connected in parallel to expand the size of demultiplexer.

1.15 Short-Answer Questions

Q1.1 Define digital logic circuits.

Electronic circuits constructed using digital logic gates and devices and designed to operate on digital inputs and outputs are called digital logic circuits.

Q1.2 What are the types of digital logic circuits.

The digital logic circuits can be broadly classified into,

1. Combinational circuits
 2. Sequential circuits.
-

Q1.3 Explain the design procedure for combinational circuits.

1. Determine the required inputs and outputs from the problem specifications.
2. Assign a symbol to each input and output.
3. Derive the truth table.
4. Draw K-map for every output and form the prime implicants.
5. Determine the simplified Boolean function for every output from the prime implicants.
6. Implement the Boolean functions of all the outputs as a digital circuit using logic gates.

Q1.4 Prove that $A + A'B = A + B$ using Boolean algebra.**Solution:**

$$\begin{aligned}
 A + A'B &= A(1 + B) + A'B \\
 &= A + AB + A'B \\
 &= A + (A + A')B \\
 &= A + B
 \end{aligned}$$

$$x + x' = 1$$

Q1.5 Simplify the three variable function $Y(A, B, C) = \sum m(1, 3, 5, 7)$ using Boolean algebra.**Solution:**

$$\begin{aligned}
 Y(A, B, C) &= \sum m(1, 3, 5, 7) \\
 &= A'B'C + A'BC + AB'C + ABC \\
 &= A'C(B' + B) + AC(B' + B) \\
 &= A'C + AC \\
 &= C(A' + A) = C
 \end{aligned}$$

$$x + x' = 1$$

Q1.6 Find the complement of $F = wx + yz$ and then show that $FF' = 0$.

$$F = wx + yz$$

$$F' = (wx + yz)' = (wx)'(yz)' = (w' + x')(y' + z')$$

$$\begin{aligned}
 FF' &= (wx + yz)(w' + x')(y' + z') \\
 &= (wx(w' + x') + yz(w' + x'))(y' + z') \\
 &= (wxw' + wx x' + yzw' + yzx')(y' + z') \\
 &= (0 + 0 + yzw' + yzx')(y' + z') \\
 &= yzw'y' + yzw'z' + yzx'y' + yzx'z' \\
 &= 0 + 0 + 0 + 0 = 0
 \end{aligned}$$

Using DeMorgan's theorem

$$(x + y)' = x'y'$$

$$(x \cdot y)' = x' + y'$$

$$x \cdot x' = 0$$

Q1.7 Reduce the following Boolean expression.

$$AB + A(B + C) + B'(B + D)$$

Solution:

$$\begin{aligned}
 AB + A(B + C) + B'(B + D) &= AB + \cancel{AB} + AC + B'B + B'D \\
 &= AB + AC + B'D \\
 &= A(B + C) + B'D
 \end{aligned}$$

Repeated terms are considered once.

$$x \cdot x' = 0$$

Q1.5 Demonstrate by means of truth table the validity of the DeMorgan's theorem for three variables:

$$(xyz)' = x' + y' + z'$$

Solution:

Table 1: Truth Table

Input Variables			Complement of inputs			xyz	(xyz)'	$x' + y' + z'$
x	y	z	x'	y'	z'			
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	1	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	0	1	1
1	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	1	1
1	1	1	0	0	0	1	0	0

Q1.6 Write a truth table for the function, $Z = (A + B)' (AB + C)$.

Solution:

Table 1: Truth Table for Function, Z

Inputs			Outputs				
A	B	C	A + B	(A + B)'	AB	(AB) + C	Z
0	0	0	0	1	0	0	0
0	0	1	0	1	0	1	1
0	1	0	1	0	0	0	0
0	1	1	1	0	0	1	0
1	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0
1	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0

Q1.7 Realize the NAND operation using NOT, AND and OR gates.

Solution:

Table 1: Truth Table of NAND Operation

x	y	Minterm	$(xy)'$
0	0	m_0	1
0	1	m_1	1
1	0	m_2	1
1	1	m_3	0

$$F = m_0 + m_1 + m_2 = x'y' + x'y + xy'$$

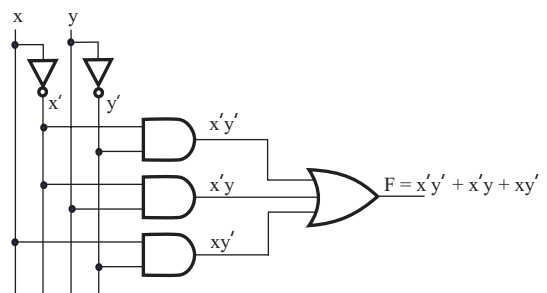


Fig. Q1.7: NAND operation using NOT, AND and OR gates.

Q1.8 Implement the result of $A + A'D + AC'$.

Solution:

Let, F be result of $A + A'D + AC'$ and it can be obtained from the logic circuit shown in Fig. Q1.8.

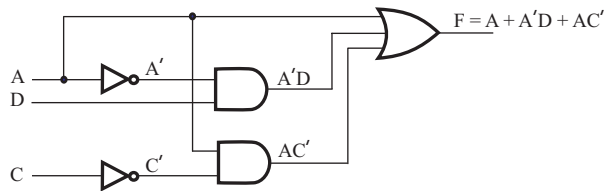


Fig. Q1.8: Logic circuit.

Q1.9 Implement $F = A'B' + A + (B + C)'$ using NAND gates only.

Solution:

$$F = A'B' + A + (B + C)'$$

$$= A'B' + A + B'C' \quad (\text{Using DeMorgan's theorem})$$

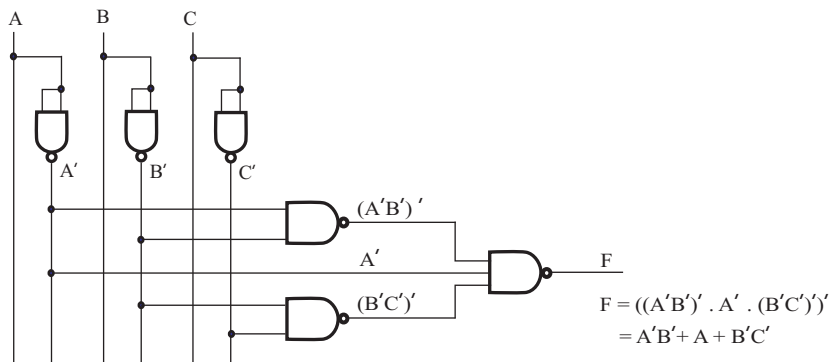


Fig. Q1.9: Logic circuit of F using only NAND gates.

Q1.10 i) When is the output of an NAND operation equal to 1?

ii) When is the output of an NOR operation equal to 1?

From the truth of NAND gate we can say that if any one of the input is 0, then the output of NAND operation is equal to 1.

From the truth table of NOR gate we can say that if all the inputs are 0, then the output of NOR operation is equal to 1.

Table 1: Truth Table of NAND Gates

A	B	$(AB)'$
0	0	1
0	1	1
1	0	1
1	1	0

Table 2: Truth Table of NOR Gates

A	B	$(A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0

Q1.11 Simplify the following Boolean function, $F = x'y' + xy + x'y$.

Solution:

$$\begin{aligned} F &= x'y' + xy + x'y = x'y' + x'y + xy \\ &= x'(y + y') + xy = x'1 + xy \\ &= x' + xy = (x' + x)(x' + y) \\ &= 1(x' + y) = x' + y \end{aligned}$$

$$x + x' = 1$$

Q1.12 Express $F = A + B'C$ in both canonical SOP and POS form.

Solution:

$$F = A + B'C$$

Canonical SOP: It is formed by taking sum of minterms for which $F = 1$ in truth table.

$$\therefore F = \sum (m_1, m_4, m_5, m_6, m_7)$$

$$= A'B'C + AB'C' + AB'C + ABC' + ABC$$

Canonical POS: It is formed by taking product of maxterms for which $F = 0$ in truth table.

$$\therefore F = \Pi (M_0, M_2, M_3)$$

$$= (A + B + C)(A + B' + C)(A + B' + C')$$

Table 1: Truth Table

A B C	Minterm	Maxterm	B'	B'C	F = A + B'C
0 0 0	m ₀	M ₀	1	0	0
0 0 1	m ₁	M ₁	1	1	1
0 1 0	m ₂	M ₂	0	0	0
0 1 1	m ₃	M ₃	0	0	0
1 0 0	m ₄	M ₄	1	0	1
1 0 1	m ₅	M ₅	1	1	1
1 1 0	m ₆	M ₆	0	0	1
1 1 1	m ₇	M ₇	0	0	1

Q1.13 Convert the following Boolean expression into standard SOP form:

$$A'B'C + A'B' + AB'C'D$$

Solution:

$$F = A'B'C + A'B' + AB'C'D$$

$$= A'B'C(D + D') + A'B'(C + C') + AB'C'D$$

$$= A'B'CD + A'B'CD' + (A'B'C)(D + D') + (A'B'C')(D + D') + AB'C'D$$

$$= AB'CD + AB'CD' + A'B'CD + A'B'CD' + A'B'C'D + A'B'C'D' + ABC'D$$

$$(m_{11}) \quad (m_{10}) \quad (m_3) \quad (m_2) \quad (m_1) \quad (m_0) \quad (m_{13})$$

$$= m_0 + m_1 + m_2 + m_3 + m_{10} + m_{11} + m_{13}$$

$$= \sum m(0, 1, 2, 3, 10, 11, 13)$$

Missing literals in first term is D and second term is C and D

Q1.14 Convert the function $Y = A + B'C$ in canonical POS form using Boolean algebra.

Solution:

$$Y = A + B'C = (A + B')(A + C)$$

$$= ((A + B') + (CC'))((A + C) + (BB'))$$

$$= (A + B' + C)(A + B' + C')(A + B + C)(\cancel{A + B' + C})$$

$$(M_2) \quad (M_3) \quad (M_0) \quad (M_2)$$

$$= (A + B + C)(A + B' + C)(A + B' + C')$$

$$= M_0 M_2 M_3$$

Missing literals in first term is C and second term is B.

Repeated maxterms are considered only one time.

Q1.15 Compare SOP and POS.

SOP (Sum-of-Products)	POS (Product-of-Sums)
1. SOP form of Boolean function is formed using minterms. 2. SOP form can be realized by AND operation of literals followed by OR operation of output of AND. 3. SOP form of a Boolean function can be used to realize the function using only NAND gates.	1. POS form of Boolean function is formed using maxterms. 2. POS form can be realized by OR operation of literals followed by AND operation of output of OR. 3. POS form of a Boolean function can be used to realize the function using only NOR gates.

Q1.16 Show that $AB'C + B + BD' + ABD' + A'C = B + C$.**Solution:**

$$\begin{aligned}
 AB'C + B + BD' + ABD' + A'C &= B + BD' + ABD' + A'C + AB'C && \boxed{1 + x = 1} \\
 &= B(1 + D' + AD') + (A' + AB')C && \boxed{x \cdot 1 = x} \\
 &= B \cdot 1 + ((A' + A)(A' + B'))C && \boxed{x + x' = 1} \\
 &= B + (A' + B')C \\
 &= (B + A' + B')(B + C) = (1 + A')(B + C) \\
 &= 1 \cdot (B + C) = B + C
 \end{aligned}$$

Q1.17 Using the Boolean algebra, simplify the expression:

$$A'B'C + (A + B + C')' + A'B'C'D$$

Solution:

$$\begin{aligned}
 A'B'C + (A + B + C')' + A'B'C'D &= A'B'C + \cancel{A'B'C} + A'B'C'D && \boxed{\text{Using DeMorgan's theorem } (x + y)' = x'y'} \\
 &= A'B'C + A'B'C'D && \boxed{\text{Repeated terms are considered once.}} \\
 &= A'B'C(1 + D) + A'B'C'D \\
 &= A'B'C + A'B'CD + A'B'C'D && \boxed{1 + x = 1} \\
 &= A'B'C + A'B'D(C + C') && \boxed{x + x' = 1} \\
 &= A'B'C + A'B'D
 \end{aligned}$$

Q1.18 Show that the dual of the XOR is equal to complement of XOR.**Solution:**

$$F = x \oplus y = xy' + x'y$$

Case i: Direct Complement

$$\begin{aligned}
 F' &= (x \oplus y)' = (xy' + x'y)' && \boxed{\text{Using DeMorgan's theorem}} \\
 &= (xy')' (x'y)' \\
 &= (x' + y)(x + y')
 \end{aligned}$$

Case ii: Duality

$$\begin{aligned}
 x \oplus y &= x \cdot y' + x' \cdot y \\
 &\quad \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \text{Replace by dual elements} \\
 (x' + y) \cdot (x + y') \\
 &\quad \downarrow \downarrow \\
 (x' + y) \cdot (x + y') &= F'
 \end{aligned}$$

Q1.19 Find the complement of the function: $F = x(y'z' + yz)$ by taking their duals and complementing each literal.

Solution:**Case i: Direct Evaluation of Complement**

$$\begin{aligned}
 F &= x(y'z' + yz) \\
 &\quad \downarrow \downarrow \text{Complement} \\
 F' &= (x(y'z' + yz))' = x' + (y'z' + yz)' \\
 &= x' + ((y'z')'(yz)) = x' + ((y + z)(y' + z')) = (x' + y + z)(x' + y' + z')
 \end{aligned}$$

Using DeMorgan's theorem
 $(x + y)' = x' \cdot y'$
 $(x \cdot y)' = x' + y'$

Case ii: Complement using Duality

$$\begin{aligned}
 &x \cdot (y' \cdot z' + y \cdot z) \\
 &\quad \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \text{Replace by dual elements} \\
 &x' + ((y + z) \cdot (y' + z')) \\
 &\quad \downarrow \\
 &(x' + y + z') \cdot (x' + y' + z') = F'
 \end{aligned}$$

$x \cdot x' = 0$

Q1.20 Draw the logic diagram of the following Boolean expression without simplification.

$$BC' + AB + ACD$$

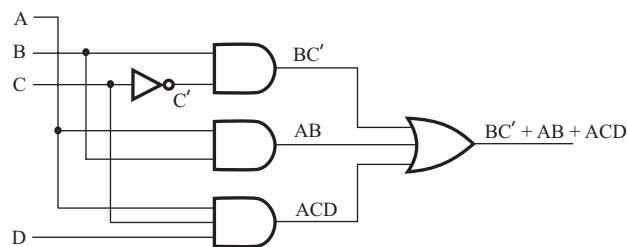
Solution:

Fig. Q1.20: Logic circuit using basic gates.

Q1.21 Draw the multiple-level NAND circuit for the following expression:

$$(AB' + CD')E + BC(A + B)$$

Solution:

$$\begin{aligned}
 (AB' + CD')E + BC(A + B) &= AB'E + CD'E + ABC + BBC \\
 &= AB'E + CD'E + ABC + BC
 \end{aligned}$$

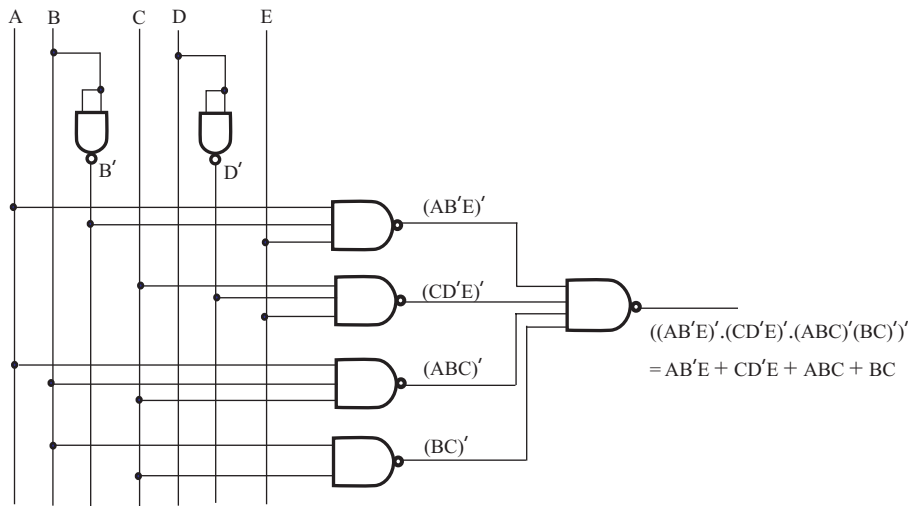


Fig. Q1.21: Logic circuit using multiple level NAND gates.

Q1.22 Simplify the function and obtain minimum SOP for $F(A, B, C, D) = \sum m(0, 2, 3, 10, 11, 14, 15) + \sum d(8, 12)$

Solution:

Minimum SOP form is given by sum of three product terms obtained from K-map.

$$\therefore F = B'D' + AC + B'C$$

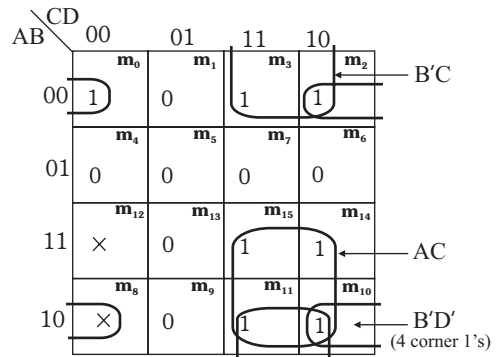


Fig. Q1.22: K-map for F.

Q1.23 Find the minimum POS expression for the function $F(A, B, C) = \Pi M(2, 6, 7)$

Solution:

Minimum POS form is given by complement of sum of two product terms obtained from K-map.

$$\therefore F = (BC' + AB)'$$

$$= (BC')' (AB)'$$

Using DeMorgan's theorem

$$= (B' + C) (A' + B')$$

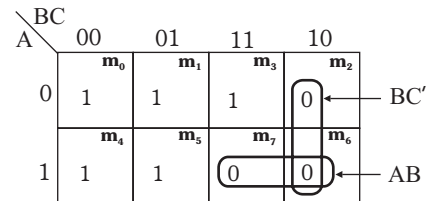


Fig. Q1.23: K-map for F.

Q1.24 Why K-map is arranged in Gray code?

In Gray code, the consecutive codes differ only in value of one bit. In K-map, the minterms are arranged similar to Gray code to identify and group the minterms that differ only by one variable.

Q1.25 Draw the two-variable K-map with single prime implicant formed using single 1's.

Solution:

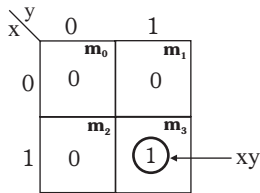


Fig. Q1.25a: K-map for F

$$F = xy$$

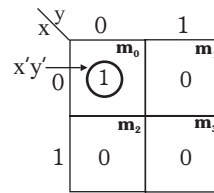


Fig. Q1.25b: K-map for F

$$F = x'y'$$

Q1.26 Draw the two-variable K-map with two prime implicants formed using single 1's.

Solution:

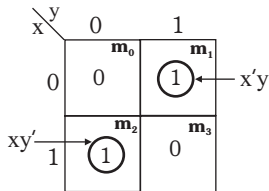


Fig. Q1.26a: K-map for F

$$F = x'y + xy'$$

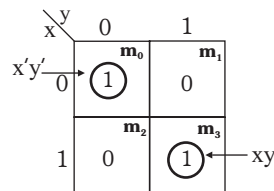


Fig. Q1.26b: K-map for F

$$F = xy + x'y'$$

Q1.27 Define Half adder and Full adder.

Half adder is a combinational circuit that performs arithmetic sum of 2-bit binary.

Full adder is a combinational circuit that performs arithmetic sum of 3-bit binary.

Q1.28 Design a half adder using basic logic gates.

Solution:

$$s = ab' + a'b \quad ; \quad c_o = ab$$

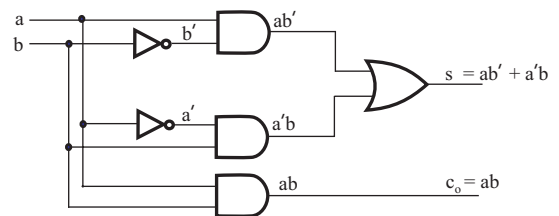


Fig. Q1.28: Logic circuit of half adder.

Q1.29 Give the logical expression for half adder and a full adder.

(AU, Nov/Dec'23, 2 Marks)

Half adder: Sum, $s = ab' + a'b$

Carry, $c_o = ab$

Full adder: Sum, $s = a'b'c_i + a'bc_i + ab'c_i + abc_i$

Carry, $c_o = ab + ac_i + bc_i$

Q1.30 Determine the exact number of half adders and full adders required for performing the addition of two binary numbers of 5 bits length each.

When addition is performed using half adders, we require two half adders to add one bit binary number. Therefore, for addition of 5-bit binary numbers 10 half adders are required.

When addition is performed using full adders we require one full adder to add one bit binary number. Therefore, for addition of 5-bit binary numbers we require 5 full adders.

Q1.31 What is binary parallel adder.

A binary parallel adder performs arithmetic sum of two n-bit binary numbers. A full adder can add two 1-bit binary along with previous carry. Hence, in order to add two n-bit binary numbers, n full adders are required. Each full adder will add one bit of binary numbers.

Q1.32 Define Half subtractor and Full subtractor.

Half subtractor is a combinational circuit that performs arithmetic subtraction of two binary bits.

Full subtractor is a combinational circuit that performs arithmetic subtraction of three binary bits.

Q1.33 Draw the logic diagram and truth table of a half subtractor.

Solution:

Table 1: Truth Table of Half Subtractor

Inputs		Minterm	Outputs	
a	b		d	c _o
0	0	m ₀	0	0
0	1	m ₁	1	1
1	0	m ₂	1	0
1	1	m ₃	0	0

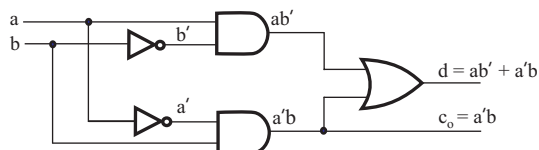


Fig. Q1.33: Logic circuit of half subtractor.

Q1.34 What is a magnitude comparator.

Magnitude comparator is a combinational circuit used to compare two binary numbers and determine whether they are equal or unequal and if unequal then it can make a decision on larger or smaller magnitude.

Q1.35 Define Decoder.

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. In a decoder, for an n-bit binary input one of the 2^n output is activated.

Q1.36 How multiplexer differ from decoder?

A decoder accept n-bit binary input to activate one of the 2^n outputs whereas the multiplexer accept n-bit address to allow one of the 2^n input on the output line.

Q1.37 Define Encoder.

An encoder is a combinational circuit that perform the inverse operation of a decoder. An encoder has 2^n input lines and n output lines. When one of 2^n inputs is activated, it generates an unique n-bit output.

Q1.38 What is priority Encoder.

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are asserted high (or equal to 1) then the output corresponds to the input having highest priority.

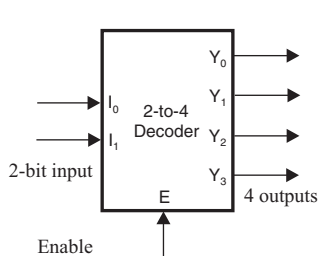
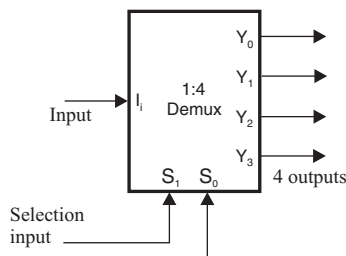
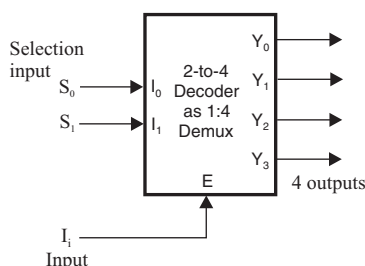
Q1.39 What is Demultiplexer.

Demultiplexer is a combinational circuit which can transmit a binary input to any one of 2^n output lines using n selection lines or n -bit address.

Q1.40 Convert a 2-to-4 decoder with enable input to 1:4 demux.**Solution:**

The truth table of 2-to-4 decoder working as 1:4 MUX is shown in Table 1.

When the 2 inputs I_0, I_1 of decoder are used as selection inputs S_0 and S_1 of demux and enable of decoder used as demux input, a 2-to-4 decoder will work as a 1:4 demux.

**Fig. Q1.40.1: 2-to-4 decoder.****Fig. Q1.40.2: 1:4 demux.****Fig. Q1.40.3: 2-to-4 decoder as 1:4 demux.****Table 1: Truth Table of 2-to-4 Decoder with 1:4 Demux**

Inputs	Selection Inputs		Outputs			
$I_i = E$	S_1	S_0	Y_0	Y_1	Y_2	Y_3
$I_i = 0$	0	0	0	0	0	0
$I_i = 0$	0	1	0	0	0	0
$I_i = 0$	1	0	0	0	0	0
$I_i = 0$	1	1	0	0	0	0
$I_i = 1$	0	0	I_i	0	0	0
$I_i = 1$	0	1	0	I_i	0	0
$I_i = 1$	1	0	0	0	I_i	0
$I_i = 1$	1	1	0	0	0	I_i

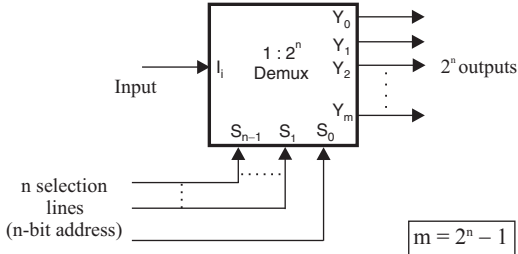
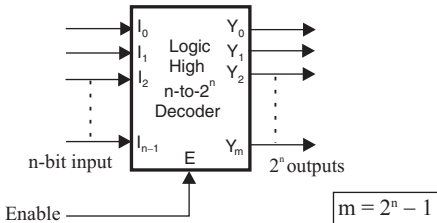
Q1.41 Differentiate between multiplexer and demultiplexer.

Multiplexer	Demultiplexer
<ol style="list-style-type: none"> 1. A multiplexer is a combinational circuit that select one binary information from many inputs. 2. In general, a MUX will have 2^n inputs and one output. In order to select one of the 2^n inputs the MUX will have n selection lines or address. 	<ol style="list-style-type: none"> 1. A demultiplexer will perform the reverse operation of multiplexer. 2. A demux is a combinational circuit which can transmit a binary input of any one of 2^n output lines using n selection lines or n-bit address.

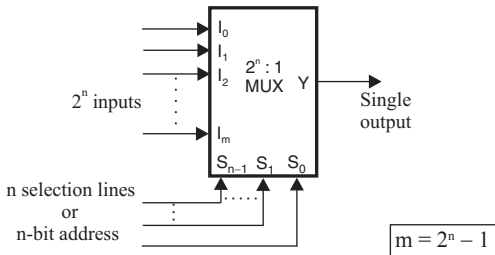
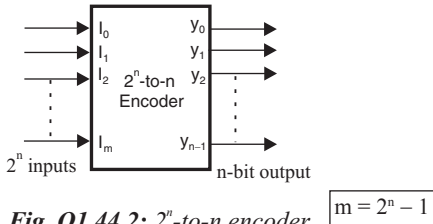
Q1.42 How many selection inputs, data inputs and outputs for 16:1 multiplexer?**(AU, Nov/Dec'23, 2 Marks)**

A 16:1 multiplexer has 16 data inputs (I_0 to I_{15}), 4 selection inputs (S_0 to S_3) and One output (Y).

Q1.43 Compare demux and decoder.

Demultiplexer	Decoder
<p>1. A demultiplexer is a combinational circuit, which can transmit a binary input to any one of 2^n output lines using n-bit address or 'n' selection lines.</p>  <p>Fig. Q1.43.1: Block diagram of demux.</p>	<p>1. A decoder is a combinational logic device that decodes a n-bit binary input to one of the 2^n binary information.</p>  <p>Fig. Q1.43.2: Block diagram of decoder.</p>
2. Two types of inputs: input and selection lines.	2. Two types of inputs: input and enable.
3. A demux is a single input to multi output device based on selection input.	3. A decoder, decodes n -bit code to 2^n binary information.

Q1.44 Compare multiplexer and encoder.

Multiplexer	Encoder
<p>1. A multiplexer is a combinational circuit, which can select one output from 2^n inputs, using 'n' selection lines.</p>  <p>Fig. Q1.44.1: Block diagram of $2^n:1$ multiplexer.</p>	<p>1. An encoder is a combinational circuit that can generate a unique n-bit output for each of 2^n binary inputs.</p>  <p>Fig. Q1.44.2: 2^n-to-n encoder.</p>
2. A MUX is multi inputs to single output device based on selection input.	2. An encoder, generate n -bit code for each of 2^n binary inputs.

Q1.45 Differentiate between multiplexer and decoder.

Multiplexer	Decoder
<p>1. MUX accepts several inputs and allow only one data output.</p> <p>2. Select line are used to select data inputs and allow only one of them.</p> <p>3. Multiplexer converts the unary code into binary code.</p>	<p>1. It takes n input binary code and convert it into a corresponding outputs.</p> <p>2. Enable inputs are used to control the operation of the decoder.</p> <p>3. Decoder converts binary code into unary.</p>

1.16 Exercises

I. Fill in the blanks

1. In combinational circuits, there is no _____ from output to input.
2. Digital logic circuits with feedback are known as _____ circuits.
3. The two voltage levels of logic circuits are called _____ and _____.
4. NAND and NOR gates are called _____.
5. The realizations in SOP and POS forms are called _____.
6. A gate in which any one input must be low to get a high output is called _____.
7. A gate in which any one input must be high to get a low output is called _____.
8. Binary _____ can be performed via addition by using the 1's or 2's complement method.
9. The _____ gate output is high when the inputs are not the same.
10. The output of a 2-input _____ gate is 0 if and only if its inputs are unequal.
11. The most suitable gate for comparing two bits is _____.
12. The _____ of a sum term is equal to the product of the complements of the variables in the sum term.
13. The operational symbol for exclusive-OR operation is _____.
14. AND-OR logic produces the output of an expression in _____ form.
15. Binary logic consists of binary variables and a set of _____.
16. NOT operation is same as _____.
17. Boolean function can be realized using only _____ or _____ gates.
18. The realizations in SOP and POS standard form are called _____.
19. The Two-level realization of SOP form of a Boolean function can be obtained using only _____.
20. Two-level realization of POS form of a Boolean function can be obtained using only _____.
21. A 3-variable Karnaugh map has _____ cells.
22. Each variable within a term of Boolean expression is called _____.
23. The Boolean Expression can be reduced or simplified using postulates and theorems of _____.
24. _____ is a pictorial form of truth table and used to simplify Boolean functions.
25. The literals of minterms are split and arranged as _____ and _____.
26. Prime implicant with single 1 represents _____ literal product term.
27. All minterms are 1's then all squares of K-map will be filled by 1 and function value is _____.
28. Two-variable, K-map will have _____ squares.
29. Three-variable, K-map will have _____ squares.
30. Four-variable, K-map will have _____ square.
31. _____ is a combinational circuit that performs arithmetic sum of three 1-bit binary.
32. _____ is a combinational circuit that performs arithmetic sum of two 1-bit binary.
33. The two outputs of a half adder are _____ and _____.
34. In a half adder, the sum can be realized using _____ gate.
35. In a half adder, the carry can be realized using _____ gate.
36. Full adder is a combinational circuit that performs arithmetic sum of _____ binary bits.

37. A full adder has _____ outputs.
38. Practically _____ is considered as addition of positive and negative numbers.
39. _____ is a combinational circuit used to compare two binary numbers.
40. Adders have been developed to perform _____ operation on binary numbers.
41. Half subtractor can perform subtraction of _____ 1-bit binary number.
42. A _____ performs arithmetic sum of two-n-bit binary numbers.
43. The logic circuit of half subtractor using _____ gate.
44. A half subtractor has _____ outputs.
45. A full subtractor has _____ inputs.
46. A decoder decodes n-bit binary information into _____ binary information.
47. An encoder performs the _____ operation of a decoder.
48. The digital multiplexer is basically a combination logic to perform the _____ operation.
49. The multiplexer is otherwise called _____.
50. A multiplexer will have _____ inputs and _____ output.
51. A demultiplexer will perform the _____ operation of multiplexer.
52. A demultiplexer is shortly called as _____.
53. Demultiplexers of smaller size can be connected in _____ to expand the size of demux.
54. An encoder has _____ input lines and _____ output lines.
55. The operation the priority encoder is such that if two or more inputs are asserted _____.
56. A multiplexer is shortly called as _____.
57. The 4:1 multiplexer is used to select one of the four inputs using _____ selection input.
58. The 8:1 multiplexer is used to select one of the eight inputs using _____ selection input.
59. Multiplexer of smaller size can be connected in _____ to expand the size of multiplexer.
60. The data distributor is otherwise called as _____.

Answers

- | | | | |
|------------------------|--------------------------------|---------------------------|--------------------------|
| 1. feedback | 16. complement operation | 31. full adder | 46. 2 ⁿ |
| 2. sequential | 17. NAND, NOR | 32. half adder | 47. inverse |
| 3. high, low | 18. two-level realization form | 33. sum, carry | 48. AND-OR |
| 4. universal gates | 19. NAND gates | 34. half adder | 49. data selector |
| 5. standard form | 20. NOR gates | 35. XOR | 50. 2 ⁿ , one |
| 6. NAND gate | 21. 8 | 36. AND | 51. reverse |
| 7. NOR gate | 22. literal | 37. two | 52. demux |
| 8. subtraction | 23. Boolean algebra | 38. subtraction | 53. parallel |
| 9. exclusive-OR | 24. K-map | 39. magnitude comparator | 54. 2 ⁿ and n |
| 10. XNOR | 25. rows, columns | 40. arithmetic addition | 55. high |
| 11. XOR | 26. n literal | 41. two | 56. mux |
| 12. Complement | 27. 1 | 42. binary parallel adder | 57. 2-bit |
| 13. \oplus | 28. 4 | 43. XOR | 58. 3-bit |
| 14. SOP | 29. 8 | 44. two | 59. cascade |
| 15. logical operations | 30. 16 | 45. three | 60. demux |

II. State whether the following statements are True or False

1. The output of NOR gate is high if and only if all its inputs are low.
2. AND gate can be used as an inverter.
3. $A + A'B = A + B$.
4. The complement of a product term is equal to the sum of the complement of the variables in the product term.
5. An example of a product-of-sums expression is $A(B + C) + AC'$.
6. The expression $A'BCD + ABCD' + AB'C'D$ cannot be simplified.
7. An inverter performs an operation known as complement.
8. Boolean algebra is an algebraic structure.
9. $(X + Y)' = X' + Y'$.
10. The basic logic gates are AND, OR and NOT gates.
11. The NAND gate is a combination of OR followed by NOT gate.
12. Logic gates can also be designed to work with negative logic levels.
13. Binary variables are denoted by either lower case or upper case alphabets.
14. Boolean function can't be evaluated for all possible combinations of binary values of the variables of the function.
15. Minterms are 2^n possible combinations of AND terms with n variables such that the logical AND of all the variables is 0.
16. On a Karnaugh map grouping the 0's produces AND-OR logic.
17. The Boolean expression can be reduced or simplified using postulates and theorems of Boolean algebra.
18. There are one methods have standard procedure for simplification of Boolean functions.
19. Maxterm can be used to construct K-map in which each squares represent a minterm.
20. Arranging the literal/ minterms only one change is allowed if we move from one row to next row or from one column to next column.
21. While forming prime implicants any number of overlapping is allowed in horizontal and vertical directions.
22. The simplified Boolean function is sum of all the product terms of prime implicant
23. If all minterms are 1's then all squares of K-map will be filled by 1 and function value is 0.
24. Reducing the number of literals in a Boolean expression or function will simplify the implementation of function by logic gates with minimum number of gates.
25. The simplified Boolean function using K-map is not unique. Sometimes there may be multiple solutions.
26. A full adder is characterized by two inputs and two outputs.
27. A 4-bit parallel adder can add two 4-bit binary numbers.
28. A comparator compares for the equality of two input numbers.
29. Half adder has two inputs and two outputs.
30. In addition of n -bit binary numbers, the addition is performed bit-by-bit.
31. n -bit binary addition can be performed by using n full adder in parallel.
32. Subtractor have been developed to perform arithmetic subtraction operation on binary numbers.
33. In the subtractor 2-bit result, first bit is called borrow and the second bit is called difference.
34. The addition of negative number and 2's complement of positive number will give the result of subtraction.
35. The BCD adder is a binary adder with additional logic circuit to perform addition of correction 6_{10} when the sum of a BCD digit exceeds 9_{10} .

36. Full subtractor is a combinational circuit that perform arithmetic subtraction of two binary bits in which one of the bit is borrow generated in previous subtraction.
37. The input to full adders has to be modified for different arithmetic and logical operations.
38. Half subtractor has two inputs and two outputs.
39. Full subtractor has two inputs and two outputs.
40. In general, a multiplexer has several data inputs, several data outputs and selection inputs.
41. A comparator compares for the equality of two input numbers.
42. A demultiplexer can transmit a binary input to any one of 2^n output lines using n -bit address.
43. Demultiplexer of smaller size can be connected in series to expand the size of demultiplexer.
44. In a decoder, n -bit binary information is decoded into 2^n binary information.
45. A n -to- 2^n decoder outputs may be logic **high** or **low**.
46. An encoder is a combinational circuit that performs the reverse of decoder function.
47. A priority encoder is a decoder.
48. A demultiplexer is a data distributor.
49. The code 10011000 exhibits even parity.
50. In combinational circuits the output at any time depends on input at that time.
51. In combinational circuits there is a storage element and there is a feedback from output to input.
52. The code converters are combinational circuits that convert one type of code to another type of codes.
53. The design of code converter starts with formation of truth table by taking one type of code as inputs and another type of code as outputs.
54. A decoder is a combinational logic device that decodes one of the 2^n binary information depending on n -bit binary input.
55. The n -bit binary information is decoded into n binary information.
56. The complement of logic high decoder output will be same as output of logic low decoder.
57. The complement of output of logic high decoder can be used to realize POS form of Boolean function.
58. A multiplexer is otherwise called data distributor.
59. 2:1 Multiplexers can be connected in series to form 4:1 multiplexer or multiplexers of higher order.
60. The logic circuit of Boolean functions in SOP form can be realised using multiplexer.

Answers

1. True	11. False	21. True	31. True	41. True	51. False
2. False	12. True	22. True	32. True	42. True	52. False
3. True	13. True	23. False	33. False	43. False	53. True
4. True	14. False	24. True	34. False	44. True	54. True
5. False	15. False	25. True	35. True	45. True	55. True
6. True	16. False	26. False	36. False	46. True	56. False
7. True	17. True	27. True	37. True	47. False	57. True
8. True	18. False	28. True	38. True	48. True	58. True
9. False	19. False	29. True	39. False	49. False	59. True
10. True	20. True	30. True	40. False	50. True	60. True

1. The Boolean expression $A'B' + (AB)' + AB$ is equivalent to

2. The Boolean expression $A \oplus B \oplus B$ is equivalent to

3. In the logic circuit shown in Fig. 3, the output F is

-

Fig. 3.

- a) 6 b) 12 c) 64 d) 32

- a) $X \text{ NAND } X$ b) $X \text{ NOR } X'$ c) $X \text{ NAND } 1$ d) $X \text{ NOR } 1$

- a) 2 b) 3 c) 4 d) 5

- a) $Y = A$ b) $Y = AB$ c) $Y = AC$ d) $Y = C'(A \oplus B)$

-

Fig. 8.

-

Fig. 9.

- a) NAND-NOR realization
 - b) NOR-NOR realization
 - c) NOR-NAND realization
 - d) NAND-NAND realization

11. The number of minterms in the function $f(x, y, z) = xy' + z'$ is

- a) 4 b) 5 c) 3 d) 6

12. If $X = 1$ in the logic equation, $(X + Z(Y' + (Z' + XY')))(X' + Z'(X + Y)) = 1$, then

- a) $Y = Z$ b) $Y = Z'$ c) $Z = 1$ d) $Z = 0$

13. The circuit shown in Fig. 13 realises the function

- a) $(A' + B').C + (DE)'$
 b) $(A + B)(C + D + E)$
 c) $AB + C + DE$
 d) $AB + C(D + E)$

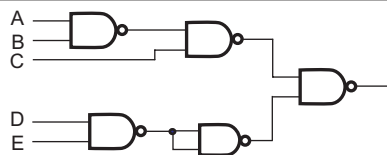


Fig. 13.

14. In the Fig. 14 X_0, X_1, X_2 will be 1's complement of ABC if

- a) $Y = 1$
 b) $Y = 0$
 c) $Y = A' = B' = C'$
 d) $Y = A = B = C$

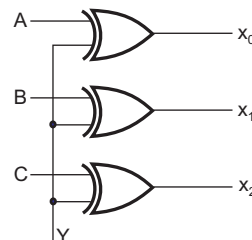


Fig. 14.

15. The circuit in Fig. 15 works as

- a) XOR
 b) XNOR
 c) NOR
 d) NAND

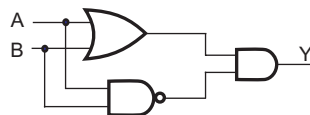


Fig. 15.

16. The Boolean function $A + BC$ is a reduced form of

- a) $AB + BC$ b) $(A + B)(A + C)$ c) $A'B + AB'C$ d) $(A' + C)B$

17. The number of distinct Boolean expression of 4 variables is

- a) 16 b) 256 c) 1024 d) 65536

18. The complete set of only those logic gates designated as universal gates is

- a) NOT, OR and AND gates b) XNOR, NOR and NAND gates
 c) NOR and NAND gates d) XOR, NOR and NAND gates

19. The logical expression $Y = A + A'B$ is equivalent to

- a) $Y = AB + A$ b) $Y = A'B + A'$
 c) $Y = AB' + B$ d) $Y = AB + A'$

20. The minimized form of logical expression $Y = A'B'C' + A'B'C + A'B'C + ABC'$ is

- a) $A'C' + BC' + A'B$ b) $AC' + B'C + A'B$
 c) $A'C + B'C + A'B$ d) $AC' + B'C + AB'$

21. For the identity $AB + AC + BC = AB + AC$ the dual form is

- a) $(A + B)(A' + C)(B + C) = (A + B)(A' + C)$
- b) $(A' + B')(A' + C')(B' + C') = (A' + B')(A' + C')$
- c) $(A + B)(A' + C)(B + C) = (A' + B')(A + C')$
- d) $A'B' + AC' + B'C' = A'B' + AC'$

22. The number of boolean functions that can be generated by n variable is equal to

- a) 2^{2^n}
- b) 2^{2n}
- c) 2^{n-1}
- d) 2^n

23. The Boolean expression $AC + BC'$ is equivalent to

- a) $A'C + BC' + AC$
- b) $B'C + AC + BC' + A'CB'$
- c) $AC + BC' + B'C + ABC$
- d) $ABC + A'BC' + ABC' + AB'C$

24. For the logic circuit in Fig. 24 the required input condition (x, y, z) to make the output $F = 1$ is

- a) 1, 0, 1
- b) 0, 0, 1
- c) 1, 1, 1
- d) 0, 1, 1

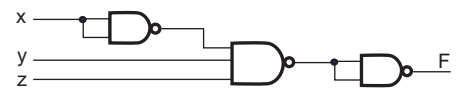


Fig. 24.

25. For the logic circuit, the simplified Boolean expression for the output F is

- a) $x + y + z$
- b) x
- c) y
- d) z

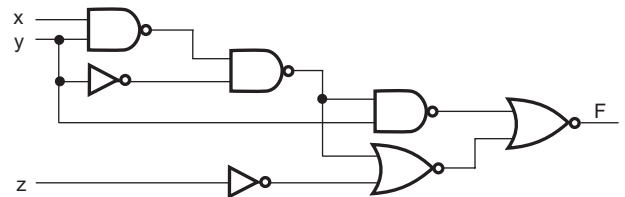


Fig. 25.

26. The minimum number of 2 input NAND gates required to implement the Boolean function $AB'C$ assuming that A , B and C are available is

- a) 2
- b) 3
- c) 5
- d) 6

27. For the logic circuit shown in Fig. 27 the output F is

- a) $(xyz)'$
- b) $x' + y' + z'$
- c) $(xy)' + (yz)' + x' + z'$
- d) $x' + z'$

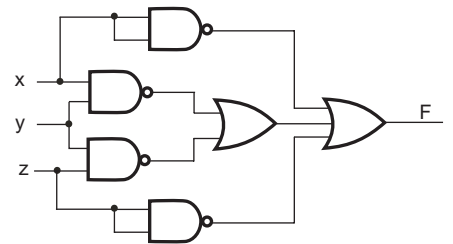


Fig. 27.

a) $A'B' + A'B$
c) $(A' + B)(A + B)$

b) $(AB)' + AB$
d) $(A' + B')(A + B)$

a) a NAND or an XOR gate b) a NOR or an XNOR gate
c) an OR or an XNOR gate d) an AND or an XOR gate

a) 0 b) 1 c) 4 d) 7

A logic diagram showing an XOR gate. The top input is labeled 'A' and the bottom input is labeled '0'. The output of the gate is a single line extending to the right.

Fig. 31.

a) $Z = A$ b) $Z = 1$ c) $Z = 0$ d) $Z = A$

Diagram illustrating the implementation of the XOR operation using two NOT gates and one OR gate:

Inputs A and B are connected to two NOT gates. The outputs of the NOT gates are connected to the inputs of an OR gate. The output of the OR gate is labeled Y = C.

Fig. 33.

a) $A'D + B'C'D$
c) $(A' + D)(B'C + D')$

a) $x' + z$
b) $y + z$
c) $y' + z'$
d) $y' + z$

Fig. 36.

The diagram shows a logic circuit for a 4-input majority gate. It consists of three 2-input AND gates and one 3-input AND gate. The first 2-input AND gate has inputs A and B. The second 2-input AND gate has inputs B and C. The third 2-input AND gate has inputs C and D. The outputs of these three 2-input AND gates are connected to the three inputs of the 3-input AND gate. The output of the 3-input AND gate is labeled F.

Fig. 37.

a) $A' + (B + C') (B' + C') (B' + C)$
b) $A' (BC' + B'C' + B'C)$
c) $C (B'A + CA + CA')$
d) $AB'C + ABC + ABC'$

39. The expression $(A + A'B)(B' + BA)$ can be simplified to

- a) $AB' + A'B$ b) $AB' + A'B'$ c) 1 d) A

40. The circuit in Fig. 40 is works as

- a) XOR b) XNOR
c) NOR d) NAND

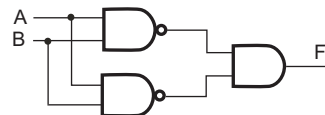


Fig. 40.

41. Each variable within a term of a Boolean expression is called

- a) Literal b) Minterm c) Maxterm d) Prime implicant

42. A prime implicant with single 1 represent

- a) n literal product term b) $n - 1$ literal product term
c) $n - 2$ literal product term d) $n - 3$ literal product term

43. A K-map is a diagram made up of squares with each square representing

- a) Literal b) Minterm c) Maxterm d) Prime implicant

44. When minterms are considered for simplification using K-map the resultant Boolean function will be in

- a) Sum-of-product form b) Product-of-sum form
c) MSOP form d) Prime implicants

45. For n -variable Boolean function the K-map will be

- a) $2n$ square b) 2^n square c) n square d) 2^{2n} square

46. A prime implicant with two 1's represent

- a) n literal product term b) $n - 1$ literal product term
c) $n - 2$ literal product term d) $n - 3$ literal product term

47. While forming prime implicants any number of overlapping is allowed in

- a) Horizontal directions b) Vertical directions
c) Both (a) and (b) d) None of the above

48. Undefined function outputs are called

- a) Literal b) Minterm c) Maxterm d) Don't-care

49. The expansion phase is the first phase in which minterms are obtained from truth table are prime implicants are identified after eliminating

- a) Literal b) $n - 1$ literal c) $n - 2$ literal d) Redundant literals

50. The literals of minterm are split and arranged

- a) Only rows b) Only columns
c) Both (a) and (b) d) None of the above

51. The K-map for a Boolean function is given. The number of essential prime implicants for this function is

- a) 4
- b) 5
- c) 6
- d) 8

AB \ CD	00	01	11	10
	m_0	m_1	m_3	m_2
00	1	1	0	1
01	m_4	m_5	m_7	m_6
11	1	0	0	0
10	m_8	m_9	m_{11}	m_{10}
	1	0	0	1

Fig. 51.

52. The minimum POS expression for K-map given in the Fig. 52 is

- a) $(B + C)(C' + D)$
- b) $(B + D)(A + C)$
- c) $(A' + C')(C + D')$
- d) $(B + D)(A + D')$

AB \ CD	00	01	11	10
	m_0	m_1	m_3	m_2
00	0	0	1	0
01	m_4	m_5	m_7	m_6
11	1	1	1	0
10	m_{12}	m_{13}	m_{15}	m_{14}
	0	0	1	0

Fig. 52.

53. While obtaining minimal SOP expression

- a) All don't-cares are ignored
- b) All don't-cares are treated as logic ones
- c) All don't-cares are treated as logic zeros
- d) Only such don't-cares that aid minimization are treated as logic ones

54. The given logic diagram in Fig. 54 represents a

- a) Multiplexer
- b) Full adder
- c) Half adder
- d) None of the above

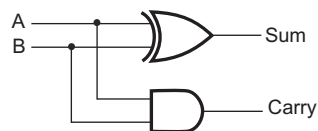


Fig. 54.

55. An instruction used to set the carry flag in a computer can be classified as

- a) data transfer
- b) arithmetic
- c) logical
- d) program control

56. Two 2's complement number having sign bit x and y are added and the sign bit of the result is 2. If z is input carry, then the occurrence of overflow is indicated by the Boolean function,

- a) xyz
- b) $x'y'z'$
- c) $x'y'z + xyz'$
- d) $xy + yz + zx$

57. The circuit in Fig. 57 works as

- a) a full subtractor
- b) a full adder
- c) a binary to gray converter
- d) a gray to binary converter

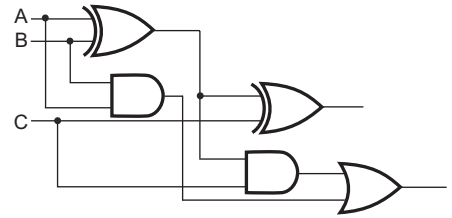


Fig. 57.

58. For a binary half subtractor having 2 input A and B, the correct set of logical expression for the outputs $D = (A - B)$ and $X = (\text{Borrow})$ are,

- a) $D = AB + A'B$; $X = A'B$
- b) $D = A'B + AB' + AB'$; $X = AB'$
- c) $D = A'B + AB'$; $X = A'B$
- d) $D = AB + A'B$; $X = AB'$

59. A half adder is characterized by

- a) two inputs and two outputs
- b) three inputs and two outputs
- c) two inputs and three outputs
- d) two inputs and one output

60. A full adder is characterized by

- a) two inputs and two outputs
- b) three inputs and two outputs
- c) two inputs and three outputs
- d) two inputs and one output

61. A full adder can add

- a) two 1-bit binary number
- b) two 2-bit binary number
- c) two 4-bit binary number
- d) four bits at a time

62. A 4-bit parallel adder can add

- a) two 2-bit binary number
- b) two 4-bit binary number
- c) two 1-bit binary number
- d) four bits at a time

63. Parallel adder are

- a) combinational circuits
- b) sequential circuits
- c) both of the above
- d) None of the above

64. The following switching functions are to be implemented using a decoder

$$f_1 = \sum m(1, 2, 4, 8, 10, 14) ; f_2 = \sum m(2, 5, 9, 11) ; f_3 = \sum m(2, 4, 5, 5, 7)$$

The minimum configuration of the decoder should be

- a) 2 to 4 line
- b) 3 to 8 line
- c) 4 to 15 line
- d) 5 to 32 line

65. The circuit of Fig. 65 is equivalent to,

- a) AND gate
- b) OR gate
- c) XOR gate
- d) XNOR gate

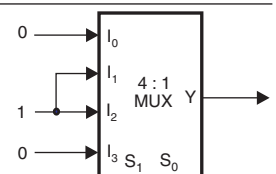


Fig. 65.

66. The minimum number of 2×1 MUX needed (with no added gates) to form a 3×1 MUX is

- a) 2 b) 3 c) 4 d) 1

67. The circuit in Fig. 67 works as

- a) binary to Gray converter
b) Gray to binary converter
c) odd parity generator
d) even parity generator

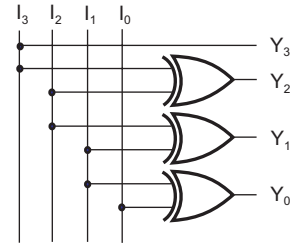


Fig. 67.

68. The function "F" implemented by the multiplexer shown in Fig. 68 is

- a) A
b) B
c) $A'B$
d) $AB' + A'B$

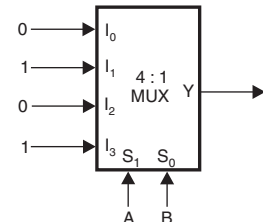


Fig. 68.

69. The MUX in Fig. 69 implements the function

- a) $F = BC'$
b) $F = BC$
c) $F = B'C'$
d) $F = B'C$

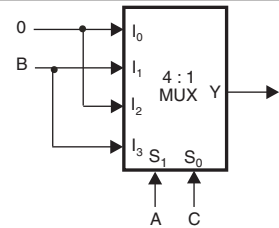


Fig. 69.

70. The circuit shown in Fig. 70 is a

- a) Odd parity generator b) Even parity generator
c) Full adder d) Comparator

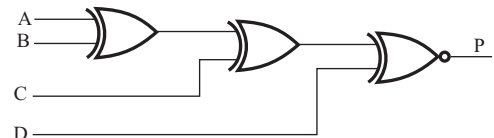


Fig. 70.

71. The multiplexer in Fig. 71 implements the function

- a) $Y = AB$
b) $Y = A'B + B'C$
c) $Y = A + BC$
d) $Y = BC$

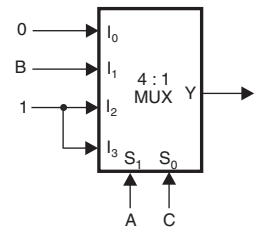


Fig. 71.

72. The MUX in Fig. 72 implements the function

- a) $Y = \sum m(1, 3, 5, 7)$
b) $Y = \sum m(1, 2, 3, 5)$
c) $Y = \sum m(2, 5, 5, 7)$
d) $Y = \sum m(1, 4, 5, 5)$

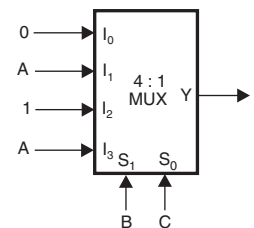


Fig. 72.

73. The MUX in Fig. 73 implements the function

- a) B'
- b) B
- c) $B \oplus C$
- d) $B \odot C$

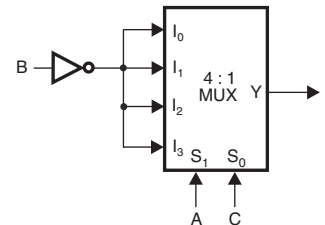


Fig. 73.

74. The output of the 4:1 MUX shown in Fig. 74 is

- a) $x'y' + x$
- b) $x + y$
- c) $x' + y'$
- d) $xy + x'$

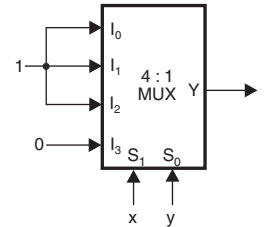


Fig. 74.

75. The boolean function realized by the given multiplexer is given by,

- a) $\sum m(0, 3, 5, 7, 8, 11, 12, 14)$
- b) $\sum m(0, 3, 5, 7, 12, 14)$
- c) $\sum m(2, 4, 5, 9, 10, 13, 15)$
- d) $\sum m(0, 1, 5, 7, 9, 12)$

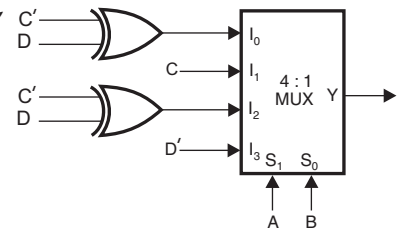


Fig. 75.

76. The boolean function implemented by the MUX is given as,

- a) $\sum m(0, 3, 4, 7)$
- b) $\prod M(3, 4, 7)$
- c) $\prod M(1, 2, 5, 5)$
- d) None of the above

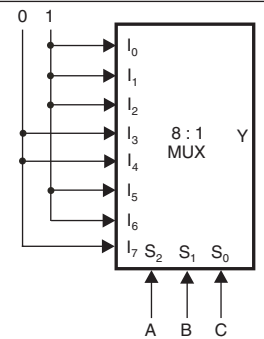


Fig. 76.

77. The circuit perform

- a) OR function
- b) AND function
- c) NOR function
- d) Inverter

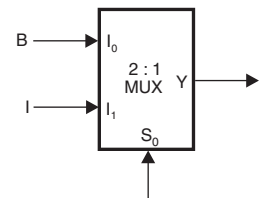


Fig. 77.

78. Without any additional circuitry, an 8 : 1 MUX can be used to obtain

- a) Some but not all boolean function of 3 variables
- b) All functions of 3 variables but none of 4 variables

- c) All functions of 3 variables and some but not all of 4 variables
 d) All functions of 4 variables

79. The minimum number of 2:1 MUX required to realize a 4:1 MUX is

- a) 1 b) 2 c) 3 d) 4

80. A digital system is required to amplify a binary encoded audio signal, the user should be able to control the gain of the amplifier from a minimum to a maximum in 248 increments. The minimum number of bits required encode in straight binary is

- a) 8 b) 5 c) 5 d) 7

Answers

1. c	11. b	21. b	31. d	41. a	51. a	61. a	71. c
2. c	12. d	22. b	32. c	42. a	52. a	62. b	72. c
3. b	13. a	23. d	33. d	43. b	53. d	63. a	73. a
4. a	14. a	24. d	34. a	44. a	54. c	64. c	74. c
5. a	15. a	25. c	35. b	45. b	55. b	65. c	75. a
6. b	16. b	26. c	36. b	46. b	56. d	66. a	76. b
7. d	17. b	27. b	37. a	47. c	57. b	67. a	77. a
8. a	18. c	28. d	38. a	48. d	58. c	68. b	78. d
9. a	19. c	29. b	39. d	49. d	59. a	69. b	79. c
10. d	20. a	30. a	40. d	50. c	60. b	70. a	80. a

IV. Answer the following questions

1. Explain absorption theorem with truth table.
2. Which are universal gates, why are they called so?
3. What are minterms and maxterms?
4. What is the difference between canonical and SOP/POS form?
5. What is truth table. Give an example.
6. What is duality?
7. What are postulates of Boolean algebra. Give two examples.
8. What is DeMorgan's theorem.
9. How will be the resultant function of K-map when minterm and maxterm are used?
10. How do we handle don't-care conditions?
11. What are the methods available to simplify a Boolean expression?
12. Explain K-map.
13. Define expansion phase and covering phase.
14. Realize half adder using K-map.
15. Draw logic circuit and truth table of full adder.
16. Explain the operation of half subtractor.
17. Realize full subtractor using K-map.

18. Design full subtractor circuit. Write a truth table.
19. Explain the working of 4-bit parallel adder.
20. Design binary adder/subtractor using full adders.
21. Draw a logic circuit of 2-bit magnitude comparator.
22. Realize the Boolean function $F(A, B, C, D) = \sum(0, 1, 5, 5)$ using appropriate multiplexer.
23. What is the purpose of enable input in a decoder?
24. How can we derive a priority encoder from an encoder?

V. Solve the following problems

E1.1 Find the complement of the following expression:

a) $x'y + x'y'$ b) $xyz + x'y + xyz'$

E1.2 Given two Boolean functions F_1 and F_2 , show that (a) the Boolean function $E = F_1 + F_2$ contains the sum of the minterms of F_1 and F_2 . (b) The Boolean function $G = F_1 F_2$ contains only the minterms that are common to F_1 and F_2 .

E1.3 List the truth table of the function: a) $F_1 = x'y' + x'y + yz'$ b) $F_2 = b'c' + ac$

E1.4 Demonstrate the validity of the following identities by means of truth tables:

DeMorgan's theorem for 3 variables: a) $(x + y + z)' = x'y'z'$ b) $(xyz)' = x' + y' + z'$

The associate law: c) $x + (y + z) = (x + y) + z$ d) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

E1.5 Express the complement of the following functions in sum of minterms form,

a) $F(A, B, C, D) = \sum m(3, 5, 9, 11, 15)$ b) $F(x, y, z) = \prod M(2, 4, 5, 7)$

E1.6 Convert each of the following expression into sum of products and product of sums.

a) $(AB + C)(B + C'D)$ b) $x + x'(x' + y')(y' + z')$

E1.7 Show that the complement of XOR is equal to XNOR.

E1.8 Determine the complement of Boolean expression $(b + d)(a' + b' + c)$ and show that POS becomes SOP.

E1.9 Determine the complement of Boolean expression $a'b + a'c' + abc$ and show that SOP becomes POS.

E1.10 Implement the Boolean function, $F = xy + x'y + y'z'$

- a) With AND, OR and NOT gates. b) With OR and NOT gates.
c) With AND and NOT gates. d) With NAND and NOT gate. e) With NOR and NOT gates.

E1.11 Obtain the truth table of the following functions and express each function in sum of minterms and product of maxterms form:

a) $(xy + z)(y + xz)$ b) $(x + y')(y' + z)$ c) $x'z + wx'y + wyz' + w'y'$ d) $(xy + yz' + x'z)(x + z)$

E1.12 For the Boolean function, $F = x'yz + xyz' + wx'y' + w'xy + wxy$

- a) Obtain the truth table of F.
- b) Draw the logic diagram, using the original Boolean expressions.
- c) Use Boolean algebra to simplify the function to a minimum number of literals.
- d) Obtain the truth table of the function from the simplified expression and show that it is the same as the one in part (a).
- e) Draw the logic diagram from the simplified expression and compare the total number of gates with the diagram in part (b).

E1.13 Simplify the following Boolean functions using 3-variable K-maps:

a) $F_1(x, y, z) = \sum m(0, 1, 5, 7)$ b) $F_2(x, y, z) = \sum m(2, 3, 4, 5)$

E1.14 Simplify the following Boolean functions using K-maps:

a) $F_1(x, y, z) = \sum m(2, 3, 6, 7)$ b) $F_2(A, B, C, D) = \sum m(4, 6, 7, 15)$

E1.15 Simplify the following Boolean functions using 4-variable K-map:

a) $F_1(w, x, y, z) = \sum m(1, 4, 5, 6, 12, 14, 15)$ b) $F_2(A, B, C, D) = \sum m(2, 3, 6, 7, 12, 13, 14)$

E1.16 Simplify the following Boolean functions using 3-variable K-maps:

a) $F_1 = x'y' + yz' + x'yz'$ b) $F_2 = x'yz + xy'z' + xy'z$

E1.17 Simplify the Boolean function using 4-variable K-maps: $F = A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$

E1.18 Find the minterms of the following Boolean expression by first plotting each function in a K-map:

a) $F_1 = xy + yz + xy'z$ b) $F_2 = C'D + ABC' + ABD' + A'B'D$

E1.19 Using K-map find the minimum sum of products (MSOP) representation for,

$F(A, B, C, D, E) = \sum m(2, 5, 7, 11, 21, 23, 25, 27) + \sum d(1, 12, 17, 28)$

Draw the logic circuit of the minimal expression using only NAND gates.

E1.20 Using K-map find the minimum product of sum (MPOS) representation for,

$F(A, B, C, D, E) = \prod (0, 1, 2, 3, 4, 5, 16, 17, 18, 19, 24, 25) + \sum d(26, 27)$

Draw the logic circuit of the minimal expression using only NOR gates.

E1.21 Find the prime implicants for the following Boolean function and determine the essential prime implicants.

$F(A, B, C, D) = \sum m(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)$

E1.22 Implement the Boolean function with NAND gate and draw the logic diagram.

a) $F(x, y, z) = (1, 2, 3, 4, 5, 7)$ b) $F(x, y, z) = (0, 1, 3, 5, 6, 7)$

E1.23 Obtain the simplified Boolean expression for output F in terms of the input variables in the circuit of Fig. E1.23.

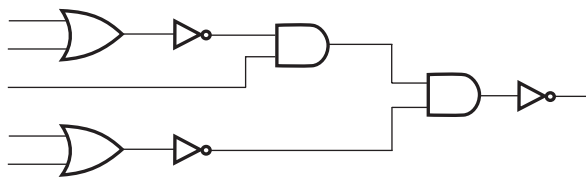


Fig. E1.23.

E1.24 Design a Boolean function with 3 inputs and 1 output.

- The output is 0 when the binary value of the input is less than 3. The output is 1 otherwise.
- The output is 1 when the binary value of the input is an odd number.

E1.25 Design a combinational circuit that compares two 3-bit numbers to check if they are equal. The circuit output is 0 if the two numbers are equal and 1 otherwise.

E1.26 Using adders,

- Design a 3-bit combinational circuit incrementer (a circuit that adds 1 to a 3-bit binary number).
- Design a 3-bit combinational circuit decrements (a circuit that subtracts 1 from a 3-bit binary number).

E1.27 Design a half adder using NAND gates.

E1.28 Design a half adder using logic low decoder.

E1.29 Design a half adder using a logic high decoder and NAND gates.

E1.30 Design a half adder using multiplexer.

E1.31 Design a full adder using NOR gates.

E1.32 Design a full adder using logic low decoder.

E1.33 Design a half subtractor using NOR gates.

E1.34 Design a half subtractor using logic low decoder.

E1.35 Design a half subtractor using a logic high decoder and NAND gates.

E1.36 Design a half subtractor using multiplexer.

E1.37 Design a full subtractor using logic low decoder.

E1.38 Design a full subtractor using multiplexer.

Answers

E1.1 a) $(x + y')(x + y)$ b) $(x' + y' + z')(x + y')(x' + y' + z)$

E1.3 a)

Input Variables			Complement of Inputs			Product Terms			Function Output
x	y	z	x'	y'	z'	x'y'	x'yz'	xyz'	F ₁
0	0	0	1	1	1	1	0	0	1
0	0	1	1	1	0	1	0	0	1
0	1	0	1	0	1	0	1	1	1
0	1	1	1	0	0	0	1	0	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	0	0	0	0	0
1	1	0	0	0	1	0	0	1	1
1	1	1	0	0	0	0	0	0	0

b)

Input Variables			Complement of Inputs			Product Terms		Function Output
a	b	c	a'	b'	c'	b'c'	ac	F ₂
0	0	0	1	1	1	1	0	1
0	0	1	1	1	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	0	1
1	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	1	1

E1.4a)

x	y	z	x + y + z	(x + y + z)'	x'	y'	z'	x'y'z'
0	0	0	0	1	1	1	1	1
0	0	1	1	0	1	1	0	0
0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	0	0
1	0	0	1	0	0	1	1	0
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	0	1	0
1	1	1	1	0	0	0	0	0

b)

x	y	z	xyz	(xyz)'	x' y' z'	x' + y' + z'
0	0	0	0	1	1 1 1	1
0	0	1	0	1	1 1 0	1
0	1	0	0	1	1 0 1	1
0	1	1	0	1	1 0 0	1
1	0	0	0	1	0 1 1	1
1	0	1	0	1	0 1 0	1
1	1	0	0	1	0 0 1	1
1	1	1	1	0	0 0 0	0

c)

x	y	z	$y + z$	$x + (y + z)$	$x + y$	$(x + y) + z$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

d)

x	y	z	$y \cdot z$	$x \cdot (y \cdot z)$	$x \cdot y$	$(x \cdot y) \cdot z$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

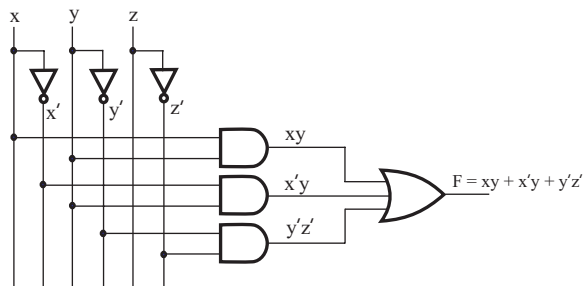
E1.5 a) $\Sigma m(0, 1, 2, 4, 6, 7, 8, 10, 12, 13, 14)$ b) $\Sigma m(2, 4, 5, 7)$ E1.6 a) $AB + BC$ (SOP), $B(A + C)$ (POS)b) $x + x'y' + x'z'$ (SOP), $x + y' + z'$ (POS)E1.8 $b'd' + abc'$ (Sum of products)E1.9 $(a + b')(a + c)(a' + b' + c')$ (Product of sums)E1.10 a) $F = xy + x'y + y'z'$ 

Fig.

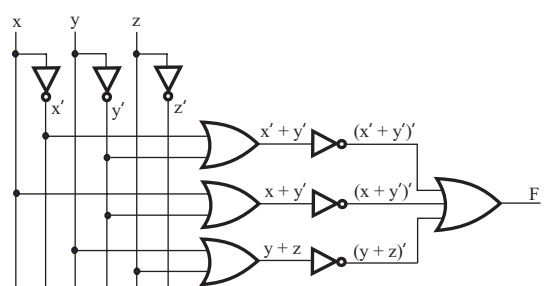
b) $F = (x' + y')' + (x + y')' + (y + z)'$ 

Fig.

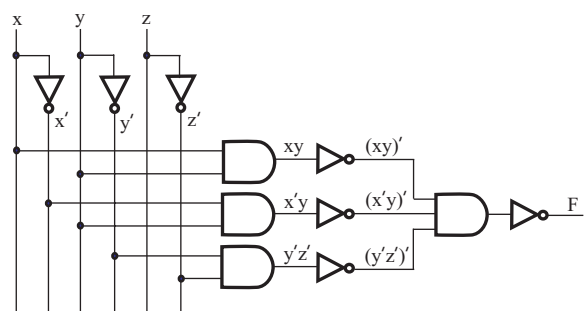
c) $F = ((xy)'(x'y)'(y'z')')'$ 

Fig.

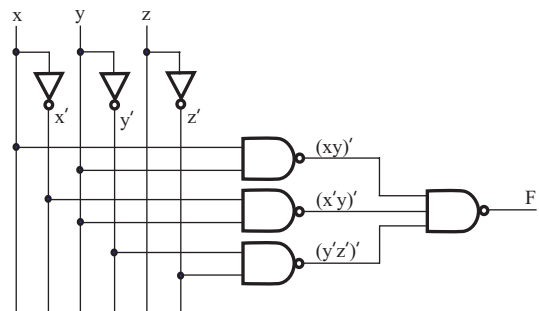
d) $F = ((xy)'(x'y)'(y'z')')$ 

Fig.

b)

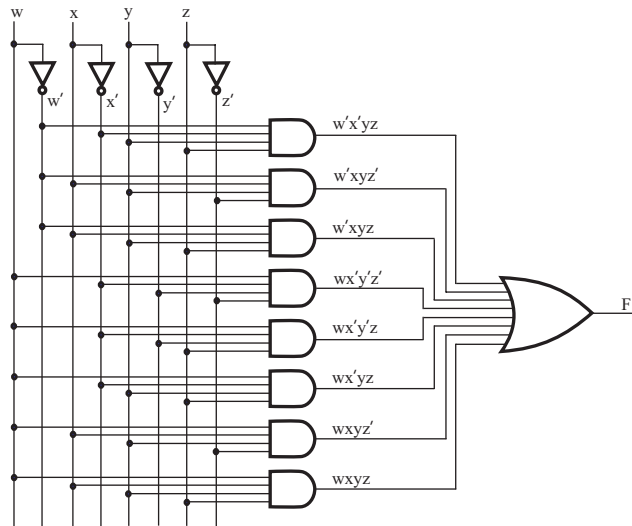


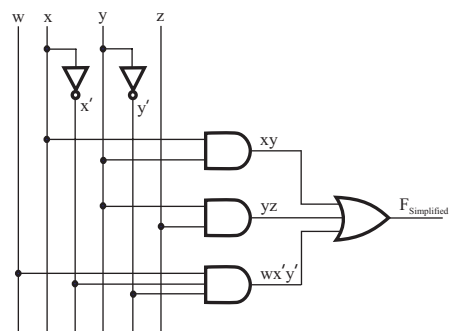
Fig: Logic circuit of F.

c) $F_{\text{simplified}} = xy + yz + wx'y'$

d)

Input Variables				Product Terms			Function Output
w	x	y	z	xy	yz	$wx'y'$	$F_{\text{simplified}}$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	1
1	0	0	1	0	0	1	1
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	1

e)

Fig: Logic circuit of $F_{\text{simplified}}$ E1.13 a) $F_1 = x'y' + xz$ b) $F_2 = x'y + xy'$ E1.14 a) $F_1 = y$ b) $F_2 = A'BD' + BCD$ E1.15 a) $F_1 = w'y'z + wxy + xz'$ b) $F_2 = ABC' + A'C + BCD'$ E1.16 a) $F_1 = x'y' + yz'$ b) $F_2 = xy' + x'yz$

E1.17 $F = ABC' + A'BD + B'D'$

E1.18 a)

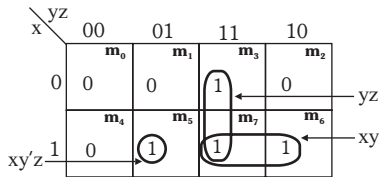


Fig: K-map for F_1 .

$$F_1 = \sum m(3, 5, 6, 7)$$

b)

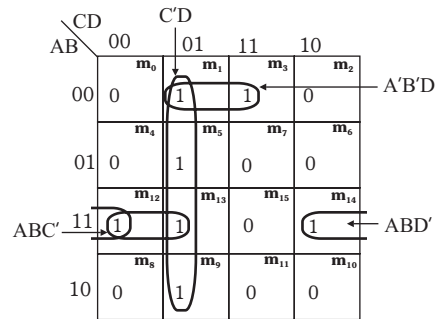


Fig: K-map for F_2 .

$$F_2 = \sum m(1, 3, 5, 9, 12, 13, 14)$$

E1.19 $F = A'B'C'DE' + BC'DE + B'CE + ABC'E$

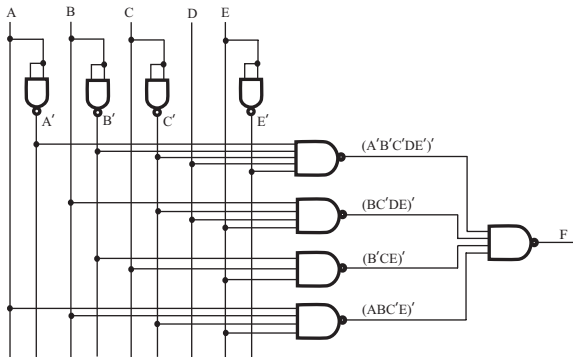


Fig: Logic circuit of function, F using only NAND gates.

E1.20 $F = (B + C)(A + B + D)(A' + C)$

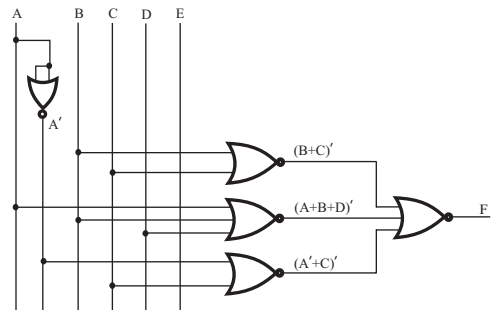


Fig: Logic circuit of function, F using only NOR gates.

E1.21 $F = B'D' + B'C + AC + A'BD$ (Alternatively, $F = B'D' + CD + AC + A'BD$)

E1.22 a)

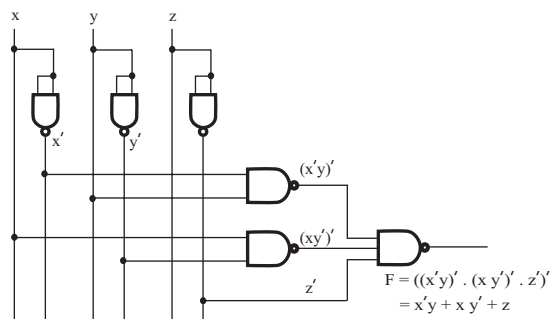


Fig: Logic circuit of F using only NAND gates.

b)

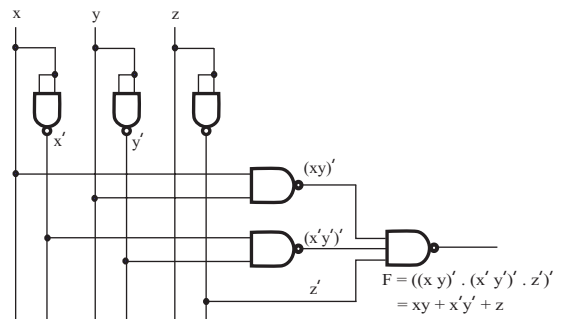


Fig: Logic circuit of F using only NAND gates.

E1.23 $F = A + B + C' + D + E$

E1.24 a) $x + yz$

b) z

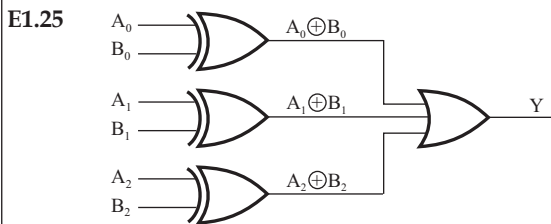


Fig: Combinational circuit to compare two 3-bit numbers for equality.

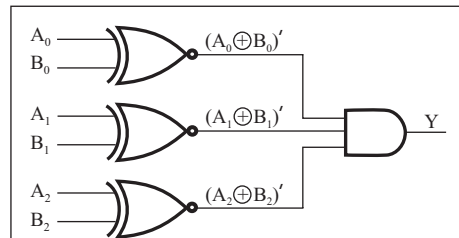


Fig: Alternate circuit.

E1.26 a)

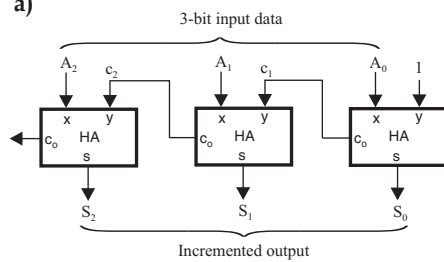


Fig.

b)

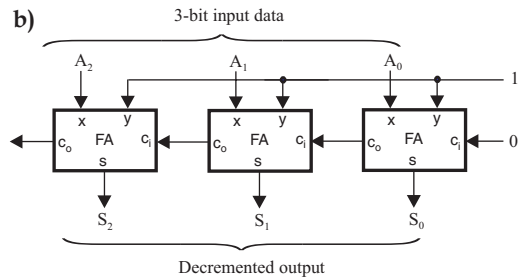


Fig.

E1.27

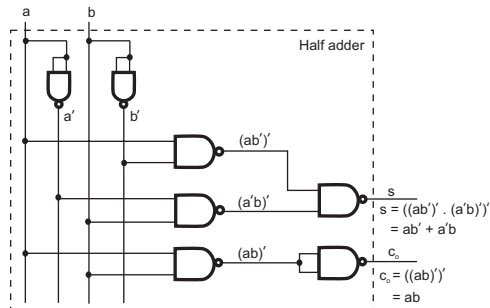


Fig: Logic circuit of half adder using NAND gates.

E1.28

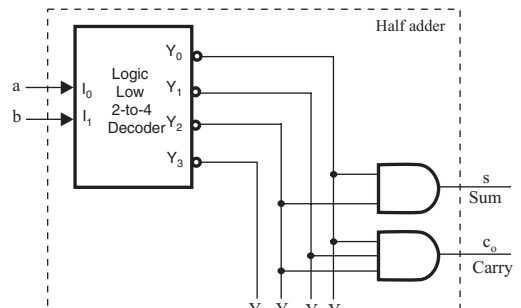


Fig: Half adder using logic low 2-to-4 decoder.

E1.29

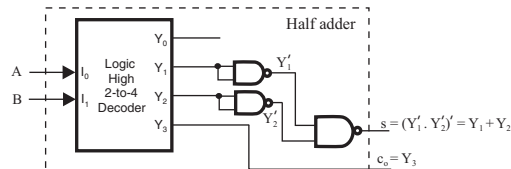


Fig.

E1.30

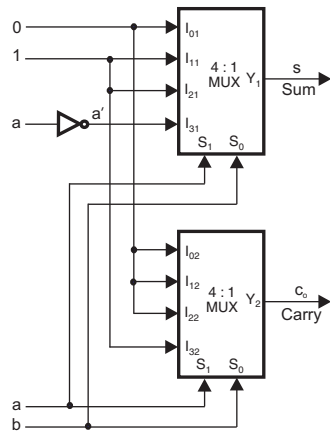


Fig: Half adder using 4:1 MUX.

E1.31

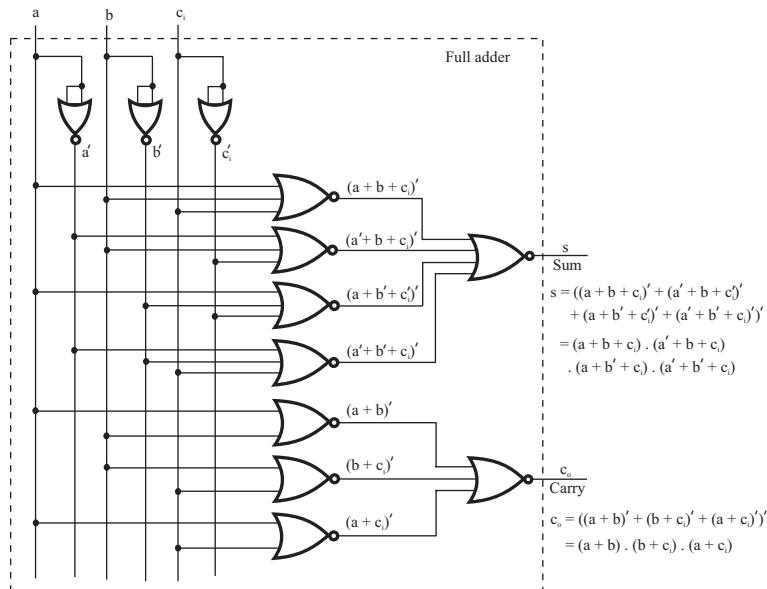


Fig: Logic circuit of full adder using NOR gates.

E1.32

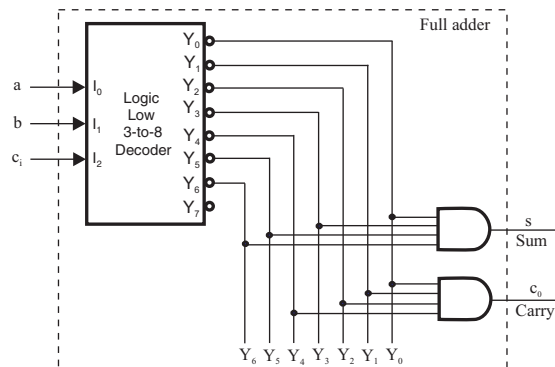


Fig: Full adder using logic low 3-to-8 decoder.

E1.33

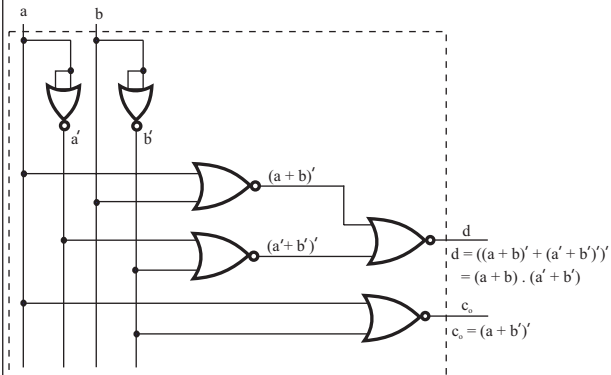


Fig: Logic circuit of half subtractor using NOR gates.

E1.34

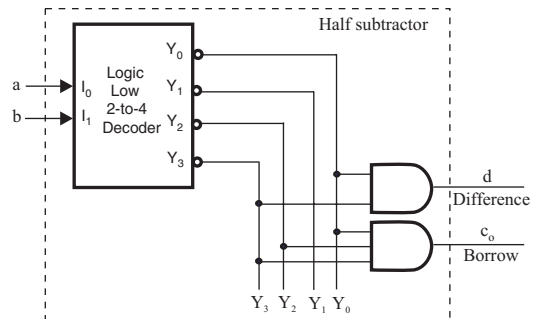


Fig: Half subtractor using logic low 2-to-4 decoder.

E1.35

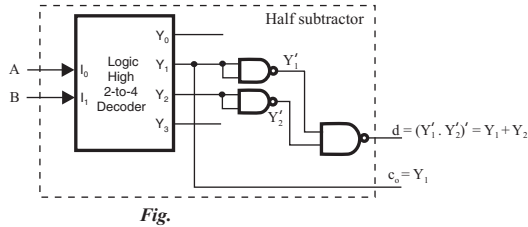


Fig.

E1.36

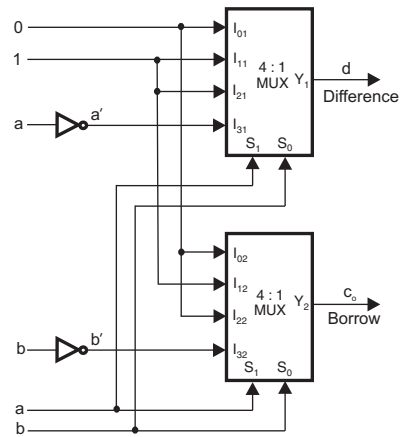


Fig: Half subtractor using 4:1 MUX.

E1.37

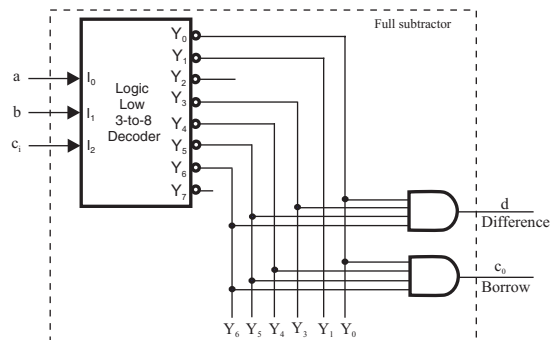


Fig: Full subtractor using logic low 3-to-8 decoder.

E1.38

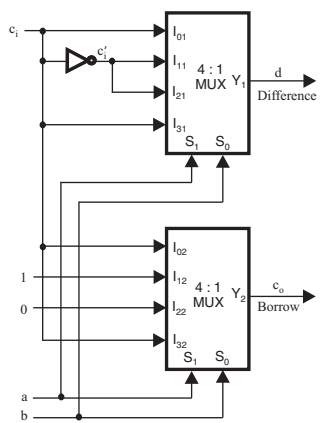


Fig: Full subtractor using two 4:1 MUX.